

Yii2 - FRAMEWORK. LEARNING SYSTEM BY CREATING CMS

Table of contents

Chapter 1. Downloading the framework and its configuration	10
Uploading the framework to the server	10
Verification of requirements	11
Using beautiful URLs	12
Setting the Cookies Validation Key	15
Adding a component	15
Application error handling	16
Configuring the mail server	16
Enabling the application debugger	17
Setting the software language	17
Creation of additional configuration data	17
Adding a connection to the database	18
Removal of unnecessary components	18
Chapter 2. Database	19
Article table	19
Table of blog entries	20
Table containing the category of the blog	21
System configuration table	22
Table of contact details	23
Table for information on downloads	23
Statistics table for downloads	24
Error404 table on our website	25
Table for administrative logs	25
Table of menu items	26
Table of text pages	27
Table of password reminders	28
Table with users	29
Chapter 3. User interface	31
User model	31
Controller for users - UserController	45
View - account activation	55
View - password change form	55
View - error on page	57
View - user login form	57
View - user profile	58
View - registration of a new account	59
View - information on correct registration	60
View - password reminder	61

View - information about changing the password	62
View - password reminder, e-mail check request	63
View - no access rights	63
Chapter 4. Page	65
Page display controller - PageController.....	65
Text page model - Page.....	67
View - index of text pages.....	71
View - no page specified.....	72
View - home page.....	72
View - showing the page.....	73
Chapter 5. Article	74
Model of articles - Article	74
Article controller - ArticleController	77
View: show articles.....	78
View: no article	79
View: one article.....	80
Chapter 6. Blog	82
Blog model - Blog	82
Blog controller - BlogController.....	87
View: blog.....	92
View: no blog entry	93
View: category of blog entry	94
View: show the entries matching the tag.....	95
View - show one entry in the blog	97
Chapter 7. Contact	98
Model contact form - Contact.....	98
Contact controller - ContactController.....	100
View - contact form.....	103
Chapter 8. Download.....	106
Download model - Download.....	106
Download controller - DownloadController.....	108
View - downloading files	110
Chapter 9. Error404	111
Error handling model - Error404	111
Chapter 10. User administration.....	113
User administration model - UserAdmin	113
User management controller - UseradminController	115
View - user administration - form.....	120
View - user administration - adding a user	122
View - user administration - list of users.....	122
View - user administration - user update.....	123

View - user administration - account details	124
Chapter 11. Management of the pages	126
Text page management model - PageAdmin	126
Controller of the administration of the pages - PageAdminController	127
View - administration of the pages - form	132
View - page administration - adding a page	133
View - page administration - index of pages	134
View - page administration - page update	135
View - page administration - page details	137
Chapter 12. Article management	139
ArticleAdmin - article management model	139
Controller of the administration of articles - ArticleAdminController	140
View - article administration - form	145
View - article administration - adding a new article	145
View - administration of articles - list of articles	147
View - article administration - article update	148
View - article administration - full data article display	150
Chapter 13. Management of the blog	152
Blog management model - BlogAdmin	152
Controller of the blog administration - BlogAdminController	153
View - administration of the blog - form	159
View - administration of the blog - adding a new entry	160
View - administration of the blog - lists of entries	161
View - administration of the blog - update of an entry	162
View - administration of the blog - viewing the entry	164
Chapter 14. Administration of blog categories	166
Category administration model in the blog - BlogCategoryAdmin	166
Administration controller for blog categories - BlogcategoryAdminController	167
View - administration of blog categories - form	172
View - administration of blog categories - creating a new category	173
View - administration of blog categories - list of categories	174
View - category administration - category update	175
View - category administration - category browsing	176
Chapter 15. Download management	178
Downloadable file management model - DownloadAdmin	178
Controller of the administration of the download department - DownloadAdminController	179
View - download administration - form	184
View - download administration - add file	185
View - download administration - file directory	185
View - download administration - file update	186
View - download administration - file details	187

Chapter 16. Management of download statistics	189
Download statistics model - DownloadStatAdmin	189
DownloadstatadminController - Download statistic administrator controller	190
View - download statistics	192
Chapter 17. Contact management.....	194
Administration model contact details - ContactAdmin	194
Controller of contact administration - ContactadminController	195
View - contacts administration - form.....	200
View - contact administration - create a new contact.....	201
View - contact administration - list of contacts	201
View - contact administration - contact update	202
View - contact management - contact details	203
Chapter 18. Password recovery management.....	205
Model displaying attempts to recover the password - PasswordAdmin	205
Administrator controller with password reminder logs - PasswordadminController	206
View - account password reminder logs	208
Chapter 19. Page setup.....	210
Page configuration model - ConfigAdmin	210
Site configuration controller - ConfigadminController	213
View - page configuration.....	215
Chapter 20. Managing the page menu	217
User menu model - LeftMenuAdmin.....	217
Controller of the administration of the page menu - LeftmenuadminController.....	219
View - administration of the system menu	226
Chapter 21. Saving and reading administrator logs	238
Model of saving administrator actions - LogAdmin	238
Controller of login attempts for the administrator - LogadminController	240
View - login information to the service.....	242
Chapter 22. Error controller	243
Error administration controller - Error404adminController	243
View - error file 404	245
Chapter 23. Views	246
Administrator panel main file.....	246
Page layout main file	249
Chapter 24. Languages	254
Setting the application language.....	254
Create language files.....	254
Chapter 25. Components and libraries.....	255
Component that logs administrator actions, retrieves configuration, creates URLs, and creates menus.....	255
JQuery - library	261
JQuery - UI.....	261

JQuery - text retrieval	261
JQuery - date stamp	261
CKEditor - WYSIWYG editor	261
Chapter 26. Web site for user	262
Account registration.....	262
Logging in to your account	263
Profile completion	263
Changing the password	263
Logging out	264
Restoring the password	264
Chapter 27. Administrator's webpage	266
Go to the admin panel	266
Page setup	266
System text pages	266
Page viewing	267
Adding a page.....	267
Page editing	267
Viewing the page details	268
Deleting a page	269
Articles.....	269
Article viewing	269
Addition of an article	270
Editing an article.....	270
Article details.....	271
Removal of article.....	271
Blog	271
Viewing entries.....	271
Addition of an entry	272
Editing an entry.....	272
Details of the entry.....	273
Deletion of an entry.....	273
Categories of blog.....	274
Category overview	274
Adding a category.....	274
Category editing.....	274
Category detail.....	275
Deleting a category	275
Users	275
Viewing users	275
Adding a user.....	276
Editing a user	276

User details.....	277
Deleting a user	277
Password reminder attempts.....	278
System logs	278
Contact.....	278
Viewing addresses	278
Adding an address.....	279
Address editing.....	279
Address details.....	279
Address deletion	279
Download.....	280
Viewing files	280
Adding a file.....	280
Download file editing.....	281
File detail.....	282
Deleting a file	282
Download statistics	282
Page menu.....	283
Viewing the menu.....	283
Adding menus.....	283
Deleting the menu.....	283
Arrange the menus in the correct order.	284
Errors 404.....	284
Logout.....	285
Summary.....	286

All rights reserved. Unauthorised distribution of the whole or any part of it no part of this book may be reproduced in any form whatsoever.

Copying by photocopying, photographic means, as well as copying books on a film, magnetic or other medium causes the infringement of the copyright of this publication.

All characters in the text are registered film marks or the goods of their owners.

The author has made every effort to ensure that the information contained in this publication is correct, complete and reliable. However, it does not assume any responsibility or liability for any of the following use of, and possible infringement of, patents or proprietary. The author also assumes no responsibility for any possible damage caused by damage resulting from the use of the information contained in the book.

Cover graphics:

- valco - <https://pixabay.com/pl/background-t%C5%82o-design-motivo-1156439/>
- suyuan333 - <https://pixabay.com/pl/pawilony-expo-odwiedz-budynku-1641171/>

Lukas Sosna

<http://en.lukasz.sos.pl>

ISBN: 978-83-950709-1-4

Copyright © Lukas Sosna 2018

When programming in **PHP**, we often have to solve problems that are already developed by others. Remember to configure our applications, connect to the database, display the templates - all of this must be programmed by yourself.

It is very easy to use the framework, which contains most of the elements we need. Simply turn them on and configure them in the settings file. This allows you to start working on the system you want to program.

Framework **Yii** has in its resources: change of operating language, components, caching, error handling, library for sending e-mail, system log, database support, URL manager or checking and providing information about errors.

Additionally, **Yii** is equipped with a module called **Gii** - it allows for automatic generation of elements such as controller, model or entire CRUD management systems.

Using the framework we will build an application based on the **MVC** pattern, which will sort different elements of the application and in the future there will be no problem with finding the right fragment of the code. By using models with queries to databases, views containing page templates and controllers containing program code, we create a solution that is consistent with developers' practices all over the world.

Additionally, **Yii** has an element called **ActiveRecord**, thanks to which the creation of simple queries to the database does not have to be done in the model. For this purpose it is enough to use controller where appropriate methods are used to create a query. The framework will process them on request in **SQL** language. Unfortunately, **ActiveRecord** must be used in moderation because of the resources used.

This book was written to show you how to use most of the popular and most frequently used libraries. Only working with the whole system from A to Z will allow you to learn the knowledge, which after a few repetitions becomes very simple and friendly.

Chapter 1. Downloading the framework and its configuration

Download the framework before you start working with it. These can be found on the following website:

<http://www.yiiframework.com/>. In the page that opens, you will find links in the header. Move the cursor over the downloads menu and select the framework option.

Scroll to the page header: **Install from an Archive File**. Select the link: **Yii 2 with basic application template** from under the header. When you click it, the download of the **Tar** compressed file in the **Gzip** will start. This file can be easily opened with the free **7-Zip**¹ compressor.

Why did we choose to download the option: **Yii 2 with basic application template** instead of: **Yii 2 with advanced application template**? We will build all the elements from scratch, so that you can learn the rules of operation and programming in a framework. I want to show you what this software can do and that's why we will do all the elements ourselves. We will repeat all the steps and thanks to that you will not have to think about which framework to use. Actually, it would not be possible without modification in files with models and controllers. From our point of view it is better to write it in one's own way and learn more about the activity of the framework.

The structure of the framework is very complicated at first glance, but after getting to know it quickly, it is easy to find out where the things responsible for control over the application, models with access to the database and view files. The authors of the framework have tried to make it possible to place it together with the content outside the directory visible on the Internet, so even if you make a mistake, for example, not deleting an important file, it will not be visible on the web anyway. This is a great solution. That is why our website is located in the **web** catalogue.

Uploading the framework to the server

Due to its structure, the Framework can be uploaded in various ways to a web server designed to operate our website. There are two possibilities.

The first is to upload all the program files to the server in the place of the directory, the content of which is visible to the public, most often it is: **public_html**. All paths in the **index.php** root file of the framework in the web directory must be configured accordingly. Open the file and edit it in three places, correcting the paths from the folder pointer higher **../** to the current folder: **/**

```
<?php

// comment out the following two lines when deployed to production
defined('YII_DEBUG') or define('YII_DEBUG', true);
defined('YII_ENV') or define('YII_ENV', 'dev');
defined('YII_ENV_TEST') or define('YII_ENV_TEST', true);

require(__DIR__ . '/vendor/autoload.php');
require(__DIR__ . '/vendor/yiisoft/yii2/Yii.php');

$config = require(__DIR__ . '/config/web.php');

(new yii\web\Application($config))->run();
```

¹ <https://www.7-zip.org/>

The second option, recommended by framewoven programmers, is to upload all the software to the directory one level higher than **public_html**. Then you should move the files from the **web**² directory to the main **public_html** folder where our website is located.

In this book I will personally focus on the second option. It is safer than the first one due to the fact that the end user will not influence the content of individual files and in case of any error in the frame of the browser will not be able to use it in such a simple way as in the case when files will be available for everyone who visits our site.

Access paths will be provided in the form of **../controller** and this means that you need to exit one level higher from the public directory where the **controller** directory is located.

Finally, we must remember one more very important thing. When our project will see the light of day and we will want to present the website to a wider audience we should comment on the first two lines in **index.php** file and turn off the debugging mode.

```
//defined('YII_DEBUG') or define('YII_DEBUG', true);  
//defined('YII_ENV') or define('YII_ENV', 'dev');
```

Verification of requirements

Checking the requirements of the Framework like any software also has its own requirements for the server and the software serviced by the software server. After uploading the files to the server, copy the **requirements.php** file from the main directory of the framework to the folder available to the public: **public_html**.

Then you have to reedit such a file by changing the path to the files so that the program will check the options of the server. Add to the **frameworkPath** path a reference to the directory above the current one. We will do it by adding the phrase **../**

```
$frameworkPath = dirname(__FILE__) . '/../vendor/yiisoft/yii2';
```

Then in the address bar of your browser you enter the URL of the page together with the name of the file, so that you can find out whether our server meets the requirements: **http://yii.phpbluedragon.eu/requirements.php**.

A script will be run in the browser, which will detect the software included in the server and allows you to check whether even the minimum requirements will be met. The information will then be displayed with details of which versions and programs are on the server. If it is not possible to run the framework on the server, what functions are missing will be presented. You can then write to the administrator of our server to enable the appropriate items.

² The most common folder on servers is **public_html**, but in some cases it may be called differently.

Yii Application Requirement Checker

Description

This script checks if your server configuration meets the requirements for running Yii application. It checks if the server is running the right version of PHP, if appropriate PHP extensions have been loaded, and if php.ini file settings are correct.

There are two kinds of requirements being checked. Mandatory requirements are those that have to be met to allow Yii to work as expected. There are also some optional requirements being checked which will show you a warning when they do not meet. You can use Yii framework without them but some specific functionality may be not available in this case.

Conclusion

Your server configuration satisfies the minimum requirements by this application.
Please pay attention to the warnings listed below and check if your application will use the corresponding features.

Details

Name	Result	Required By	Memo
PHP version	Passed	Yii Framework	PHP 5.4.0 or higher is required.
Reflection extension	Passed	Yii Framework	
PCRE extension	Passed	Yii Framework	
SPL extension	Passed	Yii Framework	
Ctype extension	Passed	Yii Framework	
MbString extension	Passed	Multibyte string processing	Required for multibyte encoding string processing.
OpenSSL extension	Passed	Security Component	Required by encrypt and decrypt methods.
Intl extension	Passed	Internationalization support	PHP Intl extension 1.0.2 or higher is required when you want to use advanced parameters formatting in <code>Yii::t()</code> , non-latin languages with <code>Inflector::slug()</code> , IDN-feature of <code>EmailValidator</code> or <code>UrlValidator</code> or the <code>yii\i18n\Formatter</code> class.
ICU version	Passed	Internationalization support	ICU 49.0 or higher is required when you want to use <code>#</code> placeholder in plural rules (for example, plural in <code>Formatter::asRelativeTime()</code>) in the <code>yii\i18n\Formatter</code> class. Your current ICU version is 57.1.
ICU Data version	Passed	Internationalization support	ICU Data 49.1 or higher is required when you want to use <code>#</code> placeholder in plural rules (for example, plural in <code>Formatter::asRelativeTime()</code>) in the <code>yii\i18n\Formatter</code> class. Your current ICU Data version is 57.1.
Fileinfo extension	Passed	File Information	Required for files upload to detect correct file mime-types.
DOM extension	Passed	Document Object Model	Required for REST API to send XML responses via <code>yii\web\XmlResponseFormatter</code> .
IPv6 support	Passed	IPv6 expansion in <code>IpValidator</code>	When <code>IpValidator::expandIPv6</code> property is set to <code>true</code> , PHP must support IPv6 protocol stack. Currently PHP constant <code>AF_INET6</code> is not defined and IPv6 is probably unsupported.
PDO extension	Passed	All DB-related classes	
PDO SQLite extension	Passed	All DB-related classes	Required for SQLite database.
PDO MySQL extension	Passed	All DB-related classes	Required for MySQL database.
PDO PostgreSQL extension	Passed	All DB-related classes	Required for PostgreSQL database.
Memcache extension	Warning	MemCache	
GD PHP extension with FreeType support	Passed	Captcha	Either GD PHP extension with FreeType support or ImageMagick PHP extension with PNG support is required for image CAPTCHA.
ImageMagick PHP extension with PNG support	Passed	Captcha	Either GD PHP extension with FreeType support or ImageMagick PHP extension with PNG support is required for image CAPTCHA.
Expose PHP	Passed	Security reasons	"expose_php" should be disabled at php.ini
PHP allow url include	Passed	Security reasons	"allow_url_include" should be disabled at php.ini
PHP mail SMTP	Passed	Email sending	PHP mail SMTP server required

Figure 1.1 The effect of a requirement check script.

Using beautiful URLs

The first element without which there is no point in doing anything in the framework is using nice URLs by the system. This means that instead of questioning the parameters to **index.php** file we will use the paths in the address. This will make it easier for you to work with the website and for end users to use it. However, that is not all. Today, websites are created with an eye to the behaviour of search engine bots on your website. The most important search engine is **Google** and it adds points for readable addresses. This way we will raise our website in the number of points awarded by the search engine. Unfortunately, there are times when websites are not written for people, but for search engine robots.

A file named **.htaccess** should be placed in the **public_html** or other directory where your website is located. Depending on whether the page is located under the main domain address, then we enter the content into the file.

```
Options +FollowSymLinks
IndexIgnore */*

RewriteEngine on
RewriteBase /

# if a directory or a file exists, use it directly
RewriteCond %{REQUEST_FILENAME} !-f
```

```

RewriteCond %{REQUEST_FILENAME} !-d

# otherwise forward it to index.php
RewriteRule . index.php

```

Thanks to the file, all addresses entered by the user after finding out that there is no directory with the name we have typed, our request will be forwarded to **index.php** file and there appropriately broken and forwarded to the controller for execution.

The next thing is to configure the system. Go to editing the file **../config/web.php**. We are looking for an **urlManager** line, and when we find one, we remove the comment from it.

We define the path to our site in the **baseUrl** parameter. We add true value in **enablePrettyUrl** which will cause nice-looking tracks to be turned on and set **showScriptName** to false value, so that **index.php** file is not shown in URL addresses.

```

'urlManager' => [
    'baseUrl' => 'http://yii.phpbluedragon.eu/',
    'enablePrettyUrl' => true,
    'showScriptName' => false,
    'enableStrictParsing' => true,

```

Then in the **rules** we declare our paths.

```

'rules' => [

```

In the event that the address does not contain any parameters, i.e. the page is opened by entering the URL into the browser or the web browser reaches the homepage, its content should be declared. We create it by typing an empty value and redirecting it to the **page** controller and the **home** method.

```

'' => 'page/home',

```

Users must be able to log in, register and log out.

```

'login' => 'user/login',
'register' => 'user/register',
'logout' => 'user/logout',

```

Users also need a password reminder action. We add redirections to the user controller with the **rempass** action, and in the second one we set the value of the **UserId** variable to the number via **\d+**, it means that there can be more than one digit. Then **UserHash1** and **UserHash2**, which will be a sequence of characters consisting of letters and numbers: **\w** and a dash mark: **\-**. Both value definitions are given in brackets followed by a + sign, which means that any number of characters can occur. The request will be redirected to the user controller, where it will be redirected to the **rempassset** method. The variables that we set in the method must be IDENTICAL in terms of name. Otherwise we will get an error with the information that the page has not been found.

```

'remind-password' => 'user/rempass',
'remind-password-set/<UserId:\d+>/<UserHash1:[\w\-\>+>/<UserHash2:[\w\-\>+>' => 'user/rempassset',

```

We define access to the profile and the option of changing the password.

```

'profile' => 'user/profile',

```

```
'change-password' => 'user/changepassword',
```

After registering your account, you must activate it by receiving an e-mail with a special link to activation. It will contain the **UserId** user identifier, which can only be a number, and **UserKey**, a special key generated for the user, which can contain letters and numbers and a - character.

```
'activate/<UserId:\d+>/<UserKey:[\w\-\>' => 'user/activate',
```

A special action for a user who does not have the right to access a separate part of the website.

```
'right' => 'user/right',
```

Access to the pages will be done by calling the URL with the page word, then a specially prepared **PageURL** address containing words, numbers and a dash. The next important element is the page identifier in the **PageId** database, which consists of numbers only. If any parameter is missing, the user will be redirected to the index method.

```
'page/<PageUrl:[\w\-\>/<PageId:\d+>' => 'page/showone',  
'page' => 'page/index',
```

Articles as in the case of the parties. A call with the **ArticleUrl** parameter - a specially constructed article title - and an **ArticleId** parameter containing an article identifier consisting of numbers only will refer to the corresponding content from the database. Otherwise, the contents of the **index** method will be displayed.

```
'article/<ArticleUrl:[\w\-\>/<ArticleId:\d+>' => 'article/showone',  
'article' => 'article/index',
```

An entry from the blog will contain in its address **BlogUrl** - a specially constructed title of arrival for browsers and **BlogId** consisting of numbers only. Then, if you click on a category, you will also be forwarded to **CategoryUrl** - the browser-friendly category title and its **CategoryId** identifier. At the very end there is loading tags where it can consist of letters, numbers and - character.

```
'blog/<BlogUrl:[\w\-\>/<BlogId:\d+>' => 'blog/showone',  
'blog' => 'blog/index',  
'category/<CategoryUrl:[\w\-\>/<CategoryId:\d+>' => 'blog/category',  
'hash/<Hash:[\w\-\>' => 'blog/hash',
```

We will contact the visitors of our website by setting the transition after entering the **contact** word after the address.

```
'contact' => 'contact/index',
```

Downloading programs will be done by entering an identifier consisting of **ProgramId** numbers only and **ProgramName** parameter consisting of letters, numbers and - character. If any parameters are missing, the index method will be called from the download controller.

```
'download/<ProgramId:\d+>/<ProgramName:[\w\-\.\>' => 'download/getprogram',  
'download' => 'download/index',
```

The admin panel will not have to rewrite URLs, so let's only add a fragment that will help you type it in your browser. Add the admin phrase, which will run the **configadmin** controller with the **index** action.

```
'admin' => 'configadmin/index',
```

At the end of the method of rewriting URLs you need to add the possibility of handling erroneous addresses or addresses that we will not set specifically. We do this by adding standard commands.

```
'<controller:\w+>/<id:\d+>' => '<controller>/view',
'<controller:\w+>/<action:\w+>/<id:\w+>' => '<controller>/<action>',
'<controller:\w+>/<action:\w+>' => '<controller>/<action>',
],
],
```

We have to remember about a very important thing, which is entering addresses in the configuration file in the right order.

I will show this on the example of the download section. After you have entered the address formatting as in the list below, the index method will always be executed from the download controller.

```
'download' => 'download/index',
'download/<ProgramId:\d+>/<ProgramName:[\w\-\.\.]+>' => 'download/getprogram',
```

That is why it is necessary to start from the definition of using the controller with the most parameters. In this way, the program will check all other options before displaying the **index** method.

```
'download/<ProgramId:\d+>/<ProgramName:[\w\-\.\.]+>' => 'download/getprogram',
'download' => 'download/index',
```

Setting the Cookies Validation Key

Our application will not work without providing a key to handle cookies set by the Yii framework. Without this key, unfortunately, we will not launch our website. In the **./config/web.php** file, find the line with the word that is the key of the request table, and then enter any random value into the **ValidationKey** cookie. This will ensure that cookies are better protected against attempted interception.

```
'request' => [
    'cookieValidationKey' => 'sT+W=/8@qGL`kARLE.;Z(toGJ?zHb0',
],
```

Adding a component

Creating a menu, formatting its appearance and system logo does not have to be done in every controller. Yii allows you to create your own component in which we include all the elements needed to implement activities that occur in more than one place. This component should be loaded and we will do it in **./config/web.php** file. Enter the name after which we will call the functions according to the file name, then enter the word class and load the component into it using the word app as the access path, which means the main folder of the software, then enter the directory of components and the file without the extension.

```
'components' => [
    'OtherFunctionsComponent' => [
        'class' => 'app\components\OtherFunctionsComponent',
    ],
],
```

Application error handling

Most often, when there are errors in the URL or missing parameters, a special controller is started, whose task is to inform the end user about the error. Open the `../config/web.php` file and in the line containing the **errorHandler** key erase all the pairs of keys and values. Then we enter so that all errors are directed to the user controller and to the error method.

```
'errorHandler' => [
    'errorAction' => 'user/error',
],
```

Configuring the mail server

When registering a user or when we want users to contact the service via e-mail we will need an e-mail account, which will allow us to establish a connection with the server. Unfortunately, the **mail()** function is becoming more and more insignificant and on some servers it has been completely abandoned.

Open the `../config/web.php` file and uncomment the mailer key line. Now set the options. The first element is the class needed to send e-mails. Fortunately, all the files are already in the system. Set it to **SwiftMailer**. Then the **useFileTransport** option is very useful if you want to test our website. Setting **true** value in this key will cause that our e-mails will not be sent, but saved in files. However, when our website reaches a server in the Internet, we set the value of false in this key, it will cause that from now on the e-mails will be sent via the mail server. The **transport** key contains detailed data for configuring the server. We will use **SMTP** class for outgoing mail. Then in the **host** key we declare the URL of the server, the **username** contains the username, **password** to the SMTP server, the port contains the port number on which the service of connection with the SMTP server is available and at the end the **encryption** is set to the encryption mode of our server. When you have finished entering the settings, save the file.

```
'mailer' => [
    'class' => 'yii\swiftmailer\Mailer',
    'useFileTransport' => false,
    'transport' => [
        'class' => 'Swift_SmtpTransport',
        'host' => 'mail.phpbluedragon.eu',
        'username' => 'yii@phpbluedragon.eu',
        'password' => 'PASSWORD',
        'port' => '587',
        'encryption' => 'tls',
    ],
],
```

Attention! Your data will be different than in my case. You need to check your mail server address, port, user and password. The most important thing, however, is whether it supports an encrypted connection or not. If you do not know the answer to any question, write to the server administrator.

Enabling the application debugger

Framework provides us with a special tool for finding and tracking errors on our website. It displays, among other things, server and **PHP** configurations, the status of the current site, variables available through **GET**, **POST**, **COOKIE** and **SESSION**, application logs, the amount of time and individual methods needed to generate the site, the amount of **RAM** used, queries to the database, e-mails sent by us and many other valuable data. I have found out more than once how useful it is to use the data from the tool.

You can activate the display of the bar in the `../config/web.php` file. At the bottom you will find the configuration in the `$config` table with `bootstrap` index. It must be commented on and it is best to add your own IP address. Without providing the address, the data concerning the details of the website's operation will be available to the public, which will be very dangerous. Therefore, an additional key should be added under the name of `allowedIPs`. It will contain tables with defined IP addresses for which the framework can display the programming bar.

```
$config['bootstrap'][] = 'debug';
$config['modules']['debug'] = [
    'class' => 'yii\debug\Module',
    'allowedIPs' => ['83.4.106.220'],
];
```

Setting the software language

The system works by default in **English** and all messages will be displayed in **English**, for example when validating the form. Therefore, if you want to set the availability of the page in a different language, you need to set the language. Open the `../config/web.php` file and type the language key into its configuration table and set its value at **en**. Since most languages are already available by default in the software, there is no need to download additional translation files.

```
'language' => 'en',
```

Creation of additional configuration data

When building the website, we will also need additional data to configure the system. We will create them in the `../config/params.php` file as an array, which is included in the main configuration array. We will provide such data as **adminEmail** - the e-mail address of the administrator, **adminName** - the name and surname or the nickname of the administrator, **pageTitle** - which will contain the title of the page created using a browser, **pageUrl** along with the website address and the most important option - **saltPassword** - it is an element inserted into each password by its encryption. Remember that changing this parameter will require resetting the password by all users.

```
return [
    'adminEmail' => 'yii@phpbluedragon.eu',
    'adminName' => 'Lukas Sosna',
    'pageTitle' => 'Page based on the Yii framework',
    'pageUrl' => 'http://yii.phpbluedragon.eu/',
    'saltPassword' => 'Z{+;r3hWP;',
];
```

Adding a connection to the database

The data presented on the website will be placed in a database so that they can be accessed both through the admin panel and directly, for example, using the **phpMyAdmin** tool. The first thing is to provide data that will help us connect to the database. Open the `../config/db.php` file that contains the table for configuring the connection.

The first element is class, which is responsible for the class of connection to the database. The second key is **DSN**, which contains the type of database in our case it will be **mysql**, then the **host** is the URL of the database and **dbname** is the name of the database. Another **username** key is used to define the user name that has the right to connect, a **password** key to define the password of a given user. Then we declare **charset** or a set of characters to communicate with the database, as well as **tablePrefix** or a prefix of all tables in the database concerning our system. With the addition of prefixes, it is possible to install countless theoretical systems in a single database. Save the file at the end.

```
return [  
    'class' => 'yii\db\Connection',  
    'dsn' => 'mysql:host=localhost;dbname=yii',  
    'username' => 'yiiuser',  
    'password' => 'yiipassword',  
    'charset' => 'utf8',  
    'tablePrefix' => 'yii_',  
];
```

Removal of unnecessary components

You've probably visited the home page and you know that the system contains controllers, models and views to operate it. It is best to delete them, because later you may wonder where the file was located in the folder. That is why we remove them one by one:

- `../controllers/SiteController.php`
- `../models/ContactForm.php`
- `../models/LoginForm.php`
- `../models/User.php`
- `../views/site` - entire catalogue

With such a pre-prepared framework we can start building our website.

Chapter 2. Database

Once the system is properly configured we can start to create our database and tables from which it will use by selecting the data to complete the page. In this case, we will not use links. If they were used, it would be sufficient to delete one record in one table for all related records in the other also to be deleted. On this example I would like to show you the possibilities of the framework which is **Yii**.

To save you having to scroll through the book, some listings have been shortened. All commands can be found in the **ZIP** file attached to the book in the **SQL** directory.

Article table

The first table will be used to store the articles in it, which will be available for viewing on the website, either individually or as a whole, together with a division into pages.

Create a `yii_article` table:

- `article_id` - the unique identifier of the article
- `article_title` - title of the article
- `article_text` - content of the article
- `article_author` - author of the article
- `article_date` - date of publication of the article
- `article_url` - part of the URL that will be the processed title

```
CREATE TABLE `yii_article` (  
  `article_id` int(11) NOT NULL,  
  `article_title` varchar(150) NOT NULL,  
  `article_text` text NOT NULL,  
  `article_author` varchar(55) NOT NULL,  
  `article_date` datetime NOT NULL,  
  `article_url` varchar(65) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

We will add the data so that the page does not look empty. We are adding several articles.

```
INSERT INTO `yii_article` (`article_id`, `article_title`, `article_text`, `article_author`,  
`article_date`, `article_url`) VALUES  
  
(1, 'Lorem ipsum dolor sit amet, consectetur adipiscing elit.', '<p>Lorem ipsum dolor sit amet,  
consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium. Suspendisse odio magna, aliquam  
sed nulla sed, rutrum auctor neque. Pellentesque tincidunt massa eget tellus vehicula malesuada.  
Pellentesque accumsan aliquet sagittis. Quisque mollis leo faucibus cursus consequat. Morbi elementum,  
velit eu aliquet pulvinar, quam ex lobortis lectus, ut blandit risus augue a felis. In gravida varius  
arcu, at vulputate purus eleifend vestibulum. Nunc efficitur felis nec libero varius vehicula. In sed  
turpis purus. Nunc feugiat nisl ac augue viverra scelerisque. Donec sit amet augue in leo lobortis dictum.  
Sed id tortor eget metus gravida convallis in et metus. Sed posuere turpis ultrices, vulputate diam ac,  
suscipit erat. Mauris eleifend urna at erat faucibus, vitae feugiat ex aliquam. Duis at diam vitae neque  
maximus vehicula. Phasellus fermentum suscipit velit, at gravida leo varius at. Nunc pharetra nibh ac  
tortor scelerisque luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per  
inceptos himenaeos. Aenean sagittis turpis quis dictum consequat. Sed a tellus non ipsum semper commodo at  
tincidunt ligula. Proin ullamcorper orci et turpis hendrerit, quis ullamcorper lectus varius. Cras  
sollicitudin diam egestas efficitur sollicitudin. Integer justo augue, fringilla vel velit non, aliquet  
luctus arcu. Nullam sodales, est at consectetur fermentum, massa tellus fermentum nisl, ac rhoncus velit  
lectus quis magna. Mauris non eros justo. Cras luctus dui a est laoreet interdum quis et quam. Quisque vel  
dolor molestie, euismod sapien ut, vehicula leo. Mauris eu rhoncus dolor. Nulla in erat at est faucibus
```

porta id nec lacus. Nam aliquet fermentum tellus vitae fermentum. Morbi euismod maximus urna eget finibus. Vestibulum ultrices, nulla eu dictum fringilla, sapien orci commodo dui, sed vestibulum turpis enim sit amet ante. Sed elementum tortor non quam ullamcorper lacinia. Etiam elit ligula, laoreet sed sollicitudin eu, feugiat sit amet nulla. Ut sit amet lorem dui. Duis ultrices, justo at euismod accumsan, arcu leo ultricies sapien, non interdum mi risus vitae mi. Fusce lectus elit, vulputate imperdiet velit nec, tempor bibendum tellus. Fusce tincidunt elementum nibh, vitae lacinia libero varius nec. Donec suscipit, diam accumsan aliquet maximus, dui dolor lobortis nibh, non tempor dolor risus sed magna. Curabitur sit amet scelerisque lacus, sit amet finibus elit. Mauris tempor, orci at tempor rhoncus, odio magna tincidunt odio, in commodo diam turpis ac erat. Suspendisse nulla diam, ultrices vel efficitur ac, imperdiet sed orci. Suspendisse egestas dui eu ipsum fringilla mollis. Duis gravida interdum consequat. Integer blandit nulla vitae dignissim vehicula. Donec placerat nisl a tempor pulvinar. Etiam viverra porta ornare. Nullam cursus orci metus, ut tempor elit elementum ut. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In leo neque, feugiat vel diam vel, molestie cursus nisl. Ut ut bibendum erat. Curabitur at lectus lobortis, molestie orci vel, volutpat sem.

</p>\r\n', 'Łukasz Sosna', '2018-02-18 10:45:18', 'lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-elit'),

After entering the content, you need to give **article_id** a key property and add the auto incrementation property to it.

```
ALTER TABLE `yii_article`
  ADD PRIMARY KEY (`article_id`);
ALTER TABLE `yii_article`
  MODIFY `article_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=72;
```

Table of blog entries

The blog table will contain our entries. Depending on what you choose as your browsing criterion, the results of your browsing will be different. You will be able to choose a general page, a blog category or a keyword for the entry.

Create a yii_blog table:

- blog_id - unique identifier of the blog entry
- blog_title - title of the entry
- blog_text - content of the entry
- blog_date - date of entry
- blog_url - modified title of the entry, which will also appear in the URL address
- blog_category - category identification number
- blog_hashtag - content description through keywords

```
CREATE TABLE `yii_blog` (
  `blog_id` int(11) NOT NULL,
  `blog_title` varchar(150) NOT NULL,
  `blog_text` text NOT NULL,
  `blog_date` datetime NOT NULL,
  `blog_url` varchar(150) NOT NULL,
  `blog_category` int(11) NOT NULL,
  `blog_hashtag` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

We will post several dozen more entries in our blog.

```
INSERT INTO `yii_blog` (`blog_id`, `blog_title`, `blog_text`, `blog_date`, `blog_url`, `blog_category`,
`blog_hashtag`) VALUES
```

```
(7, 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium.', '<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium. Suspendisse odio magna, aliquam sed nulla sed, rutrum auctor neque. Pellentesque tincidunt massa eget tellus vehicula malesuada. Pellentesque accumsan aliquet sagittis. Quisque mollis leo faucibus cursus consequat. Morbi elementum, velit eu aliquet pulvinar, quam ex lobortis lectus, ut blandit risus augue a felis. In gravida varius arcu, at vulputate purus eleifend vestibulum. Nunc efficitur felis nec libero varius vehicula. In sed turpis purus. Nunc feugiat nisl ac augue viverra scelerisque. Donec sit amet augue in leo lobortis dictum. Sed id tortor eget metus gravida convallis in et metus. Sed posuere turpis ultrices, vulputate diam ac, suscipit erat. Mauris eleifend urna at erat faucibus, vitae feugiat ex aliquam. Duis at diam vitae neque maximus vehicula. Phasellus fermentum suscipit velit, at gravida leo varius at. Nunc pharetra nibh ac tortor scelerisque luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aenean sagittis turpis quis dictum consequat. Sed a tellus non ipsum semper commodo at tincidunt ligula. Proin ullamcorper orci et turpis hendrerit, quis ullamcorper lectus varius. Cras sollicitudin diam egestas efficitur sollicitudin. Integer justo augue, fringilla vel velit non, aliquet luctus arcu. Nullam sodales, est at consectetur fermentum, massa tellus fermentum nisl, ac rhoncus velit lectus quis magna. Mauris non eros justo. Cras luctus dui a est laoreet interdum quis et quam. Quisque vel dolor molestie, euismod sapien ut, vehicula leo. Mauris eu rhoncus dolor. Nulla in erat at est faucibus porta id nec lacus. Nam aliquet fermentum tellus vitae fermentum. Morbi euismod maximus urna eget finibus. Vestibulum ultrices, nulla eu dictum fringilla, sapien orci commodo dui, sed vestibulum turpis enim sit amet ante. Sed elementum tortor non quam ullamcorper lacinia. Etiam elit ligula, laoreet sed sollicitudin eu, feugiat sit amet nulla. Ut sit amet lorem dui. Duis ultrices, justo at euismod accumsan, arcu leo ultricies sapien, non interdum mi risus vitae mi. Fusce lectus elit, vulputate imperdiet velit nec, tempor bibendum tellus. Fusce tincidunt elementum nibh, vitae lacinia libero varius nec. Donec suscipit, diam accumsan aliquet maximus, dui dolor lobortis nibh, non tempor dolor risus sed magna. Curabitur sit amet scelerisque lacus, sit amet finibus elit. Mauris tempor, orci at tempor rhoncus, odio magna tincidunt odio, in commodo diam turpis ac erat. Suspendisse nulla diam, ultrices vel efficitur ac, imperdiet sed orci. Suspendisse egestas dui eu ipsum fringilla mollis. Duis gravida interdum consequat. Integer blandit nulla vitae dignissim vehicula. Donec placerat nisl a tempor pulvinar. Etiam viverra porta ornare. Nullam cursus orci metus, ut tempor elit elementum ut. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In leo neque, feugiat vel diam vel, molestie cursus nisl. Ut ut bibendum erat. Curabitur at lectus lobortis, molestie orci vel, volutpat sem.</p>\r\n', '2018-02-18 10:44:27', 'lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-elit-maecenas-id-nulla-nec-dui-efficitur-pretium', 2, 'interests software development'),
```

Finally, for the record identifier, assign a primary key and an auto-numbering of subsequent records.

```
ALTER TABLE `yii_blog`
  ADD PRIMARY KEY (`blog_id`),
  ADD KEY `blog_category` (`blog_category`);
ALTER TABLE `yii_blog`
  MODIFY `blog_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=50;
```

Table containing the category of the blog

It's time to create a table with the blog categories. Category numbers from this table will be entered into the field in the table containing entries in the blog.

Create the table: yii_blog_category:

- category_id - unique record identifier
- category_title - title of the category
- category_url - a revised title for the category, which can be found in the URL address

```
CREATE TABLE `yii_blog_category` (
  `category_id` int(11) NOT NULL,
  `category_title` varchar(150) NOT NULL,
  `category_url` varchar(150) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

We add our sample categories.

```
INSERT INTO `yii_blog_category` (`category_id`, `category_title`, `category_url`) VALUES
(2, 'Category 2', 'category-2'),
(5, 'Category 1', 'category-1');
```

Finally, the record identifier must be assigned a primary key property and auto-numbered.

```
ALTER TABLE `yii_blog_category`
  ADD PRIMARY KEY (`category_id`);
ALTER TABLE `yii_blog_category`
  MODIFY `category_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
```

System configuration table

The purpose of the table is to store configuration data concerning our CMS system, such as: e-mail address, title, keywords, page description or copyright note.

Create the yii_config table:

- config_id - a unique field by which the record can be identified
- config_name - field containing the name of the configuration option
- config_value - field containing the value of the option

```
CREATE TABLE `yii_config` (
  `config_id` int(11) NOT NULL,
  `config_name` varchar(15) NOT NULL,
  `config_value` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

In the next step you should fill in the table with the data that will be used to generate the page.

```
INSERT INTO `yii_config` (`config_id`, `config_name`, `config_value`) VALUES
(1, 'title', 'Page written at Yii'),
(2, 'description', 'Page written using the Yii framework'),
(3, 'keywords', 'framweork, yii'),
(4, 'rootemail', 'lukasz@lukasz.sos.pl'),
(5, 'foot', 'Copyright &copy; 2018 by <a href=\"http://en.lukasz.sos.pl\" target=\"_blank\">Lukas Sosna</a>');
```

After creating a table and filling it with data, we set the **config_id** column to the value of the basic key and give the feature of auto-increment.

```
ALTER TABLE `yii_config`
  ADD PRIMARY KEY (`config_id`);
ALTER TABLE `yii_config`
  MODIFY `config_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=6;
```

Table of contact details

Since our system will have its own contact system, you should also create a table that will contain data such as name and e-mail address. With this table you will later create a list from which you can select the recipient of the letter.

Create a `yii_contact` table:

- `contact_id` - unique record identifier
- `contact_email` - e-mail address to which the letter will be sent
- `contact_name` - name to be displayed in the address selection field

```
CREATE TABLE `yii_contact` (  
  `contact_id` int(11) NOT NULL,  
  `contact_email` varchar(75) NOT NULL,  
  `contact_name` varchar(55) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Let's put a few more records in order to present the form in its full glory.

```
INSERT INTO `yii_contact` (`contact_id`, `contact_email`, `contact_name`) VALUES  
(1, 'lukasz@lukasz.sos.pl', 'Technical assistance');
```

Now set the **`contact_id`** field to the primary key and give it the auto-increment capability.

```
ALTER TABLE `yii_contact`  
  ADD PRIMARY KEY (`contact_id`);  
ALTER TABLE `yii_contact`  
  MODIFY `contact_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
```

Table for information on downloads

Files will be an important part of our system. By downloading them you will learn how to create file statistics. You can, of course, download such data from the server logs, but there is no point in doing so with our website, where you will be able to save much more data than is contained in the system logs.

Create a `yii_download` table:

- `download_id` - record identifier
- `download_title` - the title of the file placed in the download area
- `download_text` - description of the file placed on our website
- `download_file` - the name of the file that we will make available to you for downloading
- `download_version` - file version
- `download_filesize` - file size
- `download_license` - license of file

```
CREATE TABLE `yii_download` (  
  `download_id` int(11) NOT NULL,
```

```

`download_title` varchar(150) NOT NULL,
`download_text` text NOT NULL,
`download_file` varchar(65) NOT NULL,
`download_version` varchar(25) NOT NULL,
`download_filesize` varchar(25) NOT NULL,
`download_license` varchar(25) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

We add files.

```

INSERT INTO `yii_download` (`download_id`, `download_title`, `download_text`, `download_file`,
`download_version`, `download_filesize`, `download_license`) VALUES

(1, 'PHP programme', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui
efficitur pretium. Suspendisse odio magna, aliquam sed nulla sed, rutrum auctor neque. Pellentesque
tincidunt massa eget tellus vehicula malesuada. Pellentesque accumsan aliquet sagittis. Quisque mollis leo
faucibus cursus consequat. Morbi elementum, velit eu aliquet pulvinar, quam ex lobortis lectus, ut blandit
risus augue a felis. In gravida varius arcu, at vulputate purus eleifend vestibulum.', 'file.zip',
'1.0.0', '23 MB', 'freeware'),

```

At the end we give the **download_id** field the feature of the primary key and the option of auto-incrementation.

```

ALTER TABLE `yii_download`
ADD PRIMARY KEY (`download_id`);
ALTER TABLE `yii_download`
MODIFY `download_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=8;

```

Statistics table for downloads

This table will be used by us to place statistics when you download files to our website. Here you will find things like your browser, the file you downloaded, the exact date and time, and IP.

Create the yii_download_stats table:

- stat_id - unique identifier
- stat_browser - information about the user's browser
- stat_file - downloaded file
- stat_date - date and time of download
- stat_ip - the IP address of the download

```

CREATE TABLE `yii_download_stats` (
`stat_id` int(11) NOT NULL,
`stat_browser` varchar(75) NOT NULL,
`stat_file` varchar(65) NOT NULL,
`stat_date` datetime NOT NULL,
`stat_ip` varchar(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

The option of the field containing the main key should be added to **stat_id** field and it should be given auto-incremental properties.


```
ALTER TABLE `yii_download_stats`
  ADD PRIMARY KEY (`stat_id`);
ALTER TABLE `yii_download_stats`
  MODIFY `stat_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=7;
```

Error404 table on our website

Because we will use address rewriting by Apache we can create our own page containing information about error 404, i.e. lack of indicated element in URL.

Create the table: yii_error404:

- error_id - unique record identifier
- error_page_from - information from which address the entrance was made
- error_page - information on what element is missing
- error_date - the exact date and time of the error

```
CREATE TABLE `yii_error404` (
  `error_id` int(11) NOT NULL,
  `error_page_from` text NOT NULL,
  `error_page` text CHARACTER SET latin2 NOT NULL,
  `error_date` datetime NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

After creating the table, the **error_id** field should be marked as the primary key and given the auto-increment capability.

```
ALTER TABLE `yii_error404`
  ADD PRIMARY KEY (`error_id`);
ALTER TABLE `yii_error404`
  MODIFY `error_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;
```

Table for administrative logs

Administrator logs are very important to be able to track when we last logged into the system to our account and whether someone entered the administrator panel and made any changes to the service.

Create the table: yii_log:

- log_id - unique record identifier
- log_user_id - identifier of the user executing the action
- log_what - message which content has been added, changed or deleted
- log_time - date and time of execution of the action
- log_ip - the IP number of the computer from which the action was executed

```
CREATE TABLE `yii_log` (
```

```

`log_id` int(11) NOT NULL,
`log_user_id` int(11) NOT NULL,
`log_what` varchar(50) NOT NULL,
`log_time` datetime NOT NULL,
`log_ip` varchar(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

For the **log_id** field we set the value of the primary key and give the auto-increment property.

```

ALTER TABLE `yii_log`
  ADD PRIMARY KEY (`log_id`);
ALTER TABLE `yii_log`
  MODIFY `log_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=1;

```

Table of menu items

The menu will be visible on the left side of our website, it will contain the main elements, sub-elements and will display the appropriate content depending on whether you are logged in or not.

Create the table: yii_menu:

- menu_id - unique record identifier
- menu_title - the title of the menu item displayed to the user
- menu_pos - vertical position of the item
- menu_sub - a field specifying whether a given menu item is the main menu item or whether it belongs to another menu item
- menu_login - determines whether the menu item is visible to the user who is not logged in
- menu_what - contains the designation of which section the link refers to
- menu_content_id - the identifier of the item to which the menu leads
- menu_extra - additional field containing text additions to the URL

```

CREATE TABLE `yii_menu` (
  `menu_id` int(11) NOT NULL,
  `menu_title` varchar(255) NOT NULL,
  `menu_poz` int(11) NOT NULL,
  `menu_sub` int(11) NOT NULL,
  `menu_login` char(1) NOT NULL,
  `menu_what` varchar(25) NOT NULL,
  `menu_content_id` int(11) NOT NULL,
  `menu_extra` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

We add items to our menu.

```

INSERT INTO `yii_menu` (`menu_id`, `menu_title`, `menu_poz`, `menu_sub`, `menu_login`, `menu_what`,
`menu_content_id`, `menu_extra`) VALUES

```

```
(40, 'Home page', 1, 0, 'n', 'main', 0, ''),
(62, 'Text pages', 2, 0, 'n', 'page', 0, ''),
(63, 'Download', 5, 0, 'n', 'download', 0, ''),
(64, 'Blog', 3, 0, 'n', 'blog', 0, ''),
(65, 'Articles', 4, 0, 'n', 'article', 0, ''),
(66, 'Contact', 6, 0, 'n', 'contact', 0, ''),
(72, 'Curious', 7, 0, 'n', 'main', 0, ''),
(74, 'Blog entry', 1, 72, 'n', 'blogone', 7, 'lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-elit-
maecenas-id-nulla-nec-dui-efficitur-pretium'),
(75, 'Example page', 2, 72, 'n', 'pageone', 2, 'lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-elit'),
(76, 'Example article', 3, 72, 'n', 'articleone', 1, 'lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-
elit');
```

Then the menu **field_id** should be given the basic key property and auto-increment.

```
ALTER TABLE `yii_menu`
  ADD PRIMARY KEY (`menu_id`);
ALTER TABLE `yii_menu`
  MODIFY `menu_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=77;
```

Table of text pages

Our website will also contain text pages, thanks to which we will be able to present any content. Also, one of the pages will be selected as the page set on the home page.

Create the table: **yii_page**:

- **page_id** - unique record identifier in the table
- **page_title** - title of the page
- **page_text** - page content
- **page_url** - the title of the page processed in such a way that it can be part of the url address
- **page_main** - field containing information on whether the page is the home page

```
CREATE TABLE `yii_page` (
  `page_id` int(11) NOT NULL,
  `page_title` varchar(150) NOT NULL,
  `page_text` text NOT NULL,
  `page_url` varchar(150) NOT NULL,
  `page_main` char(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

We place several pages in our table.

```
INSERT INTO `yii_page` (`page_id`, `page_title`, `page_text`, `page_url`, `page_main`) VALUES
(1, 'Yii Home Page', 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aliquam tempor turpis mi,
at porttitor augue gravida id. Etiam in purus semper, placerat enim nec, porttitor magna. Proin lectus
massa, iaculis id placerat ornare, tincidunt quis arcu. Aliquam non faucibus turpis, sed mattis quam. Cras
pharetra a elit et faucibus. Cras eu purus at dolor consectetur eleifend tempus et mi. Nullam posuere
efficitur ornare. Sed maximus libero id nisi feugiat, a mattis enim tempus.\r\n\r\nMauris et euismod
```

purus. Donec lorem nibh, pretium in lacus id, consequat ultrices nisl. Nunc hendrerit, dolor non pellentesque placerat, sapien tortor gravida lacus, ac lobortis eros dolor ac enim. Quisque eget arcu blandit, iaculis nisi quis, fermentum tellus. Morbi facilisis nunc velit, at rhoncus erat tristique eget. Praesent commodo diam ipsum, sit amet dictum diam semper id. Praesent euismod ante ac nunc tincidunt, eu euismod ligula vestibulum.\r\n\r\nSuspendisse auctor diam quam, in laoreet enim sagittis quis. Quisque tincidunt ullamcorper pharetra. Vivamus dapibus rutrum efficitur. Sed malesuada tincidunt mi at porta. Ut congue orci vel dolor elementum, vel vestibulum nisl finibus. Quisque nec risus mi. Aliquam consequat nibh at tincidunt accumsan. Cras pharetra cursus cursus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Integer posuere tempus lectus, quis accumsan diam euismod et.\r\n\r\nMaecenas sagittis turpis ligula, interdum dapibus orci placerat eget. Sed a quam non ligula venenatis tincidunt non ut quam. Sed dapibus risus quis tellus lacinia ullamcorper nec ac diam. Sed sodales nisl et dolor malesuada viverra. Mauris lectus sapien, pharetra sed ante at, congue aliquet nulla. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis imperdiet urna sapien, in imperdiet diam eleifend id. Pellentesque placerat vulputate finibus. Etiam pharetra mauris imperdiet risus consectetur tincidunt. Pellentesque vel diam sem. Donec vitae posuere ante, in laoreet elit. Nulla pharetra sem scelerisque enim condimentum, id tincidunt ante placerat. Aenean egestas sem in commodo congue. Nam vel condimentum purus, non lobortis velit. Maecenas tincidunt volutpat sapien quis tempor. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.\r\n\r\nNulla euismod ex scelerisque, interdum tellus eget, vulputate purus. Integer in odio porttitor urna efficitur condimentum in at felis. Aenean fermentum elementum lectus quis tempor. Nullam consequat ultrices porta. Vestibulum nec facilisis turpis, ut finibus dolor. Proin faucibus, quam non hendrerit malesuada, massa erat egestas purus, non vulputate lacus nunc nec eros. Fusce ut lectus nisl. Aliquam tempor lorem quis sem dictum, id pharetra sem feugiat. Vivamus ut rhoncus ante. Aliquam mollis pellentesque mauris, sit amet viverra libero cursus ut.', 'lorem-ipsum', 'y');

The **page_id** field should then be assigned a primary key in the table, as well as an auto-increment option.

```
ALTER TABLE `yii_page`
  ADD PRIMARY KEY (`page_id`);
ALTER TABLE `yii_page`
  MODIFY `page_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=30;
```

Table of password reminders

Since our CMS will be equipped with the ability to recall the password for the user account, you need to create some kind of control system. If you periodically try to obtain a new password to your account, we may suspect that you are actually trying to break into your account.

Create the table: yii_password:

- password_id - unique record identifier in the database
- password_user_id - user ID
- password_hash1 - a random sequence of characters needed by the user to recall the password
- password_hash2 - a second random sequence of characters needed by the user to recall the password
- password_time - the maximum time for which the user can use the link included in the e-mail list he has maintained is the current time plus 2 hours
- password_time_used - date and time at which the link was clicked
- password_ip - the IP number of the computer from which you clicked to recall the password
- password_used - information whether the password reminder attempt has been completed

```
CREATE TABLE `yii_password` (
  `password_id` int(11) NOT NULL,
  `password_user_id` int(11) NOT NULL,
  `password_hash1` varchar(20) NOT NULL,
  `password_hash2` varchar(20) NOT NULL,
  `password_time` datetime NOT NULL,
```

```

`password_time_used` datetime NOT NULL,
`password_ip` varchar(15) NOT NULL,
`password_used` char(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

The **password_id** field must still be assigned a primary key for the table and the auto-increment option.

```

ALTER TABLE `yii_password`
  ADD PRIMARY KEY (`password_id`);
ALTER TABLE `yii_password`
  MODIFY `password_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=9;

```

Table with users

The most important table in our system, i.e. containing user data such as e-mail, password, date of creating an account or administrator's rights.

Create the table: yii_user:

- user_id - unique user identifier
- user_email - the unique e-mail address of the user, who will be his login at the same time
- user_namesurname - user's name
- user_phone - user's telephone
- user_www - address of the user's website
- user_about - information about the user such as interests, hobbies, work, etc.
- user_password - encrypted user password
- user_key - the user key used to activate the account
- user_register - date and time of registration in the service
- user_registered_ip - IP number from which the user has been registered with the service
- user_active - information whether the user has an active account
- user_activated - date and time when the user activated his account
- user_activated_ip - IP address from which the account was activated
- user_root - information whether the user has administrative privileges

```

CREATE TABLE `yii_user` (
  `user_id` int(11) NOT NULL,
  `user_email` varchar(90) NOT NULL,
  `user_namesurname` varchar(85) NOT NULL,
  `user_phone` varchar(20) NOT NULL,
  `user_www` varchar(120) NOT NULL,
  `user_about` text NOT NULL,
  `user_password` varchar(255) NOT NULL,
  `user_key` varchar(20) NOT NULL,
  `user_register` datetime NOT NULL,
  `user_registered_ip` varchar(15) NOT NULL,

```

```

`user_active` char(1) NOT NULL,
`user_activated` datetime NOT NULL,
`user_activated_ip` varchar(15) NOT NULL,
`user_root` char(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

```

Then we insert our first user into the table.

```

INSERT INTO `yii_user` (`user_id`, `user_email`, `user_namesurname`, `user_phone`, `user_www`,
`user_about`, `user_password`, `user_key`, `user_register`, `user_registered_ip`, `user_active`,
`user_activated`, `user_activated_ip`, `user_root`) VALUES
(1, 'lukasz@lukasz.sos.pl', 'Lukas Sosna', '', 'http://en.lukasz.sos.pl', '',
'$2y$10$o0.1ukLMSTYAiuQhrp69SepO6RY/Whu/XSucQJwzEs6eg03oHMa3a', 'dvm6yw4yukwrlgmsx9jq', '2018-02-03
07:18:26', '127.0.0.1', 'y', '2018-02-03 09:22:44', '127.0.0.1', 'y');

```

Add to the **user_id** field information about the fact that it is a primary key and an auto-increment field.

```

ALTER TABLE `yii_user` ADD PRIMARY KEY (`user_id`);
ALTER TABLE `yii_user` MODIFY `user_id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=2;

```

Chapter 3. User interface

User model

The purpose of the model will be to support all actions that the user will perform on our website. From registration through logging in to profile completion.

File location: ../models/User.php

Start the PHP file and set the namespace to **app\models**, then load the Yii framework and the **Model** class.

```
<?php
namespace app\models;
use Yii;
use yii\base\Model;
```

We create a class which must have the same name as the file and be an extension of the **Model** parent class in this case.

```
class User extends Model
{
```

Let's now declare variables, the visibility of which will be set to public, in order to be able to use them in controllers and in generating views. Ideally, the variable names should correspond to the columns in the table. Thanks to this we will be able to use the **ActiveRecord** interface, as we will do when creating the administrator panel.

```
public $user_id;
public $user_email;
public $user_password;
public $user_password2;
public $user_password3;
public $user_key;
public $user_register;
public $user_registered_ip;
public $user_active;
public $user_activated;
public $user_activated_ip;
public $user_root;
public $user_namesurname;
public $user_phone;
public $user_www;
public $user_about;
public $user_captcha;
public $captcha;
public $password_hash1;
```

```
public $password_hash2;
```

The method that returns the name of a table to us. We have to remember to surround the table name with bracket markers together with the percentage. This will allow you to set any prefix you like. The name of the table will be created together with the prefix, which we will not have to worry about while creating our scripts.

```
public static function tableName()  
{  
    return '{{%user}}';  
}
```

Because the class for the user will contain actions such as: logging in, registration, changing the password, updating the profile, activating the account and reminding the password, let's create different scenarios for each action, which we will define later when initializing the model.

```
const SCENARIO_LOGIN = 'login';  
const SCENARIO_REGISTER = 'register';  
const SCENARIO_CHANGEPASSWORD = 'changepassword';  
const SCENARIO_UPDATEPROFILE = 'updateprofile';  
const SCENARIO_ACTIVATE = 'activate';  
const SCENARIO_REMIND = 'remind';
```

We define appropriate fields that will enter the action. If there is no field in the scenario definition method, the value will not be visible in the variables received from the user, even though it has been sent. The scenarios are declared in the way of constant use, which we defined the line above, and details of fields that will be sent.

```
public function scenarios()  
{  
    return [  

```

The login scenario requires you to send us your email address and password.

```
self::SCENARIO_LOGIN => ['user_email', 'user_password'],
```

The registration scenario will require your e-mail address, password, password confirmation and optional website address, name, phone number and user note.

```
self::SCENARIO_REGISTER => ['user_email', 'user_password', 'user_password2', 'user_www',  
    'user_namesurname', 'user_phone', 'user_about'],
```

The change password scenario will require the user to send the old password, a new password and confirm the new one.

```
self::SCENARIO_CHANGEPASSWORD => ['user_password', 'user_password2', 'user_password3'],
```

The profile update scenario will optionally accept the website address, your name, your telephone number and any information you require.


```
self::SCENARIO_UPDATEPROFILE => ['user_www', 'user_namesurname','user_phone','user_about'],
```

The account activation scenario will require a user ID and, optionally, a web address.

```
self::SCENARIO_ACTIVATE => ['user_id', 'user_www'],
```

The password reminder scenario will require the user's e-mail address and the transmission of the **Captcha** code.

```
self::SCENARIO_REMIND => ['user_email','user_captcha'],
];
}
```

We define the appropriate fields in the **rules()** method together with the values that they may contain and the appropriate messages in case of an error in their content.

```
public function rules()
{
return [
```

The login scenario contains e-mail fields and passwords, which are required. The next line is that the user's e-mail should match the regular expression that specifies the e-mail address. The third line contains a reference to the **validateUserName** method in which you will check if such a user exists and log in to your account.

```
[['user_email', 'user_password'], 'required', 'on' => self::SCENARIO_LOGIN],
['user_email', 'email', 'on' => self::SCENARIO_LOGIN],
['user_email', 'validateUserName', 'skipOnError' => false, 'on' => self::SCENARIO_LOGIN],
```

The change password scenario requires you to enter the old password, enter the new password and repeat the new one. The second line verifies if the new passwords are between 8 and 30 characters long. The third line checks if the new password matches the regular expression, which defines the strength of the password consisting of lower case letters, upper case letters and numbers. If any character is missing in the new password, an error will be returned to the user. The fourth line concerns the comparison of new passwords, where the program checks if the new password is the same as its repetition. The fifth line is to call the **validateUserPassword** method to check whether the old password entered in the form matches the user password entered in the database.

```
[['user_password', 'user_password2', 'user_password3'], 'required', 'on' =>
self::SCENARIO_CHANGEPASSWORD],
[['user_password2', 'user_password3'], 'string', 'length' => [8, 30], 'on' =>
self::SCENARIO_CHANGEPASSWORD],
[['user_password2'], 'match', 'pattern' => '/^.*(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).*$/ ', 'message' =>
Yii::t('app', 'new_password_need_to_have'), 'on' => self::SCENARIO_CHANGEPASSWORD],
['user_password2', 'compare', 'compareAttribute' => 'user_password3', 'on' =>
self::SCENARIO_CHANGEPASSWORD],
['user_password', 'validateUserPassword', 'skipOnError' => false, 'on' => self::SCENARIO_CHANGEPASSWORD],
```

The profile update scenario, if you enter the URL in the field containing the user's web page, checks whether the address was preceded by the word http, if this word is missing, the program will correct the user error.

```
['user_www', 'url', 'defaultScheme' => 'http', 'on' => self::SCENARIO_UPDATEPROFILE],
```

The registration scenario will require fields with e-mail address, password and account repetition. The second line will check if the entered e-mail matches the regular expression and complies with the address definition standard. The third line checks if the password and its repetition are between 8 and 30 characters long. The fourth line checks whether the password contains at least one upper case letter, lower case letter and number by means of a regular expression. The fifth line will check if the password is identical to the one you repeated. The sixth line in turn is responsible for the correct formatting of the URL, i.e. adding HTTP to it. The seventh line contains a reference to the **validateUserIsEmail** method which checks whether a user with such an e-mail address already exists in the system and displays the appropriate message in case of its presence.

```
[[['user_email', 'user_password', 'user_password2'], 'required', 'on' => self::SCENARIO_REGISTER],
['user_email', 'email', 'on' => self::SCENARIO_REGISTER],
[['user_password', 'user_password2'], 'string', 'length' => [8, 30], 'on' => self::SCENARIO_REGISTER],
[['user_password'], 'match', 'pattern' => '/^.*(?:.*\d)(?=.*[a-z])(?=.*[A-Z]).*$/ ', 'message' =>
Yii::t('app', 'new_password_need_to_have'), 'on' => self::SCENARIO_REGISTER],
['user_password', 'compare', 'compareAttribute' => 'user_password2', 'on' => self::SCENARIO_REGISTER],
['user_www', 'url', 'defaultScheme' => 'http', 'on' => self::SCENARIO_REGISTER],
['user_email', 'validateUserIsEmail', 'skipOnError' => false, 'on' => self::SCENARIO_REGISTER],
```

The password reminder scenario requires the user to fill in the fields with the **Captcha** code and enter his e-mail address. The second line checks if the e-mail address matches the pattern. The third line refers to the user controller and the **captcha** method, because it is responsible for validating whether the code has been correctly rewritten. The fourth line contains a reference to the **validateUserRemind** method, which places the entries in the table responsible for executing the access code reminder.

```
[[['user_captcha', 'user_email'], 'required', 'on' => self::SCENARIO_REMIND],
['user_email', 'email', 'on' => self::SCENARIO_REMIND],
['user_captcha', 'captcha', 'captchaAction' => 'user/captcha', 'on' => self::SCENARIO_REMIND],
['user_email', 'validateUserRemind', 'skipOnError' => false, 'on' => self::SCENARIO_REMIND],
];
}
```

A method that allows the user to be activated, which accepts as arguments the user ID and its key assigned to him during registration.

```
public function ActivateUser($UserId,$UserKey)
{
```

We create a command which as a parameter takes the user identification number loaded late with the **bindParam** method, which additionally protects against injecting the malicious **SQL** code and selects only one record (**queryOne**).

```
$QueryData = Yii::$app->db->createCommand('SELECT user_id,user_key,user_active FROM {{%user}} WHERE
user_id = :user_id AND user_active = ""')
->bindParam(':user_id', $UserId)
->queryOne();
```

If the result of the query is set to false, then from the whole method we also return false by typing it into the appropriate variable.

```
if($QueryData == false)
{
$ToReturn = false;
```

```

}
else
{

```

When the record was found, we generate the current date and collect the IP address that is currently used by the user.

```

$NowDate = date('Y-m-d H:i:s');
$IpOfUser = $_SERVER['REMOTE_ADDR'];

```

Then we create a query to update the table adding information that the user already has an active account (**user_active**) and places the rest of the data by declaring them as parameters, and then adding values from the variables (**bindParam**).

```

Yii::$app->db->createCommand('UPDATE {%user} SET
user_active = "y",
user_activated = :user_activated,
user_activated_ip = :user_activated_ip
WHERE
user_id = :user_id
AND
user_key = :user_key
')
->bindParam(':user_id', $UserId)
->bindParam(':user_activated', $NowDate)
->bindParam(':user_activated_ip', $IpOfUser)
->bindParam(':user_key', $UserKey)
->execute();

```

In such a case the method must return true value, i.e. information that everything went correctly.

```

$ToReturn = true;
}

```

We return the value to the parent method.

```

return $ToReturn;
}

```

The method of registering a new user account will not take the values of the variables as parameters, but will use the variables defined by us within the class.

```

public function RegisterUser()
{

```

The first thing is to declare a "salt" which will be glued to the password given by the user in the function which encrypts the password.

```
$Salt = Yii::$app->params['saltPassword'];
$UserPassword = password_hash($this->user_password.$Salt, PASSWORD_DEFAULT);
```

Now we download the drawn string of characters, the current date and the IP address of the user of our website.

```
$UserKey = strtolower(Yii::$app->getSecurity()->generateRandomString(20));
$UserRegisterDate = date('Y-m-d H:i:s');
$UserRegisterIp = $_SERVER['REMOTE_ADDR'];
```

We create an **SQL** command that places user data in a database, where we use parameters that we later supplement with variable values.

```
$QueryData = Yii::$app->db->createCommand('INSERT INTO {%user}
(user_email,user_password,user_key,user_register,user_registered_ip,user_namesurname,user_phone,user_www,u
ser_about)
values
(:user_email,:user_password,:user_key,:user_register,:user_registered_ip,:user_namesurname,:user_phone,:us
er_www,:user_about)
')
->bindParam(':user_namesurname', htmlspecialchars($this->user_namesurname))
->bindParam(':user_phone', htmlspecialchars($this->user_phone))
->bindParam(':user_www', htmlspecialchars($this->user_www))
->bindParam(':user_about', htmlspecialchars($this->user_about))
->bindParam(':user_email', $this->user_email)
->bindParam(':user_password', $UserPassword)
->bindParam(':user_key', $UserKey)
->bindParam(':user_register', $UserRegisterDate)
->bindParam(':user_registered_ip', $UserRegisterIp)
->execute();
```

The user you just inserted must still be selected and the result passed on to the calling method.

```
$QueryData = Yii::$app->db->createCommand('SELECT user_id,user_key FROM {%user} WHERE user_email =
:user_email')
->bindParam(':user_email', $this->user_email)->queryOne();

return $QueryData;
}
```

A method that allows you to select profile data. This method does not accept any data for the purpose of identifying the user from the session.

```
public function SelectProfile()
{
```

Downloading the variable session starts with referring to the properties of the Yii framework, and then by declaring its name.

```
$session = Yii::$app->session;
```

```
$UserId = $session['yii_user_id'];
```

We create a query which will select data from the table of users concerning this one account.

```
$QueryData = Yii::$app->db->createCommand('SELECT
user_id,user_email,user_namesurname,user_phone,user_www,user_about,user_register,user_registered_ip,user_a
ctive,user_activated,user_activated_ip,user_root FROM {%user}} WHERE user_id = :user_id')
->bindParam(':user_id', $UserId)->queryOne();
```

When we have found a user, we can rewrite it to the variable that will be returned.

```
if($QueryData != false)
{
$ProfileData = $QueryData;
}
```

We return the result of the method.

```
return $ProfileData;
}
```

The method of profile update does not accept any parameters, as we will use a session to identify the user and the variables declared in the current class.

```
public function UpdateProfile()
{
```

We retrieve your user ID from the session.

```
$session = Yii::$app->session;
$UserId = $session['yii_user_id'];
```

We create a query to update all data in the profile, where the user ID is the one from the session.

```
$QueryData = Yii::$app->db->createCommand('UPDATE {%user}} SET user_namesurname = :user_namesurname,
user_phone = :user_phone, user_www = :user_www, user_about = :user_about
WHERE
user_id = :user_id')
->bindParam(':user_namesurname', htmlspecialchars($this->user_namesurname))
->bindParam(':user_phone', htmlspecialchars($this->user_phone))
->bindParam(':user_www', htmlspecialchars($this->user_www))
->bindParam(':user_about', htmlspecialchars($this->user_about))
->bindParam(':user_id', $UserId)->execute();
}
```

The method of updating the password does not have any parameters, because it retrieves data from the session and from the variables inside the class.

```
public function UpdatePassword()
{
```

We retrieve the user ID from the session.

```
$session = Yii::$app->session;
$UserId = $session['yii_user_id'];
```

Select the parameter containing the "salt" that will be attached to the end of the new password, and encrypt it using the **password_hash** function.

```
$Salt = Yii::$app->params['saltPassword'];
$ReadyPassword = password_hash($this->user_password2.$Salt, PASSWORD_DEFAULT);
```

Then we create a query which will update our user profile and replace the old one with a new one.

```
$QueryData = Yii::$app->db->createCommand('UPDATE {%user} SET user_password = :user_password WHERE
user_id = :user_id')
->bindParam(':user_password', $ReadyPassword)->
bindParam(':user_id', $UserId)->execute();
}
```

The method used to set the password to new when the user clicks on the link resetting the old password in the e-mail sent to his mailbox. The method adopts three parameters: user ID, password hash, and second password hash. Passwords are needed to check whether a click actually comes from an email and is not an attempt to misappropriate someone's account.

```
public function SetNewPassword($UserId,$UserHash1,$UserHash2)
{
```

We create a query selecting a password that matches the criteria of the loaded variables using **bindParam**.

```
$QueryDataIs = Yii::$app->db->createCommand('SELECT * FROM {%password} WHERE
password_user_id = :password_user_id
AND
password_hash1 = :password_hash1
AND
password_hash2 = :password_hash2
AND
password_used = ""
ORDER BY
password_id DESC
')
->bindParam(':password_user_id', $UserId)
->bindParam(':password_hash1', $UserHash1)
->bindParam(':password_hash2', $UserHash2)
->queryOne();
```

We check whether the result is empty.

```
if($QueryDataIs != false)
{
```

We generate the date and time with the deduction of two hours, because only for two hours you can use the sent link in the e-mail message.

```
$DateAndTimePlus = date('Y-m-d H:i:s', strtotime('-2 hours',time()));
```

We execute the query together with the time when you can change the password as a parameter.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%password}} WHERE
password_user_id = :password_user_id
AND
password_hash1 = :password_hash1
AND
password_hash2 = :password_hash2
AND
password_time > :password_time
AND
password_used = ""
ORDER BY
password_id DESC
')
->bindParam(':password_user_id', $UserId)
->bindParam(':password_hash1', $UserHash1)
->bindParam(':password_hash2', $UserHash2)
->bindParam(':password_time', $DateAndTimePlus)
->queryOne();
```

Set the value that will be returned from the method to **false**.

```
$ReturnedValue = false;
```

We check if the query returned any value.

```
if($QueryData != false)
{
```

We select the user on the basis of the identifier.

```
$QueryDataUser = Yii::$app->db->createCommand('SELECT user_id,user_email FROM {{%user}} WHERE user_id =
:user_id')
->bindParam(':user_id', $UserId)->queryOne();
```

The e-mail address is loaded into the internal class variable from the result of the query.

```
$this->user_email = $QueryDataUser['user_email'];
```

We create a temporary password by drawing a string, then downloading "salt" and adding it to the password, which is encrypted using the **password_hash()** function. Then you enter the newly generated password into the variable of our class.

```
$TemporaryPasword = strtolower(Yii::$app->getSecurity()->generateRandomString(20));  
$Salt = Yii::$app->params['saltPassword'];  
$ReadyPassword = password_hash($TemporaryPasword.$Salt, PASSWORD_DEFAULT);  
$this->user_password = $TemporaryPasword;
```

Now you can execute a query to update your password.

```
Yii::$app->db->createCommand('UPDATE {${user}} SET user_password = :user_password WHERE user_id =  
:user_id')  
->bindParam(':user_password', $ReadyPassword)->  
bindParam(':user_id', $UserId)->execute();
```

You probably wonder why to load these values into the variables inside the class instead of creating yourself a variable inside the method and enter the value into it. You need to know that after changing the password you need to send it to the user. That's why we need these values, so that you don't have to return data that can be dangerous. By maintaining these values in the controller we will simply use only the variables of the model.

We generate the current time and collect the user's IP address.

```
$PasswordUserTime = date('Y-m-d H:i:s');  
$UserPasswordIp = $_SERVER['REMOTE_ADDR'];
```

Then you need to update the record based on which you changed your password to make it impossible to use it again.

```
Yii::$app->db->createCommand('UPDATE {${password}} SET password_ip = :password_ip, password_time_used =  
:password_time_used, password_used = "y" WHERE password_id = :password_id')  
->bindParam(':password_ip', $UserPasswordIp)  
->bindParam(':password_time_used', $PasswordUserTime)  
->bindParam(':password_id', $QueryData['password_id'])->execute();
```

To the variable which we are going to return we enter the value **true**, i.e. the information that the operation went well.

```
$ReturnedValue = true;  
}  
}
```

If the query does not return the information about the existing user ID, passwords and time, then the false value is entered into the variable, which means that the password could not be changed.


```

else
{
$ReturnValue = false;
}

```

We return the value from the method.

```

return $ReturnValue;
}

```

The method used to generate the data into a table with which the password can be reset. Because a method is used in validation rules it has three arguments: name of the field, parameters and type of validation.

```

public function validateUserRemind($attribute, $params, $validator)
{

```

We create a query selecting the user who has an account with the indicated e-mail address and it is active.

```

$queryData = Yii::$app->db->createCommand('SELECT user_id FROM {%user} WHERE user_email = :user_email
AND user_active = "y"')
->bindParam(':user_email', $this->user_email)->queryOne();

```

If there is no query result, we must display the error. This is done using the **addError()** method, where in the first parameter the field name is declared, in this case it is the value of the \$attribute variable and in the second parameter the message is displayed.

```

if($queryData == false)
{
$this->addError($attribute, Yii::t('app', 'account_dont_exists'));
}
else
{

```

The user has been found so we rewrite his identifier. Then we generate the current date and two random hashes.

```

$userId = $queryData['user_id'];
$dateTimeSet = date("Y-m-d H:i:s");
$hash1 = strtolower(Yii::$app->getSecurity()->generateRandomString(20));
$hash2 = strtolower(Yii::$app->getSecurity()->generateRandomString(20));

```

We create a query placing the data generated by us in a table used to validate the password reminder process.

```

Yii::$app->db->createCommand('INSERT INTO {%password} (password_user_id, password_hash1, password_hash2,
password_time) VALUES (:password_user_id, :password_hash1, :password_hash2, :password_time)')
->bindParam(':password_user_id', $userId)
->bindParam(':password_hash1', $hash1)
->bindParam(':password_hash2', $hash2)

```

```
->bindParam(':password_time', $DateTimeSet)
->execute();
```

Because we will need this data to be transcribed to the internal variables of the class, we still need to transcribe it to have access to it from our controller.

```
$this->password_hash1 = $Hash1;
$this->password_hash2 = $Hash2;
$this->user_id = $UserId;
}
}
```

The method that will check if the user with the given e-mail address already exists in the system. It takes three parameters because it has been used in the method designed to validate data. The first variable is the field property to be validated, the next is the parameters, and finally there is the selected validator.

```
public function validateUserIsEmail($attribute, $params, $validator)
{
```

We create a query that selects one record from a table of users where the user's e-mail is the same as the one submitted to the method during validation.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%user}} WHERE user_email = :user_email')
->bindParam(':user_email', $this->user_email)->queryOne();
```

We check if the variable is not empty and in this case we should return the error due to the fact that the user with such e-mail address already exists. This is done using the **addError()** method, where the first argument is the field name and the second is the message.

```
if($QueryData != false)
{
    $this->addError($attribute, Yii::t('app', 'account_email_is_registered'));
}
}
```

The method allowing to verify whether the user password corresponds to the one entered in the form on the website. Because the method is run by the validation rule it has three parameters: the name of the field, the parameters and the validator used.

```
public function validateUserPassword($attribute, $params, $validator)
{
```

Load the session property and retrieve the user ID from it.

```
$session = Yii::$app->session;
$UserId = $session['yii_user_id'];
```

We create a query that selects your data based on your identification number.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%user}} WHERE user_id = :user_id')
->bindParam(':user_id', $UserId)->queryOne();
```

We check if the result returned is not empty.

```
if($QueryData != false)
{
```

Download the "salt" from the system configuration which will be added to the password.

```
$Salt = Yii::$app->params['saltPassword'];
```

Using the **password_verify()** function we check if the password entered in the form matches the password stored in the database.

```
if(!password_verify($this->user_password.$Salt,$QueryData['user_password']))
{
```

If the password does not match the one saved in the database, an error should be generated by means of the **addError()** method, which takes the name of the field as the first argument, and in the second one we define the content of the message.

```
$this->addError($attribute, Yii::t('app', 'password_dont_match'));
}
}
}
```

The method to be validated determines whether there is a user with the given e-mail address and writes out the data to the session which will result in his logging in. It is called from the method used to validate the contents of the form field. It has three parameters: the name of the form field, the parameters and the definition of the validator used.

```
public function validateUserName($attribute, $params, $validator)
{
```

We create a query that selects the data from the user table, where the user's e-mail is the same as in the class field.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%user}} WHERE user_email = :user_email')
->bindParam(':user_email', $this->user_email)->queryOne();
```

When the result is empty, we return the error using the **addError()** method, where in the first parameter we enter the name of the field and in the second parameter the message is displayed to the user.

```
if($QueryData == false)
{
$this->addError($attribute, Yii::t('app', 'no_user_with_email'));
}
```

```
else
```

```
{
```

If a user exists, we have to check whether he has an active account.

```
if($QueryData['user_active'] == 'y')
```

```
{
```

We take from the parameters the framework "salt", which then in the function of **password_verify()** is used for validation whether the password entered in the form field corresponds to the password saved in the database.

```
$Salt = Yii::$app->params['saltPassword'];
```

```
if(password_verify($this->user_password.$Salt,$QueryData['user_password']))
```

```
{
```

Load the session and then enter the values obtained from the database into it. This allows us to validate session variables on the website and check whether the user is logged in.

```
$session = Yii::$app->session;
```

```
$session['yii_user_id'] = $QueryData['user_id'];
```

```
$session['yii_user_email'] = $QueryData['user_email'];
```

```
$session['yii_user_root'] = $QueryData['user_root'];
```

```
}
```

```
else
```

```
{
```

If the password did not match the one saved in the database, we return the error using the **addError()** method, in which we enter the name of the field in the first parameter and a message in the second one.

```
$this->addError($attribute, Yii::t('app', 'password_dont_match'));
```

```
}
```

```
}
```

```
else
```

```
{
```

If the user account is inactive, the user should be given the appropriate error message using the **addError()** method, where the name of the field is given in the first argument and the name of the field in the second one.

```
$this->addError($attribute, Yii::t('app', 'user_account_not_active'));
```

```
}
```

```
}
```

```
}
```

The method allows to define descriptions of fields included in forms. We define an array in which the keys are the names of fields (variables inside the object), and the values of their names, which will be displayed on the page.

```

public function attributeLabels()
{
    return [
        'user_id' => Yii::t('app', 'user_id'),
        'user_email' => Yii::t('app', 'user_email'),
        'user_password' => Yii::t('app', 'user_password'),
        'user_password2' => Yii::t('app', 'user_password2'),
        'user_password3' => Yii::t('app', 'user_password3'),
        'user_key' => Yii::t('app', 'user_key'),
        'user_register' => Yii::t('app', 'user_register'),
        'user_registered_ip' => Yii::t('app', 'user_registered_ip'),
        'user_active' => Yii::t('app', 'user_active'),
        'user_activated' => Yii::t('app', 'user_activated'),
        'user_activated_ip' => Yii::t('app', 'user_activated_ip'),
        'user_root' => Yii::t('app', 'user_root'),
        'user_namesurname' => Yii::t('app', 'user_namesurname'),
        'user_phone' => Yii::t('app', 'user_phone'),
        'user_www' => Yii::t('app', 'user_www'),
        'user_about' => Yii::t('app', 'user_about'),
        'user_captcha' => Yii::t('app', 'user_captcha'),
    ];
}
}

```

Controller for users - UserController

Controller who will manage such options as: registration, logging in, logging out, profile editing, password reminder or changing the password to the account.

File location: ../controllers/SerController.php

We start the file and define the namespace.

```

<?php
namespace app\controllers;

```

Load the frame, the controller head class, the user model and the 404 error model one after the other.

```

use Yii;
use yii\web\Controller;
use app\models\User;
use app\models>Error404;

```

We define a class name identical to the file name, which will be the extension of the main controller.

```

class UserController extends Controller

```

```
{
```

The **actions** method contains defined behaviours for which there is no need to create a separate method. The **captcha** action will be responsible for generating **CAPTCHA** codes on our website. Reference is made to the class and then whether the code generated by the program is to be a test code depending on the constant setting **YII_ENV_TEST**.

```
public function actions()
{
    return [
        'captcha' => [
            'class' => 'yii\captcha\CaptchaAction',
            'fixedVerifyCode' => YII_ENV_TEST ? 'testme' : null,
        ],
    ];
}
```

A method for tracking errors. It creates an object from the **Error404** model, then generates error information using the **AddError404()** method and loads the view.

```
public function actionError()
{
    $model = new Error404();
    $model->AddError404();
    return $this->render('error');
}
```

The method is displayed by default in case of controller call without giving the action in the URL address. It only generates a view from the **index.php** file.

```
public function actionIndex()
{
    return $this->render('index');
}
```

The method responsible for logging in the user to his account available in the system.

```
public function actionLogin()
{
```

Get the properties of the session.

```
$session = Yii::$app->session;
```

We check if the user is not logged in.

```
if($session['yii_user_id'] == "")
```

```
{
```

We create an object from the **User** class for which the checking scenario is set to: **SCENARIO_LOGIN**.

```
$model = new User(['scenario' => User::SCENARIO_LOGIN]);
```

We check if the form has been sent and if all required data have been provided, and then we redirect the user to the home page.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())  
{  
    $this->goHome();  
}  
else  
{
```

If the form was not sent, it was sent incomplete or if the data entered in the fields do not meet the criteria for logging in then we show the user the view of the form by passing to him the object containing the model.

```
return $this->render('login', ['model' => $model]);  
}  
}  
else  
{
```

If the user is already logged in, we will take him to the home page.

```
$this->goHome();  
}  
}
```

The task of the method is to log out the user from the current session and then redirect him to the home page.

```
public function actionLogout()  
{
```

We take the properties of our session.

```
$session = Yii::$app->session;
```

Set the session variables to blank.

```
$session['yii_user_id'] = '';  
$session['yii_user_email'] = '';  
$session['yii_user_root'] = '';
```

We redirect to the homepage.

```
$this->goHome();  
}
```

The method displays a password reminder form and sends an e-mail with a code to reset the current password in the system.

```
public function actionRempass()  
{
```

We transform the User class into an object together with the scenario: **SCENARIO_REMIND**, which corresponds to resetting the password.

```
$model = new User(['scenario' => User::SCENARIO_REMIND]);
```

We check whether the form has been sent and whether the data sent in it meets the requirements.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())  
{
```

We collect IP address.

```
$UserRegisterIp = $_SERVER['REMOTE_ADDR'];
```

Using the mailer object, we compose an e-mail to the user, in which we enter the parameters needed for password recovery in the section **setBodyText()**, and then send it.

```
Yii::$app->mailer->compose()  
->setFrom(array(Yii::$app->params['adminEmail'] => Yii::$app->params['adminName']))  
->setTo($model->user_email)  
->setSubject(Yii::t('app', 'user_remind_pass_title').Yii::$app->params['pageTitle'])  
->setTextBody(  
    Yii::t('app', 'user_remind_pass_body', [  
        'page_title' => Yii::$app->params['pageTitle'],  
        'user_ip' => $UserRegisterIp,  
        'administrator' => Yii::$app->params['adminName'],  
        'page_link' => Yii::$app->params['pageUrl'].'remind-password-set/'.$model->user_id.'/'.$model->password_hash1.'/'.$model->password_hash2  
    ])  
)  
->send();
```

We generate a view with the information that the link was sent to the e-mail address.

```
return $this->render('repass-success', ['model' => $model]);  
}
```



```
else
{
```

When the form was not sent, or the data given in it did not match the record stored in the database, we display a form with fields to remind the password.

```
return $this->render('rempass', ['model' => $model]);
}
}
```

A method that replaces an old forgotten password with a new one generated by the system. It accepts three arguments: the first is **\$UserId** - user ID in the system, the second is **\$UserHash1** - hash generated automatically when the password is reminded, and the third is **\$UserHash2** - the second hash generated automatically.

```
public function actionRempasreset($UserId,$UserHash1,$UserHash2)
{
```

Transform the **User** class into an object.

```
$model = new User();
```

In the database access object we load the variables received from the user and there we check whether they are consistent with the value stored in the database and a new password is set.

```
$model->SetNewPassword($UserId,$UserHash1,$UserHash2);
$returnRemPass = $model->user_password;
```

The variable that will be transferred to the view as a result of changing the password is set to **false**.

```
$viewPassChanged = false;
```

We check if the password returned from the database is equal to **true** value, in our case the password that has been redefined and saved in the database.

```
if($returnRemPass)
{
```

We collect user IP address.

```
$userRegisterIp = $_SERVER['REMOTE_ADDR'];
```

We send an e-mail to the user, writing down the parameters in the **setBodyText()** method.

```
Yii::$app->mailer->compose()
->setFrom(array(Yii::$app->params['adminEmail'] => Yii::$app->params['adminName']))
->setTo($model->user_email)
```

```

->setSubject(Yii::t('app', 'user_new_password').Yii::$app->params['pageTitle'])
->setTextBody(
Yii::t('app', 'user_new_password_body', [
'page_title' => Yii::$app->params['pageTitle'],
'user_ip' => $UserRegisterIp,
'administrator' => Yii::$app->params['adminName'],
'user_password' => $ReturnRemPass,
]
))
->send();

```

The value of the variable informing about the change of the password is set to **true**.

```

$ViewPassChanged = true;
}

```

We load a view with information about changing the password, we send it the value of the variable according to which it will depend on which message is displayed.

```

return $this->render('rempass-set', ['model' => $model, 'ViewPassChanged' => $ViewPassChanged]);
}

```

This method is used to activate the user account. It retrieves two parameters: **\$UserId** - user identifier and **\$UserKey** - key which is located in the table with users.

```

public function actionActivate($UserId,$UserKey)
{

```

We create an object from the User class and load it using the scenario: **SCENARIO_ACTIVATE**, which is responsible for activating the account.

```

$model = new User(['scenario' => User::SCENARIO_ACTIVATE]);

```

The **\$MessageGood** variable will contain information on whether it was possible to activate the user account using the **ActivateUser()** method and the parameters received from the user.

```

$messageGood = $model->activateUser($UserId,$UserKey);

```

We load a view of the activation page of your account together with a model upload and information about a successful or unsuccessful attempt to activate your account.

```

return $this->render('activate', ['model' => $model, 'MessageGood' => $messageGood]);
}

```

The method allowing for the registration of a new user in the system.

```
public function actionRegister()
{
```

User class transformed into a model for which we use the scenario: **SCENARIO_REGISTER**.

```
$model = new User(['scenario' => User::SCENARIO_REGISTER]);
```

We check whether the form has been sent and whether the fields have been validated correctly.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())
{
```

This method is used to download user data in order to send him an e-mail.

```
$UserData = $model->RegisterUser();
```

We collect user IP address.

```
$UserRegisterIp = $_SERVER['REMOTE_ADDR'];
```

Using the mailer facility, we compose and send an e-mail to the user with an activation link.

```
Yii::$app->mailer->compose()
->setFrom(array(Yii::$app->params['adminEmail'] => Yii::$app->params['adminName']))
->setTo($model->user_email)
->setSubject(Yii::t('app', 'user_actvate_account').Yii::$app->params['pageTitle'])
->setTextBody(
Yii::t('app', 'user_actvate_account_body', [
'page_title' => Yii::$app->params['pageTitle'],
'user_ip' => $UserRegisterIp,
'administrator' => Yii::$app->params['adminName'],
'page_link' => Yii::$app->params['pageUrl'].'activate/'.$UserData['user_id'].'/'.$UserData['user_key'],
])
->send();
```

We load the view with the information that the account has been created.

```
return $this->render('register-success', ['model' => $model]);
}
else
{
```

If the form was not sent or was sent with the boxes not completed, the boxes that were filled in again must be completed. Because the system includes automatic character override, restore the amp character in the entered address, and then load the view as you pass the object to it.

```

$model->user_www = str_replace('&', '&', $model->user_www);
return $this->render('register', ['model' => $model]);
}
}

```

The method of changing the data in the user profile.

```

public function actionProfile()
{

```

We set the variable that informs about the correct change of data to false value.

```

$dataChanged = false;

```

Load the properties of the session and check if the user has logged in.

```

$session = Yii::$app->session;
if($session['yii_user_id'] != "")
{

```

We retrieve the property of the page request, and from it select the data sent by the form using the **POST** method.

```

$request = Yii::$app->request;
$post = $request->post();

```

We create an object from the **User** class where we load a scenario: **SCENARIO_UPDATEPROFILE**, which is responsible for updating the profile.

```

$model = new User(['scenario' => User::SCENARIO_UPDATEPROFILE]);

```

Select the data currently available in the user profile.

```

$dataToProfile = $model->selectProfile();

```

We load the data from the query into the variables available in the model.

```

$model->user_namesurname = $dataToProfile['user_namesurname'];
$model->user_phone = $dataToProfile['user_phone'];
$model->user_www = $dataToProfile['user_www'];
$model->user_about = $dataToProfile['user_about'];

```

We change the amp character, which by default is converted to ounces through the framework.

```
$model->user_www = str_replace('&', '&', $model->user_www);
```

We check whether the form has been sent and whether the data contained in it has been validated correctly.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())  
{
```

We call up a method to update the profile from the model.

```
$model->UpdateProfile();
```

The variable responsible for updating the data is set to **true**.

```
$dataChanged = true;
```

The variables in the model are loaded with data from the form.

```
$model->user_namesurname = $post['User']['user_namesurname'];  
$model->user_phone = $post['User']['user_phone'];  
$model->user_www = $post['User']['user_www'];  
$model->user_about = $post['User']['user_about'];  
}
```

We load the form to edit the profile.

```
return $this->render('profile', ['model' => $model, 'dataChanged' => $dataChanged]);  
}  
else  
{
```

If the user is not logged in, we will take him to the home page.

```
$this->goHome();  
}  
}
```

A method that allows you to change the password for a user account.

```
public function actionChangepassword()  
{
```

The variable informing about the change of the password is set to **false**.

```
$passwordChanged = false;
```

Load the properties of the session and check if the user has been logged in.

```
$session = Yii::$app->session;
if($session['yii_user_id'] != "")
{
```

We create an object from the **User** class where we set the scenario: **SCENARIO_CHANGEPASSWORD**, which checks the fields when changing the password.

```
$model = new User(['scenario' => User::SCENARIO_CHANGEPASSWORD]);
```

We check whether the form has been sent and all fields have been filled in correctly.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())
{
```

We update the user password and set the internal variables to empty.

```
$model->UpdatePassword();
$model->user_password = '';
$model->user_password2 = '';
$model->user_password3 = '';
```

For the variable informing about the change of password we set the value to true.

```
$passwordChanged = true;
}
```

Load the page view by entering the model as a variable and the information whether the change was successful or not.

```
return $this->render('changepassword', ['model' => $model, 'passwordChanged' => $passwordChanged]);
}
else
{
```

If you are not logged in, you will be directed to the home page.

```
$this->goHome();
}
}
```

The method displays information to the user that there are no access rights, mainly it is used in the admin panel. It reads the corresponding view.

```
public function actionRight()
{
    return $this->render('right');
}
}
```

View - account activation

View used when activating an account through a user made by clicking a link in an email sent to him.

File location: ../views/user/activate.php

We start the PHP file, read the helper to generate HTML and the plugin that helps to generate the form.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

Enter the title of the page and the current location.

```
$this->title = Yii::t('app', 'user_activate_account');
$this->params['breadcrumbs'][] = $this->title;
```

If all data in the link agree, the displayed information about the activation of the account.

```
if($MessageGood)
{
    echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'user_activate_yes').'</div>';
}
```

When the link data does not match the data stored in the database, we print the information about the failure.

```
else
{
    echo '<div class="alert alert-danger" role="alert">'.Yii::t('app', 'user_activate_no').'</div>';
}
?>
```

View - password change form

A form designed to change the password for your account by entering the old password, then double the new one.

File location: ../views/user/changepassword.php

We start the PHP file, load the helper for HTML and the plugin to generate the form.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

Enter the title and location of the current page.

```
$this->title = Yii::t('app', 'user_change_password');
$this->params['breadcrumbs'][] = $this->title;
```

When the password is correctly changed, the corresponding message is printed.

```
if($passwordChanged)
{
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'user_password_changed').'</div>';
}
```

We start the form.

```
$form = ActiveForm::begin([
'id' =>'login-form',
'enableAjaxValidation' => false,
'enableClientValidation' => false,
]);
```

We print the fields for the old password, the new password and the new one.

```
echo $form->field($model, 'user_password')->label(Yii::t('app', 'user_old_password'))->passwordInput();
echo $form->field($model, 'user_password2')->passwordInput();
echo $form->field($model, 'user_password3')->passwordInput();
```

We print the button for sending the form.

```
echo '<div class="form-group">';
echo Html::submitButton(Yii::t('app', 'user_password_change_button'), ['class' =>'btn btn-primary']);
echo '</div>';
```

We stop generating the form.

```
ActiveForm::end();
?>
```


View - error on page

Error message 404 on the page displayed to the user.

File location: ../views/user/error.php

Start the PHP file and load the helper to generate HTML tags.

```
<?php
use yii\helpers\Html;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'user_error404');
$this->params['breadcrumbs'][] = $this->title;
?>
```

We print out an error message.

```
<div class="site-error">
<h1><?= Html::encode($this->title) ?></h1>
<p><?php echo Yii::t('app', 'user_error404_com'); ?></p>
</div>
```

View - user login form

A page containing a form designed to log in to your account in the system.

File location: ../views/user/login.php

We start the PHP file, load the helper for generating HTML tags and the plugin for generating forms.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'login_header');
$this->params['breadcrumbs'][] = $this->title;
```

We start the form.

```
$form = ActiveForm::begin([
```

```
'id' => 'login-form',
'enableAjaxValidation' => false,
'enableClientValidation' => false,
]);
```

We print the fields for user e-mail address and password.

```
echo $form->field($model, 'user_email');
echo $form->field($model, 'user_password')->passwordInput();
```

Create a confirmation button for the form.

```
echo '<div class="form-group">';
echo Html::submitButton(Yii::t('app', 'login_page_submit'), ['class' => 'btn btn-primary']);
echo '</div>';
```

We are finishing the generation of the form.

```
ActiveForm::end();
?>
```

View - user profile

Template allowing the user to edit his data in the profile.

File location: ../views/user/profile.php

We start the PHP file, load the helper for generating HTML tags and the plugin for generating the form.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the title and the path to the page.

```
$this->title = Yii::t('app', 'profile_header');
$this->params['breadcrumbs'][] = $this->title;
```

If you update the data in the profile correctly, you will need to display a corresponding message.

```
if($dataChanged)
{
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'profile_data_updated').'</div>';
}
```

We start the form.

```
$form = ActiveForm::begin([
    'id' => 'login-form',
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]);
```

We print the fields for your name, phone number, web address and the personal information you want to include in your profile.

```
echo $form->field($model, 'user_namesurname');
echo $form->field($model, 'user_phone');
echo $form->field($model, 'user_www');
echo $form->field($model, 'user_about')->textarea(['rows' => '6']);
```

Print the form confirmation button.

```
echo '<div class="form-group">';
echo Html::submitButton(Yii::t('app', 'profile_update_submit'), ['class' => 'btn btn-primary']);
echo '</div>';
```

We are ending the generation of the form.

```
ActiveForm::end();
?>
```

View - registration of a new account

View containing a form by which users can register on the account page.

File location: ../views/user/register.php

We start the PHP file, load the helper for HTML generation and the form plugin.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the title of a page together with its location.

```
$this->title = Yii::t('app', 'register_header');
$this->params['breadcrumbs'][] = $this->title;
```

We start the form.

```
$form = ActiveForm::begin([
    'id' => 'login-form',
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]);
```

We print the fields for: e-mail address, password and its repetition, name and surname, telephone number, website and a short note about the user.

```
echo $form->field($model, 'user_email');
echo $form->field($model, 'user_password')->passwordInput();
echo $form->field($model, 'user_password2')->passwordInput();
echo $form->field($model, 'user_namesurname');
echo $form->field($model, 'user_phone');
echo $form->field($model, 'user_www');
echo $form->field($model, 'user_about')->textarea(['rows' => '6']);
```

We print the button for sending the form.

```
echo '<div class="form-group">';
echo Html::submitButton(Yii::t('app', 'register_page_submit'), ['class' => 'btn btn-primary']);
echo '</div>';
```

We are finishing the process of creating the form.

```
ActiveForm::end();
?>
```

View - information on correct registration

When a user registers their own account on a website, they must be redirected to the website to confirm their registration by clicking on the link that was sent to their e-mail address.

File location: ../views/user/register-success.php

Start the PHP file, load the helper for generating HTML tags and the form plugin.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'register_success_header');
$this->params['breadcrumbs'][] = $this->title;
```

We show user information about the correct registration of account.

```
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'register_success_comm').'</div>';
?>
```

View - password reminder

When the user forgets the password, he will be able to use the option of its reminding.

File location: ../views/user/rempass.php

We start the PHP file, load the helper for generating HTML files, the form plugin and the library for generating **Captcha**.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\captcha\Captcha;
```

We define the title of a page together with its location.

```
$this->title = Yii::t('app', 'rem_password_header');
$this->params['breadcrumbs'][] = $this->title;
```

We start the form.

```
$form = ActiveForm::begin([
    'id' => 'login-form',
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]);
```

We print a field for entering the e-mail address and the **Captcha** code together with an image containing it.

```
echo $form->field($model, 'user_email');
echo $form->field($model, 'user_captcha')->widget(Captcha::className(), [
    'template' => '<div class="row"><div class="col-lg-2">{image}</div><div class="col-lg-10">{input}</div></div>',
    'captchaAction' => 'user/captcha'
]);
```

Print the form confirmation button.

```

echo '<div class="form-group">';
echo Html::submitButton(Yii::t('app', 'rem_password_button_submit'), ['class' =>'btn btn-primary']);
echo '</div>';

```

We finish the form.

```

ActiveForm::end();
?>

```

View - information about changing the password

A page that tells you that your password has been correctly or incorrectly reminded by e-mail.

File location: ../views/user/rempass-set.php

We start the PHP file, load the helper for creating HTML elements and the form plug-in.

```

<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;

```

We define the title of a page and its location.

```

$this->title = Yii::t('app', 'rem_password_header');
$this->params['breadcrumbs'][] = $this->title;

```

When all data is correct, the password will be changed.

```

if($ViewPassChanged)
{
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'rem_password_yes').'</div>';
}

```

In case the data is incorrect, we define the information about the error.

```

else
{
echo '<div class="alert alert-danger" role="alert">'.Yii::t('app', 'rem_password_no').'</div>';
}
?>

```

View - password reminder, e-mail check request

When we send you a password reminder form where all the fields will be filled in with correct data, you will be informed accordingly.

File location: ../views/user/rempass-success.php

We start the PHP file, load the helper for HTML generation and the plugin for the forms.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the name of the page and its location.

```
$this->title = Yii::t('app', 'rem_password_header');
$this->params['breadcrumbs'][] = $this->title;
```

We inform you that a link to change your password has been sent to your e-mail address and in order to continue you should click on this link.

```
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'rem_password_check_email').'</div>';
?>
```

View - no access rights

For example, a user who enters the administration panel but does not have access rights will be notified that he does not have access rights. In this case, the user will be informed about the incorrect permissions.

File location: ../views/user/right.php

We start the PHP file and load the helper for generating HTML tags.

```
<?php
use yii\helpers\Html;
```

We define the name of the page and its location.

```
$this->title = Yii::t('app', 'no_right_header');
$this->params['breadcrumbs'][] = $this->title;
?>
```

We print information about an error in accessing content not intended for the user.

```
<div class="site-error">
```

```
<h1><?php echo Yii::t('app', 'no_right_header'); ?></h1>  
<p>  
<?php echo Yii::t('app', 'no_right_comm'); ?>  
</p>  
</div>
```


Chapter 4. Page

Page display controller - PageController

Controller responsible for displaying an inventory of all pages, a single page, and a home page.

File location: ../controllers/PageController.php

We start the file, set the namespace, load the Yii framework, the class with the master controller, the page model and the plugin for splitting into pages.

```
<?php
namespace app\controllers;
use Yii;
use yii\web\Controller;
use app\models\Page;
use yii\data\Pagination;
```

We define a class identical to the filename.

```
class PageController extends Controller
{
```

Method showing all the elements with a division into pages.

```
public function actionIndex()
{
```

We create an object from the **Page()** class.

```
$model = new Page();
```

We count all the entries available in the table.

```
$CountAll = $model->CountAll();
```

Using the **Pagination()** plugin, in which we enter the number of available records and the amount we want to have on each page, we divide the entries.

```
$pagination = new Pagination(['totalCount' => $CountAll, 'pageSize' => 10]);
```

We select the content of the pages with division by adding two parameters. The first one responsible for the record from which it is to start, and the second one is the number of records selected.

```
$SelectPages = $model->SelectPages($pagination->offset,$pagination->limit);
```

We generate a page by loading a view and transferring to it data such as model, selected text pages and the variable defining the division.

```
return $this->render('page', ['model' => $model, 'SelectPages' => $SelectPages, 'pagination' =>
$pagination]);
}
```

Method showing one selected page on the basis of parameters. The first parameter is **\$PageUrl** - the page's own address and **\$PageId** - the page identifier.

```
public function actionShowone($PageUrl,$PageId)
{
```

We create an object of the **Page** class.

```
$model = new Page();
```

Select a page from the model using the parameter specifying the identification number of the text page.

```
$IsThatPage = $model->SelectOnePage($PageId);
```

Check if the value **true** is stored in the variable, if so, load the view of the page to which the model is to be loaded.

```
if($IsThatPage)
{
return $this->render('showone', ['model' => $model]);
}
```

If such a page did not exist, we would like to show you information about this event.

```
else
{
return $this->render('page-noexists', ['model' => $model]);
}
}
```

A method that selects a page set as home to be displayed on the home page.

```
public function actionHome()
{
```

Read the `Page()` class as an object.

```
$model = new Page();
```

Check whether the home page has been set up.

```
$IsThatPage = $model->SelectHome();
```

If the home page has been set, read in the view to which you are transferring the model.

```
if($IsThatPage)
{
return $this->render('showhome', ['model' => $model]);
}
```

If the page is not loaded, the appropriate view with information about this fact will be displayed.

```
else
{
return $this->render('page-noexists', ['model' => $model]);
}
}
}
?>
```

Text page model - Page

The model is designed to handle the content of pages added to our system for the user and will therefore use only **SQL** queries instead of the **ActiveRecord** interface.

File Location: ../models/Page.php

We define the namespace set on the models and load the framework classes and the database access model.

```
<?php
namespace app\models;
use Yii;
use yii\base\Model;
```

We define a **Page** class name identical to the file name and mark it as an extension of the **Model** class.

```
class Page extends Model
{
```

We declare the intra-grade variables to be public. The variables will be identical to the field names in the table.

```
public $page_id;
public $page_title;
public $page_text;
public $page_url;
```

The method returns the name of the table containing the pages. Since we will be using the prefix, the proper name should be placed in double brackets and preceded by a percentage sign.

```
public static function tableName()
{
    return '{{%page}}';
}
```

A method that allows us to count all the pages available in the table.

```
public function CountAll()
{
    $QueryData = Yii::$app->db->createCommand('SELECT count(page_id) AS HowManyRecords FROM {{%page}}')
->queryScalar();
}
```

Set the value of the variable to be returned to zero. Then we check if the variable containing the result of the query is empty and in this case we rewrite its content.

```
$ToReturn = 0;
if($QueryData != false)
{
    $ToReturn = $QueryData;
}
```

We return the result with the number of records.

```
return $ToReturn;
}
```

The method that selects a range of records defined by variables from the table with pages: **\$Start** - it defines the number of the record from which you should start, **\$Limit** - it defines the limit, i.e. the number of records that you want to receive from the query.

```
public function SelectPages($Start,$Limit)
{
}
```

We create the query together with the parameters and then using the **queryAll()** method we select all the records that meet the given criteria.

```

$queryData = Yii::$app->db->createCommand('SELECT * FROM {{%page}} ORDER BY page_id DESC LIMIT
:start,:limit')
->bindParam(':start', $Start)
->bindParam(':limit', $Limit)
->queryAll();

```

We will return the result of our inquiry.

```

return $QueryData;
}

```

The method returns a record from a table containing a text page whose identifier is given in the first parameter of **\$PageId**.

```

public function SelectOnePage($PageId)
{

```

We create a query by loading a parameter into it and selecting only one **queryOne()** record.

```

$queryData = Yii::$app->db->createCommand('SELECT * FROM {{%page}} WHERE page_id = :page_id')
->bindParam(':page_id', $PageId)
->queryOne();

```

Check if the query returned an empty result and in this case enter the **false** value into the variable to be returned.

```

if($QueryData == false)
{
    $ToReturn = false;
}
else
{

```

If the query returns a result, set the variable that will be returned as a result of the method to true and overwrite the values from the query to the class variables.

```

$ToReturn = true;
$this->page_title = $QueryData['page_title'];
$this->page_text = $QueryData['page_text'];
$this->page_url = $QueryData['page_url'];
}

```

We return the result of the query.

```

return $ToReturn;
}

```

The method of choosing the page set for the home page of our service. The method does not accept any parameter.

```
public function SelectHome()
{
```

Create a query that checks if there is an y value in the **page_main** field and selects only one record by using the **queryOne()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%page}} WHERE page_main = "y"')
->queryOne();
```

If a variable from the query does not return any value, also the variable which will be returned from the method of such a value should not return.

```
if($QueryData == false)
{
    $ToReturn = false;
}
else
{
```

If the variable contains a value, then the value true should be inserted into the variable returned from the method, which means that the result exists. Then we transcribe the result of the query to the variables contained within the class.

```
$ToReturn = true;
$this->page_title = $QueryData['page_title'];
$this->page_text = $QueryData['page_text'];
$this->page_url = $QueryData['page_url'];
}
```

We return the value of the method.

```
return $ToReturn;
}
```

A method that returns an array in which the name of the array is the name of the array in the table and the value of its description.

```
public function attributeLabels()
{
    return [
        'page_id' => Yii::t('app', 'page_id'),
        'page_title' => Yii::t('app', 'page_title'),
        'page_text' => Yii::t('app', 'page_text'),
        'page_url' => Yii::t('app', 'page_url'),
    ];
}
}
```

?>

View - index of text pages

The view is designed to display a list of text pages available on the website.

File location: ../views/page/page.php

We start the PHP file, load the helper for generating HTML tags, the plugin for creating the form and the division into pages.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\widgets\LinkPager;
```

We define the title of the page and the location.

```
$this->title = Yii::t('app', 'a_title_pages');
$this->params['breadcrumbs'][] = $this->title;
```

We display the division into pages.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
```

We print the header, page title, abbreviated content and a link to read the whole.

```
echo '<h1>'.Yii::t('app', 'a_title_pages').</h1>';
for($Pages=0;$Pages<count($SelectPages);$Pages++)
{
    echo '<h2>'.$SelectPages[$Pages]['page_title'].'</h2>';
    echo '<p>'.substr(strip_tags($SelectPages[$Pages]['page_text']),0,350).'... '.
    Html::a('<nobr>'.Yii::t('app', 'read_more').</nobr>',
    ['/page/'.$SelectPages[$Pages]['page_url'].'/',$SelectPages[$Pages]['page_id']]).
    '<p>';
}
```

We print the division into pages.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
?>
```

View - no page specified

Displays information if you are navigating to a page that does not exist.

File location: ../views/page/page-noexists.php

We start the PHP file, load the helper to generate the HTML file and the plugin to generate the form.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We add the title of the page together with its location.

```
$this->title = Yii::t('app', 'a_title_page_dont_exists');
$this->params['breadcrumbs'][] = $this->title;
```

We print the information about the lack of the page specified in the URL address.

```
echo '<h1>'.Yii::t('app', 'a_title_page_dont_exists').'</h1>';
echo '<div class="alert alert-danger" role="alert">'.Yii::t('app', 'page_dont_exists').'</div>';
?>
```

View - home page

Prints the page set as Home on the home page.

File location: ../views/page/showhome.php

We start the PHP file, load the helper for creating HTML tags, the plugin for generating forms and the libraries for creating the navigation bar.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\bootstrap\Nav;
use yii\bootstrap\NavBar;
```

We place the title, print the page header and its content.

```
$this->title = $model->page_title;
echo '<h1>'.$model->page_title.'</h1>';
echo $model->page_text;

?>
```


View - showing the page

The file generates a view showing the full content of the selected system page.

File location: ../views/page/showone.php

We start the PHP file, read the helper generating the HTML and the plugin to generate the forms.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the title and location of the page.

```
$this->title = $model->page_title;
$this->params['breadcrumbs'][] = $this->title;
```

We print the title of the page and its content.

```
echo '<h1>'.$model->page_title.'</h1>';
echo $model->page_text;
?>
```

Chapter 5. Article

Model of articles - Article

A model whose task will be to present articles on the website in a version for the user. It will be based on the SQL query due to the speed of code execution, which is very important for the user-associated interface.

File location: ../models/Article.php

We start the PHP file, define the namespace at app\models, load the framework and the parent class for the models.

```
<?php
namespace app\models;
use Yii;
use yii\base\Model;
```

We create a class, identical to the file name, which will be an extension of the **Model** class.

```
class Article extends Model
{
```

We define internal variables that will be available to the public, for example from the controller. Ideally, the variable names should match the names of the fields from the table in the database, thanks to which it will be easier for us to program the methods.

```
public $article_id;
public $article_title;
public $article_text;
public $article_author;
public $article_date;
public $article_url;
```

The method will return the name of the table. It is located in brackets and is preceded by a percentage sign in order to add a prefix declared in the configuration file with data for connection to the database to the proper name.

```
public static function tableName()
{
    return '{{%article}}';
}
```

Use this method to return a value containing the number of all items in the table. In this case, we use the **queryScalar()** method, which returns the value of the query directly.

```
public function CountAll()
```

```
{
$queryData = Yii::$app->db->createCommand('SELECT count(article_id) AS HowManyRecords FROM {%article}')
->queryScalar();
```

Set the number of records to zero.

```
$ToReturn = 0;
```

Then we check whether the result is different from the false value and if so we enter it into the variable that will be returned from the method.

```
if($QueryData != false)
{
$ToReturn = $QueryData;
}
```

We return the value.

```
return $ToReturn;
}
```

A method that selects the range of articles from the table. It has two values. The first one is **\$Start** specifying from which record the results should be selected, and the second one is **\$Limit** specifying the number of records to be selected.

```
public function SelectArticles($Start,$Limit)
{
```

We make a query in which we use the **queryAll()** method and load the value of the variables using the **bindParam()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {%article} ORDER BY article_id DESC LIMIT
:start,:limit')
->bindParam(':start', $Start)
->bindParam(':limit', $Limit)
->queryAll();
```

We return the result that contains the article records.

```
return $QueryData;
}
```

The method is used to select one article whose identifier is given in the first parameter of **\$ArticleId**.

```
public function SelectOneArticle($ArticleId)
{
```

Create a database query in which you add a parameter using **bindParam()** and then use the method to return only one **queryOne()** record.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {%article%} WHERE article_id = :article_id')
->bindParam(':article_id', $ArticleId)
->queryOne();
```

We check if the result of the query is **false**, i.e. it is empty. In this case, enter **false** into the variable with the return of the value.

```
if($QueryData == false)
{
    $ToReturn = false;
}
else
{
```

When the result is not empty, enter true into the variable that will be returned from the method. Then we enter the data from the database into the variables of the object.

```
$ToReturn = true;

$this->article_title = $QueryData['article_title'];
$this->article_text = $QueryData['article_text'];
$this->article_author = $QueryData['article_author'];
$this->article_date = $QueryData['article_date'];
$this->article_url = $QueryData['article_url'];
}
```

The value of the variable is returned.

```
return $ToReturn;
}
```

We still need to create the **attributeLabels()** method, which contains the names of the fields of forms. Thanks to them it will be possible to generate a form using the **ActiveForm()** tool built into the framework. This method returns an array in which the keys are field names and the values of their descriptions.

```
public function attributeLabels()
{
    return [
        'article_id' => Yii::t('app', 'art_id'),
        'article_title' => Yii::t('app', 'art_title'),
        'article_text' => Yii::t('app', 'art_content'),
        'article_author' => Yii::t('app', 'art_author'),
        'article_date' => Yii::t('app', 'art_data'),
        'article_url' => Yii::t('app', 'art_url'),
    ];
}
```

```
}  
}  
?>
```

Article controller - ArticleController

The controller allows to display the articles available in the table in the database in the form of their divided into the pages of the list as well as one selected article.

File location: ../controllers/ArticleController.php

We start the file, define the names zone, load the framework, the main class of the controller, the model of articles and the plugin responsible for dividing the content into pages.

```
<?php  
namespace app\controllers;  
use Yii;  
use yii\web\Controller;  
use app\models\Article;  
use yii\data\Pagination;
```

We define a class name identical to the file name, which will be an extension of the main class of the controller.

```
class ArticleController extends Controller  
{
```

The method displays a list of articles by page.

```
public function actionIndex()  
{
```

Load the **Articles()** model as an object.

```
$model = new Article();
```

We count all the articles available in the table.

```
$CountAll = $model->CountAll();
```

We create an object of the **Pagination()** class in which we specify the number of records and the amount we want to see on one page.

```
$pagination = new Pagination(['totalCount' => $CountAll, 'pageSize' => 10]);
```

Select the entries as two parameters: the number of the record from which you want to start dialing and the number of records you want to select.

```
$SelectArticles = $model->SelectArticles($pagination->offset,$pagination->limit);
```

We load the view to which we send variables and objects in the parameter.

```
return $this->render('article', ['model' => $model, 'SelectArticles' => $SelectArticles, 'pagination' =>
$pagination]);
}
```

The method responsible for displaying one article adopts two parameters. The first is the URL of the article, and the second is its unique identifier.

```
public function actionShowone($ArticleUrl,$ArticleId)
{
```

Transform the **Article()** class of the model into an object.

```
$model = new Article();
```

We check if a given article exists by selecting it from the table where we enter its identifier as an argument.

```
$IsThatPage = $model->SelectOneArticle($ArticleId);
```

If an item exists, load the view until it is displayed.

```
if($IsThatPage)
{
return $this->render('showone', ['model' => $model]);
}
```

In case it is not available, we will load the information about the lack of the article.

```
else
{
return $this->render('article-noexists', ['model' => $model]);
}
}
}
?>
```

View: show articles

It is designed to display the list of articles with its origins, divided into pages with links to read the full content.

File location: ../views/article/article.php

We start PHP file and read HTML helper and plug-ins generating form and links of content division into pages.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\widgets\LinkPager;
```

We define the title of the page and the path displayed to the user.

```
$this->title = Yii::t('app', 'p_articles_header');
$this->params['breadcrumbs'][] = $this->title;
```

Displays the page division.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
echo '<h1>Articles</h1>';
```

We use the loop to display the articles with links to their full content.

```
for($Pages=0;$Pages<count($SelectArticles);$Pages++)
{
    echo '<h2>'.$SelectArticles[$Pages]['article_title'].'</h2>';
    echo Yii::t('app', 'p_article_author').'.$SelectArticles[$Pages]['article_author'].'. '.Yii::t('app',
    'p_article_published').'.$SelectArticles[$Pages]['article_date'];
    echo '<p>'.substr(strip_tags($SelectArticles[$Pages]['article_text']),0,350).'... '.
    Html::a('<nobr>'.Yii::t('app', 'p_article_read_more').</nobr>',
    ['/article/'.$SelectArticles[$Pages]['article_url'].'/.'.$SelectArticles[$Pages]['article_id']]).
    '</p>';
}
```

We print the division into pages.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
?>
```

View: no article

In case we delete an article, a user with an old URL will enter its content and an error message created by us will appear to his eyes.

File location: ../views/article/article-nonexists.php

We start the PHP file, load the HTML generation and the plugin generating the form.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We place a description in the title variable and define the access path.

```
$this->title = Yii::t('app', 'p_article_no_article');
$this->params['breadcrumbs'][] = $this->title;
```

We inform the user that the given article is not available on the website.

```
echo '<h1>'.Yii::t('app', 'p_article_no_article').'</h1>';
echo '<div class="alert alert-danger" role="alert">'.Yii::t('app', 'p_article_no_article_comm').'</div>';

?>
```

View: one article

Displays one article in full glory with all meta tags.

File location: ../views/article/showone.php

We start the PHP file, load the helper to generate HTML and the plugin generating the forms.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We create the title of the page and provide the location of the current page in the structure of the website.

```
$this->title = $model->article_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'p_articles_header'), 'url' => ['/article']];
$this->params['breadcrumbs'][] = $this->title;
```

We print the header, author, date of publication and content of the article.

```
echo '<h1>'.$model->article_title.'</h1>';
echo Yii::t('app', 'p_article_author').'.$model->article_author.', '.Yii::t('app',
'p_article_published').'.$model->article_date.'<br />';
```



```
echo $model->article_text;
```

```
?>
```

Chapter 6. Blog

Blog model - Blog

A model designed to support the blog module of our web application. It will be responsible for selecting entries, categories, hash tags and limiting them to an appropriate number of records. The file uses a model interface with simple **SQL** queries to generate content for the user more quickly.

File location: ../models/Blog.php

We start PHP file, declare namespace, framework class and Model class.

```
<?php
namespace app\models;
use Yii;
use yii\base\Model;
```

We create a **Blog** class which is consistent with the filename and is an extension of the **Model** class.

```
class Blog extends Model
{
```

We declare the names of table fields stored in the database, where we will also have access to their contents from the controller and view by considering access to these variables as public.

```
public $blog_id;
public $blog_title;
public $blog_text;
public $blog_date;
public $blog_url;
public $blog_category;
public $blog_hashtag;
```

The method that returns the table name is located inside two braces and is preceded by a percentage character. This will cause the prefix declared by us in the connection configuration file to be added to the database by the proper name of the table.

```
public static function tableName()
{
    return '{{%blog}}';
}
```

The method is designed to count all entries in the blog and to return this value by using the **queryScalar()** method.

```

public function CountAll()
{
    $QueryData = Yii::$app->db->createCommand('SELECT count(blog_id) AS HowManyRecords FROM {%blog}');
    ->queryScalar();
    $ToReturn = 0;

```

Check if the value returned from the table is not empty and in this case we rewrite the content of the variable with the results to the variable to be returned.

```

if($QueryData != false)
{
    $ToReturn = $QueryData;
}

```

We return the result of our query.

```

return $ToReturn;
}

```

A method that returns the number of alerts in a given category. It accepts one argument, which is a category identifier.

```

public function CountAllCategory($CategoryId)
{

```

We create a query to the database in which we create a counting of the number of entries in the blog and transferring their result directly to the variable that is responsible for the query by using the **queryScalar()** method.

```

$QueryData = Yii::$app->db->createCommand('SELECT count(blog_id) AS HowManyRecords FROM {%blog} WHERE
blog_category = :blog_category')
->bindParam(':blog_category', $CategoryId)
->queryScalar();

```

Set the variable that will be returned from the method to zero, then check if the result of the query is different than **false**, i.e. empty, and if there is one, rewrite the value from the variable with saving to the variable that will be returned.

```

$ToReturn = 0;
if($QueryData != false)
{
    $ToReturn = $QueryData;
}

```

We return the value from the method.

```

return $ToReturn;
}

```

A method that allows you to return all the entries that have the listed text provided in the hash column in the first parameter.

```
public function CountAllHash($Hash)
{
```

The text provided in the parameter is provided with preceding and ending percentage characters, which are treated as any other characters by the database (**MySQL**).

```
$ReadyHash = '%'.$Hash.'%';
```

Create a query with the hash text given as a parameter and the result returned directly to the variable using the **queryScalar()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT count(blog_id) AS HowManyRecords FROM {%blog} WHERE
blog_hashtag LIKE :blog_hashtag')
->bindParam(':blog_hashtag', $ReadyHash)
->queryScalar();
```

The content of the variable to be returned is set to zero, and then we check whether any value has been returned as a result of the query, if so, it will be set as a value of the variable.

```
$ToReturn = 0;
if($QueryData != false)
{
    $ToReturn = $QueryData;
}
```

We return the value of the method.

```
return $ToReturn;
}
```

A method that selects blog entries with a limit on their number. This limit is defined by two values. The first parameter of the method is **\$Start** - which defines the record from which you want to start, and the second is **\$Limit** - the number of records to be selected.

```
public function SelectBlog($Start,$Limit)
{
```

We create a query with parameters and select all records that will be returned to us using **queryAll()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {%blog} ORDER BY blog_id DESC LIMIT
:start,:limit')
->bindParam(':start', $Start)
->bindParam(':limit', $Limit)
->queryAll();
```

We return the result of the method.

```
return $QueryData;
}
```

A method that allows you to select only one category of blog entries with a limit on the number of records. It contains three parameters. The first **\$Start** - determines from which record to start dialling, the next **\$Limit** parameter - contains the number of records to start dialling, and the third, the last **\$CategoryId** contains the category number by which the entries should be dialled.

```
public function SelectBlogCategory($Start,$Limit, $CategoryId)
{
```

We create a query that selects records that fit only one category and imposes a selection limit.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%blog}} WHERE blog_category = :blog_category
ORDER BY blog_id DESC LIMIT :start,:limit')
->bindParam(':blog_category', $CategoryId)
->bindParam(':start', $Start)
->bindParam(':limit', $Limit)
->queryAll();
```

We return the result of the query from the method.

```
return $QueryData;
}
```

Just like the previous one, the method is designed to select from the database records containing the appropriate hash in the field. It accepts three arguments, the first **\$Start** - the record from which you want to start selecting from the table, the **\$Limit** - the number of records you want to select, and the **\$Hash** - the word by which you want to search for phrases.

```
public function SelectBlogHash($Start,$Limit, $Hash)
{
```

The word given in the parameter should be surrounded by percentage characters, which will cause that in case of database (**MySQL**) they will overwrite any characters.

```
$ReadyHash = '%'.$Hash.'%';
```

We create a query where to select appropriate entries from our blog table and return all of the data records using the **queryAll()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%blog}} WHERE blog_hashtag LIKE :blog_hashtag
ORDER BY blog_id DESC LIMIT :start,:limit')
->bindParam(':blog_hashtag', $ReadyHash)
->bindParam(':start', $Start)
->bindParam(':limit', $Limit)
->queryAll();
```

We return the result of the method.

```
return $QueryData;
}
```

A method that allows you to select all categories of the blog section. We define the query with the return of all data using **queryAll()** and return the query result as the result of the method.

```
public function SelectCategory()
{
    $QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%blog_category}}')
->queryAll();

    return $QueryData;
}
```

The method that selects a single entry in the blog, the identifier of which is provided in the first parameter of **\$BlogId**.

```
public function SelectOneBlog($BlogId)
{
```

We create a query to which we insert the content of the **\$BlogId** variable and select only one record using the **queryOne()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%blog}} WHERE blog_id = :blog_id')
->bindParam(':blog_id', $BlogId)
->queryOne();
```

We check whether any data has been returned in the query. If not, enter **false** into the variable to be returned from the method, otherwise **true**.

```
if($QueryData == false)
{
    $ToReturn = false;
}
else
{
    $ToReturn = true;
```

You still need to rewrite the fields from the found record to the variables inside the object that correspond to their field names.

```
$this->blog_title = $QueryData['blog_title'];
$this->blog_text = $QueryData['blog_text'];
$this->blog_date = $QueryData['blog_date'];
$this->blog_url = $QueryData['blog_url'];
```

```
$this->blog_category = $QueryData['blog_category'];
$this->blog_hashtag = $QueryData['blog_hashtag'];
}
```

The value of the variable is returned.

```
return $ToReturn;
}
```

A method that returns descriptions of fields, for example in forms. It returns an array in which the key is the name of the field and the value of its description.

```
public function attributeLabels()
{
    return [
        'blog_id' => Yii::t('app', 'blog_id'),
        'blog_title' => Yii::t('app', 'blog_title'),
        'blog_text' => Yii::t('app', 'blog_content'),
        'blog_date' => Yii::t('app', 'blog_date'),
        'blog_url' => Yii::t('app', 'blog_url'),
        'blog_category' => Yii::t('app', 'blog_category'),
        'blog_hashtag' => Yii::t('app', 'blog_tag'),
    ];
}
}
```

Blog controller - BlogController

Controller designed to operate a blog: show a list of entries, show one selected entry, show entries available for a given category and the hash sign.

File location: ../controllers/BlogController.php

We start the file, define the namespace, load the framework, the main class of the controller, the blog model and the plugin that allows you to share your entries on the pages.

```
<?php
namespace app\controllers;
use Yii;
use yii\web\Controller;
use app\models\Blog;
use yii\data\Pagination;
```

We define the name of the class, which is identical to the file name and will be the extension of the main class.

```
class BlogController extends Controller
{
```

The method displays blog entries by page.

```
public function actionIndex()
{
```

Read the **Blog()** model class as an object.

```
$model = new Blog();
```

A method is called which counts the number of all entries in the table.

```
$CountAll = $model->CountAll();
```

We create an object from the **Pagination()** class by entering the total number of records and the number of entries per one page.

```
$pagination = new Pagination(['totalCount' => $CountAll, 'pageSize' => 10]);
```

Select the appropriate entries from the table in the blog, specifying the place from which you want to return and the limit of records.

```
$SelectBlog = $model->SelectBlog($pagination->offset,$pagination->limit);
```

We select all categories of the blog.

```
$SelectCategoryData = $model->SelectCategory();
```

We convert the categories into an array in which the key will be the identification number of the category, and the value of the array with the title of the category and its URL.

```
for($c=0;$c<count($SelectCategoryData);$c++)
{
    $SelectCategory[$SelectCategoryData[$c]['category_id']] = array('title' =>
    $SelectCategoryData[$c]['category_title'], 'url' => $SelectCategoryData[$c]['category_url']);
}
```

Load the view of the blog by entering in the parameter the variables and objects needed to generate the file.

```
return $this->render('blog', ['model' => $model, 'SelectBlog' => $SelectBlog, 'SelectCategory' =>
$SelectCategory, 'pagination' => $pagination]);
}
```


The method returns entries in the blog for which the appropriate category has been set. It takes two parameters: the first **\$CategoryId** is the category address, and the second **\$CategoryId** contains its identifier.

```
public function actionCategory($CategoryId,$CategoryId)
{
```

Transform the **Blog()** model class into an object.

```
$model = new Blog();
```

Using the method in the model object we calculate the number of entries that belong to the given category, the identifier of which we entered in the first parameter.

```
$CountAll = $model->CountAllCategory($CategoryId);
```

We create an object from the **Pagination()** class, to which parameters we specify the number of records in the table and the limit of records we want to display on one page.

```
$pagination = new Pagination(['totalCount' => $CountAll, 'pageSize' => 10]);
```

We select all blog entries that fall within the range specified in the parameters defining the record from which you want to start the selection, the number of records and the number of the category from which you want to select.

```
$SelectBlog = $model->SelectBlogCategory($pagination->offset,$pagination->limit,$CategoryId);
```

Using the model method, we select all categories.

```
$SelectCategoryData = $model->SelectCategory();
```

Categories are converted into an array in which the key will be the identification number, and the value of the next array containing the name of the category and its URL.

```
for($c=0;$c<count($SelectCategoryData);$c++)
{
    $SelectCategory[$SelectCategoryData[$c]['category_id']] = array('title' =>
    $SelectCategoryData[$c]['category_title'], 'url' => $SelectCategoryData[$c]['category_url']);
}
```

Select the current category.

```
$CategoryIs = $SelectCategory[$CategoryId];
```

We load the view of the blog category and pass it on to it all the necessary variables and objects.

```
return $this->render('category', ['model' => $model, 'CategoryIs' => $CategoryIs, 'SelectBlog' =>
$SelectBlog, 'SelectCategory' => $SelectCategory, 'pagination' => $pagination]);
```

```
}
```

The method will display all entries in the blog table that contain the corresponding hash defined in the first parameter.

```
public function actionHash($Hash)
{
```

Convert the **Blog()** model class into an object.

```
$model = new Blog();
```

We decode the hash from the URL to the format needed to query the database.

```
$Hash = urldecode($Hash);
```

Count the number of records in the entry table using hash arguments.

```
$CountAll = $model->CountAllHash($Hash);
```

Pagination() class is transformed into an object to which we transfer parameters responsible for the number of all records and determining the number of records displayed on one page.

```
$pagination = new Pagination(['totalCount' => $CountAll, 'pageSize' => 10]);
```

We choose from the blog entries corresponding to the parameters specifying the number of the record from which you want to start, the number of records to be selected and the hash that the record should contain.

```
$SelectBlog = $model->SelectBlogHash($pagination->offset,$pagination->limit,$Hash);
```

From the model we choose all categories of blog entries.

```
$SelectCategoryData = $model->SelectCategory();
```

Using a loop, we convert the data from the table into an array with the category identifier key and the value of the next array into an array with the category identifier key. It contains the name of the category and its URL.

```
for($c=0;$c<count($SelectCategoryData);$c++)
{
    $SelectCategory[$SelectCategoryData[$c]['category_id']] = array('title' =>
    $SelectCategoryData[$c]['category_title'], 'url' => $SelectCategoryData[$c]['category_url']);
}
```

Load the view of entries with the password specified in the first parameter and send the necessary variables and objects to it.

```

return $this->render('hash', ['model' => $model, 'Hash' => $Hash, 'SelectBlog' => $SelectBlog,
'SelectCategory' => $SelectCategory, 'pagination' => $pagination]);
}

```

The method shows one entry from the blog. It takes **\$BlogUrl** parameters, i.e. the URL of the current entry, and **\$BlogId** - the identifier of the entry in the blog.

```

public function actionShowone($BlogUrl,$BlogId)
{

```

Read the **Blog()** model as an object.

```

$model = new Blog();

```

Select all categories of entries.

```

$selectCategoryData = $model->selectCategory();

```

Using the loop we create an array containing the identification number of the category in the key, and as the next value we create an array containing: the name of the category and the URL address.

```

for($c=0;$c<count($selectCategoryData);$c++)
{
$selectCategory[$selectCategoryData[$c]['category_id']] = array('title' =>
$selectCategoryData[$c]['category_title'], 'url' => $selectCategoryData[$c]['category_url']);
}

```

Choose one entry based on the **\$BlogId** parameter you submitted.

```

$isThatPage = $model->selectOneBlog($BlogId);

```

Check if the entry exists, if so, load the view.

```

if($isThatPage)
{
return $this->render('showone', ['model' => $model, 'selectCategory' => $selectCategory]);
}

```

If the entry is not present, an error message is displayed.

```

else
{
return $this->render('blog-noexists', ['model' => $model]);
}
}
}

```

?>

View: blog

A file designed to display the list of blog entries with a division into pages.

File location: ../views/blog/blog.php

We start the PHP file, load the helper to generate HTML and the plug-ins that help to generate the form and the links of division into pages.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\widgets\LinkPager;
```

Enter the title of the page and its location.

```
$this->title = Yii::t('app', 'p_blog_header');
$this->params['breadcrumbs'][] = $this->title;
```

We print the links concerning the division into pages.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
```

We print the header.

```
echo '<h1>'.Yii::t('app', 'p_blog_header').'</h1>';
```

We print all individual blog entries via a loop.

```
for($Pages=0;$Pages<count($SelectBlog);$Pages++)
{
```

We retrieve the tags, break them down into single words, and then link the links created to go to individual subpages.

```
$IsHashTag = explode(' ', $SelectBlog[$Pages]['blog_hashtag']);
$RememberHashTags = null;
for($h=0;$h<count($IsHashTag);$h++)
{
    $IsHashTag[$h] = trim($IsHashTag[$h]);
    $RememberHashTags[] = Html::a($IsHashTag[$h], ['/hash/'.urlencode($IsHashTag[$h])]);
}
```

```
$HashTagShowOnPage = implode(' ', $RememberHashTags);
```

We print the header, the date of publication, the category, the content, the tags and the link to go to the entire entry.

```
echo '<h2>'.$SelectBlog[$Pages]['blog_title'].'</h2>';
echo Yii::t('app', 'p_blog_published').$SelectBlog[$Pages]['blog_date'].'',
'.Yii::t('app', 'p_blog_category').'.';
Html::a($SelectCategory[$SelectBlog[$Pages]['blog_category']]['title'],
['/category/'.$SelectCategory[$SelectBlog[$Pages]['blog_category']]['url'].'/'.$SelectBlog[$Pages]['blog_c
ategory'])).',
'.Yii::t('app', 'p_blog_tags').'.'.$HashTagShowOnPage.'
';
echo '<p>'.substr(strip_tags($SelectBlog[$Pages]['blog_text']),0,350).'... '
Html::a('<nobr>'.Yii::t('app', 'p_blog_read_more').</nobr>',
['/blog/'.$SelectBlog[$Pages]['blog_url'].'/'.$SelectBlog[$Pages]['blog_id'])).
'</p>';
}
```

We print the division into pages.

```
echo LinkPager::widget([
'pagination' => $pagination,
]);
?>
```

View: no blog entry

It is designed to display information about the absence of an entry in the blog, the identifier of which we provided in the URL address.

File location: ../views/blog/blog-noexists.php

We start the PHP file, load the helper for generating HTML tags and the plugin for creating forms.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

Enter the title of the page and the current location.

```
$this->title = Yii::t('app', 'p_blog_no_entry');
$this->params['breadcrumbs'][] = $this->title;
```

We print out the information about the lack of an entry in the blog.

```
echo '<h1>'.Yii::t('app', 'p_blog_no_entry').</h1>';
```

```
echo '<div class="alert alert-danger" role="alert">'.Yii::t('app', 'p_blog_no_entry_comm').'</div>';
?>
```

View: category of blog entry

It contains a list of blog entries belonging to the given category.

File location: ../views/blog/category.php

We define PHP file, load helper to generate HTML, plugin to forms and division into pages.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\widgets\LinkPager;
```

We define the title of the page and the location.

```
$this->title = Yii::t('app', 'p_blog_category_title').$CategoryIs['title'];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'p_blog_header'), 'url' => ['/blog']];
$this->params['breadcrumbs'][] = Yii::t('app', 'p_blog_category_navigation').'
['.$CategoryIs['title'].']';
```

We print the division into pages.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
```

We print the header and select individual data concerning the entry in the loop together with the appropriate formatting of the content.

```
echo '<h1>Blog - '.$CategoryIs['title'].'</h1>';
for($Pages=0;$Pages<count($SelectBlog);$Pages++)
{
```

Download tags, divide the value of the field where the comma appears, then create links to move to the specified tag and connect the array with links to the whole.

```
$IsHashTag = explode(',', $SelectBlog[$Pages]['blog_hashtag']);
$RememberHashTags = null;
for($h=0;$h<count($IsHashTag);$h++)
{
    $IsHashTag[$h] = trim($IsHashTag[$h]);
    $RememberHashTags[] = Html::a($IsHashTag[$h], ['/hash/'.urlencode($IsHashTag[$h])]);
}
```

```
$HashTagShowOnPage = implode(' ', $RememberHashTags);
```

We display the title of the entry, the date of publication, the category, the link to read the entire entry and the abbreviated version.

```
echo '<h2>'.$SelectBlog[$Pages]['blog_title'].'</h2>';
echo Yii::t('app', 'p_blog_published').' '.$SelectBlog[$Pages]['blog_date'].' , '.
Yii::t('app', 'p_blog_category').' .
Html::a($SelectCategory[$SelectBlog[$Pages]['blog_category']]['title'],
['/category/'.$SelectCategory[$SelectBlog[$Pages]['blog_category']]['url'].'/'.$SelectBlog[$Pages]['blog_c
ategory'])).',
'.Yii::t('app', 'p_blog_tags').' '.$HashTagShowOnPage.'
';
echo '<p>'.substr(strip_tags($SelectBlog[$Pages]['blog_text']),0,350).'... '.
Html::a('<nobr>'.Yii::t('app', 'p_blog_read_more').'</nobr>',
['/blog/'.$SelectBlog[$Pages]['blog_url'].'/'.$SelectBlog[$Pages]['blog_id']]).
'</p>';
}
```

We print the division into pages.

```
echo LinkPager::widget([
'pagination' => $pagination,
]);
?>
```

View: show the entries matching the tag

Display blog entries that contain the correct word in the tag box.

File location: ../views/blog/hash.php

We start the PHP file, load the HTML helper and plug-ins to generate the form and divide it into pages.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\widgets\LinkPager;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'p_blog_hashtag_header').' ['. $Hash.' ]';
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'p_blog_header'), 'url' => ['/blog']];
$this->params['breadcrumbs'][] = Yii::t('app', 'p_blog_hashtag_tag').' ['. $Hash.' ]';
```

We print the division into pages.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
```

We print the header and the loop in which individual entries will be processed.

```
echo '<h1>'.Yii::t('app', 'p_blog_hashtag_header').' ['. $Hash.']</h1>';
for($Pages=0;$Pages<count($SelectBlog);$Pages++)
{
```

We retrieve the tags, break them down into individual elements of the array, transform them into links, and then combine them into one string of characters.

```
$IsHashTag = explode(' ', $SelectBlog[$Pages]['blog_hashtag']);
$RememberHashTags = null;
for($h=0;$h<count($IsHashTag);$h++)
{
    $IsHashTag[$h] = trim($IsHashTag[$h]);
    $RememberHashTags[] = Html::a($IsHashTag[$h], ['/hash/'.urlencode($IsHashTag[$h])]);
}
$HashTagShowOnPage = implode(' ', $RememberHashTags);
```

We print the header of the entry, the date of its publication, the category, tags, the abbreviated content and the link to read the entire entry.

```
echo '<h2>'.$SelectBlog[$Pages]['blog_title'].'</h2>';
echo Yii::t('app', 'p_blog_published').' '.$SelectBlog[$Pages]['blog_date'].' , '.
Yii::t('app', 'p_blog_category').' '.
Html::a($SelectCategory[$SelectBlog[$Pages]['blog_category']]['title'],
['/category/'.$SelectCategory[$SelectBlog[$Pages]['blog_category']]['url'].'/'.$SelectBlog[$Pages]['blog_c
ategory'])).' ',
'.Yii::t('app', 'p_blog_tags').' '.$HashTagShowOnPage.'
';
echo '<p>'.substr(strip_tags($SelectBlog[$Pages]['blog_text']),0,350).'... '.
Html::a('<nobr>'.Yii::t('app', 'p_blog_read_more').'</nobr>',
['/blog/'.$SelectBlog[$Pages]['blog_url'].'/'.$SelectBlog[$Pages]['blog_id']]).
'</p>';
}
```

We print the division into pages.

```
echo LinkPager::widget([
    'pagination' => $pagination,
]);
?>
```


View - show one entry in the blog

Show one entry in a blog and display its full content.

File location: ../views/blog/showone.php

We start the PHP file, load the helper to generate HTML and the plugin to generate forms.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the title and location of the content.

```
$this->title = $model->blog_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_blog_header'), 'url' => ['/blog']];
$this->params['breadcrumbs'][] = $this->title;
```

We break it down to a comma, then add a properly constructed link to each loop and merge it into one string.

```
$IsHashTag = explode(' ', $model->blog_hashtag);
$RememberHashTags = null;
for($h=0;$h<count($IsHashTag);$h++)
{
    $IsHashTag[$h] = trim($IsHashTag[$h]);
    $RememberHashTags[] = Html::a($IsHashTag[$h], ['/hash/'.urlencode($IsHashTag[$h])]);
}
$HashTagShowOnPage = implode(' ', $RememberHashTags);
```

We print the header, date of publication, category, tags and the full entry in the blog.

```
echo '<h2>'.$model->blog_title.'</h2>';
echo Yii::t('app', 'p_blog_published').' '.$model->blog_date.', '.
Yii::t('app', 'p_blog_category').' '.
Html::a($SelectCategory[$model->blog_category]['title'], ['/category/'.$SelectCategory[$model->blog_category]['url'].'/'.$model->blog_category]).',
'.Yii::t('app', 'p_blog_tags').' '.$HashTagShowOnPage.'<br /><br />
';
echo $model->blog_text;
?>
```

Chapter 7. Contact

Model contact form - Contact

The method responsible for selecting e-mail addresses for the contact form and for sending them by using the data in the validation model. We will use **SQL** queries in its design to speed it up.

File location: ../models/Contact.php

We start the file with the definition of the namespace, loading the framework and the model class.

```
<?php
namespace app\models;
use Yii;
use yii\base\Model;
```

We define a class name identical to the file name, which will be an extension of the **Model** class.

```
class Contact extends Model
{
```

We define the variables inside the class, giving them the **public** property, so that they will be visible outside the class.

```
public $contact_id;
public $contact_email;
public $contact_name;
public $contact_from;
public $contact_title;
public $contact_body;
public $contact_sendme;
public $contact_captcha;
```

The method returns the name of the table. Because the used prefix has to be surrounded with double braces and the percentage sign has to be given by the proper name.

```
public static function tableName()
{
    return '{{%contact}}';
}
```

We define a scenario that will be performed when sending the form.

```
const SCENARIO_CONTACT = 'contact';
```

A method that returns the fields to be sent when a check scenario is met.

```
public function scenarios()  
{  
    return [  

```

The scenario for executing the contact will accept fields such as: **contact_id**, **contact_email**, **contact_name**, **contact_from**, **contact_title**, **contact_body**, **contact_sendme**, **contact_captcha**.

```
self::SCENARIO_CONTACT => ['contact_id', 'contact_email', 'contact_name', 'contact_from', 'contact_title',  
    'contact_body', 'contact_sendme', 'contact_captcha'],  
];  
}
```

Method for verifying that the relevant boxes of the form have been completed with data that meet the requirements.

```
public function rules()  
{  
    return [  

```

Form fields are required during the contact scenario.

```
[['contact_from', 'contact_title', 'contact_body', 'contact_captcha'], 'required', 'on' =>  
self::SCENARIO_CONTACT],
```

The **contact_from** field must contain the correct e-mail address when checking the contact scenario.

```
['contact_from', 'email', 'on' => self::SCENARIO_CONTACT],
```

The **contact_captcha** field must contain correctly rewritten **Captcha** code which is executed by the user's controller in the captcha method.

```
['contact_captcha', 'captcha', 'captchaAction' => 'user/captcha', 'on' => self::SCENARIO_CONTACT],  
];  
}
```

A method for selecting data from a table with stored e-mail addresses. Data returned from queries are returned as a result of the method.

```
public function SelectContacts()  
{  
    $QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%contact}} ORDER BY contact_name ASC')  
->queryAll();  
    return $QueryData;  
}
```

A method that retrieves one address from the contact table to which you want to send a letter. The parameter is the contact identifier from the table.

```
public function SelectOneEmail($ContactId)
{
```

We create a query together with loading a parameter into it and select only one record using the **queryOne()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {%contact%} WHERE contact_id = :contact_id')
->bindParam(':contact_id', $ContactId)
->queryOne();
```

We transcribe the data obtained from the query to the table to be returned.

```
$GetEmail['email'] = $QueryData['contact_email'];
$GetEmail['name'] = $QueryData['contact_name'];
```

We return the result of the method.

```
return $GetEmail;
}
```

A method that returns an array in which the key is the name of a field and the value of its description.

```
public function attributeLabels()
{
return [
'contact_id' => Yii::t('app', 'contact_id'),
'contact_email' => Yii::t('app', 'contact_email'),
'contact_name' => Yii::t('app', 'contact_name'),
'contact_from' => Yii::t('app', 'contact_from'),
'contact_title' => Yii::t('app', 'contact_title'),
'contact_body' => Yii::t('app', 'contact_body'),
'contact_sendme' => Yii::t('app', 'contact_sendme'),
'contact_captcha' => Yii::t('app', 'contact_captcha'),
];
}
}
?>
```

Contact controller - ContactController

Controller for handling contact information. It displays the form and fills it with the data from the contact table. After sending the form, it also processes the task and sends an e-mail.

File location: ../controllers/ContactController.php

We start the file, define the namespace, load the framework, the superior class of the controller and the contact model.

```
<?php
namespace app\controllers;
use Yii;
use yii\web\Controller;
use app\models\Contact;
```

We define a class name identical to the file name, which will be the extension of the main class of the controller.

```
class ContactController extends Controller
{
```

This method is used to call actions for which there is no need to create special methods. Let's declare here the captcha action, which will allow to display underneath the form the inscription **Captcha**, which makes sending BOT's forms difficult.

```
public function actions()
{
    return [
        'captcha' => [
            'class' => 'yii\captcha\CaptchaAction',
            'fixedVerifyCode' => YII_ENV_TEST ? 'testme' : null,
        ],
    ];
}
```

The method displays a contact form and, depending on the validation, sends it or asks you to complete it.

```
public function actionIndex()
{
```

The **Contact** class is transformed into a model with the **SCENARIO_CONTACT** scenario, which is responsible for checking the data when sending the form.

```
$model = new Contact(['scenario' => Contact::SCENARIO_CONTACT]);
```

Select the list of email addresses to contact.

```
$ItemToContactListGet = $model->SelectContacts();
$ItemToContactList = false;
```

We transform the list selected from the database into an appropriate array, whose key is the contact's identifier and the value of the e-mail address and the name of the person or department to which we intend to write.

```
for($QRezult=0;$QRezult<count($ItemToContactListGet);$QRezult++)
{
    $ItemToContactList[$ItemToContactListGet[$QRezult]['contact_id']] =
    $ItemToContactListGet[$QRezult]['contact_email'].' - '.$ItemToContactListGet[$QRezult]['contact_name'];
}
```

The value of the variable responsible for the information about sending the form is set to **false**.

```
$FormWasSend = false;
```

We check if the form has been sent and if the fields have been validated correctly.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())
{
```

Download one e-mail address from the table based on the internal model variable to which the address identifier has been assigned.

```
$GetEmail = $model->SelectOneEmail($model->contact_email);
```

Save the user's IP address to the variable.

```
$UserRegisterIp = $_SERVER['REMOTE_ADDR'];
```

We check if the variable form with the request to send the content to the sender has been enabled and depending on the choice we send an e-mail.

```
if($model->contact_sendme == true)
{
    Yii::$app->mailer->compose()
->setFrom($model->contact_from)
->setTo(array($GetEmail['email'] => $GetEmail['name']))
->setCc($model->contact_from)
->setSubject($model->contact_title)
->setTextBody($model->contact_body.'
IP: '.$UserRegisterIp)
->send();
}
else
{
    Yii::$app->mailer->compose()
->setFrom($model->contact_from)
->setTo(array($GetEmail['email'] => $GetEmail['name']))
->setSubject($model->contact_title)
```

```

->setTextBody($model->contact_body.'
IP: '.$UserRegisterIp)
->send();
}

```

We clean all the variables inside the class so that the form fields are cleared.

```

$model->contact_email = '';
$model->contact_from = '';
$model->contact_title = '';
$model->contact_body = '';
$model->contact_captcha = '';
$model->contact_sendme = '';

```

Set true to a variable that contains information that the form has been sent.

```

$FormWasSend = true;
}

```

We load a view of the form with the model being sent to it, information whether it has been sent, and lists of e-mail recipients.

```

return $this->render('form', ['model' => $model, 'FormWasSend' => $FormWasSend, 'ItemToContactList' =>
$itemToContactList]);
}
}
?>

```

View - contact form

A file containing a form that allows you to contact us at the e-mail addresses provided by us in the admin panel.

File location: ../views/contact/form.php

We start the PHP file, land the helper to generate HTML and load the plugin to generate the forms and the **Captcha** field.

```

<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
use yii\captcha\Captcha;

```

Enter the title of the page and the location.

```

$this->title = Yii::t('app', 'p_contact');
$this->params['breadcrumbs'][] = $this->title;

```

Check if the contact list exists.

```
if($ItemToContactList != false)
{
```

We check whether the form was sent, if it was sent, we print the message.

```
if($FormWasSend == 'true')
{
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'p_contact_form_was_send').'</div>';
}
```

We start the form.

```
$form = ActiveForm::begin([
'id' =>'login-form',
'enableAjaxValidation' => false,
'enableClientValidation' => false,
]);
```

We print fields such as the drop-down list with e-mail addresses, the field for defining the sender, title or content of the letter. At the end of the form we print the field and the image of **Captcha**.

```
echo $form->field($model, 'contact_email')->dropDownList($ItemToContactList);
echo $form->field($model, 'contact_from');
echo $form->field($model, 'contact_title');
echo $form->field($model, 'contact_body')->textarea(['rows' =>'6']);
echo $form->field($model, 'contact_captcha')->widget(Captcha::className(), [
'template' =>'<div class="row"><div class="col-lg-2">{image}</div><div class="col-lg-10">{input}</div></div>',
'captchaAction' =>'contact/captcha'
]);
```

We print a field that allows you to send an e-mail copy to the address entered in the form.

```
echo $form->field($model, 'contact_sendme')->checkbox();
```

Button for accepting the form.

```
echo '<div class="form-group">';
echo Html::submitButton(Yii::t('app', 'p_contact_send_button'), ['class' =>'btn btn-primary']);
echo '</div>';
```

We finish the form.

```
ActiveForm::end();
```



```
}
```

If there are no e-mail addresses to which you can send messages, we will display a corresponding message.

```
else
{
echo '<div class="alert alert-danger" role="alert">.Yii::t('app', 'p_contact_no_email').</div>';
}
?>
```

Chapter 8. Download

Download model - Download

The model responsible for selecting files in the download area. All queries are processed with clean SQL queries to reduce response time.

File location: ../models/Download.php

We start the file by defining the namespace and loading the framework and model.

```
<?php
namespace app\models;
use Yii;
use yii\base\Model;
```

We define the **Download** class the same as the file name, which will be the extension of the **Model** class.

```
class Download extends Model
{
```

We declare ground names that are identical in their name with the field names in the table. We set them to public so that they are visible outside the class.

```
public $download_id;
public $download_title;
public $download_text;
public $download_file;
public $download_version;
public $download_filesize;
public $download_license;
```

A method that contains the name of a table that, in order for a prefix to be added to it, must be surrounded by double braces and have a percentage mark at the beginning.

```
public static function tableName()
{
    return '{{%download}}';
}
```

The method is designed to select all files added to the download section.

```
public function SelectDownloads()
```

```
{
```

Create a query selecting all records from the table and sorting them by the **download_id** field in descending order.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {%download} ORDER BY download_id DESC')
->queryAll();
```

We return the result of the method.

```
return $QueryData;
}
```

A method to select one file from the download section. The parameter will be the identifier of the file stored in the table.

```
public function SelectOneDownload($ProgramId)
{
```

We create the query together with loading the parameter and selecting only one record using **queryOne()** method.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {%download} WHERE download_id = :download_id')
->bindParam(':download_id', $ProgramId)
->queryOne();
```

We return the result of the query.

```
return $QueryData;
}
```

A method that allows you to place information about a downloaded file in a database. It takes the parameters **\$browserName** - name and version of the browser, **\$fileDownload** - file number to download, **\$dateAndTime** - date and time to download, and **\$ipClient** - IP address of the user who downloaded the file.

```
public function InsertDownloadAction($browserName,$fileDownload,$dateAndTime,$ipClient)
{
```

We create a query to place the data in the table using the **execute()** method.

```
$QueryData = Yii::$app->db->createCommand('INSERT INTO {%download_stats} (stat_browser, stat_file,
stat_date, stat_ip) VALUES (:stat_browser, :stat_file, :stat_date, :stat_ip)')
->bindParam(':stat_browser', $browserName)
->bindParam(':stat_file', $fileDownload)
->bindParam(':stat_date', $dateAndTime)
->bindParam(':stat_ip', $ipClient)
->execute();
}
```

A method that returns an array of field names placed in the key and its description in a value.

```
public function attributeLabels()
{
    return [
        'download_id' => Yii::t('app', 'download_id'),
        'download_title' => Yii::t('app', 'download_title'),
        'download_text' => Yii::t('app', 'download_text'),
        'download_file' => Yii::t('app', 'download_file'),
        'download_version' => Yii::t('app', 'download_version'),
        'download_filesize' => Yii::t('app', 'download_filesize'),
        'download_license' => Yii::t('app', 'download_license'),
    ];
}
}
```

Download controller - DownloadController

Controller designed to present the department for downloading files and in case of downloading is to calculate downloading and enter them in the table with statistics.

File location: ../controllers/DownloadController.php

We start the file, define the namespace, load the framework, the main class of controllers, the file download model and the add-on in the form of the definition of the server response to the request.

```
<?php
namespace app\controllers;
use Yii;
use yii\web\Controller;
use app\models\Download;
use yii\web\Response;
```

We define a class name the same as the file name, which will be an extension of the **Controller** class.

```
class DownloadController extends Controller
{
```

The method allows you to select the information about the files to be downloaded from the database.

```
public function actionIndex()
{
```

We create an object from the **Download()** model class.

```
$model = new Download();
```

The variable that returns the query value will be set to false if there is no record to show.

```
$modelData = false;
```

We execute a query show all the files available in the table.

```
$modelData = $model->SelectDownloads();
```

Load the view with the transfer of the parameter containing the information about the files to it.

```
return $this->render('download', ['modelData' => $modelData]);  
}
```

The method is used while downloading the program. It takes two parameters. The first **\$ProgramId** contains the identification number of the program, the second **\$ProgramName** contains the name of the file with the program to download.

```
public function actionGetprogram($ProgramId,$ProgramName)  
{
```

From **Download()** class we create the object in which the model will be included.

```
$model = new Download();
```

Select one program from the table by passing its identifier in the first parameter.

```
$fileData = $model->SelectOneDownload($ProgramId);
```

We retrieve the headings of the page request.

```
$headers = Yii::$app->request->headers;
```

We successively define the name of your browser, the name of the file, the date and time of the download, and the IP address of the person who performs the download.

```
$browserName = $headers->get('User-Agent');  
$fileDownload = $fileData['download_file'];  
$dateAndTime = date('Y-m-d H:i:s');  
$ipClient = $_SERVER['REMOTE_ADDR'];
```

The data is entered into the table using the model method.

```
$model->InsertDownloadAction($browserName,$fileDownload,$dateAndTime,$ipClient);
```

We redirect the user to the file that was selected for download. Thanks to this, after saving the statistics, the user will receive the appropriate file.

```
$this->redirect(Yii::$app->params['pageUrl'].'getfiles/'.$fileData['download_file']);  
}  
}  
?>
```

View - downloading files

It will be responsible for displaying the download section of our website.

File location: ../views/download/download.php

Start the PHP file and load the helper to create HTML tags.

```
<?php  
use yii\helpers\Html;
```

We add a title and the location of the page.

```
$this->title = Yii::t('app', 'p_download_header');  
$this->params['breadcrumbs'][] = $this->title;
```

Thanks to the loop, we display all the elements of the table transferred to the view, which contains data such as: the name of the file, the file identifier, the program version, the file size, the license and the content.

```
for($Files=0;$Files<count($modelData);$Files++)  
{  
echo '<h2>'.$modelData[$Files]['download_title'].'</h2>';  
echo Html::a(Yii::t('app', 'p_download_button').' '.$modelData[$Files]['download_title'],  
['/download/'.$modelData[$Files]['download_id'].'/'.urlencode($modelData[$Files]['download_file'])],  
['class'=>'btn btn-primary grid-button'])."<br /><br />";  
echo Yii::t('app', 'p_download_version').'<strong>'.$modelData[$Files]['download_version'].'</strong>,'  
'.'Yii::t('app', 'p_download_filesize').'<strong>'.$modelData[$Files]['download_filesize'].'</strong>,'  
'.'Yii::t('app', 'p_download_license').'<strong>'.$modelData[$Files]['download_license'].'</strong><br />';  
echo $modelData[$Files]['download_text'];  
}  
?>
```

Chapter 9. Error404

Error handling model - Error404

A model designed to track errors on our website. For example, if we delete an entry from our blog and a user from another website moves to her address, then the **Error404** page will be displayed, informing him that the entry has not been found. At the same time, the information about such a situation will be saved in the database where it will be possible to view it later in the admin panel.

File location: ../models/Error404.php

We start the file with the definition of the namespace, load the Yii framework.

```
<?php
namespace app\models;
use Yii;
```

Now we declare **Error404** class according to file name, which in this case will be **ActiveRecord** extension.

```
class Error404 extends \yii\db\ActiveRecord
{
```

A method that returns the name of a table that is surrounded by a double-click bracket and is preceded by an interest character to automatically assign a prefix to it from the configuration file.

```
public static function tableName()
{
    return '{{error404}}';
}
```

A method whose purpose is to return the result of validation of individual fields.

```
public function rules()
{
    return [
```

We define the fields required to add an error message.

```
[['error_page_from', 'error_page', 'error_date'], 'required'],
```

We define the fields that contain the string of characters.

```
[['error_page_from', 'error_page'], 'string'],
```

We define **error_date** as a safe attribute.

```
[['error_date'], 'safe'],  
];  
}
```

A method that stores information about an error in a table. It has two parameters: **\$FromPage** - the page from which you came and **\$OnPage** - the page which does not exist in the system.

```
public function AddError404($FromPage,$OnPage)  
{
```

The variable with the page from which we came will contain the **HTTP_REFERER** server variable. The variable containing the current page will be built using two variables: **HTTP_HOST** and **REQUEST_URI**. Then to the date of creating the error enter the current day together with the time.

```
$FromPage = isset($_SERVER['HTTP_REFERER']) ? $_SERVER['HTTP_REFERER'] : 'null';  
$OnPage = (isset($_SERVER['HTTPS']) ? "https" : "http")."://".$_SERVER['HTTP_HOST'].$_SERVER['REQUEST_URI'];  
$DateError = date("y-m-d H:i:s");
```

Create a query in which to place the information in a table, and then **execute()** the command.

```
$QueryData = Yii::$app->db->createCommand('INSERT INTO {%error404%} (error_page_from, error_page,  
error_date)  
values  
(:error_page_from, :error_page, :error_date)  
)  
'->bindParam(':error_page_from', $FromPage)  
'->bindParam(':error_page', $OnPage)  
'->bindParam(':error_date', $DateError)  
'->execute();  
}
```

A method that returns an array in which the key is the name of a field and the value of its description.

```
public function attributeLabels()  
{  
return [  
'error_id' => Yii::t('app', 'error_id'),  
'error_page_from' => Yii::t('app', 'error_page_from'),  
'error_page' => Yii::t('app', 'error_page'),  
'error_date' => Yii::t('app', 'error_date'),  
];  
}  
}
```


Chapter 10. User administration

User administration model - UserAdmin

A model needed for user administration, allowing to edit, delete and change some of the adjustable properties.

File location: ../models/Useradmin.php

Start the file, set the namespace, and load the Yii framework.

```
<?php
namespace app\models;
use Yii;
```

We define the name of the class, identical to the file name, and that it is an extension of the **ActiveRecord** interface.

```
class Useradmin extends \yii\db\ActiveRecord
{
```

A method that returns the name of a table that is surrounded by a double brace together with a percentage character that precedes the name of the table. This automatically adds a prefix to the table.

```
public static function tableName()
{
    return '{{%user}}';
}
```

We are defining two scenarios. The first is to create a new user, and the second is to edit the data already in the system.

```
const SCENARIO_NEW = 'new';
const SCENARIO_EDIT = 'edit';
```

The method that defines the fields to be returned during the execution of particular scenarios.

```
public function scenarios()
{
    return [
```

Scenario for adding a new user.

```
self::SCENARIO_NEW => ['user_email', 'user_namesurname', 'user_phone', 'user_www', 'user_about',
    'user_root'],
```

Scenario for user editing.

```
self::SCENARIO_EDIT => ['user_namesurname', 'user_phone', 'user_www', 'user_about', 'user_root'],  
];  
}
```

A method that returns the result of checking the contents of fields when a scenario is executed.

```
public function rules()  
{
```

Adding a user, during the adding scenario, requires filling in two fields: **user_email** and **user_namesurname**, so that the **user_email** field contains the correct e-mail address. Additionally, the **user_email** field will check if the given address already exists in the table using the **validateUserIsEmail** method.

```
return [  
[['user_email', 'user_namesurname'], 'required', 'on' => self::SCENARIO_NEW],  
['user_email', 'email', 'on' => self::SCENARIO_NEW],  
['user_email', 'validateUserIsEmail', 'skipOnError' => false, 'on' => self::SCENARIO_NEW],
```

In the edit scenario, only the **user_namesurname** field is required.

```
['user_namesurname', 'required', 'on' => self::SCENARIO_EDIT],  
];  
}
```

The method invoked to check if a user exists. It has three parameters. The first \$attribute contains the field data that will be checked, the second \$params contains the parameters for the given field, and the third \$validator contains information about the type of check used.

```
public function validateUserIsEmail($attribute, $params, $validator)  
{
```

We download the content of your e-mail address from a variable inside the class and create and execute a query by selecting only one record.

```
$UserEmail = $this->user_email;  
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {{%user}} WHERE user_email = :user_email')  
->bindParam(':user_email', $UserEmail)->queryOne();
```

If the variable containing the result of the query is not empty, then we write the error by using the **addError()** method, where in the first parameter we enter the name of the field, and in the second parameter we write the message.

```
if($QueryData != false)  
{  
$this->addError($attribute, Yii::t('app', 'a2_account_exist_email'));
```

```
}
}
```

A method that returns an array in which the keys are field names and the values are their description.

```
public function attributeLabels()
{
    return [
        'user_id' => Yii::t('app', 'user_id'),
        'user_email' => Yii::t('app', 'user_email'),
        'user_password' => Yii::t('app', 'user_password'),
        'user_password2' => Yii::t('app', 'user_password2'),
        'user_password3' => Yii::t('app', 'user_password3'),
        'user_key' => Yii::t('app', 'user_key'),
        'user_register' => Yii::t('app', 'user_register'),
        'user_registered_ip' => Yii::t('app', 'user_registered_ip'),
        'user_active' => Yii::t('app', 'user_active'),
        'user_activated' => Yii::t('app', 'user_activated'),
        'user_activated_ip' => Yii::t('app', 'user_activated_ip'),
        'user_root' => Yii::t('app', 'user_root'),
        'user_namesurname' => Yii::t('app', 'user_namesurname'),
        'user_phone' => Yii::t('app', 'user_phone'),
        'user_www' => Yii::t('app', 'user_www'),
        'user_about' => Yii::t('app', 'user_about'),
        'user_captcha' => Yii::t('app', 'user_captcha'),
    ];
}
}
```

User management controller - UserAdminController

Controller is used to manage system users including: adding, editing data and deleting entire accounts. Due to its use in the admin panel, we can use it to build **ActiveRecord** interface.

File location: ../controllers/SeradminController.php

We start the PHP file, define the names zone, load the Yii framework, data model, class to generate content, main controller class, error handling and filtering the input data.

```
<?php
namespace app\controllers;
use Yii;
use app\models\Useradmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

We define a class name identical to the file name, which will be the extension of the main class of the controller.

```
class UserAdminController extends Controller
{
```

The variable responsible for selecting the template for the page is set to admin to enable the management template.

```
public $layout = 'admin';
```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
    if($session['yii_user_root'] != "y")
    {
        return $this->redirect(['/right']);
    }
}
```

If the user is not logged in, we will take him to the login form page.

```
else
{
    return $this->redirect(['/login']);
}
```

At the end we call the method with the same name from the master controller.

```
return parent::beforeAction($action);
}
```

The method allows us to define the behavior of the application.

```
public function behaviors()
```

```
{
return [
'verbs' => [
```

We add that variables sent in the delete method can be sent only by the **POST** method.

```
'class' => VerbFilter::className(),
'actions' => [
'delete' => ['POST'],
],
],
];
}
```

The method responsible for displaying data on users in the system, as well as links enabling to perform actions on them.

```
public function actionIndex()
{
```

We save in administrative logs information about access to user sections.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_users'));
```

Using the **ActiveDataProvider()** class, you can download data from the user information table.

```
$dataProvider = new ActiveDataProvider([
'query' => Useradmin::find(),
]);
```

Load the view to which you want to transfer the data object.

```
return $this->render('index', [
'dataProvider' => $dataProvider,
]);
}
```

A method designed to display the data of a single user whose identifier is provided in the first parameter.

```
public function actionView($id)
{
```

Save menus in administrative logs to access single user data.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_user_one'));
```

We load the view to which we pass the model parameter, in which the data of one user extracted from the table will be found using the **findModel()** method.

```
return $this->render('view', [
    'model' => $this->findModel($id),
]);
}
```

A method that allows you to create a user account.

```
public function actionCreate()
{
```

We load a model from **Useradmin()** class together with a scenario of adding a new user.

```
$model = new Useradmin(['scenario' => Useradmin::SCENARIO_NEW]);
```

We create a temporary password by drawing, adding salt and encrypting it. We also create the user key.

```
$TemporaryPasword = strtolower(Yii::$app->getSecurity()->generateRandomString(20));
$Salt = Yii::$app->params['saltPassword'];
$ReadyPassword = password_hash($TemporaryPasword.$Salt, PASSWORD_DEFAULT);
$model->user_password = $ReadyPassword;
$model->user_key = strtolower(Yii::$app->getSecurity()->generateRandomString(20));
```

We add data such as registration date, IP address, user account activation, activation date and computer IP number.

```
$model->user_register = date('Y-m-d H:i:s');
$model->user_registered_ip = $_SERVER['REMOTE_ADDR'];
$model->user_active = "Y";
$model->user_activated = date('Y-m-d H:i:s');
$model->user_activated_ip = $_SERVER['REMOTE_ADDR'];
```

We check if the form with user data was sent, the fields were validated correctly and if the new record was saved to the database.

```
if ($model->load(Yii::$app->request->post()) && $model->save())
{
```

We save the information in logs about adding a new user.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_create_user_account'));
```

We redirect the user to the **view** method together with the parameter specifying the identifier of the added record.

```
return $this->redirect(['view', 'id' => $model->user_id]);
} else {
```

When a form has not been sent or validated correctly, it should be displayed with a model submitted to it for field generation.

```
return $this->render('create', [
    'model' => $model,
]);
}
}
```

This method allows you to edit user data. The identifier of the user whose data will be changed is taken from the parameter.

```
public function actionUpdate($id)
{
```

We load the user data in the model by referring to the internal **findModel()** method, and we also set the scenario for editing.

```
$model = $this->findModel($id);
$model->scenario = 'edit';
```

We check if the form was sent, if the data was validated correctly and if it was saved in the table.

```
if ($model->load(Yii::$app->request->post()) && $model->save())
{
```

We include information about user updates in our logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_updated_user_account'));
```

We move the user to the **view** method together with the definition of the updated record identifier in the parameter.

```
return $this->redirect(['view', 'id' => $model->user_id]);
} else {
```

If the form is not sent, we load a view of the form together with the transfer of data from the model to it.

```
return $this->render('update', [
    'model' => $model,
]);
}
}
```

This method is used to delete a user whose identification number is given in the parameter.

```
public function actionDelete($id)
{
```

Use **findModel()** to find the user and then **delete()** on the user.

```
$this->findModel($id)->delete();
```

We save the information about the deletion of the user.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_deleted_user_account'));
```

We redirect the administrator to the main view.

```
return $this->redirect(['index']);
}
```

The method returns data on one record and on the basis of the first parameter that contains its identifier.

```
protected function findModel($id)
{
```

We create a query and check if the returned value is different from the empty one, and then return the result to the **model** object.

```
if (($model = Useradmin::findOne($id)) !== null) {
    return $model;
} else {
```

If there is no compliant record, we will report the error and return it to you.

```
throw new NotFoundException(Yii::t('app', 'page_does_not_exists'));
}
}
}
```

View - user administration - form

A form designed to add and edit user data in our system.

File location: ../views/useradmin/_form.php

Start the PHP file, load the helper to generate HTML tags and the form plug-in.


```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<div class="useradmin-form">
```

We start the form.

```
<?php $form = ActiveForm::begin([
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]); ?>
```

Check whether you are adding a new record or editing an existing record. If it is edited from the database, the field containing the e-mail address should be omitted.

```
<?php
if($model->isNewRecord)
{
    echo $form->field($model, 'user_email')->textInput(['maxlength' => true]);
}
?>
```

We print the fields for your name, telephone number, website and information.

```
<?= $form->field($model, 'user_namesurname')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'user_phone')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'user_www')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'user_about')->textarea(['rows' => 6]) ?>
```

We create a list, which we add to the field containing a statement whether the user has administrator privileges or not.

```
<?php
$itemToList['y'] = Yii::t('app', 'a_yes');
$itemToList['n'] = Yii::t('app', 'a_no');
echo $form->field($model, 'user_root')->dropDownList($itemToList);
?>
```

We print the button confirming the form.

```
<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? Yii::t('app', 'a_create_user_button') : Yii::t('app',
'a_update_user_button'), ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>
```

We finish the form.

```
<?php ActiveForm::end(); ?>
</div>
```

View - user administration - adding a user

The template displays a form to add a new user to the system.

File location: ../views/useradmin/create.php

Start the PHP file and load the HTML helper.

```
<?php
use yii\helpers\Html;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_users_create_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_users_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="useradmin-create">
```

We print the header and load a file with the form.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
```

View - user administration - list of users

The page on which user accounts will be displayed together with the buttons used to add a new account, edit the account, see its details, and delete the user.

File location: ../views/useradmin/index.php

Start the PHP file, load the HTML helper and the plugin generating the grid.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title of the page and the location of the page.

```
$this->title = Yii::t('app', 'a_users_main_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="useradmin-index">
```

We print the header.

```
<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_users_create_header'), ['create'], ['class' =>'btn btn-success']) ?>
</p>
```

We print the grid and the data together with the table displaying, among others: user ID, e-mail address, name and surname, date of registration and IP from which the subscription was made. The buttons for deleting the user are not displayed at the administrator.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' =>'yii\grid\SerialColumn'],
        'user_id',
        'user_email:email',
        'user_namesurname',
        'user_register',
        'user_registered_ip',

        ['class' =>'yii\grid\ActionColumn',
        'visibleButtons' => [
            'delete' => function ($model, $key, $index) {
                return $model->user_id == 1 ? false : true;
            }
        ]
        ],
    ],
]); ?>
</div>
```

View - user administration - user update

A form designed to update the data on a user selected from a census.

File location: ../views/useradmin/update.php

Start the PHP file and load the helper to generate HTML characters.

```
<?php
use yii\helpers\Html;
```

We define the title and location of the file.

```
$this->title = Yii::t('app', 'a_users_update_header').': '.$model->user_email;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_users_update_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="useradmin-update">
```

We print the header and load a form that allows the administration of user data.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
```

View - user administration - account details

All data about the user added to the system is displayed in a view.

File location: ../views/useradmin/view.php

We start the PHP file, load the HTML helper and the plugin that allows to display the details of the record.

```
<?php
use yii\helpers\Html;
use yii\widgets\DetailView;
```

We define the title of a page together with its location.

```
$this->title = $model->user_email;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_users_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="useradmin-view">
```

We print the header and the button for editing the data.

```

<h1><?= Html::encode($this->title) ?></h1>

<p>

<?= Html::a(Yii::t('app', 'a_users_update_header'), ['update', 'id' => $model->user_id], ['class' =>'btn btn-primary']) ?>

<?php

```

Check if the user you are editing has an identifier with the number 1, if this is the case you should not show the button for deleting the account.

```

if($model->user_id != 1)
{
echo Html::a(Yii::t('app', 'a_users_delete_header'), ['delete', 'id' => $model->user_id], [
'class' =>'btn btn-danger',
'data' => [
'confirm' => Yii::t('app', 'a_delete_shure'),
'method' =>'post',
],
]);
}
?>
</p>

```

We print detailed data on all columns in the selected record.

```

<?= DetailView::widget([
'model' => $model,
'attributes' => [
'user_id',
'user_email:email',
'user_namesurname',
'user_phone',
'user_www',
'user_about:ntext',
'user_password',
'user_key',
'user_register',
'user_registered_ip',
'user_active',
'user_activated',
'user_activated_ip',
'user_root',
],
]) ?>
</div>

```

Chapter 11. Management of the pages

Text page management model - PageAdmin

The model will be responsible for the possibility of editing the pages, i.e. selecting, adding, editing or deleting them. We will use **ActiveRecord** class to create it, so we can use special methods for constructing queries in controllers.

File location: ../models/Pageadmin.php

Start the file, define the namespace, and load the Yii framework.

```
<?php
namespace app\models;
use Yii;
```

We define the name of the class having the same name as the file, which is an extension of **ActiveRecord** class.

```
class Pageadmin extends \yii\db\ActiveRecord
{
```

The method that returns the name of a table. The name is surrounded by double braces and a percentage mark. This will add a prefix to it, which we defined earlier in the file with the configuration of the database connection.

```
public static function tableName()
{
    return '{{%page}}';
}
```

A method that returns the rules to be executed before adding a record to the database. List the required fields and information on the type of data together with the maximum number of characters.

```
public function rules()
{
    return [
        [['page_title', 'page_text', 'page_url'], 'required'],
        [['page_text'], 'string'],
        [['page_title', 'page_url'], 'string', 'max' => 150],
    ];
}
```

The method returns a table containing the field names in the form as keys, the values are their description.

```

public function attributeLabels()
{
    return [
        'page_id' => Yii::t('app', 'page_id'),
        'page_title' => Yii::t('app', 'page_title'),
        'page_text' => Yii::t('app', 'page_text'),
        'page_url' => Yii::t('app', 'page_url'),
    ];
}
}

```

Controller of the administration of the pages - PageadminController

Controller designed for administration of pages, i.e. adding, editing and deleting them. It will be based on the **ActiveRecord** interface.

File location: ../controllers/PageadminController.php

We start PHP file, define names zone, load framework, data model, class responsible for data generation, main controller, error handling and filtering input data.

```

<?php
namespace app\controllers;
use Yii;
use app\models\Pageadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

```

We define a class name identical to the file name, it will be the extension of the main controller.

```

class PageadminController extends Controller
{

```

We set the appearance of the page to the administrator's template.

```

public $layout = 'admin';

```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```

public function beforeAction($action)
{

```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
    if($session['yii_user_root'] != "y")
    {
        return $this->redirect(['/right']);
    }
}
```

If the user is not logged in, we will take him to the login form page.

```
else
{
    return $this->redirect(['/login']);
}
```

At the end we call the method with the same name from the master controller.

```
return parent::beforeAction($action);
}
```

The method allows us to define the behavior of the application.

```
public function behaviors()
{
    return [
        'verbs' => [
```

We add that variables sent in the delete method can be sent only by the POST method.

```
'class' => VerbFilter::className(),
'actions' => [
    'delete' => ['POST'],
],
];
}
```


The method will be used to display pages and actions available for them.

```
public function actionIndex()
{
```

We save in the logs of the system the information that the user was browsing the pages.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_page'));
```

Using the interface, all the pages in the table are selected according to the default limit.

```
$dataProvider = new ActiveDataProvider([
    'query' => Pageadmin::find(),
]);
```

We generate the appearance from the template together with the transfer of the object containing the data to it.

```
return $this->render('index', [
    'dataProvider' => $dataProvider,
]);
}
```

A method that allows access to the data of a single page whose identifier is passed on in a parameter.

```
public function actionView($id)
{
```

We save the information about the fact that the user has seen detailed data of one page.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_one_page'));
```

We create from the template the appearance of the page. Load the model in which the data of one record was selected into the **view**.

```
return $this->render('view', [
    'model' => $this->findModel($id),
]);
}
```

A method that allows you to create a new page in the system.

```
public function actionCreate()
{
```

Read the **Pageadmin()** class as a model.

```
$model = new Pageadmin();
```

We check if the form was sent, if it was validated and if the new page in the table was saved.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

We save the information about the creation of a new website in our logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_create_page'));
```

We move the user to the **view** action together with the transfer of the parameter with the page identifier.

```
return $this->redirect(['view', 'id' => $model->page_id]);  
} else {
```

If the form is not sent we generate a view to which we pass the model.

```
return $this->render('create', [  
    'model' => $model,  
]);  
}  
}
```

A method that updates the record with the page whose identifier is transmitted in the first parameter.

```
public function actionUpdate($id)  
{
```

Load the record data into the model using the internal **findModel()** method.

```
$model = $this->findModel($id);
```

We check whether the form has been sent and whether its validation and data saving have been carried out.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

We save the information about the website update in our logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_update_page'));
```

We move the user to the view action together with the identifier of the updated record.

```
return $this->redirect(['view', 'id' => $model->page_id]);  
} else {
```

If the form is not sent, we generate its appearance and pass it on to the model.

```
return $this->render('update', [  
    'model' => $model,  
]);  
}  
}
```

A method that removes one page from a table, the identifier of which is provided in the first parameter.

```
public function actionDelete($id)  
{
```

Select the page using the internal **findModel()** method and then delete it.

```
$this->findModel($id)->delete();
```

We save the information about deleting the record.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_deleted_page'));
```

We redirect you to the main method.

```
return $this->redirect(['index']);  
}
```

A method that allows you to load a model object together with the data. The data is taken from an argument with a row identifier.

```
protected function findModel($id)  
{
```

Load the record based on the identifier into the model.

```
if (($model = Pageadmin::findOne($id)) !== null) {  
    return $model;  
} else {
```

If the record cannot be found, an error is displayed to the user.

```

throw new NotFoundException(Yii::t('app', 'page_does_not_exists'));
}
}
}

```

View - administration of the pages - form

The form is designed for adding and editing pages available in the system.

File location: ../views/pageadmin/_form.php

Start the PHP file, load the HTML helper and the form generation plugin.

```

<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<div class="pageadmin-form">

```

We start the form.

```

<?php $form = ActiveForm::begin([
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]); ?>

```

We create and print fields such as page title, page content and URL.

```

<?= $form->field($model, 'page_title')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'page_text')->textarea(['rows' => 6]) ?>
<?= $form->field($model, 'page_url')->textInput(['maxlength' => true]) ?>

```

We print the button used for adding or editing the record.

```

<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? Yii::t('app', 'a_create_user_button') : Yii::t('app',
'a_update_user_button'), ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>

```

We finish the form.

```

<?php ActiveForm::end(); ?>
</div>

```

View - page administration - adding a page

Adding a new page to the system, where you can define elements such as title, content using HTML editor and part of URL.

File location: ../views/pageadmin/create.php

Start the PHP file and load the helper for generating HTML tags.

```
<?php
use yii\helpers\Html;
```

We define the title and location of the current page.

```
$this->title = Yii::t('app', 'a_create_text_page_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_page_text_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="pageadmin-create">
```

We print the header and process the file with the form.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
<script>
```

With JavaScript we create a code that is run in case of changes in the field with the page title. It will rewrite the value of the field to the field containing the URL, but without the characters that should not be in the url.

```
$("#pageadmin-page_title").change(function () {
    str = $("#pageadmin-page_title").val();
```

We convert all Polish diacritic signs into their equivalents.

```
strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
.replace(/ć/g, 'c').replace(/Ć/g, 'C')
.replace(/ę/g, 'e').replace(/Ę/g, 'E')
.replace(/ł/g, 'l').replace(/Ł/g, 'L')
.replace(/ń/g, 'n').replace(/Ń/g, 'N')
.replace(/ó/g, 'o').replace(/Ó/g, 'O')
.replace(/ś/g, 's').replace(/Ś/g, 'S')
.replace(/ź/g, 'z').replace(/Ź/g, 'Z')
.replace(/ż/g, 'z').replace(/Ż/g, 'Z');
```

Convert a string into small characters, replace all characters that cannot be in the URL, and convert spaces into dashes.

```
strReady = strChangeLang.toLowerCase().replace(/[^\a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
$("#pageadmin-page_url").val(strReady);
}).change();
```

Configuration of the HTML editor plug-in **CKEditor**.

```
CKEDITOR.config.toolbarGroups = [
    { name: 'document', groups: [ 'mode', 'document', 'doctools' ] },
    { name: 'clipboard', groups: [ 'clipboard', 'undo' ] },
    { name: 'editing', groups: [ 'find', 'selection', 'spellchecker', 'editing' ] },
    { name: 'forms', groups: [ 'forms' ] },
    '/',
    { name: 'basicstyles', groups: [ 'basicstyles', 'cleanup' ] },
    { name: 'paragraph', groups: [ 'list', 'indent', 'blocks', 'align', 'bidi', 'paragraph' ] },
],

    { name: 'links', groups: [ 'links' ] },
    { name: 'insert', groups: [ 'insert' ] },
    '/',
    { name: 'styles', groups: [ 'styles' ] },
    { name: 'colors', groups: [ 'colors' ] },
    { name: 'tools', groups: [ 'tools' ] },
    { name: 'others', groups: [ 'others' ] },
    { name: 'about', groups: [ 'about' ] }

];

CKEDITOR.config.removeButtons = 'Save,NewPage,Preview,Print,Templates';
CKEDITOR.config.allowedContent = true;
CKEDITOR.config.entities_latin = false;
CKEDITOR.replace('pageadmin-page_text');
CKEDITOR.config.height = 500;
CKEDITOR.config.skin = 'office2013';
</script>
```

View - page administration - index of pages

It contains a list of pages generated with buttons for adding a new page and editing, deleting and previewing existing pages.

File location: ../views/pageadmin/index.php

We start PHP, load HTML helper and grid generation plug.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_page_text_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="pageadmin-index">
```

We print the header and the button for adding a new page.

```
<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_create_text_page_header'), ['create'], ['class' =>'btn btn-success']) ?>
</p>
```

The grid allows us to automatically generate a view in which columns such as the page identifier and its title will be available. By defining the function in **visibleButtons**, the column for operation will not be displayed next to page **1**, which is set as the home page.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' =>'yii\grid\SerialColumn'],
        'page_id',
        'page_title',
        ['class' =>'yii\grid\ActionColumn',
        'visibleButtons' => [
            'delete' => function ($model, $key, $index) {
                return $model->page_id == 1 ? false : true;
            }
        ],
    ],
]); ?>
</div>
```

View - page administration - page update

The page can be edited in the update view. It is also equipped with an **WYSIWYG** editor to make it easy to insert HTML content.

File location: ../views/pageadmin/update.php

Start the PHP file and load the helper to generate HTML tags.

```
<?php
```

```
use yii\helpers\Html;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_edit_text_page_header').': ' . $model->page_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_page_text_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="pageadmin-update">
```

We print the header and load the file containing the form.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
<script>
```

The JavaScript code allows us to properly process the contents of the title field by cutting or replacing the letters so that all characters are compliant with the URL standards.

```
$("#pageadmin-page_title").change(function () {
    str = $("#pageadmin-page_title").val();
```

We convert Polish diacritic signs into their equivalents.

```
strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
.replace(/ć/g, 'c').replace(/Ć/g, 'C')
.replace(/ę/g, 'e').replace(/Ę/g, 'E')
.replace(/ł/g, 'l').replace(/Ł/g, 'L')
.replace(/ń/g, 'n').replace(/Ń/g, 'N')
.replace(/ó/g, 'o').replace(/Ó/g, 'O')
.replace(/ś/g, 's').replace(/Ś/g, 'S')
.replace(/ż/g, 'z').replace(/Ż/g, 'Z')
.replace(/ź/g, 'z').replace(/Ź/g, 'Z');
```

Convert a string into small characters, remove everything except letters and numbers, and replace spaces with a dash.

```
strReady = strChangeLang.toLowerCase().replace(/[^a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
$("#pageadmin-page_url").val(strReady);
}).change();
```

We will configure the WYSIWYG editor **CKEditor**.


```

CKEDITOR.config.toolbarGroups = [
    { name: 'document', groups: [ 'mode', 'document', 'doctools' ] },
    { name: 'clipboard', groups: [ 'clipboard', 'undo' ] },
    { name: 'editing', groups: [ 'find', 'selection', 'spellchecker', 'editing' ] },
    { name: 'forms', groups: [ 'forms' ] },
    '/',
    { name: 'basicstyles', groups: [ 'basicstyles', 'cleanup' ] },
    { name: 'paragraph', groups: [ 'list', 'indent', 'blocks', 'align', 'bidi', 'paragraph' ]
},

    { name: 'links', groups: [ 'links' ] },
    { name: 'insert', groups: [ 'insert' ] },
    '/',
    { name: 'styles', groups: [ 'styles' ] },
    { name: 'colors', groups: [ 'colors' ] },
    { name: 'tools', groups: [ 'tools' ] },
    { name: 'others', groups: [ 'others' ] },
    { name: 'about', groups: [ 'about' ] }

];

CKEDITOR.config.removeButtons = 'Save,NewPage,Preview,Print,Templates';
CKEDITOR.config.allowedContent = true;
CKEDITOR.config.entities_latin = false;
CKEDITOR.replace('pageadmin-page_text');
CKEDITOR.config.height = 500;
CKEDITOR.config.skin = 'office2013';
</script>

```

View - page administration - page details

The detailed data displays all the columns available in the table for storing pages.

File location: ../views/pageadmin/view.php

We start PHP file, read HTML helper and plugin thanks to which we can display details about the record.

```

<?php
use yii\helpers\Html;
use yii\widgets\DetailView;

```

We define the title of a page together with its location.

```

$this->title = $model->page_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_page_text_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="pageadmin-view">

```

We print the page header and the button for editing the content.

```
<h1><?= Html::encode($this->title) ?></h1>

<p>

<?= Html::a(Yii::t('app', 'a_edit_text_page_header'), ['update', 'id' => $model->page_id], ['class' =>'btn btn-primary']) ?>

<?php
```

We check whether the page identifier is different from one. If this is the case, a button can also be printed out for deletion.

```
if($model->page_id != 1)
{
echo Html::a(Yii::t('app', 'a_delete_text_page_button'), ['delete', 'id' => $model->page_id], [
'class' =>'btn btn-danger',
'data' => [
'confirm' => Yii::t('app', 'a_delete_shure'),
'method' =>'post',
],
]) ;
}
?>
</p>
```

Using the plugin, we print details of records from the database, such as: record ID, title of the page, its content and a fragment of the URL.

```
<?= DetailView::widget([
'model' => $model,
'attributes' => [
'page_id',
'page_title',
'page_text:ntext',
'page_url',
],
]) ?>
</div>
```

Chapter 12. Article management

ArticleAdmin - article management model

This article administration model will be an extension of the ActiveRecord model, allowing you to query using special methods instead of creating a pure SQL query.

File location: ../models/Articleadmin.php

We start the PHP file, set the namespace to **app\models** and load the framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class according to the filename, which will be an extension of **ActiveRecord** class allowing for automation of queries processing.

```
class Articleadmin extends \yii\db\ActiveRecord
{
```

This method returns the name of the table on which the operations will be performed. It is in two braces and is preceded by a percentage sign. This will add to the table name its prefix defined by us in the connection configuration file.

```
public static function tableName()
{
    return '{{%article}}';
}
```

A method that contains rules for adding or editing a record. Defines the required fields and the maximum number of characters they can accept.

```
public function rules()
{
    return [
        [['article_title', 'article_text', 'article_author', 'article_date', 'article_url'], 'required'],
        [['article_text'], 'string'],
        [['article_date'], 'safe'],
        [['article_title'], 'string', 'max' => 150],
        [['article_author'], 'string', 'max' => 55],
        [['article_url'], 'string', 'max' => 65],
    ];
}
```

A method that returns field names, for example in their descriptions, when generating forms. The table contains the name of a field as a key and its description as a value.

```
public function attributeLabels()
{
    return [
        'article_id' => Yii::t('app', 'art_id'),
        'article_title' => Yii::t('app', 'art_title'),
        'article_text' => Yii::t('app', 'art_content'),
        'article_author' => Yii::t('app', 'art_author'),
        'article_date' => Yii::t('app', 'art_data'),
        'article_url' => Yii::t('app', 'art_url'),
    ];
}
```

Controller of the administration of articles - ArticleAdminController

Controller designed for administration of the article section on our website. It contains the possibility of displaying the content of the article, adding new content, editing and deleting the one present in the table.

File location: ../controllers/ArticleAdminController.php

We start the file, define the names zone, load: framework, model for article administration, tool for placing data on the page, controller class, error information and filter.

```
<?php
namespace app\controllers;
use Yii;
use app\models\Articleadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

We create a class name identical to the file name, which will be an extension of the main class of the controller.

```
class ArticleAdminController extends Controller
{
```

We set the main view of the page on administration.

```
public $layout = 'admin';
```

The method is performed before proceeding to further implementation of the methods. It is possible to place in it various types of checks, which we want to perform before the realization of the object.

```
public function beforeAction($action)
{
```

Load the session and check if the user is logged in and has the administrator status. Otherwise, we will redirect you to the login page or show you information that you do not have the appropriate access rights.

```
$session = Yii::$app->session;
if($session['yii_user_id'] != "")
{
    if($session['yii_user_root'] != "y")
    {
        return $this->redirect(['/right']);
    }
}
else
{
    return $this->redirect(['/login']);
}
return parent::beforeAction($action);
}
```

The method allows to determine the behavior of the framework in case of variables transmission. This case allows you to delete an entry if the parameters defining this function and the record identifier have not been sent by **POST**.

```
public function behaviors()
{
    return [
        'verbs' => [
            'class' => VerbFilter::className(),
            'actions' => [
                'delete' => ['POST'],
            ],
        ],
    ];
}
```

The method is designed to show articles.

```
public function actionIndex()
{
```

Using the method for saving **WriteLog()** logs we enter into the table the information that this section of the admin panel was used by the user.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_articles'));
```

We create **ActiveDataProvider()** object in which we specify **Articleadmin** model together with **find()** query.

```
$dataProvider = new ActiveDataProvider([  
'query' => Articleadmin::find(),  
]);
```

We load a view to which the **dataProvider** object should be passed in order to generate content on the page.

```
return $this->render('index', [  
'dataProvider' => $dataProvider,  
]);  
}
```

This method is used to display the data of one article whose identifier is the first parameter.

```
public function actionView($id)  
{
```

We save the message in the administrator logs for access to this section.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_one_article'));
```

We load the view together with the definition of the model object referring to the method used to search for the record together with the transfer of its parameter.

```
return $this->render('view', [  
'model' => $this->findModel($id),  
]);  
}
```

Method for creating a new article.

```
public function actionCreate()  
{
```

The class of the **Articleadmin()** model is transformed into an object.

```
$model = new Articleadmin();
```

We check whether the form has been sent and save the data in the table.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

We create an entry in the system log.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_create_article'));
```

We redirect the page request to **view** action.

```
return $this->redirect(['view', 'id' => $model->article_id]);  
} else {
```

If the form has not been sent, read the view with the fields to be filled in.

```
return $this->render('create', [  
    'model' => $model,  
]);  
}  
}
```

Method responsible for keeping the alert up to date. It takes one parameter, which is the identifier of the article to be updated.

```
public function actionUpdate($id)  
{
```

The model will refer to a method that allows you to search for a record with its identifier.

```
$model = $this->findModel($id);
```

Check if all fields have been filled in according to your preferences and save the data in the table.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

We enter the information about the article update.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_update_article'));
```

We redirect the user to the view method and enter the record identifier in the parameter.

```
return $this->redirect(['view', 'id' => $model->article_id]);  
} else {
```

If the form is not sent we read its view and pass it in the model parameter.

```
return $this->render('update', [  
    'model' => $model,
```

```
'model' => $model,
]);
}
}
```

This method is used to delete one record, the identifier of which is given in the first parameter.

```
public function actionDelete($id)
{
```

Using the **findModel()** method, load a given record based on its identifier and then perform a deletion operation on it.

```
$this->findModel($id)->delete();
```

We save the information in the logs that the record has been deleted.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_deleted_article'));
```

We redirect you to the main view.

```
return $this->redirect(['index']);
}
```

The method shall allow a search to be made for the record corresponding to the identifier given in the first argument.

```
protected function findModel($id)
{
```

Read the **Articleadmin** class as an object and select one record, then check if the returned value is not empty.

```
if (($model = Articleadmin::findOne($id)) !== null) {
```

We return the model with the record.

```
return $model;
} else {
```

In case the method does not return the record, we generate an error with information about a page that does not exist.

```
throw new NotFoundException(Yii::t('app', 'page_does_not_exists'));
}
}
}
```


View - article administration - form

Form designed for adding a new article or editing an existing one.

File location: ../views/articleadmin/_form.php

Start the PHP file, load the HTML helper and the form plugin.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;

?>
```

We create the form.

```
<div class="articleadmin-form">
<?php $form = ActiveForm::begin([
'enableAjaxValidation' => false,
'enableClientValidation' => false,
]); ?>
```

We include fields such as article title, content, author, date of publication, and URL.

```
<?= $form->field($model, 'article_title')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'article_text')->textarea(['rows' => 6]) ?>
<?= $form->field($model, 'article_author')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'article_date')->textInput() ?>
<?= $form->field($model, 'article_url')->textInput(['maxlength' => true]) ?>
```

Print the button to add or update the record.

```
<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? Yii::t('app', 'a_article_add') : Yii::t('app',
'a_article_edit'), ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>
```

We finish the form.

```
<?php ActiveForm::end(); ?>
</div>
```

View - article administration - adding a new article

The view file contains the form and functions you need to create a new article.

File location: ../views/articleadmin/create.php

Start the PHP file and load the helper generating the HTML tags.

```
<?php
use yii\helpers\Html;
```

We define the title and location of the current page.

```
$this->title = Yii::t('app', 'a_article_add');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_article_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
```

Print the header and load the form.

```
<div class="articleadmin-create">
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
'model' => $model,
]) ?>
</div>
```

We define the JavaScript code so that you can add the current date and time to the page by clicking in the field.

```
<script>
$('#articleadmin-article_date').datetimepicker({
timeFormat: "HH:mm:ss",
dateFormat: "yy-mm-dd"
});
```

We define in **JavaScript** the conversion of diacritic signs into ordinary letters in the case of Polish language.

```
$("#articleadmin-article_title").change(function () {
str = $("#articleadmin-article_title").val();
strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
.replace(/ć/g, 'c').replace(/Ć/g, 'C')
.replace(/ę/g, 'e').replace(/Ę/g, 'E')
.replace(/ł/g, 'l').replace(/Ł/g, 'L')
.replace(/ń/g, 'n').replace(/Ń/g, 'N')
.replace(/ó/g, 'o').replace(/Ó/g, 'O')
.replace(/ś/g, 's').replace(/Ś/g, 'S')
.replace(/ź/g, 'z').replace(/Ź/g, 'Z')
.replace(/ż/g, 'z').replace(/Ż/g, 'Z');
```

```

strReady = strChangeLang.toLowerCase().replace(/[a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
$("#articleadmin-article_url").val(strReady);
}).change();

```

Configure the **CKEditor** plugin.

```

CKEDITOR.config.toolbarGroups = [
    { name: 'document', groups: [ 'mode', 'document', 'doctools' ] },
    { name: 'clipboard', groups: [ 'clipboard', 'undo' ] },
    { name: 'editing', groups: [ 'find', 'selection', 'spellchecker', 'editing' ] },
    { name: 'forms', groups: [ 'forms' ] },
    '/',
    { name: 'basicstyles', groups: [ 'basicstyles', 'cleanup' ] },
    { name: 'paragraph', groups: [ 'list', 'indent', 'blocks', 'align', 'bidi', 'paragraph' ]
},

    { name: 'links', groups: [ 'links' ] },
    { name: 'insert', groups: [ 'insert' ] },
    '/',
    { name: 'styles', groups: [ 'styles' ] },
    { name: 'colors', groups: [ 'colors' ] },
    { name: 'tools', groups: [ 'tools' ] },
    { name: 'others', groups: [ 'others' ] },
    { name: 'about', groups: [ 'about' ] }

];

CKEDITOR.config.removeButtons = 'Save,NewPage,Preview,Print,Templates';
CKEDITOR.config.allowedContent = true;
CKEDITOR.config.entities_latin = false;
CKEDITOR.replace('articleadmin-article_text');
CKEDITOR.config.height = 500;
CKEDITOR.config.skin = 'office2013';
</script>

```

View - administration of articles - list of articles

The list of articles allows us to browse them by page, there is a menu item next to the articles for editing, deleting or viewing details.

File location: ../views/articleadmin/index.php

Start the PHP file, load the helper for generating HTML tags, and load the grid transformation library.

```

<?php
use yii\helpers\Html;
use yii\grid\GridView;

```

We define the title of the page and the current position.

```

$this->title = Yii::t('app', 'a_article_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>

```

We print the title and the button allowing you to add a new article.

```

<div class="articleadmin-index">
<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_article_add'), ['create'], ['class' => 'btn btn-success']) ?>
</p>

```

Using the grid, you can create the page appearance by selecting columns such as: entry ID, title, author, and date of publication.

```

<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'article_id',
        'article_title',
        'article_author',
        'article_date',
        ['class' => 'yii\grid\ActionColumn', 'template' => '{view} {update} {delete}'],
    ],
]); ?>
</div>

```

View - article administration - article update

The view file is used to display the article editing form.

File location: ../views/articleadmin/update.php

Start the PHP file and load the helper for generating HTML tags.

```

<?php
use yii\helpers\Html;

```

We define the title of a page and its location.

```

$this->title = Yii::t('app', 'a_article_edit').': '.$model->article_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];

```

```

$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_article_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="articleadmin-update">

```

We print the header and load the file containing the form.

```

<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
<script>

```

We define the action of developing the calendar with the date and time after clicking on the field.

```

$('#articleadmin-article_date').datetimepicker({
    timeFormat: "HH:mm:ss",
    dateFormat: "yy-mm-dd"
});

```

Because a title can be multilingual, you must remove characters that you do not want in the URL.

```

$("#articleadmin-article_title").change(function () {
    str = ($("#articleadmin-article_title").val());
    strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
        .replace(/ć/g, 'c').replace(/Ć/g, 'C')
        .replace(/ę/g, 'e').replace(/E/g, 'E')
        .replace(/ł/g, 'l').replace(/Ł/g, 'L')
        .replace(/ń/g, 'n').replace(/Ń/g, 'N')
        .replace(/ó/g, 'o').replace(/Ó/g, 'O')
        .replace(/ś/g, 's').replace(/Ś/g, 'S')
        .replace(/ż/g, 'z').replace(/Ż/g, 'Z')
        .replace(/ź/g, 'z').replace(/Ź/g, 'Z');
    strReady = strChangeLang.toLowerCase().replace(/[^a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
    $("#articleadmin-article_url").val(strReady);
}).change();

```

We will configure the **CKEditor** editor.

```

CKEDITOR.config.toolbarGroups = [
    { name: 'document', groups: [ 'mode', 'document', 'doctools' ] },
    { name: 'clipboard', groups: [ 'clipboard', 'undo' ] },
    { name: 'editing', groups: [ 'find', 'selection', 'spellchecker', 'editing' ] },
    { name: 'forms', groups: [ 'forms' ] },
    '/',

```

```

        { name: 'basicstyles', groups: [ 'basicstyles', 'cleanup' ] },
        { name: 'paragraph', groups: [ 'list', 'indent', 'blocks', 'align', 'bidi', 'paragraph' ]
    },

    { name: 'links', groups: [ 'links' ] },
    { name: 'insert', groups: [ 'insert' ] },
    '/',
    { name: 'styles', groups: [ 'styles' ] },
    { name: 'colors', groups: [ 'colors' ] },
    { name: 'tools', groups: [ 'tools' ] },
    { name: 'others', groups: [ 'others' ] },
    { name: 'about', groups: [ 'about' ] }

    ];
CKEDITOR.config.removeButtons = 'Save,NewPage,Preview,Print,Templates';
CKEDITOR.config.allowedContent = true;
CKEDITOR.config.entities_latin = false;
CKEDITOR.replace('articleadmin-article_text');
CKEDITOR.config.height = 500;
CKEDITOR.config.skin = 'office2013';
</script>

```

View - article administration - full data article display

The article has only part of the data in the table of contents. Clicking on an article in the list will take you to its full details.

File location: ../views/articleadmin/view.php

We start the PHP file, load the HTML helper and the plug-in for detailed data presentation.

```

<?php
use yii\helpers\Html;
use yii\widgets\DetailView;

```

We define the title of the page and our current location.

```

$this->title = $model->article_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_article_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>

```

We print the title and links for editing the article and for deleting it.

```

<div class="articleadmin-view">
<h1><?= Html::encode($this->title) ?></h1>

<p>

<?= Html::a(Yii::t('app', 'a_article_edit'), ['update', 'id' => $model->article_id], ['class' => 'btn btn-
primary']) ?>

```

```

<?= Html::a(Yii::t('app', 'a_article_delete'), ['delete', 'id' => $model->article_id], [
    'class' =>'btn btn-danger',
    'data' => [
        'confirm' => Yii::t('app', 'a_delete_shure'),
        'method' =>'post',
    ],
]) ?>
</p>

```

In the detail view, we define all the columns that are contained in the parts table.

```

<?= DetailView::widget([
    'model' => $model,
    'attributes' => [
        'article_id',
        'article_title',
        'article_text:ntext',
        'article_author',
        'article_date',
        'article_url',
    ],
]) ?>
</div>

```

Chapter 13. Management of the blog

Blog management model - BlogAdmin

A model that allows us to administer a blog on our website. Thanks to it we will be able to add entries, edit their content and also delete them.

File location: ../models/Blogadmin.php

Start the file, define the namespace, and then load the Yii framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class name, which is identical to the file name, which is an extension of **ActiveRecord** class.

```
class Blogadmin extends \yii\db\ActiveRecord
{
```

The method returns the name of the table, which is located inside the double braces and is preceded by a percentage character. This will add the prefix defined in the database connection configuration file to the table name.

```
public static function tableName()
{
    return '{{%blog}}';
}
```

The method returns validation rules for data when adding or editing a record. It checks whether the required fields have been completed and defines the fields as strings of characters of the maximum length, as safe parameters and numbers.

```
public function rules()
{
    return [
        [['blog_title', 'blog_text', 'blog_date', 'blog_url', 'blog_category', 'blog_hashtag'], 'required'],
        [['blog_text'], 'string'],
        [['blog_date'], 'safe'],
        [['blog_category'], 'integer'],
        [['blog_title', 'blog_url'], 'string', 'max' => 150],
        [['blog_hashtag'], 'string', 'max' => 255],
    ];
}
```


The method retrieves the list of blog categories from the database and returns it in a processed form.

```
public function CreateCategoryList()
{
```

We create and execute queries regarding categories in the blog.

```
$QueryData = Yii::$app->db->createCommand('SELECT * FROM {%blog_category}%')->queryAll();
```

The result of the query using the fork loop is processed into a special **\$CategoryList** array in which the key will be the identification number of the category and the value of its name.

```
for($c=0;$c<count($QueryData);$c++)
{
    $CategoryList[$QueryData[$c]['category_id']] = $QueryData[$c]['category_title'];
}
```

We return the tables resulting from the method.

```
return $CategoryList;
}
```

The method returns an array in which the keys are the names of the form fields and the value of their description.

```
public function attributeLabels()
{
    return [
        'blog_id' => Yii::t('app', 'blog_id'),
        'blog_title' => Yii::t('app', 'blog_title'),
        'blog_text' => Yii::t('app', 'blog_content'),
        'blog_date' => Yii::t('app', 'blog_date'),
        'blog_url' => Yii::t('app', 'blog_url'),
        'blog_category' => Yii::t('app', 'blog_category'),
        'blog_hashtag' => Yii::t('app', 'blog_tag'),
    ];
}
```

Controller of the blog administration - BlogAdminController

Controller created to administer the blog section of our website. It is responsible for adding, editing and deleting individual entries available in our logbook. Due to the fact that it is the administrator's side we can use the **ActiveRecord** interface without any problems.

File location: ../controllers/BlogadminController.php

We start PHP file, define namespace, load **Blogadmin** model, **ActiveDataProvider** interface, controller main class, class allowing to define errors and filter the way of data transfer.

```
<?php
namespace app\controllers;
use Yii;
use app\models\Blogadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

We define a class name identical to the file name, which will be the extension of the main class of the controller.

```
class BlogadminController extends Controller
{
```

We set the page template to administrative.

```
public $layout = 'admin';
```

The method will cause its content to be run before any method is executed. This makes it perfect for checking whether the user has administrative privileges, instead of doing it in each method separately.

```
public function beforeAction($action)
{
```

Download the session settings.

```
$session = Yii::$app->session;
```

We check if the user is logged in.

```
if($session['yii_user_id'] != "")
{
```

We check whether the user has administrator rights. In case he does not have these rights, we redirect him to the page on which the information that he does not have the appropriate rights is placed.

```
if($session['yii_user_root'] != "y")
{
return $this->redirect(['/right']);
}
```

If the user is not logged in, we will redirect him to the login page.

```
}  
else  
{  
return $this->redirect(['/login']);  
}
```

We refer to the main class of controller in order to implement this method in the object being the mother.

```
return parent::beforeAction($action);  
}
```

The method that defines the ways of sending variables through the URL or **POST** method.

```
public function behaviors()  
{
```

We return an array of information in which there is a method of receiving data for the delete action as sent only by **POST** method.

```
return [  
'verbs' => [  
'class' => VerbFilter::className(),  
'actions' => [  
'delete' => ['POST'],  
],  
],  
];  
}
```

The main method, whose task is to show us a panel of blog administration.

```
public function actionIndex()  
{
```

We save the information about the section you visited.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_blog'));
```

Using **ActiveDataProvider** in which query parameter we enter to select data from **Blogadmin()** model.

```
$dataProvider = new ActiveDataProvider([  
'query' => Blogadmin::find(),
```

```
]);
```

We generate the appearance by passing to it the result of the search for entries from the **Blogadmin()** model.

```
return $this->render('index', [  
    'dataProvider' => $dataProvider,  
]);  
}
```

The method created to view a single entry takes the record identifier as a parameter.

```
public function actionView($id)  
{
```

We save the action of viewing a single entry in the database.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_one_entry_blog'));
```

We generate the appearance to which we pass the object containing data from the model on the basis of the given record identifier.

```
return $this->render('view', [  
    'model' => $this->findModel($id),  
]);  
}
```

The method is designed to create a new entry in the blog section.

```
public function actionCreate()  
{
```

We load the **Blogadmin()** class as a model.

```
$model = new Blogadmin();
```

We download the categories available in our blog.

```
$ItemToList = $model->CreateCategoryList();
```

We check whether the form has been sent and validated, and then we save the data.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

Enter the log of the new entry into the database.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_blog_add'));
```

We redirect the user to the view action and pass on the identifier of the placed entry.

```
return $this->redirect(['view', 'id' => $model->blog_id]);  
} else {
```

When the form has not been sent yet, we load it by sending it a model and a list of categories.

```
return $this->render('create', [  
'model' => $model, 'ItemList' => $ItemList,  
]);  
}  
}
```

A method that allows a record to be updated on the basis of its identifier, which is its first argument.

```
public function actionUpdate($id)  
{
```

The **Blogadmin()** class is read as a model object.

```
$model = new Blogadmin();
```

We create an array with a list of categories.

```
$ItemList = $model->CreateCategoryList();
```

Using the model, we search for an entry whose parameter was obtained in the first argument of the method.

```
$model = $this->findModel($id);
```

We check whether the form was sent, its fields validated and saved in the table.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

We add information to the logs about updating the entry.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_blog_updated'));
```

We redirect the user to the **view** action together with the transfer of the record identifier.

```
return $this->redirect(['view', 'id' => $model->blog_id]);
} else {
```

In case the form has not been sent yet, we load the view and transfer the model object and the category array to it.

```
return $this->render('update', [
'model' => $model, 'ItemList' => $ItemList,
]);
}
}
```

A method that allows you to delete a record whose identifier you pass on in the first parameter.

```
public function actionDelete($id)
{
```

Delete an entry by referring to a model, selecting a record, and defining the **delete()** method in it.

```
$this->findModel($id)->delete();
```

Save the deletion of the record in the table.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_blog_delete'));
```

We redirect the user to the main page of the controller.

```
return $this->redirect(['index']);
}
```

A method that returns an object with a selected record, which it selects on the basis of the first parameter in the form of an identifier.

```
protected function findModel($id)
{
```

We check if the record exists in the table and in this case we return it to its value.

```
if (($model = Blogadmin::findOne($id)) !== null) {
return $model;
} else {
```

If the specified record does not exist, we will return the error.

```

throw new NotFoundException(Yii::t('app', 'page_does_not_exists'));
}
}
}

```

View - administration of the blog - form

The form is designed to add a new entry to the blog section as well as edit an existing one.

File location: ../views/blogadmin/_form.php

Start the PHP file, load the HTML helper and the form generation plugin.

```

<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<div class="blogadmin-form">

```

We start the form.

```

<?php $form = ActiveForm::begin([
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]); ?>

```

We print the fields: title of the entry, content of the entry, date of entry, URL, category and hash tag.

```

<?= $form->field($model, 'blog_title')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'blog_text')->textarea(['rows' => 6]) ?>
<?= $form->field($model, 'blog_date')->textInput() ?>
<?= $form->field($model, 'blog_url')->textInput(['maxlength' => true]) ?>
<?php
echo $form->field($model, 'blog_category')->dropDownList($ItemToList);
?>
<?= $form->field($model, 'blog_hashtag')->textInput(['maxlength' => true]) ?>

```

Create a button to add or update the current entry.

```

<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? Yii::t('app', 'a_blog_add') : Yii::t('app', 'a_blog_edit'),
    ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>

```

We finish the form.

```
<?php ActiveForm::end(); ?>
</div>
```

View - administration of the blog - adding a new entry

Creating a new entry will be done by correctly filling in the fields of the form that will be included in the file.

File location: ../views/blogadmin/create.php

We start the PHP file and load the helper to generate HTML.

```
<?php
use yii\helpers\Html;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_blog_add');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_blog_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
```

We print the page header and include the form.

```
<div class="blogadmin-create">
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
'model' => $model, 'ItemToList' => $ItemToList,
]) ?>
</div>
<script>
```

We define that after clicking in the field appears the calendar with the date and time of the addition.

```
$('#blogadmin-blog_date').datetimepicker({
timeFormat: "HH:mm:ss",
dateFormat: "yy-mm-dd"
});
```

Because you need to create a URL in the title, you need to replace all the diacritical characters with letters and replace other characters that should not be added to the web address.

```
$("#blogadmin-blog_title").change(function () {
str = $("#blogadmin-blog_title").val();
```



```

strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
.replace(/ć/g, 'c').replace(/Ć/g, 'C')
.replace(/ę/g, 'e').replace(/Ę/g, 'E')
.replace(/ł/g, 'l').replace(/Ł/g, 'L')
.replace(/ń/g, 'n').replace(/Ń/g, 'N')
.replace(/ó/g, 'o').replace(/Ó/g, 'O')
.replace(/ś/g, 's').replace(/Ś/g, 'S')
.replace(/ż/g, 'z').replace(/Ż/g, 'Z')
.replace(/ź/g, 'z').replace(/Ź/g, 'Z');
strReady = strChangeLang.toLowerCase().replace(/[a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
$("#blogadmin-blog_url").val(strReady);
}).change();

```

Configuration of the CKEditor editor library.

```

CKEDITOR.config.toolbarGroups = [
    { name: 'document', groups: [ 'mode', 'document', 'doctools' ] },
    { name: 'clipboard', groups: [ 'clipboard', 'undo' ] },
    { name: 'editing', groups: [ 'find', 'selection', 'spellchecker', 'editing' ] },
    { name: 'forms', groups: [ 'forms' ] },
    '/',
    { name: 'basicstyles', groups: [ 'basicstyles', 'cleanup' ] },
    { name: 'paragraph', groups: [ 'list', 'indent', 'blocks', 'align', 'bidi', 'paragraph' ] },

    { name: 'links', groups: [ 'links' ] },
    { name: 'insert', groups: [ 'insert' ] },
    '/',
    { name: 'styles', groups: [ 'styles' ] },
    { name: 'colors', groups: [ 'colors' ] },
    { name: 'tools', groups: [ 'tools' ] },
    { name: 'others', groups: [ 'others' ] },
    { name: 'about', groups: [ 'about' ] }
];

CKEDITOR.config.removeButtons = 'Save,NewPage,Preview,Print,Templates';
CKEDITOR.config.allowedContent = true;
CKEDITOR.config.entities_latin = false;
CKEDITOR.replace('blogadmin-blog_text');
CKEDITOR.config.height = 500;
CKEDITOR.config.skin = 'office2013';
</script>

```

View - administration of the blog - lists of entries

All entries available in the blog will be available on the website together with actions such as: adding a new one, editing, deleting or viewing details.

File location: ../views/blogadmin/index.php

We start the PHP file, load the HTML helper and the tools for generating the grid.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title and location of the page.

```
$this->title = Yii::t('app', 'a_blog_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
```

We print the title and the transfer button to the form where we can add a new entry.

```
<div class="blogadmin-index">
<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_blog_add'), ['create'], ['class' =>'btn btn-success']) ?>
</p>
```

We generate a grid from the data received from the controller. The grid will contain fields such as: the identifier of the entry, the title of the entry and the date of publication.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' =>'yii\grid\SerialColumn'],
        'blog_id',
        'blog_title',
        'blog_date',
        ['class' =>'yii\grid\ActionColumn'],
    ],
]); ?>
</div>
```

View - administration of the blog - update of an entry

Each blog entry will be able to be updated by means of a special form.

File location: ../views/blogadmin/update.php

We start the PHP file and load the helper in order to generate HTML tags.

```
<?php
use yii\helpers\Html;
```

We define the title and location of the current page.

```
$this->title = Yii::t('app', 'a_blog_edit').': '.$model->blog_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_blog_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogadmin-update">
```

We print the page header.

```
<h1><?= Html::encode($this->title) ?></h1>
```

Include the view file that contains the form.

```
<?= $this->render('_form', [
    'model' => $model, 'ItemToList' => $ItemToList,
]) ?>
</div>
<script>
```

JavaScript code allowing you to click in the field with date and time to display the calendar.

```
$('#blogadmin-blog_date').datetimepicker({
    timeFormat: "HH:mm:ss",
    dateFormat: "yy-mm-dd"
});
```

Create a portion of the URL that you need to modify in order to use in your browser. Replace diacritical (Polish) characters and spaces or other special characters with characters that may be present in the page address.

```
$("#blogadmin-blog_title").change(function () {
    str = ($("#blogadmin-blog_title").val());
    strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
        .replace(/ć/g, 'c').replace(/Ć/g, 'C')
        .replace(/ę/g, 'e').replace(/Ę/g, 'E')
        .replace(/ł/g, 'l').replace(/Ł/g, 'L')
        .replace(/ń/g, 'n').replace(/Ń/g, 'N')
        .replace(/ó/g, 'o').replace(/Ó/g, 'O')
        .replace(/ś/g, 's').replace(/Ś/g, 'S')
        .replace(/ź/g, 'z').replace(/Ź/g, 'Z')
        .replace(/ż/g, 'z').replace(/Ż/g, 'Z');
    strReady = strChangeLang.toLowerCase().replace(/[^a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
```

```

$("#blogadmin-blog_url").val(strReady);
}).change();

```

Configuration of the **CKEditor** editor.

```

CKEDITOR.config.toolbarGroups = [
    { name: 'document', groups: [ 'mode', 'document', 'doctools' ] },
    { name: 'clipboard', groups: [ 'clipboard', 'undo' ] },
    { name: 'editing', groups: [ 'find', 'selection', 'spellchecker', 'editing' ] },
    { name: 'forms', groups: [ 'forms' ] },
    '/',
    { name: 'basicstyles', groups: [ 'basicstyles', 'cleanup' ] },
    { name: 'paragraph', groups: [ 'list', 'indent', 'blocks', 'align', 'bidi', 'paragraph' ] },
],

    { name: 'links', groups: [ 'links' ] },
    { name: 'insert', groups: [ 'insert' ] },
    '/',
    { name: 'styles', groups: [ 'styles' ] },
    { name: 'colors', groups: [ 'colors' ] },
    { name: 'tools', groups: [ 'tools' ] },
    { name: 'others', groups: [ 'others' ] },
    { name: 'about', groups: [ 'about' ] }

];

CKEDITOR.config.removeButtons = 'Save,NewPage,Preview,Print,Templates';
CKEDITOR.config.allowedContent = true;
CKEDITOR.config.entities_latin = false;
CKEDITOR.replace('blogadmin-blog_text');
CKEDITOR.config.height = 500;
CKEDITOR.config.skin = 'office2013';
</script>

```

View - administration of the blog - viewing the entry

The details of the entry shall be available together with all the fields defined in the database. In addition to the entry, buttons for editing and deleting the entry are also displayed.

File location: ../views/blogadmin/view.php

We start the PHP file, turn on the helper who generates HTML tags and a plugin that allows you to display the details of the record.

```

<?php
use yii\helpers\Html;
use yii\widgets\DetailView;

```

We define the title of a page and its location.

```

$this->title = $model->blog_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_blog_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogadmin-view">

```

Print the header, the edit button and the button to delete the entry.

```

<h1><?= Html::encode($this->title) ?></h1>

<p>

<?= Html::a(Yii::t('app', 'a_blog_edit'), ['update', 'id' => $model->blog_id], ['class' =>'btn btn-
primary']) ?>

<?= Html::a(Yii::t('app', 'a_blog_delete'), ['delete', 'id' => $model->blog_id], [
'class' =>'btn btn-danger',
'data' => [
'confirm' => Yii::t('app', 'a_delete_shure'),
'method' =>'post',
],
]) ?>
</p>

```

Use the loaded plugin to define the fields to be displayed.

```

<?= DetailView::widget([
'model' => $model,
'attributes' => [
'blog_id',
'blog_title',
'blog_text:ntext',
'blog_date',
'blog_url',
'blog_category',
'blog_hashtag',
],
]) ?>
</div>

```

Chapter 14. Administration of blog categories

Category administration model in the blog - BlogCategoryAdmin

The model is designed for administration of categories of our blog. We can add, edit and delete them. In order to implement the model we will use **ActiveRecord** class, which will make it much easier to create queries.

File location: ../models/Blogcategoryadmin.php

Start the file where you use the model name zone, then load the framework.

```
<?php
namespace app\models;
use Yii;
```

We create a class name the same as the file name, our class will be the extension for the parent class **ActiveRecord**.

```
class Blogcategoryadmin extends \yii\db\ActiveRecord
{
```

A method that returns the name of a table that is surrounded by a double brace with a percentage sign. This will add a prefix to the table name.

```
public static function tableName()
{
    return '{{blog_category}}';
}
```

The method returns the validation rules to the data by, inter alia, specifying the required fields and the type of field as a string and as the maximum length it can take.

```
public function rules()
{
    return [
        [['category_title', 'category_url'], 'required'],
        [['category_title', 'category_url'], 'string', 'max' => 150],
    ];
}
```

Use this method to delete entries from a blog whose category has been deleted. It is very important to delete all data concerning the category in order not to create blog entries which will become "bold" later. The function parameter is the identifier of the category from which the entries are to be deleted.

```
public function DeleteBlogEntries($CategoryId)
{
    Yii::$app->db->createCommand('DELETE FROM {%blog%} WHERE blog_category = :blog_category')
->bindParam(':blog_category', $CategoryId)->execute();
}
```

The method returns field names in the form of an array where the key is the name of the field and the value of its description.

```
public function attributeLabels()
{
    return [
        'category_id' => Yii::t('app', 'category_id'),
        'category_title' => Yii::t('app', 'category_title'),
        'category_url' => Yii::t('app', 'category_url'),
    ];
}
```

Administration controller for blog categories - BlogcategoryAdminController

Controller designed for administration of blog categories, i.e. their adding, editing and deleting. We will create a script which, when deleting a given category, will also delete all entries assigned to it. We will use **ActiveRecord** interface.

File location: ../controllers/BlogcategoryAdminController.php

We start the PHP file and define the namespace. We load the framework, the entry administration model, the **ActiveRecord** interface, the main class of the controller, the error class and the filters for processing the requests.

```
<?php
namespace app\controllers;
use Yii;
use app\models\Blogcategoryadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

We define a class identical to the file name, which will be the extension of the main controller.

```
class BlogcategoryAdminController extends Controller
{
```

Set the appearance of the main file to admin template.

```
public $layout = 'admin';
```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
    if($session['yii_user_root'] != "y")
    {
        return $this->redirect(['/right']);
    }
}
```

If the user is not logged in, we will take him to the login form page.

```
else
{
    return $this->redirect(['/login']);
}
```

At the end we call the method with the same name from the master controller.

```
return parent::beforeAction($action);
}
```

The method allows us to define the behavior of the application.

```
public function behaviors()
{
    return [
        'verbs' => [
```

We add that variables sent in the **delete** method can be sent only by the **POST** method.


```
'class' => VerbFilter::className(),
'actions' => [
'delete' => ['POST'],
],
],
];
}
```

The method of displaying the homepage where the categories are contained and the actions assigned to them.

```
public function actionIndex()
{
```

We save access to this section in administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_blog_category'));
```

Using **ActiveDataProvider()** we create an object in which we place a query selecting all data from a table containing categories.

```
$dataProvider = new ActiveDataProvider([
'query' => Blogcategoryadmin::find(),
]);
```

We process the view file by loading an object with the blog category data into it.

```
return $this->render('index', [
'dataProvider' => $dataProvider,
]);
}
```

The method allows you to display information about a given category, the identifier of which has been given as an argument.

```
public function actionView($id)
{
```

Save the display of category data in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_blog_one_category'));
```

We load the view and transfer all data concerning the category to it using **findModel()** class where we enter its identification number of the record as a parameter.

```
return $this->render('view', [
```

```
'model' => $this->findModel($id),
]);
}
```

This method allows you to create a new category of blog entries.

```
public function actionCreate()
{
```

Read **Blogcategoryadmin()** as a model object.

```
$model = new Blogcategoryadmin();
```

We check whether the form has been sent, all fields have been completed in accordance with the rules, and we save the data.

```
if ($model->load(Yii::$app->request->post()) && $model->save())
{
```

We save the information about creating a new category in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_create_category_blog'));
```

We redirect the user to the **view** action and pass the identifier of the newly added category.

```
return $this->redirect(['view', 'id' => $model->category_id]);
} else {
```

We load a view of the form to which we pass the model in order to generate fields and display descriptions.

```
return $this->render('create', [
    'model' => $model,
]);
}
}
```

The method designed to update the entry, the identifier of which is provided in the first parameter.

```
public function actionUpdate($id)
{
```

Using **findModel()** you can select the data of the entry by giving its identifier as an argument.

```
$model = $this->findModel($id);
```

We check whether the form has been sent, whether the fields have been validated, and we save the record.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

We save the information about the update of the record.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_update_blog_category'));
```

We move the user to the view method and enter the record identifier that we modified as an argument.

```
return $this->redirect(['view', 'id' => $model->category_id]);  
} else {
```

If the form has not been sent we read its view together with the transfer of data from the model to it.

```
return $this->render('update', [  
    'model' => $model,  
]);  
}  
}
```

A method that is used to delete a record takes as its argument the identifier of the category it wants to delete.

```
public function actionDelete($id)  
{
```

Delete the record using the **findModel()** method, in which you enter the record identifier, and then call the **delete()** method.

```
$this->findModel($id)->delete();
```

We save the information about the fact of removing the category in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_delete_blog_category'));
```

Load the **Blogcategoryadmin()** model and use the method to remove blog entries that have been added under the category that has just been removed.

```
$modelDelete = new Blogcategoryadmin();  
$modelDelete->DeleteBlogEntries($id);
```

We redirect you to the main action.

```
return $this->redirect(['index']);
```

```
}
```

A method that allows you to select a record by the identifier provided in the first parameter.

```
protected function findModel($id)
{
```

We make a query by selecting a record from the database and returning it to the model object.

```
if (($model = Blogcategoryadmin::findOne($id)) !== null) {
return $model;
} else {
```

In the event that a record does not exist, an error is read out and a message is displayed to the user.

```
throw new NotFoundException(Yii::t('app', 'page_does_not_exists'));
}
}
}
```

View - administration of blog categories - form

A form designed for editing or adding a new category of entries in our blog.

File location: ../views/blogcategoryadmin/_form.php

Start the PHP file, load the helper generating the HTML tags and the form plug-in.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<div class="blogcategoryadmin-form">
```

We start the form.

```
<?php $form = ActiveForm::begin([
'enableAjaxValidation' => false,
'enableClientValidation' => false,
]); ?>
```

We print the fields with the title and URL of the category.

```
<?= $form->field($model, 'category_title')->textInput(['maxlength' => true]) ?>
```

```
<?= $form->field($model, 'category_url')->textInput(['maxlength' => true]) ?>
```

Print the button for adding or editing.

```
<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? Yii::t('app', 'a_blog_category_add') : Yii::t('app',
'a_blog_category_edit'), ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>
```

We finish the form.

```
<?php ActiveForm::end(); ?>
</div>
```

View - administration of blog categories - creating a new category

We create a new category using the view containing the form.

File location: ../views/blogcategoryadmin/create.php

Start the PHP file, load the helper for HTML generation.

```
<?php
use yii\helpers\Html;
```

We define the title of the page and the location.

```
$this->title = Yii::t('app', 'a_blog_category_add');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_blog_category_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogcategoryadmin-create">
```

We print the page header and include a file containing the form.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
'model' => $model,
]) ?>
</div>
<script>
```

When creating a part of the URL that will be available, we must remember to remove from the title Polish characters, spaces, and other characters that cannot be found in the address.

```
$("#blogcategoryadmin-category_title").change(function () {
    str = $("#blogcategoryadmin-category_title").val();
    strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
    .replace(/ć/g, 'c').replace(/Ć/g, 'C')
    .replace(/ę/g, 'e').replace(/Ę/g, 'E')
    .replace(/ł/g, 'l').replace(/Ł/g, 'L')
    .replace(/ń/g, 'n').replace(/Ń/g, 'N')
    .replace(/ó/g, 'o').replace(/Ó/g, 'O')
    .replace(/ś/g, 's').replace(/Ś/g, 'S')
    .replace(/ż/g, 'z').replace(/Ż/g, 'Z')
    .replace(/ż/g, 'z').replace(/Ż/g, 'Z');
    strReady = strChangeLang.toLowerCase().replace(/^[a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
    $("#blogcategoryadmin-category_url").val(strReady);
}).change();
</script>
```

View - administration of blog categories - list of categories

The list of categories will show us all the departments available in the database together with the possibility of their editing, deleting or previewing complete information.

File location: ../views/blogcategoryadmin/index.php

Start the PHP file, load the HTML helper and the plugin to create the grid.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_blog_category_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogcategoryadmin-index">
```

Print the header and the button to create a new category.

```
<h1><?= Html::encode($this->title) ?></h1>
<p>
```

```
<?= Html::a(Yii::t('app', 'a_blog_category_add'), ['create'], ['class' =>'btn btn-success']) ?>
</p>
```

Use the grid to define the view of columns with record ID, title, and url.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' =>'yii\grid\SerialColumn'],
        'category_id',
        'category_title',
        'category_url',
        ['class' =>'yii\grid\ActionColumn'],
    ],
]); ?>
</div>
```

View - category administration - category update

A file containing a form that allows you to edit the category of blog entries.

File location: ../views/blogcategoryadmin/update.php

Start the PHP file and load the HTML helper.

```
<?php
use yii\helpers\Html;
```

We define the title and location of the page.

```
$this->title = Yii::t('app', 'a_blog_category_edit').': '.$model->category_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_blog_category_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = 'Update';
?>
<div class="blogcategoryadmin-update">
```

We print the title and load the form.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
<script>
```

When defining a URL, you must remove all diacritical characters, spaces, and other kinds of characters that cannot be found in the page addresses.

```
$("#blogcategoryadmin-category_title").change(function () {
    str = $("#blogcategoryadmin-category_title").val();
    strChangeLang = str.replace(/a/g, 'a').replace(/A/g, 'A')
        .replace(/ć/g, 'c').replace(/Ć/g, 'C')
        .replace(/ę/g, 'e').replace(/E/g, 'E')
        .replace(/ł/g, 'l').replace(/Ł/g, 'L')
        .replace(/ń/g, 'n').replace(/Ń/g, 'N')
        .replace(/ó/g, 'o').replace(/Ó/g, 'O')
        .replace(/ś/g, 's').replace(/Ś/g, 'S')
        .replace(/ż/g, 'z').replace(/Ż/g, 'Z')
        .replace(/ź/g, 'z').replace(/Ź/g, 'Z');
    strReady = strChangeLang.toLowerCase().replace(/[^a-z0-9\s]/gi, '').replace(/[_\s]/g, '-');
    $("#blogcategoryadmin-category_url").val(strReady);
}).change();
</script>
```

View - category administration - category browsing

View a category with all the values it contains in the database.

File location: ../views/blogcategoryadmin/view.php

We start PHP and load an HTML generation helper and a plug-in to display details about the record.

```
<?php
use yii\helpers\Html;
use yii\widgets\DetailView;
```

We define the title and location of the page.

```
$this->title = $model->category_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_blog_category_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogcategoryadmin-view">
```

We print the name of the category and the buttons for editing and deleting it.

```
<h1><?= Html::encode($this->title) ?></h1>
<p>
```



```

<?= Html::a(Yii::t('app', 'a_blog_category_edit'), ['update', 'id' => $model->category_id], ['class'
=>'btn btn-primary']) ?>
<?= Html::a(Yii::t('app', 'a_blog_category_delete'), ['delete', 'id' => $model->category_id], [
'class' =>'btn btn-danger',
'data' => [
'confirm' => Yii::t('app', 'a_delete_shure'),
'method' =>'post',
],
]) ?>
</p>

```

Using the plug-in loaded by us we print the details of the record from the database.

```

<?= DetailView::widget([
'model' => $model,
'attributes' => [
'category_id',
'category_title',
'category_url',
],
]) ?>
</div>

```

Chapter 15. Download management

Downloadable file management model - DownloadAdmin

The model is designed for file administration, which can be added, edited or removed from the table of downloadable elements. Everything will be done with the **ActiveRecord** class, which will make it much easier for us to construct queries.

File location: ../models/Downloadadmin.php

Start the file, define the namespace, and load the Yii framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class name identical to the file name that will be the **ActiveRecord** extension.

```
class Downloadadmin extends \yii\db\ActiveRecord
{
```

The method returns the name of a table surrounded by double braces and preceded by a percentage. This will add a predefined prefix to the table name.

```
public static function tableName()
{
    return '{{%download}}';
}
```

The method reverses the rules for validating the form fields when adding or editing a form. It contains fields that are required, defines the type, in this case the character string, and defines their maximum length.

```
public function rules()
{
    return [
        [['download_title', 'download_text', 'download_file', 'download_version', 'download_filesize',
        'download_license'], 'required'],
        [['download_text'], 'string'],
        [['download_title'], 'string', 'max' => 150],
        [['download_file'], 'string', 'max' => 65],
        [['download_version', 'download_filesize', 'download_license'], 'string', 'max' => 25],
    ];
}
```

The method returns an array in which the keys are field names and in which the description of the array is written.

```

public function attributeLabels()
{
    return [
        'download_id' => Yii::t('app', 'download_id'),
        'download_title' => Yii::t('app', 'download_title'),
        'download_text' => Yii::t('app', 'download_text'),
        'download_file' => Yii::t('app', 'download_file'),
        'download_version' => Yii::t('app', 'download_version'),
        'download_filesize' => Yii::t('app', 'download_filesize'),
        'download_license' => Yii::t('app', 'download_license'),
    ];
}
}

```

Controller of the administration of the download department - DownloadAdminController

Controller allows you to manage files that are in the download section. Due to its access only the administration of the service is based on **ActiveRecord**, which greatly facilitates the creation, editing and removal of content.

File location: ../controllers/DownloadAdminController.php

We start the PHP file, set up the namespace, add the model, the data generation tool, the main class of the controller, error handling and filtering the variables.

```

<?php
namespace app\controllers;
use Yii;
use app\models\Downloadadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

```

The class name is identical to the file name, this class is the parent extension.

```

class DownloadAdminController extends Controller
{

```

This variable informs that the system should load the administrator's appearance.

```

public $layout = 'admin';

```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
if($session['yii_user_root'] != "y")
{
return $this->redirect(['/right']);
}
}
}
```

If the user is not logged in, we will take him to the login form page.

```
else
{
return $this->redirect(['/login']);
}
}
```

At the end we call the method with the same name from the master controller.

```
return parent::beforeAction($action);
}
```

The method allows us to define the behavior of the application.

```
public function behaviors()
{
return [
'verbs' => [
```

We add that variables sent in the delete method can be sent only by the **POST** method.

```
'class' => VerbFilter::className(),
'actions' => [
'delete' => ['POST'],
],
],
```

```
];
}
```

Method of displaying added files and links to perform appropriate actions on them.

```
public function actionIndex()
{
```

We save the information in the administrator logs about the user's access to file management.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_download'));
```

Using the **ActiveDataProvider()** class, you can select the data based on the query included in the parameter.

```
$dataProvider = new ActiveDataProvider([
    'query' => Downloadadmin::find(),
]);
```

We generate the appearance by passing to it an object containing data about the placed files.

```
return $this->render('index', [
    'dataProvider' => $dataProvider,
]);
}
```

A method that displays the data of a file whose identifier it receives in a parameter.

```
public function actionView($id)
{
```

Save the information that the log file is displayed.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_download_one'));
```

We generate a view to which we pass the model with information about the file.

```
return $this->render('view', [
    'model' => $this->findModel($id),
]);
}
```

A method that allows you to add a new file to the table.

```
public function actionCreate()
```

```
{
```

Load the **Downloadadmin()** class into the object creating the model.

```
$model = new Downloadadmin();
```

We check whether the form has been sent, the fields have been validated and we save the record in the table.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

The administrator's logs are enriched with a message about adding a new record.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_add_file_download'));
```

We redirect the user to the method with the view of the record and its identifier.

```
return $this->redirect(['view', 'id' => $model->download_id]);  
} else {
```

If the form has not been sent, we load the view together with the model transfer in order to generate the fields of this form.

```
return $this->render('create', [  
    'model' => $model,  
]);  
}  
}
```

A method that allows the record to be updated by specifying its identifier in the parameter.

```
public function actionUpdate($id)  
{
```

Use **findModel()** to select the record.

```
$model = $this->findModel($id);
```

We check whether the form has been sent and the fields validated. Then we update the data in the table.

```
if ($model->load(Yii::$app->request->post()) && $model->save())  
{
```

Save the log for the file update.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_update_file_download'));
```

We redirect the user to the method that displays the record and its identifier is passed on.

```
return $this->redirect(['view', 'id' => $model->download_id]);  
} else {
```

We generate a form to update the record by sending it a model containing information about the fields.

```
return $this->render('update', [  
    'model' => $model,  
]);  
}  
}
```

The method deletes a record by providing its identifier in the parameter.

```
public function actionDelete($id)  
{
```

Use this method to select a record and then delete it.

```
$this->findModel($id)->delete();
```

We save the information about deleting the record.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_delete_file_download'));
```

We redirect you to the main method.

```
return $this->redirect(['index']);  
}
```

A method that selects a single record, the identifier of which is given in the parameter.

```
protected function findModel($id)  
{
```

On the basis of this parameter we select the record and transfer it to the model object.

```
if (($model = Downloadadmin::findOne($id)) !== null) {  
    return $model;  
} else {
```

If there is no record with the specified identifier, an error is displayed.

```
throw new NotFoundException(Yii::t('app', 'page_does_not_exists'));
}
}
}
```

View - download administration - form

A form designed to add or edit an existing file in the download section.

File location: ../views/downloadadmin/_form.php

We start the PHP file, load the HTML helper and the form creation plugin.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<div class="downloadadmin-form">
```

We start the form.

```
<?php $form = ActiveForm::begin([
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]); ?>
```

We print fields such as title, description, file name, version, file size and license.

```
<?= $form->field($model, 'download_title')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'download_text')->textarea(['rows' => 6]) ?>
<?= $form->field($model, 'download_file')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'download_version')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'download_filesize')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'download_license')->textInput(['maxlength' => true]) ?>
```

We print a button which, depending on where it is used, will be a button creating a new file or updating existing data.

```
<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? Yii::t('app', 'a_download_add_file') : Yii::t('app',
'a_download_update_file'), ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>
```

We finish the form.


```
<?php ActiveForm::end(); ?>
</div>
```

View - download administration - add file

The template will be responsible for the presentation of the form by means of which you will be able to add a new file to the download section.

File location: ../views/downloadadmin/create.php

We start the PHP file and load the helper that will help in generating the tags.

```
<?php
use yii\helpers\Html;
```

We define the title of the page and its location.

```
$this->title = Yii::t('app', 'a_download_add_file');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_download_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="downloadadmin-create">
```

We print the page header and load the file containing the form.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
```

View - download administration - file directory

It lists all added files with the ability to edit, delete, and go to their details.

File location: ../views/downloadadmin/index.php

We start the PHP file, read the helper designed to generate HTML and the plugin allowing to generate the grid, in which the information intended for it will be placed.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_download_main_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="downloadadmin-index">
```

We print the title and the button allowing you to add a new item.

```
<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_download_add_file'), ['create'], ['class' =>'btn btn-success']) ?>
</p>
```

Using the plugin, print the list of files available in the download area.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' =>'yii\grid\SerialColumn'],
        'download_id',
        'download_title',
        'download_file',
        'download_version',
        ['class' =>'yii\grid\ActionColumn'],
    ],
]); ?>
</div>
```

View - download administration - file update

The purpose of the file is to present a form with the data of the file that needs to be updated.

File location: ../views/downloadadmin/update.php

Start the PHP file and load the HTML helper.

```
<?php
use yii\helpers\Html;
```

We define the title and location of the page.

```
$this->title = Yii::t('app', 'a_download_update_file').': '.$model->download_title;
```

```

$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_download_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="downloadadmin-update">

```

We print the header and include the file editing form.

```

<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>

```

View - download administration - file details

View details of a single file with all data from the database.

File location: ../views/downloadadmin/view.php

Start the PHP file, load the helper for HTML generation and the plugin for displaying the details grid.

```

<?php
use yii\helpers\Html;
use yii\widgets\DetailView;

```

We define the title and location of the page.

```

$this->title = $model->download_title;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_download_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="downloadadmin-view">

```

We print the header and buttons for updating and deleting file information.

```

<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_download_add_file'), ['update', 'id' => $model->download_id], ['class' => 'btn btn-primary']) ?>
<?= Html::a(Yii::t('app', 'a_download_delete_file'), ['delete', 'id' => $model->download_id], [
    'class' => 'btn btn-danger',
    'data' => [
        'confirm' => Yii::t('app', 'a_delete_shure'),
        'method' => 'post',

```

```
],  
  ])?>  
</p>
```

Using the plugin, the details of the record are displayed from the database.

```
<?= DetailView::widget([  
  'model' => $model,  
  'attributes' => [  
    'download_id',  
    'download_title',  
    'download_text:ntext',  
    'download_file',  
    'download_version',  
    'download_filesize',  
    'download_license',  
  ],  
  ])?>  
</div>
```

Chapter 16. Management of download statistics

Download statistics model - DownloadStatAdmin

The model will be used to view statistics about downloaded files. In the admin panel we will create a special controller, which will display us statistics of downloads including both the file and the browser, the IP address and the exact time of download.

File location: ../models/Downloadstatadmin.php

We start the PHP file, define the namespace and load the framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class name identical to the file name. This class will be an extension of the **ActiveRecord** class.

```
class Downloadstatsadmin extends \yii\db\ActiveRecord
{
```

A method that returns the name of a table that is located in double braces and is preceded by a percentage. This will add the prefix we defined in the connection configuration file to the table.

```
public static function tableName()
{
    return '{{%download_stats}}';
}
```

The method of verifying whether the form sent meets the requirements. It checks whether all required fields have been sent, which data are safe, and whether the character strings have a specified maximum value.

```
public function rules()
{
    return [
        [['stat_browser', 'stat_file', 'stat_date', 'stat_ip'], 'required'],
        [['stat_date'], 'safe'],
        [['stat_browser'], 'string', 'max' => 75],
        [['stat_file'], 'string', 'max' => 65],
        [['stat_ip'], 'string', 'max' => 15],
    ];
}
```

A method that returns an array with field names as keys and field descriptions as values.

```
public function attributeLabels()
{
    return [
        'stat_id' => Yii::t('app', 'stat_id'),
        'stat_browser' => Yii::t('app', 'stat_browser'),
        'stat_file' => Yii::t('app', 'stat_file'),
        'stat_date' => Yii::t('app', 'stat_date'),
        'stat_ip' => Yii::t('app', 'stat_ip'),
    ];
}
```

DownloadstatadminController - Download statistic administrator controller

Controller designed to check download statistics. It will contain detailed information about which file was downloaded, which browser from which IP address and when. All these data can be analysed. We will use **ActiveRecord** to build the controller due to very limited access to sections and no worries about excessive server load.

File location: ../controllers/DownloadstatadminController.php

We start PHP file, define namespace, load model, data processing tool, main class of controller, library processing error information and data filter.

```
<?php
namespace app\controllers;
use Yii;
use app\models\Downloadstatsadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

We start with the definition of the class identical to the file name, which is the extension of the main controller.

```
class DownloadstatsadminController extends Controller
{
```

Variables with information about the type of template used to generate the appearance of the panel.

```
public $layout = 'admin';
```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
    if($session['yii_user_root'] != "y")
    {
        return $this->redirect(['/right']);
    }
}
```

If the user is not logged in, we will take him to the login form page.

```
else
{
    return $this->redirect(['/login']);
}
```

At the end we call the method with the same name from the master controller.

```
return parent::beforeAction($action);
}
```

The method displays statistics for downloading files.

```
public function actionIndex()
{
```

We save the information about the fact that the user browsed the statistics.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_stats_download'));
```

Using **ActiveDataProvider()** class we generate data from **Downloadstatsadmin()** model and set the sort in descending order from the record ID.

```
$dataProvider = new ActiveDataProvider([
```

```
'query' => Downloadstatsadmin::find(),
'sort'=> ['defaultOrder' => ['stat_id'=>SORT_DESC]]
]);
```

We generate the appearance by passing the object containing the data to it.

```
return $this->render('index', [
'dataProvider' => $dataProvider,
]);
}
```

View - download statistics

Displays statistics for downloading files.

File location: ../views/downloadstatadmin/index.php

Start the PHP file, load the HTML helper and the grid display plugin.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_download_stats_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogadmin-index">
```

We print the page header.

```
<h1><?= Html::encode($this->title) ?></h1>
```

We print a record grid.

```
<?= GridView::widget([
'dataProvider' => $dataProvider,
'columns' => [
['class' =>'yii\grid\SerialColumn'],
'stat_id',
'stat_browser',
'stat_file',
```



```
'stat_date',  
'stat_ip',  
],  
]); ?>  
</div>
```

Chapter 17. Contact management

Administration model contact details - ContactAdmin

The model will be used to manage the e-mail addresses that are listed in the contact table. Thanks to this model we will be able to add, edit and delete values according to the rules.

File location: ../models/Contactadmin.php

Start the file, use the model name space, and then load the framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class name identical to the file name that will be the ActiveRecord extension.

```
class Contactadmin extends \yii\db\ActiveRecord
{
```

The method returns the name of the table. It is surrounded by triple braces and is preceded by a percentage sign. This will enrich the name with a prefix.

```
public static function tableName()
{
    return '{{%contact}}';
}
```

The method responsible for validating the form defines which fields are required, how long the values of the character strings they accept can be, and that the value of the **contact_email** field is to be a validly stored e-mail address.

```
public function rules()
{
    return [
        [['contact_email', 'contact_name'], 'required'],
        [['contact_email'], 'string', 'max' => 75],
        [['contact_name'], 'string', 'max' => 55],
        [['contact_email'], 'email'],
    ];
}
```

A method that returns an array in which the keys are the field names in the form and their description in value.

```

public function attributeLabels()
{
    return [
        'contact_id' => Yii::t('app', 'contact_id'),
        'contact_email' => Yii::t('app', 'contact_email'),
        'contact_name' => Yii::t('app', 'contact_name'),
        'contact_from' => Yii::t('app', 'contact_from'),
        'contact_title' => Yii::t('app', 'contact_title'),
        'contact_body' => Yii::t('app', 'contact_body'),
        'contact_sendme' => Yii::t('app', 'contact_sendme'),
        'contact_captcha' => Yii::t('app', 'contact_captcha'),
    ];
}
}

```

Controller of contact administration - ContactAdminController

Controller designed to configure contact details on our website. It is based on ActiveRecord due to the fact that it will be used only by administrators, so there will be no problems with server load.

File location: ../controllers/ContactAdminController.php

We start PHP file, define namespace, load model, plugin allowing to generate data on page, main class of controller, plugin allowing to generate errors and data filing tool.

```

<?php
namespace app\controllers;
use Yii;
use app\models>Contactadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

```

We define the name of the controller identical to the file name that will be the extension of the main cash register.

```

class ContactAdminController extends Controller
{

```

Variable containing information about the fact that the administrative appearance should be loaded.

```

public $layout = 'admin';

```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check if the user is logged in and if there are any permissions on the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
if($session['yii_user_root'] != "y")
{
return $this->redirect(['/right']);
}
}
```

If the user is not logged in, we will take him to the login form page.

```
else
{
return $this->redirect(['/login']);
}
```

At the end we call the method with the same name from the master controller.

```
return parent::beforeAction($action);
}
```

The method allows us to define the behavior of the application.

```
public function behaviors()
{
return [
'verbs' => [
```

We add that variables sent in the **delete** method can be sent only by the **POST** method.

```
'class' => VerbFilter::className(),
'actions' => [
'delete' => ['POST'],
],
],
```

```
];
}
```

The method responsible for generating the values currently added to the contact.

```
public function actionIndex()
{
```

We save the information about browsing through the contacts in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_contacts'));
```

Using the **ActiveDataProvider()** class we create an object in which the reference to the data object we select information about addresses.

```
$dataProvider = new ActiveDataProvider([
    'query' => Contactadmin::find(),
]);
```

We return the view to which we load an object containing contact details.

```
return $this->render('index', [
    'dataProvider' => $dataProvider,
]);
}
```

The method allows you to display information about a contact based on the identifier given in the argument.

```
public function actionView($id)
{
```

We save the information about browsing the contact data in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_one_contact'));
```

We generate the appearance by passing the model object to it using the method that finds one record on the basis of the given identifier.

```
return $this->render('view', [
    'model' => $this->findModel($id),
]);
}
```

A method that allows you to create a new contact.

```
public function createAction()
{
```

Read the **Contactadmin()** class as a model object.

```
$model = new Contactadmin();
```

We check whether the form was sent, the fields validated in it and save the new record.

```
if ($model->load(Yii::$app->request->post()) && $model->save())
{
```

We save the information about adding a new record.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_create_new_contact'));
```

We redirect the user to the **view** method together with entering the identification number as a parameter in order to display the added record.

```
return $this->redirect(['view', 'id' => $model->contact_id]);
} else {
```

When the form has not been sent yet, we load its appearance to which we transfer a model containing data on fields and their values.

```
return $this->render('create', [
'model' => $model,
]);
}
}
```

The method allows the record to be updated on the basis of its identifier provided in the parameter.

```
public function actionUpdate($id)
{
```

The **findModel()** method reads the data of the current record by passing its identifier.

```
$model = $this->findModel($id);
```

We check if the form was sent, if the fields were properly validated and update the record.

```
if ($model->load(Yii::$app->request->post()) && $model->save())
{
```

We save the information about the update in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_updated_contact'));
```

We redirect the user to **view** action together with adding the parameter which is the record identifier.

```
return $this->redirect(['view', 'id' => $model->contact_id]);  
} else {
```

If the form has not been sent, we generate the appearance together with the transfer of the model to it.

```
return $this->render('update', [  
    'model' => $model,  
]);  
}  
}
```

A method that allows you to delete a record, the identifier of which is provided in the parameter.

```
public function actionDelete($id)  
{
```

Using the **findModel()** method, the record identifier is passed to it in order to select it, and then deleted using the **delete()** method.

```
$this->findModel($id)->delete();
```

We save the information about deleting the record in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_deleted_contact'));
```

We move you to the main method.

```
return $this->redirect(['index']);  
}
```

A method that selects a record by its identifier.

```
protected function findModel($id)  
{
```

Select the record on the basis of the identifier transferred to the method, and then return it in the variable model.

```
if (($model = Contactadmin::findOne($id)) !== null) {
```

```
return $model;
} else {
```

In case the given record is not found, we display information about the error to the users.

```
throw new NotFoundHttpException(Yii::t('app', 'page_does_not_exists'));
}
}
}
```

View - contacts administration - form

Contacts available on the website for the user can be freely modified in the admin panel. The file contains the form used to add and edit e-mail addresses.

File location: ../views/contactadmin/_form.php

Start the PHP file, load the HTML generation helper and the active forms plugin.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
?>
<div class="contactadmin-form">
```

We start the form.

```
<?php $form = ActiveForm::begin([
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]); ?>
```

We print the fields for e-mail and contact description.

```
<?= $form->field($model, 'contact_email')->textInput(['maxlength' => true]) ?>
<?= $form->field($model, 'contact_name')->textInput(['maxlength' => true]) ?>
```

We print the button for sending the form.

```
<div class="form-group">
<?= Html::submitButton($model->isNewRecord ? Yii::t('app', 'a_create_user_button') : Yii::t('app',
'a_update_user_button'), ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>
```

We are finishing the form.


```
<?php ActiveForm::end(); ?>
</div>
```

View - contact administration - create a new contact

Adding a new contact will be done using the form defined in the file.

File location: ../views/contactadmin/create.php

We start by defining PHP and loading the HTML helper.

```
<?php
use yii\helpers\Html;
```

We define the title of a page and its location.

```
$this->title = Yii::t('app', 'a_create_contact_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_contact_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="contactadmin-create">
```

We print the page header and include the form.

```
<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>
```

View - contact administration - list of contacts

All your contacts are on a page with links to edit, delete, or view details next to them.

File location: ../views/contactadmin/index.php

We start the PHP file, load the helper for HTML generation and the plugin that allows you to generate the grid.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title of the page and its location.

```
$this->title = Yii::t('app', 'a_contact_main_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="contactadmin-index">
```

We print the page header and the transfer button to the form where you can add a new person.

```
<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_add_contact_button'), ['create'], ['class' => 'btn btn-success']) ?>
</p>
```

Using the plugin, you can generate a preview of the following columns: record ID, e-mail address, and contact name. In addition to the data columns, there will also be buttons for editing the data, allowing you to delete them and view the details.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'contact_id',
        'contact_email:email',
        'contact_name',
        ['class' => 'yii\grid\ActionColumn'],
    ],
]); ?>
</div>
```

View - contact administration - contact update

A special page where we can edit the data of a single contact from the database.

File location: ../views/contactadmin/update.php

Start the PHP file and load the helper to generate HTML tags.

```
<?php
use yii\helpers\Html;
```

We define the title and position of the page.

```
$this->title = Yii::t('app', 'a_update_contact_header').': '.$model->contact_email;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
```

```

$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_contact_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="contactadmin-update">

```

We print the header and load the form.

```

<h1><?= Html::encode($this->title) ?></h1>
<?= $this->render('_form', [
    'model' => $model,
]) ?>
</div>

```

View - contact management - contact details

View contact details with all columns defined in the database.

File location: ../views/contactadmin/view.php

We start the PHP file, read the helper to generate HTML stamps and the plugin that allows you to view the details of the record.

```

<?php
use yii\helpers\Html;
use yii\widgets\DetailView;

```

We define the title of a page and its location.

```

$this->title = $model->contact_email;
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_contact_main_header'), 'url' => ['index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="contactadmin-view">

```

Print the title of the page, the button to update the record, and the button to delete the record.

```

<h1><?= Html::encode($this->title) ?></h1>
<p>
<?= Html::a(Yii::t('app', 'a_update_contact_header'), ['update', 'id' => $model->contact_id], ['class'
=>'btn btn-primary']) ?>
<?= Html::a(Yii::t('app', 'a_delete_contact_button'), ['delete', 'id' => $model->contact_id], [
    'class' =>'btn btn-danger',
    'data' => [
        'confirm' => Yii::t('app', 'a_delete_shure'),
        'method' =>'post',

```

```
],  
  }) ?>  
</p>
```

The plugin is used to define all data from the record to be displayed.

```
<?= DetailView::widget([  
  'model' => $model,  
  'attributes' => [  
    'contact_id',  
    'contact_email:email',  
    'contact_name',  
  ],  
  ]) ?>  
</div>
```

Chapter 18. Password recovery management

Model displaying attempts to recover the password - PasswordAdmin

A model that allows the site administrator to view who, when, and from what IP address asked for his account password to be restored. Currently, this is very important in order to try to access your account for an unauthorized user. Thanks to this functionality, we will be able to quickly check whether there has been any access by unauthorized persons.

File location: ../models/Passwordadmin.php

We start the file, set the namespace and load the framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class identical to the filename that will be the **ActiveRecord** claim.

```
class Passwordadmin extends \yii\db\ActiveRecord
{
```

A method that returns the name of a table that is surrounded by double braces and is preceded by a percentage. This will add the prefix defined by us in the connection configuration file.

```
public static function tableName()
{
    return '{{%password}}';
}
```

A method that includes rules for the validation of fields. It defines which fields are required, which are to be a number, safe or a string of characters, and how many of these fields can have maximum characters.

```
public function rules()
{
    return [
        [['password_user_id', 'password_hash1', 'password_hash2', 'password_time', 'password_time_used', 'password_ip', 'password_used'], 'required'],
        [['password_user_id'], 'integer'],
        [['password_time', 'password_time_used'], 'safe'],
        [['password_hash1', 'password_hash2'], 'string', 'max' => 20],
```

```

[['password_ip'], 'string', 'max' => 15],
[['password_used'], 'string', 'max' => 1],
];
}

```

A method that returns an array in which the key is the name of a field and the value of its description.

```

public function attributeLabels()
{
return [
'password_id' => Yii::t('app', 'password_id'),
'password_user_id' => Yii::t('app', 'password_user_id'),
'password_hash1' => Yii::t('app', 'password_hash1'),
'password_hash2' => Yii::t('app', 'password_hash2'),
'password_time' => Yii::t('app', 'password_time'),
'password_time_used' => Yii::t('app', 'password_time_used'),
'password_ip' => Yii::t('app', 'password_ip'),
'password_used' => Yii::t('app', 'password_used'),
];
}
}

```

Administrator controller with password reminder logs - PasswordAdminController

Controller designed to browse logs for requests from users for forgotten passwords. This will allow you to identify very quickly whether a person who does not have access to your account can attempt to do so.

File location: ../controllers/PasswordAdminController.php

We start the file with the name zone definition, load the frame, model, plug-in for data generation, the main class of the controller, error handling and data filtering.

```

<?php
namespace app\controllers;
use Yii;
use app\models>Passwordadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

```

We define a class identical to the file wind, which will be the extension of the main controller.

```

class PasswordAdminController extends Controller
{

```

The variable with the page template is set to the administrative appearance.

```
public $layout = 'admin';
```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
    if($session['yii_user_root'] != "y")
    {
        return $this->redirect(['/right']);
    }
}
```

If the user is not logged in, we will take him to the login form page.

```
else
{
    return $this->redirect(['/login']);
}
```

At the end we call the method with the same name from the master controller.

```
return parent::beforeAction($action);
}
```

The method displays data concerning the action of reminding the user of the password and its change by the user.

```
public function actionIndex()
{
```

We save the information about access to the section in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_recomm_password'));
```

Using **ActiveDataProvider()** we create an object in which we search for a change of password according to the limit and sort the records according to the **password_id** field in descending order.

```
$dataProvider = new ActiveDataProvider([
    'query' => Passwordadmin::find(),
    'sort'=> ['defaultOrder' => ['password_id'=>SORT_DESC]]
]);
```

We generate a view by passing on data concerning the change of passwords in the object.

```
return $this->render('index', [
    'dataProvider' => $dataProvider,
]);
}
```

View - account password reminder logs

You can use the password reminder option. The administrator can see when the password was reminded and what was the result of the operation.

File location: ../views/passwordadmin/index.php

Start the PHP file, load the HTML generation helper and the grid generation plugin.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title and location.

```
$this->title = Yii::t('app', 'a_password_remind_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogadmin-index">
```

We print the header.

```
<h1><?= Html::encode($this->title) ?></h1>
```

Using the grid, we print the details about the attempted password change where the columns contain the following data: record ID, user ID, passwords, time of generation and use, IP address and information about whether the link was clicked and the password changed.


```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'password_id',
        'password_user_id',
        'password_hash1',
        'password_hash2',
        'password_time',
        'password_time_used',
        'password_ip',
        'password_used',
    ],
]); ?>
</div>
```

Chapter 19. Page setup

Page configuration model - ConfigAdmin

The model will be used to set the data concerning the page configuration, such as: title, e-mail address or keywords and description. It will be built on the basis of **ActiveRecord** class.

File location: ../models/Configadmin.php

We start the file, define the namespace and load the framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class name identical to the file name that will be the **ActiveRecord** extension.

```
class Configadmin extends \yii\db\ActiveRecord
{
```

The class variables that we use to add values to the form are defined as public so that they can be used in controllers and views.

```
public $config_title;
public $config_description;
public $config_keywords;
public $config_rootemail;
public $config_foot;
```

The method returns the name of the table in the database. The name is surrounded by a double brace together with a percentage mark. All this is done in order to automatically add a prefix from a file with a defined database configuration to our table.

```
public static function tableName()
{
    return '{{%config}}';
}
```

The method that validates the field. It has defined fields that are required and information that **config_rootemail** field was correctly saved e-mail address.

```
public function rules()
{
    return [
```

```

[['config_title', 'config_description', 'config_keywords', 'config_rootemail', 'config_foot'],
'required'],

[['config_rootemail'], 'email'],

];

}

```

The method of saving the data sent from the form to the table in the database. It contains a number of queries, to which we load the contents of the class variables using the **bindParam()** method.

```

public function SaveUserData()
{
Yii::$app->db->createCommand('UPDATE {%config} SET config_value = :config_value WHERE config_name =
"title")->bindParam(':config_value', $this->config_title)->execute();

Yii::$app->db->createCommand('UPDATE {%config} SET config_value = :config_value WHERE config_name =
"description")->bindParam(':config_value', $this->config_description)->execute();

Yii::$app->db->createCommand('UPDATE {%config} SET config_value = :config_value WHERE config_name =
"keywords")->bindParam(':config_value', $this->config_keywords)->execute();

Yii::$app->db->createCommand('UPDATE {%config} SET config_value = :config_value WHERE config_name =
"rootemail")->bindParam(':config_value', $this->config_rootemail)->execute();

Yii::$app->db->createCommand('UPDATE {%config} SET config_value = :config_value WHERE config_name =
"foot")->bindParam(':config_value', $this->config_foot)->execute();

}

```

The method takes values from the configuration table and writes them down inside the class variables by means of a loop in which the condition **if** is used to check whether the name of the variable is equal to the defined value.

```

public function DataInsert()
{
$queryDataIs = Yii::$app->db->createCommand('SELECT * FROM {%config}')
->queryAll();

for($c=0;$c<count($queryDataIs);$c++)
{

```

Enter the title of the page into the variable.

```

if($queryDataIs[$c]['config_name'] == 'title')
{
$this->config_title = $queryDataIs[$c]['config_value'];
}

```

Enter the description of the page into the variable.

```

if($queryDataIs[$c]['config_name'] == 'description')
{
$this->config_description = $queryDataIs[$c]['config_value'];
}

```

Enter the keywords in the variable.

```

if($QueryDataIs[$c]['config_name'] == 'keywords')
{
$this->config_keywords = $QueryDataIs[$c]['config_value'];
}

```

Enter the e-mail address of the administrator into the variable.

```

if($QueryDataIs[$c]['config_name'] == 'rootemail')
{
$this->config_rootemail = $QueryDataIs[$c]['config_value'];
}

```

Enter the contents of the page footer into the variable value.

```

if($QueryDataIs[$c]['config_name'] == 'foot')
{
$this->config_foot = $QueryDataIs[$c]['config_value'];
}
}
}

```

A method that retrieves the configuration of a page, which is transformed into an array by means of a loop, and then returned.

```

public function GetConfig()
{
$queryDataIs = Yii::$app->db->createCommand('SELECT * FROM {%config}%')
->queryAll();

for($c=0;$c<count($QueryDataIs);$c++)
{
$config[$QueryDataIs[$c]['config_name']] = $QueryDataIs[$c]['config_value'];
}

return $Config;
}

```

A method that returns an array in which the keys are field names and the values are their descriptions.

```

public function attributeLabels()
{
return [
'config_id' => Yii::t('app', 'config_id'),
'config_name' => Yii::t('app', 'config_name'),
'config_value' => Yii::t('app', 'config_value'),
'config_title' => Yii::t('app', 'config_title'),

```

```

'config_description' => Yii::t('app', 'config_description'),
'config_keywords' => Yii::t('app', 'config_keywords'),
'config_rootemail' => Yii::t('app', 'config_rootemail'),
'config_foot' => Yii::t('app', 'config_foot'),

];
}
}

```

Site configuration controller - ConfigadminController

Controller designed to configure our site, which we can read and save together with the changes made.

File location: ../controllers/ConfigadminController.php

We start the PHP file with the definition of the namespace, load the Yii framework, add the model to the settings configuration, load the data generation plugin, load the error management plugin and the data filter.

```

<?php
namespace app\controllers;
use Yii;
use app\models\Configadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;

```

We define a class name that is identical to the file name. This class is an extension of the **Controller**.

```

class ConfigadminController extends Controller
{

```

This variable loads the appearance of the system administrator's page.

```

public $layout = 'admin';

```

The **beforeAction()** method is executed before all others so you can define in it all actions that you want to perform before loading the method to which you are currently referring. We will use this to validate your permissions.

```

public function beforeAction($action)
{

```

We retrieve session configuration data.

```

$session = Yii::$app->session;

```

We check whether the user is logged in and whether his rights are set to administrative. If they were not transferred to the corresponding no-permission message.

```
if($session['yii_user_id'] != "")
{
if($session['yii_user_root'] != "y")
{
return $this->redirect(['/right']);
}
}
else
{
```

If the user was not logged in, he will be automatically redirected to the login window.

```
return $this->redirect(['/login']);
}
```

At the very end we need to run a method with exactly the same name, only that from a superior class.

```
return parent::beforeAction($action);
}
```

The method is designed to generate a configuration form and to validate the data entered by us.

```
public function actionIndex()
{
```

Load the **Configadmin()** class as a model object.

```
$model = new Configadmin();
```

Set the update status to **false**.

```
$ConfigUpdated = false;
```

We check whether the form has been sent and whether the fields have been validated correctly.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())
{
```

We save the administrator's log stating that the configuration has been updated.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_update_configuration'));
```

Save the current configuration to the table.

```
$model->SaveUserData();
```

Set the value of the variable to **true**, informing about the correct recording.

```
$ConfigUpdated = true;
}
else
{
```

If we have not sent the form, it means that you have to fill it in with data. We create this by using our model, which will enter the relevant information into the fields.

```
$model->DataInsert();
}
```

We generate the look and feel with the transfer of the model and information about the updated fields.

```
return $this->render('config', [
    'model' => $model, 'ConfigUpdated' => $ConfigUpdated
]);
}
```

View - page configuration

Elements such as the title of a page, its description or keywords can be configured using a special form.

File location: ../views/configadmin/config.php

We start the PHP file, load the HTML helper and the form generator.

```
<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;
```

We define the title and location of the page.

```
$this->title = Yii::t('app', 'a_title_config');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="useradmin-create">
```

We print the title of the page.

```
<h1><?= Html::encode($this->title) ?></h1>
<?php
```

When the update is completed, we print the information.

```
if($ConfigUpdated)
{
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'a_config_options_updated').'</div>';
}
?>
<div class="useradmin-form">
```

We start the form.

```
<?php $form = ActiveForm::begin([
'enableAjaxValidation' => false,
'enableClientValidation' => false,
]); ?>
```

We define fields such as: title, page description, keywords, administrator's e-mail address and footer.

```
<?= $form->field($model, 'config_title'); ?>
<?= $form->field($model, 'config_description'); ?>
<?= $form->field($model, 'config_keywords'); ?>
<?= $form->field($model, 'config_rootemail'); ?>
<?= $form->field($model, 'config_foot'); ?>
```

We print the button for sending the form.

```
<div class="form-group">
<?= Html::submitButton(Yii::t('app', 'a_config_update_button'), ['class' => $model->isNewRecord ? 'btn btn-success' : 'btn btn-primary']) ?>
</div>
```

We finish the form.

```
<?php ActiveForm::end(); ?>
</div>
</div>
```


Chapter 20. Managing the page menu

User menu model - LeftMenuAdmin

The model allows you to administer the left menu, which will be located next to the content presented on our website. Among other things, it allows you to add, delete and store menu items in any order you like.

File location: ../models/Leftmenuadmin.php

Start the PHP file, define the namespace, and load the Yii framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class which is identical to the filename and is an extension of **ActiveRecord** class.

```
class Leftmenuadmin extends \yii\db\ActiveRecord
{
```

Variables inside the class will be identical to the names of fields in the table, additionally we will make them public so that you can use them in controllers and views.

```
public $menu_id;
public $menu_title;
public $menu_poz;
public $menu_sub;
public $menu_login;
public $menu_what;
public $menu_content_id;
public $menu_extra;
```

A method that returns the name of a table that is surrounded by double braces and is preceded by a percentage. This will automatically add the prefix you defined in the database configuration file.

```
public static function tableName()
{
    return '{{%menu}}';
}
```

The method that defines the rules. When adding a record, these fields are required.

```
public function rules()
```

```

{
return [
[['menu_title', 'menu_sub', 'menu_login', 'menu_what', 'menu_content_id'], 'required'],
];
}

```

The **SaveData()** method adds new items to the table containing the system menu.

```

public function SaveData()
{

```

Check if the variables related to the menu items and to the fact that the menu is subordinate have been filled in. If the value is empty, enter zero in it.

```

if($this->menu_poz == ""){$this->menu_poz = 0;}
if($this->menu_sub == ""){$this->menu_sub = 0;}

```

We create and execute a query for our table in order to place a new item in the menu. We load the variable values into the query using **bindParam()** method and execute them without returning the result using **execute()** method.

```

$queryData = Yii::$app->db->createCommand('INSERT INTO {%menu} (menu_title, menu_poz, menu_sub,
menu_login, menu_what, menu_content_id, menu_extra)
values
(:menu_title,:menu_poz, :menu_sub, :menu_login, :menu_what, :menu_content_id, :menu_extra)
')
->bindParam(':menu_title', $this->menu_title)
->bindParam(':menu_poz', $this->menu_poz)
->bindParam(':menu_sub', $this->menu_sub)
->bindParam(':menu_login', $this->menu_login)
->bindParam(':menu_what', $this->menu_what)
->bindParam(':menu_content_id', $this->menu_content_id)
->bindParam(':menu_extra', $this->menu_extra)
->execute();
}

```

The method by which a menu item can be deleted uses the record identifier as a parameter.

```

public function DeleteMenu($MenuId)
{

```

Create a query to remove the main menu item.

```

$queryData = Yii::$app->db->createCommand('DELETE FROM {%menu} WHERE menu_id = :menu_id')
->bindParam(':menu_id', $MenuId)
->execute();

```

We create a query that removes all menu items from the submenu.

```

$queryData = Yii::$app->db->createCommand('DELETE FROM {%menu} WHERE menu_sub = :menu_sub')
->bindParam(':menu_sub', $MenuId)
->execute();
}

```

The method is designed to set menu items according to the transmitted data. It takes two parameters: the first **\$Key** is the identification number of a menu item, and the second **\$Value** is the number of a menu item.

```

public function SetPozition($Key,$Value)
{
$queryData = Yii::$app->db->createCommand('UPDATE {%menu} SET menu_poz = :menu_poz WHERE menu_id = :menu_id')
->bindParam(':menu_poz', $Value)
->bindParam(':menu_id', $Key)
->execute();
}

```

A method that returns a table in which the key is the name of the field and the value of its description.

```

public function attributeLabels()
{
return [
'menu_id' => Yii::t('app', 'menu_id'),
'menu_title' => Yii::t('app', 'menu_title'),
'menu_what' => Yii::t('app', 'menu_what'),
'menu_content_id' => Yii::t('app', 'menu_content_id'),
'menu_poz' => Yii::t('app', 'menu_poz'),
'menu_sub' => Yii::t('app', 'menu_sub'),
'menu_login' => Yii::t('app', 'menu_login'),
'menu_extra' => Yii::t('app', 'menu_extra')
];
}
}

```

Controller of the administration of the page menu - LeftmenuadminController

Controller designed to manage the page menu on the left side. It will allow you to add and remove elements, create branches from them and specify the user status for which they should be visible. We will use **ActiveRecord** interface because only administrators have access to this section of the page.

File location: ../controllers/LeftmenuadminController.php

We start the file, define the namespace, load the framework, menu data models, pages, blog categories, god, articles, data generation plugin, controller main class, error handling and data filtering.

```
<?php
namespace app\controllers;
use Yii;
use app\models\Leftmenuadmin;
use app\models\Pageadmin;
use app\models\Blogcategoryadmin;
use app\models\Blogadmin;
use app\models\Articleadmin;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

We define a class name identical to the file name that will be the extension of the main controller.

```
class LeftmenuadminController extends Controller
{
```

In the variable that defines the appearance of the page we define the administrator's template.

```
public $layout = 'admin';
```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
{
    if($session['yii_user_root'] != "y")
    {
        return $this->redirect(['/right']);
    }
}
```

If the user is not logged in, we will take him to the login form page.

```

else
{
return $this->redirect(['/login']);
}

```

At the end we call the method with the same name from the master controller.

```

return parent::beforeAction($action);
}

```

A method that shows all menu items, adds a new item and removes an existing one.

```

public function actionIndex()
{

```

Load the **Leftmenuadmin()** class into the model object.

```

$model = new Leftmenuadmin();

```

We define the variables that inform about adding a new item and about changing the menu items to **false**.

```

$WasAdded = false;
$WasPoz = false;

```

We check if the form for changing menu items has been sent.

```

if(Yii::$app->request->post('setpozition') == 'yes')
{

```

We save the information in the admin panel about the change of position.

```

Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'com_set_menu_poz'));

```

Using a loop, we retrieve subsequent elements of the array as a key and its value, which are placed in the method allowing to add values to the table.

```

foreach(Yii::$app->request->post('poz') AS $Key=>$Value)
{
$model->SetPozition($Key,$Value);
}

```

We set the information about changing the position to **true**.

```

$WasPoz = true;
}

```

Check if the deletion action for the menu item has been sent.

```
if(Yii::$app->request->get('delete') != "")  
{
```

We save the information about removing an item from the menu to the admin panel.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'com_del_menu_element'));
```

We delete a menu item by the method in which we pass its identifier.

```
$model->DeleteMenu(Yii::$app->request->get('delete'));  
}
```

We check whether the form has been sent and whether the fields contained in it have been validated in a correct way.

```
if ($model->load(Yii::$app->request->post()) && $model->validate())  
{
```

When you choose to add a page, select it from the table.

```
if($model->menu_what == 'pageone')  
{  
    $Pages = Pageadmin::find()->where('page_id = :page_id', ['page_id' => $model->menu_content_id])-  
>asArray()->one();  
    $model->menu_extra = $Pages['page_url'];  
}
```

If you have selected a category for a blog, please select it from the table.

```
if($model->menu_what == 'blogcategory')  
{  
    $Blogcategory = Blogcategoryadmin::find()->where('category_id = :category_id', ['category_id' => $model->  
menu_content_id])->asArray()->one();  
    $model->menu_extra = $Blogcategory['category_url'];  
}
```

Select the data from the table about a single blog entry.

```
if($model->menu_what == 'blogone')  
{  
    $Blog = Blogadmin::find()->where('blog_id = :blog_id', ['blog_id' => $model->menu_content_id])->asArray()-  
>one();  
    $model->menu_extra = $Blog['blog_url'];  
}
```

We select an item on the basis of the submitted data, which contained its identifier.

```
if($model->menu_what == 'articleone')
{
$Article = Articleadmin::find()->where('article_id = :article_id', ['article_id' => $model->menu_content_id])->asArray()->one();
$model->menu_extra = $Article['article_url'];
}
```

Save the information about adding a new item to the menu in the administrator logs.

```
Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'com_add_menu_element'));
```

If the **menu_extra** field is empty, set an empty string in it, so that it contains a value and passes the validation.

```
if($model->menu_extra == null)
{
$model->menu_extra = '';
}
```

Save the new menu item.

```
$model->SaveData();
```

We clean the fields in the model from the data.

```
$model->menu_id = "";
$model->menu_title = "";
$model->menu_poz = "";
$model->menu_sub = "";
$model->menu_login = "";
$model->menu_what = "";
$model->menu_content_id = "";
$model->menu_extra = "";
```

For the variable informing about adding a new element enter **true** value.

```
$WasAdded = true;
}
```

Select the main pages from the menu.

```
$MainAllPages = Leftmenuadmin::find()
->where(['menu_sub' => 0])
```

```

->asArray()
->orderBy('menu_poz')
->all();

```

Select all subpages from the menu.

```

$MainSubPages = Leftmenuadmin::find()
->where('menu_sub != 0')
->asArray()
->orderBy('menu_poz')
->all();

```

Select all sub-elements from the menu.

```

$MainPages = Leftmenuadmin::find()
->where(['menu_sub' => 0])
->asArray()
->orderBy('menu_poz')
->all();

```

We define the array.

```

$MainsTable = array();

```

Use the loop to enter the menu items into the array.

```

for($p=0;$p<count($MainPages);$p++)
{
    $MainsTable[$MainPages[$p]['menu_id']] = $MainPages[$p]['menu_title'];
}

```

We select the pages available in the system.

```

$Pages = Pageadmin::find()
->asArray()
->orderBy('page_id')
->all();

```

We define the array and create it with the help of a loop from the data coming from the query.

```

$PageTable = array();

for($p=0;$p<count($Pages);$p++)
{
    $PageTable[$Pages[$p]['page_id']] = $Pages[$p]['page_title'];
}

```



```
}
```

We select all categories of the blog.

```
$Categories = Blogcategoryadmin::find()  
->orderBy('category_id')  
->asArray()  
->all();
```

We create tables and fill them with data from applying for blog categories.

```
$CategoryTable = array();  
  
for($p=0;$p<count($Categories);$p++)  
{  
$CategoryTable[$Categories[$p]['category_id']] = $Categories[$p]['category_title'];  
}
```

We select all individual entries in the blog.

```
$Entries = Blogadmin::find()  
->orderBy('blog_id')  
->asArray()  
->all();
```

We create tables and enter into them the data received from the query for entries contained in the blog.

```
$EntriesTable = array();  
  
for($p=0;$p<count($Entries);$p++)  
{  
$EntriesTable[$Entries[$p]['blog_id']] = $Entries[$p]['blog_title'];  
}
```

We will process your request for articles.

```
$Article = Articleadmin::find()  
->orderBy('article_id')  
->asArray()  
->all();
```

We create an array in which we enter the received data from the request for articles.

```
$ArticlesTable = array();
```

```

for($p=0;$p<count($Article);$p++)
{
$ArticlesTable[$Article[$p]['article_id']] = $Article[$p]['article_title'];
}

```

We read out the layout of the page to which we are transferring the model, all the tables we created and the variables containing the information.

```

return $this->render('menu', ['model' => $model, 'ArticlesTable' => $ArticlesTable, 'WasPoz' => $WasPoz,
'MainSubPages' => $MainSubPages, 'MainAllPages' => $MainAllPages, 'WasAdded' => $WasAdded, 'PageTable' =>
$PageTable, 'CategoryTable' => $CategoryTable, 'EntriesTable' => $EntriesTable, 'MainsTable' =>
$MainsTable]);
}
}

```

View - administration of the system menu

The file allows us to generate menus that appear on the left side of the website.

File location: ../views/leftmenuadmin/menu.php

We start the PHP file, load the helper to generate HTML and the plugin to generate forms.

```

<?php
use yii\helpers\Html;
use yii\widgets\ActiveForm;

```

We define the title and the current location.

```

$this->title = Yii::t('app', 'a_left_menu');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>

```

We print the title.

```

<h1><?= $this->title; ?></h1>
<?php

```

Displays information if you have placed menu items in a menu item.

```

if($WasPoz)
{
echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'a_pozition_set').'</div>';
}

```

We start the form.

```
$form = ActiveForm::begin([
    'id' => 'menu-form',
    'action' => Yii::$app->params['pageUrl'].'leftmenuadmin/index',
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
]);
```

We define an array with the contents of the main menu items.

```
$ItemContentList2['main'] = Yii::t('app', 'a_menu_main');
$ItemContentList2['none'] = Yii::t('app', 'a_menu_main');
$ItemContentList2['login'] = Yii::t('app', 'a_menu_login');
$ItemContentList2['logout'] = Yii::t('app', 'a_menu_logout');
$ItemContentList2['changepassword'] = Yii::t('app', 'a_menu_change_password');
$ItemContentList2['profil'] = Yii::t('app', 'a_menu_profil');
$ItemContentList2['register'] = Yii::t('app', 'a_menu_register');
$ItemContentList2['password'] = Yii::t('app', 'a_menu_pass_reminder');
$ItemContentList2['page'] = Yii::t('app', 'a_menu_page');
$ItemContentList2['pageone'] = Yii::t('app', 'a_menu_page_one');
$ItemContentList2['download'] = Yii::t('app', 'a_menu_download');
$ItemContentList2['blog'] = Yii::t('app', 'a_menu_blog');
$ItemContentList2['blogcategory'] = Yii::t('app', 'a_menu_blog_category');
$ItemContentList2['blogone'] = Yii::t('app', 'a_menu_blog_one');
$ItemContentList2['article'] = Yii::t('app', 'a_menu_articles');
$ItemContentList2['articleone'] = Yii::t('app', 'a_menu_article_one');
$ItemContentList2['contact'] = Yii::t('app', 'a_menu_contact');

$TableMakeMenu = array();
$HowManyElements = count($MainAllPages);
```

We process the array depending on the information whether the menu item requires logging in or not.

```
for($m=0;$m<count($MainAllPages);$m++)
{
    if($MainAllPages[$m]['menu_login'] == 'y')
    {
        $NeedLogin = Yii::t('app', 'comm_yes');
    }
    else
    {
        $NeedLogin = Yii::t('app', 'comm_no');
    }
}
```

Set the menu item type to the variable.

```
$GoTo = $ItemContentList2[$MainAllPages[$m]['menu_what']];
```

Create a field to move the item to the correct position.

```
echo '<div class="row" style="margin-bottom: 5px;">

<div class="col-md-2"><select name="poz['.$MainAllPages[$m]['menu_id'].']" class="form-control"
style="width: 90px;">;
```

Check how many items there are in the menu and print the number of items in the drop-down list, selecting the current item at the same time.

```
for($p=0;$p<$HowManyElements;$p++)
{
$Selected = null;
if($p == $m)
{
$Selected = 'selected="selected"';
}
echo '<option value="'.($p+1).'"'.$Selected.'>'.($p+1).</option>';
}

echo '</select></div>
```

Print the position with the link to delete the position.

```
<div class="col-md-4">'.$MainAllPages[$m]['menu_title'].</div>
<div class="col-md-2">'.$NeedLogin.</div>
<div class="col-md-2">'.$GoTo.</div>
<div class="col-md-2">'<Html::a(Yii::t('app', 'com_delete_button'),
['/leftmenuadmin/index?delete='.$MainAllPages[$m]['menu_id']], ['data' => ['confirm' => Yii::t('app',
'a_delete_shure')]))</div>
</div>;
```

Count the items in the current menu item.

```
$HowManyElementsSub = count($MainSubPages);
$IsSubOdThisMenu = 0;
```

Check if the menu contains added sub-menu elements.

```
for($sub=0;$sub<count($MainSubPages);$sub++)
{
if($MainSubPages[$sub]['menu_sub'] == $MainAllPages[$m]['menu_id'])
{
$IsSubOdThisMenu++;
}
}
```

We go through the elements of the sub-menu.

```
for($sub=0;$sub<count($MainSubPages);$sub++)  
{
```

Check whether the menu item belongs to the current menu.

```
if($MainSubPages[$sub]['menu_sub'] == $MainAllPages[$m]['menu_id'])  
{
```

If it requires logging in, a corresponding message is displayed.

```
if($MainSubPages[$sub]['menu_login'] == 'y')  
{  
$NeedLogin = Yii::t('app', 'comm_yes');  
}  
else  
{  
$NeedLogin = Yii::t('app', 'comm_no');  
}  
  
$GoTo = $ItemContentList2[$MainSubPages[$sub]['menu_what']];
```

We print a list to set the sub-menu position.

```
echo '<div class="row" style="margin-bottom: 5px;">  
<div class="col-md-2" style="padding-left: 30px;"><select name="poz['.$MainSubPages[$sub]['menu_id'].']" class="form-control" style="width: 90px;">';
```

We execute the loop and display an appropriate number of fields containing the position.

```
for($p=0;$p<$IsSubOdThisMenu;$p++)  
{  
$Selected = null;  
if($p == $sub)  
{  
$Selected = 'selected="selected"';  
}  
echo '<option value="'.($p+1).'"'.$Selected.'>'.($p+1).'</option>';  
}
```

We print the sub-menu element together with a button enabling its removal.

```
echo '</select></div>  
<div class="col-md-4" style="padding-left: 30px;">'.$MainSubPages[$sub]['menu_title'].'</div>
```

```
<div class="col-md-2">'. $NeedLogin.'</div>

<div class="col-md-2">'. $GoTo.'</div>

<div class="col-md-2">'. Html::a(Yii::t('app', 'com_delete_button'),
['/leftmenuadmin/index?delete='.$MainSubPages[$sub]['menu_id']], ['data' => ['confirm' => Yii::t('app',
'a_delete_shure')]]).'</div>

</div>;

}

}

}
```

Print a button that allows you to put the menu items in the correct order.

```
echo '<div class="row">

<div class="col-md-2"><br /><input type="hidden" name="setpoztion" value="yes" /><input type="submit"
value="'.Yii::t('app', 'a_set')." class="btn btn-primary" /></div>

<div class="col-md-4"></div>

<div class="col-md-2"></div>

<div class="col-md-2"></div>

<div class="col-md-2"></div>

</div>';
```

We finish the form.

```
ActiveForm::end();
?>
<?php
```

We print the header of the form that allows you to add a new item.

```
echo '<h2>.Yii::t('app', 'a_add menu_position').'</h2>';
```

A message is displayed when a new position is added.

```
if ($WasAdded)
{
    echo '<div class="alert alert-success" role="alert">'.Yii::t('app', 'a_option_menu_saved').'</div>';
}

?>

<div class="leftmenuadmin-index">
```

We start a form that allows you to place items in the menu.

```
<?php $form = ActiveForm::begin([
    'id' => 'login-form',
    'action' => Yii::$app->params['pageUrl'].'leftmenuadmin/index',
    'enableAjaxValidation' => false,
    'enableClientValidation' => false,
```

```
]); ?>
```

Print the field with the title of the menu item.

```
<?= $form->field($model, 'menu_title') ?>
<?php
```

We create a array with the main items that can be the main menu items.

```
$ItemContentList['main'] = Yii::t('app', 'a_menu_main');
$ItemContentList['login'] = Yii::t('app', 'a_menu_login');
$ItemContentList['logout'] = Yii::t('app', 'a_menu_logout');
$ItemContentList['changepassword'] = Yii::t('app', 'a_menu_change_password');
$ItemContentList['profil'] = Yii::t('app', 'a_menu_profil');
$ItemContentList['register'] = Yii::t('app', 'a_menu_register');
$ItemContentList['password'] = Yii::t('app', 'a_menu_pass_reminder');
$ItemContentList['page'] = Yii::t('app', 'a_menu_page');
$ItemContentList['pageone'] = Yii::t('app', 'a_menu_page_one');
$ItemContentList['download'] = Yii::t('app', 'a_menu_download');
$ItemContentList['blog'] = Yii::t('app', 'a_menu_blog');
$ItemContentList['blogcategory'] = Yii::t('app', 'a_menu_blog_category');
$ItemContentList['blogone'] = Yii::t('app', 'a_menu_blog_one');
$ItemContentList['article'] = Yii::t('app', 'a_menu_articles');
$ItemContentList['articleone'] = Yii::t('app', 'a_menu_article_one');
$ItemContentList['contact'] = Yii::t('app', 'a_menu_contact');
```

Place the data from the table in the list field.

```
echo $form->field($model, 'menu_what')->dropDownList($ItemContentList);
?>
<?php
$ItemList = array();
?>
```

We create a drop-down list field that will be filled in by JavaScript.

```
<?= $form->field($model, 'menu_content_id')->dropDownList($ItemList) ?>
<?php
```

Check whether the item is in the submenu.

```
$ItemSubList['n'] = Yii::t('app', 'a_no');
```

We are transforming the array of the main elements.

```
foreach($MainsTable as $Key=>$Value)
{
    $ItemSubList[$Key] = $Value;
}
```

Create a field with the submenus in the menu.

```
echo $form->field($model, 'menu_sub')->dropDownList($ItemSubList);
?>
<?php
```

We define a field in which we can specify whether a user must be logged in to see a given menu item.

```
$ItemLoginList['n'] = Yii::t('app', 'a_no');
$ItemLoginList['y'] = Yii::t('app', 'a_yes');
echo $form->field($model, 'menu_login')->dropDownList($ItemLoginList);
?>
```

We print the button for sending the form.

```
<div class="form-group">
<?= Html::submitButton(Yii::t('app', 'a_menu_add'), ['class' =>'btn btn-primary']) ?>
</div>
<?php ActiveForm::end(); ?>
</div>
<script>
```

When you change a main menu item, select the contents of the submenu.

```
$("#leftmenuadmin-menu_what").change(function () {
var str = "";
$("#leftmenuadmin-menu_what option:selected").each(function()
{
```

Set the home page.

```
if($(this).val() == "main")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_main'); ?>": "main"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}
```


Set the login page.

```
else if($(this).val() == "login")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_login'); ?>": "login"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}
```

Set the logout page.

```
else if($(this).val() == "logout")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_logout'); ?>": "logout"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}
```

Set the password change page.

```
else if($(this).val() == "changepassword")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_change_password'); ?>": "changepassword"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}
```

Set the profile editing page.

```
else if($(this).val() == "profil")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_profil'); ?>": "profil"};
var $el = $("#leftmenuadmin-menu_content_id");
```

```

$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}

```

We set the registration page for the new user.

```

else if($(this).val() == "register")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_register'); ?>": "register"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}

```

Set the password reminder page.

```

else if($(this).val() == "password")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_pass_reminder'); ?>": "password"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}

```

Set the menu to the text page.

```

else if($(this).val() == "page")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_page'); ?>": "page"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}

```

Set the menu to one of the text pages you define in the drop-down box.

```

else if($(this).val() == "pageone")
{
$("#leftmenuadmin-menu_content_id").empty();
<?php
empty($TableReady);
$TableReady = array();
foreach($PageTable as $Key=>$Value)
{
$TableReady[] = ''.$Value.': '.$Key.''';
?>
$("#leftmenuadmin-menu_content_id").append( "<option value=\"<?php echo $Key; ?>\"><?php echo $Key.' -
' . $Value; ?></option>" );
<?php
}
?>
}

```

Set the menu to the download section.

```

else if($(this).val() == "download")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_download'); ?>": "download"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}

```

We set the section on the system blog.

```

else if($(this).val() == "blog")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_blog'); ?>": "blog"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}

```

We set the section on the blog category, which we define in the drop-down box.

```

else if($(this).val() == "blogcategory")

```

```

{
$("#leftmenuadmin-menu_content_id").empty();
<?php
empty($TableReady);
$TableReady = array();
foreach($CategoryTable as $Key=>$Value)
{
$TableReady[] = ''.$Value.': '.$Key.''';
?>
$("#leftmenuadmin-menu_content_id").append( "<option value=\"<?php echo $Key; ?>\"><?php echo $Key.' -
' . $Value; ?></option>" );
<?php
}
?>
}

```

Set the menu to one selected entry in the blog, which will be selected from the list.

```

else if($(this).val() == "blogone")
{
$("#leftmenuadmin-menu_content_id").empty();
<?php
empty($TableReady);
$TableReady = array();
foreach($EntriesTable as $Key=>$Value)
{
$TableReady[] = ''.$Value.': '.$Key.''';
?>
$("#leftmenuadmin-menu_content_id").append( "<option value=\"<?php echo $Key; ?>\"><?php echo $Key.' -
' . $Value; ?></option>" );
<?php
}
?>
}

```

Set the menu to the editorial contact section.

```

else if($(this).val() == "contact")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_contact'); ?>": "contact"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("#<option></option>").attr("value", value).text(key));
});
}

```

Set the menu for the article list.

```
else if($(this).val() == "article")
{
var newOptions = {"<?php echo Yii::t('app', 'a_menu_articles'); ?>": "article"};
var $el = $("#leftmenuadmin-menu_content_id");
$el.empty();
$.each(newOptions, function(key,value)
{
$el.append($("<option></option>").attr("value", value).text(key));
});
}
```

Set the menu item to a single item that you define in the drop-down list.

```
else if($(this).val() == "articleone")
{
$("#leftmenuadmin-menu_content_id").empty();
<?php
empty($ArticlesTable);
$TableReady = array();
foreach($ArticlesTable as $Key=>$Value)
{
$TableReady[] = "'".$Value.'"': "'".$Key.'"';
?>
$("#leftmenuadmin-menu_content_id").append( "<option value=\"<?php echo $Key; ?>\"><?php echo $Key.' -
'.$Value; ?></option>\"");
<?php
}
?>
}
});
})
.change();
</script>
```

Chapter 21. Saving and reading administrator logs

Model of saving administrator actions - LogAdmin

A model that saves information about actions taken by the user in the admin panel. This allows you to see exactly who edited it, when it was done and from what IP address.

File location: ../models/Logadmin.php

We start the file, define the namespace and load the framework.

```
<?php
namespace app\models;
use Yii;
```

We define a class name identical to the file name that will be the **ActiveRecord** extension.

```
class Logadmin extends \yii\db\ActiveRecord
{
```

A method that returns the name of a table whose name is located inside two brackets and is preceded by a percentage. This will add a prefix to the table name.

```
public static function tableName()
{
    return '{{%log}}';
}
```

The method responsible for processing the values of the boxes on the form and for determining whether or not they meet the requirements.

```
public function rules()
{
```

We define the required fields one by one, the **log_user_id** field must be a number, **log_time** must be considered a safe value, **log_what** must be a string of maximum 50 characters and **log_ip** must also be a string of maximum 15 characters.

```
return [
    [['log_user_id', 'log_what', 'log_time', 'log_ip'], 'required'],
    [['log_user_id'], 'integer'],
    [['log_time'], 'safe'],
```

```

[['log_what'], 'string', 'max' => 50],
[['log_ip'], 'string', 'max' => 15],
];
}

```

A method that allows you to select an e-mail address using the **\$UserId** parameter, which is a user ID.

```

public function SelectUser($UserId)
{

```

We create a query by loading the first parameter into it and selecting only one record using the **queryOne()** method.

```

$queryDataIs = Yii::$app->db->createCommand('SELECT user_email FROM {%user} WHERE
user_id = :user_id
')
->bindParam(':user_id', $UserId)
->queryOne();

```

We check whether the result of the query contains the e-mail address, in case it is empty we enter into it a string of characters informing the controller about the lack of a user meeting the requirements.

```

if($queryDataIs['user_email'] == "")
{
    $toReturn = '[no user]';
}
else
{
    $toReturn = $queryDataIs['user_email'];
}

```

We return the result of the method.

```

return $toReturn;
}

```

A method that returns tables in which the keys are the names of the form fields and the values of their descriptions.

```

public function attributeLabels()
{
    return [
        'log_id' => Yii::t('app', 'log_id'),
        'log_user_id' => Yii::t('app', 'log_user_id'),
        'log_what' => Yii::t('app', 'log_what'),
        'log_time' => Yii::t('app', 'log_time'),
        'log_ip' => Yii::t('app', 'log_ip'),
    ];
}

```

```
}  
}
```

Controller of login attempts for the administrator - LogadminController

Controller designed to track messages and errors made by users. On the basis of logs you can quickly catch people or bots trying to log in to your account using trial and error methods. The **ActiveRecord** interface will be used for the design.

File location: ../controllers/LogadminController.php

We start the PHP file, load the framework, model with data access, plugin for generating reports, main controller file, error handling and data filter.

```
<?php  
namespace app\controllers;  
use Yii;  
use app\models\Logadmin;  
use yii\data\ActiveDataProvider;  
use yii\web\Controller;  
use yii\web\NotFoundHttpException;  
use yii\filters\VerbFilter;
```

We define a class name identical to the file name, which is the extension of the main class.

```
class LogadminController extends Controller  
{
```

We set the appearance of the page to the administrative section.

```
public $layout = 'admin';
```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)  
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```
if($session['yii_user_id'] != "")
```



```

{
if($session['yii_user_root'] != "y")
{
return $this->redirect(['/right']);
}
}

```

If the user is not logged in, we will take him to the login form page.

```

else
{
return $this->redirect(['/login']);
}

```

At the end we call the method with the same name from the master controller.

```

return parent::beforeAction($action);
}

```

The method will be responsible for selecting the records from the table and displaying them in the appropriate form.

```

public function actionIndex()
{

```

We save the administrator log with information about access to the section.

```

Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_browse_logs'));

```

Using **ActiveDataProvider()** class we pass the model from which data will be selected and information about the fact that records are to be sorted by **log_id** key in descending order.

```

$dataProvider = new ActiveDataProvider([
'query' => Logadmin::find(),
'sort'=> ['defaultOrder' => ['log_id'=>SORT_DESC]]
]);

```

We generate a view to which we pass the result with the information contained in it.

```

return $this->render('index', [
'dataProvider' => $dataProvider,
]);
}
}

```

View - login information to the service

A set of all login attempts to our website with their exact time, IP address and result.

File location: ../views/logadmin/index.php

We start the PHP file, load the HTML tag helper and the grid for data presentation.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title of the page and the position.

```
$this->title = Yii::t('app', 'a_aplication_logs_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogadmin-index">
```

The page header is displayed.

```
<h1><?= Html::encode($this->title) ?></h1>
```

Using the loaded grid generator we build an appearance in which the following data will be displayed: user ID, user login, how the attempt ended, login time and IP address from which it was made.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'log_id',
        [
            'label' => 'User',
            'class' => 'yii\grid\DataColumn',
            'value' => function ($data) {
                global $TableUsers;
                return $data->SelectUser($data->log_user_id);
            },
        ],
        'log_what',
        'log_time',
        'log_ip',
    ],
]); ?>
</div>
```

Chapter 22. Error controller

Error administration controller - Error404adminController

Controller designed to view information about errors in our service. If the user does not find the page, for example, through a search engine or by clicking on a link inside the service, it will be saved. Thanks to this, we will be able to manage the page by removing type 404 errors.

File location: ../controllers/Error404adminController.php

We start the PHP file, define the namespace, load the framework, model, plugin for data presentation on the website, the main class of the controller, error handling and data filtering.

```
<?php
namespace app\controllers;
use Yii;
use app\models>Error404;
use yii\data\ActiveDataProvider;
use yii\web\Controller;
use yii\web\NotFoundHttpException;
use yii\filters\VerbFilter;
```

We define the main class identical to the file name, which is the controller extension.

```
class Error404adminController extends Controller
{
```

Set the variable page appearance on the admin panel.

```
public $layout = 'admin';
```

The method is called up automatically when you load a class and you can check whether you are logged in or not.

```
public function beforeAction($action)
{
```

Load the properties of the session.

```
$session = Yii::$app->session;
```

We check whether the user is logged in and whether his rights are at the administrator level. In case, however, they are not redirected to the information about the lack of rights.

```

if($session['yii_user_id'] != "")
{
if($session['yii_user_root'] != "y")
{
return $this->redirect(['/right']);
}
}
}

```

If the user is not logged in, we will take him to the login form page.

```

else
{
return $this->redirect(['/login']);
}

```

At the end we call the method with the same name from the master controller.

```

return parent::beforeAction($action);
}

```

The method is designed to generate a list of erroneous links to a web page.

```

public function actionIndex()
{

```

We save the administrator log for access to the section.

```

Yii::$app->OtherFunctionsComponent->WriteLog(Yii::t('app', 'log_user_watched_error404'));

```

Using **ActiveDataProvider()** we define selected data from the model with error 404 and put them in descending order according to the main index contained in the **error_id** field.

```

$dataProvider = new ActiveDataProvider([
'query' => Error404::find(),
'sort' => ['defaultOrder' => ['error_id' => SORT_DESC]]
]);

```

We generate the appearance of the page and send it an object with the data.

```

return $this->render('index', [
'dataProvider' => $dataProvider,
]);
}
}

```

View - error file 404

If you call up an incorrect page on our site, the address of which does not exist, a special page will be called up. All these calls will be available in the admin panel.

File location: ../views/error404admin/index.php

Start the PHP file, load the helper for HTML generation and the grid plugin.

```
<?php
use yii\helpers\Html;
use yii\grid\GridView;
```

We define the title and location of the page.

```
$this->title = Yii::t('app', 'a_error404_header');
$this->params['breadcrumbs'][] = ['label' => Yii::t('app', 'a_admin'), 'url' => ['/admin/index']];
$this->params['breadcrumbs'][] = $this->title;
?>
<div class="blogadmin-index">
```

We print the header.

```
<h1><?= Html::encode($this->title) ?></h1>
```

The plugin we have loaded will be used to display the data broken down by pages. These will be the identifier of the error, the page from which you came, the page on which the error occurred, and the date when the error occurred.

```
<?= GridView::widget([
    'dataProvider' => $dataProvider,
    'columns' => [
        ['class' => 'yii\grid\SerialColumn'],
        'error_id',
        'error_page_from',
        'error_page',
        'error_date',
    ],
]); ?>
</div>
```

Chapter 23. Views

Views are a part of the system corresponding to the appearance of our website, individual content and forms. By dividing the software structure into parts: model, controller and view after the error, we are able to see in which file it occurred. Views relieve us of the burden of building controllers, because you only need to put data into the array, then pass it to a file with a defined view and display it there.

Administrator panel main file

It is responsible for generating the appearance of the admin panel into which particular sections of the website will be inserted.

File location: ../views/layouts/admin.php

We start the PHP file, load the HTML generator helper, location bars, generate the bar with the current position and application data.

```
<?php
use yii\helpers\Html;
use yii\bootstrap\Nav;
use yii\bootstrap\NavBar;
use yii\widgets\Breadcrumbs;
use app\assets\AppAsset;
```

Add an object to the application data.

```
AppAsset::register($this);
```

We check if a session exists and if it is active, in case it is not, we start a new one.

```
$session = Yii::$app->session;
if (!$session->isActive)
{
    $session->open();
}
```

We retrieve data concerning the configuration of the page.

```
$ConfigPage = Yii::$app->OtherFunctionsComponent->GetConfigData();
```

We print the page header.

```
?>
<?php $this->beginPage() ?>
<!DOCTYPE html>
```

```

<html lang="<?= Yii::$app->language ?>">
<head>
<meta charset="<?= Yii::$app->charset ?>">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<?= Html::csrfMetaTags() ?>
<meta name="description" content="<?php echo $ConfigPage['description']; ?>" />
<meta name="keywords" content="<?php echo $ConfigPage['keywords']; ?>" />
<title><?php echo Yii::t('app', 'a_logo'); ?></title>
<?php $this->head() ?>

```

We load the libraries we need.

```

<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-latest.min.js"
type="text/javascript"></script>
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery.selection.js"></script>
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui.min.js"></script>
<link href="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui.css" rel="stylesheet" />
<link href="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui-timepicker-addon.css"
rel="stylesheet" />
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui-timepicker-addon.js"></script>
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/ckeditor/ckeditor.js"></script>

```

We define additional styles.

```

<style>
.my-navbar
{
background-color: #F5F5F5;
}
</style>
</head>

```

We start the page.

```

<body>
<?php $this->beginBody() ?>

```

We print the logo and the main navigation bar.

```

<div class="wrap">
<?php
NavBar::begin([
'brandLabel' => Yii::t('app', 'a_logo'),
'brandUrl' => Yii::$app->homeUrl.'configadmin/index',
'options' => [
'class' => 'my-navbar navbar-fixed-top',

```

```

],
]);

echo Nav::widget([
'options' => ['class' => 'navbar-nav navbar-right'],
'items' => [
['label' => Yii::t('app', 'a_amenu_config'), 'url' => ['/configadmin/index']],
['label' => Yii::t('app', 'a_amenu_page'), 'url' => ['/pageadmin/index']],
['label' => Yii::t('app', 'a_amenu_articles'), 'url' => ['/articleadmin/index']],
['label' => Yii::t('app', 'a_amenu_blog'), 'items' =>
[
['label' => Yii::t('app', 'a_amenu_blog'), 'url' => ['/blogadmin/index']],
['label' => Yii::t('app', 'a_amenu_blog_category'), 'url' => ['/blogcategoryadmin/index']],
]
],
['label' => Yii::t('app', 'a_amenu_users'), 'items' =>
[
['label' => Yii::t('app', 'a_amenu_users'), 'url' => ['/useradmin/index']],
['label' => Yii::t('app', 'a_amenu_password_change'), 'url' => ['/passwordadmin/index']],
['label' => Yii::t('app', 'a_amenu_logs'), 'url' => ['/logadmin/index']],
]
],
['label' => Yii::t('app', 'a_amenu_contact'), 'url' => ['/contactadmin/index']],
['label' => Yii::t('app', 'a_amenu_download'), 'items' =>
[
['label' => Yii::t('app', 'a_amenu_download'), 'url' => ['/downloadadmin/index']],
['label' => Yii::t('app', 'a_amenu_statistics'), 'url' => ['/downloadstatsadmin/index']],
]
],
['label' => Yii::t('app', 'a_amenu_left_menu'), 'url' => ['/leftmenuadmin/index']],
['label' => Yii::t('app', 'a_error404_header'), 'url' => ['/error404admin/index']],
['label' => Yii::t('app', 'a_amenu_logout'), 'url' => ['/logout']],
],
]);

NavBar::end();
?>

```

Display the navigation bar and the page content.

```

<div class="container">
<?= Breadcrumbs::widget([
'links' => isset($this->params['breadcrumbs']) ? $this->params['breadcrumbs'] : [],
]) ?>
<?= $content ?>
</div>
</div>

```


We print the page footer.

```
<footer class="footer">
<div class="container">
<p class="pull-left"><?php echo $ConfigPage['foot']; ?></p>
<p class="pull-right"><?= Yii::powered() ?></p>
</div>
</footer>
<?php $this->endBody() ?>
</body>
</html>
<?php $this->endPage() ?>
```

The session closed.

```
<?php
$session->close();
?>
```

Page layout main file

The file is designed to insert the content of all sub-pages that the system generates. It contains items such as the main menu and loads all the necessary libraries.

File location: ../views/layouts/main.php

We start the PHP file and load the tools to generate HTML, to prepare the navigation bar and to generate it.

```
<?php

use yii\helpers\Html;
use yii\bootstrap\Nav;
use yii\bootstrap\NavBar;
use yii\widgets\Breadcrumbs;
use app\assets\AppAsset;
```

We add the current page to the testing module.

```
AppAsset::register($this);
```

Check if the session exists, if it is not open, then perform such an operation.

```
$session = Yii::$app->session;
if (!$session->isActive)
```

```
{
$session->open();
}
```

We download the configuration data from the database.

```
$ConfigPage = Yii::$app->OtherFunctionsComponent->GetConfigData();
```

In this variable you will find the menu generated for the user.

```
$ItemsMenu = Yii::$app->OtherFunctionsComponent->GetMenu($session['yii_user_id']);
```

We start the page with the header declaration.

```
?>
<?php $this->beginPage() ?>
<!DOCTYPE html>
<html lang="<?= Yii::$app->language ?>">
<head>
<meta charset="<?= Yii::$app->charset ?>">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1">
<?= Html::csrfMetaTags() ?>
<meta name="description" content="<?php echo $ConfigPage['description']; ?>" />
<meta name="keywords" content="<?php echo $ConfigPage['keywords']; ?>" />
<title><?php
```

Check if the variable containing the title has been set. In case it contains a value, we write it down, but when it is empty we set only the title of the page that comes from its configuration.

```
if($this->title != "")
{
echo Html::encode($this->title).' - '.$ConfigPage['title'];
}
else
{
echo $ConfigPage['title'];
}
?></title>
<?php $this->head() ?>
```

We load the necessary scripts to generate the page.

```
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-latest.min.js"
type="text/javascript"></script>
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery.selection.js"></script>
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui.min.js"></script>
```

```

<link href="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui.css" rel="stylesheet" />
<link href="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui-timepicker-addon.css"
rel="stylesheet" />
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/jquery-ui-timepicker-addon.js"></script>
<script src="<?php echo Yii::$app->params['pageUrl']; ?>library/ckeditor/ckeditor.js"></script>

<style>
.my-navbar
{
background-color: #F5F5F5;
}
</style>
</head>
<body>
<?php $this->beginBody() ?>

```

We create the upper menu with the logo and the name of the page.

```

<div class="wrap">
<?php
NavBar::begin([
'brandLabel' => Yii::t('app', 'page_title'),
'brandUrl' => Yii::$app->homeUrl,
'options' => [
'class' =>'my-navbar navbar-fixed-top',
],
]);

```

We check if the user is logged in. For non-logged users, we print a menu containing items such as logging in, registration or password reminders.

```

if($session['yii_user_id'] == "")
{
echo Nav::widget([
'options' => ['class' =>'navbar-nav navbar-right'],
'items' => [
['label' => Yii::t('app', 'link_home'), 'url' => ['/']],
['label' => Yii::t('app', 'link_login'), 'url' => ['/login']],
['label' => Yii::t('app', 'link_register'), 'url' => ['/register']],
['label' => Yii::t('app', 'link_remind'), 'url' => ['/remind-password']],
],
]);
}
else

```

When you are logged in to your account, you need to print links that allow you to change your password, change profile data or log out.

```

{
echo Nav::widget([
'options' => ['class' =>'navbar-nav navbar-right'],
'items' => [
['label' => Yii::t('app', 'link_home'), 'url' => ['/']],
['label' => Yii::t('app', 'link_change'), 'url' => ['/change-password']],
['label' => Yii::t('app', 'link_profile'), 'url' => ['/profile']],
['label' => Yii::t('app', 'link_logout'), 'url' => ['/logout']],
],
]);
}
NavBar::end();
?>

```

We print the bar of the current position.

```

<div class="container">
<?= Breadcrumbs::widget([
'links' => isset($this->params['breadcrumbs']) ? $this->params['breadcrumbs'] : [],
]) ?>

```

Create a menu from the variable in which the data array has been placed.

```

<div class="row">
<div class="col-md-2"><?php
echo Nav::widget([
'options' => ['class' =>'sidebar-menu treeview'],
'items' => $ItemsMenu ]);
?></div>

```

We print the content of the page.

```

<div class="col-md-10">
<?= $content ?></div>
</div>
</div>
</div>

```

We create a page footer.

```

<footer class="footer">
<div class="container">
<p class="pull-left"><?php echo $ConfigPage['foot']; ?></p>
<p class="pull-right"><?= Yii::powered() ?></p>
</div>
</footer>

```

```
<?php $this->endBody() ?>
</body>
</html>
<?php $this->endPage() ?>
<?php
$session->close();
?>
```

Chapter 24. Languages

Because our website will be multilingual, you will also need to think about how to translate it into other languages. We will do it with the `t` (translate) method coming directly from the framework. Creating a file for translating a page into another language is very easy and consists in defining an appropriate array, where the keys will be strings of characters placed in the place where the text is to appear, and the value of the translated content.

Setting the application language

The first element will be to set the default language of the application so that the framework knows where to download the translations. Open the configuration file: `../config/web.php` and place a line inside the main array:

```
'language' => 'en',
```

This means that the default language of the framework will be English. International language labelling should be used in this case, particularly for South American countries.

Create language files

Once you have configured the software to work in the correct language, it's time to create folders and files. In the main framework directory we create a directory named **messages**, then in this directory we create two subdirectories: **en** and **pl**. They will be needed in order to define language versions, which will be English and Polish.

In both folders we create a file called **app.php**, in which the table with translations will be returned.

```
<?php

return [
    'page_title' => 'Aktywna strona WWW w Yii',
    'link_home' => 'Start',
    'link_login' => 'Zaloguj',
    'link_register' => 'Zarejestruj',
    'link_remind' => 'Przypomnij hasło',
    'link_change' => 'Zmień hasło',
    [...]
];

?>
```

This file is made up of a returned array where the keys are words placed by us in controllers, models or views, and the values in the table are the texts that are to appear in the place specified by us.

The text is called up using the `t` method, in which two parameters are defined. The first is the name of the file in which the translation is located (without extension), and the second is the array key from which you want to select the text.

```
echo Yii::t('app', 'page_title');
```

Chapter 25. Components and libraries

Components are additional possibilities of extending our framework, which allow us to add both additions already published on the website and to write our own ones.

Component that logs administrator actions, retrieves configuration, creates URLs, and creates menus

The first element before creating a component should be for us to consider whether the given methods will be used in more than one place. The component will contain logging of administrative actions, downloading configuration, creating URLs and creating menus. These activities will be repeated on almost every website, so it is very cost effective to put them in a shared separate file.

File location: ../components/OtherFunctionsComponent.php

We define the first part of the **OtherFunctions** file name according to our own idea, it is any name. The second part of the Component defines the function of the file. We loaded this file to the framework in the chapter on downloading and configuration.

We start the PHP file, define the namespace, load the framework, the parent component, the configuration error component, the administrator model, the settings model and the left menu model.

```
<?php

namespace app\components;

use Yii;
use yii\base\Component;
use yii\base\InvalidConfigException;
use app\models\Logadmin;
use app\models\Configadmin;
use app\models\Leftmenuadmin;
```

We create a component class that is the equivalent of a file, which is an extension of the main class containing the component.

```
class OtherFunctionsComponent extends Component
{
```

The **WriteLog** method is designed for saving application logs, it takes one parameter, and it is a string of characters corresponding to the message we want to save.

```
public function WriteLog($Message)
{
```

Load the session settings.

```
$session = Yii::$app->session;
```

We create an object from the **Logadmin** model designed to save information in a log table.

```
$logUser = new Logadmin();
```

The following variables are filled in: user ID, message, current date, time and IP address of the computer of the person using the website.

```
$logUser->log_user_id = $session['yii_user_id'];
$logUser->log_what = $Message;
$logUser->log_time = date('Y-m-d H:i:s');
$logUser->log_ip = $_SERVER['REMOTE_ADDR'];
```

Using the **save()** method, we save the acquired values.

```
$logUser->save();
}
```

The **GetConfigData()** method is responsible for obtaining configuration data and does not have any parameters transferred to it.

```
public function GetConfigData()
{
```

We create an object from **Configadmin()** class and then we refer to the **GetConfig()** method, which downloads the settings and returns the content to the **\$Config** variable.

```
$ConfigAdmin = new Configadmin();
$Config = $ConfigAdmin->GetConfig();
```

The variable should still be returned as a result of the method.

```
return $Config;
}
```

The method responsible for creating links on our website. This combination of URLs makes them look more professional. In addition, most search engines add extra points to a page whose content title is also found in the URL.

The method takes four parameters: **\$OtherData** - additional data added in the URL address, **\$What** - the element to which the link is to be created, **\$Id** - the identifier of the element coming from the database and **\$Extra** - processed information to be included in the URL address.

```
private function MakeUrl($OtherData,$What,$Id='', $Extra='')
{
```

We check the value of the **\$What** variable and build appropriate addresses depending on it.

```
if($What == 'main')
{$Url = Yii::$app->params['pageUrl'];}
elseif($What == 'none')
{$Url = Yii::$app->params['pageUrl'];}
elseif($What == 'login')
{$Url = Yii::$app->params['pageUrl'].'login';}
elseif($What == 'logout')
{$Url = Yii::$app->params['pageUrl'].'logout';}
elseif($What == 'changepassword')
{$Url = Yii::$app->params['pageUrl'].'change-password';}
elseif($What == 'profil')
{$Url = Yii::$app->params['pageUrl'].'profile';}
elseif($What == 'register')
```



```

{$Url = Yii::$app->params['pageUrl'].'register';}
elseif($What == 'password')
{$Url = Yii::$app->params['pageUrl'].'remind-password';}
elseif($What == 'contact')
{$Url = Yii::$app->params['pageUrl'].'contact';}
elseif($What == 'download')
{$Url = Yii::$app->params['pageUrl'].'download';}
elseif($What == 'page')
{$Url = Yii::$app->params['pageUrl'].'page';}
elseif($What == 'blog')
{$Url = Yii::$app->params['pageUrl'].'blog';}
elseif($What == 'article')
{$Url = Yii::$app->params['pageUrl'].'article';}
elseif($What == 'pageone')
{$Url = Yii::$app->params['pageUrl'].'page/'.$Extra.'/'.$Id;}
elseif($What == 'blogcategory')
{$Url = Yii::$app->params['pageUrl'].'category/'.$Extra.'/'.$Id;}
elseif($What == 'blogone')
{$Url = Yii::$app->params['pageUrl'].'blog/'.$Extra.'/'.$Id;}
elseif($What == 'articleone')
{$Url = Yii::$app->params['pageUrl'].'article/'.$Extra.'/'.$Id;}

```

Check if anything has been added to the variable containing the address, if we do not include the information about the error.

```

if(!isset($Url))
{
    $Url = 'error';
}

```

We return the variable as a result of calling the method.

```

return $Url;
}

```

A method that allows you to create an array of menus. It accepts one argument determining whether the user has been logged in to the system.

```

public function GetMenu($IsLogged='')
{

```

Create the array.

```

$Items = array();
$AboutData = array();

```

With the help of **Leftmenuadmin** class we create an object in which we search all items in the main menu.

```

$MainAllPages = Leftmenuadmin::find()
->where(['menu_sub' => 0])
->asArray()
->orderBy('menu_poz')
->all();

```

Then with the help of **Leftmenuadmin** class we create an object in which we search all submenus positions.

```
$MainSubPages = Leftmenuadmin::find()
->where('menu_sub != 0')
->asArray()
->orderBy('menu_poz')
->all();
```

Set the login value of the user to **false** and check whether the value in the **\$IsLogged** parameter has been sent. When this has happened, set the value in the variable responsible for the information whether the user is logged in to **true**.

```
$UserLogged = false;

if($IsLogged != "")
{
    $UserLogged = true;
}
```

We create a array with elements.

```
$TableMakeMenu = array();
```

We count the main items in the menu as well as the items in the sub-menu.

```
$HowManyElements = count($MainAllPages);
$HowManyElementsSub = count($MainSubPages);
```

Set the submenu to zero.

```
$IsSubOdThisMenu = 0;
```

All main menu items are viewed via the loop.

```
for($m=0;$m<count($MainAllPages);$m++)
{
```

Set the information about the submenu to **false**.

```
$MainAllPages[$m]['has_submenu'] = false;
```

Use the forum loop to check whether the current menu item has its own menu. In case it is true, we write **true** value into the table.

```
for($sub=0;$sub<count($MainSubPages);$sub++)
{
    if($MainSubPages[$sub]['menu_sub'] == $MainAllPages[$m]['menu_id'])
    {
        $MainAllPages[$m]['has_submenu'] = true;
    }
}
```

When the menu has a submenu, we execute entering them into the table.

```
if($MainAllPages[$m]['has_submenu'])
```

```
{
```

Set up an array containing the submenus.

```
$SubPages = null;
$SubPages = array();
```

Use the loop to go through all the menu items.

```
for($sub=0;$sub<count($MainSubPages);$sub++)
{
```

Check whether the submenu item belongs to the menu currently being processed.

```
if($MainSubPages[$sub]['menu_sub'] == $MainAllPages[$m]['menu_id'])
{
```

Check if the item has been set to view only for people who are logged in.

```
if($MainSubPages[$sub]['menu_login'] == 'y')
{
```

We check if the user is logged in.

```
if($UserLogged)
{
```

Enter the item into the table containing the items present under the menu, creating the correct URL using the **MakeUrl()** method.

```
$SubPages[] = array('label' => $MainSubPages[$sub]['menu_title'], 'url' => $this->MakeUrl($AboutData,$MainSubPages[$sub]['menu_what'],$MainSubPages[$sub]['menu_content_id'],$MainSubPages[$sub]['menu_extra']));
}
}
else
{
```

When the user does not need to be logged in, we create a menu by entering the address obtained using the **MakeUrl()** method.

```
$SubPages[] = array('label' => $MainSubPages[$sub]['menu_title'], 'url' => $this->MakeUrl($AboutData,$MainSubPages[$sub]['menu_what'],$MainSubPages[$sub]['menu_content_id'],$MainSubPages[$sub]['menu_extra']));
}
}
}
```

Check if the menu item has been accepted.

```
if(count($SubPages) > 0)
{
```

If there is at least one menu, then you can create the current one and add it to it under the menu.

```
$ItemsElement = array('label' => $MainAllPages[$m]['menu_title'], 'items' => $SubPages);
}
else
```

```
{
```

In case they are not available, only the main element should be entered.

```
$ItemsElement = array('label' => $MainAllPages[$m]['menu_title'], 'url' => $this->MakeUrl($AboutData, $MainAllPages[$m]['menu_what'], $MainAllPages[$m]['menu_content_id'], $MainAllPages[$m]['menu_extra']));
}
```

We add an element to our table.

```
array_push($Items, $ItemsElement);
}
else
{
```

The menu does not have a submenu so you can add them directly. We check whether the user must be logged in to see the item.

```
if($MainAllPages[$m]['menu_login'] == 'y')
{
```

We check if the user is logged in.

```
if($UserLogged)
{
```

Create a menu item and then attach an array to it.

```
$ItemsElement = array('label' => $MainAllPages[$m]['menu_title'], 'url' => $this->MakeUrl($AboutData, $MainAllPages[$m]['menu_what'], $MainAllPages[$m]['menu_content_id'], $MainAllPages[$m]['menu_extra']));
array_push($Items, $ItemsElement);
}
}
else
{
```

When the user does not need to be logged in, we create a menu item and also add it to the array.

```
$ItemsElement = array('label' => $MainAllPages[$m]['menu_title'], 'url' => $this->MakeUrl($AboutData, $MainAllPages[$m]['menu_what'], $MainAllPages[$m]['menu_content_id'], $MainAllPages[$m]['menu_extra']));
array_push($Items, $ItemsElement);
}
}
}
```

At the end of the method, we return the menu items.

```
return $Items;
}
}

?>
```

JQuery - library

The system will be equipped, especially in the admin panel, with elements facilitating the execution of operations, such as HTML editor, date and automatic rewriting of URLs.

Please visit: <http://jquery.com/download/> where you can download the latest version. Then the JavaScript file is placed in the **library** directory in the main location of the framework.

JQuery - UI

Some plug-ins will require a graphical interface that will be created through the user interface.

Download the file from <http://jqueryui.com/download/>, then extract the files and place them in the **library** framework directory.

JQuery - text retrieval

The admin panel will automatically create URLs based on the content in the field with the title of the article, page or blog. To do this, install a **JQuery** plug that retrieves text from the fields.

Go to <http://madapaja.github.io/jquery.selection/> and download the files from there. Then we unpack the archive and place it in the system in the **library** folder in the main directory of the framework.

JQuery - date stamp

Placing the date of publication in the Blog can be facilitated by using a graphical generator that will show you the calendar. Files can be found at: <http://trentrichardson.com/examples/timepicker/>. We download files and place them in the **library** framework directory.

CKEditor - WYSIWYG editor

The editor will be useful in case of filling in content fields for example for text pages, articles or blog entries. Go to <https://ckeditor.com/ckeditor-4/download/> and download the software. After unpacking the files, we upload them to the **library** directory in the main location of the framework.

Chapter 26. Web site for user

The user has the possibility not only to browse the content generated by us in the admin panel, but also to open an account, log in to the service, edit his profile, remind the password to his account, and contact with the service using a special form.

Account registration

The first thing you need to do when using your account is to register it and then activate it to confirm your email address.

Go to the homepage and click on the **Register** link in the top menu. The registration form contains the following fields: **E-mail** - correct e-mail address, which at the same time will be our login, **Password** and **New Password** - for entering the password and its repetition, **Name and surname** - for our personal data, **Phone** - telephone number to contact, **Website** - website address, **Information** - any information we would like to include on the website. After completing the form, click on the button **Register account**.



The registration form consists of several input fields and a submit button. The fields are labeled as follows: 'E-mail', 'Password', 'New password', 'Name and surname', 'Phone', 'Website', and 'Information'. The 'Information' field is a larger text area. Below the fields is a blue button labeled 'Register account'.

Figure 26.1 Registration form

After completing the form correctly, we will see a message that our account has been created, but it requires activation by clicking a special link in the message sent to us. We check your e-mail inbox. It should contain an e-mail from the website within a few minutes. There is a link in it that you can click on. When we do this, the system will inform us that our account is already active and that you can log in to it.

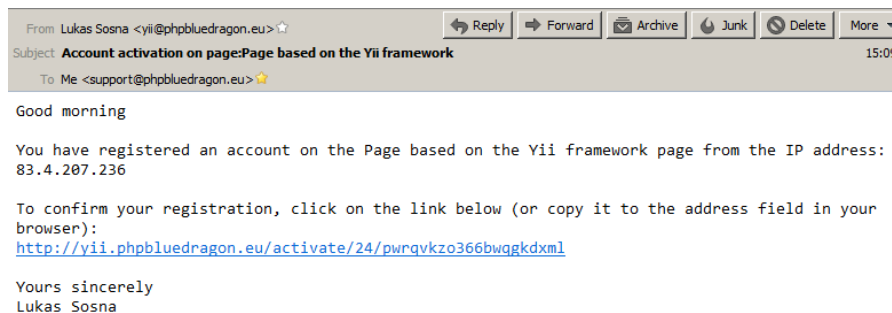


Figure 26.2 E-mail message with link to be activated

Logging in to your account

Logging in to the account is done by clicking on the **Login** option in the upper menu. We fill in the form successively with the data we provided during registration: **E-mail** - the e-mail address through which we created the account, **Password** - the password we entered during registration. Click on the **Login** button.



The login form consists of two text input fields. The first field is labeled "E-mail" and the second field is labeled "Password". Below these fields is a blue button with the text "Login".

Figure 26.3 Login page

Profile completion

You can complete or change the profile data on your account by clicking on the **Profile** link from the top bar (you must be logged in to your account). We go to the form where we fill in the following fields: **Name and surname** - for our personal data, **Phone** - contact telephone number, **Website** - website address, **Information** - any information we would like to include on the website. After completing the form, click on the button **Update profile**.



The profile editing form contains four text input fields. The first field is labeled "Name and surname", the second is labeled "Phone", the third is labeled "Website", and the fourth is labeled "Information". Below these fields is a blue button with the text "Update profile".

Figure 26.4 Profile editing form

Changing the password

Changing the password of our account is done in a special form, which we will go to by clicking on the link **Change password** from the top bar (we have to be logged in to our own account). In the form we fill in the fields: **Old password** - enter our current password, **New password** and **Repeat new password** - the fields are used to enter the new password and then repeat it. After filling in the form, click on the button **Change password**.

Old password

New password

Repeat new password

Change password

Figure 26.5 Change of password form

Attention! The new password must be at least 8 characters long. These must include at least one: a small letter, a capital letter and a number.

Logging out

Always log out of your account at the end of your work. When you are logged in, click on the **Log out** link on the top bar.

Restoring the password

It may happen that we forget the password, forgotten the time comes with the help of the option of reminding the password by the system. Click on the **Remind password** link in the upper menu. The form should be filled in by typing in the following fields: **E-mail** - e-mail to the account we want the password to be restored to us, **CAPTCHA** - text appearing next to us in the form of an image. Click the **Send e-mail** button.

E-mail

CAPTCHA



Send e-mail

Figure 26.6 Password reminder form

The next step will be to check our email inbox. It contains a letter and a link to remind you of your password. Click on the link or copy it to the address bar of your browser.

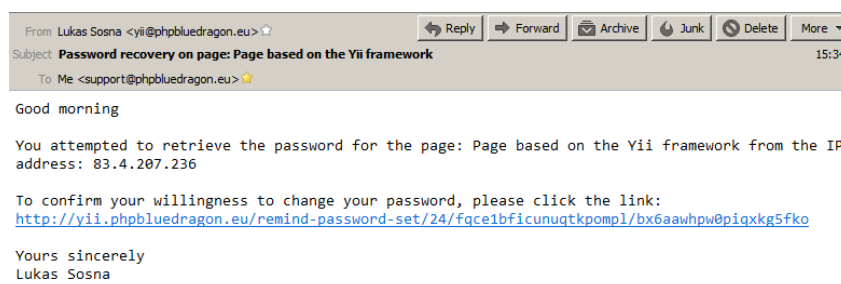


Figure 26.7 E-mail with password reminder link

After entering the URL into your browser, you will receive a page on which you can find information that the password has been changed. Then we check our inbox again. It contains an e-mail with a new password.

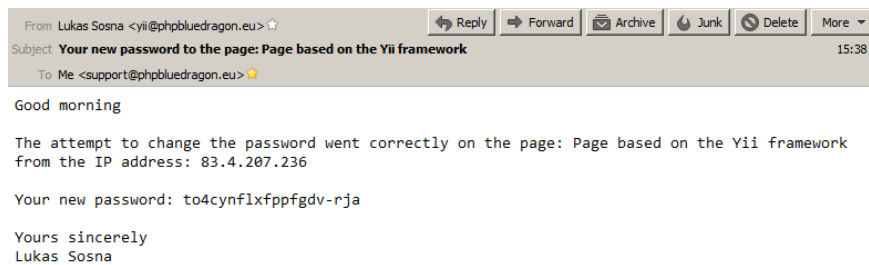


Figure 26.8 New password for our account

Chapter 27. Administrator's webpage

The administrator panel is a place where we can add, edit and delete content, manage users and browse the logs of the system. Access to the panel will be granted only to persons logged in to accounts where appropriate privileges have been set.

Go to the admin panel

The first thing you need to do is log in to your own user account by entering your e-mail address and password. Make sure that you have administrative access rights to your account. After logging into the account after the main page address we enter the word **admin**.

After installing the system provided with the login data book:

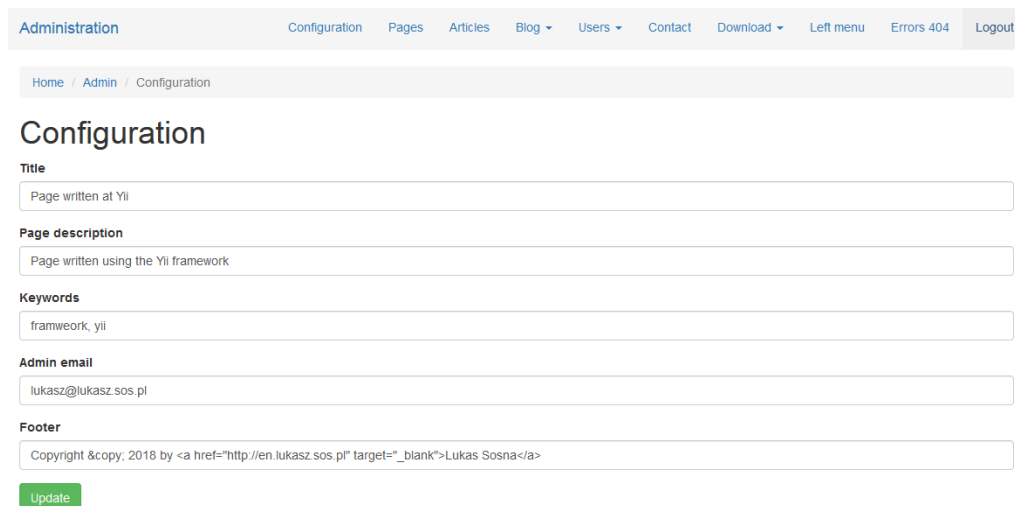
E-mail/Login: **lukasz@lukasz.sos.pl**

Password: **YiiFramework2**

An example of a website is located at: **http://yii.phpbluedragon.eu/**, so please enter the address at: **http://yii.phpbluedragon.eu/admin**.

Page setup

The administrator panel contains a menu located at the top of it, click on it on the **Configuration** link. In the form we fill in: **Title** - title of the page in the TITLE bar, **Page description** of the page - short description of what elements the page contains, **Keywords** - description of the page in the form of words most corresponding to the content, **Admin e-mail** - main e-mail address of the page administrator, **Footer** - information contained in the page footer at the bottom. After filling in the fields, click on the **Update button**.



The screenshot shows the 'Configuration' page of the Yii Framework administrator interface. At the top, there is a navigation bar with links: Administration, Configuration, Pages, Articles, Blog, Users, Contact, Download, Left menu, Errors 404, and Logout. Below this is a breadcrumb trail: Home / Admin / Configuration. The main heading is 'Configuration'. The form contains five sections: 'Title' with the value 'Page written at Yii', 'Page description' with 'Page written using the Yii framework', 'Keywords' with 'framework, yii', 'Admin email' with 'lukasz@lukasz.sos.pl', and 'Footer' with 'Copyright © 2018 by Lukas Sosna'. A green 'Update' button is at the bottom left.

Figure 27.1 Page configuration

System text pages

Elements containing content that can be easily used in the system, thanks to which it is possible to present any information.

Page viewing

Select the Pages link in the upper menu. The section presenting text pages available in the system will be moved. At the top of the page there is a button allowing you to add a new item. The table below shows the text pages and their associated features, such as editing, details and deleting.

[Home](#) / [Admin](#) / [Text pages](#)

Text pages

Create a text page

Showing 1-20 of 28 items.



























#	ID	Title	
1	1	Yii Home Page	 
2	2	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  
3	3	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  
4	4	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  
5	5	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  
6	6	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  
7	7	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  
8	8	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  
9	9	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	  

Figure 27.2 Viewing pages

Adding a page

Click on the **Pages** link from the top menu, then on the page on the **Create text page** button. The form on the new page contains the following fields: **Title** - name of the page and its title, **Content** - content of the page, **URL** - end of the URL created automatically from the title. After completing the fields, click on the **Create** button.

[Home](#) / [Admin](#) / [Text pages](#) / [Create a text page](#)

Create a text page

Title

Content

URL

Create


Figure 27.3 Article insertion form

Page editing

Click on **Pages** in the main menu. Next, you can browse the text pages and click on the **Update** button on the right side of the item you want to edit.

[illegible]

Viewing the page details



A screenshot of the 'View' button in the 'Add New' dialog. The button is labeled 'View' and is located next to the 'Add New' button.

We will be moved to the page where all data of the selected item will be displayed. At the top of the page there are also buttons for editing and removing the item.

Home / Admin / Text pages / Lorem ipsum dolor sit amet, consectetur adipiscing elit.	
Edit text page Delete page	
ID	2
Title	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Content	<p><p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium. Suspendisse odio magna, aliquam sed nulla sed, rutrum auctor neque. Pellentesque tincidunt massa eget tellus vehicula malesuada. Pellentesque accumsan aliquet sagittis. Quisque mollis leo faucibus cursus consequat. Morbi elementum, velit eu aliquet pulvinar, quam ex lobortis lectus, ut blandit risus augue a feis. In gravida varius arcu, at vulputate purus eleifend vestibulum. Nunc efficitur feis nec libero varius vehicula. In sed turpis purus. Nunc feugiat nisi ac augue viverra scelerisque. Donec sit amet augue in leo lobortis dictum. Sed id tortor eget metus gravida convallis in et metus. Sed posuere turpis ultrices, vulputate diam ac, suscipit erat. Mauris eleifend urna at erat faucibus, vitae feugiat ex aliquam. Duis at diam vitae neque maximus vehicula. Phasellus fermentum suscipit velit, at gravida leo varius at. Nunc pharetra nibh ac tortor scelerisque luctus. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Aenean sagittis turpis quis dictum consequat. Sed a tellus non ipsum semper commodo at tincidunt ligula. Proin ullamcorper orci et turpis hendrerit, quis ullamcorper lectus varius. Cras sollicitudin diam egestas efficitur sollicitudin. Integer justo augue, fringilla vel velit non, aliquet luctus arcu. Nullam sodales, est at consectetur fermentum, massa tellus fermentum nisi, ac rhoncus velit lectus quis magna. Mauris non eros justo. Cras luctus dui a est laoreet interdum quis et quam. Quisque vel dolor molestie, euismod sapien ut, vehicula leo. Mauris eu rhoncus dolor. Nulla in erat at est faucibus porta id nec lacus. Nam aliquet fermentum tellus vitae fermentum. Morbi euismod maximus urna eget finibus. Vestibulum ultrices, nulla eu dictum fringilla, sapien orci commodo dui, sed vestibulum turpis enim sit amet ante. Sed elementum tortor non quam ullamcorper lacinia. Etiam elit ligula, laoreet sed sollicitudin eu, feugiat sit amet nulla. Ut sit amet lorem dui. Duis ultrices, justo at euismod accumsan, arcu leo ultrices sapien, non interdum mi risus vitae mi. Fusce lectus elit, vulputate imperdiet velit nec, tempor bibendum tellus. Fusce tincidunt elementum nibh, vitae lacinia libero varius nec. Donec suscipit, diam accumsan aliquet maximus, dui dolor lobortis nibh, non tempor dolor risus sed magna. Curabitur sit amet scelerisque lacus, sit amet finibus elit. Mauris tempor, orci at tempor rhoncus, odio magna tincidunt odio, in commodo diam turpis ac erat. Suspendisse nulla diam, ultrices vel efficitur ac, imperdiet sed orci. Suspendisse egestas dui eu ipsum fringilla mollis. Duis gravida interdum consequat. Integer blandit nulla vitae dignissim vehicula. Donec placerat nisi a tempor pulvinar. Etiam viverra porta ornare. Nullam cursus orci metus, ut tempor elit elementum ut. Orci varius natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. In leo neque, feugiat vel diam vel, molestie cursus nisi. Ut ut bibendum erat. Curabitur at lectus lobortis, molestie orci vel, vulputat sem.</p></p>
URL	lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-elit

Figure 27.7 Page details

Deleting a page

Click on the Pages link in the upper menu. Then select the page you want to delete and click on the Delete button in its action columns. The system will ask you if you are sure that you want to delete the page, click on the OK button. The page we have selected has now been deleted.

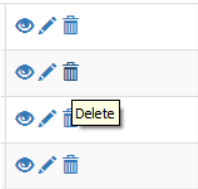


Figure 27.8 Page deletion button

Articles

You can manage your articles in the Admin panel selected from the top menu. With the management option you can add new content, edit, view details and delete articles that are no longer needed.

Article viewing

You can browse the articles available in the system and perform actions on them, such as adding a new one, editing, viewing details and deleting those already in the system. Select **Articles** from the top menu.

[Home](#)
[Admin](#)
[Articles](#)

Create article

Showing 1-20 of 67 items.

#	ID	Title	Author	Date	
1	1	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit
2	2	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit
3	3	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit
4	4	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit
5	5	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit
6	6	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit
7	7	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit
8	8	Lorem ipsum dolor sit amet, consectetur adipiscing elit.	Lukas Sosna	2018-02-18 10:45:18	View Edit

Figure 27.9 Viewing the articles

Addition of an article

You can add an item by selecting **Article** from the top menu. The page contains the **Create an article** button, which we also click. On the next page there is a form with the following fields: **Title** - intended for the title of the article, **Content** - enter the content of the article, **Author** - intended for entering the author, **Date** - date of publication of the article, click in the field to show the calendar in order to facilitate inserting the date in the correct format, **URL** - part of the URL created automatically from the title. When you have finished filling in the fields, click on the **Create article** button.

Home / Admin / Articles / Create article

Create article

Title

Content

body

Author

Date

URL

Create article

Figure 27.10 Article creation form

Editing an article

You can edit the selected article by clicking on the **Article** item in the main menu. Then, from the table below, click on the **Update** icon next to the selected item. The next page contains a form with the following fields: **Title** - intended for the title of the article, **Content** - enter the content of the article, **Author** - intended for entering the author, **Date** - date of publication of the article, after clicking in the field a calendar will appear in order to facilitate inserting the date in the correct format, **URL** - a fragment of the URL created automatically from the title. When you have finished filling in the fields, click on the **Update article** button.

Home / Admin / Articles / Update article: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Update article: Lorem ipsum dolor sit amet, consectetur adipiscing elit.

Title

Content

body

Author

Date

URL

Update article

Figure 27.11 Editing the content of an article

Article details

The full text of the article together with all the data can be seen in the detail view. Click on **Articles** in the upper menu. Then on the page next to the articles in the action column there is the button **View details** that we click. The page with the details of the text also contains buttons that allow you to edit and delete the article.



Figure 27.12 Article details

Removal of article

You can also remove items from our system by clicking on the **Articles** option from the main menu. Then, from the table containing the texts on the right, in the action column, click the **Delete** button next to the article. The system will ask you if you are sure you want to delete it. Click the **OK** button. The article has now been deleted.

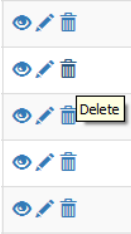


Figure 27.13 Article deletion button.

Blog

The blog contains entries in our diary in which we can describe our hobbies, interests or various other elements. Thanks to its structure, entries can be grouped into categories and tags. Administration allows you to add a new entry and edit, view and delete details.

Viewing entries

You can browse the entries by clicking on the **Blog** link in the upper menu. The sub-menu will develop and from there we will again choose the Blog position. On the website there are available entries in our blog together with actions such as adding a new one, editing, details or deleting an already placed entry.



















Home / Admin / Blog				
Blog				
Create entry				
Showing 1-20 of 38 items.				
#	ID	Title	Publication date	
1	7	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium.	2018-02-18 10:44:27	  
2	8	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium.	2018-02-18 10:44:27	  
3	9	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium.	2018-02-18 10:44:27	  
4	10	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium.	2018-02-18 10:44:27	  
5	11	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium.	2018-02-18 10:44:27	  
6	12	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium.	2018-02-18 10:44:27	  

Figure 27.14 Viewing blog entries

Addition of an entry

Adding a new blog entry is done by clicking on **Blog** in the main menu, from the sub-menu choose **Blog** again. The page that we call contains the Create an entry button.

The form you can use to add a new entry contains fields: **Title** - for the title of the entry, **Content** - content of the entry in the blog, **Publication date** - date of entry in the blog, when you go to this field and click on it, calendar help will appear to make it easier to define the date, **URL** - part of the URL that will be automatically created from the title, **Category** - category to which our entry is assigned, **Tag** - space for tags separated by a space sign. When we have finished writing out the fields, click on the button **Create entry**.

Home / Admin / Blog / Create entry
Create entry
Title
Content
Publication date
URL
Category
Tag
Create entry

Figure 27.15 Addition form for an entry

Editing an entry

All entries in the blog can be edited. This will be done by clicking on **Blog** in the main menu, from the developed sub-menu choose **Blog** option. A list of entries will then be displayed to you. To the right of each entry there is an action column. Select an entry and click Update.

The form used to edit an entry contains fields: **Title** - for the title of the entry, **Content** - content of the entry in the blog, **Publication date** - date of entry in the blog, when you go to this field and click on it, calendar help will appear to make it easier to define the date, **URL** - part of the URL that will be automatically created from the title, **Category** - category to which our entry is assigned, **Tag** - space for tags separated by a space sign. When you have finished deleting the fields, click on the **Update entry** button.

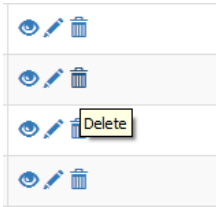


Figure 27.18 Delete entry button

Categories of blog

Categories are designed to divide our blog into sections thanks to which you can easily find the entries we are looking for thematically. You can add, edit, view or delete categories.

Category overview

You can browse by selecting **Blog** option from the main menu and **Blog categories** option from the sub-menu. We are currently in the category view. There is a button to add a new category here. For the categories on their right side, there are buttons such as: edit, see details or delete.

Home / Admin / Blog categories

Blog categories

Create categories

Showing 1-2 of 2 items.







#	ID	Title	Url	
1	2	Category 2	category-2	  
2	5	Category 1	category-1	  

Figure 27.19 Viewing the blog category

Adding a category

You can add a new category by clicking on **Blog** in the menu. Then you can select **Blog categories** of blog from the list. On the new page click on the button **Create categories**. We will be transferred to the page where you can find a special form. Form fields: **Title** - is used to define the name of the category, **Url** - contains part of the URL of the category, it will be automatically created from the text entered by us in the field with the title. After filling in the fields, click on the button **Create categories**.

Home / Admin / Blog categories / Create categories

Create categories

Title

Url

Create Categories

Figure 27.20 Category addition form

Category editing

You can edit the category of entries in the blog by clicking on **Blog** in the main menu, from the submenu choose the option **Blog category**. On the next page, select the category you want to edit and click on its right hand side in the action column on the **Update** icon. A form will appear which fields: **Title** - is used to define the name of the category, **Url** - contains part of the URL of the category, it will be automatically created on the basis of the text entered by us in the field with the title. After filling in the fields, click on the **Update category** button.

Home / Admin / Blog categories / Update

Update category: Category 2

Title

Uri

[Update category](#)

Figure 27.21 Edit form for the selected category

Category detail

To obtain full data on the category, click on the menu **Blog**, then from the sub-menu select the option **Blog category**. On the next page from among the category suggestions choose one whose details we want to learn and then click on the link **View** in the action column. In addition to the details, there are also buttons for updating the category and for deleting it.

Home / Admin / Blog categories / Category 2

Category 2

[Update category](#) [Delete category](#)

ID	2
Title	Category 2
Uri	category-2

Figure 27.22 Viewing details of categories

Deleting a category

You can delete a category by clicking **Blog** in the main menu and then selecting **Blog categories** from the sub-menu. In the list of all categories you can find the one you want to delete and then click on the **Delete** link in the action column on the right. The system will ask you if you want to delete the item for sure. Click on the **OK** button.

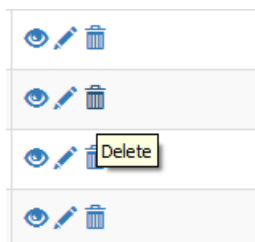


Figure 27.23 Category deletion button

Attention! If you delete a category, all entries that belong to the category will also be deleted!

Users

The system is equipped with the ability to create accounts and all the elements related to their operation, such as logging in, logging out, password reminder or filling in the profile. The administrator has the ability to add, edit and delete user accounts. Additionally, it can check when an e-mail reminding the user of the password was sent and check the logs where all the operations performed by the administrators were listed.

Viewing users

Select the Users item in the upper menu, click on the **Users** option from the sub-menu. On the website there are visible users in our system and the ability to add a new, edit an already existing account, see details or delete a user.






Home / Admin / Users						
Users						
Add user						
Showing 1-2 of 2 items.						
#	ID	E-mail	Name and surname	Registration date	IP registration	
1	1	lukasz@lukasz.sos.pl	Lukas Sosna	2018-02-03 07:18:26	127.0.0.1	 
2	24	support@phpbluedragon.eu		2018-04-01 15:09:57	83.4.207.236	  

Figure 27.24 Users in the system

Adding a user

In the main menu click on the position **Users**, from the sub-menu select again the option **Users**. On the website there is a button **Add user**, which we click. We are moved to a page where you can define a new account. The form contains the following fields: **E-mail** - unique and correct e-mail address of the user, **Name and surname** - name of the user, **Phone** - contact phone, **Website** - address of the website which the user has, **Information** - any additional information about the user, **Administrator** - whether the user will have the rights of the administrator. After filling in the form, click on the **Create** button.

Home / Admin / Users / Add user
Add user
E-mail
<input type="text"/>
Name and surname
<input type="text"/>
Phone
<input type="text"/>
Website
<input type="text"/>
Information
<input type="text"/>
Administrator
<input type="text" value="Yes"/>
Create

Figure 27.25 User addition form

Attention! The user created using the form has an active account - he does not have to activate it via a special link sent to the e-mail address.

Editing a user

Each user's data can be edited using the form to change it. From the top menu select the option **Users**, from the sub-menu click on the item **Users**. Go to the page with the listed accounts. The **Update** icon is located on the right hand side of the table.

The form contains the following fields: **Name and surname** - user's name and surname, **Phone** - contact phone, **Website** - user's website address, **Information** - any additional information about the user, **Administrator** - whether the user will have administrator's rights. After filling in the form, click on the **Edit** button.

Home / Admin / Update user / Update user: support@phpbluedragon.eu

Update user: support@phpbluedragon.eu

Name and surname

Phone

Website

Information

Administrator

Yes

Edit

Figure 27.26 User edit form

User details

Details of the user account can be found by clicking on the menu **Users**, then from the sub-menu select **Users**. On the page there are user accounts in the system, we select the account whose data we want to see and click on the right hand side in the action column on the icon **View**. The page will show us details about the user account and contains additional buttons to edit the data and delete the account from the page.

Home / Admin / Users / lukasz@lukasz.sos.pl

lukasz@lukasz.sos.pl

Update user

ID	1
E-mail	lukasz@lukasz.sos.pl
Name and surname	Lukas Sosna
Phone	
Website	http://en.lukasz.sos.pl
Information	
Password	\$2y\$10\$00.1ukLMSTYAUQhP69Sep06RY/WHuXSucQJwE5feg03ohMa3a
Key	dvmE9w4yukarigmsx9jq
Registration date	2018-02-03 07:18:26
IP registration	127.0.0.1
Active	y
Activation date	2018-02-03 09:22:44
IP activation	127.0.0.1
Administrator	y

Figure 27.27 User account details

Deleting a user

We will delete the user account by clicking on the menu **Users**, from the sub-menu we will select the option **Users**. Then on the page we browse the list of created accounts and select the ones we want to delete. Click on the **Delete** icon to the right of the screen. The system will ask you if you are sure that you want to delete your account. Click on the **OK** button.

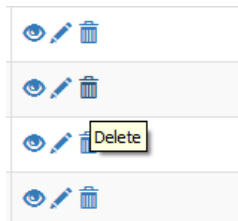


Figure 27.28 Icon to delete account.

Attention! User with identifier 1 cannot be deleted. It is the main administrator of the system.

Password reminder attempts

Every user has the right to use the password reminder option for their account. All such attempts are recorded in special logs. This way you can check if there has been an attempted break-in to someone's account. Click on the **Users** menu, select **Change password** from the submenu. You will see a table in which you can see: your user ID, two passwords used for validation, the date you filled out the form, the date you clicked on the link in the email with a password reminder, the IP address from which it was done, and information about the fact you used the reminder option if all operations were done correctly.

Home / Admin / Password reminder

Password reminder

Showing 1-3 of 3 items.

#	ID	User ID	Hash1	Hash2	Generated date	Date of use	IP	Used
1	3	24	fqce1bfcunuqtkpompl	bx6aawhpw0pikpg5fko	2018-04-01 15:34:37	2018-04-01 15:38:11	83.4.207.236	y
2	2	22	3wca3awqaxiind7k63pu	ykvylkice65wctfkyk8i	2018-02-26 21:22:26	2018-02-26 21:26:54	83.5.166.21	y
3	1	1	ws6rlpwd47vft7ro9dn2	few2yao4icce49tbvmle	2018-02-20 14:09:45	0000-00-00 00:00:00		

Figure 9.29 Table of attempted password reminders

System logs

Logs are used to track the work of the administrator, so that you can check which administrator has performed the operation in the service. In order to display the details click on the menu **Users**, and from the sub-menu select the option **Logs**. On the website there are available data such as: e-mail address of the user, action he performed, time and IP address.

Home / Admin / Application logs

Application logs

Showing 1-20 of 27 items.

#	ID	User	Action	Time	IP
1	27	lukasz@lukasz.sos.pl	Browsed system logs	2018-04-01 17:29:11	83.4.211.95
2	26	lukasz@lukasz.sos.pl	Browse requests for password remind	2018-04-01 17:25:58	83.5.190.179
3	25	lukasz@lukasz.sos.pl	Browsed one user	2018-04-01 17:22:32	83.7.135.163
4	24	lukasz@lukasz.sos.pl	Browsed users	2018-04-01 17:22:25	83.7.135.163
5	23	lukasz@lukasz.sos.pl	Browsed users	2018-04-01 17:18:40	83.4.247.139
6	22	lukasz@lukasz.sos.pl	Browsed users	2018-04-01 17:18:29	83.4.247.139
7	21	lukasz@lukasz.sos.pl	Browsed users	2018-04-01 17:13:32	83.7.135.104
8	20	lukasz@lukasz.sos.pl	Browse one blog category	2018-04-01 17:08:53	83.4.76.180
9	19	lukasz@lukasz.sos.pl	Browse the categories of the blog	2018-04-01 17:08:50	83.4.76.180

Figure 27.30: Administrator action logs

Contact

The system has a form by means of which it is possible to establish contact with persons responsible for a given website or the company to which it belongs.

Viewing addresses

All added addresses can be viewed by clicking on the **Contact** item from the main menu. The page contains all e-mail addresses with the possibility of adding a new one and editing, viewing details or deleting.

Home / Admin / Contact

Contact

[Add contact](#)

Showing 1-1 of 1 item.



#	ID	To	Recipient	
1	1	lukasz@lukasz.sos.pl	Technical assistance	 

Figure 27.31 E-mail addresses from the contact section

Adding an address

You can add a new address by clicking on **Contact** from the main menu. On the next page, click on the **Add contact** button. We go to the form which fields: **To** - e-mail address, **Recipient** - description of the contact person. Click on the **Create** button.



Home / Admin / Contact / Create contact

Create contact

To

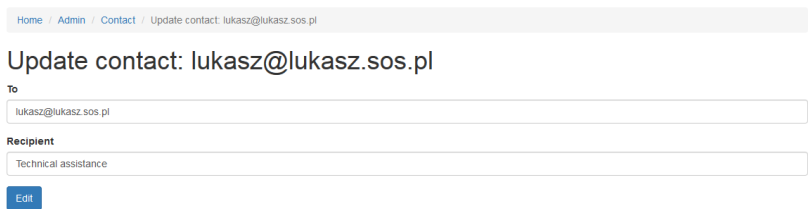
Recipient

Create

Figure 27.32: E-mail Addition Form

Address editing

The system has also been equipped with the possibility of editing e-mail addresses, which we will do by clicking on the **Contact** button in the upper menu. Next, from the table in which the addresses to contact were defined, select **Update** option from the right-hand side of the table in which the addresses to contact were defined. We go to the form which fields: **To** - e-mail address, **Recipient** - description of the contact person. Click on the **Edit** button.



Home / Admin / Contact / Update contact: lukasz@lukasz.sos.pl

Update contact: lukasz@lukasz.sos.pl

To

lukasz@lukasz.sos.pl

Recipient

Technical assistance

Edit

Figure 27.33 Editing contact details

Address details

You can view the details of a given contact by clicking on the **Contact** item in the main menu. Then on the page we choose the address, the details of which we are interested in. Click on the **View details** option in the action column. In addition to the details, the buttons for editing and deleting the contact are also displayed.



Home / Admin / Contact / lukasz@lukasz.sos.pl

lukasz@lukasz.sos.pl

Update contact Delete contact

ID	1
To	lukasz@lukasz.sos.pl
Recipient	Technical assistance

Figure 27.34 Selected contact details

Address deletion

The system allows us to delete each of the contacts defined by us. Click on **Contact** in the main menu. Then select the contact you want to delete from the table and click on the **Delete** button in the action column. The system will ask us if we are sure of our action. Confirm by clicking on the **OK** button.

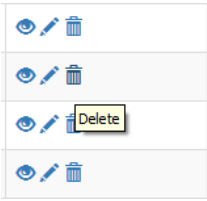


Figure 9.35 Contact deletion button

Download

Downloading is a place to place files in our system that we make available to others. Every time you download a file, your data will be saved on the system.

Viewing files



















Downloadable files can be viewed by clicking on the **Download** option in the main menu, and from the sub-menu select **Download** again. The site contains files in the section with the ability to add a new item, edit, view details or delete the file.

Home / Admin / Download

Download

Add file

Showing 1-6 of 6 items.

#	ID	Title	File	Version	
1	1	PHP programme	file.zip	1.0.0	  
2	2	PHP programme	file.zip	1.0.0	  
3	3	PHP programme	file.zip	1.0.0	  
4	4	PHP programme	file.zip	1.0.0	  
5	5	PHP programme	file.zip	1.0.0	  
6	6	PHP programme	file.zip	1.0.0	  

27.36 View of downloads

Adding a file

Adding a new file to the system is possible by selecting the option **Download** from the menu, from the sub-menu select **Download**. On the new page, click on the **Add file** button. The form contains the following fields: **Title** - title of the program we are adding, **Description** - summary of what the file is about, **File** - name of the file, **Version** - version number of the file, **File size** - size of the file on disk, **License** - license on which we are making the indicated file available. Click on the **Add file** button.

Home / Admin / Download / Add file

Add file

Title

Description

File

Version

File size

License

Add file

Figure 27.37 File insertion form

Attention! The File field must contain the exact name of our file, which we then place in the getfiles directory on the root directory of our website.

Download file editing

You can edit the file by clicking on **Download** in the menu and then selecting **Download** from it. A table with the files is displayed in front of us. Select the file and click the **Update** button in the action column on the right. The form contains the following fields: **Title** - title of the program we are adding, **Description** - summary of what the file is about, **File** - name of the file, **Version** - version number of the file, **File size** - size of the file on disk, **License** - license on which we are making the indicated file available. Click the **Update file** button.

Home / Admin / Download / Update file: PHP programme

Update file: PHP programme

Title

Description

File

Version

File size

License

Update file

Figure 27.38: File Edit Form

Attention! The File field must contain the exact name of our file, which we then place in the getfiles directory on the root directory of our website.

File detail

You can browse the details of the file by clicking on the **Download** option in the menu, and from the sub-menu select **Download**. From the list of files presented by the system we select the item we are interested in and click on the **View** details icon. In addition to file data, the website also contains buttons for editing and deleting the item.

Home / Admin / Download / PHP programme

PHP programme

Add fileDelete file

ID	1
Title	PHP programme
Description	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Maecenas id nulla nec dui efficitur pretium. Suspendisse odio magna, aliquam sed nulla sed, rutrum auctor neque. Pellentesque tincidunt massa eget tellus vehicula malesuada. Pellentesque accumsan aliquet sagittis. Quisque mollis leo faucibus cursus consequat. Morbi elementum, velit eu aliquet pulvinar, quam ex lobortis lectus, ut blandit risus augue a felis. In gravida varius arcu, at vulputate purus eleifend vestibulum.
File	file.zip
Version	1.0.0
File size	23 MB
License	freeware

Figure 27.39: One file details

Deleting a file

All files available on the system can be deleted when we no longer want to have them on the system. Click on the **Download** menu and from the sub-menu select the **Download** option. Next, from the table with files select the item you want to delete and click on the **Delete** button in the right-hand column of the action. The system will ask you if you would like to delete the selected file, click on the **OK** button.













  
  
  
  

Figure 27.40: Filename deletion button

Download statistics

Presentation of download statistics is available by clicking on the option **Download** in the main menu and then selecting the option **Statistics** from the sub-menu. On the website we have at our disposal columns presenting data such as user's browser, file, date and time of downloading and user's IP address.

Home / Admin / Download statistics

Download statistics

Showing 1-4 of 4 items.

#	ID	Browser	File	Date of download	IP
1	4	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59	file.zip	2018-04-01 14:22:33	83.4.207.236
2	3	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:59.0) Gecko/20100101 Firefox/59	file.zip	2018-04-01 14:22:09	83.4.207.236
3	2	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:58.0) Gecko/20100101 Firefox/58	file.zip	2018-02-20 15:11:04	83.7.129.242
4	1	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:58.0) Gecko/20100101 Firefox/58	file.zip	2018-02-20 14:45:55	83.7.129.242

Figure 27.41 Download statistics

Page menu

Menu located on the left side of our website where it shows all the links that we want you to see. We can add any section to it, or even one element of the structure.

Viewing the menu

Select the **Left menu** option from the main menu. We will be taken to the page where you can find the wiring diagram on our website. You will see the main menu items, menu sub-menu items, information whether the user must be logged in to see the item, the content to which it leads, and a link to delete it.

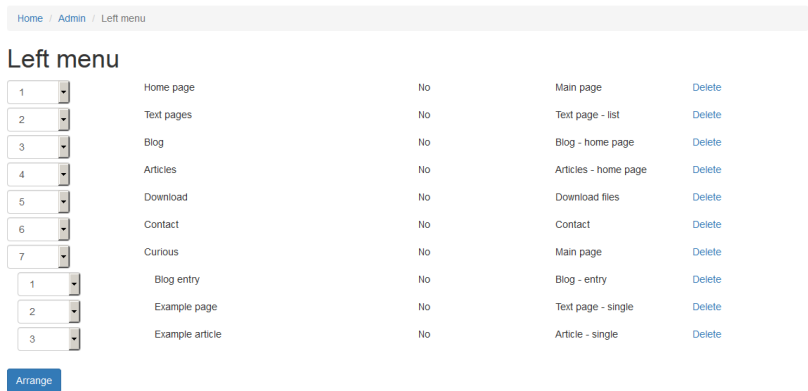


Figure 27.42 Menu administration

Adding menus

A new menu item can be added by clicking on the **Left menu** item at the top of the page. Then scroll through the go page of the **Add menu item** header. The Title field is used for the name that will be presented in the menu. **Type of destination** - choose from it to which element the link should be connected. **Destination page** - contains individual elements such as articles, blog entries or texts. The next field is **Sub-page** - it informs if the page is a subpage of another page in our menu. When you want to create a nested menu, select the parent element from this field. Last field **Visible to logged in** - defines whether a given element is visible to the user who logged into his account. After filling in the fields, click on the **Add** button.



Figure 27.43 Adding a new menu item

Deleting the menu

Click on the **Left menu** item at the top. Then from the table with the menu contents click on the **Delete** link on the right side of the item. The system will ask you if you are sure that you want to delete the item, click **OK**.

Delete

Delete

Delete

Delete

Figure 27.44 Link to remove

Arrange the menus in the correct order.

The menu should be arranged according to the order in which you want to store the links in it. Click on the **Left menu** option in the upper bar. Next, in the table with the contents in the first field, we select the positions in which the links are to be placed. After selecting, click on the **Arrange** button.

Left menu

1

2

3

4

5

6

7

1

2

3

Home page

Text pages

Blog

Articles

Download

Contact

Curious

Blog entry

Example page

Example article

Arrange

Figure 27.45 Arranging menu items

Errors 404

It is good to have information in the service about missing elements referred to by the service itself as well as the person entering our site. In this way, it is possible to quickly activate and improve unwanted links. Click the **Errors 404** option in the main menu and you will find a list of such actions on the website. The column **Link on page** - defines the page from which the selected content was accessed, **Page** - contains the address which does not exist in our service and must be corrected, the **Data** column - contains information about when the error occurred.

Home / Admin / Errors 404

Errors 404

Showing 1-8 of 8 items

#	ID	Link on page	Page	Date
1	18	null	http://yii.phpblue.dragon.eu/category/category-2/	2018-04-01 15:22:56
2	17	http://yii.phpblue.dragon.eu/download	http://yii.phpblue.dragon.eu/getfiles/file.zip	2018-04-01 14:22:09
3	8	http://yii.phpblue.dragon.eu/library/query-ui.css	http://yii.phpblue.dragon.eu/library/images/ui-bg_glass_55_b9f9e6_1x400.png	2018-02-21 11:05:42
4	7	http://yii.phpblue.dragon.eu/library/query-ui.css	http://yii.phpblue.dragon.eu/library/images/ui-bg_glass_75_e6e6e6_1x400.png	2018-02-21 11:05:42
5	6	http://yii.phpblue.dragon.eu/library/query-ui.css	http://yii.phpblue.dragon.eu/library/images/ui-icons_222222_256x240.png	2018-02-21 11:05:42
6	3	null	http://yii.phpblue.dragon.eu/article/lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-ellit	2018-02-20 12:00:22
7	2	null	http://yii.phpblue.dragon.eu/page/lorem-ipsum-dolor-sit-amet-consectetur-adipiscing-ellit	2018-02-18 12:26:31
8	1	null	http://yii.phpblue.dragon.eu/root	2018-02-18 11:11:50

Figure 27.46 Missing URLs

Logout

When the work in the admin panel is finished, it is very important to log out of the system, i.e. end the session. Click on the **Logout** option in the upper menu.

Summary

I hope that this book will help you in programming using the **Yii framework**. I presented the technical side of creating applications from A to Z. I presented the framework download, ways of its installation, configuration, adding database support, I showed how to create controllers, models and views, I taught how to create components and add languages.

I created this book mainly for the purpose of educating programmers for whom the pure manual available on the Yii framework website is not enough and would like to have a ready example where they can learn how to use the software itself.

When I started programming, **PHP** version 3 was in force, where most of the functions created to manage the database, files and user-friendly URLs did not exist. Much has changed, especially in making it easier for developers programming in PHP to create new applications by enriching the interpreter itself, but also in creating programs such as the framework, which the programming of applications reduce only to creating the elements that the client wanted. Thanks to the availability of a mass of functions and methods, programming of the application becomes very easy. Instead of focusing on creating an entire application frame, the framework allows you to create only the heart of the application.

I recommend it to all people who create small **CodeIgniter** framework services, but if you are going to create a medium or large website, then I recommend **Yii**.

Let the force be with you!