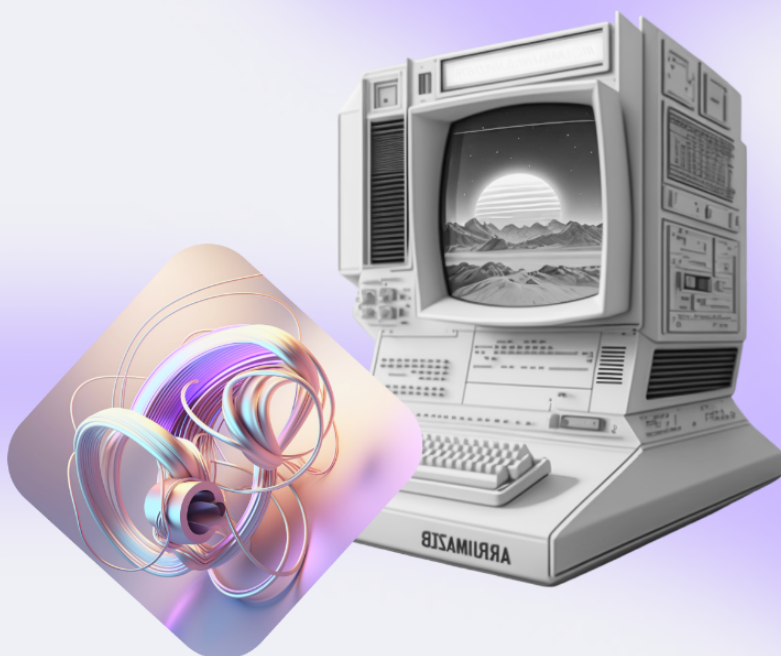


Frontend

ОСНОВЫ PHP



Оглавление

Введение	2
Публичная директория	3
Bootstrap	5
Структура страницы	6
Заголовок	7
Подвал	9
Контент	10
Асинхронные запросы	11
Реализация	12
Экономим данные	18
Заключение	22
Домашнее задание	22
Что можно почитать еще?	22

Введение

Мы вплотную подошли к завершению первого блока нашего курса. И сейчас самое время, чтобы уделить внимание организации того, без чего веб-приложения практически не существуют – интерфейсу. Иными словами, в этой лекции мы поговорим об:

- организации верстки в нашем приложении
- аспектах реализации интерфейса

Перед тем, как начинать непосредственно организацию более сложного фронтэнда, нам нужно познакомиться с одной несложной, но важной концепцией – публичной директорией. Она является корневой директорией, доступной публично через

веб-сервер и служит для хранения файлов, которые могут быть непосредственно доступны клиентам (пользователям) в браузере.

Это означает, что все файлы, расположенные в директории "public", могут быть доступны по URL-адресу веб-сайта. Это включает в себя файлы HTML, CSS, JavaScript, изображения, видео и другие статические ресурсы. Если файлы не находятся в публичной директории, они обычно не будут доступны клиентам.

Публичная директория

Точка входа в наше приложение располагается в файле index.php. Этот файл, расположенный в директории с кодом, крайне важен для нас. Но с точки зрения безопасности здесь мы имеем потенциальную уязвимость.

Конечно, наш движок устроен таким образом, что все запросы к php-файлам перенаправляются на index.php. Но, например, злоумышленник может получить прямой доступ к статическому файлу composer.json <http://mysite.local/composer.json>

Он может увидеть, какие библиотеки использует наше приложение, изучить их известные уязвимости и произвести атаку через них. В своё время таким образом была проведена атака на Java приложения через библиотеку Log4j, которая позволила взломщикам запускать неавторизованный удаленный код, просто отправив команду на логирование определенной строки.



Конечно, от уязвимостей самих библиотек мы можем избавиться только через обновление (при наличии таковых) или отказ от библиотек. Но мы можем усложнить получение доступа к информации о библиотеках.

Распространенной практикой тут является размещение файла index.php в отдельной директории public на одном уровне с src. В таком случае все остальные файлы остаются уровнем выше, а в URL никак не получится подняться уровнем выше в иерархии директорий.

При переносе сам index.php нужно будет немного отредактировать. В нем мы подключаем автозагрузчик, который теперь находится уровнем выше:

```
require_once('../vendor/autoload.php');
```

Также нужно будет поменять настройки Nginx, ведь он теперь должен смотреть в другую (более глубокую) директорию.

Изменение нужно будет сделать в `mysite.local.conf`, перезапустив контейнер:

```
root /data/mysite.local/public;
```

С изменением этого параметра у нас изменится значение в суперглобальном массиве:

```
$_SERVER['DOCUMENT_ROOT']
```

Поэтому нам нужно встроить «подъём» на уровень выше во всех местах вхождения.

Например, в классе `Render`:

```
$this->loader = new FilesystemLoader($_SERVER['DOCUMENT_ROOT'] . '/../' .  
$this->viewFolder);
```

Изменения надо будет сделать в:

- `Render`
- `Config`

Проделав эти несложные изменения, мы не только обезопасили наше приложение, но и обеспечили очень важный фундамент для размещения статических файлов.

Теперь все CSS и JS файлы, а также изображения мы можем размещать именно в этой директории. Поэтому создадим соответствующие поддиректории

- `css` – для файлов стилей
- `js` – для js-скриптов
- `img` – для картинок

Также создадим файлы по умолчанию:

- `main.css` – для хранения стилей
- `main.js` – для размещения скриптов

И теперь нам надо подключить их к приложению, что мы сделаем в файле main.tpl:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <link rel="stylesheet" href="/css/main.css">
    <script src="/js/main.js"></script>
    <title>{{ title }}</title>
  </head>
```

Bootstrap

Мы можем сверстать наше приложение с нуля, но для нас сейчас важнее backend. Поэтому для упрощения работы мы будем использовать [Bootstrap](#). Вы наверняка уже знаете о нем, но напомним, что это бесплатный CSS-фреймворк для быстрой вёрстки интерфейсов веб-приложений.



Он применяется многими разработчиками – статистика показывает, что он используется на 19% всех веб-сайтов.

Фреймворк может использоваться для верстки веб-приложений любого типа.

Фреймворк представляет собой набор CSS и JS файлов. Поэтому для того, чтобы его использовать, эти файлы необходимо подключить к странице. После этого вам станут доступны инструменты данного фреймворка: колоночная система, CSS-классы и компоненты.

Давайте подключим Bootstrap к нашему приложению.

Начнем с CSS-файла. Он располагается на сервере Bootstrap, поэтому его не нужно отдельно скачивать. Подключаем мы его строкой выше, чем main.css, чтобы иметь возможность определять свои собственные стили на странице:

```
<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-9ndCyUaIbzAi2FUVXJi0CjmCapSmO7SnpJef0486qhLnuZ2cdeRhO02iuK6FUU
VM" crossorigin="anonymous">
```

Файл JS подключается прямо перед закрывающимся тегом body. Это делается затем, чтобы к моменту включения скриптов Bootstrap вся DOM-структура страницы гарантированно скачалась с сервера и загрузилась:

```
<script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"
integrity="sha384-geWF76RCwLtnZ8gwWowPQNguL3RmwHVBC9FhGdlKrxdiJJigb/j/68SIy3Te4Bkz" crossorigin="anonymous"></script>
```

Если мы все сделали правильно, то увидим, что внешний вид нашей главной страницы уже чуть изменился.

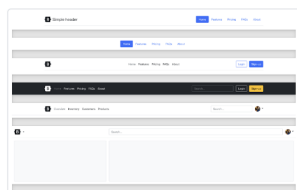
Структура страницы

Теперь мы можем сформировать структуру или скелет страницы, вокруг которой будет строиться размещение различных блоков нашего приложения. Благо, Bootstrap предлагает для этого множество встроенных инструментов.

Примеры типовых разметок всегда можно посмотреть на специальной [странице](#) сайта фреймворка

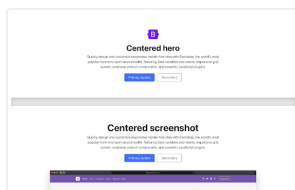
Фрагменты

Общие шаблоны для создания сайтов и приложений, основанных на существующих компонентах и утилитах с настраиваемым CSS и т.д.



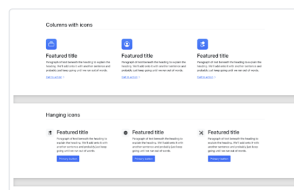
Заголовки

Отобразите свой брендинг, навигацию, поиск и многое другое с помощью этих компонентов заголовка.



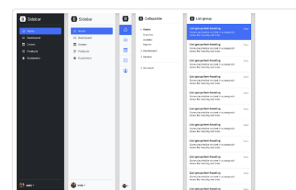
Герои

Создайте сцену на своей домашней странице с помощью героев с четкими призывами к действию.



Функции

Объясните особенности, преимущества или другие детали Вашего маркетингового контента.

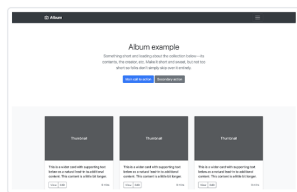


Боковые панели

Общие шаблоны навигации идеально подходят для макетов вне холста или нескольких столбцов.

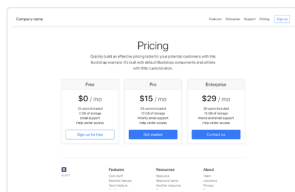
Пользовательские компоненты

Совершенно новые компоненты и шаблоны, которые помогут людям быстро приступить к работе с Bootstrap и продемонстрируют лучшие практики для добавления в платформу.



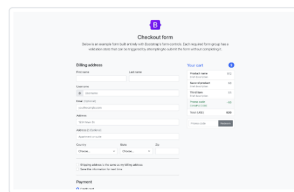
Альбом

Простой односторонний шаблон для



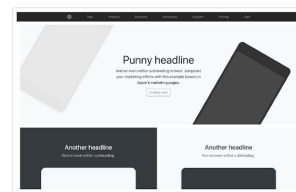
Ценообразование

Пример страницы с ценами, созданной



Оформление заказа

Пользовательская форма оформления



Продукт

Экономичная маркетинговая страница,

Мы уже можем применить к нашему скелету основные элементы:

- Заголовок
- Закрепленный футер

Начнем с заголовка. Скачать примеры можно по [ссылке](#). Далее в директории headers мы можем найти необходимые нам варианты реализации.

Заголовок

В main.tpl мы разместим следующий код:

```
<div class="container">
  <header class="d-flex flex-wrap align-items-center
justify-content-center justify-content-md-between py-3 mb-4 border-bottom">
    <a href="/" class="d-flex align-items-center col-md-3 mb-2
mb-md-0 text-dark text-decoration-none">
      <svg class="bi me-2" width="40" height="32" role="img"
aria-label="Bootstrap"><use xlink:href="#bootstrap"></use></svg>
      </a>

    <ul class="nav col-12 col-md-auto mb-2 justify-content-center
mb-md-0">
      <li><a href="#" class="nav-link px-2
link-secondary">Главная</a></li>
      <li><a href="#" class="nav-link px-2
link-dark">Пользователи</a></li>
    </ul>

    <div class="col-md-3 text-end">
      <button type="button" class="btn btn-outline-primary
me-2">Войти</button>
    </div>
  </header>
</div>
```

Основным преимуществом Bootstrap является система классов. Применяя класс к нужному элементу, мы можем получить сразу же ожидаемый вид.

Например, обычный блок:

```
<div>
  <strong>Замечание.</strong> Тут идет важный текст.
</div>
```

превратится в оформленный блок, если мы просто обозначим класс у div-a:

```
<div class="alert alert-dark">
  <strong>Замечание.</strong> Тут идет важный текст.
</div>
```

С подробной документацией можно ознакомиться по [ссылке](#).

Еще одним отличным инструментом является колоночная сетка.



Колоночная система Bootstrap основана на концепции 12 колонок. Каждая строка экрана делится на 12 равных колонок, и вы можете комбинировать эти колонки, чтобы создать различные макеты.

Для того, чтобы применять сетку, нужно иметь правильное именование блоков. Все начинается с блока-обертки с классом `container`, в котором будут применяться нужные нам стили. Ключевым элементом верстки в Bootstrap является колоночная система – она позволяет размещать содержимое на странице в виде гибкой сетки. С помощью этой системы вы можете создавать отзывчивые и адаптивные макеты для различных устройств и размеров экранов.

Пример использования колоночной системы выглядит следующим образом:

```
<div class="container">
  <div class="row">
    <div class="col-md-6">
      <!-- Содержимое колонки 1 -->
    </div>
    <div class="col-md-6">
      <!-- Содержимое колонки 2 -->
    </div>
  </div>
</div>
```

В приведенном выше примере у нас есть контейнер (`.container`), в котором располагается строка (`.row`). Внутри строки мы создаем две колонки (`.col-md-6`), каждая из которых занимает половину ширины родительской строки.

Классы `col-md-6` указывают на то, что колонка будет занимать 6 из 12 доступных колонок (половину ширины). Кроме того, модификатор `md` в имени класса определяет средний размер экрана, есть и другие классы размеров, такие как `sm` (маленький), `lg` (большой) и `xl` (очень большой), которые позволяют настраивать отзывчивость колонок под различные устройства.

В нашем случае мы указываем для блока навигации класс `col-md-auto`, который будет автоматически задавать ширину в зависимости от наполнения страницы.

Классов в Bootstrap огромное количество – обязательно ознакомьтесь с ними в [официальной документации](#).

Но теперь у нас осталось две ссылки на вход, поэтому нам нужно перенести кнопку «Войти» в файл `auth-template.tpl`. Также кнопку надо переделать в ссылку.

Добавим в наше навигационное меню нужные нам ссылки.

Итак, наш код шапки в main.tpl примет вид:

```
<div class="container">
  <header class="d-flex flex-wrap align-items-center
justify-content-center justify-content-md-between py-3 mb-4 border-bottom">
    <a href="/" class="d-flex align-items-center col-md-3 mb-2
mb-md-0 text-dark text-decoration-none">
      <svg class="bi me-2" width="40" height="32" role="img"
aria-label="Bootstrap"><use xlink:href="#bootstrap"></use></svg>
    </a>

    <ul class="nav col-12 col-md-auto mb-2 justify-content-center
mb-md-0">
      <li><a href="/" class="nav-link px-2
link-secondary">Главная</a></li>
      <li><a href="/user/index/" class="nav-link px-2
link-dark">Пользователи</a></li>
    </ul>

    {% include "auth-template.tpl" %}
  </header>
</div>
```

В auth-template код будет следующим:

```
{% if not user_authorized %}
  <div class="col-md-3 text-end">
    <a href="/user/login/" class="btn btn-primary">Войти</a>
  </div>
{% else %}
  <p>Добро пожаловать на сайт!</p>
{% endif %}
```

Подвал

В нижней части нашего сайта мы разместим подвал, который будет содержать в себе справочную информацию.

Подвал при этом должен быть всегда «приклеен» к нижней части нашего экрана.

Для этого сделаем следующие шаги:

1. Назначим класс h-100, который говорит о том, что контент страницы всегда растягивается на всю высоту для тега html
2. Для тега body назначим классы h-100, d-flex и flex-column
3. Обернем контентную часть в тег main

```
<main class="flex-shrink-0">
  <div class="container">
    {% include content_template_name %}
  </div>
</main>
```

Теперь уже добавляем непосредственно наш footer:

```
<footer class="footer mt-auto py-3 bg-light">
  <div class="container">
    <span class="text-muted">Место для контента прикрепленного футера
здесь.</span>
  </div>
</footer>
```

Контент

Для закрепления полученных знаний теперь мы можем сделать красивым наш список пользователей. Превратим его в таблицу.

В Bootstrap предоставляется встроенный инструмент формирования таблиц. Разумеется, внутри него будет все та же знакомая вам HTML-таблица (table). Но мы обернем её в блок с классами table-responsive и small.

Класс table-responsive сделает размеры таблицы гибкими – если вы будете менять размеры экрана, таблица будет перестраиваться так, чтобы гармонично помещаться.

Класс small говорит о размерах шрифтов и ячеек в таблице:

```
<div class="table-responsive small">
  <table class="table table-striped table-sm">
    <thead>
      <tr>
        <th scope="col">Имя</th>
        <th scope="col">Фамилия</th>
        <th scope="col">День рождения</th>
      </tr>
    </thead>
  </table>
```

Пока мы создали только заголовок. Затем следует тело таблицы.

Строки в нем будут генерироваться циклом по простому правилу: 1 пользователь – 1 строка:

```
<tbody>
  {% for user in users %}
    <tr>
      <td>{{ user.getUserName() }}</td>
      <td>{{ user.getUserLastName() }}</td>
      <td>{% if user.getUserBirthday() is not empty %}
        {{ user.getUserBirthday() | date('d.m.Y') }}
      {% else %}
        <b>Не задан</b>
      {% endif %}
    </td>
  </tr>
{% endfor %}
</tbody>
```

Теперь мы получили четко читаемый формат таблицы с данными о наших пользователях.

Асинхронные запросы

Крайне важной для удобства работы пользователя функциональностью вашего приложения является возможность отправлять асинхронные запросы. До текущего момента мы работали исключительно с синхронным взаимодействием. Это означает, что данные на странице загружаются один раз при обращении к серверу по URI, и до перезагрузки или перехода между страницами никак не будут обновлены.



Асинхронные запросы в вебе относятся к методу обмена данными между клиентом (обычно браузером) и сервером, при котором клиент может отправлять запросы на сервер и получать ответы без блокировки основного потока выполнения.

Вместо того, чтобы ждать ответа синхронно и блокировать пользовательский интерфейс, асинхронные запросы позволяют клиенту продолжать выполнять другие задачи, пока запрос обрабатывается на сервере.

Асинхронные запросы осуществляются с помощью технологии **AJAX** (Asynchronous JavaScript and XML) или с использованием более новых технологий, таких как Fetch API или XMLHttpRequest. При выполнении асинхронного запроса клиент отправляет запрос на сервер, продолжая выполнять свою работу, не ожидая ответа. Когда сервер завершает обработку запроса, он отправляет ответ обратно клиенту, и клиент выполняет соответствующие действия на основе полученных данных.

🔥 Асинхронные запросы веб-страниц позволяют создавать более отзывчивые и интерактивные пользовательские интерфейсы, поскольку клиент может отправлять запросы и обновлять содержимое страницы без необходимости полной перезагрузки страницы.

Это особенно полезно для динамических элементов страницы, таких как обновление списка задач, загрузка дополнительного контента или отправка данных на сервер без перезагрузки страницы.

Примеры асинхронных запросов включают отправку формы без перезагрузки страницы, загрузку данных из базы данных или API без блокировки интерфейса и динамическую загрузку контента по мере прокрутки страницы.

Реализация

Когда мы создаём нового пользователя, список существующих пользователей никак не обновляется. Мы узнаем о том, что был добавлен новый пользователь только при перезагрузке страницы. Но это неудобно. А значит, нам надо обеспечить возможность обновления таблицы пользователей в реальном времени.

С точки зрения нашей системы это означает, что мы должны будем обновить нашу таблицу по некоему событию. Проще всего сделать это по интервалу времени, например, в 10 секунд.

То есть, после загрузки пользователем страница раз в 10 секунд отправляет асинхронный запрос на обновление данных. С точки зрения JavaScript мы можем сделать наш код максимально простым и лаконичным, если подключим библиотеку jQuery. То же самое можно сделать и на нативном JavaScript, но для нас принципиально обеспечить работу backend части.

Обратите внимание на то, что Bootstrap 5, используемый нами корректно работает и без jQuery, и с нативным JS. Поэтому здесь окончательный выбор остаётся за вами, но лежит за пределами курса PHP, так как серверу безразлично, каким именно образом JavaScript формирует к нему запрос.

Для подключения библиотеки в блоке head шаблона main.tpl мы вставим строку:

```
<script
  src="https://code.jquery.com/jquery-3.7.0.min.js"
  integrity="sha256-2Pmvv0kuTBOenSvLm6bvfbSSHrUU+3A7x6P5Ebd07/g="
  crossorigin="anonymous"></script>
```

Сам же код будет выглядеть следующим образом:

```
<script>
  setInterval(function () {
    $.ajax({
      method: 'POST',
      url: "/user/indexRefresh/"
    }).done(function (data) {
      let users = $.parseJSON(response);

      if(users.length != 0){
        for(var k in users){

          let row = "<tr>";

          row += "<td>" + users[k].id + "</td>";
          maxId = users[k].id;


          row += "<td>" + users[k].username + "</td>";
          row += "<td>" + users[k].userlastname + "</td>";
          row += "<td>" + users[k].userbirthday + "</td>";

          row += "</tr>";

          $('<div>content-template tbody').append(row);
        }
      }
    });
  }, 10000);
</script>
```

Его мы разместим в шаблон user-index.tpl.

Данный код на JavaScript устанавливает интервал, используя функцию `setInterval`, которая выполняет определенный блок кода с определенным интервалом времени. В данном случае, код внутри `setInterval` будет выполняться каждые 10 секунд (10000 миллисекунд).

 Внутри функции `setInterval` есть вызов метода `$.ajax()`, который выполняет асинхронный HTTP-запрос на сервер. Параметры метода `$.ajax()` определяют метод запроса (`method: 'POST'`) и URL-адрес запроса (`url: "/user/indexRefresh/"`).

Когда сервер успешно обработает этот запрос, функция `done()` будет вызвана с полученными данными в качестве аргумента (`function (data)`). В данном случае, эти данные будут использованы для обновления содержимого элемента с классом `content-template`. Метод `html()` используется для замены HTML-содержимого элемента на полученные данные.

Таким образом, данный код устанавливает периодический запрос на сервер каждые 10 секунд и обновляет содержимое элемента с классом content-template веб-страницы с данными, возвращенными сервером в ответ на этот запрос.

Исходя из этого, нам надо произвести ещё одну доработку шаблона main.tpl. Там мы блоку container добавим ещё один класс – content-template. Именно по нему JS будет находить на странице часть, которую нужно обновить.

Теперь нам надо описать в контроллере UserController метод actionIndexRefresh, который будет формировать обновлённое содержимое. Мы будем отдавать HTML-код для простоты:

```
public function actionIndexRefresh(){
    $users = User::getAllUsersFromStorage();

    $render = new Render();

    if(!$users){
        return $render->renderPartial(
            'user-empty.tpl',
            [
                'title' => 'Список пользователей в хранилище',
                'message' => "Список пуст или не найден"
            ]
        );
    }
    else{
        return $render->renderPartial(
            'user-index.tpl',
            [
                'title' => 'Список пользователей в хранилище',
                'users' => $users
            ]
        );
    }
}
```

Как видите, здесь происходит вызов нового метода renderPartial. Дело тут в том, что уже существующий метод генерирует для нас всю страницу целиком – с шапкой, подвалом и всеми остальными элементами.

Но нам нужно получить исключительно контентную часть. То есть, сгенерировать то, что у нас передаётся в параметр contentTypeName метода renderPage.

Поэтому нам и нужен отдельный метод:

```
public function renderPartial(string $contentTemplateName, array
$templateVariables = []): string {
    $template = $this->environment->load($contentTemplateName);

    if(isset($_SESSION['user_name'])) {
        $templateVariables['user_authorized'] = true;
    }

    return $template->render($templateVariables);
}
```

В отличие от `renderPage` мы сюда передаём нужное имя шаблона контента. При этом параметр уже не имеет значения по умолчанию, так как это бессмысленно.

Внутри мы оставляем флаг аутентификации нашего пользователя, так как это может влиять на отображение контентной части.

Но в итоге все также вызываем метод `render`.

Также нам нужно будет немного доработать модель. В прошлых занятиях мы создали поле `login` в таблице `users` нашей базы данных. Оно не может принимать значение `null`, поэтому пользователю надо дать возможность заполнить его.

Добавим его в форму. А также мы добавим возможность задавать пароль:

```
<p>
    <label for="user-login">Логин:</label>
    <input id="user-login" type="text" name="login">
</p>
<p>
    <label for="user-password">Пароль:</label>
    <input id="user-password" type="text" name="password">
</p>
```

Новые поля нам нужно учесть в модели и валидации.

В модели `User` добавим два новых поля:

```
private ?string $userLogin;
private ?string $userPassword;
```

Теперь добавим валидацию этих полей в метод `validateRequestData`:

```
if(! (
    isset($_POST['name']) && !empty($_POST['name']) &&
    isset($_POST['lastname']) && !empty($_POST['lastname']) &&
    isset($_POST['birthday']) && !empty($_POST['birthday']) &&
    isset($_POST['login']) && !empty($_POST['login']) &&
    isset($_POST['password']) && !empty($_POST['password'])
)){
    $result = false;
}
```

Еще один метод, который надо обновить, это метод задания свойств объекта из параметров запроса:

```
public function setParamsFromRequestData(): void {
    $this->userName = htmlspecialchars($_POST['name']);
    $this->userLastName = htmlspecialchars($_POST['lastname']);
    $this->setBirthdayFromString($_POST['birthday']);
    $this->userLogin = htmlspecialchars($_POST['login']);
    $this->userPassword = Auth::getPasswordHash($_POST['password']);
}
```

В случае, если у нас в наличии все поля, нам надо обновить логику записи в БД:

```
public function saveToStorage(){
    $sql = "INSERT INTO users(user_name, user_lastname,
user_birthday_timestamp, `login`, password_hash) VALUES (:user_name,
:user_lastname, :user_birthday, :user_login, :user_password)";

    $handler = Application::$storage->get()->prepare($sql);
    $handler->execute([
        'user_name' => $this->userName,
        'user_lastname' => $this->userLastName,
        'user_birthday' => $this->userBirthday,
        'user_login' => $this->userLogin,
        'user_password' => $this->userPassword
    ]);
}
```

Обратите внимание на то, как мы обозначаем в запросе поле `login`. Оно неспроста обрамлено в обратные кавычки.



В MySQL обратные кавычки (`) используются для обозначения имен объектов базы данных, таких как таблицы, столбцы, индексы, представления и др., которые содержат специальные символы или совпадают с зарезервированными словами MySQL.

Использование обратных кавычек позволяет MySQL интерпретировать имя объекта как литерал, игнорируя возможные конфликты с ключевыми словами или специальными символами. Например, если вы хотите создать таблицу с именем "order", которое является зарезервированным словом в MySQL, вы можете обернуть это имя в обратные кавычки:

```
CREATE TABLE `order` (  
  `id` INT,  
  `name` VARCHAR(255)  
);
```

В этом примере обратные кавычки позволяют использовать "order" в качестве имени таблицы, несмотря на то, что оно является ключевым словом.

Также обратные кавычки полезны, когда имя объекта содержит специальные символы или пробелы. Например:

```
CREATE TABLE `my-table` (  
  `id` INT,  
  `column name` VARCHAR(255)  
);
```

Обратные кавычки позволяют использовать имена типа "my-table" и "column name", содержащие пробелы или другие символы, в качестве имен таблицы и столбца соответственно.

Однако в большинстве случаев использование обратных кавычек не является обязательным и рекомендуется избегать использования имен объектов, которые требуют обратных кавычек. Это помогает упростить синтаксис SQL и избежать возможных проблем при работе с базой данных.

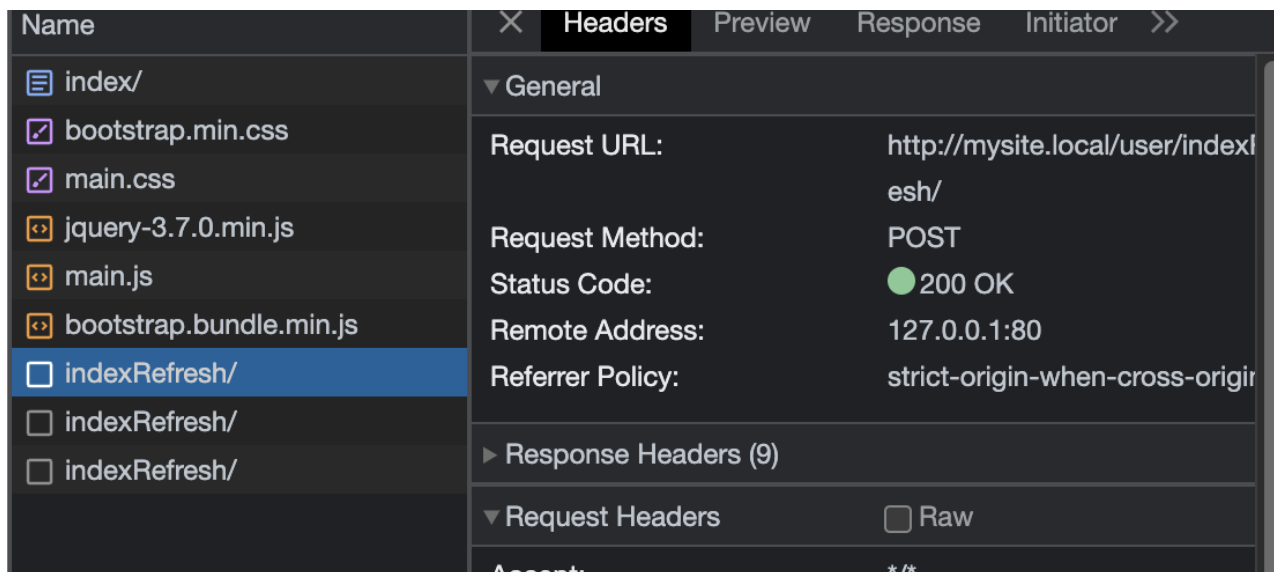
Теперь наш метод полностью готов. При создании нового пользователя таблица будет обновляться раз в 10 секунд. Подтверждение этому мы видим в инспекторе кода нашего браузера.



Важно здесь отметить, что в данном примере мы отдаём html-код целиком, при этом достаточно большой частью. Это **не очень хорошо** с точки зрения загрузки.

В идеальной ситуации нужно сокращать подобного рода обращения к серверу, а также передаваемые данные.

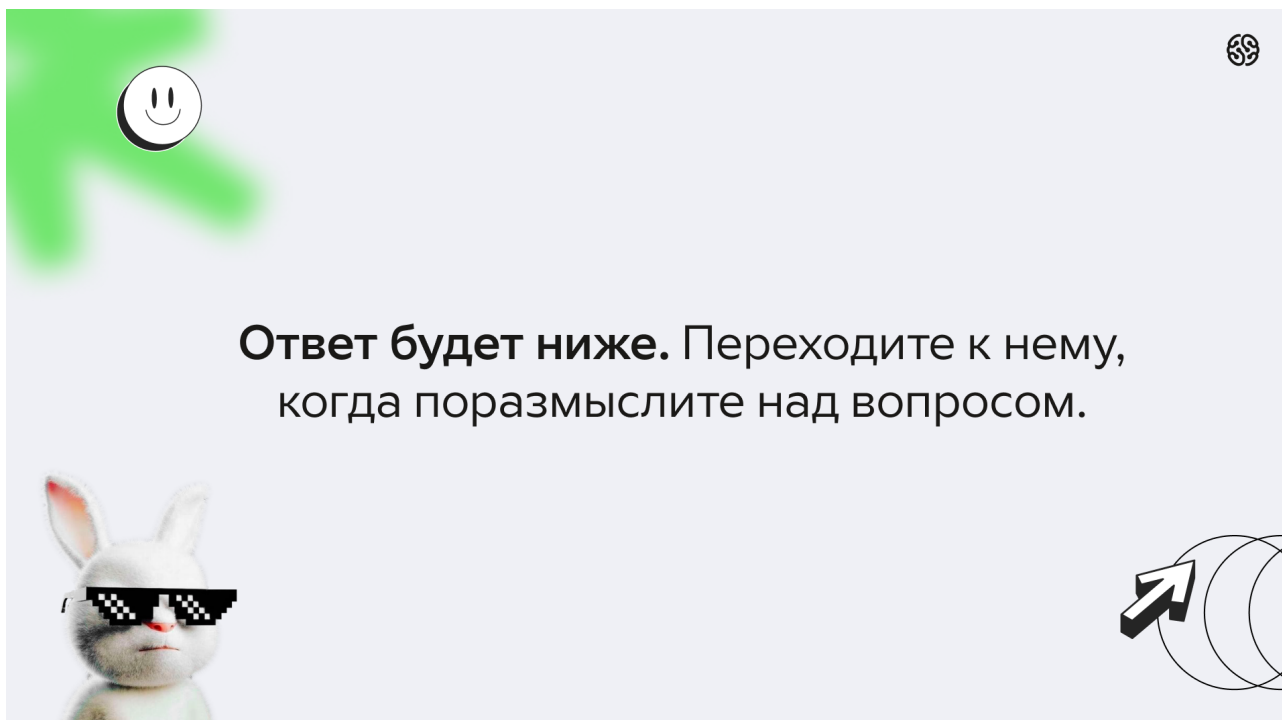
Например, мы можем сократить объём, передавая не всю таблицу целиком, а только обновлённую запись. В таком случае надо будет не обновлять страницу, а всего лишь добавлять данные в самый её конец.



Экономим данные

Итак, нам надо переделать нашу систему с возвращения всего подряд на возвращение только обновленных данных.

Но как мы поймем, какого пользователя добавили?



Мы можем посмотреть, какой ID был последним!

Ведь они увеличиваются с течением времени – это свойство автоинкремента. Если на странице последний ID был 19, то все пользователи с ID больше 19 априори созданы позже времени загрузки страницы.

Но у нас пока нет ID на странице. Давайте добавим его в вывод. Начнем с шаблона:

```
<div class="table-responsive small">
  <table class="table table-striped table-sm">
    <thead>
      <tr>
        <th scope="col">ID</th>
        <th scope="col">Имя</th>
        <th scope="col">Фамилия</th>
        <th scope="col">День рождения</th>
      </tr>
    </thead>
    <tbody>
      {% for user in users %}
      <tr>
        <td>{{ user.getUserId() }}</td>
        <td>{{ user.getUserName() }}</td>
        <td>{{ user.getUserLastName() }}</td>
        <td>{% if user.getUserBirthday() is not empty %}
          {{ user.getUserBirthday() | date('d.m.Y') }}
        {% else %}
          <b>Не задан</b>
        {% endif %}
        </td>
      </tr>
      {% endfor %}
    </tbody>
  </table>
</div>
```

Очевидно, что в классе User надо добавить ID как свойство и проработать его во все методы:

```
private ?int $userId;
public function __construct(int $id = null, string $name = null, string
$lastName = null, int $birthday = null){
    $this->userId = $id;
    $this->userName = $name;
    $this->userLastName = $lastName;
    $this->userBirthday = $birthday;
}
public function getUserId(): ?int {
    return $this->userId;
}
```

Также в цикле внутри метода `getAllUsersFromStorage` добавим `id` в создание коллекции пользователей:

```
foreach($result as $item){
    $user = new User($item['id_user'], $item['user_name'],
    $item['user_lastname'], $item['user_birthday_timestamp']);
    $users[] = $user;
}
```

Теперь, когда мы уже выводим на экран ID пользователей, вычислить максимальный ID вполне несложно – это ID из последней строки таблицы. То есть, надо взять первую ячейку из последней строки таблицы. На JQuery это можно описать вот так:

```
let maxId = $('<table-responsive tbody tr:last-child td:first-child>').html();
```

Теперь мы можем передавать наш максимальный ID в запросе:

```
setInterval(function () {
    $.ajax({
        method: 'POST',
        url: "/user/indexRefresh/",
        data: { maxId: maxId }
    }).done(function (response) {
        // ...
    })
});
```

Переделаем наш метод в контроллере так, чтобы у нас возвращался не HTML-код, а JSON-коллекция:

```
public function actionIndexRefresh(){
    $limit = null;

    if(isset($_POST['maxId']) && ($_POST['maxId'] > 0)){
        $limit = $_POST['maxId'];
    }

    $users = User::getAllUsersFromStorage($limit);
    $usersData = [];

    if(count($users) > 0) {
        foreach($users as $user){
            $usersData[] = $user->getUserDataAsArray();
        }
    }

    return json_encode($usersData);
}
```

Теперь мы понимаем – запрашивают ли у нас все подряд или только часть данных. Как видите, теперь нам надо передавать ID в метод выборки пользователей, чтобы не выбирать все данные, а возвращать только обновленные:

```
public static function getAllUsersFromStorage(?int $limit = null): array {
    $sql = "SELECT * FROM users";

    if(isset($limit) && $limit > 0) {
        $sql .= " WHERE id_user > " .(int)$limit;
    }
    // ...
}
```

Мы добавляем необязательный параметр фильтра. Если он задан, то в запрос выборки в методе `getAllUsersFromStorage` будет добавляться WHERE-условие, обеспечивающее возврат только пользователей с ID, который больше переданного.

Теперь мы возвращаемся в шаблон. Скрипт в нем будет выглядеть на данный момент следующим образом:

```
<script>
    let maxId = $('<table>.table-responsive tbody tr:last-child td:first-child').html();

    setInterval(function () {
        $.ajax({
            method: 'POST',
            url: "/user/indexRefresh/",
            data: { maxId: maxId }
        }).done(function (response) {
            let users = $.parseJSON(response);

            if(users.length != 0){
                for(var k in users){

                    let row = "<tr>";

                    row += "<td>" + users[k].id + "</td>";
                    maxId = users[k].id;

                    row += "<td>" + users[k].username + "</td>";
                    row += "<td>" + users[k].userlastname + "</td>";
                    row += "<td>" + users[k].userbirthday + "</td>";

                    row += "</tr>";

                    $('<div>.content-template tbody').append(row);
                }
            }
        });
    }, 10000);
</script>
```

Теперь мы явно указываем, что нам приходит не строка, а JSON-объект. Далее мы обходим его при помощи цикла и создаём новую строку в таблице.

Обратите также внимание на то, что `maxID` надо также обновлять, потому что иначе запрос будет возвращать одни и те же данные.

Произведя подобные изменения, мы значительно сократили нагрузку на сеть и сервер, сделав наше приложение более легковесным!

Заключение

В рамках данной лекции мы научились организовывать Frontend составляющую нашего сайта, а также познакомились с асинхронными запросами к серверу.

На этом первый блок нашего курса завершается, а мы с вами получили работающее приложение, которое можно адаптировать к своим задачам.

Домашнее задание

Скорректируйте список пользователей так, чтобы все пользователи с правами администратора в таблице видели две дополнительные ссылки – редактирование и удаление пользователя. При этом редактирование будет переходить на форму, а удаление в асинхронном режиме будет удалять пользователя как из таблицы, так и из БД.

Что можно почитать еще?

1. <https://phptherightway.com/>
2. <https://php.net>
3. PHP 8 | Котеров Дмитрий Владимирович
4. Объектно-ориентированное мышление | Мэтт Вайсфельд
5. <https://stackoverflow.com/questions/34034730/how-to-enable-color-for-php-cli> - как покрасить вывод консоли
6. <https://phptoday.ru/post/gotovim-lokalnuyu-sredu-docker-dlya-razrabotki-na-php> - варианты образа Composer
7. <https://anton-pribora.ru/articles/php/mvc/>