Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## РГЗ по веб-программированию на тему: «Использование шаблона проектирования MVC при разработке веб-приложений»

**Выполнил:**  
студент группы ПВ-41  
Адаменко И. И.

**Проверил:**  
старший преподаватель  
Гарибов А. И.

Белгород  
2015

**Цель работы:** изучить шаблон проектирования MVC и его применение в качестве архитектурного каркаса в веб-приложениях.

# Задание

Разработать веб-приложение на языке PHP или C# (ASP.NET) на выбранную тему. В качестве архитектурного каркаса использовать подход MVC.

# Теоретическая часть

MVC — это фундаментальный шаблон проектирования, который нашел применение во многих технологиях, дал развитие новым технологиям и каждый день облегчает жизнь разработчикам. Расшифровывается как Model-View-Controller, или по-русски Модель-Представление-Контроллер.

Впервые паттерн MVC появился в языке SmallTalk. Разработчики должны были придумать архитектурное решение, которое позволяло бы отделить графический интерфейс от бизнес-логики, а бизнес-логику от данных. Таким образом, в классическом варианте, MVC состоит из трех частей, которые и дали ему название. Рассмотрим их подробнее.

## Модель

Под моделью, обычно понимается часть, содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму модель.

Модель обладает следующими признаками:

* Модель — это бизнес-логика приложения;
* Модель обладает знаниями о себе самой и не знает о контроллерах и представлениях;
* Для некоторых проектов модель — это просто слой данных (DAO, база данных, XML-файл);
* Для других проектов модель — это менеджер базы данных, набор объектов или просто логика приложения.

## Представление

В обязанности представления входит отображение данных полученных от модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

Представление обладает следующими признаками:

* В представлении реализуется отображение данных, которые получаются от модели любым способом;
* В некоторых случаях, представление может иметь код, который реализует некоторую бизнес-логику.

Примеры представления: HTML-страница, WPF-форма, Windows Form.

## Контроллер

Контроллер обеспечивает связь между пользователем и системой: контролирует ввод данных пользователем и использует модель и представление для реализации необходимой реакции.

Признаки контроллера:

* Контроллер определяет, какие представление должно быть отображено в данный момент;
* События представления могут повлиять только на контроллер, а он может повлиять на модель и определить другое представление;
* Возможно несколько представлений только для одного контроллера.

## Взаимодействие

Контроллер перехватывает событие извне (от пользователя) и, в соответствии с заложенной в него логикой, реагирует на это событие изменяя модель, посредством вызова соответствующего метода. После изменения модель использует событие о том, что она изменилась, и все подписанные на это события представления, получив его, обращаются к модели за обновленными данными, после чего их и отображают.

Существует также несколько вариантов того, как именно должна вести себя модель после обновления данных. Например, она может не оповещать представления самостоятельно, а просто возвращать новые данные контроллеру, который и будет передавать их представлениям (такой способ взаимодействия реализован в этой работе).

Таким образом, реализацию MVC в рамках этой работы можно описать с помощью следующей схемы:

Пользователь

Контроллер

Представление

Модель

Помимо описанного выше, иногда в схему также включается база данных, которую использует модель, однако, мы этого делать не будем, по причине того, что при правильной программной реализации шаблона вместо базы данных может быть любой объект, который эти данные хранит, и ни контроллер, ни представление знать о нём не должны. Чаще всего о нём не знает даже модель, поскольку использует какой-то специальный объект для взаимодействия с БД, который просто получает и отдаёт данные, реализуя тем самым для всего приложения нечто вроде чёрного ящика.

## Другие варианты реализаций

Помимо стандартного MVC, описанного выше, есть много других реализаций, самые известные из них — это MVVM и MVP.

* MVP (Model-View-Presenter) — реализует двустороннюю коммуникацию уровня Presenter с уровнем View. То есть, View напрямую взаимодействует с Presenter, путём вызова соответствующих функций или событий его экземпляра. Таким образом, один экземпляр Presenter всегда связан с одним View.
* MVVM (Model-View-View Model) — как и MVP реализует двустороннюю коммуникацию View Model-View, однако, View Model не имеет ссылки на интерфейс View, так как изменение состояния View Model автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (binding).

# Практическая часть

В качестве темы этой работы был выбран блог на языке PHP. В этом языке, в отличие от ASP.NET нет встроенных средств для реализации MVC, однако, используя работу с классами и включение одних файлов в другие, этого можно добиться.

Общая структура получившегося приложения выглядит следующим образом:

* app — каталог с основной логикой приложения;
  + config — каталог с конфигурационными файлами;
  + controller — каталог с контроллерами;
  + core — каталог с основными классами приложения;
  + libs — каталог с дополнительными библиотеками;
  + model — каталог с моделями;
  + view — каталог с представлениями;
* public — каталог со статическими файлами приложения и точкой входа (index.php).

## Точка входа

С помощью конфигурации сервера, все запросы в корень сайта перенаправляются в файл public/index.php, который содержит следующий код:

1. <?
2. // constant that holds the project's folder path, like "/var/www/".
3. // DIRECTORY\_SEPARATOR adds a slash to the end of the path
4. define('ROOT', dirname(\_\_DIR\_\_) . DIRECTORY\_SEPARATOR);
5. // constant that holds the project's "app" folder, like "/var/www/app".
6. define('APP', ROOT . 'app' . DIRECTORY\_SEPARATOR);
8. // load application config
9. **require** APP . 'config/config.php';
11. // load class for DB CRUD
12. **require** APP . 'libs/db.php';
14. // load helpers
15. **require** APP . 'libs/helper.php';
17. // load application class
18. **require** APP . 'core/app.php';
20. // load model class
21. **require** APP . 'core/model.php';
23. // load controller class
24. **require** APP . 'core/controller.php';
26. // start the application
27. $app = **new** Application();
28. ?>

В нём подключаются все необходимые файлы для работы приложения, а также создаётся экземпляр класса Application. Код описания класса содержится в файле app/core/app.php и представлен ниже:

1. <?
2. **class** Application
3. {
4. **private** $url\_controller = null;
5. **private** $url\_action = null;
6. **private** $url\_params = **array**();
8. **public** **function** \_\_construct()
9. {
10. // create array with URL parts in $url\_\*
11. $this->splitUrl();
13. // check for controller: no controller given? then load home-page
14. **if** (!$this->url\_controller) {
15. **require** APP . 'controller/home.php';
17. $page = **new** Home();
18. $page->index();
20. } **elseif** (file\_exists(APP . 'controller/' . $this->url\_controller . '.php')) {
21. // here we did check for controller: does such a controller exist?
23. // if so, then load this file and create this controller
24. **require** APP . 'controller/' . $this->url\_controller . '.php';
26. $this->url\_controller = **new** $this->url\_controller();
28. // check for method: does such a method exist in the controller?
29. **if** (method\_exists($this->url\_controller, $this->url\_action)) {
31. **if** (!**empty**empty($this->url\_params)) {
32. // call the method and pass arguments to it
33. call\_user\_func\_array(**array**($this->url\_controller,  
     $this->url\_action),  
     $this->url\_params);
34. } **else** {
35. // if no parameters are given, just call the method without parameters,  // like $this->home->method();
36. $this->url\_controller->{$this->url\_action}();
37. }
39. } **else** {
40. **if** (strlen($this->url\_action) == 0) {
41. // no action: call the default index() method of a selected controller
42. $this->url\_controller->index();
43. }
44. **elseif** (is\_numeric($this->url\_action)) {
45. // is action numeric? call method byID (for instance for /post/@ID)
46. $this->url\_controller->byID($this->url\_action);
47. }
48. **else** {
49. header('location: ' . URL . 'error');
50. }
51. }
52. } **else** {
53. header('location: ' . URL . 'error');
54. }
55. }
57. /\*\*
58. \* Get and split the URL
59. \*/
60. **private** **function** splitUrl()
61. {
62. **if** (isset($\_GET['url'])) {
64. // split URL
65. $url = trim($\_GET['url'], '/');
66. $url = filter\_var($url, FILTER\_SANITIZE\_URL);
67. $url = explode('/', $url);
69. // put URL parts into according properties
70. $this->url\_controller = isset($url[0]) ? $url[0] : null;
71. $this->url\_action = isset($url[1]) ? $url[1] : null;
73. // remove controller and action from the split URL
74. unset($url[0], $url[1]);
76. // rebase array keys and store the URL params
77. $this->url\_params = array\_values($url);
78. }
79. }
80. }
81. ?>

В ходе разбора адреса, к которому пытается достучаться пользователь, мы получаем контроллер и его метод, который нужно вызвать. Все контроллеры наследуются от базового, который при создании экземпляра создаёт соединение к базе данных (с помощью класса DB из app/libs/db.php) и инициализирует необходимую ему модель. Код базового контроллера содержится в файле app/core/controller.php и представлен ниже:

1. <?
2. **class** Controller
3. {
4. **public** $db = null;
5. **public** $model = null;
7. /\*\*
8. \* whenever controller is created, open a database connection too and load "the model"
9. \*/
10. **function** \_\_construct()
11. {
12. $this->connectDB();
13. $this->loadModel();
14. }
16. /\*\*
17. \* open the database connection with the credentials from application/config/config.php
18. \*/
19. **private** **function** connectDB()
20. {
21. // create a database connection
22. $this->db = **new** DB(
23. DB\_NAME,
24. DB\_USER,
25. DB\_PASS,
26. DB\_HOST
27. );
28. }
30. /\*\*
31. \* Loads the "model".
32. \*/
33. **public** **function** loadModel()
34. {
35. // for each child model:
36. // $this->model = new Model($this->db);
37. }
38. }
39. ?>

В классе Application (код которого выше), можно заметить, что если пользователь заходит в корневой каталог, то в качестве контроллера выбирается Home. Его код расположен в файле app/controller/home.php и представлен ниже:

1. <?
2. **class** Home **extends** Controller
3. {
4. /\*\*
5. \* PAGE: index
6. \*/
7. **public** **function** index()
8. {
9. $title = 'Блог';
10. $posts = $this->model->getList(10);
12. **require** APP . 'view/\_templates/header.php';
13. **require** APP . 'view/home/index.php';
14. **require** APP . 'view/\_templates/footer.php';
15. }
17. **public** **function** loadModel()
18. {
19. **require** APP . 'model/posts.php';
21. $this->model = **new** Posts($this->db);
22. }
23. }
24. ?>

Как видно, этот контроллер реализует метод loadModel, который используется в конструкторе базового класса. В нём подключается необходимая модель и в поле model записывается экземпляр этой модели. В данном случае это модель Posts.

Все модели наследуются от одного базового класса Model, который расположен в файле app/core/model.php и состоит из одного конструктора, сохраняющего ссылку на соединение с БД:

1. <?
2. **class** Model
3. {
4. /\*\*
5. \* $db is database connection
6. \*/
7. **function** \_\_construct($db)
8. {
9. $this->db = $db;
10. }
11. }
12. ?>

Код самой модели Posts расположен в файле app/model/posts.php и представлен ниже:

1. <?
2. **class** Posts **extends** Model
3. {
4. /\*
5. \* Convert DB strings to post array
6. \*/
7. **function** preparePost($rec, $authors) {
8. **return** **array**(
9. 'id' => $rec['post\_id'],
10. 'link' => URL . 'post/' . $rec['post\_id'],
11. 'title' => $rec['title'],
12. 'author' => array\_key\_exists('username', $authors)
13. ? $authors['username']
14. : $authors[$rec['author\_id']],
15. 'excerpt' => nl2br(strip\_tags(mb\_substr($rec['post'], 0, 200,  
     'UTF-8')) . '..'),
16. 'text' => '<p>' . str\_replace('<br />', '</p><p>', nl2br($rec['post']))  
     . '</p>',
17. 'textRaw' => $rec['post'],
18. 'date' => date("d.m.y H:i", strtotime($rec['date']))
19. );
20. }
22. /\*
23. \* Convert DB strings to comment array
24. \*/
25. **function** prepareComment($rec) {
26. **return** **array**(
27. 'name' => $rec['name'],
28. 'text' => '<p>' . str\_replace('<br />', '</p><p>', nl2br($rec['comment']))  
     . '</p>',
29. 'email' => $rec['email'],
30. 'date' => date("d.m.y H:i", strtotime($rec['date']))
31. );
32. }
34. /\*
35. \* Get last $n posts from DB
36. \*/
37. **public** **function** getList($n) {
38. $posts = **array**();
39. $records = $this->db->Select('posts', false, 'post\_id DESC', $n);
41. $author\_ids = **array**();
43. **if** (is\_array($records[0])) {
44. $author\_ids = array\_unique(array\_map(**function**($o) {
45. **return** $o['author\_id'];
46. }, $records));
47. } **elseif** ($records) {
48. $author\_ids[] = $records['author\_id'];
49. }
51. $authors = **array**();
53. **foreach** ($author\_ids **as** $author\_id) {
54. $rec = $this->db->Select('authors', **array**(
55. 'author\_id' => $author\_id
56. ));
58. $authors[$author\_id] = $rec['username'];
59. }
61. **if** (is\_array($records[0])) {
62. **foreach** ($records **as** $rec) {
63. $posts[] = $this->preparePost($rec, $authors);
64. }
65. } **elseif**($records) {
66. $posts[] = $this->preparePost($records, $authors);
67. }
69. **return** $posts;
70. }
72. /\*
73. \* Get post from DB by ID
74. \*/
75. **public** **function** getByID($id) {
76. $record = $this->db->Select('posts', **array**(
77. 'post\_id' => $id
78. ));
80. **if** ($record) {
81. $author = $this->db->Select('authors', **array**(
82. 'author\_id' => $record['author\_id']
83. ));
85. $post = $this->preparePost($record, $author);
87. $dbcomments = $this->db->Select('comments', **array**('post\_id' => $id));
88. $comments = **array**();
90. **if** (!empty**empty**($dbcomments)) {
91. **if** (is\_array($dbcomments[0])) {
92. **foreach** ($dbcomments **as** $dbcomment) {
93. $comments[] = $this->prepareComment($dbcomment);
94. }
95. } **else** {
96. $comments[] = $this->prepareComment($dbcomments);
97. }
98. }
100. $tag\_ids = $this->db->Select(
101. 'post\_tags',
102. **array**('post\_id' => $id),
103. false,
104. false,
105. 'tag\_id'
106. );
108. $tags = **array**();
110. **if** (array\_key\_exists(0, $tag\_ids)) {
111. **foreach** ($tag\_ids **as** $tag\_id) {
112. $tag = $this->db->Select('tags', **array**(
113. 'tag\_id' => $tag\_id['tag\_id']
114. ));
116. $tags[] = $tag['name'];
117. }
118. } **elseif** ($tag\_ids) {
119. $tag = $this->db->Select('tags', **array**(
120. 'tag\_id' => $tag\_ids['tag\_id']
121. ));
122. $tags[] = $tag['name'];
123. }
125. $post['tags'] = implode(', ', $tags);
126. $post['comments'] = $comments;
128. **return** $post;
129. } **else** {
130. **return** null;
131. }
132. }
134. /\*
135. \* Add comment to post with ID == $postid
136. \*/
137. **public** **function** addComment($name, $postid, $comment, $email) {
138. $this->db->Insert('comments', **array**(
139. 'name' => $name,
140. 'post\_id' => $postid,
141. 'comment' => $comment,
142. 'email' => $email,
143. 'date' => date('Y-m-d H:i:s', time())
144. ));
145. }
147. /\*
148. \* Add post
149. \*/
150. **public** **function** addPost($title, $text, $tags, $login, $password)
151. {
152. $user = $this->db->Select('authors', **array**(
153. 'username' => $login,
154. 'password' => $password
155. ));
157. $this->db->Insert('posts', **array**(
158. 'title' => $title,
159. 'post' => $text,
160. 'author\_id' => $user['author\_id'],
161. 'date' => date('Y-m-d H:i:s', time())
162. ));
164. $id = $this->db->GetLastInsertID();
166. **if** (!empty**empty**($tags)) {
167. $tags = array\_filter(array\_unique(array\_map('trim', explode(',',  
      $tags))));
169. **foreach** ($tags **as** $tag) {
170. $found = $this->db->Select('tags', **array**(
171. 'name' => $tag
172. ));
174. **if** ($found) {
175. $this->db->Insert('post\_tags', **array**(
176. 'post\_id' => $id,
177. 'tag\_id' => $found['tag\_id']
178. ));
179. } **else** {
180. $this->db->Insert('tags', **array**(
181. 'name' => $tag
182. ));
184. $tagid = $this->db->GetLastInsertID();
186. $this->db->Insert('post\_tags', **array**(
187. 'post\_id' => $id,
188. 'tag\_id' => $tagid
189. ));
190. }
191. }
192. }
193. }
195. /\*
196. \* Update post
197. \*/
198. **public** **function** updatePost($id, $title, $text, $tags, $login, $password)
199. {
200. $user = $this->db->Select('authors', **array**(
201. 'username' => $login,
202. 'password' => $password
203. ));
205. $this->db->Update('posts', **array**(
206. 'title' => $title,
207. 'post' => $text,
208. 'author\_id' => $user['author\_id'],
209. 'date' => date('Y-m-d H:i:s', time())
210. ), **array**(
211. 'post\_id' => $id
212. ));
214. **if** (!**empty**empty($tags)) {
215. $tags = array\_filter(array\_unique(array\_map('trim', explode(',',  
      $tags))));
217. $this->db->Delete('post\_tags', **array**(
218. 'post\_id' => $id
219. ));
221. **foreach** ($tags **as** $tag) {
222. $found = $this->db->Select('tags', **array**(
223. 'name' => $tag
224. ));
226. **if** ($found) {
227. $this->db->Insert('post\_tags', **array**(
228. 'post\_id' => $id,
229. 'tag\_id' => $found['tag\_id']
230. ));
231. } **else** {
232. $this->db->Insert('tags', **array**(
233. 'name' => $tag
234. ));
236. $tagid = $this->db->GetLastInsertID();
238. $this->db->Insert('post\_tags', **array**(
239. 'post\_id' => $id,
240. 'tag\_id' => $tagid
241. ));
242. }
243. }
244. }
245. }
247. /\*
248. \* Delete post
249. \*/
250. **public** **function** deletePost($id) {
251. $this->db->Delete('posts', **array**(
252. 'post\_id' => $id
253. ));
254. }
255. }
256. ?>

Эта модель инкапсулирует в себе всю логику работы с записями в блоге. Также в контроллере Home подключается необходимое представление, которое является HTML-разметкой с небольшими шаблонными вставками на языке PHP. В этом примере оно состоит из трёх файлов, header.php и footer.php из app/view/\_templates и index.php из app/view/home. Код их представлен ниже в том порядке, в каком они подключаются:

header.php:

1. <!DOCTYPE html**>**
2. **<html** lang="ru"**>**
3. **<head>**
4. **<meta** charset="UTF-8"**>**
5. **<meta** name="viewport" content="width=device-width, initial-scale=1.0"**>**
6. **<title><?**=empty($title) ? "Блог" : $title**?></title>**
7. **<link** rel="stylesheet" href="<?=URL?>css/awsm.min.css"**>**
8. **</head>**
9. **<body>**
10. **<header>**
11. **<h1><?**=empty($title) ? "Блог" : $title**?></h1>**

index.php:

1. **<nav>**
2. **<ul>**
3. **<li><a** href="<?=URL?>post/add"**>**новый пост**</a></li>**
4. **<li><a** href="<?=URL?>user"**>**пользователи**</a></li>**
5. **<li><a** href="<?=URL?>tag"**>**теги**</a></li>**
6. **</ul>**
7. **</nav>**
8. **</header>**
9. **<main>**
11. **<?** **if** ($posts): **?>**
13. **<?** **foreach** ($posts as $post): **?>**
15. **<section>**
16. **<h2><a** href="<?=$post['link']?>"**><?**=$post['title']**?></a></h2>**
17. **<aside>**
18. **<?**=$post['author']**?>**
19. **<br>**
20. **<?**=$post['date']**?>**
21. **</aside>**
22. **<p><?**=$post['excerpt']**?></p>**
23. **</section>**
25. **<?** **endforeach** **?>**
27. **<?** **else:** **?>**
29. **<p><em>**Увы, но постов пока нет.**</em></p>**
31. **<?** **endif** **?>**
33. **</main>**

footer.php:

1. **<footer>**
2. **<p>**Игорь Адаменко**</p>**
3. **</footer>**
4. **</body>**
5. **</html>**

# Скриншоты



Рисунок 1. Главная страница блога

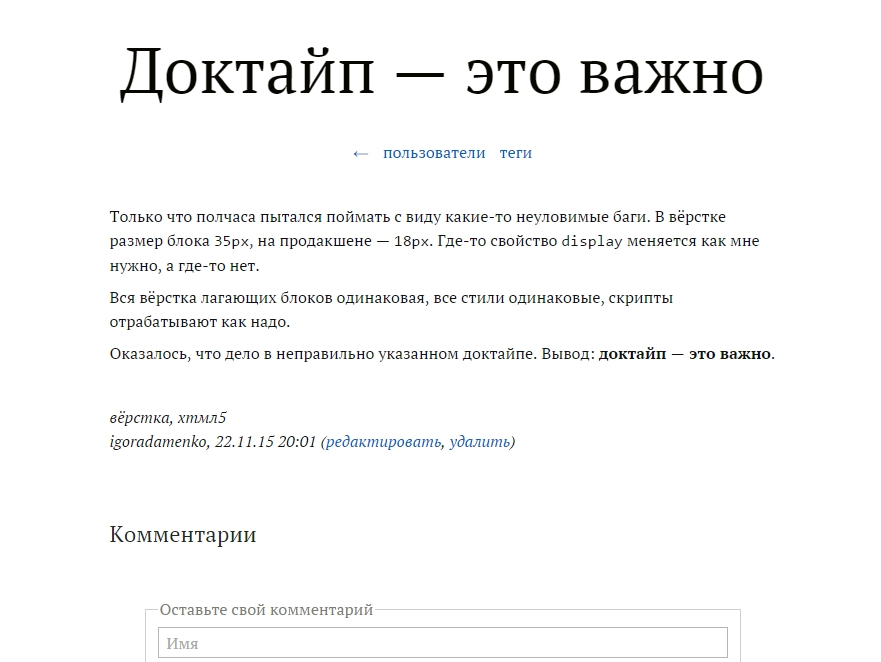


Рисунок 2. Страница записи в блоге

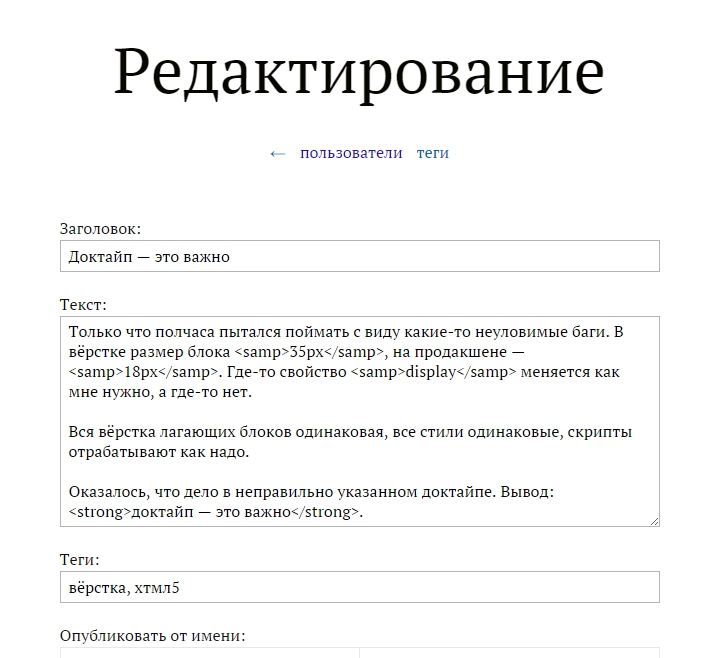


Рисунок 3. Страница редактирования записи

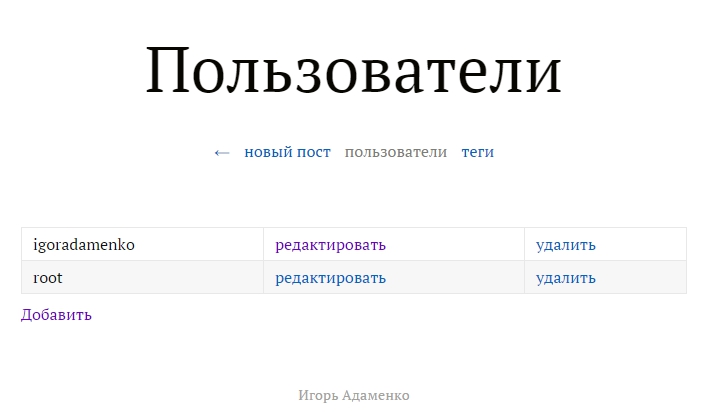


Рисунок 4. Страница со списком пользователей

# Вывод

В ходе этой работы я изучил шаблон проектирования MVC и создал веб-приложение для блога на языке PHP, использовав MVC в качестве архитектурного каркаса.