**22 Вставка с помощью Active Record**

Привет! В предыдущем уроке мы написали метод save(), который в зависимости от того, есть ли у объекта id, решает – обновить запись или создать новую. Он вызывает в свою очередь другие методы:

* update(), если id у объекта есть;
* insert(), если это свойство у объекта равно null.

В прошлом уроке мы реализовали метод update(), который позволяет обновлять уже существующие записи в базе. В этом уроке мы напишем метод insert(), который будет создавать новые записи в нашей базе данных.

Давайте добавим новый роут, с помощью которого будут создаваться новые статьи.

**src/routes.php**

<?php

return [

...

'~^articles/add$~' => [\MyProject\Controllers\ArticlesController::class, 'add'],

...

];

Добавим новый экшн в контроллере (не забудьте прописать неймспейс для модели User):

**src/MyProject/Controllers/ArticlesController.php**

public function add(): void

{

$author = User::getById(1);

$article = new Article();

$article->setAuthor($author);

$article->setName('Новое название статьи');

$article->setText('Новый текст статьи');

$article->save();

var\_dump($article);

}

И добавим сеттер автора в сущности Article:

**src/MyProject/Models/Articles/Article.php**

/\*\*

\* @param User $author

\*/

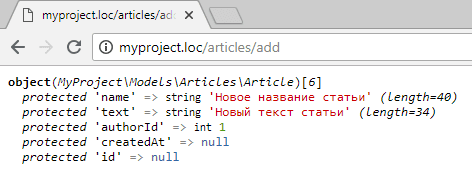
public function setAuthor(User $author): void

{

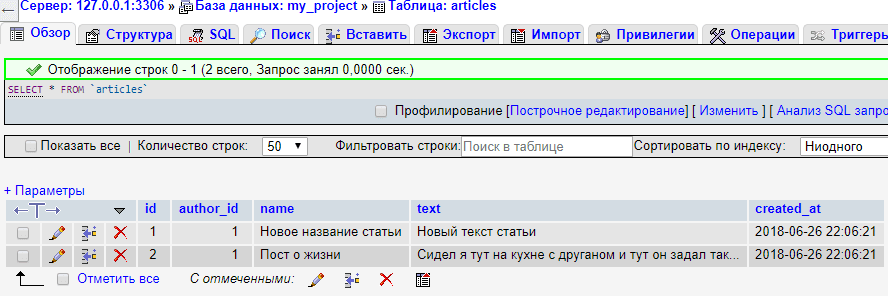
$this->authorId = $author->getId();

}

Если сейчас мы перейдём по новому адресу:  
<http://myproject.loc/articles/add>  
То увидим созданный нами объект.

[](https://php.zone/uploads/img/cc407cbe309d0cfc7a25e7255a18d3eb165455f0dcaf32f25fc9080732442d66.png)

Разумеется, в базе данных он сейчас не появится, так как метод insert() в классе ActiveRecordEntity сейчас пуст.

[](https://php.zone/uploads/img/a5054affac8695c6ce793a02867f00adceb5703a7fc8bb5ba4b4c60706643e00.png)

Самое время его написать :)

## Реализация метода insert()

Давайте рассмотрим синтаксис SQL-запроса, который нам нужно написать:

INSERT INTO <имя таблицы> (<имя столбца 1>, <имя столбца 2>,...) VALUES (<значение столбца 1>, <значение столбца 2>,…);

Для сущности Article запрос на вставку будет выглядеть вот так:

INSERT INTO `articles` (`author\_id`, `name`, `text`) VALUES (:author\_id, :name, :text)

А затем нам нужно будет передать массив с параметрами, вроде такого:

[':author\_id' => 1, ':name' => 'Название', ':text' => 'Текст']

Первым делом давайте посмотрим на то, какие данные вообще у нас есть:

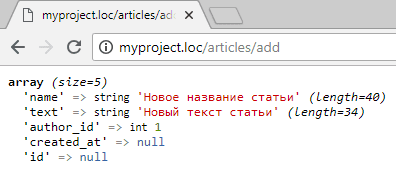
**src/MyProject/Models/ActiveRecordEntity.php**

private function insert(array $mappedProperties): void

{

var\_dump($mappedProperties);

}

[](https://php.zone/uploads/img/a670207f697117e64a1855e5406ed7605f562e243f8ea9402c4008f88b860b40.png)

Мы видим, что у нас есть 2 поля со значением null. Это поля id и created\_at. Поле id нам в запросе не нужно, так как для него будет автоматически выдано значение на уровне базы данных, так как оно типа AUTOINCREMENT, а для поля created\_at задано значение по умолчанию – CURRENT\_TIMESTAMP. Таким образом, эти поля можно вообще убрать из запроса. Для этого мы отфильтруем элементы в массиве от тех, значение которых = NULL:

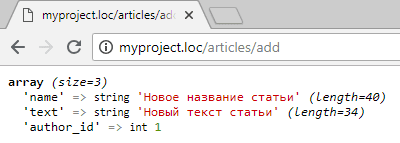
private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

var\_dump($filteredProperties);

}

[](https://php.zone/uploads/img/097ca53d08020e68c5f2d78916646f3a580c45f4a0d3d1b393cbf65a5e6b33f3.png)

Отлично, теперь у нас осталось всего 3 поля. Давайте для начала сформируем массив, содержащий названия столбцов в таблице:

private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

$columns = [];

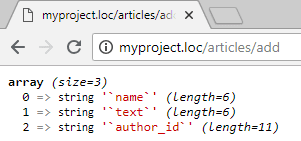
foreach ($filteredProperties as $columnName => $value) {

$columns[] = '`' . $columnName. '`';

}

var\_dump($columns);

}

[](https://php.zone/uploads/img/8a6e52a6fb4c200f070c4db2a4390dddaf3056800b3bf3af964649e58c1c8021.png)

А теперь подготовим массив с именами подстановок, вроде :author\_id и :name.

private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

$columns = [];

$paramsNames = [];

foreach ($filteredProperties as $columnName => $value) {

$columns[] = '`' . $columnName. '`';

$paramName = ':' . $columnName;

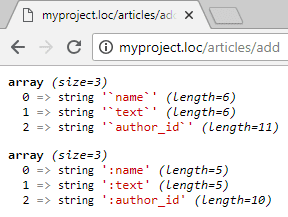
$paramsNames[] = $paramName;

}

var\_dump($columns);

var\_dump($paramsNames);

}

[](https://php.zone/uploads/img/64a5c65772d81de5d9b10e5d2701b217b00f8d02c9944bd12c5bdce4e0f61ed1.png)

Ну и, наконец, подготовим параметры, которые нужно будет подставить в запрос.

private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

$columns = [];

$paramsNames = [];

$params2values = [];

foreach ($filteredProperties as $columnName => $value) {

$columns[] = '`' . $columnName. '`';

$paramName = ':' . $columnName;

$paramsNames[] = $paramName;

$params2values[$paramName] = $value;

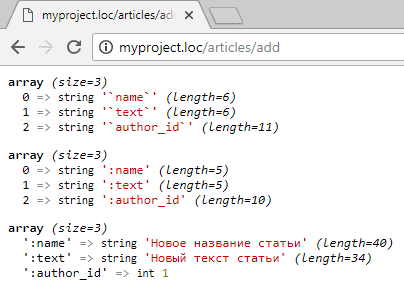
}

var\_dump($columns);

var\_dump($paramsNames);

var\_dump($params2values);

}

[](https://php.zone/uploads/img/676a2743dfc42dcd89f1e521fa48faec29016caa7175b213631c9e3251b3e267.png)

Все части для запроса готовы. Остаётся лишь собрать готовый запрос.

private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

$columns = [];

$paramsNames = [];

$params2values = [];

foreach ($filteredProperties as $columnName => $value) {

$columns[] = '`' . $columnName. '`';

$paramName = ':' . $columnName;

$paramsNames[] = $paramName;

$params2values[$paramName] = $value;

}

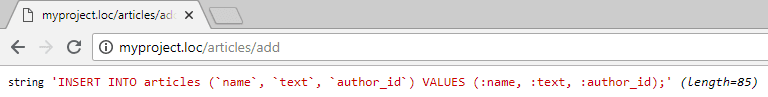
$columnsViaSemicolon = implode(', ', $columns);

$paramsNamesViaSemicolon = implode(', ', $paramsNames);

$sql = 'INSERT INTO ' . static::getTableName() . ' (' . $columnsViaSemicolon . ') VALUES (' . $paramsNamesViaSemicolon . ');';

var\_dump($sql);

}

[](https://php.zone/uploads/img/e76911ec20f98ea68211f112f0ed7eba44ddb5585d8321d93a0a819590a50adc.png)

Остаётся выполнить запрос, подставив нужные параметры.

private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

$columns = [];

$paramsNames = [];

$params2values = [];

foreach ($filteredProperties as $columnName => $value) {

$columns[] = '`' . $columnName. '`';

$paramName = ':' . $columnName;

$paramsNames[] = $paramName;

$params2values[$paramName] = $value;

}

$columnsViaSemicolon = implode(', ', $columns);

$paramsNamesViaSemicolon = implode(', ', $paramsNames);

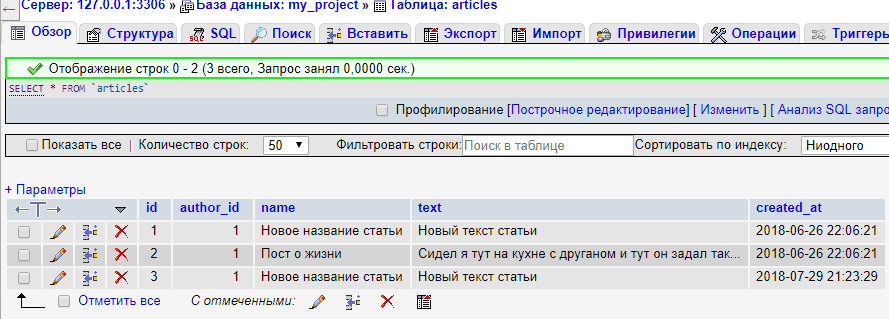
$sql = 'INSERT INTO ' . static::getTableName() . ' (' . $columnsViaSemicolon . ') VALUES (' . $paramsNamesViaSemicolon . ');';

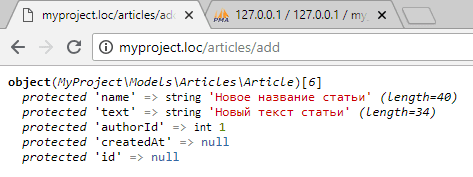
$db = Db::getInstance();

$db->query($sql, $params2values, static::class);

}

Если мы сейчас обновим нашу страничку, то увидим, что в базе данных успешно добавилась новая запись.

[](https://php.zone/uploads/img/9706cdf6f6e3cd12643ac92d09bf434ae6b0915f00789a051a3aa184025945a8.png)

[](https://php.zone/uploads/img/569237a58a1bc4c81613677ed837c21c6bbda325f8052f586f60157cc07419bd.png)

Обратите внимание, что в контроллере мы вывели наш объект уже после сохранения в базу:

**src/MyProject/Controllers/ArticlesController.php**

public function add(): void

{

$author = User::getById(1);

$article = new Article();

$article->setAuthor($author);

$article->setName('Новое название статьи');

$article->setText('Новый текст статьи');

$article->save();

var\_dump($article);

}

Однако, у него не обновилось поле id, а ведь это нам необходимо. Потому что если мы что-либо изменим сейчас в этом объекте и снова вызовем метод save(), то вместо обновления записи в базе, будет создана ещё одна. Так произойдет, потому что в методе save() мы проверяем значение поля id, и если оно равно null, то мы вызываем insert(), а не update(). Давайте исправим это недоразумение. Для того, чтобы получить id последней вставленной записи в базе (в рамках текущей сессии работы с БД) можно использовать метод lastInsertId() у объекта PDO. Давайте в нашем классе Db добавим следующий метод:

**src/MyProject/Services/Db.php**

public function getLastInsertId(): int

{

return (int) $this->pdo->lastInsertId();

}

Теперь используем его в методе insert():

private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

$columns = [];

$paramsNames = [];

$params2values = [];

foreach ($filteredProperties as $columnName => $value) {

$columns[] = '`' . $columnName. '`';

$paramName = ':' . $columnName;

$paramsNames[] = $paramName;

$params2values[$paramName] = $value;

}

$columnsViaSemicolon = implode(', ', $columns);

$paramsNamesViaSemicolon = implode(', ', $paramsNames);

$sql = 'INSERT INTO ' . static::getTableName() . ' (' . $columnsViaSemicolon . ') VALUES (' . $paramsNamesViaSemicolon . ');';

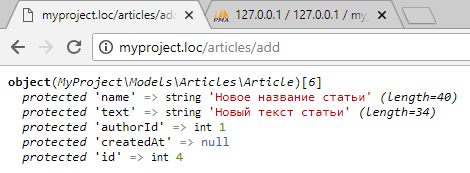
$db = Db::getInstance();

$db->query($sql, $params2values, static::class);

$this->id = $db->getLastInsertId();

}

Снова перезагрузим страницу.

[](https://php.zone/uploads/img/59d0f98ea0f05b2aad8baf839ba6f089407b5f3ad43f94a52020e237a9d2be9b.png)

Как видим, теперь у объекта после вызова метода save() появился id.  
Однако, в свойстве createdAt по-прежнему null. Доработать этот недостаток вы сможете в домашнем задании. А на этом данный урок заканчивается.

Текущая версия проекта [на гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/7003a493d83cb8bc4f84843d11abf5d344aaf3ca).

## Домашнее задание

Доработайте метод insert() таким образом, чтобы поля объекта обновлялись значениями из БД. Например, в поле createdAt должна появиться строка с датой.

[ivashkevich](https://php.zone/user/1) 31.07.2018 в 17:33

Пример решения домашки:

private function insert(array $mappedProperties): void

{

$filteredProperties = array\_filter($mappedProperties);

$columns = [];

$paramsNames = [];

$params2values = [];

foreach ($filteredProperties as $columnName => $value) {

$columns[] = '`' . $columnName. '`';

$paramName = ':' . $columnName;

$paramsNames[] = $paramName;

$params2values[$paramName] = $value;

}

$columnsViaSemicolon = implode(', ', $columns);

$paramsNamesViaSemicolon = implode(', ', $paramsNames);

$sql = 'INSERT INTO ' . static::getTableName() . ' (' . $columnsViaSemicolon . ') VALUES (' . $paramsNamesViaSemicolon . ');';

$db = Db::getInstance();

$db->query($sql, $params2values, static::class);

$this->id = $db->getLastInsertId();

$this->refresh();

}

private function refresh(): void

{

$objectFromDb = static::getById($this->id);

$reflector = new \ReflectionObject($objectFromDb);

$properties = $reflector->getProperties();

foreach ($properties as $property) {

$property->setAccessible(true);

$propertyName = $property->getName();

$this->$propertyName = $property->getValue($objectFromDb);

}

}

**23 Удаление в Active Record**

Всем привет! Сегодня мы дошли до самого интересного – операции удаления в Active Record :)

Удаление объекта мы будем производить вот так:

$article->delete();

То есть берём какой-то объект-наследник класса ActiveRecordEntity и вызываем у него метод delete(). Это должно привести к двум вещам:

Должна удалиться запись в базе данных, соответствующая этому объекту

Свойство id у этого объекта должно стать null (ведь его больше нет, логично же)

Удаление записей из таблиц выполняется следующим запросом:

DELETE FROM `название таблицы` WHERE id = :id;

А дальше идёт текст для совсем ленивых. Если Вы не такой – напишите метод delete() самостоятельно – это очень просто.

Итак, создаём в нашем классе ActiveRecordEntity метод delete().

**src/MyProject/Models/ActiveRecordEntity.php**

public function delete(): void

{

$db = Db::getInstance();

$db->query(

'DELETE FROM `' . static::getTableName() . '` WHERE id = :id',

[':id' => $this->id]

);

$this->id = null;

}

Всё! Метод для удаления готов!

Испытать его я вам предлагаю самостоятельно. О том, что именно требуется сделать, вы узнаете в домашнем задании.

Текущая версия проекта на [гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/244058bdf30e7ac1c43606a8e5d6f76fb3ac13f6).

# 24 CRUD-операции

Всем привет! В этом небольшом уроке мы затронем такое понятие как **CRUD**.

CRUD – это аббревиатура четырех слов, означающих следующие операции:

* Create (создание);
* Read (чтение);
* Update (обновление);
* Delete (удаление).

При работе с базами данных практически любое приложение должно поддерживать эти операции.  
В базе данных MySQL этим операциям соответствуют запросы:

* INSERT;
* SELECT;
* UPDATE;
* DELETE.

Если вы дошли до этого урока, то вы знаете, что все эти операции реализованы в нашем приложении в классе ActiveRecordEntity.

Весь этот урок – лишь пояснение того, что это за аббревиатура, так как Вас на собеседовании могут просто спросить: «Что такое CRUD?». И Вы с лёгкостью ответите, что это, а также, что Вы самостоятельно этот функционал реализовывали. Вот в общем-то и всё :)

# 25 M в MVC

Всем привет! В этом уроке мы добрались до последнего элемента архитектуры MVC, скрывающегося за буквой M. Речь сегодня пойдёт о Model (модели).

Model в MVC – это слой приложения, отвечающий за работу с данными, и содержащий в себе бизнес-логику. Бизнес-логика - это логика приложения, которая описывает то, что требуется от кода со стороны бизнеса. Например, бизнесу требуется, чтобы админ мог создавать записи в блоге. Значит при создании новой записи в блоге нужно проверять, что пользователь, создающий запись, является админом. И конкретно эта логика должна описываться в слое модели.

Таким образом, работа с базой данных, CRUD-операции и бизнес-логика – всё это должно описываться в модели.

Если говорить о моделях в нашем приложении, то это будет класс ActiveRecordEntity и его наследники – ведь именно они работают с базой данных и хранят в себе данные. При этом это не мешает нам создать какие-то дополнительные классы, которые будут промежуточным звеном между классами, реализующими Active Record, и контроллерами. Эти классы тоже могут содержать в себе бизнес-логику, но при этом не будут работать напрямую с базой данных. Они тоже будут относиться к слою модели.

Вот и весь урок о том, что же такое Модель в MVC. Как видите, мы эту часть приложения уже успешно реализовали. Если есть вопросы – задавайте в комментариях.

# 26 Работа с исключениями в PHP

Сегодня мы с вами разберем такую тему как исключения в PHP. Но прежде чем перейти к деталям, давайте дадим простое понятие термину «Исключительная ситуация». Исключительная ситуация в программе – это ситуация, при которой дальнейшее выполнение кода не имеет смысла. Например, новости на страничке /post/add могут добавлять только администраторы, но при этом это пытается сделать неавторизованный пользователь. Здесь имеет смысл проверить в самом начале права пользователя и если их недостаточно, обработать эту исключительную ситуацию. Или вот ещё один пример: наше приложение работает с базой данных, но при подключении к серверу MySQL обнаруживается, что такой базы данных на сервере нет. При такой ситуации тоже не имеет смысла продолжать выполнение скрипта – это исключительная ситуация.

В PHP для работы с такими исключительными ситуациями есть специальный механизм - исключения. Исключение – это такой объект специального класса. Этот класс является встроенным в PHP и называется Exception. Создание исключения выглядит следующим образом:

**<?php**

$exception = **new** **Exception**();

В качестве аргументов в конструктор можно передать сообщение и код ошибки:

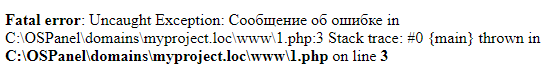
$exception = **new** **Exception**('Сообщение об ошибке', 123);

При этом класс Exception это немного нестандартный класс – объекты этого класса можно «бросать». Для этого используется оператор throw. Делается это вот так:

**<?php**

$exception = **new** **Exception**('Сообщение об ошибке', 123);

**throw** $exception;

Если мы сейчас запустим этот скрипт, то увидим следующее:  
[](https://php.zone/uploads/img/703db5eb0ec0978f94ebaf5b9891e9666ec2dce08ad80e8d7c0b0edbc8750c76.png)

В ошибке говорится о том, что произошло «непойманное» исключение, или как это принято говорить в русскоязычном варианте, необработанное исключение.

Для того, чтобы исключение поймать, используется специальная конструкция try-catch:

**<?php**

**try** {

**throw** **new** **Exception**('Сообщение об ошибке', 123);

} **catch** (**Exception** $e) {

**echo** 'Было поймано исключение: ' . $e->getMessage() . '. Код: ' . $e->getCode();

}

Результат будет следующим:  
[Обработанное стандартное исключение](https://php.zone/uploads/img/33ff01d99ae0b55638500b69f83554646c632cf10a639f8334fb487c10391ecb.png)

В чем же смысл такой сложной конструкции? Дело в том, что исключение будет подниматься по стеку вызовов выше и выше, до тех пор, пока оно не будет поймано. Вот что это значит:

Пусть у нас есть 3 функции: func1, func2 и func3. func1 вызывает внутри себя func2, а func2 вызывает func3.

**function** **func1**()

{

*// какой-то код*

func2();

}

**function** **func2**()

{

*// какой-то код*

func3();

}

**function** **func3**()

{

*// код, в котором возможна исключительная ситуация*

**throw** **new** **Exception**('Ошибка при подключении к БД');

}

Для того, чтобы обработать это исключение уровнем выше, достаточно написать блок try-catch на уровне func2, обернув вызов func3 внутри секции try:

**<?php**

**function** **func1**()

{

*// какой-то код*

func2();

}

**function** **func2**()

{

**try** {

*// какой-то код*

func3();

} **catch** (**Exception** $e) {

**echo** 'Было поймано исключение: ' . $e->getMessage();

}

}

**function** **func3**()

{

*// код, в котором возможна исключительная ситуация*

**throw** **new** **Exception**('Ошибка при подключении к БД');

}

func1();

[Исключение, пойманное на другом уровне](https://php.zone/uploads/img/dacd7c1c54a57196ed207c2e792a599e044a1161348c23a461bcc9518aac222e.png)

А можем и вовсе поймать его и в func1:

**<?php**

**function** **func1**()

{

**try** {

*// какой-то код*

func2();

} **catch** (**Exception** $e) {

**echo** 'Было поймано исключение: ' . $e->getMessage();

}

}

**function** **func2**()

{

*// какой-то код*

func3();

}

**function** **func3**()

{

*// код, в котором возможна исключительная ситуация*

**throw** **new** **Exception**('Ошибка при подключении к БД');

}

func1();

Результат будет таким же:  
[То же исключение](https://php.zone/uploads/img/dacd7c1c54a57196ed207c2e792a599e044a1161348c23a461bcc9518aac222e.png)

В этом случае исключение будет брошено внутри func3, поднимется на уровень func2, там его никто не поймает, и оно пойдет на еще уровень выше, в место, где была вызвана func2 – внутри func1. И вот здесь-то оно и будет поймано и обработано. После того как исключение обработано, будет выполнен код, который идет после блока try-catch.

**<?php**

**function** **func1**()

{

**try** {

*// какой-то код*

func2();

} **catch** (**Exception** $e) {

**echo** 'Было поймано исключение: ' . $e->getMessage();

}

**echo** 'А теперь выполнится этот код';

}

**function** **func2**()

{

*// какой-то код*

func3();

}

**function** **func3**()

{

*// код, в котором возможна исключительная ситуация*

**throw** **new** **Exception**('Ошибка при подключении к БД');

**echo** 'Этот код не выполнится, так как идет после места, где было брошено исключение';

}

func1();

Код, который идёт после того, где было брошено исключение, выполнен не будет. Исключение прерывает выполнение кода, и только после места, где оно было поймано и обработано, код продолжит выполняться.

## Наследование классов-исключений

От класса Exception можно наследоваться. Таким образом, мы можем создавать свои классы исключений для разных ситуаций. Например, для ошибок при работе с базой данных мы можем создать класс DbException, а для ошибок при работе с файлами – FileSystemException.

Блок try-catch позволяет обрабатывать разные типы исключений, это выглядит так:

**try** {

*// тут какой-то код*

} **catch** (DbException $e) {

*// обработка исключений, связанных с базой данных*

} **catch** (FileSystemException $e) {

*// обработка исключений, связанных с файловой системой*

}

Давайте теперь попробуем использовать этот механизм в нашем приложении. В качестве примера я покажу, как это можно использовать для обработки ошибок при подключении к БД.

Для начала давайте зададим некорректное название базы данных для подключения и попробуем запустить наше приложение.

**src/settings.php**

**<?php**

**return** [

'db' => [

'host' => 'localhost',

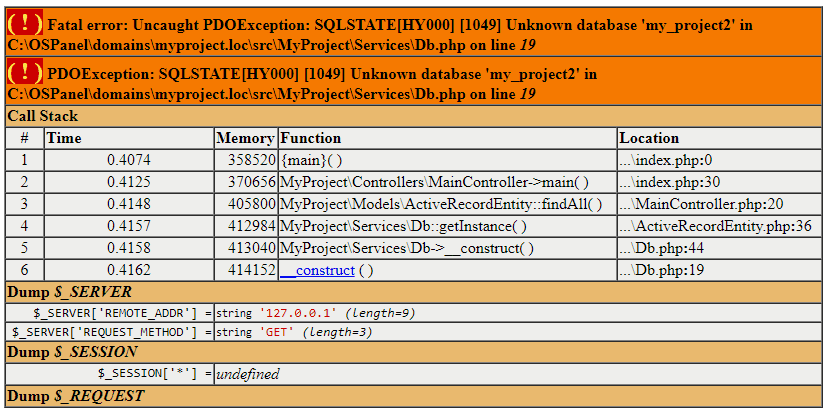
'dbname' => 'my\_project2',

'user' => 'root',

'password' => '',

]

];

[](https://php.zone/uploads/img/1925a032953d50018e38bc626d7a178cc314f918c6accb4c8f55ea087fc37f11.png)

Как видим, возникла ошибка – непойманное исключение типа PDOException. Это такой встроенный в PHP класс для исключений при работе с базой данных через PDO.

Давайте теперь создадим собственный класс исключений специально для базы данных. Назовём его DbException.

**src/MyProject/Exceptions/DbException.php**

**<?php**

**namespace** **MyProject**\**Exceptions**;

**class** **DbException** **extends** \**Exception**

{

}

Теперь создадим «ловушки» для стандартных исключениий класса PDOException, и будем заменять их своими исключениями. Вот так:

**src/MyProject/Services/Db.php**

**private** **function** **\_\_construct**()

{

$dbOptions = (**require** **\_\_DIR\_\_** . '/../../settings.php')['db'];

**try** {

**$this**->pdo = **new** \PDO(

'mysql:host=' . $dbOptions['host'] . ';dbname=' . $dbOptions['dbname'],

$dbOptions['user'],

$dbOptions['password']

);

**$this**->pdo->exec('SET NAMES UTF8');

} **catch** (\PDOException $e) {

**throw** **new** DbException('Ошибка при подключении к базе данных: ' . $e->getMessage());

}

}

Снова перезагрузим страничку и увидим уже наше сообщение об ошибке.

Давайте теперь обработаем наше исключение. Для этого нам требуется поймать исключение уже с типом DbException. Так как ошибка при работе с базой данных – это критичная ошибка, которая наверняка не позволит выполняться программе дальше, нам стоит ловить её на самом низком уровне нашего приложения – во фронт-контроллере. Обернем в блок try-catch код фронт-контроллера.

**www/index.php**

**try** {

spl\_autoload\_register(**function** (string $className) {

**require\_once** **\_\_DIR\_\_** . '/../src/' . $className . '.php';

});

$route = $\_GET['route'] ?? '';

$routes = **require** **\_\_DIR\_\_** . '/../src/routes.php';

$isRouteFound = **false**;

**foreach** ($routes **as** $pattern => $controllerAndAction) {

preg\_match($pattern, $route, $matches);

**if** (!**empty**($matches)) {

$isRouteFound = **true**;

**break**;

}

}

**if** (!$isRouteFound) {

**echo** 'Страница не найдена!';

**return**;

}

**unset**($matches[0]);

$controllerName = $controllerAndAction[0];

$actionName = $controllerAndAction[1];

$controller = **new** $controllerName();

$controller->$actionName(...$matches);

} **catch** (\MyProject\Exceptions\DbException $e) {

**echo** $e->getMessage();

}

Если мы теперь обновим страничку, то увидим уже только наше сообщение, без каких-либо необработанных исключений. Мы поймали исключение и просто вывели текст ошибки через echo.

А теперь давайте во фронт-контроллере научимся выводить ошибки, используя наш компонент View. То есть мы будем выводить ошибки через шаблоны. Это позволит избежать дублирования кода и всегда использовать один механизм для вывода.

Первым делом создадим новый шаблон для критичных ошибок, которые говорят о том, что сайт в данный момент недоступен. Для этого принято использовать код ответа 500. Поэтому мы так и назовём этот шаблон.

**templates/errors/500.php**

<h1>Хьюстон, у нас проблема!</h1>

**<?**= $error **?>**

А теперь просто обработаем исключение во фронт-контроллере по-новому.

**www/index.php**

...

$controller = **new** $controllerName();

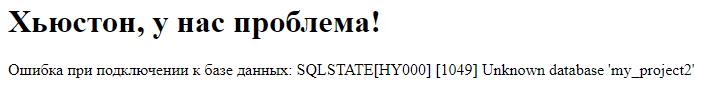
$controller->$actionName(...$matches);

} **catch** (\MyProject\Exceptions\DbException $e) {

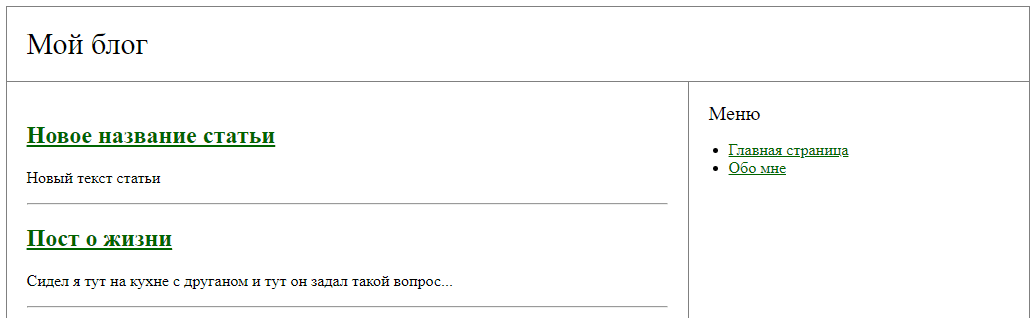
$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('500.php', ['error' => $e->getMessage()], 500);

}

Перезагружаем страничку.  
[](https://php.zone/uploads/img/d41c0167d6c18b20c457a1a9d72a1964e970faf378cc254342f4ca0c8a698ec4.png)

Отлично, теперь мы видим красивую, а главное – понятную нам ошибку. Вернём настройки базы данных, и убедимся, что блог работает.

[](https://php.zone/uploads/img/ff53289067154e5b5f4b81694168a623ac4c57e213a1da7bf1e02de9d1c696f3.png)

А теперь давайте посмотрим на вот этот кусок кода во фронт-контроллере:

**if** (!$isRouteFound) {

echo 'Страница не найдена!';

return;

}

Этот код работает, когда не найден нужный роутинг. Это ведь тоже исключительная ситуация! Давайте сделаем исключение для случаев, когда страничка не найдена.

**src/MyProject/Exceptions/NotFoundException.php**

**<?php**

**namespace** **MyProject**\**Exceptions**;

**class** **NotFoundException** **extends** \**Exception**

{

}

И начнём бросать его в случае, когда роут не найден. Для этого заменяем код

**www/index.php**

**if** (!$isRouteFound) {

echo 'Страница не найдена!';

return;

}

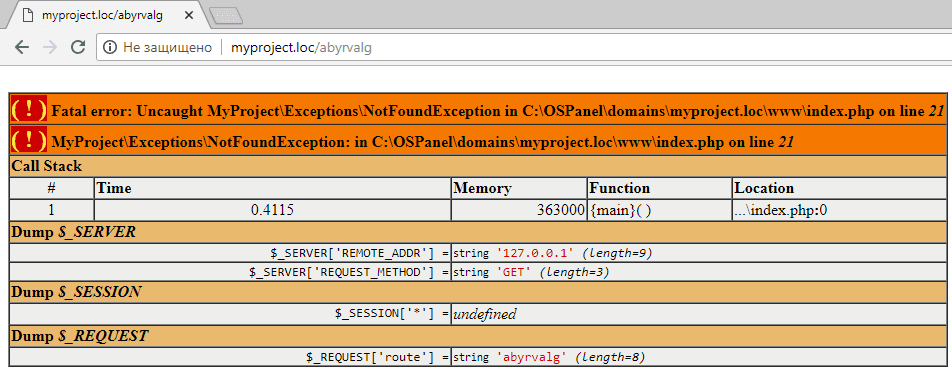
на

**if** (!$isRouteFound) {

**throw** **new** \MyProject\Exceptions\NotFoundException();

}

И проверяем на несуществующем роуте, например:

[](https://php.zone/uploads/img/4b397e5dac131ac4d71cbf5cc2c52c63f392bddb68fe7e1589dcb77712b1bb03.png)

Видим ошибку о непойманном исключении, так давайте же его поймаем! Для этого добавляем еще один блок catch.

**www/index.php**

**<?php**

**try** {

spl\_autoload\_register(**function** (string $className) {

**require\_once** **\_\_DIR\_\_** . '/../src/' . $className . '.php';

});

$route = $\_GET['route'] ?? '';

$routes = **require** **\_\_DIR\_\_** . '/../src/routes.php';

$isRouteFound = **false**;

**foreach** ($routes **as** $pattern => $controllerAndAction) {

preg\_match($pattern, $route, $matches);

**if** (!**empty**($matches)) {

$isRouteFound = **true**;

**break**;

}

}

**if** (!$isRouteFound) {

**throw** **new** \MyProject\Exceptions\NotFoundException();

}

**unset**($matches[0]);

$controllerName = $controllerAndAction[0];

$actionName = $controllerAndAction[1];

$controller = **new** $controllerName();

$controller->$actionName(...$matches);

} **catch** (\MyProject\Exceptions\DbException $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('500.php', ['error' => $e->getMessage()], 500);

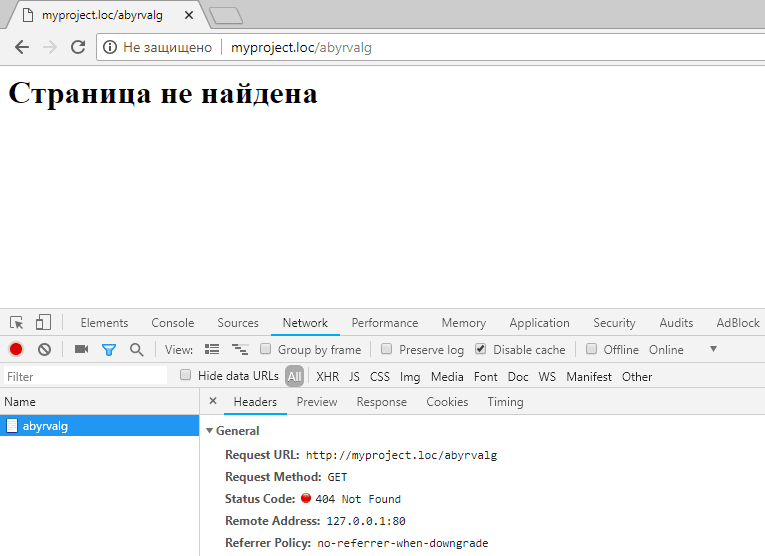
} **catch** (\MyProject\Exceptions\NotFoundException $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('404.php', ['error' => $e->getMessage()], 404);

}

Пробуем снова зайти по несуществующему адресу:

[](https://php.zone/uploads/img/75af6f7c0a075921f6df689452c2f6d2abfe89d7198febe7ea1d4921655eabaa.png)

Теперь мы видим ошибку из шаблона, и при этом сервер вернул код 404 – то, что нужно!

А теперь – внимание. Помните, мы уже использовали шаблон ошибки 404? Мы тогда писали в контроллере статей что-то типа такого:

**src/MyProject/Controllers/ArticlesController.php**

**if** ($article === **null**) {

**$this**->view->renderHtml('errors/404.php', [], 404);

**return**;

}

А теперь мы можем просто кинуть там исключение, вот так:

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

И оно всплывет через все слои нашей программы до фронт-контроллера, где будет успешно поймано!

Давайте перепишем контроллер статей:

**src/MyProject/Controllers/ArticlesController.php**

**<?php**

**namespace** **MyProject**\**Controllers**;

**use** **MyProject**\**Exceptions**\**NotFoundException**;

**use** **MyProject**\**Models**\**Articles**\**Article**;

**use** **MyProject**\**Models**\**Users**\**User**;

**use** **MyProject**\**View**\**View**;

**class** **ArticlesController**

{

*/\*\* @var View \*/*

**private** $view;

**public** **function** **\_\_construct**()

{

**$this**->view = **new** View(**\_\_DIR\_\_** . '/../../../templates');

}

**public** **function** **view**(int $articleId)

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

**$this**->view->renderHtml('articles/view.php', [

'article' => $article

]);

}

**public** **function** **edit**(int $articleId)

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

$article->setName('Новое название статьи');

$article->setText('Новый текст статьи');

$article->save();

}

**public** **function** **add**(): **void**

{

$author = User::getById(1);

$article = **new** Article();

$article->setAuthor($author);

$article->setName('Новое название статьи');

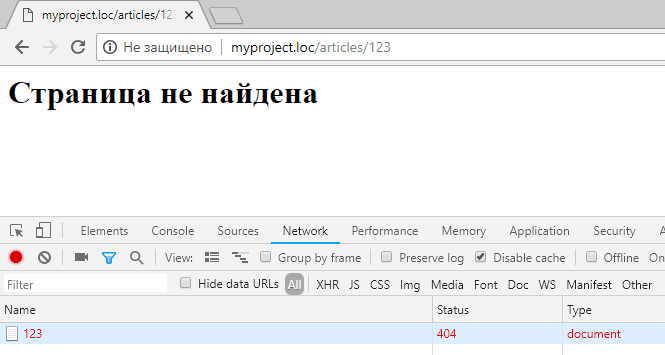
$article->setText('Новый текст статьи');

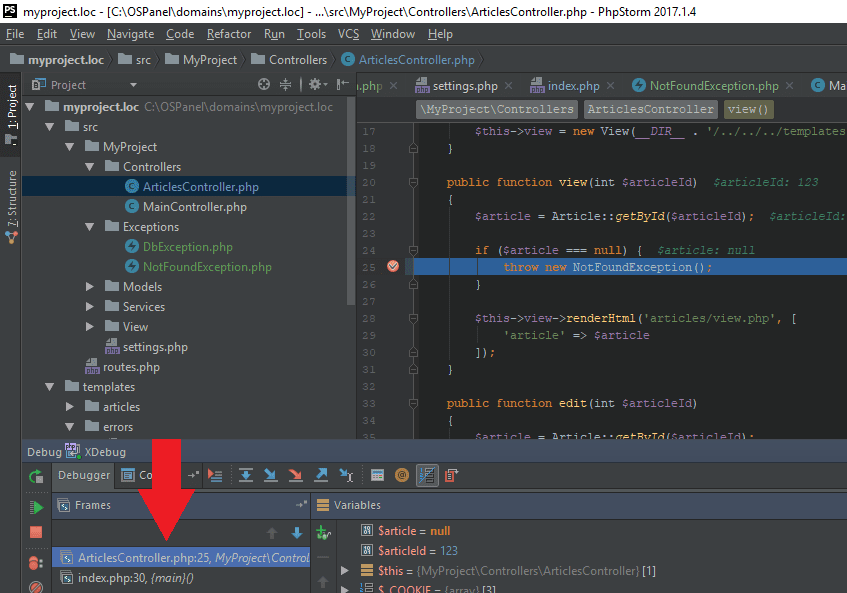
$article->save();

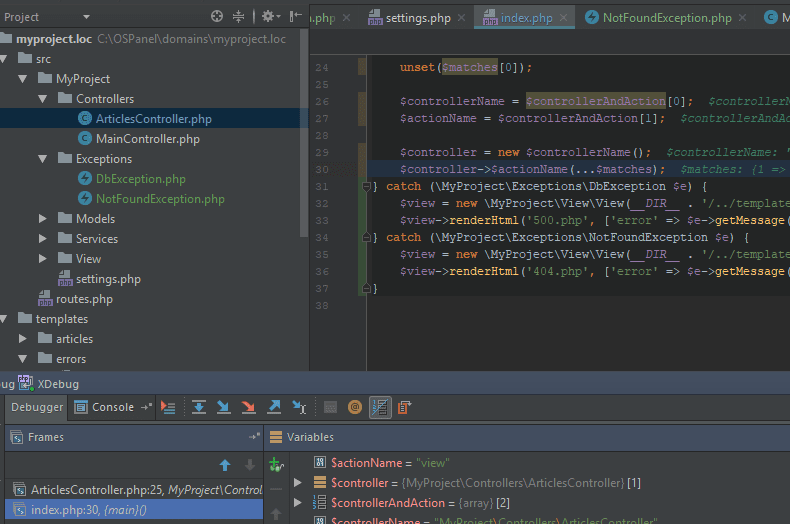
var\_dump($article);

}

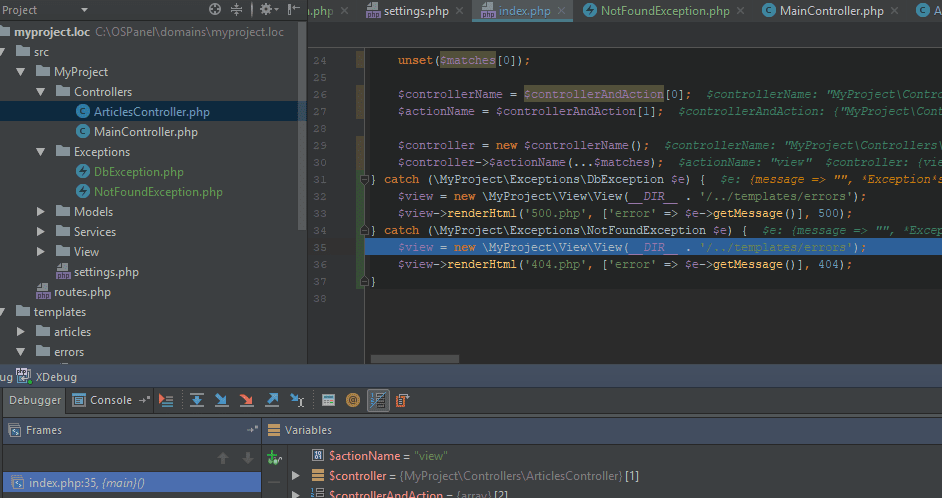
}

А теперь попробуем зайти на адрес с несуществующей статьёй: <http://myproject.loc/articles/123>  
[](https://php.zone/uploads/img/a6b8f1ca1aec9917b0bd751175d71ac5aed8e9c1e4da3c0fd5925d01f4d943aa.png)

Видим всё ту же ошибку! Давайте теперь посмотрим, что произошло. Поставим breakpoint в отладчике на место, где бросается исключение и обновим страничку.  
[](https://php.zone/uploads/img/90c57895331cdf62dd25b6c3626f6ad644079ecb8a61df74138f09fc906097f3.png)

Видите, слева внизу есть стек вызовов. Если переключиться на предыдущий уровень (index.php), мы увидим, где был вызван экшн контроллера:  
[](https://php.zone/uploads/img/33bbb93c0264ab8d2dcda792c0b8072618dae156eba205616cafcddc88b61649.png)

Это произошло в файле index.php, внутри блока try-catch. Так как в самом экшене брошенное исключение никто не ловит, оно всплывет на уровень, где был вызван экшен – в файл index.php. Если мы сейчас нажмем F8, то увидим, что исключение было успешно поймано и мы попали в нужный нам блок catch.

[](https://php.zone/uploads/img/c28d64d48eabc304b1e4bba0e0df5922d44abc0076ce1623158a70af7495c400.png)

Здесь мы вывели сообщение об ошибке – вот и всё! Теперь, если нам в приложении потребуется где-то вывести сообщение о несуществующей странице – мы просто бросим исключение с нужным типом.

На этом всё, за домашку =)

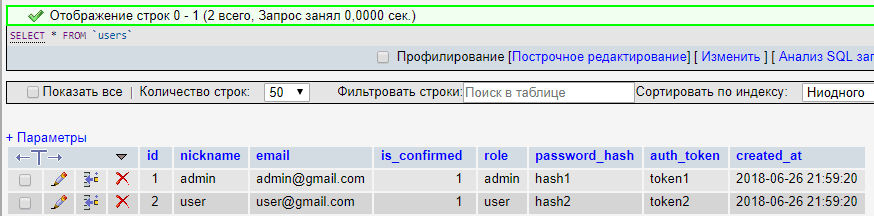
Текущая версия проекта [на гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/b4124ba9edc2aca7e278fc92a9cd4937beea355b).

## Домашнее задание

Самостоятельно прочитайте в официальной документации о новом типе ошибок в PHP 7 – Error. Их тоже можно бросать, однако на данном этапе мы их использовать не будем и пока ограничимся только типом Exception. Также изучите самостоятельно интерфейс Throwable. Попробуйте реализовать этот интерфейс в каком-нибудь классе.

# 27 Пишем регистрацию на сайте на PHP

Ну вот мы с вами и закончили изучение основных возможностей объектно-ориентированного программирования. Теперь мы будем учиться применять все эти возможности в деле. А именно – напишем движок для блога на чистом PHP. И в сегодняшнем уроке мы узнаем как **сделать регистрацию на сайте**. Благодаря ей новые пользователи смогут регистрироваться на нашем сайте. Поехали.

В [одном из прошлых уроков](https://php.zone/oop-v-php-prodvinutyj-kurs/15) мы с вами уже создавали таблицу users, которая имеет следующий вид:  
[](https://php.zone/uploads/img/0310aa99e74da05b82f6f3bf6b68f752d9701070b0b83556c6cd1f3af51b23ac.png)

Именно её мы и будем использовать **для регистрации и авторизации пользователей c помощью PHP и MySQL**. Давайте первым делом составим список полей, которые мы должны будем принимать от пользователя для регистрации:

* nickname – должен быть уникальным и содержать только символы латинского алфавита и цифры;
* email – должен быть уникальным и быть корректным email-ом;
* password – должен быть не менее 8 символов (будет захеширован и будет храниться в поле password\_hash).

Все остальные поля будут заполняться значениями по-умолчанию на стороне сервера.

## Создаём контроллер

Итак, давайте создадим контроллер для работы с пользователями.

**src/MyProject/Controllers/UsersController.php**

**<?php**

**namespace** **MyProject**\**Controllers**;

**use** **MyProject**\**View**\**View**;

**class** **UsersController**

{

*/\*\* @var View \*/*

**private** $view;

**public** **function** **\_\_construct**()

{

**$this**->view = **new** View(**\_\_DIR\_\_** . '/../../../templates');

}

**public** **function** **signUp**()

{

**echo** 'здесь будет код для регистрации пользователей';

}

}

## Добавляем роут

Теперь давайте пропишем роутинг для данной странички. Пусть это будет myproject.loc/users/register

**src/routes.php**

**<?php**

**return** [

'~^articles/(\d+)$~' => [\MyProject\Controllers\ArticlesController::class, 'view'],

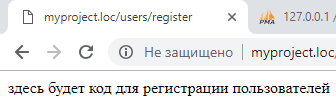
'~^articles/(\d+)/edit$~' => [\MyProject\Controllers\ArticlesController::class, 'edit'],

'~^articles/add$~' => [\MyProject\Controllers\ArticlesController::class, 'add'],

'~^users/register$~' => [\MyProject\Controllers\UsersController::class, 'signUp'],

'~^$~' => [\MyProject\Controllers\MainController::class, 'main'],

];

Проверим, что всё работает:  
[](https://php.zone/uploads/img/978abc1fb87f45031d55bcbfaf32fd1dc2f4ba71865e3f8f9eab4edc895573b9.png)

## Создаём шаблон

Отлично, давайте теперь создадим шаблон для нашей новой странички. Шаблон простейший и поэтому кривоватый, вы можете выровнять поля друг под другом с помощью стилей, но это не тема данного курса и здесь фронтенд будет крайне простым, сконцентрируемся на бэкенде.

**templates/users/signUp.php**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<div style="text-align: center;">

<h1>Регистрация</h1>

<form action="/users/register" method="post">

<label>Nickname <input type="text" name="nickname"></label>

<br><br>

<label>Email <input type="text" name="email"></label>

<br><br>

<label>Пароль <input type="password" name="password"></label>

<br><br>

<input type="submit" value="Зарегистрироваться">

</form>

</div>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

И теперь отрендерим этот шаблон в нашем контроллере:

**src/MyProject/Controllers/UsersController.php**

**<?php**

**namespace** **MyProject**\**Controllers**;

**use** **MyProject**\**View**\**View**;

**class** **UsersController**

{

*/\*\* @var View \*/*

**private** $view;

**public** **function** **\_\_construct**()

{

**$this**->view = **new** View(**\_\_DIR\_\_** . '/../../../templates');

}

**public** **function** **signUp**()

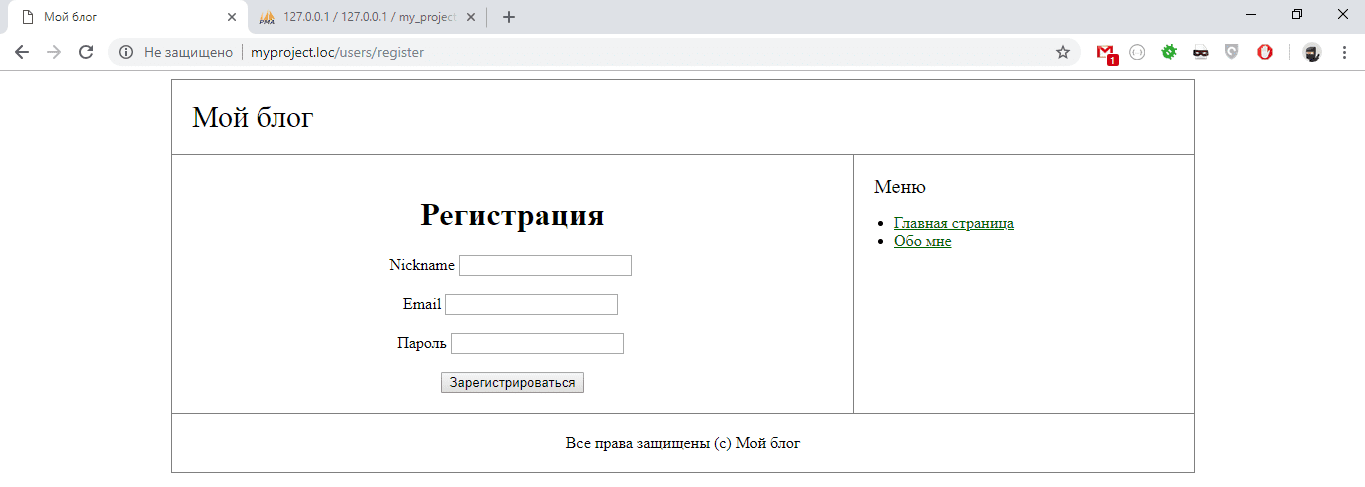
{

**$this**->view->renderHtml('users/signUp.php');

}

}

Проверяем, что все работает

[](https://php.zone/uploads/img/e8b506ec468b80771c501d502a896b79525afa945376e31cd95ef942d6909682.png)

## Пишем логику регистрации

Итак, заготовочку сделали, теперь – самое интересное. Нам нужно обработать данные, пришедшие от пользователя. Принимать данные из запроса мы будем внутри контроллера, однако вся логика по проверке этих данных будет находиться внутри модели пользователя. То есть принять данные из запроса – это всегда ответственность контроллера. Далее его задача – передать эти данные модели, чтобы она произвела с ними какие-то действия(проверила на валидность, сохранила в базу). Контроллеры не должны содержать в себе бизнес-логику (то есть то, как должны обрабатываться и храниться данные). За счет того, что такая логика всегда содержится в моделях, код становится проще реиспользовать (использовать в нескольких местах). Например, мы можем из разных контроллеров обращаться к одному и тому же коду, который хранится в модели. Итак, давайте создадим в модели пользователя статический метод, который будет принимать на вход массив с данными, пришедшими от пользователя, и будет пытаться создать нового пользователя и сохранить его в базе данных.

**src/MyProject/Models/Users/User.php**

...

**public** **static** **function** **signUp**(array $userData)

{

var\_dump($userData);

}

Именно этот код мы и будем вызывать в контроллере, если пришел POST-запрос.

**src/MyProject/Controllers/UsersController.php**

**public** **function** **signUp**()

{

**if** (!**empty**($\_POST)) {

$user = User::signUp($\_POST);

}

**$this**->view->renderHtml('users/signUp.php');

}

## Проверка входных данных на пустоту

Итак, приступаем к написанию логики в модели. Для начала стоит убедиться в том, что все необходимые данные были переданы в запросе. Если это не так – будем кидать исключение. Для такого рода ошибок заведем отдельное исключение:

**src/MyProject/Exceptions/InvalidArgumentException.php**

**<?php**

**namespace** **MyProject**\**Exceptions**;

**class** **InvalidArgumentException** **extends** \**Exception**

{

}

Мы будем использовать его для случаев, когда были переданы некорректные параметры или данные.

Итак, пишем проверки на то, что все данные были переданы:

src/MyProject/Models/Users/User.php

**public** **static** **function** **signUp**(array $userData)

{

**if** (**empty**($userData['nickname'])) {

**throw** **new** InvalidArgumentException('Не передан nickname');

}

**if** (**empty**($userData['email'])) {

**throw** **new** InvalidArgumentException('Не передан email');

}

**if** (**empty**($userData['password'])) {

**throw** **new** InvalidArgumentException('Не передан password');

}

}

В контроллере теперь нужно научиться обрабатывать эти исключения:

**src/MyProject/Controllers/UsersController.php**

**public** **function** **signUp**()

{

**if** (!**empty**($\_POST)) {

**try** {

$user = User::signUp($\_POST);

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('users/signUp.php', ['error' => $e->getMessage()]);

**return**;

}

}

**$this**->view->renderHtml('users/signUp.php');

}

Видите, мы начали передавать в шаблон переменную error? Нужно бы уметь выводить её в шаблоне, если она не пустая:

**templates/users/signUp.php**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<div style="text-align: center;">

<h1>Регистрация</h1>

**<?php** **if** (!**empty**($error)): **?>**

<div style="background-color: red;padding: 5px;margin: 15px">**<?**= $error **?>**</div>

**<?php** **endif**; **?>**

<form action="/users/register" method="post">

<label>Nickname <input type="text" name="nickname"></label>

<br><br>

<label>Email <input type="text" name="email"></label>

<br><br>

<label>Пароль <input type="password" name="password"></label>

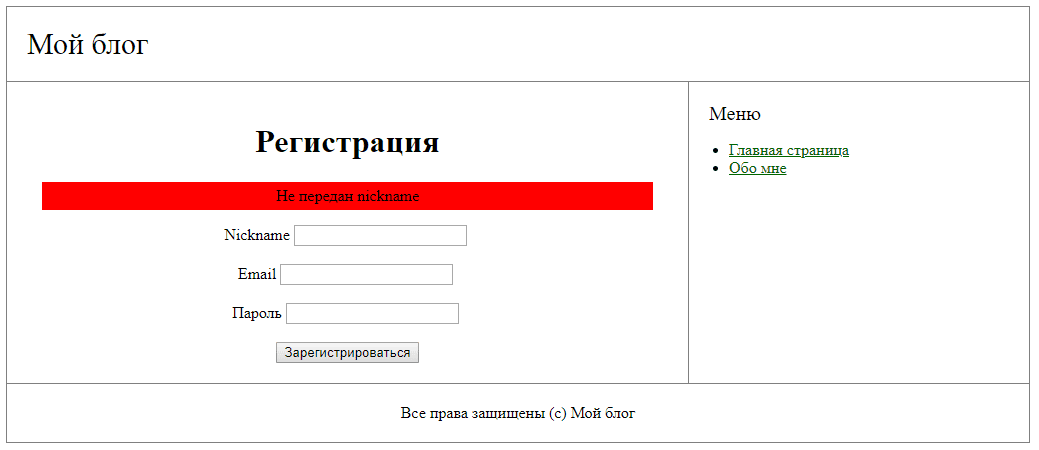
<br><br>

<input type="submit" value="Зарегистрироваться">

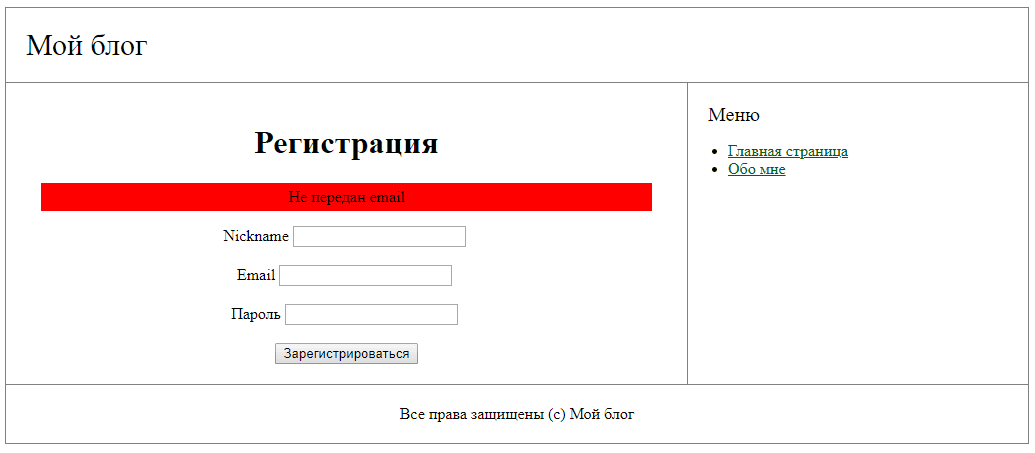
</form>

</div>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Теперь попробуем отправить форму регистрации с пустыми полями:  
[](https://php.zone/uploads/img/5fbcd51102e2c4fcad1a2fd188791a61ae0690b68ac3fedcfba5433eea09253c.png)

Увидим, что код упал на первой же проверке.

Попробуем заполнить теперь поле nickname и снова отправим запрос.  
[](https://php.zone/uploads/img/c050db73d22957dcf1b947655779184e1429b8640f6e885d6b15e9d4b30ab814.png)

Как видим, теперь уже ошибка о том, что мы не заполнили email. Отлично, значит наш код действительно проверяет наличие полей. Однако данные, которые мы заполняли перед отправкой формы потеряны, что, согласитесь неудобно, так как придется возвращаться назад. Давайте будем выводить в шаблоне данные, которые были переданы в запросе. Для этого используется атрибут value у тегов input.

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<div style="text-align: center;">

<h1>Регистрация</h1>

**<?php** **if** (!**empty**($error)): **?>**

<div style="background-color: red;padding: 5px;margin: 15px">**<?**= $error **?>**</div>

**<?php** **endif**; **?>**

<form action="/users/register" method="post">

<label>Nickname <input type="text" name="nickname" value="<?= $\_POST['nickname'] ?? '' ?>"></label>

<br><br>

<label>Email <input type="text" name="email" value="<?= $\_POST['email'] ?? '' ?>"></label>

<br><br>

<label>Пароль <input type="password" name="password" value="<?= $\_POST['password'] ?? '' ?>"></label>

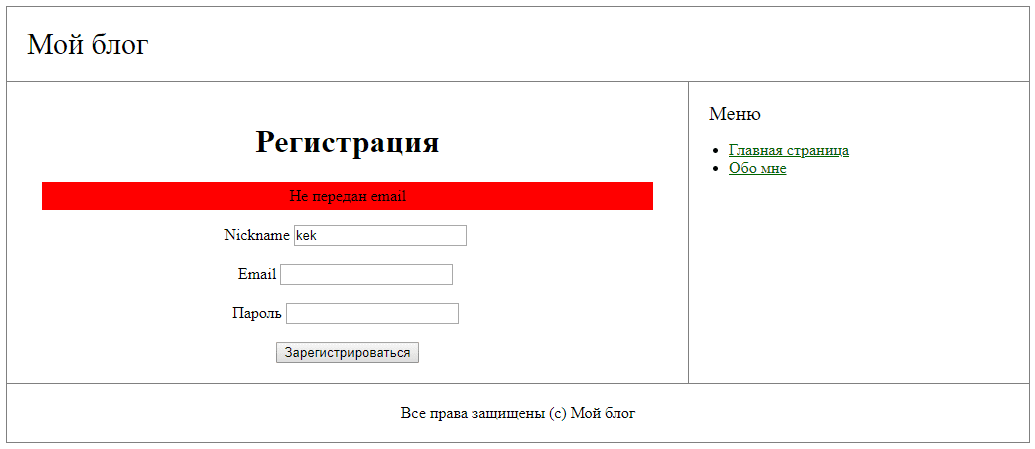
<br><br>

<input type="submit" value="Зарегистрироваться">

</form>

</div>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Попробуем снова отправить форму с заполненным полем nickname.  
[](https://php.zone/uploads/img/ba4b256f43fb91610c9ae749ad97e49abce12c78e4b0c87052716d4024cb8a25.png)

Видим всё ту же ошибку, но теперь данные формы не потерялись. А это значит, что наш интерфейс стал гораздо более удобным для пользователя – если возникнет ошибка, ему не нужно будет вводить всё заново.

## Проверка данных на валидность

Добавим также проверку на то, что длина пароля не менее восьми символов, а также что nickname содержит только допустимые символы, а email является корректным email-ом:

**src/MyProject/Models/Users/User.php**

**public** **static** **function** **signUp**(array $userData)

{

**if** (**empty**($userData['nickname'])) {

**throw** **new** InvalidArgumentException('Не передан nickname');

}

**if** (!preg\_match('/^[a-zA-Z0-9]+$/', $userData['nickname'])) {

**throw** **new** InvalidArgumentException('Nickname может состоять только из символов латинского алфавита и цифр');

}

**if** (**empty**($userData['email'])) {

**throw** **new** InvalidArgumentException('Не передан email');

}

**if** (!filter\_var($userData['email'], FILTER\_VALIDATE\_EMAIL)) {

**throw** **new** InvalidArgumentException('Email некорректен');

}

**if** (**empty**($userData['password'])) {

**throw** **new** InvalidArgumentException('Не передан password');

}

**if** (mb\_strlen($userData['password']) < 8) {

**throw** **new** InvalidArgumentException('Пароль должен быть не менее 8 символов');

}

}

## Поиск дубликатов

Теперь стоит проверить, что пользователя с такими email и nickname нет в базе. Для этого нам понадобится метод, позволяющий находить записи в базе по какому-то одному столбцу. Давайте добавим его в класс ActiveRecordEntity.

**src/MyProject/Models/ActiveRecordEntity.php**

**public** **static** **function** **findOneByColumn**(string $columnName, $value): ?**self**

{

$db = Db::getInstance();

$result = $db->query(

'SELECT \* FROM `' . **static**::getTableName() . '` WHERE `' . $columnName . '` = :value LIMIT 1;',

[':value' => $value],

**static**::class

);

**if** ($result === []) {

**return** **null**;

}

**return** $result[0];

}

Этот метод будет принимать два параметра:

1. имя столбца, по которому искать;
2. значение, которое мы ищем в этом столбце.

Если ничего не найдено – вернётся null. Если же что-то нашлось – вернётся первая запись.  
С помощью этого метода мы сможем искать пользователей по email и nickname:

**src/MyProject/Models/Users/User.php**

**public** **static** **function** **signUp**(array $userData)

{

...

**if** (**static**::findOneByColumn('nickname', $userData['nickname']) !== **null**) {

**throw** **new** InvalidArgumentException('Пользователь с таким nickname уже существует');

}

**if** (**static**::findOneByColumn('email', $userData['email']) !== **null**) {

**throw** **new** InvalidArgumentException('Пользователь с таким email уже существует');

}

}

Ну, если все эти проверки пройдены - всё готово для того, чтобы создать нового пользователя.

**public** **static** **function** **signUp**(array $userData): **User**

{

*// ... тут все проверки*

$user = **new** User();

$user->nickname = $userData['nickname'];

$user->email = $userData['email'];

$user->passwordHash = password\_hash($userData['password'], PASSWORD\_DEFAULT);

$user->isConfirmed = **false**;

$user->role = 'user';

$user->authToken = sha1(random\_bytes(100)) . sha1(random\_bytes(100));

$user->save();

**return** $user;

}

Тут думаю всё понятно, кроме параметра authToken – это специально случайным образом сгенерированный параметр, с помощью которого пользователь будет авторизовываться. Мы не будем передавать после того как вошли на сайт в cookie ни пароль, ни его хеш. Мы будем использовать только этот токен, который у каждого пользователя будет свой и он никак не будет связан с паролем – так безопаснее.

В конце метода мы сохраняем этого нового пользователя в базу и возвращаем его из метода. Обратите внимание, что у метода появился тип возвращаемого значения: User – не забывайте их указывать.  
Теперь в контроллере нужно получать результат этого метода и проверять, что нам вернулся только что созданный пользователь. Если всё ок – писать об этом на форме регистрации.

**src/MyProject/Controllers/UsersController.php**

**public** **function** **signUp**()

{

**if** (!**empty**($\_POST)) {

**try** {

$user = User::signUp($\_POST);

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('users/signUp.php', ['error' => $e->getMessage()]);

**return**;

}

**if** ($user **instanceof** User) {

**$this**->view->renderHtml('users/signUpSuccessful.php');

**return**;

}

}

**$this**->view->renderHtml('users/signUp.php');

}

Я решил не загромождать наш шаблон всем подряд и создал для успешной регистрации отдельный шаблончик - signUpSuccessful.php. Как видите, именно его я и рендерю в контроллере при успешном исходе.

**templates/users/signUpSuccessful.php**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

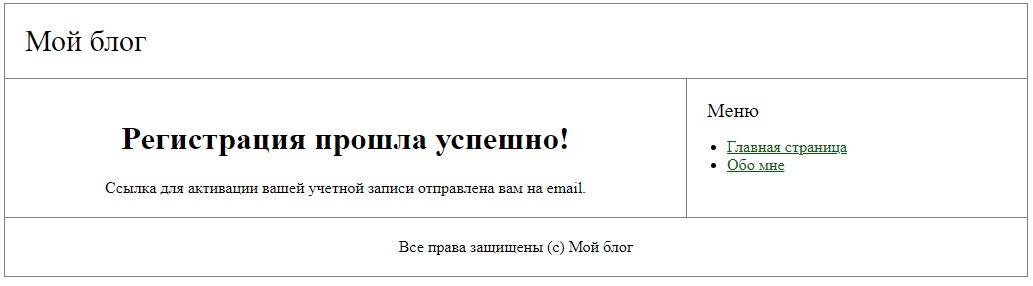
<div style="text-align: center;">

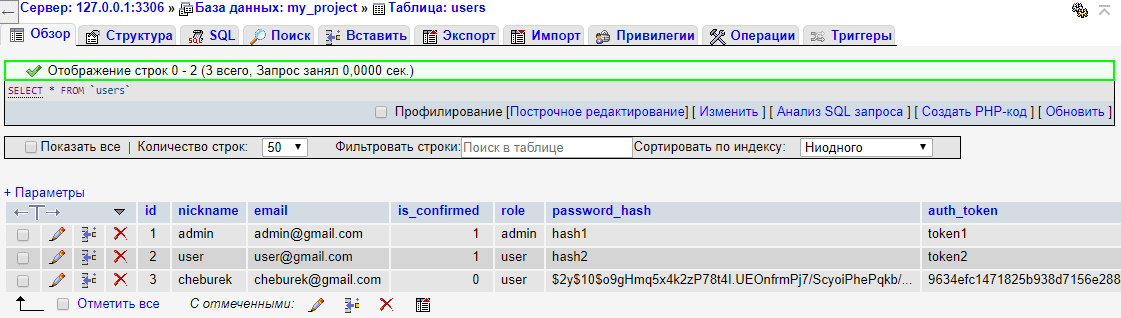
<h1>Регистрация прошла успешно!</h1>

Ссылка для активации вашей учетной записи отправлена вам на email.

</div>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Пробуем теперь заполнить нашу формочку корректными данными и отправляем запрос.  
[](https://php.zone/uploads/img/634de2748bcb0219ef76fe6050afcc57e6688753b40d1d46371b7ebd318cf8ac.png)

Заглянем теперь в базу и убедимся, что действительно появился новый пользователь  
[](https://php.zone/uploads/img/4482d9e08a163be395de5fd285c4a9da5f93b5973254af8f3ae63d6287fdc366.png)

Вот мы и сделали регистрацию для нашего блога. В следующем уроке мы сделаем активацию пользователя по email-у.

Текущая версия проекта [на github](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/0d1852b6f1dfd12ba488c6dd24e06591d69f29f4).

# 28 Система активации пользователей по email на PHP

В этом уроке мы напишем **систему активации пользователей по email**. После регистрации пользователь будет получать письмо со специальной ссылкой, содержащей код. При открытии этой ссылки будет происходить активация этого пользователя.

Прежде чем начать, вам нужно [настроить OpenServer для отправки писем](https://php.zone/post/939).

После того, как вы это сделали, можно приступать к написанию кода. Первое, что нам нужно – создать новую табличку, в которой мы будем хранить коды для активации пользователей.

Называем её «users\_activation\_codes», и указываем, что нам требуются три столбца:

* id – это просто id записи в таблице;
* user\_id – id пользователя;
* code – код для активации этого пользователя.

**CREATE** **TABLE** `users\_activation\_codes` (

`id` int(11) **NOT** NULL,

`user\_id` int(11) **NOT** NULL,

`code` varchar(255) **NOT** NULL

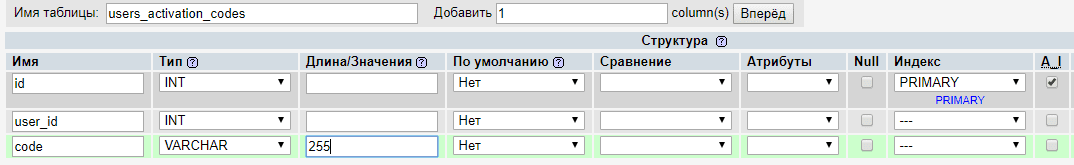
) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8;

**ALTER** **TABLE** `users\_activation\_codes`

**ADD** PRIMARY **KEY** (`id`);

**ALTER** **TABLE** `users\_activation\_codes`

**MODIFY** `id` int(11) **NOT** NULL AUTO\_INCREMENT;

[](https://php.zone/uploads/img/feecc9fdd8a47e7985408be524e5f48dc5d9f288c19fe80606a2267aaece5269.png)

Теперь нам нужно написать специальный сервис, который позволит нам создавать новые коды активации для пользователей, а также проверять код активации для конкретного пользователя. Пишем.

**src/MyProject/Models/Users/UserActivationService.php**

**<?php**

**namespace** **MyProject**\**Models**\**Users**;

**use** **MyProject**\**Services**\**Db**;

**class** **UserActivationService**

{

**private** **const** TABLE\_NAME = 'users\_activation\_codes';

**public** **static** **function** **createActivationCode**(User $user): **string**

{

*// Генерируем случайную последовательность символов, о функциях почитайте в документации*

$code = bin2hex(random\_bytes(16));

$db = Db::getInstance();

$db->query(

'INSERT INTO ' . **self**::TABLE\_NAME . ' (user\_id, code) VALUES (:user\_id, :code)',

[

'user\_id' => $user->getId(),

'code' => $code

]

);

**return** $code;

}

**public** **static** **function** **checkActivationCode**(User $user, string $code): **bool**

{

$db = Db::getInstance();

$result = $db->query(

'SELECT \* FROM ' . **self**::TABLE\_NAME . ' WHERE user\_id = :user\_id AND code = :code',

[

'user\_id' => $user->getId(),

'code' => $code

]

);

**return** !**empty**($result);

}

}

И теперь нам нужно сделать ещё один сервис, который будет предназначен для отправки email-сообщений.

**src/MyProject/Services/EmailSender.php**

**<?php**

**namespace** **MyProject**\**Services**;

**use** **MyProject**\**Models**\**Users**\**User**;

**class** **EmailSender**

{

**public** **static** **function** **send**(

User $receiver,

string $subject,

string $templateName,

array $templateVars = []

): **void** {

extract($templateVars);

ob\_start();

**require** **\_\_DIR\_\_** . '/../../../templates/mail/' . $templateName;

$body = ob\_get\_contents();

ob\_end\_clean();

mail($receiver->getEmail(), $subject, $body, 'Content-Type: text/html; charset=UTF-8');

}

}

Здесь вам все функции уже знакомы – мы использовали похожий функционал для рендеринга шаблонов. Теперь у нас появится еще один тип шаблонов – специально для email-ов.

Давайте создадим наш первый шаблон, который будет предназначен для писем активации.

**templates/mail/userActivation.php**

Добро пожаловать на наш портал!<br>

Для активации вашего аккаунта нажмите <a href="http://myproject.loc/users/<?=$userId?>/activate/<?=$code?>">сюда</a>.

Ну и, собственно, остаётся только взять эти компоненты и собрать воедино. Делаем это в контроллере пользователя в экшене с регистрацией.

**src/MyProject/Controllers/UsersController.php**

**public** **function** **signUp**()

{

**if** (!**empty**($\_POST)) {

**try** {

$user = User::signUp($\_POST);

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('users/signUp.php', ['error' => $e->getMessage()]);

**return**;

}

**if** ($user **instanceof** User) {

$code = UserActivationService::createActivationCode($user);

EmailSender::send($user, 'Активация', 'userActivation.php', [

'userId' => $user->getId(),

'code' => $code

]);

**$this**->view->renderHtml('users/signUpSuccessful.php');

**return**;

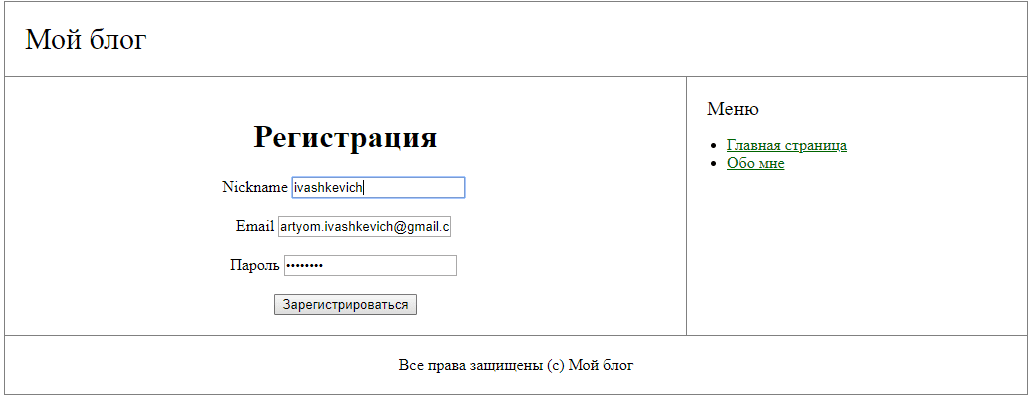
}

}

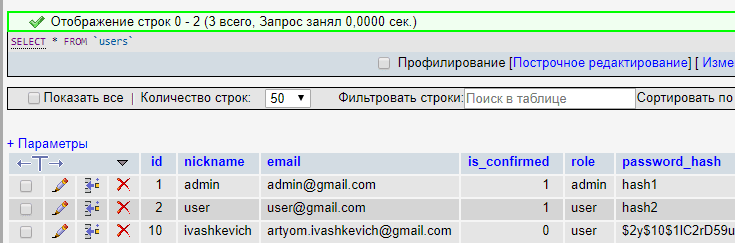
**$this**->view->renderHtml('users/signUp.php');

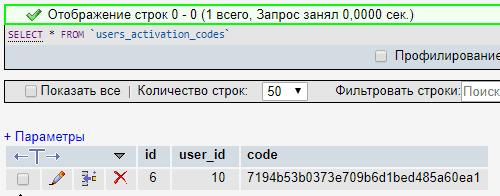
}

Теперь пробуем зарегистрироваться на свою почту в нашей системе.

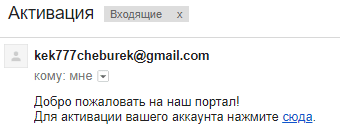
[](https://php.zone/uploads/img/41eb505479faf71e391c7074f949ce724af501299c4d59f9dcb8ca649e43e0ea.png)

И после этого смотрим в базу данных.

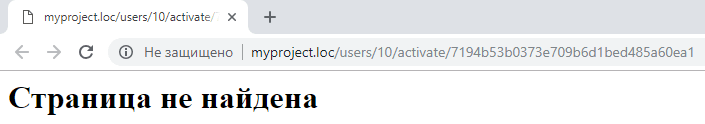
Пользователи:  
[](https://php.zone/uploads/img/d90faf3f3bd474d8ab260b4a658bcc66f377b8d0ae73d051ba7db82e9fd12c35.png)

Коды активации:  
[](https://php.zone/uploads/img/80837bee52d450cc67931d0202630422c2ce63fd32a76f2aea3e02b83cb7dbf3.png)

Как видим, все успешно отработало, и кроме того, нам пришло письмо на почту!

[](https://php.zone/uploads/img/6bb2b5c2292e09981aad4f6dd5406168a8e9fa4e90488ad7a6e09b2a1c867b9f.png)

После перехода по ссылке мы видим, что такой страницы не существует.

[](https://php.zone/uploads/img/bfbb295df3aafd2e14df24d3407ebf045ce2cd914df177a337c698162f2df645.png)

Еще бы, ведь мы не добавляли для нее соответствующий роутинг. Добавляем его.

**src/routes.php**

...

'~^users/(\d+)/activate/(.+)$~' => [\MyProject\Controllers\UsersController::class, 'activate'],

...

И добавляем соответствующий экшен в контроллере:

src/MyProject/Controllers/UsersController.php

**public** **function** **activate**(int $userId, string $activationCode)

{

$user = User::getById($userId);

$isCodeValid = UserActivationService::checkActivationCode($user, $activationCode);

**if** ($isCodeValid) {

$user->activate();

**echo** 'OK!';

}

}

И добавляем у модели пользователя метод activate().

**src/MyProject/Models/Users/User.php**

**public** **function** **activate**(): **void**

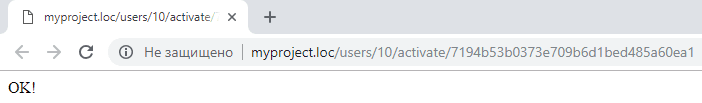
{

**$this**->isConfirmed = **true**;

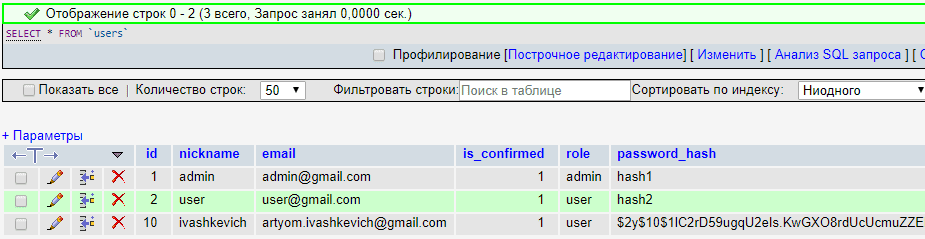
**$this**->save();

}

Снова пробуем обновить страничку для активации.

[](https://php.zone/uploads/img/246e5e9da6e33b81819f8b09d66a79585c5da3a90bedfc88618f32532370c142.png)

Видим заветное “OK!”.

Проверяем, что в базе наш пользователь теперь подтвержден.  
[](https://php.zone/uploads/img/5acb898e0aa9bdfbdc04914e3b5011d08f2e8647016251ec97cec865a81a6e71.png)

Успех! Теперь осталось довести систему до ума – создать нормальные шаблоны для странички активации и обрабатывать возможные ошибки. Это вам предоставляется сделать самостоятельно в домашнем задании.

Текущая версия проекта [на гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/f352cd14279b9567eba476ba72c1db7a77bdf5ce).

## Домашнее задание

Реализуйте следующий функционал:

* после того как код был использован, он должен удаляться из базы;
* обработайте все возможные ошибки в экшене activate() - подумайте, что будет, если в ссылку активации подставить несуществующего пользователя, что будет, если код не найден, и т.д. Чем больше сделаете - тем лучше;
* добавьте шаблоны для всех этих случаев.

# 29 Пишем систему авторизации на PHP

Сегодня мы напишем **авторизацию пользователя на сайте**. Вся система будет работать следующим образом: пользователь вводит логин и пароль на форме входа, если они правильные – в Cookie браузера будет установлена специальная запись – auth token (авторизационный токен). При дальнейших запросах на сервер этот токен будет проверяться и если он будет правильным, то пользователь считается авторизованным.

Первым делом решаем, что страница с формой логина и пароля будет находиться по адресу <http://myproject.loc/users/login>. Создаём соответствующий роут.

**src/routes.php**

...

'~^users/login$~' => [\MyProject\Controllers\UsersController::class, 'login'],

...

После этого добавляем новый экшен в контроллере.

**src/MyProject/Controllers/UsersController.php**

**public** **function** **login**()

{

**$this**->view->renderHtml('users/login.php');

}

И, наконец, создаём шаблон с формой для этого экшена.

**templates/users/login.php**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<div style="text-align: center;">

<h1>Вход</h1>

**<?php** **if** (!**empty**($error)): **?>**

<div style="background-color: red;padding: 5px;margin: 15px">**<?**= $error **?>**</div>

**<?php** **endif**; **?>**

<form action="/users/login" method="post">

<label>Email <input type="text" name="email" value="<?= $\_POST['email'] ?? '' ?>"></label>

<br><br>

<label>Пароль <input type="password" name="password" value="<?= $\_POST['password'] ?? '' ?>"></label>

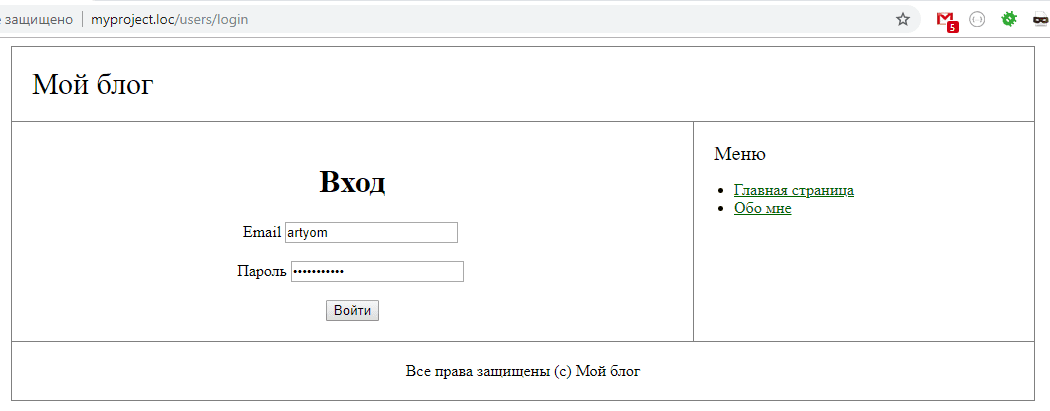
<br><br>

<input type="submit" value="Войти">

</form>

</div>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Теперь можно зайти в браузер и убедиться, что форма открывается.  
[](https://php.zone/uploads/img/7472579e63f2f2684dbc91b421e0122b36c3d629d2171a091b11e47e4c01f9a0.png)

Теперь нам нужно добавить обработку отправленной формы и добавить в модели пользователя метод для логина.

**src/MyProject/Models/Users/User.php**

**public** **static** **function** **login**(array $loginData): **User**

{

**if** (**empty**($loginData['email'])) {

**throw** **new** InvalidArgumentException('Не передан email');

}

**if** (**empty**($loginData['password'])) {

**throw** **new** InvalidArgumentException('Не передан password');

}

$user = User::findOneByColumn('email', $loginData['email']);

**if** ($user === **null**) {

**throw** **new** InvalidArgumentException('Нет пользователя с таким email');

}

**if** (!password\_verify($loginData['password'], $user->getPasswordHash())) {

**throw** **new** InvalidArgumentException('Неправильный пароль');

}

**if** (!$user->isConfirmed) {

**throw** **new** InvalidArgumentException('Пользователь не подтверждён');

}

$user->refreshAuthToken();

$user->save();

**return** $user;

}

**public** **function** **getPasswordHash**(): **string**

{

**return** **$this**->passwordHash;

}

**private** **function** **refreshAuthToken**()

{

**$this**->authToken = sha1(random\_bytes(100)) . sha1(random\_bytes(100));

}

Обратите внимание – при успешном входе auth token пользователя в базе обновляется – все его предыдущие сессии станут недействительными.

**src/MyProject/Controllers/UsersController.php**

**public** **function** **login**()

{

**if** (!**empty**($\_POST)) {

**try** {

$user = User::login($\_POST);

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('users/login.php', ['error' => $e->getMessage()]);

**return**;

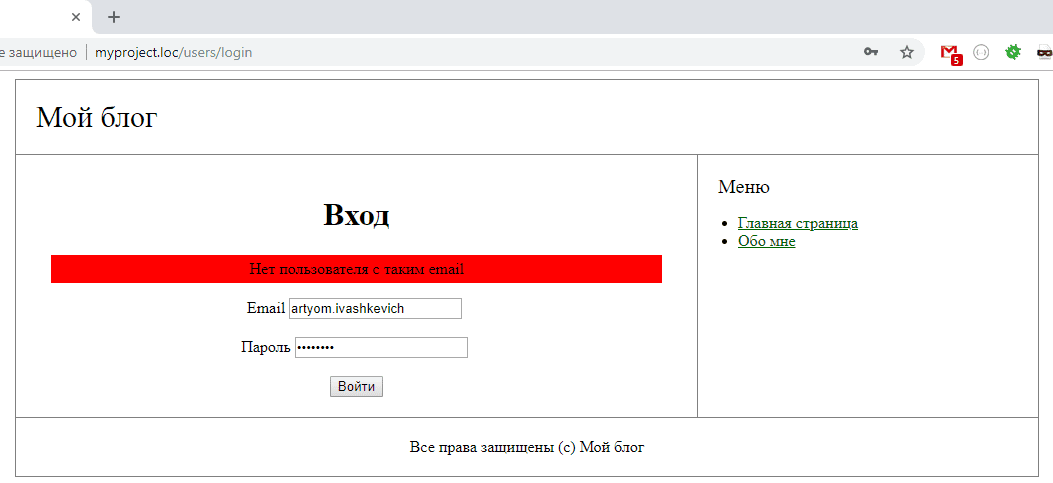
}

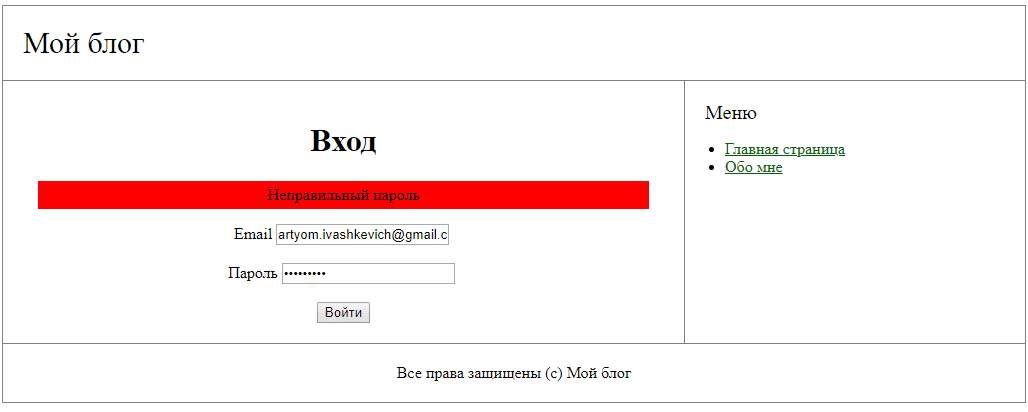
}

**$this**->view->renderHtml('users/login.php');

}

Проверяем, что ошибки корректно обрабатываются. Для этого пробуем вводить некорректные логин и пароль, а также отправлять форму с пустыми полями.

[](https://php.zone/uploads/img/1fa4f4f8bb6d30a8e5e4f3ab34b36226cc08ecdad9f1346083f2e2c30ba92fc0.png)

[](https://php.zone/uploads/img/d62a1bd74beb19253b7867ed1366c23fe879b9c1ea222cbe2e69c000552032c6.png)

Отлично, теперь нужно добавить обработку ситуации, когда логин и пароль верны и метод login в модели User вернул нам пользователя.

Создадим специальный сервис, который будет работать с пользовательскими сессиями через Cookie. Назовём его UsersAuthService.

**src/MyProject/Models/Users/UsersAuthService.php**

**<?php**

**namespace** **MyProject**\**Models**\**Users**;

**class** **UsersAuthService**

{

**public** **static** **function** **createToken**(User $user): **void**

{

$token = $user->getId() . ':' . $user->getAuthToken();

setcookie('token', $token, 0, '/', '', **false**, **true**);

}

**public** **static** **function** **getUserByToken**(): ?**User**

{

$token = $\_COOKIE['token'] ?? '';

**if** (**empty**($token)) {

**return** **null**;

}

[$userId, $authToken] = explode(':', $token, 2);

$user = User::getById((int) $userId);

**if** ($user === **null**) {

**return** **null**;

}

**if** ($user->getAuthToken() !== $authToken) {

**return** **null**;

}

**return** $user;

}

}

А теперь мы можем использовать его для удобного создания нужной Cookie в контроллере.

**src/MyProject/Controllers/UsersController.php**

**public** **function** **login**()

{

**if** (!**empty**($\_POST)) {

**try** {

$user = User::login($\_POST);

UsersAuthService::createToken($user);

header('Location: /');

**exit**();

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('users/login.php', ['error' => $e->getMessage()]);

**return**;

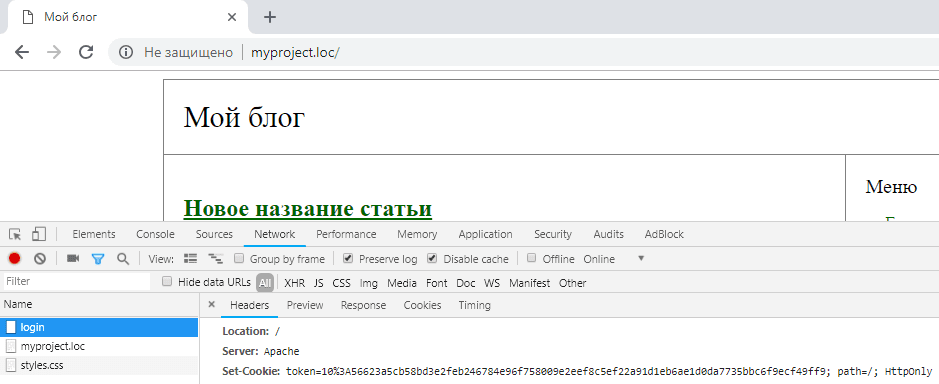
}

}

**$this**->view->renderHtml('users/login.php');

}

Теперь откроем консоль разработчика в Google Chrome и введем правильные логин и пароль. Видим, что нас перекинуло на главную страницу нашего блога, и что была установлена Cookie с именем token.

[](https://php.zone/uploads/img/bb17ad9054cd511397df2485264d9a748aebd9c84651aeb4fa907390575b777b.png)

Теперь нам нужно научиться передавать пользователя во View. Согласитесь, пользователь нам понадобится почти на каждой странице сайта, в каждом экшене. И будет неудобно каждый раз при рендеринге шаблона прокидывать пользователя:

**src/MyProject/Controllers/MainController.php**

**public** **function** **main**()

{

$articles = Article::findAll();

**$this**->view->renderHtml('main/main.php', [

'articles' => $articles,

'user' => UsersAuthService::getUserByToken()

]);

}

Поэтому мы сделаем во View возможность добавлять переменные еще перед рендерингом, вот так:

**src/MyProject/View/View.php**

**<?php**

**namespace** **MyProject**\**View**;

**class** **View**

{

**private** $templatesPath;

**private** $extraVars = [];

**public** **function** **\_\_construct**(string $templatesPath)

{

**$this**->templatesPath = $templatesPath;

}

**public** **function** **setVar**(string $name, $value): **void**

{

**$this**->extraVars[$name] = $value;

}

**public** **function** **renderHtml**(string $templateName, array $vars = [], int $code = 200)

{

http\_response\_code($code);

extract(**$this**->extraVars);

extract($vars);

ob\_start();

**include** **$this**->templatesPath . '/' . $templateName;

$buffer = ob\_get\_contents();

ob\_end\_clean();

**echo** $buffer;

}

}

И теперь мы можем в контроллерах прямо в конструкторах задать нужные переменные.

**src/MyProject/Controllers/MainController.php**

*/\*\* @var User|null \*/*

**private** $user;

**public** **function** **\_\_construct**()

{

**$this**->user = UsersAuthService::getUserByToken();

**$this**->view = **new** View(**\_\_DIR\_\_** . '/../../../templates');

**$this**->view->setVar('user', **$this**->user);

}

И добавить в шапке сайта (в шаблонах) вывод пользователя, если он был передан во View:

**templates/header.php**

**<!DOCTYPE html>**

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Мой блог</title>

<link rel="stylesheet" href="/styles.css">

</head>

<body>

<table class="layout">

<tr>

<td colspan="2" class="header">

Мой блог

</td>

</tr>

<tr>

<td colspan="2" style="text-align: right">

**<?**= !**empty**($user) ? 'Привет, ' . $user->getNickname() : 'Войдите на сайт' **?>**

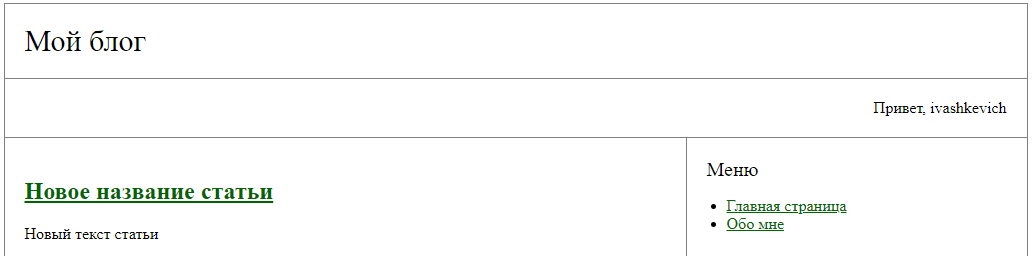
</td>

</tr>

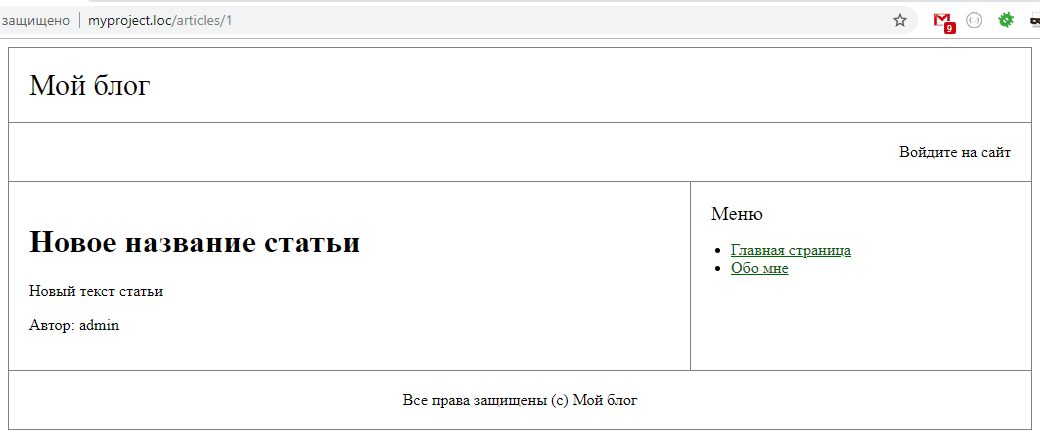
<tr>

<td>

Обновим теперь страничку и увидим приветствие.

[](https://php.zone/uploads/img/34f5e5ea92ff105a3d52df3a3df0121daf766cfa4f3b13b36206264eb39fecbc.png)

Если мы сейчас перейдём на страницу со статьёй, то увидим, что система просит нас залогиниться.

[](https://php.zone/uploads/img/7a283746d55dfd7841b7e06f94abca736564f921689a75bf770f3e49286e0ffc.png)

Это потому что мы в контроллере статей не прокинули пользователя во View. Давайте добавим тот же код, что и в конструкторе MainController.

**src/MyProject/Controllers/ArticlesController.php**

*/\*\* @var User|null \*/*

**private** $user;

**public** **function** **\_\_construct**()

{

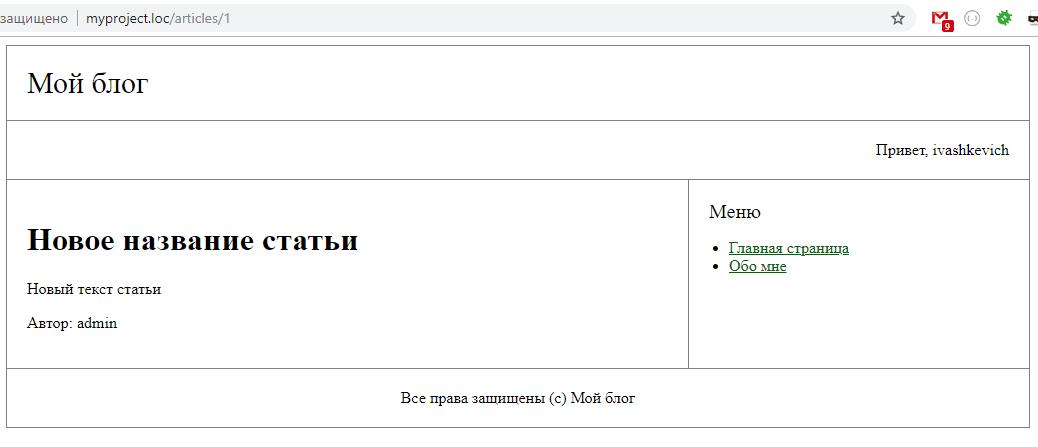
**$this**->user = UsersAuthService::getUserByToken();

**$this**->view = **new** View(**\_\_DIR\_\_** . '/../../../templates');

**$this**->view->setVar('user', **$this**->user);

}

Обновим страничку, и увидим, что теперь пользователь был передан в шаблон.

[](https://php.zone/uploads/img/86e390727f95566b1aea79b9bb428195488132e3fe2662bc8383e40c463a9c5f.png)

Прежде чем добавлять этот же код в контроллер пользователей давайте подумаем – ведь этот код будет одинаковым во всех трех контроллерах. Так давайте же создадим отдельный абстрактный контроллер, куда поместим этот код, а все наши контроллеры просто от него отнаследуем.

**src/MyProject/Controllers/AbstractController.php**

**<?php**

**namespace** **MyProject**\**Controllers**;

**use** **MyProject**\**Models**\**Users**\**User**;

**use** **MyProject**\**Models**\**Users**\**UsersAuthService**;

**use** **MyProject**\**View**\**View**;

**abstract** **class** **AbstractController**

{

*/\*\* @var View \*/*

**protected** $view;

*/\*\* @var User|null \*/*

**protected** $user;

**public** **function** **\_\_construct**()

{

**$this**->user = UsersAuthService::getUserByToken();

**$this**->view = **new** View(**\_\_DIR\_\_** . '/../../../templates');

**$this**->view->setVar('user', **$this**->user);

}

}

Обратите внимание, свойства user и view теперь с типом protected – они будут доступны в наследниках. Ну а теперь нам достаточно просто отнаследоваться в наших контроллерах от этого класса и можно удалить в них конструкторы и свойства view и user – они будут унаследованы от AbstractController. Это существенно упростит их код.

**src/MyProject/Controllers/MainController.php**

**<?php**

**namespace** **MyProject**\**Controllers**;

**use** **MyProject**\**Models**\**Articles**\**Article**;

**class** **MainController** **extends** **AbstractController**

{

**public** **function** **main**()

{

$articles = Article::findAll();

**$this**->view->renderHtml('main/main.php', ['articles' => $articles]);

}

}

**src/MyProject/Controllers/ArticlesController.php**

**<?php**

**namespace** **MyProject**\**Controllers**;

**use** **MyProject**\**Exceptions**\**NotFoundException**;

**use** **MyProject**\**Models**\**Articles**\**Article**;

**use** **MyProject**\**Models**\**Users**\**User**;

**class** **ArticlesController** **extends** **AbstractController**

{

**public** **function** **view**(int $articleId)

{

...

**src/MyProject/Controllers/UsersController.php**

**<?php**

**namespace** **MyProject**\**Controllers**;

**use** **MyProject**\**Exceptions**\**InvalidArgumentException**;

**use** **MyProject**\**Models**\**Users**\**User**;

**use** **MyProject**\**Models**\**Users**\**UserActivationService**;

**use** **MyProject**\**Models**\**Users**\**UsersAuthService**;

**use** **MyProject**\**Services**\**EmailSender**;

**class** **UsersController** **extends** **AbstractController**

{

**public** **function** **signUp**()

{

...

Теперь можно пройтись по всем страничкам сайта и убедиться, что всё по-прежнему работает. Если теперь нам нужно будет добавить какой-то функционал для всех контроллеров, то мы просто сделаем это в AbstractController.

Ну вот и всё – наша система авторизации готова! Разумеется, есть еще несколько вещей, которые нужно сделать. Их вы реализуете самостоятельно в домашнем задании.

Текущее состояние проекта [на гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/b5649d3dba2e53c302059df8b529d9295f3d4b57).

## Домашнее задание

* Если пользователь залогинен, то сделайте рядом с приветствием ссылку для выхода из этой учетной записи. Реализуйте функционал для разлогинивания (роутинг + экшен, удаляющий cookie).

[Разлогин](https://php.zone/uploads/img/cf050ed388dc778deb702404bd216fe37d1a22775cae2c5f051d3c75dd86abe4.png)

* Если пользователь не залогинен, показывайте ссылки для регистрации и входа на сайт.

Этот урок набрал набрал достаточно большое количество комментариев и дальнейшее его комментирование отключено. Если вы хотели убедиться в правильности выполнения ДЗ или у вас возник вопрос по уроку, посмотрите ранее добавленные комментарии, кликнув по кнопке ниже. Скорее всего вы найдете там то, что искали. Если это не помогло - задайте вопрос в чате в телеграме - <https://t.me/php_zone>

* [Sparkqy](https://php.zone/user/2514) 03.10.2018 в 17:44
* Можешь кинуть код реализации ?
* [ivashkevich](https://php.zone/user/1) 04.10.2018 в 07:18
* Код экшена? Или вообще весь?)
* [Sparkqy](https://php.zone/user/2514) 04.10.2018 в 16:35
* У меня почему-то не получается разлогинить пользователя, я пытаюсь удалить куки с токеном и после этого вызвать getUserByToken() что бы "удалить" пользователя (null), но безуспешно. Или это по другому делается?
* [ivashkevich](https://php.zone/user/1) 06.10.2018 в 14:23
* Просто удаляйте куки и перенаправляйте на другую страницу.
* [Sparkqy](https://php.zone/user/2514) 08.10.2018 в 09:35
* так при этом же остается объект пользователя на всех страницах, то есть любой контроллер да и вьюхи будут его видеть и это = что он все еще в системе (у нас же все проверки в шаблонах и контроллерах на !empty($user)). Разве не так?
* [ivashkevich](https://php.zone/user/1) 08.10.2018 в 23:06
* Он у вас там берется из cookie. Не будет куки - не будет и юзера.
* [Kirill.K](https://php.zone/user/2501) 14.10.2018 в 17:29
* **Header:**
* <td colspan="2" style="text-align: right">
* **<?php** **if**(!**empty**($user)): **?>**
* Привет, **<?**= $user->getNickname() **?>** | <a href="http://myproject.loc/users/logOut">Выйти</a>
* **<?php** **else**: **?>**
* <a href="http://myproject.loc/users/login">Войти</a> | <a href="http://myproject.loc/users/register">Зарегестрироваться</a>
* **<?** **endif**; **?>**
* </td>
* **UsersController:**
* **public** **function** **logOut**()
* {
* setcookie('token', '', -1, '/', '', **false**, **true**);
* header('Location: /');
* }
* **routes:**
* '~^users/logOut~' => [\MyProject\Controllers\UsersController::class, 'logOut'],
* [ivashkevich](https://php.zone/user/1) 15.10.2018 в 23:03
* Отлично!
* [Dram](https://php.zone/user/151) 21.01.2023 в 18:54
* ~^users/logOut~
* адрес url должен быть в нижнем регистре

# 30 Добавляем статьи в блог на PHP

В этом уроке мы с вами добавим самый главный функционал блога – добавление новых записей.

В одном из прошлых уроков мы уже добавили роут /articles/add и соответствующий экшен ArticlesController->add(). Так он выглядит сейчас:

**src/MyProject/Controllers/ArticlesController.php**

**public** **function** **add**(): **void**

{

$author = User::getById(1);

$article = **new** Article();

$article->setAuthor($author);

$article->setName('Новое название статьи');

$article->setText('Новый текст статьи');

$article->save();

$article->delete();

var\_dump($article);

}

Теперь давайте сделаем форму, которую можно будет заполнить в браузере и создавать через неё новые записи.

**templates/articles/add.php**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<h1>Создание новой статьи</h1>

**<?php** **if**(!**empty**($error)): **?>**

<div style="color: red;">**<?**= $error **?>**</div>

**<?php** **endif**; **?>**

<form action="/articles/add" method="post">

<label for="name">Название статьи</label><br>

<input type="text" name="name" id="name" value="<?= $\_POST['name'] ?? '' ?>" size="50"><br>

<br>

<label for="text">Текст статьи</label><br>

<textarea name="text" id="text" rows="10" cols="80">**<?**= $\_POST['text'] ?? '' **?>**</textarea><br>

<br>

<input type="submit" value="Создать">

</form>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Теперь сделаем набросок для контроллера, чтобы просто увидеть нашу форму.

**src/MyProject/Controllers/ArticlesController.php**

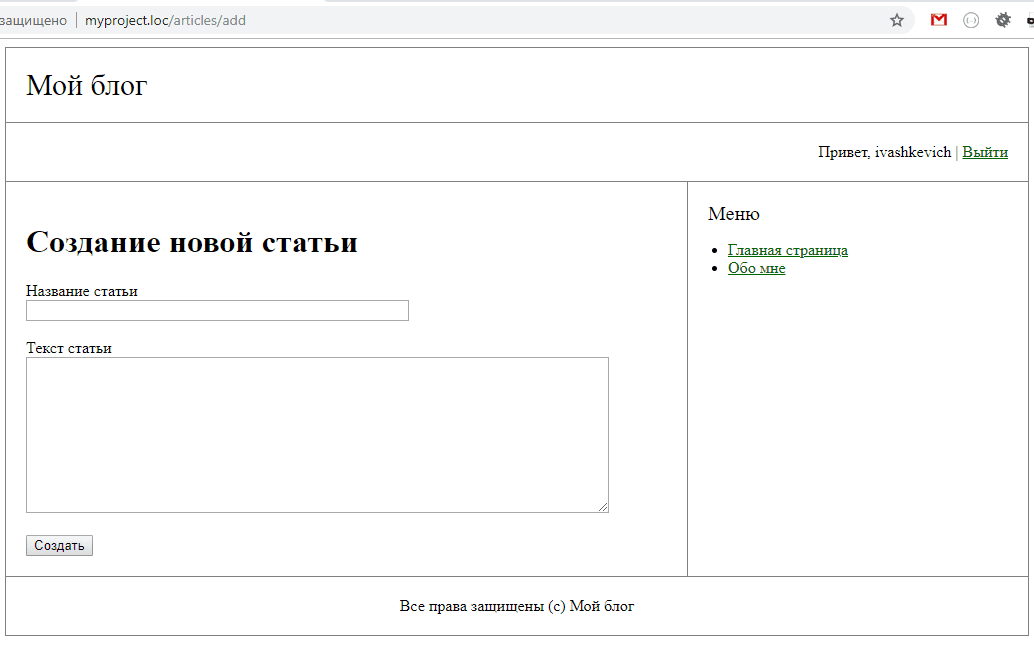
**public** **function** **add**(): **void**

{

**$this**->view->renderHtml('articles/add.php');

}

Посмотрим на результат:

[](https://php.zone/uploads/img/404d0542b705304b87a79159019972fdf756b8015213434f8908285ef0889bce.png)

Итак, в форме у нас есть 2 поля: name и text. На их основе мы должны научиться создавать статьи. Кроме того, в контроллере у нас также есть свойство user – текущий пользователь. Его нужно будет указывать автором при создании статей. Если мы соберем все эти три значения – мы сможем создать статью.

Первым делом нужно убедиться, что пользователь авторизован. Если это не так – будем кидать исключение в контроллере и ловить его во фронт-контроллере. Назовём такое исключение UnauthorizedException.

**src/MyProject/Exceptions/UnauthorizedException.php**

**<?php**

**namespace** **MyProject**\**Exceptions**;

**class** **UnauthorizedException** **extends** \**Exception**

{

}

Добавляем проверку в самом начале экшена.

**src/MyProject/Controllers/ArticlesController.php**

**public** **function** **add**(): **void**

{

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**$this**->view->renderHtml('articles/add.php');

}

Обрабатываем исключение во фронт-контроллере. Добавляем в конце еще один catch.

**www/index.php**

...

} **catch** (\MyProject\Exceptions\UnauthorizedException $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('401.php', ['error' => $e->getMessage()], 401);

}

И, наконец, добавляем шаблон для ошибки.

**templates/errors/401.php**

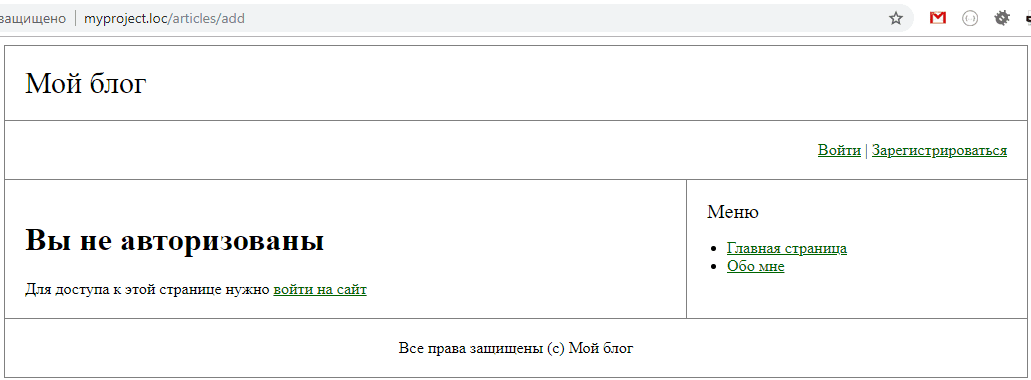
**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<h1>Вы не авторизованы</h1>

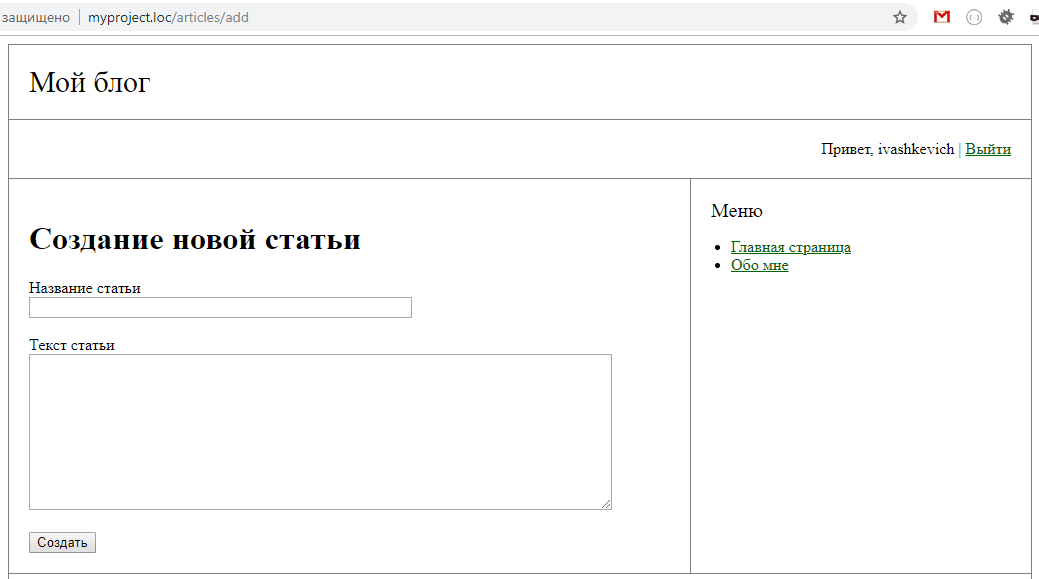
Для доступа к этой странице нужно <a href="/users/login">войти на сайт</a>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Разлогиниваемся на сайте и проверяем, что ошибка действительно появляется.

[](https://php.zone/uploads/img/28270c4c062378049df23eeb6c033bd75c6b16faf033180d73091e9c9922debd.png)

Затем входим на сайт и видим, что ошибка пропала и нам снова доступна форма для создания статьи.

[](https://php.zone/uploads/img/c76ab44f5ed1ef08a9937c080c52ae2ee87ef638c745e94d25611d6f3decd3c2.png)

Теперь можно создать в модели статьи метод для создания новой статьи.

**src/MyProject/Models/Articles/Article.php**

**public** **static** **function** **createFromArray**(array $fields, User $author): **Article**

{

**if** (**empty**($fields['name'])) {

**throw** **new** InvalidArgumentException('Не передано название статьи');

}

**if** (**empty**($fields['text'])) {

**throw** **new** InvalidArgumentException('Не передан текст статьи');

}

$article = **new** Article();

$article->setAuthor($author);

$article->setName($fields['name']);

$article->setText($fields['text']);

$article->save();

**return** $article;

}

Дописываем экшен:

**src/MyProject/Controllers/ArticlesController.php**

**public** **function** **add**(): **void**

{

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**if** (!**empty**($\_POST)) {

**try** {

$article = Article::createFromArray($\_POST, **$this**->user);

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('articles/add.php', ['error' => $e->getMessage()]);

**return**;

}

header('Location: /articles/' . $article->getId(), **true**, 302);

**exit**();

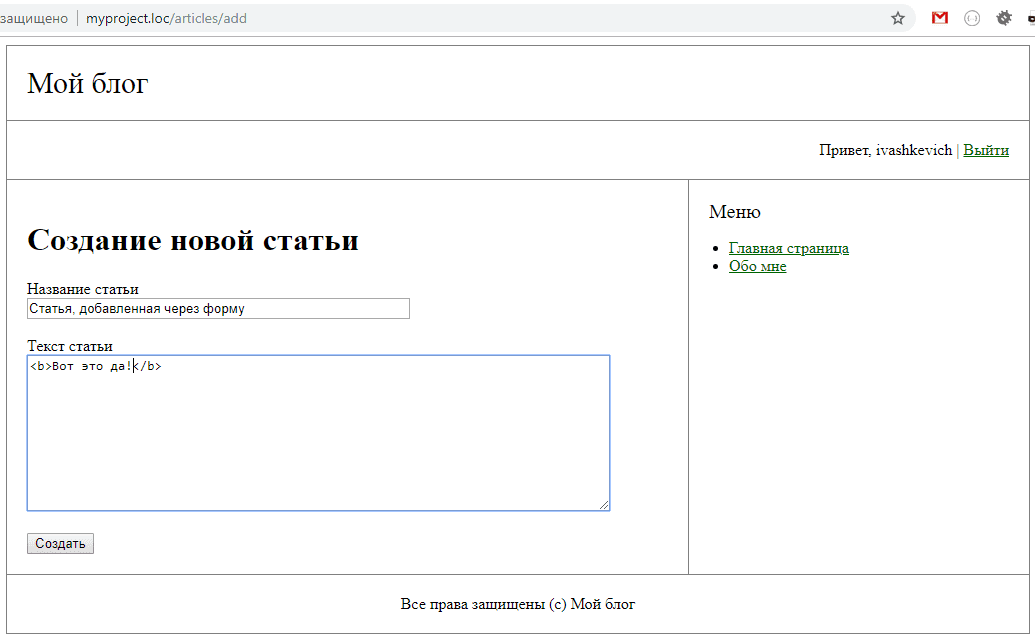
}

**$this**->view->renderHtml('articles/add.php');

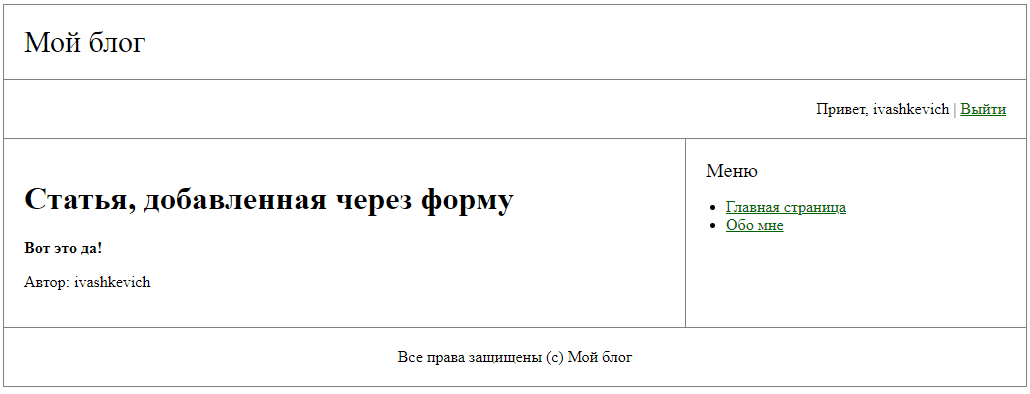
}

Здесь мы пытаемся создать новую статью, и если возникают ошибки – то мы показываем их в шаблоне. Если же все проходит хорошо – то мы переадресовываем пользователя на страничку с новой статьёй. Давайте теперь попробуем.

Заполняем формочку по адресу <http://myproject.loc/articles/add>

[](https://php.zone/uploads/img/e62b1982a86dc39cdab7d99bb924f3ea5d6cbce48a0319e056366924d326e6a4.png)

И вуаля – создалась новая статья.

[](https://php.zone/uploads/img/1bf8c1aef5dfe5ad1846815221a589880829159b8425e50f61eb02f10ad3c559.png)

А теперь – за домашку.

Текущая версия проекта [на гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/eb85848f2171ac93204c144f20cc741691b10868).

## Домашнее задание

Сделайте так, чтобы добавлять статьи могли только пользователи с правами админа. Если это не так - бросайте исключение с новым типом - Forbidden. При этом страница должна вернуть код 403.

[Kirill.K](https://php.zone/user/2501) 15.10.2018 в 21:16

**ArticlesController:**

...

**if** (**$this**->user->getRole() !== admin) {

**throw** **new** Forbidden();

}

...

**Templates\errors\403:**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<h1>Для добавления статьи нужно обладать правами администратора</h1>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

**Index:**

**catch** (\MyProject\Exceptions\Forbidden $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage()], 403);

}

[ivashkevich](https://php.zone/user/1) 15.10.2018 в 23:10

Хорошо, но в Templates\errors\403 стоит выводить переменную error в шаблоне, а не хардкодить текст ошибки.

А для проверки того, является ли юзер админом можно создать в модели User метод isAdmin, который будет это проверять:

**if**(!$user->isAdmin()) {

...

[Kirill.K](https://php.zone/user/2501) 16.10.2018 в 22:35

Так?:  
**User:**

**public** **function** **isAdmin**(): **bool**

{

**return** **$this**->role === admin;

}

**ArticlesController:**

**if**(!**$this**->user->isAdmin()) {

**throw** **new** Forbidden('Для добавления статьи нужно обладать правами администратора');

}

**Templates\errors\403:**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

**<?**= $error **?>**

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

[ivashkevich](https://php.zone/user/1) 17.10.2018 в 00:39

Perfect!

[Kirill.K](https://php.zone/user/2501) 17.10.2018 в 19:43

Ещё хотел спросить - при отображении ошибки наша шапка с ником юзера слетает. Куки живы, но отсутствует объект $user, от наличия которого у нас зависит вывод. Как бы это исправить, не могу додуматься?

[ivashkevich](https://php.zone/user/1) 19.10.2018 в 00:21

На странице с ошибкой не нужны данные о пользователе. Ошибка - это ошибка, больше ничего быть там не должно.

[Kirill.K](https://php.zone/user/2501) 19.10.2018 в 19:02

Хорошо) Но тогда нужно сделать отдельный **HeaderForErrors:**

**<!DOCTYPE html>**

<html lang="ru">

<head>

<meta charset="UTF-8">

<title>Мой блог</title>

<link rel="stylesheet" href="/styles.css">

</head>

<body>

<table class="layout">

<tr>

<td colspan="2" class="header">

Мой блог

</td>

</tr>

<tr>

<td>

А то наш базовый предлагает войти или зарегистрироваться при выводе ошибки

[ivashkevich](https://php.zone/user/1) 19.10.2018 в 21:55

Да, можно так.

[andreskrip](https://php.zone/user/10557) 15.02.2020 в 13:10

А нельзя просто передать во фронт-контроллере данные user и не делать новый шаблон?

**catch** (\MyProject\Exceptions\ForbiddenException $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage(), 'user' => \MyProject\Services\UsersAuthService::getUserByToken()], 403);

}

У меня тогда всё отображается и в стандартном шаблоне

[ivashkevich](https://php.zone/user/1) 18.02.2020 в 15:52

Не очень хорошо фронт-контроллеру обращаться к слою бизнес-логики. Он всё же должен решать архитектурные задачи.

[tomsonst](https://php.zone/user/2565) 20.12.2018 в 21:21

ArticlesController.php

**if** (**$this**->user->getRole !== 'admin') {

**throw** **new** Forbidden();

}

index.php

**catch** (\MyProject\Exceptions\Forbidden $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage(), 'user' => UsersAuthService::getUserByToken()], 403);

}

Создаем файл Forbidden.php и создаем шаблон 403.php

[ivashkevich](https://php.zone/user/1) 22.12.2018 в 12:57

Хорошо

[excent63](https://php.zone/user/5583) 07.04.2019 в 11:41

Добрый день! д/з:

**ArticlesController.php**

**public** **function** **add**(): **void**

{

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**if** (**$this**->user->getRole() !== 'admin') {

**throw** **new** ForbiddenException('У вас нет прав, за помощью обратитесь к администратору!');

}

...

**403.php**

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<strong>**<?**= $error **?>**</strong>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

**index.php**

...

**catch** (\MyProject\Exceptions\ForbiddenException $e) {

$view = **new** MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage(), 'user' => \MyProject\Services\UsersAuthService::getUserByToken()], 403);

}

[ivashkevich](https://php.zone/user/1) 08.04.2019 в 21:36

Отлично! У юзера можно завести отдельный метод:

**public** **function** **isAdmin**(): **bool**

{

**return** **$this**->getRole() === 'admin';

}

[excent63](https://php.zone/user/5583) 08.04.2019 в 22:27

Вот дошёл уже до 31 урока, вроде задачи в домашках понятны, пока получается их решить))) Но вот не совсем доходит момент, когда нужно создать отдельный метод или какой тип этот метод должен отдать) Немного не понятно где создать тот или иной метод, видимо где то недочитал или недопонял) Надо как то оптимизировать своё обучение или начинать доводить блог до ума чтобы началось включение воображения, побольше практики или вернуться к началу и перечитать все по новой )))

[ivashkevich](https://php.zone/user/1) 08.04.2019 в 22:33

Все придет с практикой, не переживай.

[excent63](https://php.zone/user/5583) 08.04.2019 в 22:35

Будем стараться) Главное что есть кому задать вопрос и получить на него правильный ответ! Спасибо за обучение!)

[ivashkevich](https://php.zone/user/1) 09.04.2019 в 09:27

Пожалуйста)

[Metey](https://php.zone/user/7866) 25.07.2019 в 18:30

Вначале сделал, а затем учел некоторые рекомендации и немного переделал:  
в index добавил ловлю эксепшнов:

**catch** (\MyProject\Exceptions\ForbiddenException $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage()], 403);

}

добавил метод isAdmin в User.php :

**public** **function** **isAdmin**(): **bool**

{

**return** **$this**->role === 'admin';

}

и в метод ArticlesController add() :

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**if** (!**$this**->user->isAdmin()) {

**throw** **new** ForbiddenException('Для доступа к данной странице необходимы права администратора!');

}

........

сделал отдельно headerError.php и в нем изменил это :

<tr>

<td colspan="2" style="text-align: right">

<a href="/users/login">Войти как администратор</a> |

<a href="/users/register">Зарегистрироваться</a>

</td>

</tr>

[ivashkevich](https://php.zone/user/1) 25.07.2019 в 18:56

А для чего headerError.php? Не вижу, чтобы он где-то использовался.

[Metey](https://php.zone/user/7866) 25.07.2019 в 19:14

забыл сюда вставить шаблон 403.php

**<?php** **include** **\_\_DIR\_\_** . '/../headerError.php'; **?>**

<h1>**<?**= $error **?>**</h1>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

[ivashkevich](https://php.zone/user/1) 25.07.2019 в 19:17

Так ведь, то что прав нет, ещё не означает, что пользователя надо просить войти. Ну да ладно, это уже вопрос бизнес логики. В целом ок.

[Metey](https://php.zone/user/7866) 25.07.2019 в 19:19

понял)) спасибо, возьму на заметку

[Iliusha99](https://php.zone/user/7802) 03.08.2019 в 16:42

Articles Controller:

**if** (**$this**->user->getRole() !== 'admin'){

**throw** **new** AccessForbidden('Forbidden', 403);

}

User Model:

**public** **function** **getRole**(): **string**

{

**return** **$this**->role;

}

index.php:

...

**catch** (AccessForbidden $e) {

$view = **new** View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage()], 403);

}

[ivashkevich](https://php.zone/user/1) 03.08.2019 в 20:36

Норм. Но проверку на то, что юзер админ стоит проверять на уровне модели. Для этого стоит завести в ней метод isAdmin(): bool

[Moskva](https://php.zone/user/4665) 07.08.2019 в 15:53

User.php

...

**public** **function** **IsAdmin**(): **bool**

{

**return** **$this**->role === 'admin';

}

...

ArticlesController.php

**public** **function** **add**(): **void** // добавление статьи в БД

{

**if**(**$this**->user === **null**){

**throw** **new** UnauthorizedException();

}

**if**(!**$this**->user->IsAdmin()){

**throw** **new** ForbiddenException();

}

...

index.php

...

} **catch** (\MyProject\Exceptions\ForbiddenException $e){

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage(), 'user' => \MyProject\Services\UsersAuthService::getUserByToken()], 403);

}

403.php

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<h1>У вас нет прав администратора.</h1>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

[ivashkevich](https://php.zone/user/1) 07.08.2019 в 18:50

**public** **function** **IsAdmin**(): **bool**

Имена методов пишутся с маленькой буквы.

В остальном все хорошо.

[Reechniy](https://php.zone/user/7539) 02.09.2019 в 13:48

index:

**catch** (\MyProject\Exceptions\ForbiddenException $e) {

$view = **new** \MyProject\View\View(**\_\_DIR\_\_** . '/../templates/errors');

$view->renderHtml('403.php', ['error' => $e->getMessage()], 403);

}

User:

**public** **function** **isAdmin**(): **bool**

{

**return** **$this**->role === 'admin';

}

AticleController:

**if**(!**$this**->user->isAdmin()) {

**throw** **new** ForbiddenException('Недостаточно прав, для добавления статьи');

}

**$this**->view->renderHtml('articles/add.php');

**return**;

}

Создал исключение ForbiddenException.php и 403.php как у всех

[ivashkevich](https://php.zone/user/1) 02.09.2019 в 18:47

Отлично

# 31 Редактируем статьи в блоге на PHP

Сегодня мы создадим еще один компонент нашего блога – систему редактирования статей.

В одном из прошлых уроков мы уже добавляли роутинг для редактирования статей:

**src/routes.php**

'~^articles/(\d+)/edit$~' => [\MyProject\Controllers\ArticlesController::class, 'edit'],

Он останется прежним. Переходим к экшену. Сейчас он выглядит вот так:

**src/MyProject/Controllers/ArticlesController.php**

**public** **function** **edit**(int $articleId)

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

$article->setName('Новое название статьи');

$article->setText('Новый текст статьи');

$article->save();

}

Нам требуется его переписать, чтобы статья обновлялась данными из POST-запроса. Делаем:

**public** **function** **edit**(int $articleId)

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**if** (!**empty**($\_POST)) {

**try** {

$article->updateFromArray($\_POST);

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('articles/edit.php', ['error' => $e->getMessage(), 'article' => $article]);

**return**;

}

header('Location: /articles/' . $article->getId(), **true**, 302);

**exit**();

}

**$this**->view->renderHtml('articles/edit.php', ['article' => $article]);

}

Теперь давайте добавим соответсвующий шаблон.

**templates/articles/edit.php**

**<?php**

*/\*\**

*\* @var \MyProject\Models\Articles\Article $article*

*\*/*

**include** **\_\_DIR\_\_** . '/../header.php';

**?>**

<h1>Редактирование статьи</h1>

**<?php** **if**(!**empty**($error)): **?>**

<div style="color: red;">**<?**= $error **?>**</div>

**<?php** **endif**; **?>**

<form action="/articles/<?= $article->getId() ?>/edit" method="post">

<label for="name">Название статьи</label><br>

<input type="text" name="name" id="name" value="<?= $\_POST['name'] ?? $article->getName() ?>" size="50"><br>

<br>

<label for="text">Текст статьи</label><br>

<textarea name="text" id="text" rows="10" cols="80">**<?**= $\_POST['text'] ?? $article->getText() **?>**</textarea><br>

<br>

<input type="submit" value="Обновить">

</form>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

А теперь добавим соответствующий метод в модели:

**src/MyProject/Models/Articles/Article.php**

**public** **function** **updateFromArray**(array $fields): **Article**

{

**if** (**empty**($fields['name'])) {

**throw** **new** InvalidArgumentException('Не передано название статьи');

}

**if** (**empty**($fields['text'])) {

**throw** **new** InvalidArgumentException('Не передан текст статьи');

}

**$this**->setName($fields['name']);

**$this**->setText($fields['text']);

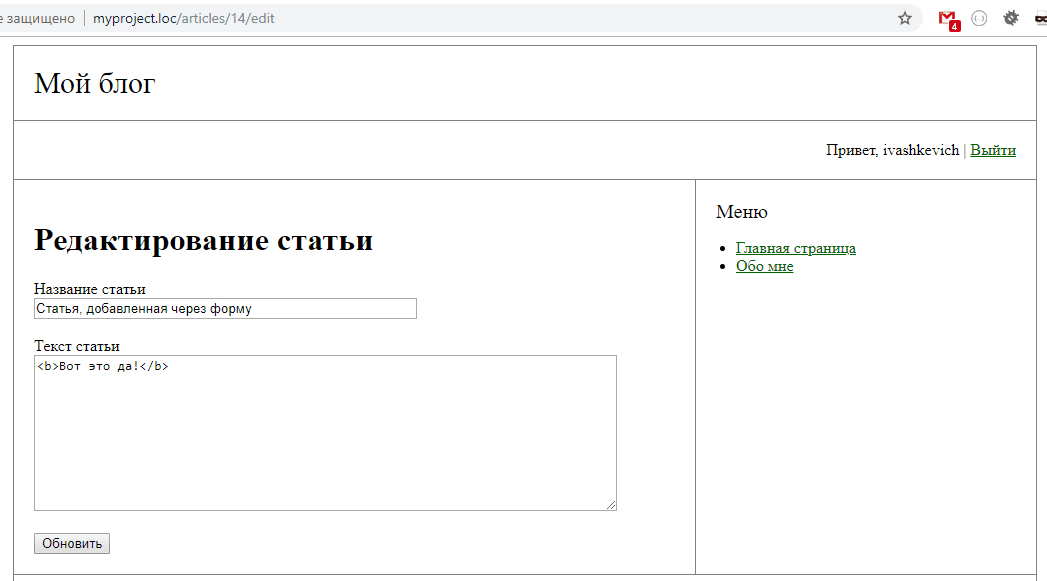
**$this**->save();

**return** **$this**;

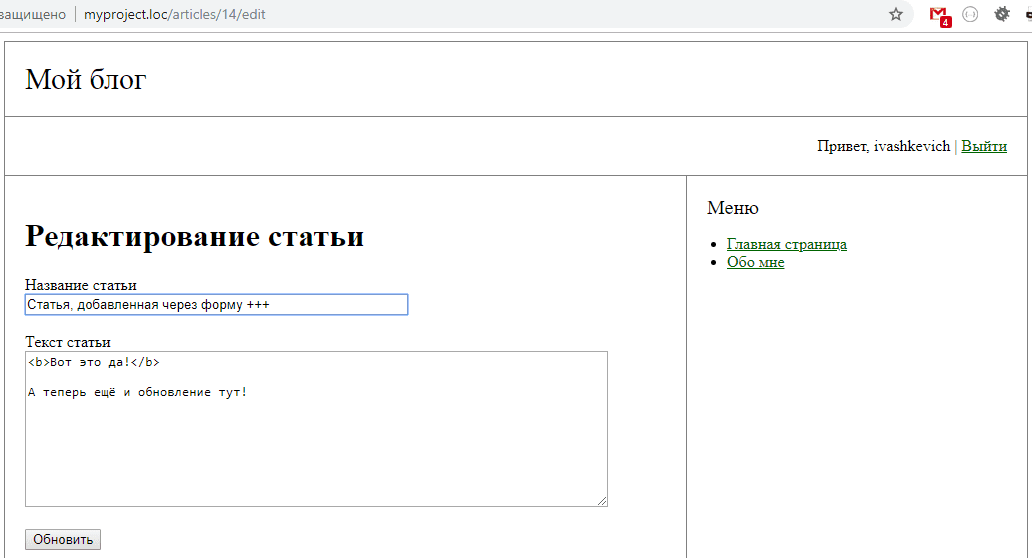
}

Заходим теперь на страничку редактирования статьи:

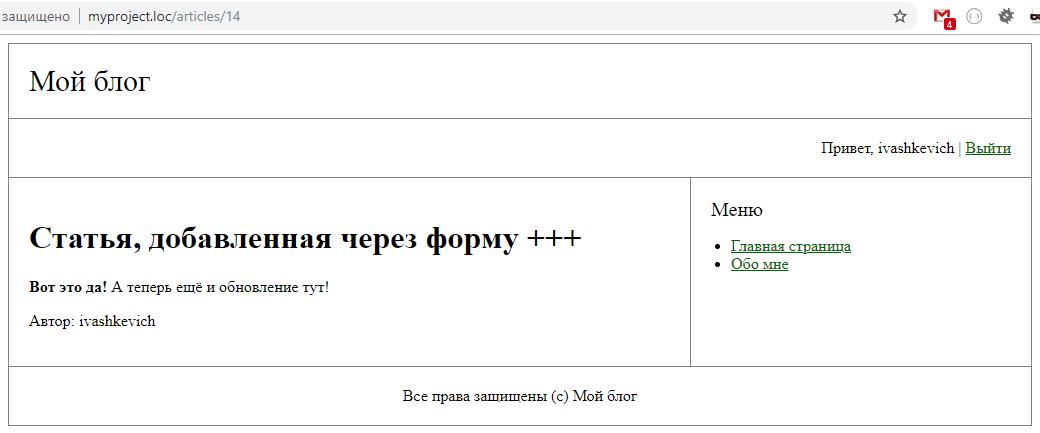
<http://myproject.loc/articles/14/edit>

[](https://php.zone/uploads/img/918e05793d382178136d0f23574f4878f09c8de3f9e7c9c97dd9fe37a18377e6.png)

Вводим какой-нибудь новый текст и заголовок.

[](https://php.zone/uploads/img/352a5d70cba829a9904748f2cfa6b06c770ad627eef397216a15ff7de23a1e61.png)

Нажимаем на кнопку обновления и вуаля – статья обновлена.

[](https://php.zone/uploads/img/b5d6794742859ccc1d3f417495da578020cd563c605066c7ab004ec32f7ca18a.png)

Текущая версия кода [на гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/0da717bb50d3b9646ea8f23abeff9253d6410fe5).

## Домашнее задание

* При попытке обновления статьи НЕ админом – бросайте исключение ForbiddenException, как в прошлом уроке.
* Добавьте ссылку на странице показа статьи с текстом «Редактировать», которая будет вести на страницу редактирования этой статьи.
* Сделайте так, чтобы эта ссылка показывалась только если пользователь залогинен и он админ.

[Kirill.K](https://php.zone/user/2501) 17.10.2018 в 20:35

**ArticlesController/edit:**

**if**(!**$this**->user->isAdmin()) {

**throw** **new** Forbidden('Для редактирования статьи нужно обладать правами администратора');

}

**Templates/articles/view:**

**<?php** **if**(**isset**($\_COOKIE['token']) **and** $user->isAdmin()): **?>**

<p><a href="http://myproject.loc/articles/<?= $article->getId() ?>/edit">Редактировать статью</a></p>

**<?** **endif**; **?>**

[ivashkevich](https://php.zone/user/1) 19.10.2018 в 00:23

**isset**($\_COOKIE['token'])

Этого достаточно, чтобы знать, что пользователь залогинен? А если в этой куке ерунда какая-нибудь бессмысленная записана? Тогда $user будет null, и при попытке вызвать у него метод произойдет ошибка.

[Kirill.K](https://php.zone/user/2501) 19.10.2018 в 19:10

Хм... тогда, наверное, так:

**<?php** **if**($\_COOKIE['token'] === $user->getId() . ':' . $user->getAuthToken() **and** $user->isAdmin()): **?>**

<p><a href="http://myproject.loc/articles/<?= $article->getId() ?>/edit">Редактировать статью</a></p>

**<?** **endif**; **?>**

Громоздко, конечно, как то получается...

[ivashkevich](https://php.zone/user/1) 19.10.2018 в 21:58

Да нет, все проще. Если пользователь не авторизован, то в $user будет null.  
Достаточно проверить:

**if** ($user !== **null** && $user->isAdmin())

[OneMoreTime](https://php.zone/user/13021) 26.05.2020 в 16:18

**if** ($user !== **null** && $user->isAdmin())

Наверное перед этим стоит делать проверку юзера на isset?

[ivashkevich](https://php.zone/user/1) 27.05.2020 в 17:57

У нас $user всегда в шаблоне есть же.

[Arhimedovich](https://php.zone/user/23760) 06.02.2022 в 16:57

А зачем проверять $user!==null? Если пользователь не залогинен то у него точно не может быть прав админа! Поэтому как мне кажется данная проверка избыточна, можно просто проверить является ли пользователь админом или нет. <?php if ($user->isAdmin()==true): ?> Или я ошибаюсь?

[ivashkevich](https://php.zone/user/1) 19.02.2022 в 19:06

Если пользователь не авторизован, то в $user будет null, при попытке вызова метода не у объекта а у null будет фатальная ошибка.

[ilyaOrlov](https://php.zone/user/3743) 02.12.2018 в 21:54

Проверил свой код несколько раз. Всё так, как вы написали. Но при редактировании, если передавать какое-то поле пустым, то вылетает ошибка о том, что $article->getId() становится null в шаблоне edit.php. Решил проблему передачей вместе с ошибкой еще и массива $article.

**$this**->view->renderHtml('articles/edit.php', ['error' => $e->getMessage(), 'article' => $article]);

Тогда все работает. В чем проблема, не подскажите?

[ivashkevich](https://php.zone/user/1) 02.12.2018 в 22:58

Была ошибка, исправил. Спасибо. В шаблоне использовалась $article, я почему-то подумал, что если произойдет ошибка, то в шаблоне мы статью не будем выводить.

[Metey](https://php.zone/user/7866) 27.07.2019 в 19:27

При попытке обновления статьи НЕ админом – бросайте исключение ForbiddenException, как в прошлом уроке.

**public** **function** **edit**(int $articleId): **void**

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**if** (!**$this**->user->isAdmin()) {

**throw** **new** ForbiddenException('Для доступа к данной странице необходимы права администратора!');

}

......

Добавьте ссылку на странице показа статьи с текстом «Редактировать», которая будет вести на страницу редактирования этой статьи.  
Сделайте так, чтобы эта ссылка показывалась только если пользователь залогинен и он админ.

<h1>**<?**= $article->getName() **?>**</h1>

<p>**<?**= $article->getText() **?>**</p>

<p>Автор: **<?**= $article->getAuthor()->getNickname() **?>**</p>

**<?if** ($user!==**null** && $user->isAdmin()):**?>**

<p><a href="/articles/<?= $article->getId() ?>/edit">Редактировать</a></p>

**<?endif?>**

[ivashkevich](https://php.zone/user/1) 28.07.2019 в 13:38

Отлично

[Iliusha99](https://php.zone/user/7802) 03.08.2019 в 17:57

Добавил метод в модель Юзера:

**public** **function** **allowEdit**(): **bool**

{

**if** (**$this** === **null** || **$this**->getRole() !== 'admin') {

**return** **false**;

}

**return** **true**;

}

В ArticleControllers при попытке переходить по ссылке ред. :

**if** (!**$this**->user->allowEdit()) {

**throw** **new** AccessForbidden();

}

И в шаблоне в зависимости от значения $user->allowEdit() решаем показать <<Редактировать>> или нет :

**<?php** **if**($user->allowEdit()):**?>** <p><a href="<?= $article->getId() ?>/edit">Редактировать</a></p> **<?php** **endif**; **?>**

[ivashkevich](https://php.zone/user/1) 03.08.2019 в 20:39

$this === null - такого не бывает. Если есть $this, значит есть объект. null там быть не может

[ivashkevich](https://php.zone/user/1) 03.08.2019 в 20:42

**public** **function** **allowEdit**(): **bool**

{

**if** (**$this** === **null** || **$this**->getRole() !== 'admin') {

**return** **false**;

}

**return** **true**;

}

Сокращается до

**public** **function** **allowEdit**(): **bool**

{

**return** **$this**->getRole() === 'admin';

}

А вообще, метод для проверки возможности редактирования должен быть у статьи, а не у пользователя. В этот метод аргументом должен передаваться объект пользователя, права которого нужно проверить.

[Iliusha99](https://php.zone/user/7802) 03.08.2019 в 20:44

ок, поправлю

[ivashkevich](https://php.zone/user/1) 03.08.2019 в 20:44

В шаблон лучше передавать уже готовую переменную, которая будет говорить о том, разрешено ли редактирование статьи, например, isEditable. Эта переменная должна формироваться в контроллере и передаваться во view.

[Moskva](https://php.zone/user/4665) 07.08.2019 в 18:24

1.ArticlesConroller.php

...

**if**(!**$this**->user->IsAdmin()){

**throw** **new** ForbiddenException();

}

...

2-3.view.php

...

**<?** **if**(!**empty**($user) && ($user->isAdmin())): **?>**

<hr>

<div style="text-align: right;"><a href="/articles/<?= $article->getId()?>/edit">Редактировать</a><div>

**<?** **endif**; **?>**

...

[ivashkevich](https://php.zone/user/1) 07.08.2019 в 18:57

if(!$this->user->IsAdmin()) - имена методов пишутся с маленькой буквы!

[Moskva](https://php.zone/user/4665) 08.08.2019 в 20:17

Не уследил, спасибо.

[artemship](https://php.zone/user/4977) 06.09.2019 в 18:21

**articles/view.php**

**<?php** **if** ($isEditable): **?>**

<p><a href="/articles/<?= $article->getId() ?>/edit">Редактировать</a></p>

**<?php** **endif**; **?>**

**ArticlesController**

**public** **function** **edit**(int $articleId): **void**

{

...

**if** (!**$this**->user->isAdmin()) {

**throw** **new** ForbiddenException('Статьи могут редактировать только администраторы');

}

...

**public** **function** **view**(int $articleId): **void**

{

...

**if** (**$this**->user === **null**) {

$isEditable = **false**;

} **else** {

$isEditable = (**$this**->user->isAdmin());

}

**$this**->view->renderHtml('articles/view.php', [

'article' => $article,

'isEditable' => $isEditable

]);

...

Тут вопрос: можно ли в экшене view проверку заменить на

**if** (**$this**->user !== **null**) {

$isEditable = (**$this**->user->isAdmin());

}

В таком случае в методе $this->view->renderHtml высвечивается предупреждение, что переменная $isEditable "might have not been defined". При этом ошибок в рендере шаблона нет. Как лучше поступить?

[ivashkevich](https://php.zone/user/1) 08.09.2019 в 09:37

$isEditable = (**$this**->user->isAdmin())

Для чего скобки?

Ответ на твой вопрос - нет, нельзя. Потому что если пользователь будет не залогинен, возникнет ошибка.

[andreskrip](https://php.zone/user/10557) 15.02.2020 в 14:15

Спасибо за урок!

**ArticlesController::edit**

**if** (!**$this**->user->isAdmin()) {

**throw** **new** ForbiddenException();

}

**templates/articles/view.php**

**<?php** **if** ($user !== **null** && $user->isAdmin()): **?>**

<a href="/articles/<?= $article->getId() ?>/edit">Редактировать</a>

**<?php** **endif**; **?>**

Вопрос не по уроку: немного неудобно было при отладке проверки на авторизацию постоянно заново заходить в одиночную статью, т.к. после входа редиректит на главную. Каким образом можно сделать редирект на ту страницу, с которой ты перешел на страницу авторизации? В гугле были предложения джавасриптом

header("location:javascript://history.go(-1)");

или через глобальную переменную (во-первых оно меня вернуло на страницу авторизации, вместо нужной страницы, а во-вторых пишут что это нехорошая практика в принципе)

header('Location: ' . $\_SERVER['HTTP\_REFERER']);

exit();

[ivashkevich](https://php.zone/user/1) 18.02.2020 в 15:54

Я бы использовал второй вариант - при переходе на страницу логина устанавливал бы куку с реферером и после логина переходил по ней. Хорошая практика или нет - решать вам) Если проблем она вам не создает, значит не такая уж и плохая.

[andreskrip](https://php.zone/user/10557) 18.02.2020 в 15:56

спасибо большое за совет)

[ivashkevich](https://php.zone/user/1) 18.02.2020 в 18:24

Пожалуйста)

[OneMoreTime](https://php.zone/user/13021) 23.03.2020 в 10:59

В контоллере статей есть повторяющиеся проверки в разных методах:

**if** (**$this**->user === **null**){

**throw** **new** UnAuthorizedException();

}

**if** (!**$this**->user->isAdmin()){

**throw** **new** ForbiddenException('You don\'t have Admin rights');

}

в модели статьи так же:

**if** (**empty**($fields['name'])){

**throw** **new** InvalidArgumentException('Name field cannot be empty');

}

**if** (**empty**($fields['text'])){

**throw** **new** InvalidArgumentException('Text field cannot be empty');

}

Если такие небольшие повторяющиеся участки встречаются всего два-три раза, стоит их выделять в отдельные общие методы? А то вроде получается масло масляное - повторяющийся код убираем, но делаем нагромождение методов на каждый чих. Или все таки лучше любую повторяемость обобщать отдельно , чтобы потом было легче и быстрее при необходимости модифицировать код?

[ivashkevich](https://php.zone/user/1) 23.03.2020 в 11:58

Такие небольшие кусочки можно вынести, но только если это даст реальное упрощение кода. Ну и есть "правило трёх" - если требуется такой же код в еще одном месте - возьми и скопируй, но если уже в трёх местах - вынеси в отдельный компонент (класс, метод, функцию и пр.)

[Dimitry](https://php.zone/user/13890) 08.04.2020 в 07:48

*//ArticlesController*

**public** **function** **view**(int $articleId)

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

**$this**->view->renderHtml('articles/view.php', [

'article' => $article,

'admin' =>**$this**->user->isAdmin()

]);

}

**public** **function** **edit**(int $articleId)

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException();

}

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**if** (!**$this**->user->isAdmin()){

**throw** **new** Forbidden();

}

**if** (!**empty**($\_POST)) {

**try** {

$article->updateFromArray($\_POST);

} **catch** (InvalidArgumentException $e) {

**$this**->view->renderHtml('articles/edit.php', ['error' => $e->getMessage(), 'article' => $article]);

**return**;

}

header('Location: /articles/' . $article->getId(), **true**, 302);

**exit**();

}

**$this**->view->renderHtml('articles/edit.php', ['article' => $article]);

}

*//view.php*

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<h1>**<?**= $article->getName() **?>**</h1>

<p>**<?**= $article->getText() **?>**</p>

<p>Автор: **<?**= $article->getAuthor()->getNickname() **?>**</p>

**<?**

**if**($admin){

**echo** '<a href="http://myproject.loc/articles/'.$article->getId().'/edit">Редактировать</a>';

}

**?>**

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Домашка

[ivashkevich](https://php.zone/user/1) 08.04.2020 в 08:29

В шаблоне правильнее тоже использовать имя isAdmin, потому что это не сама сущность админа, а только флажок.

[Dimitry](https://php.zone/user/13890) 08.04.2020 в 07:56

У меня вопрос: Когда происходит обработка исключений, то почему-то отображаются ссылки для входа и регистрации, хотя пользователь уже зарегистрирован и выполнен вход, при переключении на страницу главная все в порядке - пользователь авторизован, как это исправить?

[ivashkevich](https://php.zone/user/1) 08.04.2020 в 08:30

А в какой момент прокидывается пользователь в шаблон?

[Ed](https://php.zone/user/14956) 17.04.2020 в 20:52

1. Задание (ArticleController)
2. **if**($article === **null**) {
3. **throw** **new** NotFoundException();
4. }
5. **try** {
6. **if**(**$this**->user === **null**) {
7. **throw** **new** UnautorizedException();
8. }
9. **if**(**$this**->user->getRole() !== 'admin') {
10. **throw** **new** ForbiddenException();
11. }
12. } **catch**(ForbiddenException | UnautorizedException $e) {
13. header('Location: /');
14. **exit**();

}

Там же:

**public** **function** **view**(int $articleId)

{

$article = Article::getById($articleId);

**if**($article === **null**) {

**throw** **new** NotFoundException();

}

$author = User::getById($article->getAuthor()->getId());

$autorizedUserId = **$this**->user->getId();

$articleAccess = **false**;

**if**($autorizedUserId === $author->getId()) {

$articleAccess = **true**;

}

**$this**->view->renderHtml('articles\view.php',

['article' => $article,

'access' => $articleAccess

]);

}

templates/articles/View

**<?php** **if**($access): **?>**

<a href="/articles/<?=$article->getId()?>/edit"> Редактировать </a>

**<?php** **endif**; **?>**

Сделал так, чтобы только автор статьи мог редактировать её, а не любой admin)

[ivashkevich](https://php.zone/user/1) 18.04.2020 в 06:05

} **catch**(ForbiddenException | UnautorizedException $e) {

header('Location: /');

**exit**();

}

Что это за обработка исключений такая? Нужно завести соответствующие шаблоны и показывать информацию об ошибках пользователю.

Также нет смысла бросать исключения и ловить их в одном и том же слое приложения. Можно же сразу написать

**if**(**$this**->user === **null**) {

header('Location: /');

**exit**();

}

Исключения удобны тем, что можно бросать их в одном слое (например, модели) а ловить в другом (контроллере). Или в вашем случае, бросили в контроллере, а поймали во фронт-контроллере.

[babls2332@gmail.com](https://php.zone/user/6047) 19.04.2020 в 11:23

ArticlesController.php

$userController->isAdmin(**$this**->user);

User.php

**public** **function** **isAdmin**(User $user) {

**if** ($user->role !== 'admin') {

**throw** **new** Forbidden('Недостаточно прав');

} **else** {

**return** **true**;

}

}

articles/view.php

**<?** **if** ((!**empty**($user)) && ($user->isAdmin($user))) { **?>**

<a href="/articles/<?= $article->getId() ?>/edit">Редактировать</a>

**<?** } **?>**

[ivashkevich](https://php.zone/user/1) 19.04.2020 в 15:51

**public** **function** **isAdmin**(User $user) {

**if** ($user->role !== 'admin') {

**throw** **new** Forbidden('Недостаточно прав');

} **else** {

**return** **true**;

}

}

Это не исключительная ситуация, и исключение здесь бросать не к месту. Метод называется isAdmin, значит он должен вернуть булево значение true/false, в зависимости от того, является ли пользователь админом или нет. Если не является - возвращайте false, ничего исключительного здесь нет.

В то же время, для того чтобы бросать исключения, когда какое-то условие не выполняется, применяются методы с префиксом ensure...  
Например:

**public** **function** **ensureIsAdmin**(): **void**

{

**if** (!$user->isAdmin()) {

**throw** ...

}

}

Обратите внимание, что такие методы ничего не возвращают, лишь бросают исключение, когда что-то не так.

Ensure можно перевести как "убедись что..." и далее пишем, в чем нужно убедиться. Если не убедились - это уже исключительная ситуация.

[babls2332@gmail.com](https://php.zone/user/6047) 19.04.2020 в 16:57

В прошлом уроке ничего не сказали по-поводу этого метода, потому так и оставил

[Alexann](https://php.zone/user/13102) 22.04.2020 в 21:52

ArticlesController.php:

...

**public** **function** **edit**(int $articleId): **void**

{

$article = Article::getById($articleId);

**if** ($article === **null**) {

**throw** **new** NotFoundException('Не найдена такая статья');

}

**if** (**$this**->user === **null**) {

**throw** **new** UnauthorizedException();

}

**if** (!**$this**->user->isAdmin()) {

**throw** **new** ForbiddenException('Для редактирования статьи необходимы права администратора');

}

...

view.php:

**<?php** **include** **\_\_DIR\_\_** . '/../header.php';

**if** ($user!==**null** && $user->isAdmin()) :**?>**

<a href="/articles/<?= $article->getId() ?>/edit">Редактировать</a>

**<?** **endif**; **?>**

<h1>**<?**= $article->getName() **?>**</h1>

<p>**<?**= $article->getText() **?>**</p>

<p>Автор: **<?**= $article->getAuthor()->getNickname() **?>**</p>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

[ivashkevich](https://php.zone/user/1) 23.04.2020 в 10:36

Отлично!

# 32 Задание для самостоятельной работы – доводим блог до ума

Сегодня в рамках данной статьи вам будет дано задание самостоятельно доработать блог до конца. Все необходимые знания для данного урока у вас уже имеются, если вам непонятно как делать то или иное задание – рекомендую вам пройтись по списку всех уроков – скорее всего вы просто что-то подзабыли.

Итак, что же нам требуется сделать для того, чтобы наш блог стал похож на настоящий блог.

1. Комментарии.  
   Добавьте возможность добавлять комментарии к статьям. Для хранения комментариев заведите отдельную таблицу – comments. В ней будут храниться:
   * id комментария
   * автор комментария (id пользователя),
   * id статьи, к которой он написан,
   * текст комментария,
   * дата его публикации.

Для добавления комментария к статье стоит использовать url типа /articles/123/comments – сюда будет отправляться POST-запрос. Саму же форму добавления комментария можно сделать непосредственно на странице со статьей. Чтобы было проще – показывайте эту форму только авторизованным пользователям, а для неавторизованных просто пишите текст о том, что нужно зарегистрироваться для добавления комментария. Если POST-запрос выполнился успешно – отправляйте пользователя на страницу статьи с якорем для добавленного комментария. Типа: /articles/123#comment1.

Автор комментария и администратор могут редактировать комментарии по url типа такого: / comments/456/edit. Соответственно, ссылку «Редактировать» рядом с комментом стоит показывать только админам и авторам комментария.

1. Админка.  
   В админку могут попасть только администраторы сайта. Это стоит проверять в контроллере самой админки. Кроме того, в шаблоне стоит проверять является ли текущий пользователь админом. В админке для начала сделайте две страницы – список последних статей и список последних комментариев. Напротив каждой из этих сущностей – кнопку для редактирования. Статьи в админке не нужно показывать целиком – выводите только первые 100 символов текста – для этого в модели Article добавьте метод для получения короткой ссылки.
2. Включайте фантазию. Теперь добавьте что-нибудь от себя – добавьте возможность загружать аватары для пользователей и выводите их в комментариях, добавьте в админке возможность менять название сайта и т.д. – что вам больше понравится, то и делайте. Если фантазия подводит – можно взять блоговый движок, например wordpress, и попробовать реализовать какой-нибудь функционал оттуда.

На это задание я советую потратить вам не менее 24 часов чистого времени – растяните его на несколько дней, в зависимости от того, сколько времени вы можете посвящать этому ежедневно. Можно выделить под это дело выходные – погрузитесь в это задание с головой и получите удовольствие от того, что ваши мысли через ваши пальцы становятся реальностью. Только задумайтесь – еще несколько месяцев назад вы делали первые шаги в программировании, а сегодня – можете реализовывать свои собственные идеи.

Этот урок в этом курсе не последний, но, пожалуй, самый важный. Займитесь им основательно, а затем переходите к следующим. До встречи!

<https://www.dropbox.com/sh/fl1un4gugpsjpty/AADqAIUGg8k84Jj3Xulyd2CXa?dl=0>

# 33 Пишем свой фреймворк на PHP

В этом уроке мы с вами напишем свой мини-фреймворк на PHP. Фреймворк – это такой каркас приложения, на базе которого строится приложение. Он позволяет разрабатывать приложение быстрее за счет того, что содержит в себе реализацию основных компонентов: роутинг, контроллеры, слой для работы с базой данных, работу с шаблонами.

Так вот, дорогие мои ученики. Я вас поздравляю с тем, что вы уже написали свой собственный фреймворк для сайта, и в этом уроке ничего писать не нужно.

Только задумайтесь, какой проект вы уже написали самостоятельно! Для того, чтобы добавить новый функционал на блог, вам достаточно создать экшен в контроллере, прописать роутинг, добавить класс для новой модели и создать шаблончик – вся остальная обвязка уже имеется.

Большинство современных фреймворков включают в себя все то, что мы с вами написали самостоятельно. Конечно, там все несколько сложнее, чем в наших примерах. Но суть – та же. Это реализация архитектуры MVC с роутингом, просто везде сделанная по-своему. Теперь вы сможете использовать другие фреймворки и понимать, как все работает внутри, а не думая, что происходит какая-то магия.

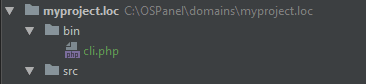
Так что, поздравляю вас с написанием своего фреймворка =)

И до встречи в следующем уроке!

# 1 Пишем свой фреймворк на PHP

До этого момента мы с Вами знали, что PHP работает на сервере. Клиент обращается к серверу по протоколу HTTP с каким-либо запросом, запрос на сервере обрабатывается и формируется ответ. После этого клиенту снова по протоколу HTTP в ответе отдаётся сформированный ответ. Однако, если взять какой-нибудь более-менее продвинутый сайт, то мы увидим, что есть задачи, которые не решаются стандартным клиент-серверным путем. Например: поздравлять пользователей с днём рождения и дарить им скидку на какой-нибудь продукт. Для того, чтобы это сделать, нам придется обновлять раз в день php-скрипт в браузере, чтобы он выбирал пользователей, у которых сегодня ДР, затем создавал для них скидки, и отправлял им сообщения по почте. Согласитесь, неудобно это делать вручную и в браузере. Для таких случаев в PHP предусмотрен Command Line Interface (CLI) – интерфейс командной строки.

CLI позволяет запускать программы на PHP не через привычную нам клиент-серверную архитектуру, а как простые программы в командной строке. Давайте создадим простейший скрипт, чтобы показать, как это работает. Создаём новую папку bin в корне проекта, а в ней файл – cli.php.

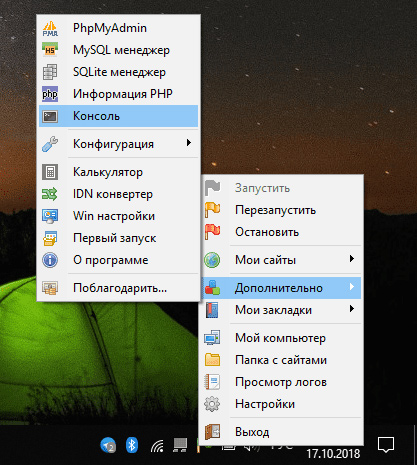
[](https://php.zone/uploads/img/7f4fd4d0b182257c957262721100ab332ef7e763adb6e059d9f03377e16a4ece.png)

Пишем простейший код:

bin/cli.php

**<?php**

**echo** 2 + 2;

А теперь запускаем консоль из OpenServer:  
[](https://php.zone/uploads/img/201909229056c7fcc6e14f2154efebe7699eb05af548ce5a6a2c9fe017d54542550bc7ae.jpg)

Переходим в папку с нашим проектом, выполнив:

cd domains\myproject.loc

И пишем следующую команду:

php bin/cli.php

В ответ получаем:  
[Вывод результата в терминал](https://php.zone/uploads/img/7717fd5c44b17ac46b6bc17738b62168cbe093ff8ed2b990ebc597b9f3b85e6b.png)

Написали простейшее консольное приложение! Уже неплохо. Но что если мы захотим сложить 2 числа, которые нужно передать скрипту? Как Вы понимаете, сделать это с помощью GET- или POST- запросов уже не получится. Так как же быть?

## **Аргументы консольного приложения**

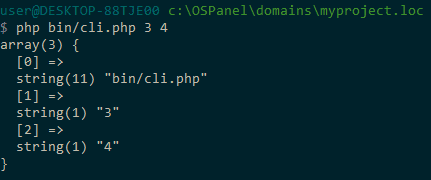
На помощь нам приходят аргументы, которые мы можем передать в скрипт, указав их после имени скрипта в командной строке. Вот так:  
[Аргументы консольного приложения](https://php.zone/uploads/img/bea01f2e9393785d6caaf7fd92e16234c85ebe0c70b413bf833422ae7d5ce99b.png)

А для того, чтобы получить к ним доступ из php-скрипта используется магическая переменная $argv. Она представляет собой массив, в котором нулевой элемент – это путь до скрипта, а все последующие – это его аргументы в консоли.

bin/cli.php

**<?php**

var\_dump($argv);

Давайте теперь запустим наш скрипт с параметрами:  
[](https://php.zone/uploads/img/b1d29f788d1bf496df15b747551443756cc79936da81d74719adffedd71892d5.png)

Как видим, наши аргументы попали в этот массив. Давайте напишем простейший скрипт, который будет складывать все переданные ему аргументы.

**<?php**

**unset**($argv[0]);

$sum = 0;

**foreach** ($argv **as** $item) {

$sum += $item;

}

**echo** $sum;

Запустим его, и убедимся, что все работает:  
[Сумма всех аргументов](https://php.zone/uploads/img/e4eaa9c7bb56bec0b3815dbcc188c7e10fc2fd55e924a0da44a9865577c25763.png)

И он действительно работает: 3 + 4 + 5 = 12.

А что если мы хотим передавать аргументы с именами? Вроде такого:  
[Именованные аргументы](https://php.zone/uploads/img/e768243cf65b189df50d547fb8b0ac964849b6971829d448d33276bcb5f4407b.png)

И затем в коде получать их в коде по их именам? Для этого нам следует написать простейший парсер, который будет находить вот такие именованные параметры и их значения. Пишем.

**<?php**

**unset**($argv[0]);

$params = [];

**foreach** ($argv **as** $argument) {

preg\_match('/^-(.+)=(.+)$/', $argument, $matches);

**if** (!**empty**($matches)) {

$paramName = $matches[1];

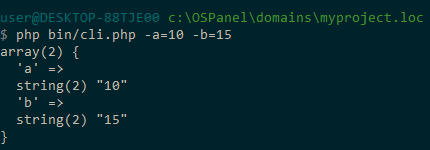
$paramValue = $matches[2];

$params[$paramName] = $paramValue;

}

}

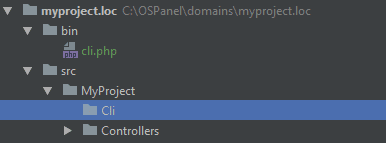
var\_dump($params);

И проверяем его работу:  
[](https://php.zone/uploads/img/3572a9cffcf4a4e73526ebfe1ecaff1dbc9677944d824ef450919ca92ebf2687.png)

Отлично, теперь мы можем обращаться к элементам массива params, чтобы выяснить, были ли нам переданы какие-то аргументы или нет.

## **CLI и ООП**

Мы с вами изучили некоторые основы работы с CLI. Давайте теперь перенесем эти знания на объектно-ориентированный подход и научимся работать через интерфейс командной строки с объектами.

Для этого нам понадобится создать отдельную директорию под «команды». Команды – так мы будем называть наши специальные классы, которые будут выполнять какой-то код через запуск из командной строки. Создаем новую директорию: src/MyProject/Cli.  
[](https://php.zone/uploads/img/098f2a5d320c42fb39d7b59f008b5dd7f21dbe40d69dc55978065bf7a1a8f4af.png)

И теперь создадим наш первый класс, который будет заниматься тем, что считает сумму переданных в него аргументов: -a и -b.

src/MyProject/Cli/Summator.php

**<?php**

**namespace** **MyProject**\**Cli**;

**use** **MyProject**\**Exceptions**\**CliException**;

**class** **Summator**

{

*/\*\* @var array \*/*

**private** $params;

**public** **function** **\_\_construct**(array $params)

{

**$this**->params = $params;

**$this**->checkParams();

}

**public** **function** **execute**()

{

**echo** **$this**->getParam('a') + **$this**->getParam('b');

}

**private** **function** **checkParams**()

{

**$this**->ensureParamExists('a');

**$this**->ensureParamExists('b');

}

**private** **function** **getParam**(string $paramName)

{

**return** **$this**->params[$paramName] ?? **null**;

}

**private** **function** **ensureParamExists**(string $paramName)

{

**if** (!**isset**(**$this**->params[$paramName])) {

**throw** **new** CliException('Param with name "' . $paramName . '" is not set!');

}

}

}

В конструкторе класса мы принимаем список параметров, сохраняем их, а затем вызываем метод checkParams(), который проверяет наличие обязательных параметров для этого скрипта. В нём просто поочередно вызывается метод для проверки в массиве нужных ключей. Если их нет – метод кинет исключение. И, наконец, есть метод execute(), который содержит бизнес-логику. В нем используется метод getParam(), который вернет параметр (при его наличии), либо вернет null (при его отсутствии).

И также создаём исключение, специально для ошибок, возникающих при работе с CLI.

src/MyProject/Exceptions/CliException.php

**<?php**

**namespace** **MyProject**\**Exceptions**;

**class** **CliException** **extends** \**Exception**

{

}

Теперь давайте снова вернемся в нашу точку входа для консольных приложений cli.php. Этот файл можно назвать фронт-контроллером для консольных команд, он как index.php в случае с клиент-серверным подходом будет создавать другие объекты и запускать весь процесс.

Дополним этот код так, чтобы он создавал экземпляр нужного класса и передавал ему аргументы.

bin/cli.php

**<?php**

**try** {

**unset**($argv[0]);

*// Регистрируем функцию автозагрузки*

spl\_autoload\_register(**function** (string $className) {

**require\_once** **\_\_DIR\_\_** . '/../src/' . $className . '.php';

});

*// Составляем полное имя класса, добавив нэймспейс*

$className = '\\MyProject\\Cli\\' . array\_shift($argv);

**if** (!class\_exists($className)) {

**throw** **new** \MyProject\Exceptions\CliException('Class "' . $className . '" not found');

}

*// Подготавливаем список аргументов*

$params = [];

**foreach** ($argv **as** $argument) {

preg\_match('/^-(.+)=(.+)$/', $argument, $matches);

**if** (!**empty**($matches)) {

$paramName = $matches[1];

$paramValue = $matches[2];

$params[$paramName] = $paramValue;

}

}

*// Создаём экземпляр класса, передав параметры и вызываем метод execute()*

$class = **new** $className($params);

$class->execute();

} **catch** (\MyProject\Exceptions\CliException $e) {

**echo** 'Error: ' . $e->getMessage();

}

Теперь мы можем запустить наш скрипт с помощью вот такой команды:  
[Сумматор](https://php.zone/uploads/img/986a3cf4b3143c69d84980c5ffd98533637a3d915920f078b687c2914c58426f.png)

Если мы захотим создать еще один класс, в котором мы будем вычитать из аргумента a аргумент b, то нам нужно будет продублировать довольно большой объем кода. Но ведь если присмотреться – большую часть кода из класса Summator можно вынести в отдельный класс и использовать его повторно.

Давайте создадим абстрактный класс, который будет заниматься тем, что будет сохранять переданные в него параметры и запускать метод для их проверки.

src/MyProject/Cli/AbstractCommand.php

**<?php**

**namespace** **MyProject**\**Cli**;

**use** **MyProject**\**Exceptions**\**CliException**;

**abstract** **class** **AbstractCommand**

{

*/\*\* @var array \*/*

**private** $params;

**public** **function** **\_\_construct**(array $params)

{

**$this**->params = $params;

**$this**->checkParams();

}

**abstract** **public** **function** **execute**();

**abstract** **protected** **function** **checkParams**();

**protected** **function** **getParam**(string $paramName)

{

**return** **$this**->params[$paramName] ?? **null**;

}

**protected** **function** **ensureParamExists**(string $paramName)

{

**if** (!**isset**(**$this**->params[$paramName])) {

**throw** **new** CliException('Param with name "' . $paramName . '" is not set!');

}

}

}

Теперь нам в классе Summator достаточно отнаследоваться от этого класса и он значительно упростится:

src/MyProject/Cli/Summator.php

**<?php**

**namespace** **MyProject**\**Cli**;

**class** **Summator** **extends** **AbstractCommand**

{

**protected** **function** **checkParams**()

{

**$this**->ensureParamExists('a');

**$this**->ensureParamExists('b');

}

**public** **function** **execute**()

{

**echo** **$this**->getParam('a') + **$this**->getParam('b');

}

}

Запустим скрипт снова и убедимся, что все успешно отработало:  
[Результат суммы в терминале](https://php.zone/uploads/img/420c7cd71ed80adb25d729619f5b092cf1ef5d1ad67e4443dde60beb4c27db45.png)

Давайте создадим по аналогии скрипт, который будет вычитать из аргумента x аргумент y.

src/MyProject/Cli/Minusator.php

**<?php**

**namespace** **MyProject**\**Cli**;

**class** **Minusator** **extends** **AbstractCommand**

{

**protected** **function** **checkParams**()

{

**$this**->ensureParamExists('x');

**$this**->ensureParamExists('y');

}

**public** **function** **execute**()

{

**echo** **$this**->getParam('x') - **$this**->getParam('y');

}

}

Проверим его в деле:  
[Разница в консоли](https://php.zone/uploads/img/88c2394ceb6553140ab3de36fdf0970e1705a1ed61563170e3e87613a36b815f.png)

А теперь давайте попробуем не указать один из аргументов – получим ошибку.  
[Ошибка об отсутствии обязательного аргумента](https://php.zone/uploads/img/9e2b8862822cd3c3f20abad6f5b7bc4e055d58a12191f5ebc4990d28f2f57110.png)

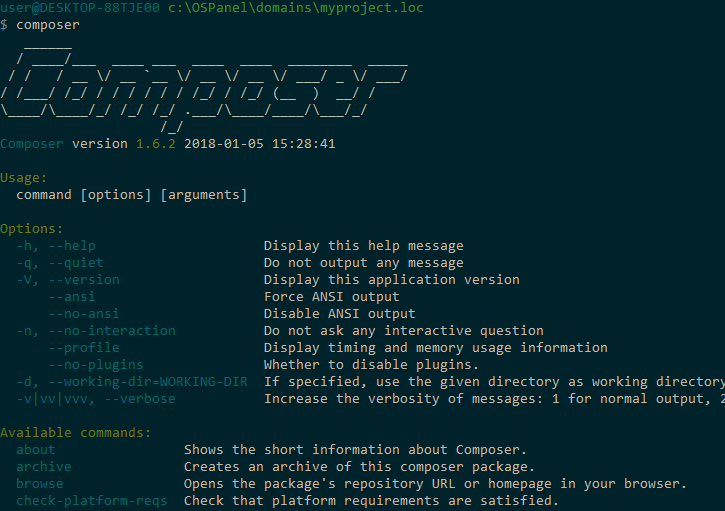
Вот таким вот нехитрым образом мы с вами научились создавать простейшие программы для запуска в консоли на PHP. А в следующем уроке мы с вами научимся запускать эти команды по расписанию.

# 2 Менеджер пакетов Composer

В этом уроке речь пойдёт не о написании кода на PHP, а о том, как использовать уже готовый. Для этого в любом языке программирования используются библиотеки – это уже написанный код, который делает какую-то часть работы, и который достаточно подключить в своём проекте чтобы начать использовать. Для многих современных языков программирования есть менеджеры пакетов, которые позволяют находить и устанавливать библиотеки в автоматическом режиме. В PHP для этого используется Composer – это менеджер пакетов, который сам скачивает нужные нам библиотеки и добавляет их в автозагрузку вашего приложения. После этого мы можем сразу использовать уже готовые классы и функции в наших программах. Удобно, да? Кроме того, репозиторий с открытыми бесплатными библиотеками Packagist.org насчитывает на сегодняшний день более 200 000 пакетов. Как вы понимаете, готовый код есть практически на все случаи жизни.

Сам по себе composer представляет собой консольное приложение, написанное на языке PHP. И для того чтобы его использовать, нужен только установленный PHP. Скачать composer можно с официального сайта - <https://getcomposer.org/download/>. Заходите и скачивайте версию, подходящую для вашей ОС. Если же Вы используете OpenServer, спешу вас обрадовать – composer у Вас уже установлен. Чтобы начать им пользоваться, нужно открыть уже знакомую вам по прошлым урокам консоль OpenServer. Для того, чтобы запустить composer, достаточно простейшей команды:

composer

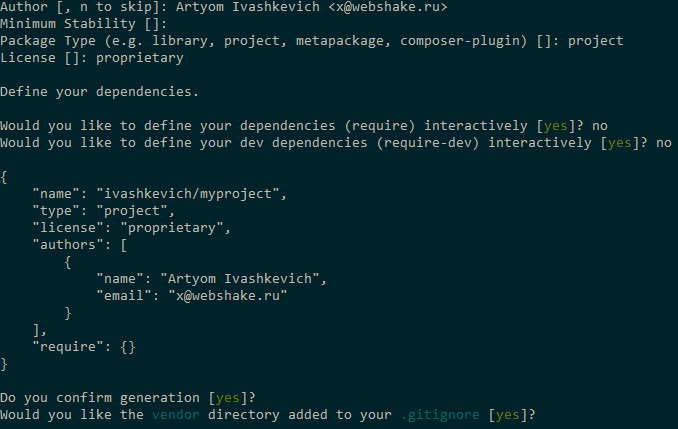
[](https://php.zone/uploads/img/20bc62e9d2e8f159682501772b7383ae3f4030e07fac1ca80382d78204df7142.png)

В ответ мы получаем список доступных команд. Первое, что нам нужно сделать – это выполнить в директории с нашим проектом (не забудьте в нее перейти с помощью команды cd) команду:

composer init

Эта команда задаст нам несколько вопросов:

* **Package name (<vendor>/<name>) [user/myproject.loc]** – в composer название пакета состоит из двух частей: vendor (имя производителя) и name (имя пакета). Я вам предлагаю использовать какой-нибудь свой никнейм или фамилию в качестве вендора, а в качестве имени пакета – myproject. В моём случае ответ для этого пункта будет такой: ivashkevich/myproject.
* **Description []:** - описание проекта, можно оставить пустым.
* **Author [, n to skip]:** - автор проекта. В формате John Smith <john@example.com>
* **Minimum Stability []:** - минимальная степень стабильности проекта. Об этом позже, пока оставим пустым.
* **Package Type (e.g. library, project, metapackage, composer-plugin) []:** - тип пакета. Пишем project.
* **License []:** - тип лицензии, по которой будет распространяться наш код. Пишем proprietary.
* **Would you like to define your dependencies (require) interactively [yes]?** – Желаете ли Вы определить зависимости для проекта в интерактивном режиме. То есть, хотим ли мы прямо сейчас загружать библиотеки. Отвечаем no.
* **Would you like to define your dev dependencies (require-dev) interactively [yes]?** – То же, что и выше, только для разработческого окружения – снова no.
* **Do you confirm generation [yes]** – Подтверждаем ли мы генерацию конфигурационного файла. Жмем yes.
* **Would you like the vendor directory added to your .gitignore [yes]?** – Нужно ли добавить папку vendor в исключения для git. Yes.

[](https://php.zone/uploads/img/80287cf2cd12d28422721cf4e681fbef9ad66a8c1be33bbec4827b0cb8a817f8.png)

После всего этого заходим в папку с проектом, и видим в корне проекта файлик composer.json.

{

"name": "ivashkevich/myproject",

"type": "project",

"license": "proprietary",

"authors": [

{

"name": "Artyom Ivashkevich",

"email": "x@php.zone"

}

],

"require": {}

}

Это файл, который будет использоваться composer для дальнейшей работы. Он описан в формате JSON – об этом формате я вам советую прочитать прямо сейчас в гугле, чтобы избежать лишних вопросов. Но в принципе – это простейший конфиг, в котором перечислены настройки по типу

ключ: значение

## **Поиск пакетов**

Теперь давайте зайдём в репозиторий пакетов Packagist - <https://packagist.org/>. И найдём какую-нибудь библиотеку, которую будем использовать в нашем проекте. Я предлагаю взять в качестве примера библиотеку для парсинга markdown-разметки. Если вы не знаете, что это такое – погуглите. Итак, идем на Packagist и вбиваем в поиск слово markdown - <https://packagist.org/?query=markdown>. И выбираем первый пакет, у которого больше всего скачиваний - <https://packagist.org/packages/erusev/parsedown>.

На страничке пакета мы можем посмотреть краткую (или не очень) документацию по проекту, посмотреть исходники на github и увидеть информацию об авторе.

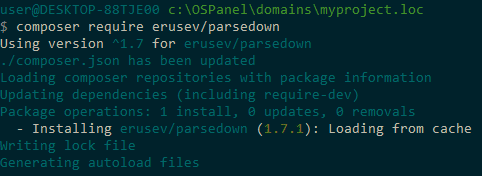
## **Установка пакета**

Для того, чтобы установить пакет, используется команда:

composer require вендор/имя\_пакета

Конкретно в нашем случае команда будет:

composer require erusev/parsedown

[](https://php.zone/uploads/img/516b2bd00a1f831c409730bf1a99bb7d100842757baf44003159683414b62743.png)

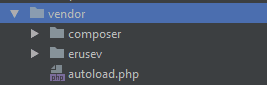
Первым делом идём в composer.json и видим новую строчку в разделе require:

"erusev/parsedown": "^1.7"

Это наш подключаемый пакет и его версия. Так как мы не указывали конкретную версию, которая нам нужна, была установлена последняя на данный момент. А значок «^» перед цифрами означает, что нам требуется версия пакета не ниже указанной.

## **Папка vendor**

В нашем проекте появилась папка vendor – здесь хранятся все пакеты, которые были скачаны composer-ом.

[](https://php.zone/uploads/img/b87cfb830862450ad8ffda9a1f0d886ef4ca6566318677f66db98e185f324d9f.png)

Кроме того, в ней есть файл autoload.php – его нужно подключить в нашем проекте через require, и после этого мы сможем использовать все файлы из библиотек в нашем проекте! Давайте сделаем это. Подключим этот файл в двух наших фронт-контроллерах: index.php и cli.php.

www/index.php

**<?php**

**require** **\_\_DIR\_\_** . '/../vendor/autoload.php';

...

bin/cli.php

**<?php**

**require** **\_\_DIR\_\_** . '/../vendor/autoload.php';

...

Всё! Теперь можно использовать библиотеки!

## **Использование библиотек**

Давайте для постов нашего блога начнем использовать Markdown-разметку. Сделать это с помощью подключенной нами библиотеки проще простого. Смотрим документацию на [страничке пакета](https://packagist.org/packages/erusev/parsedown" \t "_blank) и просто повторяем для нашей ситуации.

Добавляем в модели статьи метод, который будет прогонять текст статьи через парсер, прежде чем его вернуть:

src/MyProject/Models/Articles/Article.php

**public** **function** **getParsedText**(): **string**

{

$parser = **new** \Parsedown();

**return** $parser->text(**$this**->getText());

}

И добавляем вывод прошедшего парсинг текста в шаблоне:

templates/articles/view.php

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

<h1>**<?**= $article->getName() **?>**</h1>

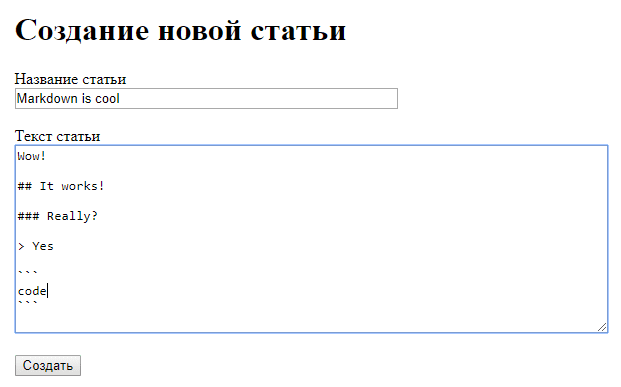
<p>**<?**= $article->getParsedText() **?>**</p>

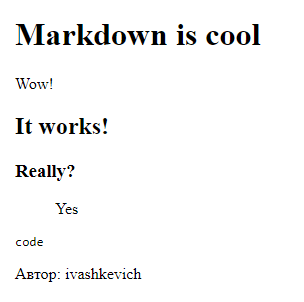
<p>Автор: **<?**= $article->getAuthor()->getNickname() **?>**</p>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Теперь при выводе статей содержимое поля text будет предварительно пропущено через markdown-парсер.

Пробуем создать новую статью, используя markdown-разметку.

[](https://php.zone/uploads/img/1de2104a1a78c1f02c34aad23e388e93b721b55dd4f8f72a48872c4f439f24fe.png)

Получаем на выходе отформатированный [HTML-тегами](https://php.zone/kurs-html-dlya-nachinayushih/tegi-v-html) текст.  
[](https://php.zone/uploads/img/6dc2b47d38152e8a917e0f042397301134b61108308290ca6d3d2558981eb49d.png)

Делаем то же самое для списка статей и вуаля – у нас блог, редактируемый с помощью markdown-разметки.

## **Автозагрузка для файлов проекта**

Кроме того, что композер создает файл с автозагрузкой для библиотек, он позволяет включить в эту автозагрузку еще и файлы нашего проекта. Для этого нужно в composer.json:

1. указать стандарт, который используется для автозагрузки. В нашем случае – PSR-4.
2. указать корневую папку и неймспейс для проекта.

composer.json

{

"name": "ivashkevich/myproject",

"type": "project",

"license": "proprietary",

"authors": [

{

"name": "Artyom Ivashkevich",

"email": "x@php.zone"

}

],

"autoload": {

"psr-4": {

"MyProject\\": "src/MyProject/"

}

}, "require": {

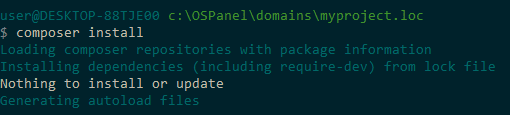
"erusev/parsedown": "^1.7"

}

}

После этого следует выполнить команду:

composer install

[](https://php.zone/uploads/img/f73a808dfa4d50cfc4bd9284e7bfd6a4cc693832314c6d0e78758c5a5f1b1756.png)

Эта команда устанавливает все заданные нами ранее зависимости (если они еще не установлены) и перегенерирует файл автозагрузки. После этого нужно в наших фронт-контроллерах убрать функции автозагрузки, которые мы ранее писали сами:

spl\_autoload\_register(**function** (string $className) {

**require\_once** **\_\_DIR\_\_** . '/../src/' . $className . '.php';

});

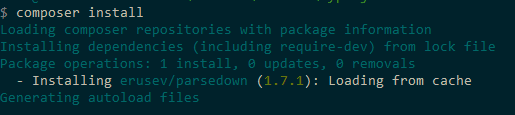
Убираем этот код из index.php и cli.php, после чего проверяем работоспособность сайта. Он по-прежнему работает, но теперь за автозагрузку функций отвечает код, сгенерированный композером.

## **composer.lock**

Кроме того, в корне проекта появился еще один файл – composer.lock. Это очень важный файл – в нем содержится конкретная версия пакета, которая использовалась в момент выполнения команды composer require.

В дальнейшем, если мы будем выкладывать проект куда-либо, мы не будем выкладывать папку vendor – ведь достаточно выполнить composer install и она снова появится. Нет смысла гонять туда-сюда большую папку с библиотеками. Можем проверить прямо сейчас – удалите папку vendor и выполните команду:

composer install

[](https://php.zone/uploads/img/1cde14c6c3f938f683b6d7d23170a20bbd3c5ef8e2d0351af0cfe56448525d5c.png)

Библиотеки снова скачались. Так вот сейчас была установлена ровно та версия библиотеки, которая задана в composer.lock. Если библиотека обновится и авторы что-то в ней изменят (сломают), то нам это не страшно – мы будем использовать ровно ту версию, которую использовали при разработке. В дальнейшем вы столкнетесь с необходимостью в этом, когда будете выкладывать ваши рабочие проекты на боевые сервера – очень важно чтобы на боевом сервере, который приносит бизнесу деньги, вдруг не обновилась библиотека и всё не рухнуло.

Если же вы хотите обновить какую-то библиотеку – нужно просто выполнить команду:

composer update имя\_библиотеки

Эта команда проверит наличие новой версии пакета, и если такой есть, то скачает и установит его. При этом также обновится и composer.lock.

Этот файл должен всегда сопровождаться вместе с вашим приложением. Он – гарантия того, что приложение на сервере будет работать с теми же пакетами, которые вы использовали, когда писали код. Он должен быть и в репозитории git вместе с проектом, и на сервере, и у разработчика. Проект и composer.lock должны всегда находиться рядом.

В заключение скажу - изучайте официальную документацию Composer-а и привыкайте использовать его - в реальных проектах он используется повсеместно. И будьте уверены, на вашей будущей работе composer тоже будет.

3

# 9 Делаем пагинацию на PHP

[#пагинация на php](https://php.zone/tag/%D0%BF%D0%B0%D0%B3%D0%B8%D0%BD%D0%B0%D1%86%D0%B8%D1%8F_%D0%BD%D0%B0_php)

[@ivashkevich](https://php.zone/user/1)

07.02.2021 в 19:02

 23180

+12

[9](https://php.zone/php-dlya-professionalov/pagination-php#comments)

Зачастую в тестовых заданиях на джунов присутствует пункт о реализации пагинации в какой-нибудь системе. И мне довольно часто прилетают просьбы показать как это делается, так как зачастую это задание ставит в тупик начинающих разработчиков. Что-ж, давайте разбираться.

Делать пагинацию я хочу предложить на базе написанного нами в курсе ООП в PHP движка для блога. Готовый проект можно взять [по ссылке](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs" \t "_blank). Накатите дамп базы и настройте веб-сервер так, чтобы корневой директорией для домена myproject.loc была папка www.

Теперь давайте добавим побольше статей в базу, чтобы было по чему пагинацию делать.

**INSERT** **INTO** `articles` (`author\_id`, `name`, `text`) **VALUES**

(1, 'Название статьи 1', 'Текст статьи 1'),

(1, 'Название статьи 2', 'Текст статьи 2'),

(1, 'Название статьи 3', 'Текст статьи 3'),

(1, 'Название статьи 4', 'Текст статьи 4'),

(1, 'Название статьи 5', 'Текст статьи 5'),

(1, 'Название статьи 6', 'Текст статьи 6'),

(1, 'Название статьи 7', 'Текст статьи 7'),

(1, 'Название статьи 8', 'Текст статьи 8'),

(1, 'Название статьи 9', 'Текст статьи 9'),

(1, 'Название статьи 10', 'Текст статьи 10'),

(1, 'Название статьи 11', 'Текст статьи 11'),

(1, 'Название статьи 12', 'Текст статьи 12'),

(1, 'Название статьи 13', 'Текст статьи 13'),

(1, 'Название статьи 14', 'Текст статьи 14'),

(1, 'Название статьи 15', 'Текст статьи 15'),

(1, 'Название статьи 16', 'Текст статьи 16'),

(1, 'Название статьи 17', 'Текст статьи 17'),

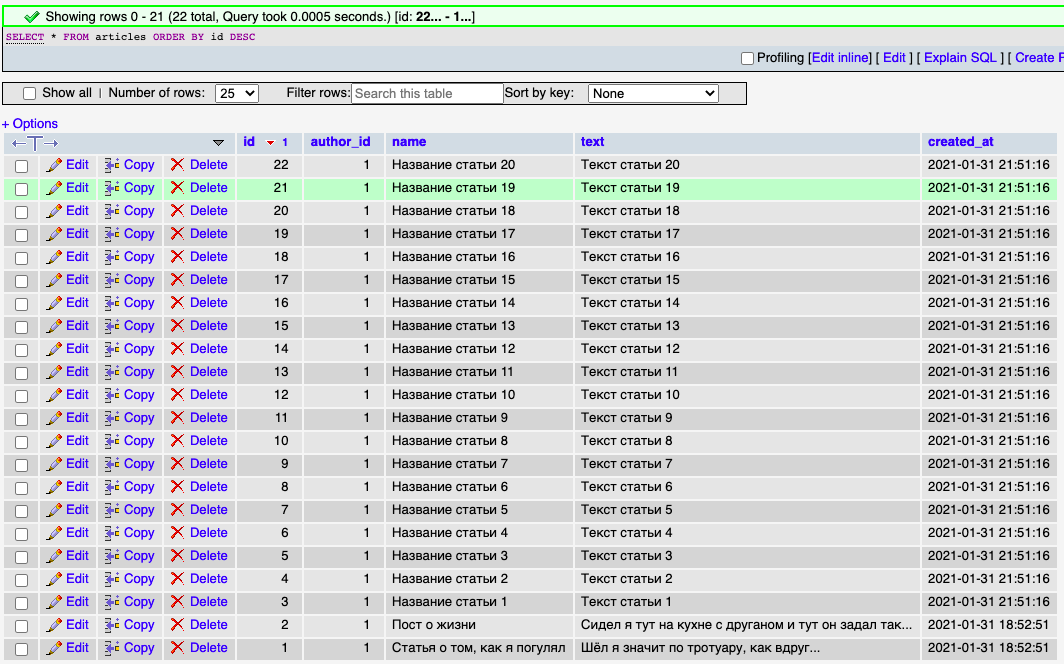
(1, 'Название статьи 18', 'Текст статьи 18'),

(1, 'Название статьи 19', 'Текст статьи 19'),

(1, 'Название статьи 20', 'Текст статьи 20');

Посмотрим на имеющиеся у нас статьи. Отсортируем их в обратном порядке.

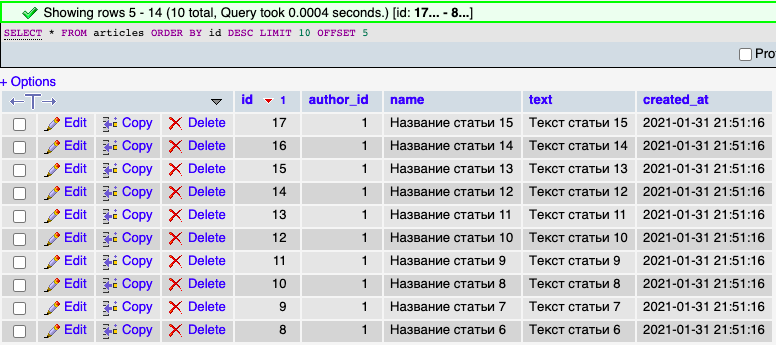
**SELECT** \* **FROM** articles **ORDER** **BY** **id** **DESC**;

[](https://php.zone/uploads/img/202102077b13456f2c2ebe0c35b6850c15534e80716bca4c593f69add4a7cbb94dab9e85.png)

Теперь, если вы еще не в курсе, нужно познакомиться с двумя конструкциями языка SQL: LIMIT и OFFSET. Они позволяют получить только часть строк из тех, что были получены запросом. LIMIT задаёт лимит записей, OFFSET задает количество строк, которые нужно пропустить.

Например, мы хотим пропустить первые 5 строк, и вывести следующие 10 строк:

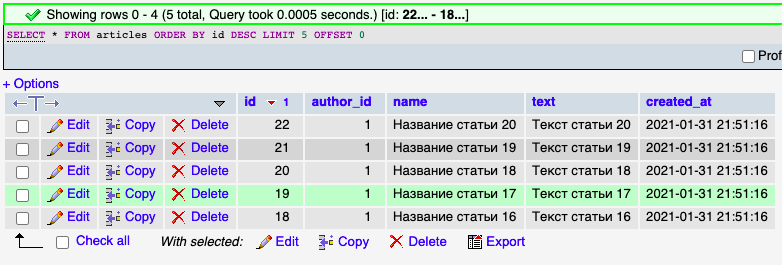
**SELECT** \* **FROM** articles **ORDER** **BY** **id** **DESC** **LIMIT** 10 **OFFSET** 5;

[](https://php.zone/uploads/img/20210207af16ef43e82138ca3e804c41cac9148caf5dcf5754155e0be1a967f2d4dacd7f.png)

Как видим, у нас пропустились первые 5 строк с id: 22,21,20,19,18, которые присутствовали в результате запроса без LIMIT и OFFSET. Также мы видим, что строк у нас здесь 10.

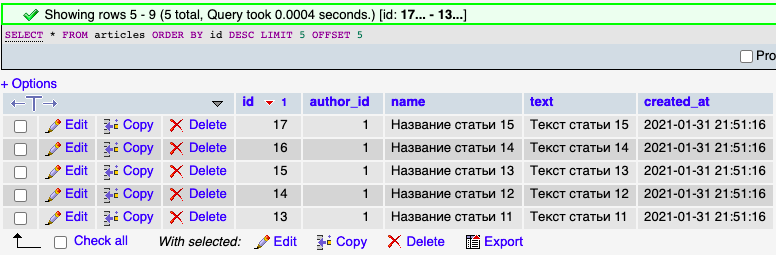
Как нетрудно догадаться, для реализации пагинации нам можно воспользоваться этими конструкциями. Если мы хотим выводить на каждой странице нашего блога по 5 записей, то для получения записей на первой странице стоит использовать запрос:

**SELECT** \* **FROM** articles **ORDER** **BY** **id** **DESC** **LIMIT** 5 **OFFSET** 0;

[](https://php.zone/uploads/img/20210207a1c8d2c372050989766e22727b39e0df6d836620528090d06b1f246a2b433943.png)

Для второй страницы запрос будет следующим:

**SELECT** \* **FROM** articles **ORDER** **BY** **id** **DESC** **LIMIT** 5 **OFFSET** 5;

[](https://php.zone/uploads/img/20210207babbde9d5338f1ff9acb4b13e1309db1ca6d36617513a7c3b4b7547c127205b6.png)

То есть здесь мы пропустили первые 5 записей и вывели следующие 5.

Аналогично строим запрос для третьей страницы:

**SELECT** \* **FROM** articles **ORDER** **BY** **id** **DESC** **LIMIT** 5 **OFFSET** 10;

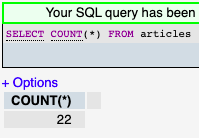
И так далее, пока записи не закончатся.

Получаем формулу для получения запроса, которые выводит записи на n-ой странице блога, где k-число записей на одной странице:

**SELECT** \* **FROM** articles **ORDER** **BY** **id** **DESC** **LIMIT** k **OFFSET** (n-1)\*k;

Чтобы понять, сколько у нас будет страниц, нам нужно число записей разделить на число записей, выводимых на одной странице. Полученное число округляем в большую сторону. К примеру, если у нас 22 записи, и на одной странице выводим 5, то число страниц будет 22/5 = 4,4. Округляем в большую сторону и получаем 5. На первых четырех страницах у нас будет по 5 записей, а на последней будет 2 записи.  
Получить общее число записей можно с помощью запроса:

**SELECT** **COUNT**(\*) **FROM** articles;

[](https://php.zone/uploads/img/20210207397e7d25043fd5dcc18c47fd2e41b3592648f5678c209aa2277162911019224d.png)

Осталось всё это дело закодить. Начнём с самого простого – выведем ссылки на странички со статьями.  
В классе ActiveRecordEntity добавляем метод для получения количества страниц. Метод будет принимать на вход количество записей на одной странице.

src/MyProject/Models/ActiveRecordEntity.php

**public** **static** **function** **getPagesCount**(int $itemsPerPage): **int**

{

$db = Db::getInstance();

$result = $db->query('SELECT COUNT(\*) AS cnt FROM ' . **static**::getTableName() . ';');

**return** ceil($result[0]->cnt / $itemsPerPage);

}

В MainController вызовем этот метод и передадим в шаблон число страниц.

src/MyProject/Controllers/MainController.php

**public** **function** **main**()

{

$articles = Article::findAll();

**$this**->view->renderHtml('main/main.php', [

'articles' => $articles,

'pagesCount' => Article::getPagesCount(5),

]);

}

Добавим в шаблоне внизу странички номера страниц.

templates/main/main.php

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

**<?php** **foreach** ($articles **as** $article): **?>**

<h2><a href="/articles/<?= $article->getId() ?>">**<?**= $article->getName() **?>**</a></h2>

<p>**<?**= $article->getText() **?>**</p>

<hr>

**<?php** **endforeach**; **?>**

<div style="text-align: center">

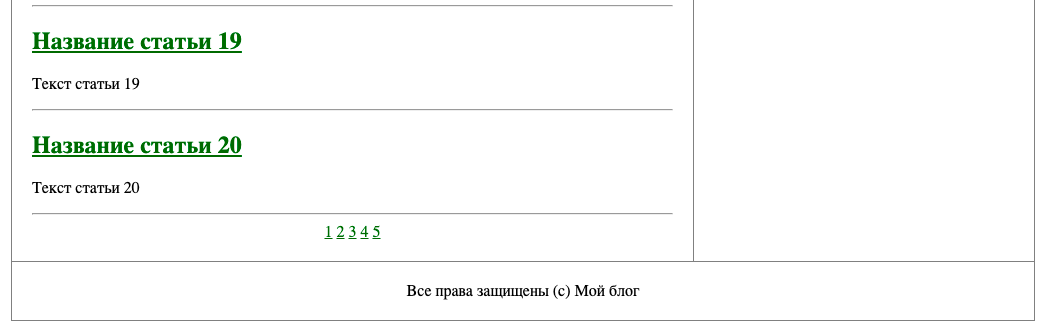
**<?php** **for** ($pageNum = 1; $pageNum <= $pagesCount; $pageNum++): **?>**

<a href="/<?= $pageNum === 1 ? '' : $pageNum ?>">**<?**= $pageNum **?>**</a>

**<?php** **endfor**; **?>**

</div>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Обратите внимание: при формировании ссылки на страницу мы проверяем, является ли страничка первой, и если это так, то формируем ссылку на главную страницу, иначе формируем ссылку вида: /n, где n – номер страницы.  
Посмотрим на результат:  
[](https://php.zone/uploads/img/20210207c117daf057c814ad16c891af39b57b1ef17ec5eaf6e47983a461f3d9564897e3.png)

Теперь давайте напишем метод для получения записей на n-ой страничке. Для этого в классе ActiveRecordEntity добавим метод getPage(), принимающий на вход количество записей на одной странице и номер страницы.

src/MyProject/Models/ActiveRecordEntity.php

*/\*\**

*\* @return static[]*

*\*/*

**public** **static** **function** **getPage**(int $pageNum, int $itemsPerPage): **array**

{

$db = Db::getInstance();

**return** $db->query(

sprintf(

'SELECT \* FROM `%s` ORDER BY id DESC LIMIT %d OFFSET %d;',

**static**::getTableName(),

$itemsPerPage,

($pageNum - 1) \* $itemsPerPage

),

[],

**static**::class

);

}

Добавим в MainController экшен для странички.

src/MyProject/Controllers/MainController.php

**public** **function** **page**(int $pageNum)

{

**$this**->view->renderHtml('main/main.php', [

'articles' => Article::getPage($pageNum, 5),

'pagesCount' => Article::getPagesCount(5),

]);

}

Переписываем экшен main.

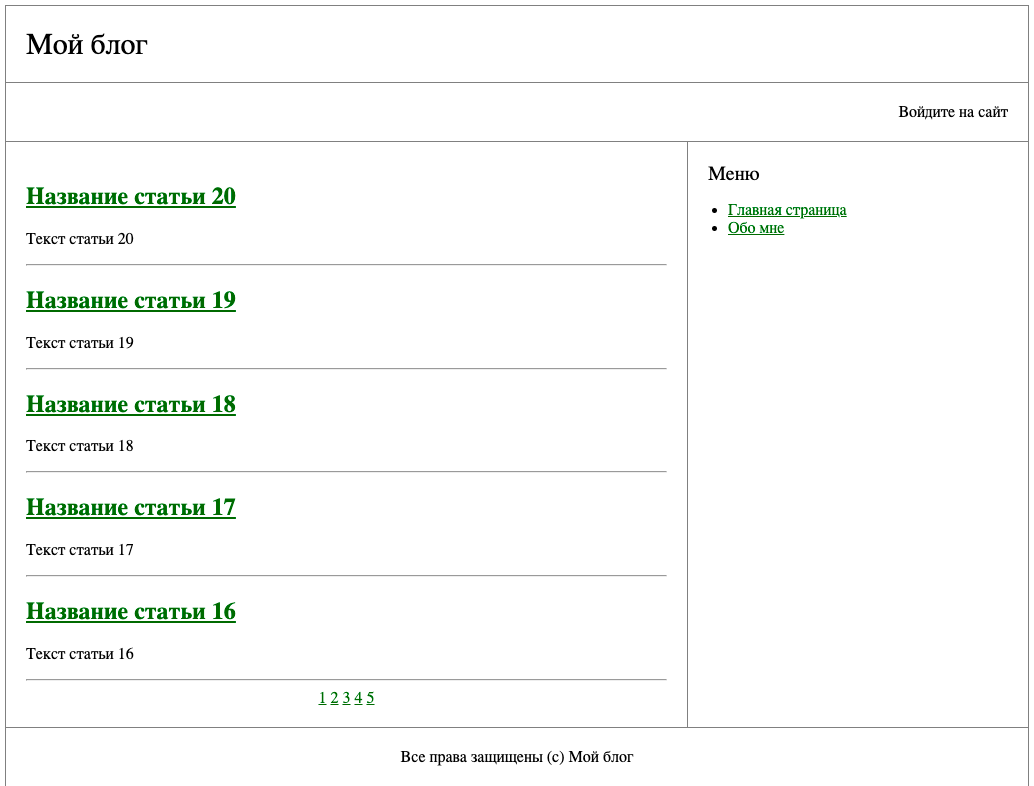
src/MyProject/Controllers/MainController.php

**public** **function** **main**()

{

**$this**->page(1);

}

Проверяем главную страницу:  
[](https://php.zone/uploads/img/2021020727d9ee8b85609ce594338542fc2593fea49cce0420aba7be1afaa9a03a346d2f.png)

Добавляем роут для страничек с номерами.

src/routes.php

'~^(\d+)$~' => [\MyProject\Controllers\MainController::class, 'page'],

Всё. Пагинация готова. Можно переходить по страничкам. Так выглядит пятая страница.

Из улучшений можно сделать текущий номер страницы не ссылкой, а просто текстом. Для этого передадим номер текущей странички в шаблон.

src/MyProject/Controllers/MainController.php

**public** **function** **page**(int $pageNum)

{

**$this**->view->renderHtml('main/main.php', [

'articles' => Article::getPage($pageNum, 5),

'pagesCount' => Article::getPagesCount(5),

'currentPageNum' => $pageNum,

]);

}

В шаблоне будем сравнивать текущий номер страницы с тем, который в текущей итерации.

templates/main/main.php

**<?php** **include** **\_\_DIR\_\_** . '/../header.php'; **?>**

**<?php** **foreach** ($articles **as** $article): **?>**

<h2><a href="/articles/<?= $article->getId() ?>">**<?**= $article->getName() **?>**</a></h2>

<p>**<?**= $article->getText() **?>**</p>

<hr>

**<?php** **endforeach**; **?>**

<div style="text-align: center">

**<?php** **for** ($pageNum = 1; $pageNum <= $pagesCount; $pageNum++): **?>**

**<?php** **if** ($currentPageNum === $pageNum): **?>**

<b>**<?**= $pageNum **?>**</b>

**<?php** **else**: **?>**

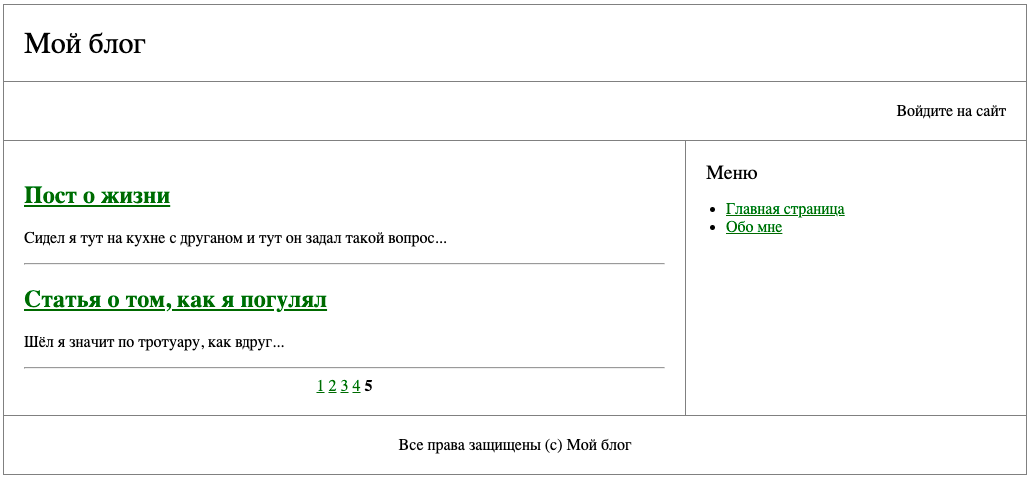
<a href="/<?= $pageNum === 1 ? '' : $pageNum ?>">**<?**= $pageNum **?>**</a>

**<?php** **endif**; **?>**

**<?php** **endfor**; **?>**

</div>

**<?php** **include** **\_\_DIR\_\_** . '/../footer.php'; **?>**

Результат:  
[](https://php.zone/uploads/img/202102072a0dc6e08a69bf8ec27cc9cd46ef2c491fe6ec26bffce7e156a2fc422400745d.png)

Вот такая простая пагинация у нас получилась. Она подойдет для большинства проектов без высокой нагрузки и с небольшим количеством записей. А для крупных проектов потребуется более специфичное решение, которое я рассмотрю в одном из ближайших уроков.

Изменения, сделанные в ходе урока, вы можете посмотреть [на гитхабе](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/pull/1/commits/1b8c3466b898c61c55887a0e48b53742da9adda0" \t "_blank).

# 10 Пагинация на PHP для большого количества записей

[#php](https://php.zone/tag/php) [#highload](https://php.zone/tag/highload) [#sql](https://php.zone/tag/sql) [#optimization](https://php.zone/tag/optimization) [#indexes](https://php.zone/tag/indexes)

[@ivashkevich](https://php.zone/user/1)

10.06.2021 в 15:49

 13643

+7

[4](https://php.zone/php-dlya-professionalov/paginaciya-na-php-dlya-bolshogo-kolichestva-zapisey#comments)

Всем привет. В [прошлом уроке](https://php.zone/php-dlya-professionalov/pagination-php" \t "_blank) мы рассмотрели пример простой пагинации, которая отлично работает при относительно небольшом количестве записей. Если же их становится много, мы столкнемся с проблемами.

Чтобы не делать всё с нуля я предлагаю доработать пагинацию, реализованную в прошлом уроке. Стяните проект с гитхаба и сделайте чекаут на [вот этот коммит](https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/tree/1b8c3466b898c61c55887a0e48b53742da9adda0" \t "_blank).

Чтобы обозначить проблемы, давайте напишем скрипт, который добавит в нашу базу 1 млн статей.

create\_many\_articles.php

**<?php**

spl\_autoload\_register(**function** (string $className) {

**require\_once** **\_\_DIR\_\_** . '/src/' . str\_replace('\\', '/', $className) . '.php';

});

$db = \MyProject\Services\Db::getInstance();

$query = '';

**for** ($i = 1; $i <= 1000000; $i++) {

$query .= sprintf(

'INSERT INTO articles (`author\_id`, `name`, `text`) VALUES (1, \'%s\', \'%s\');',

'Статья #' . $i,

'Текст статьи ' . $i

);

**if** ($i % 1000 === 0) {

$db->query($query);

$query = '';

**echo** $i . PHP\_EOL;

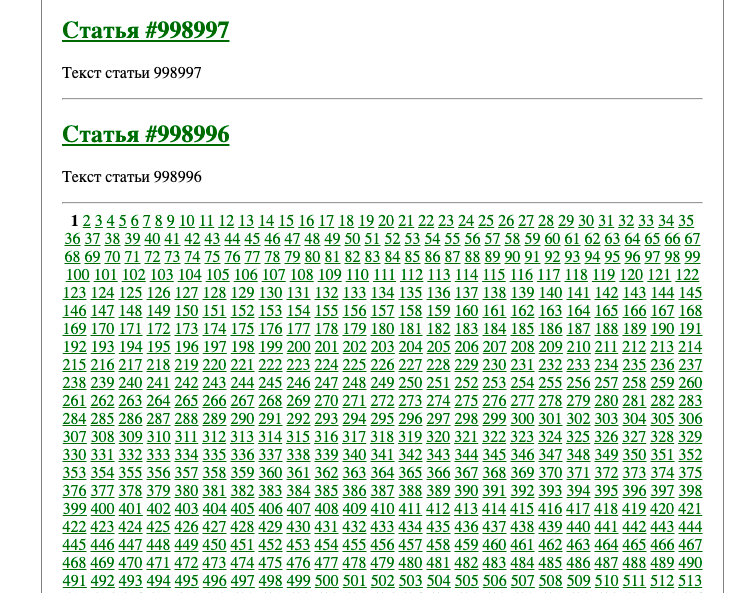
}

}

Запускаем его командой:

**php** **create\_many\_articles**.php

И идём пить кофе. После завершения работы скрипта заходим на главную страницу нашего блога <http://myproject.loc/>

Как видим, у нас появилось довольно большое число страниц, которое просто разрывает наш шаблончик. Не знаю как у вас, а у меня вкладка в браузере просто-напросто зависла.  
[](https://php.zone/uploads/img/2021061080f100b71482169198872d0092cb3a64feca1b6068a1c4d36e625d85f2c9752a.png)

Давайте сделаем пагинацию в стиле туда-сюда. Для этого в экшене \MyProject\Controllers\MainController::page() передадим в шаблон 2 ссылки на предыдущую и следующую страницы.

**public** **function** **page**(int $pageNum)

{

$pagesCount = Article::getPagesCount(5);

**$this**->view->renderHtml('main/main.php', [

'articles' => Article::getPage($pageNum, 5),

'previousPageLink' => $pageNum > 1

? '/' . ($pageNum - 1)

: **null**,

'nextPageLink' => $pageNum < $pagesCount

? '/' . ($pageNum + 1)

: **null**

]);

}

А в шаблоне выведем эти ссылки.

myproject.loc/templates/main/main.php

<div style="text-align: center">

**<?php** **if** ($previousPageLink !== **null**): **?>**

<a href="<?= $previousPageLink ?>">&lt; Туда</a>

**<?php** **else**: **?>**

<span style="color: grey">&lt; Туда</span>

**<?php** **endif**; **?>**

&nbsp;&nbsp;&nbsp;

**<?php** **if** ($nextPageLink !== **null**): **?>**

<a href="<?= $nextPageLink ?>">Сюда &gt;</a>

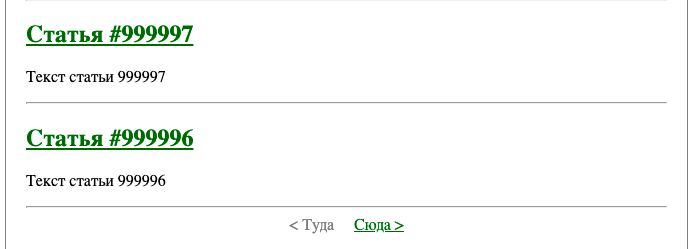
**<?php** **else**: **?>**

<span style="color: grey">Сюда &gt;</span>

**<?php** **endif**; **?>**

</div>

Любуемся аккуратным пагинатором.

[](https://php.zone/uploads/img/20210610411995de5d399714b6da29ed1951410eb020a980fd98e3e5a63afa71bb34a4ca.png)

Давайте теперь посмотрим на время генерации страницы. Для этого обернем код в index.php в следующие строки:

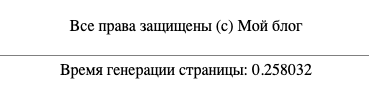
**<?php**

$startTime = microtime(**true**);

*// тут весь остальной код*

printf('<div style="text-align: center; padding: 5px">Время генерации страницы: %f</div>', $endTime - $startTime );

Обновим первую страницу блога. У меня время генерации заняло 0.258032 секунды.

[](https://php.zone/uploads/img/20210610fb3a7c95685079cef7a8b9c7c1516b55d95df3aa370324582bdc180f56d7e726.png)

А теперь давайте откроем последнюю страницу блога - <http://myproject.loc/200000>  
Время генерации страницы: 1.045724  
Вот это да! Разница почти в 4 раза!

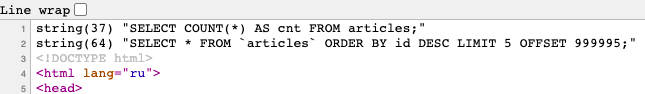
Давайте посмотрим, какие запросы у нас выполняются во время загрузки страницы. Для этого в коде \MyProject\Services\Db::query добавим var\_dump:

**public** **function** **query**(string $sql, array $params = [], string $className = 'stdClass'): ?**array**

{

var\_dump($sql);

Обновим страничку и увидим, что у нас выполнилось 2 запроса:

[](https://php.zone/uploads/img/202106105652e4766eac3e81865d50c7fd41ed98c5e30d75fe1afc0600ae2e64fbd69316.png)

Так как при загрузке первой страницы запрос:

**SELECT** **COUNT**(\*) **AS** cnt **FROM** articles;

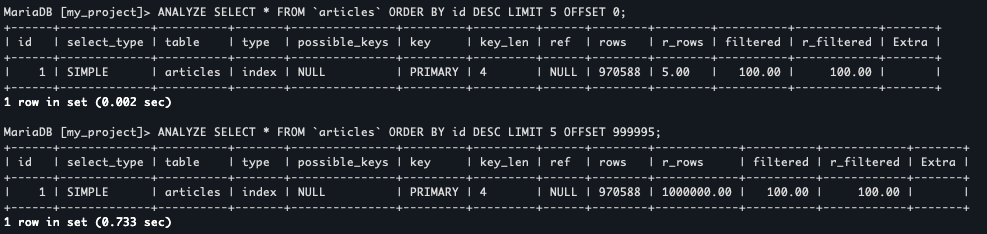
остаётся неизменным, мы понимаем, что в случае последней страницы работу существенно замедляет второй запрос:

**SELECT** \* **FROM** `articles` **ORDER** **BY** **id** **DESC** **LIMIT** 5 **OFFSET** 999995;

В случае загрузки первой страницы он будет выглядеть следующим образом:

**SELECT** \* **FROM** `articles` **ORDER** **BY** **id** **DESC** **LIMIT** 5 **OFFSET** 0;

Давайте сделаем ANALYZE (я использую mariadb, для mysql будет EXPLAIN ANALYZE) для обоих запросов.

[](https://php.zone/uploads/img/20210610a14293beb77dd42d255e32a9356b05d6e10a921d6e3e01fe3806fd0ab1bcb329.png)

Как видим, в случае первой страницы базке пришлось пройти 5 строк (r\_rows), а в случае последней страницы 1000000 строк! Дело в том, что OFFSET N работает таким образом, что он вычитывает строки, подходящие под условие запроса до тех пор, пока не дойдёт до N-ой строки. Всё что до N будет пропущено, а далее будут выгружены строки после N-ой, до необходимого значения LIMIT. То есть, к примеру, в нашем случае при получении 200й страницы блога база будет прогонять при запросе следующие строки:

Id -> 1000000

Id -> 999999

…*// Тут ещё очень много строк*

Id -> 6

Id -> 5 *// Все строки, что выше, пропускаем. Их там 999995. Начиная с этой выгребаем ещё X строк, где X – значение, указанное в LIMIT (в нашем случае = 5)*

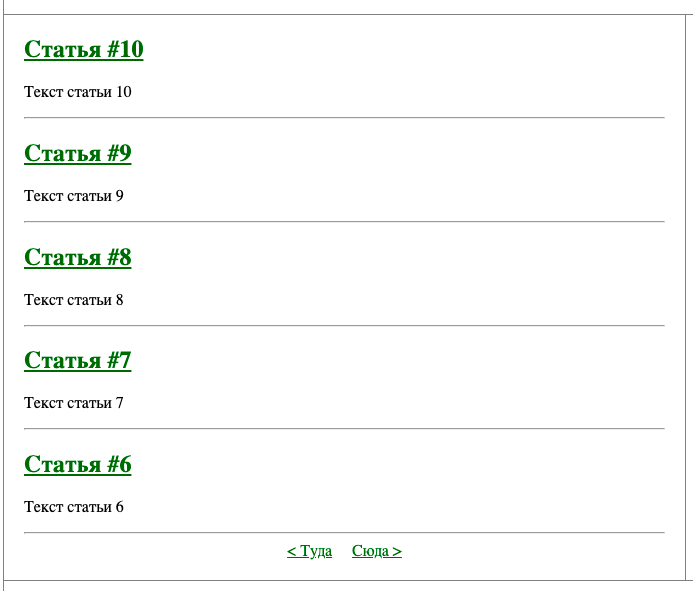
Id -> 4

Id -> 3

Id -> 2

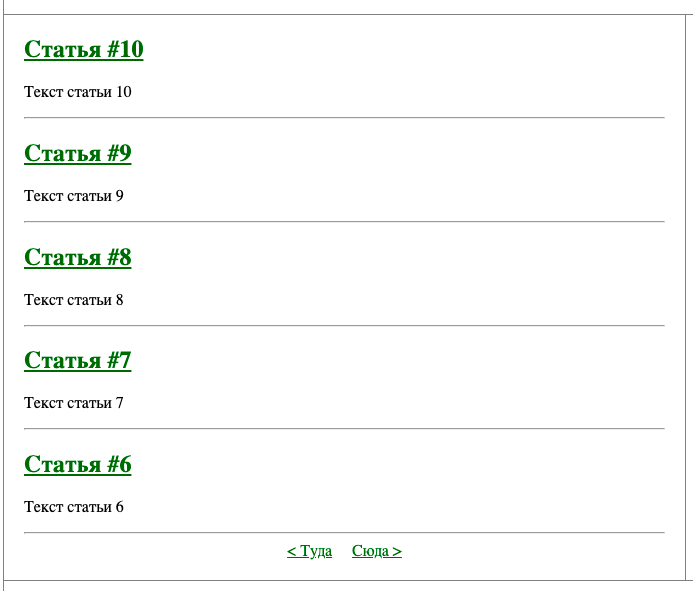
Id -> 1 *// Выгрузили 5 строк, на этом всё*

Как видим, базе пришлось сделать много лишней работы. Так уж работает OFFSET. Какие есть альтернативы? Давайте для начала посмотрим на предпоследнюю страницу блога.  
<http://myproject.loc/199999>

[](https://php.zone/uploads/img/202106101150e4a20c82a7e30416999d01a0b38e92dab91989e781c0839cdfd9347b88c6.png)

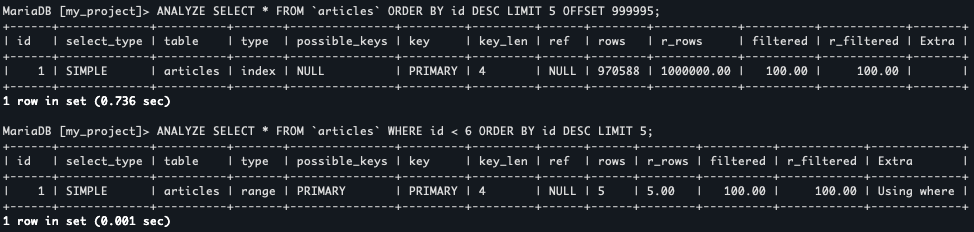
Как мы можем получить следующие 5 записей, чтобы сформировать следующую страницу?  
Зная, что текущая страница заканчивается на id = 6, нам можно получить следующие 5 записей, сформировав следующий запрос:

**SELECT** \* **FROM** `articles` **WHERE** **id** < 6 **ORDER** **BY** **id** **DESC** **LIMIT** 5;

[](https://php.zone/uploads/img/202106101150e4a20c82a7e30416999d01a0b38e92dab91989e781c0839cdfd9347b88c6.png)

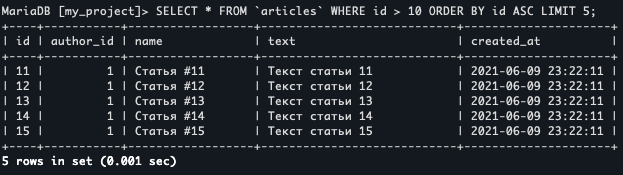
Как видим, результаты выполнения запросов совпадают. Однако, время их выполнения существенно отличается. Давайте проанализируем запрос.

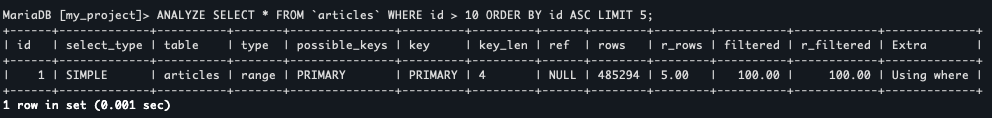
**ANALYZE** **SELECT** \* **FROM** `articles` **WHERE** **id** < 6 **ORDER** **BY** **id** **DESC** **LIMIT** 5;

[](https://php.zone/uploads/img/202106106141f7e2d24900b41ad8516510dab81949f21089325aa65b3f04388f9f254744.png)

Как мы видим, теперь базе пришлось считать всего 5 строк из таблицы. Это произошло благодаря использованию индекса по полю id. Мы просто отфильтровали ненужные нам строки и начиная с нужной получили первые 5.  
Теперь давайте напишем запрос, который позволит получить записи с предыдущей страницы. Для этого нам нужно получить 5 статей, которые идут до первой статьи на текущей странице. В нашем случае это записи с id > 10:

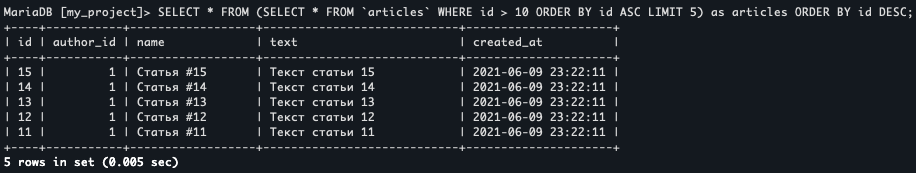
**SELECT** \* **FROM** `articles` **WHERE** **id** > 10 **ORDER** **BY** **id** **ASC** **LIMIT** 5;

[](https://php.zone/uploads/img/2021061063148b7e428ab7ddb6289b05e4773da048367c88e96fbd0385c3bd9a0e66c82f.png)

Как видим, запрос тоже выполнился быстро. Давайте проанализируем и его.  
[](https://php.zone/uploads/img/20210610f8edbaba81bfd4bee1e04f1de7aa387f27a1251518a7ccc0e97eb2645a9603b8.png)

Видим r\_rows = 5, всё в порядке.  
Однако, как вы могли заметить, записи отсортированы не в том порядке, в котором они должны выдаваться в блоге. Давайте исправим это.

**SELECT** \* **FROM** (**SELECT** \* **FROM** `articles` **WHERE** **id** > 10 **ORDER** **BY** **id** **ASC** **LIMIT** 5) **as** articles **ORDER** **BY** **id** **DESC**;

[](https://php.zone/uploads/img/202106108e9f4f311cee2e2c56cc172b7ea8140813ef3d5bd13acbd34748fab8cdf506bd.png)

Теперь всё в порядке. Это можно сделать и средствами PHP, вызвав функцию array\_reverse() после получения результатов из базы. Как вы это сделаете – не так важно, это дешевая операция, относительно той, что была, когда мы работали через OFFSET.

Что-ж, остаётся только запрограммировать логику. Вместо url-ов вида myproject.loc/777 нам нужно сделать 2 поддерживаемых урла типа:

* myproject.loc/before/777
* myproject.loc/after/777

для получения страницы статей до id=777 и после id=777, соответственно.

Добавляем в контроллер 2 новых экшена:

\MyProject\Controllers\MainController

**public** **function** **before**(int $id)

{

}

**public** **function** **after**(int $id)

{

}

и соответствующие им роуты:

myproject.loc/src/routes.php

'~^before/(\d+)$~' => [\MyProject\Controllers\MainController::class, 'before'],

'~^after/(\d+)$~' => [\MyProject\Controllers\MainController::class, 'after'],

Добавим в сущность статьи метод для получения статей для страницы до заданного id:

\MyProject\Models\Articles\Article

*/\*\**

*\* @return Article[]*

*\*/*

**public** **static** **function** **getPageBefore**(int $id, int $limit): **array**

{

$db = Db::getInstance();

$sql = sprintf('SELECT \* FROM (SELECT \* FROM '.**self**::getTableName().' WHERE id > :id ORDER BY id ASC LIMIT %d) as articles ORDER BY id DESC;', $limit);

**return** $db->query($sql, ['id' => $id], **self**::class);

}

И аналогичный метод для получения статей после заданного id:

*/\*\**

*\* @return Article[]*

*\*/*

**public** **static** **function** **getPageAfter**(int $id, int $limit): **array**

{

$db = Db::getInstance();

$sql = sprintf('SELECT \* FROM '.**self**::getTableName().' WHERE id < :id ORDER BY id DESC LIMIT %d;', $limit);

**return** $db->query($sql, ['id' => $id], **self**::class);

}

Помимо этого, нам также понадобятся ещё 2 метода, которые позволят узнать, есть ли после какого-то (или до какого-то) id ещё записи. Это потребуется, чтобы узнать, есть ли следующая или предыдущая страница, чтобы нарисовать ссылки.

**public** **static** **function** **hasNextPage**(int $pageLastId): **bool**

{

$db = Db::getInstance();

$sql = 'SELECT id FROM '.**self**::getTableName().' WHERE id < :id LIMIT 1;';

$result = $db->query($sql, ['id' => $pageLastId]);

**return** !**empty**($result);

}

**public** **static** **function** **hasPreviousPage**(int $pageFirstId): **bool**

{

$db = Db::getInstance();

$sql = 'SELECT id FROM '.**self**::getTableName().' WHERE id > :id LIMIT 1;';

$result = $db->query($sql, ['id' => $pageFirstId]);

**return** !**empty**($result);

}

Осталось написать логику в экшенах контроллера.

**public** **function** **before**(int $id)

{

**$this**->page(Article::getPageBefore($id, 5));

}

**public** **function** **after**(int $id)

{

**$this**->page(Article::getPageAfter($id, 5));

}

**private** **function** **page**(array $articles)

{

**if** ($articles === []) {

**throw** **new** NotFoundException();

}

$firstID = $articles[0]->getId();

$lastID = $articles[count($articles)-1]->getId();

**$this**->view->renderHtml('main/main.php', [

'articles' => $articles,

'previousPageLink' => Article::hasPreviousPage($firstID) ? '/before/' . $firstID : **null**,

'nextPageLink' => Article::hasNextPage($lastID) ? '/after/' . $lastID : **null**,

]);

}

Пробуем зайти на страницу, на которой будут записи до id=20: <http://myproject.loc/before/20>

[](https://php.zone/uploads/img/20210610f90d2ff56786190ac3bb2ca2189b821cf56c5afb6223ae0f7324c4c7fa363a9d.png)

Обратите внимание на время генерации страницы. 0.017035 секунды! То есть произошло ускорение в 1,045724/0,017035 = 61,4 раза! Согласитесь, неплохо!

Осталось сделать главную страницу, которая будет рисоваться при запросе <http://myproject.loc/>  
Для этого нам нужно получить id последней статьи, прибавить к нему 1, и вызвать экшен after() с полученным значением.  
Добавим в сущность метод для получения последнего id.

\MyProject\Models\Articles\Article

**public** **static** **function** **getLastID**(): ?**int**

{

$db = Db::getInstance();

$sql = 'SELECT id FROM '.**self**::getTableName().' ORDER BY id DESC LIMIT 1;';

$result = $db->query($sql);

**return** !**empty**($result) ? $result[0]->id : **null**;

}

И дополним наш контроллер:

**public** **function** **main**()

{

$lastID = Article::getLastID();

**if** ($lastID === **null**) {

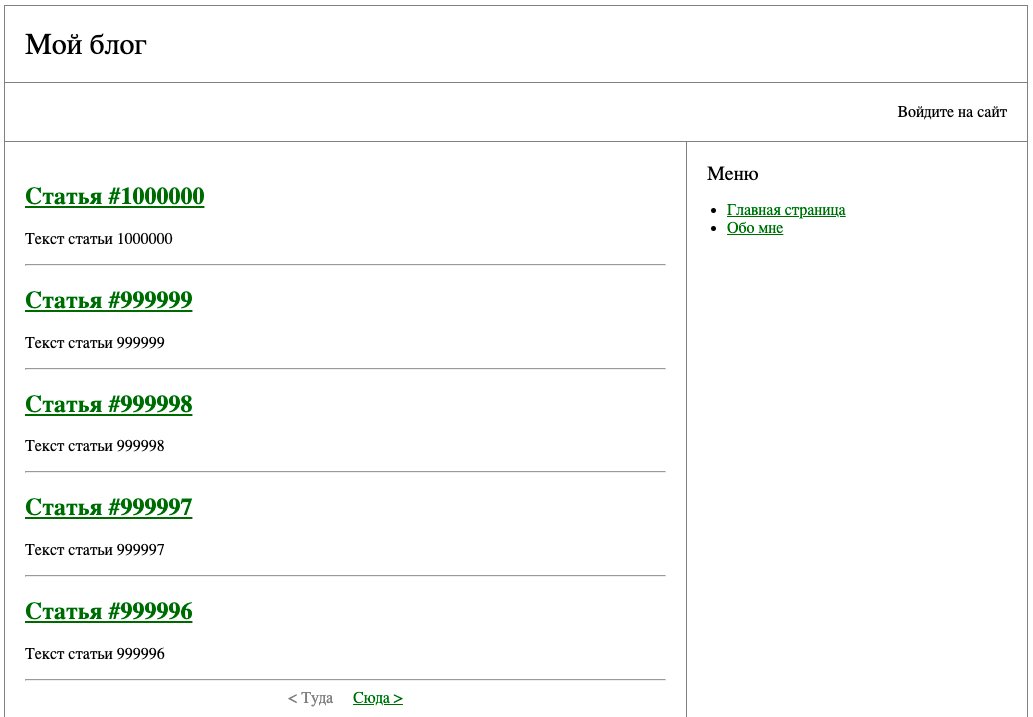
**throw** **new** NotFoundException();

}

**$this**->after($lastID + 1);

}

Проверяем главную страницу <http://myproject.loc/>

[](https://php.zone/uploads/img/20210610542826fe60aaf0e88df1cfd630675bcd4e24cf833ac5e78b6a8d8794229f7b48.png)

Всё в порядке. Страница сгенерировалась за 0.010365.

Код, расположенный в сущности Article легко выносится в класс ActiveRecordEntity, и масштабируется на все другие возможные модели.

Дифф с изменениями вы найдете здесь - <https://github.com/ivashkevitch/oop-v-php-prodvinutyj-kurs/pull/2/files>

А, ну и кстати, осталась ещё пара мест, где можно сэкономить. Во-первых, если мы знаем (а мы знаем), что id статей начинается с 1, то можно избавиться от лишнего похода в базу при проверке наличия следующей страницы:

\MyProject\Controllers\MainController

**private** **function** **page**(array $articles)

{

...

'nextPageLink' => $lastID > 1 ? '/after/' . $lastID : **null**,

...

}

А ещё можно закешировать id последней добавленной статьи, и избежать еще одного запроса в базу. Я сделал файловый кеш прямо в контроллере. Так делать не надо :) Это просто для быстрой демонстрации. Выносите такое куда-нибудь на уровень доменного сервиса. Там же при добавлении новых статей сбрасывайте/обновляйте кеш.

...

**$this**->view->renderHtml('main/main.php', [

'articles' => $articles,

'previousPageLink' => $firstID < **$this**->getLastArticleID() ? '/before/' . $firstID : **null**,

'nextPageLink' => $lastID > 1 ? '/after/' . $lastID : **null**,

]);

}

**private** **function** **getLastArticleID**(): **int**

{

$cacheFile = **\_\_DIR\_\_** . '/../../../cache/last\_article\_id';

$value\_from\_cache = file\_get\_contents($cacheFile);

**if** (!**empty**($value\_from\_cache)) {

**return** (int)$value\_from\_cache;

}

$lastID = Article::getLastID();

file\_put\_contents($cacheFile, $lastID);

**return** $lastID;

}

После этого остаётся единственный запрос в базу, а время генерации страницы уменьшается до 0.005457 секунды.

На этом всё, спасибо что дочитали :)