

Практические занятия №25, 27, 29, 31

Создание простого интернет магазина

На данном этапе Вы уже знаете достаточно, чтобы написать небольшой простой интернет магазин. На это задание мы отвели 4 практических занятия, но Вам что бы все успеть нужно будет самостоятельно работать на этом проектом дома.

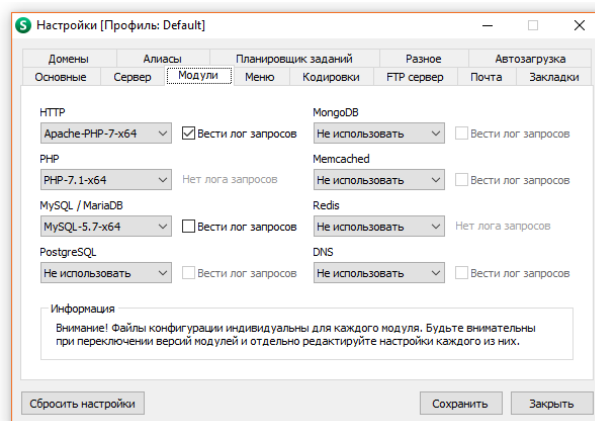
Мы подготовили для Вас инструкцию, в которой описаны самые важные моменты в разработке.

Ваша задача в процессе выполнения данного проекта – получить опыт функционального программирования на PHP, а так же опыт комбинации четырёх технологий: HTML, CSS, JavaScript, PHP.

Почему это задание важно и нужно для Вас? Перед тем как начать писать проект в объектно-ориентированном стиле, как это сейчас принято в серьёзном вебе, Вы должны отточить навыки процедурного программирования. Плюс Вы получите общее представление о процессе разработки web-проектов.

Этап №1 Настройка сервера

Настройка OpenServer. Зайдите в настройки OpenServer и убедитесь, что у Вас установлены следующий модули, как на изображении ниже:



Создайте у себя папку для домена «shop.loc». В ней создайте файл «index.php». В нем разместите функцию «phpinfo();». Перезапустите OpenServer и проверьте работоспособность домена.

Теперь наша задача. Создать одну точку входа на сайт. Для этого мы должны настроить веб сервер Apache, советуем образом. Создаём в корневой директории файл «.htaccess», в нем прописываем следующие настройки сервера:

```
php_value error_reporting E_ALL
```

```
php_value session.auto_start 1
```

```
RewriteEngine on
```

```
RewriteCond %{REQUEST_FILENAME} !-f
```

```
RewriteCond %{REQUEST_FILENAME} !-d
```

```
RewriteRule !(\.(js|json|ico|xml|gif|jpg|jpeg|jpe|png|svg|css|txt|doc|pdf|swf|woff|woff2|eot|otf|svg|ttf)$) index.php
```

php_value session.auto_start 1 - указывает о том, чтобы в начале каждого PHP файла запускалась сессия. Удобно тем, что не нужно вызывать функцию **session_start()**.

php_value error_reporting E_ALL – еще один пример настройки сервера, который указывает какой уровень ошибок нужно показывать интерпретатору PHP.

RewriteEngine on - разрешает использования правил перенаправления запросов.

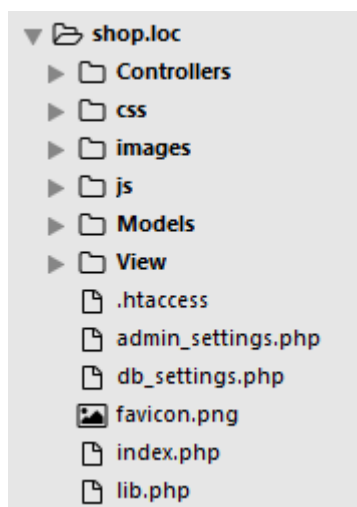
RewriteCond – задаёт правило перенаправления. Подробный разбор этих правил настройки сервера выходит за рамки данного курса. Если просто эти строки означают, что при любой строке запроса необходимо обращаться к файлу `index.php` в корне сайта, кроме запросов к файлам, чьи расширения указаны в скобках.

Теперь Вам нужно проверить как происходят переходы на Вашем сайте. Любой произвольный URL, который не обращается к файлу с расширениями `js`, `json`, `ico`, `xml`, `gif`, `jpg`, `jpeg`, `jpe`, `png`, `svg`, `css`, `txt`, `doc`, `pdf`, `swf`, `woff`, `woff2`, `eot`, `otf`, `svg`, `ttf` должен запускать файл `index.php`.

Этап №2 Создание файлового каркаса

На этом этапе мы создадим структуру файлов и папок и разместим предоставленные материалы.

В конечном варианте наша файловая структура должна выглядеть вот так:



Создаем папки:

- **Controllers** - содержит php файлы с так называемыми контроллерами, которые отвечают за взаимодействия с моделями (папка Models), а так же, за вызов html отображения (папка View).
- **css** - содержит css файлы стилей, подключаемые на html странице.
- **images** - содержит изображения для сайта, такие как иконки, бэкграунд, лого, а также, изображения товаров
- **js** - содержит javascript файлы, подключаемые на html странице.
- **Models** - содержит php файлы, которые работают с базой данных и обеспечивают базовый функционал.
- **View** - содержит html файлы.

Создаём отдельные файлы:

- **admin_settings.php** - файл, в котором будут храниться логины и пароли для входа в административную панель сайта.
- **db_settings.php** – тут будут находиться настройки для подключения к базе данных
- **favicon.png** - иконка для закладок в браузере, копируйте её из предоставленных материалов.
- **lib.php** - содержит маршрутизатор страниц и общие функции.

Сейчас Вам нужно правильно разместить «Предоставленные материалы». В них хранятся стили CSS для публичных страниц и для административной части сайта. Картинки, используемые на сайте. Java Script код для интерактивности страниц.

Начнем размещение материалов:

- скопируйте изображение favicon.png в корень сайта, если Вы это еще не сделали.
- содержимое папки "Стили", скопируйте в уже созданную вами папку "css", которая находится в корне сайта.
- Содержимое папки "Картинки", необходимо скопировать в папку "images".
- Содержимое папки "javascript", скопировать в папку "js".

На этом вся подготовительная работа закончена.

Этап №3 Создание маршрутизатора

Для удобной дальнейшей работы создайте функцию просмотра содержания переменных в файле `lib.php`:

```
function dd() {  
    $arg_list = func_get_args();  
    header('Content-Type: text/html; charset=utf-8');  
    echo '<pre>';  
    foreach ($arg_list as $arg) {  
        var_dump($arg);  
    }  
    echo '</pre>';  
    die();  
}
```

Фактически это функция-обёртка над `var_dump`.

Переходим к маршрутизации. Маршрутизация страниц или роутинг - это механизм вызова необходимых функций PHP в зависимости от строки запроса. Предположим, на нашем сайте есть 2 страницы - страница категории и страница товара. Вы можете перейти на страницу какой-нибудь 13ой категории введя следующий запрос:

shop.loc/category/13/

Или посмотреть подробное описание некоего товара:

shop.loc/product/13/

На странице категории отображается несколько товаров с кратким содержанием. А на странице товара, выводится полное описание, которое предоставил интернет магазин. По итогу работы этих запросов, выбираются разные html-шаблоны и выполняются разные запросы к базе данных. Для того, чтобы выбирать какую функцию выполнить - необходимо как-то различать строки запроса. Этим и занимается роутер(маршрутизатор), что разбирает строку запроса и, в зависимости от значений, составляющих запрос, выбирает алгоритм дальнейших действий.

Откройте файл `lib.php` и создайте функцию `route`:

```
function route() {}
```

Затем, добавим заголовок с указанием типа контента и кодировки в файл **`index.php`**. Так же подключим файл **`lib.php`** и вызовем функцию **`route()`**. Эта функция является нашим маршрутизатором и будет вызываться каждый раз, когда мы будем обращаться к нашему сайту.

```
header("Content-Type: text/html; charset=utf-8");  
  
include 'lib.php';  
  
route();
```

В суперглобальном массиве `$_SERVER` в ячейке **`REQUEST_URI`** храниться строка запроса без домена, мы будем ее использовать. Подробнее о `$_SERVER` можно прочитать тут

<http://php.net/manual/ru/reserved.variables.server.php>

Начинаем обработку запроса. Первое что нам нужно это отсечь GET параметры если они есть. Для этого используем `explode` по символу «?». Далее срезаем «слеши» с обеих сторон.

```
function route() {  
    $explodeURI = explode('?', $_SERVER['REQUEST_URI']);  
    $url = trim( $explodeURI[0] , '/');  
}
```

Идём дальше. Если в строке запроса встретятся кириллические символы, или другие символы из много байтовой кодировки, то они будут закодированы. Например так:

<http://shop.loc/%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0/13?id=13&type=page>

Чтобы вместо закодированного сервером значения получить исходное, нам его надо раскодировать с помощью функции **`urldecode`**. Обернем результат функции **`trim`** функцией **`urldecode`**:

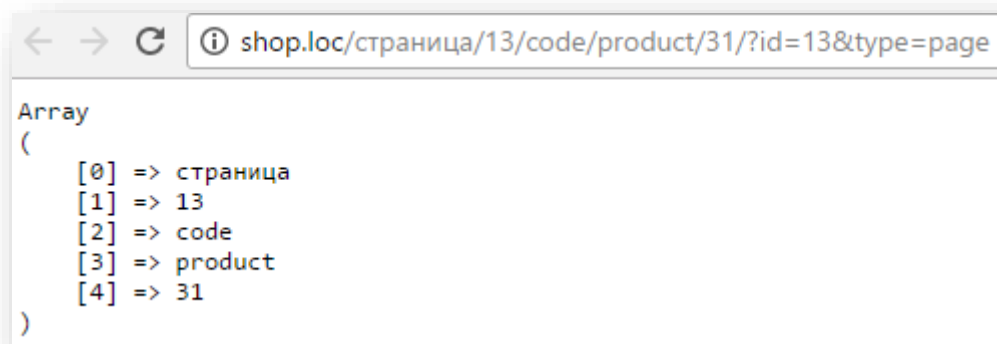
```
function route() {  
    $explodeURI = explode('?', $_SERVER['REQUEST_URI']);  
    $url = urldecode( trim( $explodeURI[0] , '/') );  
}
```

Уже лучше. Но и это еще не всё. Нам нужно получить отдельные компоненты строки запроса. Для этого нужно еще раз разбить строку запроса по слешам используя функцию `explode`.

```
function route() {  
    $explodeURI = explode('?', $_SERVER['REQUEST_URI']);  
    $url = urldecode( trim( $explodeURI[0] , '/') );  
  
    // тут будут отдельные компоненты строки запроса  
    $slices = explode('/', $url);  
  
    dd($slices);  
}
```

Итак, разберем что мы уже сделали. Взяли адресную строку, мы отрезали от неё ту часть, в которой содержатся параметры строки (после символа ?). Далее, удалили символ слеши в начале и конце. Потом, декодировали строку, чтоб получить корректные данные, если там не латинские буквы и цифры. И наконец, очищенную от ненужных нам данных строку, мы разбиваем на части, содержащиеся между слешами. Перейдите по ссылке

<http://shop.loc/%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0/13/code/product/31/?id=13&type=page> .



Теперь переходим к основной работе роутера - определение что вызвать при определенном запросе.

После того, как мы научились разделять полученную адресную строку на кусочки, можно их использовать для подключения нужного контроллера (файла с PHP кодом), и определения, какую его функцию выполнить.

Базовый шаблон у нас такой:

http://shop.loc/<имя_файла_php>/<имя_функции_обработчика>/

Это очень удобно - содержать несколько функций в одном контроллере (файле) для работы с подобными друг другу адресами.

Например, у нас будет контроллер, который отвечает за страницу корзины. На этой странице можно совершить несколько действий, таких как:

- Отображение страницы корзины (index)
- Изменение количества товаров (changeCount)
- Удаление товара (delete)
- Добавление товара (add)
- Покупка содержимого корзины (makeBid)

Т.е. для выполнения каждой функции существует своя строка запроса, но неизменно первым элементом разобранного запроса будет, к примеру, basket, а следующий элемент массива будет определять - какую функцию необходимо выполнить. Итак, первый элемент массива определяет имя контроллера, а второй функцию - это в простом случае.

Объявим три переменные в функции route:

\$controllerName - содержит имя контроллера

\$action - функция контроллера

\$params - параметры для функции \$action.

Про параметры еще ничего не было сказано. Параметры будут использоваться на странице категорий и странице товара. Например, перейдя на страницу shop.loc/category/13/ число 13 будет параметром, т.к. мы не сможем создать функцию на каждую категорию.

Но вернемся к отображению главной странице. Если мы хотим, чтобы, перейдя по адресу `http://shop.loc/` отобразилась главная, значит, что нам нужно написать условие такого вида:

```
function route() {  
  
    $explodeURI = explode('?', $_SERVER['REQUEST_URI']);  
    $url = urldecode( trim( $explodeURI[0] , '/' ) );  
  
    // тут будут отдельные компоненты строки запроса  
    $slices = explode('/', $url);  
  
    $controllerName = '';  
    $action = '';  
    $params = NULL;  
  
    if($slices[0] == '') {  
        $controllerName = 'index';  
        $action = 'index';  
    }  
  
}
```

Теперь, надо немного преобразовать имя контроллера, чтобы оно соответствовало нашей структуре файлов. Т.к. наши контроллеры лежат в папке `Controllers`, то необходимо перед именем контроллера написать имя папки:

```
$controllerPath = 'Controllers'.DIRECTORY_SEPARATOR.$controllerName.'Controller.php';
```

DIRECTORY_SEPARATOR – это встроенная константа PHP, которая содержит символ разделителя в текущей файловой системе.

Далее подключаем файл `$controllerPath` и вызываем нужную нам функцию:

```
$controllerPath = 'Controllers'.DIRECTORY_SEPARATOR.$controllerName.'Controller.php';  
  
include $controllerPath;  
  
$action($params);
```

Создайте файл ***indexController.php*** в папке ***Controller*** и напишем в нем функцию ***index***. Проверьте действительно вызывается эта функция при переходе на главную страницу. Если появились ошибки найдите и устраните их.

Задание:

Самостоятельно допишите проверку существует ли файл по пути в переменной ***\$controllerPath***, если такого файла нет, выводите сообщение об ошибке. В конце процесса разработки мы заменим сообщения об ошибке на перенаправление на страницу 404.

Так же добавьте проверку на существование функции с именем, которое храниться в переменной ***\$action***. Если такой функции не оказалось, выводите сообщение об ошибке.

Этап 4 Создание функции вывода HTML страниц.

За вывод HTML страниц у нас будет отвечать функция **view**, которую мы сейчас создадим в файле **lib.php**.

На картинке ниже представлен самый простой вариант функции view.

```
function view($dir, $filename) {  
    include 'View'.DIRECTORY_SEPARATOR.$dir.DIRECTORY_SEPARATOR.$filename;  
}
```

Тут аналогично тому, как мы подключаем контроллер, подключаем вид. Первый параметр **\$dir** - содержит имя папки, в которой содержится html файл с именем **\$filename**.

Давайте перейдем в файл **indexController.php** и напишем вызов функции view:

```
function index() {  
    view('index', 'index.html');  
}
```

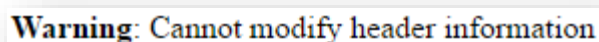
Давайте зайдём в папку **View** (которая находится в корне сайта), создаём там файл вида **index/index.html** с таким содержимым:

```
<h1>Онлайн магазин</h1>
```

Переходим в браузере на главную и получаем вот такой результат:



Как вы, возможно знаете, в файлах php не рекомендуется закрывать php-код с помощью тега ">", его желательно вообще опустить. Для чего это делается? Если вы закроете php тег и после него поставите любой символ, хоть пробел, то он выведется на экран. Чем это может быть плохо? Возможно, вы встречались с таким предупреждением:</p



Оно говорит о том, что браузер уже начал вывод текста для пользователя, а вы до сих пор пытаетесь послать какой-то заголовок, например - определение кодировки документа.

Чтобы такой проблемы не возникало, используется буферизированный вывод. Добавим использования буфера вывода в *index.php*

```
// инициализируем буфер вывода
ob_start();

header("Content-Type: text/html; charset=utf-8");

include 'lib.php';

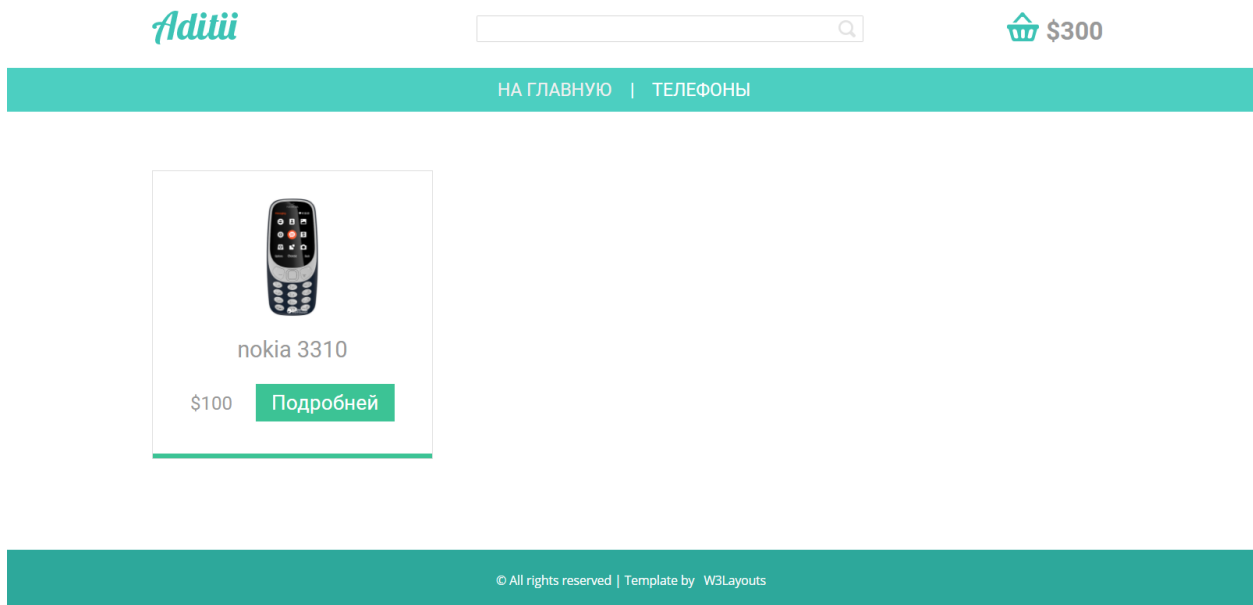
route();

// выводим содержимое буфера
ob_end_flush();
```

Этап №5 Подключение основных HTML страниц

В конце предыдущего этапа мы остановились на том, что подключили тестовый пример html страницы с помощью нашей функции **view**. В этом этапе рассмотрим более подробный пример, а также, вас ждет небольшое задание.

Воспользуйтесь папкой с дополнительными материалами – “Предоставленные файлы”. Там есть папка “Виды”, скопируйте ее содержимое в папку View в корне нашего сайта с заменой. На главной странице у Вас должен стать этот макет:



Давайте подключим таким же образом страницу товара. Откройте файл `lib.php` и продолжим писать маршрутизацию. Для этого нам необходимо написать условие, при котором отображается страница товара. Сделаем так, что при переходе по адресу `shop.loc/product/` отображался товар. Но этого адреса недостаточно, товаров может быть один, а может быть миллион. Нам нужно знать, какой конкретно товар отображать. Поэтому, окончательный вид будет такой:

`shop.loc/product/{id}/`

Где, **`{id}`** – число, соответствующее идентификатору товара в базе данных.

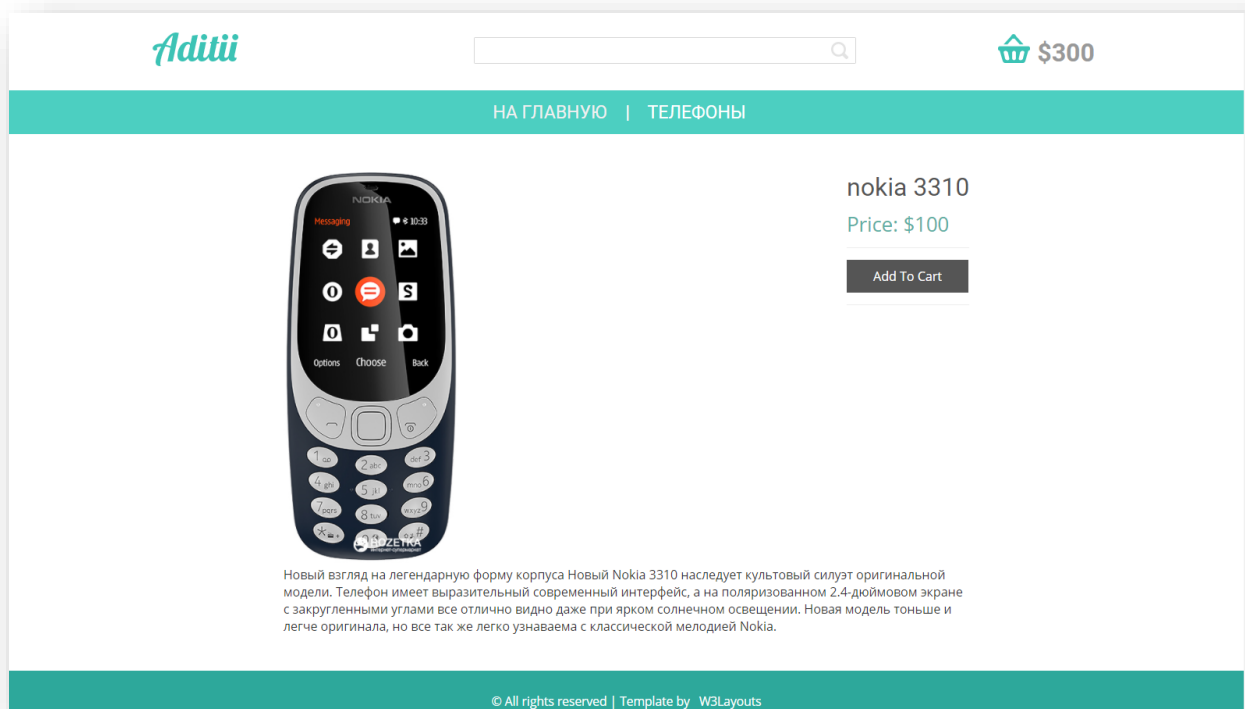
Допишем условие маршрутизации:

```
if($slices[0] == '') {  
    $controllerName = 'index';  
    $action = 'index';  
}  
elseif($slices[0] == 'product') {  
    $controllerName = 'product';  
    $action = 'index';  
    $params = (int)$slices[1];  
}
```

Создайте в папке **Controllers** файл **`productController.php`** напишите в нем функцию `index`:

```
function index($productId) {  
    view('product', 'index.html');  
}
```

Теперь, перейдя по адресу <http://shop.loc/product/1/>, вы увидите следующее:



Сравните файлы:

product/index.html

index/index.html

Есть ли у них повторяющиеся части? Да, есть – начало html документа и конец. Эти совпадающие части будут присутствовать на всех страница доступных пользователю. Поэтому, давайте сделаем их подключаемыми чтобы в разных файлах, таких как ***product/index.html*** и ***index/index.html*** содержалась бы только различная информация. Этот подход уменьшит размер каждого html файла, а также, если мы захотим изменить какую-то общую информацию – подключение еще одной java script библиотеки – то мы сделаем это в одном месте, а не на каждой html странице.

В папке **View\helpers** у вас лежат следующие файлы отдельных частей вёрстки:

- head.html
- header.html
- footer.html

В **head.html** содержится начало html документа начиная от doctype и заканчивая открывающимся тегом body.

В **header.html** находится верхнее меню сайта, которое содержит лого, поиск, информацию о корзине пользователя, а также, категории сайта.

В **footer.html** нижняя часть сайта.

Перед тем как подключать head, header и footer ко всем html страницам файла, давайте удалим дублирующую информацию из **product/index.html u index/index.html**. Для этого выделите html-код между следующими комментариями и удалите:

<!-- start head --> и <!--end head --> (содержимое этой секции находится в helpers\head.html)

<!-- start header --> и <!-- end header --> (содержимое этой секции находится в helpers\ header.html)

<!-- start footer --> и <!-- end footer -->(содержимое этой секции находится в helpers\ footer.html)

Теперь, если вы перейдете на главную(shop.loc/) или на страницу товара (shop.loc/product/1/), то увидите примерно следующее:

Главная страница



Страница товара



Мы видим такое отображение, т.к. у нас не подключены стили, java script и всё то, что написано в тегах head. Давайте это исправим.

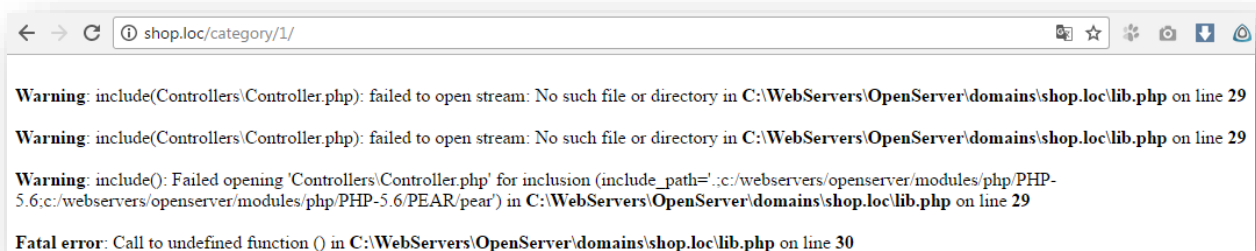
Изменим функцию **view** в файле **lib.php**. Добавим подключение head, header и footer файлов:

```
function view($dir, $filename) {
    include 'View'.DIRECTORY_SEPARATOR.'helpers'.DIRECTORY_SEPARATOR.'head.html';
    include 'View'.DIRECTORY_SEPARATOR.'helpers'.DIRECTORY_SEPARATOR.'header.html';
    include 'View'.DIRECTORY_SEPARATOR.$dir.DIRECTORY_SEPARATOR.$filename;
    include 'View'.DIRECTORY_SEPARATOR.'helpers'.DIRECTORY_SEPARATOR.'footer.html';
}
```

Теперь, наша функция view будет оборачивать каждый html шаблон страницы в head, header и footer.

Обновите страницы главной и товара, и вы увидите, что всё вернулось к прежнему виду.

Подошел к концу очередной этап конструирования интернет магазина, и вы получите задание для выполнения. Сейчас вы можете перейти по адресу `shop.loc/category/1/` и будет выведено такое содержание страницы:



Исправьте это, выполнив задание:

1. Создайте маршрут, по которому будут отображаться категории.
2. Создайте файл контроллера для категорий
3. Используйте вид главной страницы для отображения категорий.

Немного разъясним по пунктам:

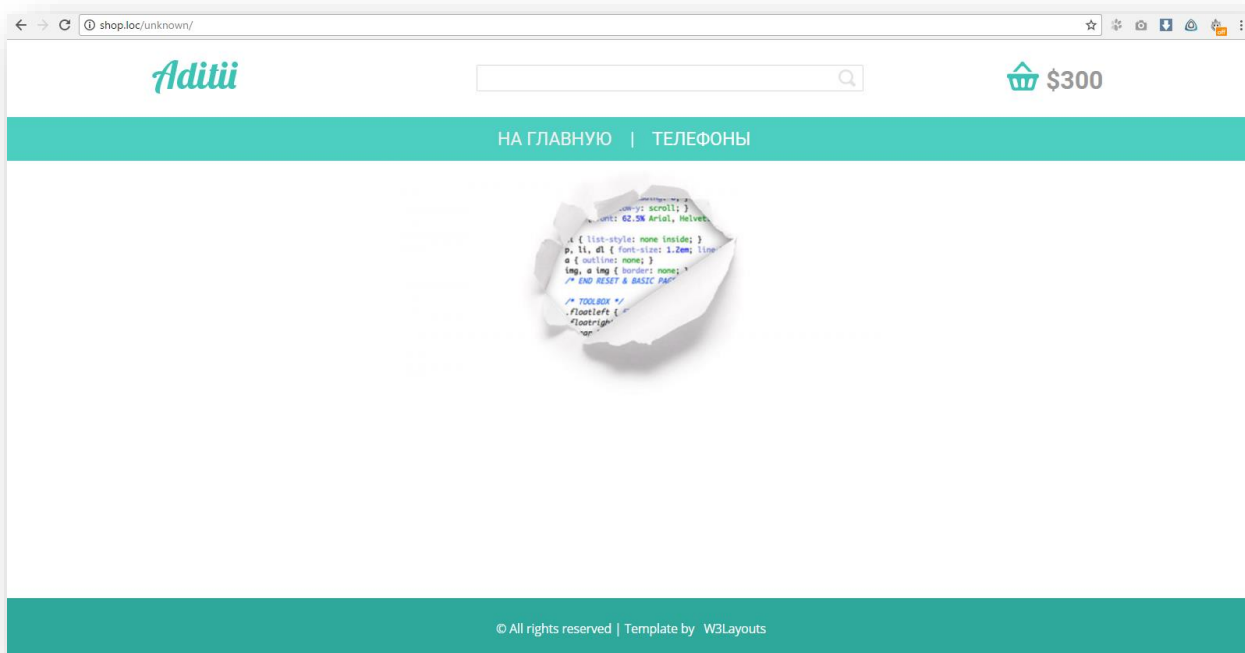
1. Чтобы создать маршрут на категории, дополните if-else маршрутизатора используя в качестве условия проверки строку **"category"**. Категорий, также, как и продуктов, может быть много, поэтому, контроллер category и его функция index будут аналогичны productController. Т.е вам необходимо использовать первый элемент массива \$slices как имя контроллера, а второй – как параметр.
2. Создайте контроллер **category**.
3. Содержимое функции index в контроллере категорий будет таким же т.к. страницы категорий отличаются от главной лишь выборкой из базы данных.

Когда вы правильно сделаете задание, то вместо ошибки будет выведено тоже содержимое что и на главной.

Дополнительное задание: Создание страницы 404.

Продолжить if-else роутера и написать его заключительную часть - else. Записать в переменную имя контроллера – error, а функция этого контроллера – status404. Написать error контроллер, который использует для вывода на экран страницу, находящуюся в **View\error\404.html**. Её вы можете взять из дополнительных материалов Виде\error\404.html.

Если сделаете всё правильно, то при вводе в адресной строки вида shop.loc/unknown/, вы получите:



Этап 6. Пути обработчики и административная панель сайта

На данный момент у вас есть проверка для главной страницы, страницы продуктов, которую мы написали вместе на предыдущем этапе, страницы категорий, если вы выполнили задание и наконец, проверка ввода несуществующей страницы, у тех, кто сделал дополнительное задание, итого - 4 проверки.

Все остальные пути на сайты мы представили в таблице ниже. Вы можете обращаться к этой таблице в процессе разработки сайта. Вы так же можете настроить роутер сразу же для всех страниц сейчас, и можете дописывать роутер в процессе разработки по частям

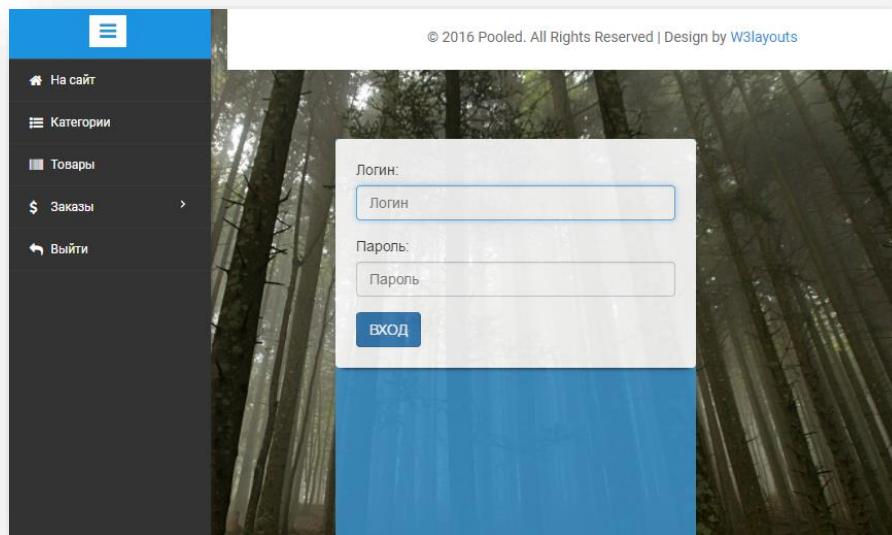
URL	Назначение	Контролер	Функция обработчик	Файл вида
/basket/	Главная страница корзины	basket	index	basket/index.html
basket/add/	(ajax) Добавление товара в корзину	basket	add	
basket/changeCount/	(ajax) Изменение кол-ва товаров в корзине	basket	add	
/basket/delete/	(ajax) Удаление позиции из корзины	basket	delete	
basket/makeBid/	(ajax) Обработка заказа	basket	makeBid	используется модальное окно basket/modal_orderForm.html
basket/thanks/	Благодарности	basket	thanks	basket/thanks.html
/search/	Страница поиска	search	index	search/index.html
Административная часть сайта				
Все пути будут начинаться с /admin/				
admin/auth/	Страница авторизации			используется модальное окно admin/auth.html

admin/logout/	«выход» из админки, сброс авторизации			
/admin/category/	Управление категориями с функциями добавления, удаления, изменения категорий			admin/category.html
admin/products/	Управление товарами			admin/products.html
admin/products/{\$id}/	Страница одного товара			admin/productDetail.html admin/productEdit.html
admin/orders	Управление заказами (статусы заказов: отклонен, выполнен, оплачен)			admin/orders.html
shop.loc/admin/order/{id},	Просмотр определённого заказа Статусы заказа: 1 - новый заказ 2 - заказ выполнен 3 - отклонен 4 - архив 5 - удалён			admin/order.html

Сейчас, ваша цель отобразить страницу админки. Для этого нужно написать `admin` контроллер и функцию `orders`, которая бы вывела **orders.html** из папки **admin**. Всё бы ничего, но в админке используется другой шаблон, поэтому, давайте напишем еще одну функцию, аналогичную функции `view` в **lib.php**. Откройте этот файл и после функции `view`, напишите функцию **adminView**:

```
function adminView($dir, $filename) {  
  
    include 'View'.DIRECTORY_SEPARATOR.'helpers'.DIRECTORY_SEPARATOR.'admin_head.html';  
    include 'View'.DIRECTORY_SEPARATOR.$dir.DIRECTORY_SEPARATOR.$filename;  
    include 'View'.DIRECTORY_SEPARATOR.'helpers'.DIRECTORY_SEPARATOR.'admin_footer.html';  
  
}
```

Всё тоже самое что и раньше, но на страницах админки будет использоваться функция `adminView` для отображения видов. Также, напишите функцию для уже описанного маршрута `auth` используя шаблон `auth.html` из папки `admin`. Если сделаете всё правильно, то получится что-то такое:



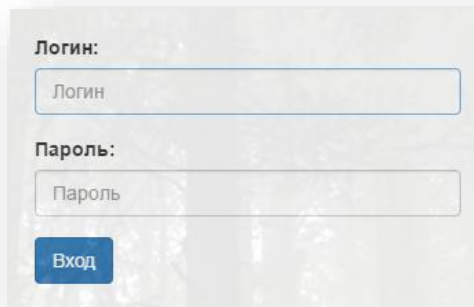
это не совсем то, что надо, поэтому, вернитесь в функцию **`adminView`** и добавьте проверку в начале функции:

```
if($filename == 'auth.html') {  
    include 'View'.DIRECTORY_SEPARATOR.$dir.DIRECTORY_SEPARATOR.$filename;  
    exit;  
}
```

Для страницы авторизации администратора, нам не нужны дополнительные шаблоны. Проверьте, теперь форма авторизации должна выглядеть более атмосферно.

Мы плавно перешли к необходимости проверки, а администратор ли к нам зашел или пользователь "случайно" ввел существующий адрес административной части сайта?

Взглянем на форму авторизации:



Логин:

Пароль:

Вход

Простейшая форма, состоящая из двух полей. При отправке формы, данные посылаются по адресу ***shop.loc/admin/auth/*** и там обрабатываются, согласно нами описанному маршруту, с помощью функции ***auth*** контроллера ***admin***. В этой функции необходимо проверить, соответствуют ли введенные данные, данным администратора. Но откуда взять данные администратора? Введите эти данные в файл ***admin_settings.php***.

```
$admins = array(  
    array('login' => 'root', 'pass' => '123'),  
);
```

Значения логина и пароля можете придумать свои.

Вам необходимо в начале функции ***auth*** подключить файл ***admin_settings.php*** и пройти по всем элементам массива ***\$admins*** из этого файла. Если введенные данные совпали с именем и паролем одного и того же элемента, значит данные введены верно и надо их записать в сессию таким образом:

В файле ***lib.php*** напишите под функцией ***view***, функцию ***isAuth***, которая будет проверять авторизован ли пользователь или нет:

```
function isAuth(){  
    if(empty($_SESSION['login']) || empty($_SESSION['pass'])) return false;  
    return true;  
}
```

Проверяем, если в сессии нет значения для ключа ***login*** или ***pass***, то возвращаем ***false***. А если значения есть, то ***true***. Теперь мы можем использовать эту функцию для проверки доступа к административной части сайта. Если пользователь не авторизован, то перенаправим его на страницу авторизации.

Этап №6 Создание базы данных

Необходимо создать базу данных в phpMyAdmin под именем online-shop.

А так же, 3 таблицы:

- categories
- products
- orders

categories		
id	int(11)	
name	varchar(50)	
url	varchar(50)	
display	tinyint(1)	
Add field		

products		
id	int(11)	
title	varchar(60)	
img_src	varchar(50)	
price	float	
description	text	
category_id	int(11)	
on_main	tinyint(1)	
display	tinyint(1)	
Add field		

orders		
id	int(11)	
time	int(11)	
phone_number	varchar(25)	
email	varchar(254)	
fio	varchar(66)	
address	varchar(66)	
serialize_data	blob	
paid	tinyint(1)	
status	tinyint(1)	
common_price	float	
Add field		

Начнем по порядку, как на картинке.

Таблица **categories**:

Содержит категории товаров.

Имя	Тип	Описание	Значения
id	int(11)	первичный ключ, автоинкремент	
name	varchar(50)	используется для отображения названия категории на сайте в меню выбора категорий	
url	varchar(50)	Содержит латинскую интерпретацию поля name	
display	tinyint(1)	выводить на сайт	Выводить - 1 Не выводить - 0

Таблица **products**:

Содержит данные о товарах

Имя	Тип	Описание	Значения
id	int(11)	первичный ключ, автоинкремент	
title	varchar(60)	Название	
img_src	varchar(50)	Путь к картинке относительно корня сайта	
price	float	цена товара	
description	text	описание товара	
category_id	int(11)	идентификатор категории к которой принадлежит товар	
on_main	tinyint(1)	выводить на главной или нет	Выводить - 1 Не выводить - 0
display	tinyint(1)	выводить на сайт	Выводить - 1 Не выводить - 0

Таблица **orders**:

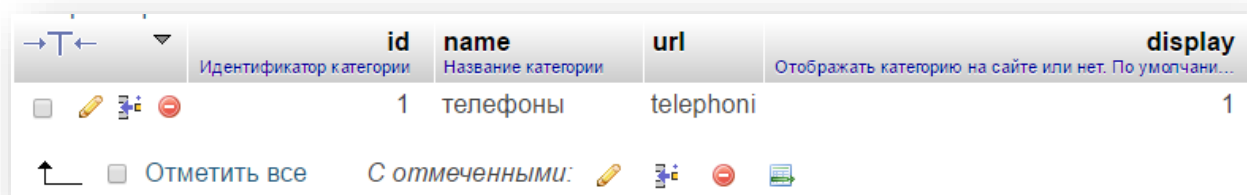
Содержит данные о заказах пользователей

Имя	Тип	Описание	Значения
id	int(11)	первичный ключ, автоинкремент	
time	int(11)	Содержит время создания заказа.	Возвращаемое значение от функции time() в языке PHP.

phone_number	varchar(25)	Содержит телефонный номер заказчика	
email	varchar(254)	Содержит email адрес заказчика	
fio	varchar(66)	Фамилия, Имя, Отчество заказчика	
address	varchar(66)	Адрес для доставки заказа	
serialize_data	blob	Содержит сериализованные данные заказа	
paid	tinyint(1)	Оплачен заказ или нет	0 - заказа не оплачен. 1 - оплачен
status	tinyint(1)	Содержит статус заказа	1 - Новый 2 - Выполнен 3 - Отклонен 4 - Архив 5 - Удалён
common_price	float	общая стоимость заказа	

Добавьте вручную одну категорию и товар через phpMyAdmin такого вида:

Категория



	id	name	url	display
	Идентификатор категории	Название категории		Отображать категорию на сайте или нет. По умолчанию...
<input type="checkbox"/>	1	телефоны	telephoni	1

↑ ☐ Отметить все С отмеченными:

Товар

id Идентификатор товара	title Название товара	img_src Ссылка на изображение товара	price Цена
1	nokia 3310	images/nokia_3310.jpg	74.5

description

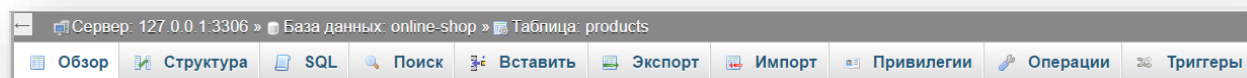
Описание

Новый взгляд на легендарную форму корпуса

Новый Nokia 3310 наследует культовый силуэт оригинальной модели. Телефон имеет выразительный современный интерфейс, а на поляризованном 2.4-дюймовом экране с закругленными углами все отлично видно даже при ярком солнечном освещении. Новая модель тоньше и легче оригинала, но все так же легко узнаваема с классической мелодией Nokia.

category_id Идентификатор категории в которой находится товар	on_main Выводить товар на главной странице?	display Отображать товар или нет. По умолчанию товар скрыт
1	1	1

Для того чтобы добавить запись в таблицу, в верхнем меню phpMyAdmin кликните на "Вставить"



Этап №6 Подключение к базе данных, создание инструментов для работы с базами данных.

Открываем файл **db_settings.php** и определяем в нем константы для работы с базами данных, далее подключаем этот файл через include к нашему сайту.

```
const DB_HOST      = 'localhost';
const DB_USER_NAME  = 'root';
const DB_PASSWORD   = '';
const DB_NAME       = 'online-shop';
const DB_PORT       = 3306;
```

На рисунке выше представлены настройки по умолчанию для OpenServer, у Вас могут быть другие настройки.

Для работы с базой данных будем использовать библиотеку [MySQLi](#). Напишем свой абстрактный слой для работы с БД. Четыре функции для каждого типа запроса - **insert, select, update, delete**. Но для начала, напомним 3 функции для подключения к БД, выполнения запроса, закрытия подключения.

Откройте файл **lib.php** в самом верху php кода объявите глобальную переменную \$mysqli:

```
$mysqli;
```

Делаем её глобальной чтобы не передавать от одной функции в другую и не создавать дополнительные локальные переменные.

Далее пишем функцию **db_connect** с таким содержимым:

```
function db_connect(){
    require_once 'db_settings.php';
    global $mysqli;

    $mysqli = mysqli_connect(DB_HOST, DB_USER_NAME, DB_PASSWORD, DB_NAME, DB_PORT);
    if (mysqli_connect_errno($mysqli)) {
        echo "Не удалось подключиться к серверу MySQL: " . mysqli_connect_error();
        exit;
    }
}
```

Подключаем файл с объявленными константами необходимыми для подключения к серверу БД. Используем глобальную переменную в которой содержится подключение к серверу БД. Осуществляем подключение с помощью функции **mysqli_connect** и констант. Если соединиться не получилось, то выводим ошибку.

Далее, напишите функцию для выполнения запросов:


```
function db_query($sql, $dml = false){
    global $mysqli;

    $res = mysqli_query($mysqli, $sql);

    if ($res === false) {
        print_r($sql);
        echo "<br><br><br><br>";
        printf("Error message: %s\n", mysqli_error($mysqli));
        die();
    }

    if($dml){
        return $res;
    }

    $result = array();
    while ($row = mysqli_fetch_assoc($res)) {
        $result[] = $row;
    }

    return $result ? $result : false;
}
```

Первый параметр - sql-запрос. Второй - является ли запрос insert, update или delete. Это значит, если запрос select, то нужно вернуть выборку, иначе, только результат запроса - выполнен или нет. Если запрос выполнен с ошибкой и то выводится тело запроса и его ошибка. Далее в цикле извлекаются данные из результатов запроса (если это select запрос). В конце возвращаются данные или false.

Следующей функцией будет, функция закрытия соединения с сервером БД:

```
function db_close(){
    global $mysqli;
    if(!mysqli_close($mysqli)) echo 'Не удалось закрыть соединение с базой данных!';
}
```

Закрываем соединение. Если не получилось - выводим ошибку.

Итак, у нас есть базовые функции для работы с БД. Мы уже можем выполнить любой запрос передав его тело в функцию db_query:

```
db_connect();

$result = db_query("SELECT * FROM categories");

db_close();
```

Но мы пойдем дальше и сделаем наш программный интерфейс более удобным. Напишем 4 функции:

- insertQuery
- selectQuery
- updateQuery
- deleteQuery

Перепишите функции со слайдов в конец lib.php, после функции db_close.

Разберем по порядку.

```
function insertQuery($table, $data){
    db_connect();

    global $mysqli;

    $sql = "INSERT INTO $table (";
    $values = "VALUES (";

    $newDataColumns = array();
    $newDataValues = array();
    foreach ($data as $columnName => $value) {
        $newDataColumns[] = $columnName;
        $newDataValues[] = $value;
    }

    $sql .= implode(' ', $newDataColumns) . ') ' . $values . implode(' ', $newDataValues) . ')';

    $result = db_query($sql, true);

    $lastId = $result !== false ? mysqli_insert_id($mysqli) : 0;

    db_close();

    return $lastId;
}
```

Первый параметр - имя таблицы, второй - ассоциативный массив с именем колонки в качестве ключа и значение поля.

Здесь вы можете увидеть на слайде 3 функции которые мы только что описали. db_connect в 3ей, db_query в 19ой, db_close в 23ей. Также, используется глобальная переменная \$mysqli для того, чтобы получить идентификатор последней вставленной строки.

В цикле foreach, в 12ой строке, разделяем названия колонок таблицы и их значения. Далее используем функцию [implode](#) для того, чтоб соединить все элементы массива в строку по разделителю запятая и пробел (", "). Далее проделываем ту же процедуру со значениями этих полей. Потом выполняем собранный запрос.

```
function selectQuery($selectFields, $from, $additional_data = null, $limit = null){
    db_connect();

    $sql = "SELECT " . implode($selectFields, ', ');
    $sql .= " FROM $from";
    if(!empty($additional_data))
        $sql .= " $additional_data";
    if(!empty($limit))
        $sql .= " LIMIT $limit";

    return db_query($sql);

    db_close();
}
```

Первым параметром **selectQuery** принимает массив полей, которые необходимо взять из таблицы. Второй параметр - строка, - название таблицы из которой необходимо взять данные. Третий параметр, является необязательным - строка, содержащая в себе условие выборки. И четвертый параметр, также необязательный - число, которое ограничивает выборку по количеству строк.

```
function updateQuery($table, $data, $where, $limit = null){
    db_connect();

    $sql = "UPDATE `{table}` SET ";

    $newData = array();
    foreach ($data as $columnName => $value) {
        $newData[] = "`{$columnName}` = {$value}";
    }

    $sql .= implode(' ', $newData);
    $sql .= " where " . $where;

    if(!empty($limit)){
        $sql .= " LIMIT {$limit}";
    }

    $result = db_query($sql, true);

    db_close();

    return $result;
}
```

У **updateQuery** первый параметр — это имя таблицы. Второй - ассоциативный массив с названием поля и его новым значением. Третий - условие where. И четвертый, необязательный - ограничение на количество строк для обновления.

```
function deleteQuery($table, $where, $limit = null){
    db_connect();

    $sql = "DELETE FROM {table} WHERE $where";

    if(!empty($limit)) $sql .= " LIMIT $limit";

    $result = db_query($sql, true);

    db_close();

    return $result;
}
```

Четвертая функция для работы с базой данных осуществляет удаление. Первый параметр - имя таблицы, второй - условие для удаления, третий, необязательный - ограничение на количество удаляемых строк.

Этап №7 Модели

Модели — это часть кодовой базы, которая будет реализовывать основную логику нашего проекта.

Для начала, используем *init* модель. Эта модель используется на всех страницах, видимых покупателю, и она будет подключаться на всех этих страницах. Остальные модели нужны не на всех страницах, поэтому они будут подключаться выборочно.

Создайте и откройте файл *initModel.php* в папке **Models** и напишите следующий код:

```
function getPublicCategories(){  
    return selectQuery(array('*'), 'categories', 'where display = 1');  
}
```

функция *getPublicCategories*, возвращает массив категорий, которые открыты для покупателей.

```
function getBasketPrice(){  
    $basket = isset($_SESSION['basket']) ? $_SESSION['basket'] : '';  
    if(empty($basket)) return 0;  
    $ids = array();  
    foreach ($basket as $item) {  
        $ids[] = $item['id'];  
    }  
    $ids = implode(' ', $ids);  
    // db_connect();  
    // $db_res = db_query("SELECT id, price from products WHERE id in({$ids})");  
    // db_close();  
    $db_res = selectQuery(array('id', 'price'), 'products', "where id in({$ids})");  
    $basketPrice = 0;  
    foreach ($basket as $k => $item) {  
        $db_item_key = array_search($item['id'], array_column($db_res, 'id'));  
        if($db_item_key === false) {  
            unset($_SESSION['basket'][$k]);  
            continue;  
        }  
        $basketPrice += $db_res[$db_item_key]['price'] * $item['count'];  
    }  
    return $basketPrice;  
}
```

Функция *getBasketPrice*, будет возвращать общую стоимость выбранных товаров пользователем. В нашем случае, она постоянно будет возвращать 0 и вам нужно будет её дописать. Сейчас, можно сказать только то, что содержимое корзины пользователя будет записываться в сессию с ключом *basket*.

У нас есть модель с двумя функциями возвращающие некоторые значения, которыми мы можем воспользоваться. Но для начала, нам нужно вызвать эти функции, а значит - необходимо подключить файл *initModel.php*.

Для подключения моделей, будем использовать функцию ***addModel***. Давайте опишем её в файле ***lib.php***:

```
function addModel($name){  
    include_once 'Models' . DIRECTORY_SEPARATOR . $name . 'Model.php';  
}
```

Как видите, функция очень похожа на ***view*** и ***adminView*** - подключение видов. ***addModel*** даже проще, т.к. нет промежуточной директории.

Подключим ***init*** модель в ***index*** контроллере:

```
addModel('init');
```

Модель подключена, теперь надо вызвать эти функции.

Посмотрите на 4ую строчку скриншота, в ней произойдет вызов функции, которая находится в ***lib.php*** и подключение файла модели, т.е. содержимое файла ***Models\initModel.php*** попадёт в ***lib.php***, далее у нас подключается вид в 7ой строке. В ней тоже используется функция из ***lib.php*** и файл ***View\index\index.html*** также подключится в ***lib.php***. Это значит, если мы объявим в теле функции ***index*** переменную, то НЕ сможем использовать её значение тут же в ***html*** разметке:

```
function index(){  
    addModel('init');  
  
    $data = array(  
        'categories' => getPublicCategories(),  
        'basketPrice' => getBasketPrice(),  
    );  
  
    view('index', 'index.html');  
}
```

Переменная ***\$data*** объявлена в функции ***index***, а содержимое файла вида, подключается в ***lib.php***, поэтому, переменную не получится использовать в ***html*** разметке.

Чтобы всё работало как надо, нам необходимо передать массив ***\$data*** в функцию ***view***. Для этого, давайте изменим сигнатуру реализации функции ***view*** и ***adminView*** в файле ***lib.php***.

Для ***view*** это будет так:

```
function view($path, $name, $data=array()){
```

Для ***adminView***, аналогично:

```
function adminView($path, $name, $data=array()){
```

Т.к. некоторые страницы отображаются без необходимости использования каких-либо данных - т.е. статичные страницы, - то последний аргумент является необязательным.

Перепишем вызов функции view в indexController:

```
view('index', 'index.html', $data);
```

Теперь, на веб-странице в правом углу, мы можем увидеть значение 0 возле корзины.

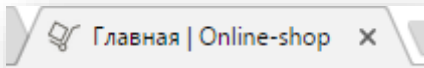
Можем добавить в массив \$data, ключ title

```
$data = array(  
    'title' => 'Главная',  
    'categories' => getPublicCategories(),  
    'basketPrice' => getBasketPrice(),  
);
```

и использовать его в файле **View\helpers\head.php** таким образом:

```
<title><?= isset($data['title']) ? $data['title'] . ' | ' : ''; echo 'Online-shop' ?></title>
```

Если есть title, как в нашем случае, то в закладке браузера будет выведено:



Иначе, отобразится только Online-shop.

Далее выведите значение **basketPrice** в файле **View\helpers\header.php** в элементе с идентификатором basketPrice.

Далее, давайте выведем список категорий всё в том же файле **header.php**. Замените содержимое элемента div.h_menu ul на содержимое скриншота:

```
<? $active = '' ?>  
<? if('/') == $_SERVER['REQUEST_URI']) $active = 'active';?>  
<li class="<?=$active?>"><a href="/">На главную</a></li>  
  
<? if(!empty($data['categories'])) :?>  
    <span class="vl"> |</span>  
    <? $countCat = count($data['categories']); ?>  
    <? $counter = 0 ?>  
    <? foreach($data['categories'] as $cat):  
        $counter++;  
        $active = '';  
        if(strpos($_SERVER['REQUEST_URI'], $cat['url'])) $active = 'active';  
    ?>  
    <li class="<?=$active ?>"><a href="/category/<?=$cat['url'] ?>/"><?=$cat['name'] ?></a></li>  
    <? if($counter < $countCat): ?>  
        <span class="vl"> |</span>  
    <? endif; ?>  
    <? endforeach; ?>  
<? endif ?>
```

Разберем что тут происходит. Для начала, нам нужно вывести элемент меню "На главную", он будет всегда, вопрос только в том, находимся мы на главной или нет. Чтобы определить текущий URL используем массив `$_SERVER` с ключом `REQUEST_URI`. Далее, если есть хоть одна категория из базы данных выводим на экран вертикальную черту, которая будет отделять "На главную", от следующей категории. Далее инициализируем счетчик, он необходим для вывода вертикальных чёточек после каждого элемента меню, кроме последнего. Далее, проходим по всем категориям, полученным из базы данных. Берем поле `url` категории и ищем, есть ли такая подстрока в URL-запросе. Если строка найдена, то текущий элемент меню необходимо сделать активным.

Посетив сейчас главную страницу сайта, вы не заметите разницы, но перейдите в категорию телефонов, и вы заметите, что остался только один пункт меню - На главную. Это произошло по тому, что вы используете маршрут, предназначенный для отображения категорий, т.е. используется контроллер `category`, а в нем мы еще не используем модель `init`.

Этап 8 Создание и редактирование категорий и товаров в корзине

На данном этапе у Вас есть все инструменты для работы: это и роутер, контролеры и виды. Самостоятельно создайте функционал работы с категориями и товарами в административной части сайта.

Администратор сайта должен иметь возможность создавать категории, редактировать их название, скрывать или показывать на сайте, удалять.

Так же в админке должно быть создание новых товаров. Администратор должен иметь возможность загрузить изображение для товара, ввести его название, описание, определить в какой категории этот товар размещен, показывать ли этот товар на главной странице.

Так же товары можно редактировать – менять все их параметры.

Этап 9 Вывести все категории и товары в публичной части сайта

После того как Вы убедились, что предыдущий этап выполнен, создайте несколько товаров и категорий. Выведите их в публичной части сайта т. е. Вам нужно создать каталог товаров.

Создайте функционал страницы поиска по названиям товаров.

Этап 10 Добавление в корзину и работа с корзиной

Создайте функционал добавления товаров в корзину через сессию. В самой корзине добавьте возможность изменять количество товаров в позиции, удалять товар из заказа и обработку заказа.

Этап 11 Добавление в корзину и работа с корзиной

В админке выведите список заказов пользователей. Позвольте администратору изменять статусы заказов (оплачен, отклонён и пр.)

