Приднестровский государственный университет им. Т.Г. Шевченко
Инженерно-технический институт

РАЗРАБОТКА АДАПТИВНОЙ ВЁРСТКИ ВЕБ-САЙТА ПРИ ПОМОЩИ ТЕХНОЛОГИЙ HTML5 И CSS3

Лабораторный практикум

Разработал:
ст. преподаватель
кафедры ИТиАУПП
Бричаг Д.В.

г. Тирасполь
2021 г.

Лабораторная работа №5

Подключение и работа с библиотекой jQuery 3.5 для веб-страниц

Цель работы: ознакомиться с подключением и принципами работы библиотек для JavaScript. Применить функции jQuery для работы с элементами DOM. Познакомиться с типовыми методами библиотеки и применить их на практике.

Теоретическая справка

Изначально JavaScript был создан, чтобы «сделать веб-страницы живыми». Программы на этом языке называются скриптами. Они могут встраиваться в HTML и выполняться автоматически при загрузке веб-страницы. Скрипты распространяются и выполняются, как простой текст. Им не нужна специальная подготовка или компиляция для запуска.

Это отличает JavaScript от другого языка – Java. Но почему JavaScript?

Когда JavaScript создавался, у него было другое имя – «LiveScript». Однако, язык Java был очень популярен в то время, и было решено, что позиционирование JavaScript как «младшего брата» Java будет полезно.

Со временем JavaScript стал полностью независимым языком со своей собственной спецификацией, называющейся ECMAScript, и сейчас не имеет никакого отношения к Java.

Сегодня JavaScript может выполняться не только в браузере, но и на сервере или на любом другом устройстве, которое имеет специальную программу, называющуюся «движком» JavaScript.

У браузера есть собственный движок, который иногда называют «виртуальная машина JavaScript».

Разные движки имеют разные «кодовые имена». Например:

1. V8 – в Chrome и Opera.
2. SpiderMonkey – в Firefox.

3. Ещё есть «Trident» и «Chakra» для разных версий IE, «ChakraCore» для Microsoft Edge, «Nitro» и «SquirrelFish» для Safari и т.д.

Эти названия полезно знать, так как они часто используются в статьях для разработчиков. Основные тезисы движков:

- Движок (встроенный, если это браузер) читает («парсит») текст скрипта.
- Затем он преобразует («компилирует») скрипт в машинный язык.
- После этого машинный код запускается и работает достаточно быстро.

Движок применяет оптимизации на каждом этапе. Он даже просматривает скомпилированный скрипт во время его работы, анализируя проходящие через него данные, и применяет оптимизации к машинному коду, полагаясь на полученные знания. В результате скрипты работают очень быстро.

Что может JavaScript в браузере

Современный JavaScript – это «безопасный» язык программирования. Он не предоставляет низкоуровневый доступ к памяти или процессору, потому что изначально был создан для браузеров, не требующих этого.

Возможности JavaScript сильно зависят от окружения, в котором он работает. Например, Node.JS поддерживает функции чтения/записи произвольных файлов, выполнения сетевых запросов и т.д.

В браузере для JavaScript доступно всё, что связано с манипулированием веб-страницами, взаимодействием с пользователем и веб-сервером.

Например, в браузере JavaScript может:

- Добавлять новый HTML-код на страницу, изменять существующее содержимое, модифицировать стили.
- Реагировать на действия пользователя, щелчки мыши, перемещения указателя, нажатия клавиш.
- Отправлять сетевые запросы на удалённые сервера, скачивать и загружать файлы (технологии AJAX и COMET).

- Получать и устанавливать куки, задавать вопросы посетителю, показывать сообщения.
- Запоминать данные на стороне клиента («local storage»).

Что не может JavaScript в браузере

Возможности JavaScript в браузере ограничены ради безопасности пользователя. Цель заключается в предотвращении доступа недобросовестной веб-страницы к личной информации или нанесения ущерба данным пользователя.

Примеры таких ограничений включают в себя:

- JavaScript на веб-странице не может читать/записывать произвольные файлы на жёстком диске, копировать их или запускать программы. Он не имеет прямого доступа к системным функциям ОС.

Современные браузеры позволяют ему работать с файлами, но с ограниченным доступом, и предоставляют его, только если пользователь выполняет определённые действия, такие как «перетаскивание» файла в окно браузера или его выбор с помощью тега <input>.

Существуют способы взаимодействия с камерой/микрофоном и другими устройствами, но они требуют явного разрешения пользователя. Таким образом, страница с поддержкой JavaScript не может незаметно включить веб-камеру, наблюдать за происходящим и отправлять информацию в ФСБ.

- Различные окна/вкладки не знают друг о друге. Иногда одно окно, используя JavaScript, открывает другое окно. Но даже в этом случае JavaScript с одной страницы не имеет доступа к другой, если они пришли с разных сайтов (с другого домена, протокола или порта).

Это называется «Политика одинакового источника» (Same Origin Policy). Чтобы обойти это ограничение, обе страницы должны согласиться с этим и содержать JavaScript-код, который специальным образом обменивается данными.

Это ограничение необходимо, опять же, для безопасности пользователя. Страница https://anysite.com, которую открыл пользователь, не должна иметь доступ к другой вкладке браузера с URL https://gmail.com и воровать информацию оттуда.

- JavaScript может легко взаимодействовать с сервером, с которого пришла текущая страница. Но его способность получать данные с других сайтов/доменов ограничена. Хотя это возможно в принципе, для чего требуется явное согласие (выраженное в заголовках HTTP) с удалённой стороной. Опять же, это ограничение безопасности.

Подобные ограничения не действуют, если JavaScript используется вне браузера, например — на сервере. Современные браузеры предоставляют плагины/расширения, с помощью которых можно запрашивать дополнительные разрешения.

Что Такое jQuery

jQuery это популярная библиотека JavaScript. Она была создана Джоном Резигом в 2006 году с целью облегчить разработчикам использование JavaScript на веб-сайтах. Это не отдельный язык программирования и работает в сочетании с JavaScript. С jQuery вы будете делать намного больше с меньшими затратами — позвольте нам объяснить, что такое jQuery более детально.

Написание кода может стать утомительным, особенно если в него включено много строк. jQuery сжимает несколько строк кода в одну функцию, поэтому вам не нужно переписывать целые блоки кода для выполнения одной задачи.

Особенность того, почему jQuery стала настолько успешной и популярной, это, вероятно, кроссплатформенные возможности. Она автоматически исправляет ошибки и работает таким же образом в наиболее часто используемых браузерах, таких как Chrome, Firefox, Safari, MS Edge, IE, Android и iOS.

JQuery также делает Ajax намного проще. Ajax работает асинхронно с остальной частью кода. Это означает, что код, написанный на Ajax, может

взаимодействовать с сервером и обновлять его содержимое без необходимости перезагрузки страницы.

Однако это связано с проблемами. Различные браузеры выполняют Ajax API по-разному. Таким образом, код должен соответствовать всем браузерам. Вручную, это тяжелая и трудоёмкая работа. К счастью, jQuery выполняет всю тяжелую работу и адаптирует код для всех веб-браузеров.

Затем есть манипулирование DOM (Document Object Model) (англ), в котором есть несколько методов, как это сделать. Проще говоря, он позволяет вставлять и/или удалять элементы DOM на HTML-странице, а также упрощает перенос строк. Создание анимации также упрощено с помощью jQuery. Как и в приведённом выше фрагменте кода об анимации, он покрыт несколькими строками кода, всё что вам нужно сделать, это вставить переменные.

Обход документов HTML, а также выполнение эффектов и обработка событий также улучшены с помощью jQuery.

Код на jQuery компактнее, чем на JavaScript

Сравните код на JavaScript и jQuery. Вот код на чистом JavaScript:

```
document.getElementById('id').innerHTML = 'Тише мыши, кот на крыше';
```

В этом коде происходит обращение к тегу с определённым "id": внутреннее содержание этого тега меняется на строку «Тише мыши, кот на крыше». Перепишем эту строчку, используя функции из jQuery:

```
$('#id').html('Тише мыши, кот на крыше');
```

Очевидно, код стал компактнее. Приведём ещё один пример, в котором разница в удобстве будет более очевидной. Допустим, нам надо изменить CSS свойство в нескольких тегах с одним и тем же классом. К примеру, сделать текст синим. На чистом JavaScript нам надо будет получить массив ссылок на найденные элементы с помощью метода getElementsByClassName, затем

пройтись по массиву и установить CSS свойство для каждого из найденных тегов. Выглядеть это будет так:

```
const a = document.getElementsByClassName('class');
for (let i = 0; i < a.length; i++) {
   a[i].style.color = 'blue';
}
```

Перепишем этот пример, используя функцию из jQuery:

```
$('.class').css('color', 'blue');
```

Получилось, что четыре строчки кода были заменены одной, в которой находится довольно наглядная функция изменения CSS свойства.

Селекторы jQuery

Знак доллара $ в начале строки в приведённых выше примерах — это универсальная функция jQuery. Чаще всего её используют для выбора HTML элементов, поэтому после этого знака стоят круглые скобки, в которых находится указание на селектор.

Селекторы в jQuery точно такие же, как и в CSS. Попробуем выбрать все элементы с классом "mouse" используя jQuery:

```
const a = $('.mouse');
```

Теперь выберем все с id="mouse":

```
const a = $('#mouse');
```

А теперь выберем все с классом "mouse", но являющиеся div контейнерами:

```
const a = $('div.mouse');
```

Всё обращение к элементам происходит точно так же, как и в селекторах CSS.

Цепочки методов jQuery

Большое количество методов jQuery возвращают набор элементов. Поэтому можно составлять цепочки из этих методов. Попробуем

модифицировать примеры из первого параграфа этой статьи. То есть совместим изменение цвета с изменением HTML содержания тега. Получится такой код:

```
$('.mouse').html('Тише мыши, кот на крыше').css('color', 'blue');
```

В этом примере мы выбрали все HTML теги на странице, у которых есть класс "mouse" и вставили в их содержание строчку 'Тише мыши, кот на крыше'. Затем окрасили текст в каждом из этих тегов в синий цвет.

Как и в CSS мы можем указывать цепочку селекторов из нескольких классов и тегов. Например:

1) `$('div span p').hide('slow');`

Скроет тег p внутри тега спан, расположенных внутри тега div:

```
<div>
    <p>Этот текст не будет скрыт</p>
    <span>
        <p>А этот текст будет скрыт</p>
    </span>
</div>
```

2) `$('.mouse .leg').css('color', 'blue');`

Перекрасит дочерний тег с классом .leg

```
<div class=”mouse”>
    <p class=”leg”>Этот текст будет синим</p>
    <p class=”tail”>А этот текст будет чёрным</p>
</div>
```

3) `$('.mouse.big.gray').css('color', 'gray');`

Перекрасит только тег со всеми перечисленными классами

```
<div>
    <p class=”mouse big gray”>Этот текст будет серого цвета</p>
    <p class=”mouse gray”>А этот текст будет чёрного</p>
</div>
```

Практическая часть

Мы уже создали раздел с портфолио в предыдущих работах. Часто на веб-сайтах картинки из карточек можно увеличить по клику. Как правило картинки открываются на той же вкладке, в модальном окне. То есть не происходит перехода на другую страницу или перерисовка всего содержимого. Тоже самое касается и закрытия этого окна.

Чтобы создать такую потребуется некоторое количество манипуляций с DOM деревом.

Во-первых, необходимо навесить событие щелчка левой кнопкой мыши на элемент, как это уже было реализовано с нажатием по ссылкам в меню.

Во-вторых, необходимо нарисовать новый HTML элемент, куда будет помещена картинка в увеличенном масштабе. Этот элемент будет размещаться «поверх» остальной страницы.

В-третьих, нужно передать в это модальное окно ту картинку, по которой был произведён клик.

В-четвёртых, такой элемент нужно уметь закрывать, например, по щелчку на крестик. То есть нужно создавать новый обработчик на динамический элемент.

И, наконец, в-пятых, с началом события закрытия окна, его нужно удалить из дерева.

Чтобы упростить эту задачу воспользуемся библиотекой jQuery. Как и ранее её можно подключить через ссылку на внешний ресурс или скачать в папку вашего проекта и указать локальный путь.

Для того, чтобы подключить её из интернета, напишем в head нашей страницы подключение скрипта и ссылкой укажем путь:

https://code.jquery.com/jquery-3.5.1.min.js

После чего создадим в нашем проекте в папке js (или другой, в зависимости от того, как вы её назвали) ещё один файл popup.js

```
    <!-- Подключение внешних библиотек -->
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>

    <!-- Подключаем сценарии -->
    <script src="js/script.js"></script>
    <script src="js/popup.js"></script>
</head>
```

Рис. 62 – Подключение внешней библиотеки

В этом файле мы будем писать код с использованием функций jQuery. Раньше мы инициализировали наши функции по событию window.onload. В jQuery есть аналог этого события под названием ready(), который задаётся к странице. Записывается как jQuery(document).ready(); – где document это объект самой HTML страницы.

Далее в ready() передаётся в качества аргумента анонимная функция, выполняющая дальнейший код в нашем файле.

Обратите внимание, краткая запись обращения к функциям jQuery пишется как: «$(document).ready();» Условимся в дальнейшем писать через краткую запись, так как знак $ === jQuery по умолчанию.

Чтобы повесить обработчик событий на все элементы портфолио, достаточно написать следующий код:

```
// ожидаем загрузки документа
$(document).ready(() => {
  // после загрузки ищём все элементы с классом .portfolio-item
  // и задаём событие щелчка
  $('.portfolio-item').on('click', (e) => {

  })
});
```

Рис. 63 – Создание обработчика клика в jQuery

Библиотека jQuery сама применит этот обработчик ко всем элементам с указанным селектором, которые найдёт в документе. Вспомним, что селекторы,

передаваемые корневой функции jQuery записываются так же, как для правил CSS. Аналогично для id нужно будет начинать селектор со знака «решётка», а для тегов записывать просто их названия.

Хорошо, вызовем стандартный метод stopPropagation() для события клика. Это необходимо для того, чтобы другие события браузера не применялись к нашему клику. Например, браузер не скролил страницу в фокус того объекта, на который мы нажали.

Далее вызовем функцию createPopup(), а в качестве параметра передадим ей объект, по которому был сделан щелчок внутри функции jQuery. По щелчку JS получает объект «e» события щелчка. У него есть несколько методов и свойств. Нас интересует свойство e.currentTarget в котором хранится HTML элемент, по которому мы нажали.

Ниже создаём функцию createPopup() с параметром item. Таким образом мы передали нашей функции тот HTML фрагмент, который был сделан активным выше. Чтобы в этом убедиться напишем простую функцию вывода в консоль. А в качестве параметра передадим сам элемент item.

```
// ожидаем загрузки документа
$(document).ready(() => {
  // после загрузки ищём все элементы с классом .portfolio-item
  // и задаём событие щелчка
  $('.portfolio-item').on('click', (e) => {
    e.stopPropagation();
    createPopup(e.currentTarget);
  });
});

// функция, выполняющаяся по клику на активный элемент
function createPopup(item) {
  console.log(item);
}
```

Рис. 64 – Создание и вызов функции в ES6

Проверим, что обработчики и функция работают. Сохранив изменения, возвращаемся на страницу, открываем панель разработчика во вкладке Console, после кликаем п картинке портфолио. В консоли должен отобразиться HTML элемент, по которому мы нажали (и который передали в качестве аргумента). Также здесь будут отображаться ошибки, если что-то пошло не так.

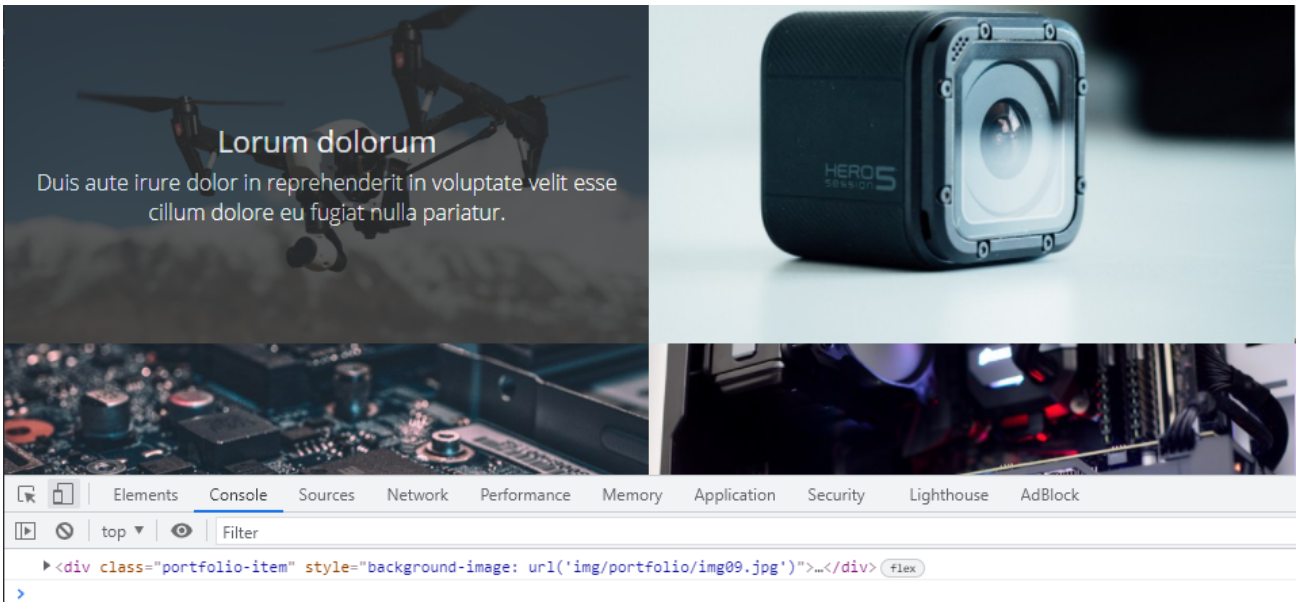

Рис. 65 – Вывод в консоль вспомогательной информации

Если по клику появляются ошибки – проверьте, правильно ли подключена библиотека, файлы скриптов и верно ли указаны селекторы портфолио. Если же по клику ничего не происходит, убедитесь, что файл popup подключен к странице, а также, что селекторы в JS и классы в файле HTML совпадают.

Следующим шагом создадим несколько переменных. Первой создадим переменную jQuery, которой передадим наш HTML-элемент, чтобы работать с ним, как с jQuery объектом. А также создадим два HTML элемента: контейнер для увеличенной картинки, и пустой тег изображения:

```
const clicked = $(item);
const container = $('<div>', {'class': 'popup-container'});
const img = $('<img>', {'class':'popup'});
```

Где первым параметром jQuery указывается с каким тегом должен быть элемент, а вторым – какие атрибуты ему необходимо установить. Здесь можно задавать любые атрибуты, которые существуют в стандартном HTML. Теперь, чтобы картинка находилась внутри контейнера поместим её при помощи метода append(). Кроме метода append(), в jQuery существуют другие методы, например, prepend() или html() – подробнее о них вы можете прочитать в документации jQuery на официальном сайте:

https://api.jquery.com/html/#html

Однако блок с картинкой сейчас существуют лишь в переменных JS. Для того, чтобы они появились в HTML документе, их нужно туда **разместить**. Сделаем это при помощи метода append к тегу body. То есть главному тегу нашей HTML-страницы. На рисунке 66 код добавления элементов.

```
11  // функция, выполняющаяся по клику на активный элемент
12  function createPopup(item) {
13    console.log(item);
14    // передаём HTML в jQuery переменную
15    const clicked = $(item);
16    // создаём блок-контейнер
17    const container = $('<div>', {'class': 'popup-container'});
18    // создаём картинку
19    const img = $('<img>', {'class':'popup'});
20    // добавляем картинку в родительский блок
21    container.append(img);
22    // блок с картинкой РИСУЕМ в HTML-документ
23    $('body').append(container);
24  }
```

Рис. 66 – Создание HTML элементов в jQuery

Сохранив изменения и перейдя на страницу после того, как мы кликнули по картинке, можно обнаружить в консоли разработчика, что на страницу добавились новые элементы в конце тега body.

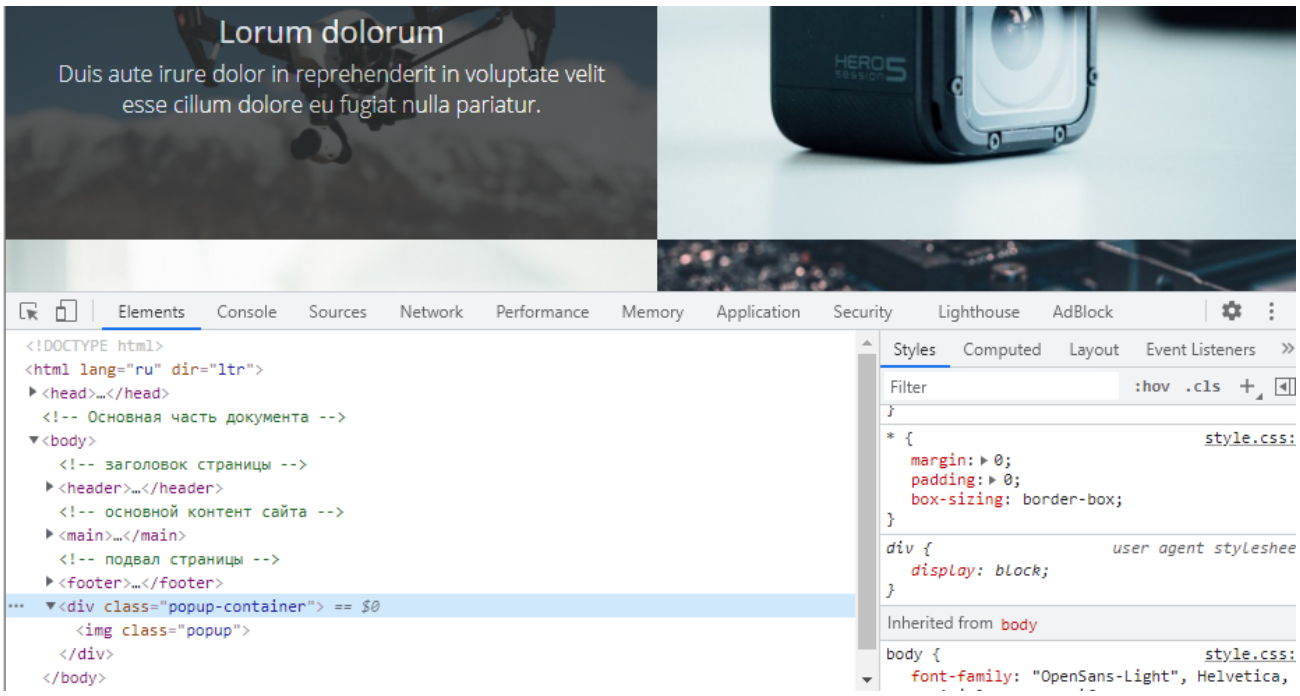

Рис. 67 – Изменённое DOM дерево после события

Но о том, что мы добавили что-то на страницу ничего не говорит, кроме нового элемента в DOM-е. К тому же у изображения нет источника картинки. А нам необходимо открывать именно ту, по которой мы кликнули. Тут нам поможет атрибут data для тега с нашим изображением. Запишем к каждому элементу атрибут data-src равный пути для фона. Например, как на рисунке 68.

```
<div class="portfolio-item" style="background-image: url('img/portfolio/img11.jpg')" data-src="img/portfolio/img11.jpg">
  <div class="description-container">
    <p class="portfolio-title">Lorum dolorum</p>
    <p class="portfolio-description">
      Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
    </p>
  </div>
</div>
<div class="portfolio-item" style="background-image: url('img/portfolio/img10.jpg')" data-src="img/portfolio/img10.jpg">
  <div class="description-container">
    <p class="portfolio-title">Lorum dolorum</p>
    <p class="portfolio-description">
      Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
    </p>
  </div>
</div>
<div class="portfolio-item" style="background-image: url('img/portfolio/img12.jpg')" data-src="img/portfolio/img12.jpg">
  <div class="description-container">
    <p class="portfolio-title">Lorum dolorum</p>
    <p class="portfolio-description">
      Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
    </p>
  </div>
```

Рис. 68 – Атрибут data-src для элементов портфолио

Теперь мы можем прочитать эти данные в коде JavaScript у элемента, который мы передали нашей функции. А затем передадим этот путь параметром, при создании HTML элемента в функции jQuery. Посмотрите на рисунок и обратите внимание, где получен путь к картинке, и как он передаётся в jQuery

```
11  // функция, выполняющаяся по клику на активный элемент
12  function createPopup(item) {
13    console.log(item);
14    // передаём HTML в jQuery переменную
15    const clicked = $(item);
16    // получаем ссылку из атрибута дата
17    const src = clicked.data('src');
18    // создаём блок-контейнер
19    const container = $('<div>', {'class': 'popup-container'});
20    // создаём картинку, задаём класс и путь
21    const img = $('<img>', {'class':'popup', 'src': src});
22    // добавляем картинку в родительский блок
23    container.append(img);
24    // блок с картинкой РИСУЕМ в HTML-документ
25    $('body').append(container);
26  }
```

Рис. 69 – Получение данных из HTML атрибута

Через атрибут data можно задавать любые данные в виде ключ-значение. Это могут быть ссылки, значения цветов, размеров, идентификаторов и т.п.

Теперь кликнув по картинке и спустившись в конец страницы, мы заметим, что под подвалом добавилась картинка. Хорошо, мы добавили элемент на страницу. Дальше всё можно поправить стилями, чтобы окно было поверх других элементов. Также будет хорошо смотреться, если остальная страница будет слегка затемнена. Необходимые стили приведены на рисунке 70.

```
.popup-container {
  position: fixed;
  z-index: 2;
  background: rgba(0, 0, 0, 0.5);
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  transition: all 250ms ease;
}
.popup {
  background: white;
  border-radius: 10px;
  padding: 15px 15px 35px;
  max-width: 75%;
  box-shadow: 0 0 0px 10px rgba(0,0,0,0.5);
  cursor: pointer;
}
```

Рис. 70 – Стили модального окна

Если всё хорошо – то результат должен быть похож как на рисунке ниже.

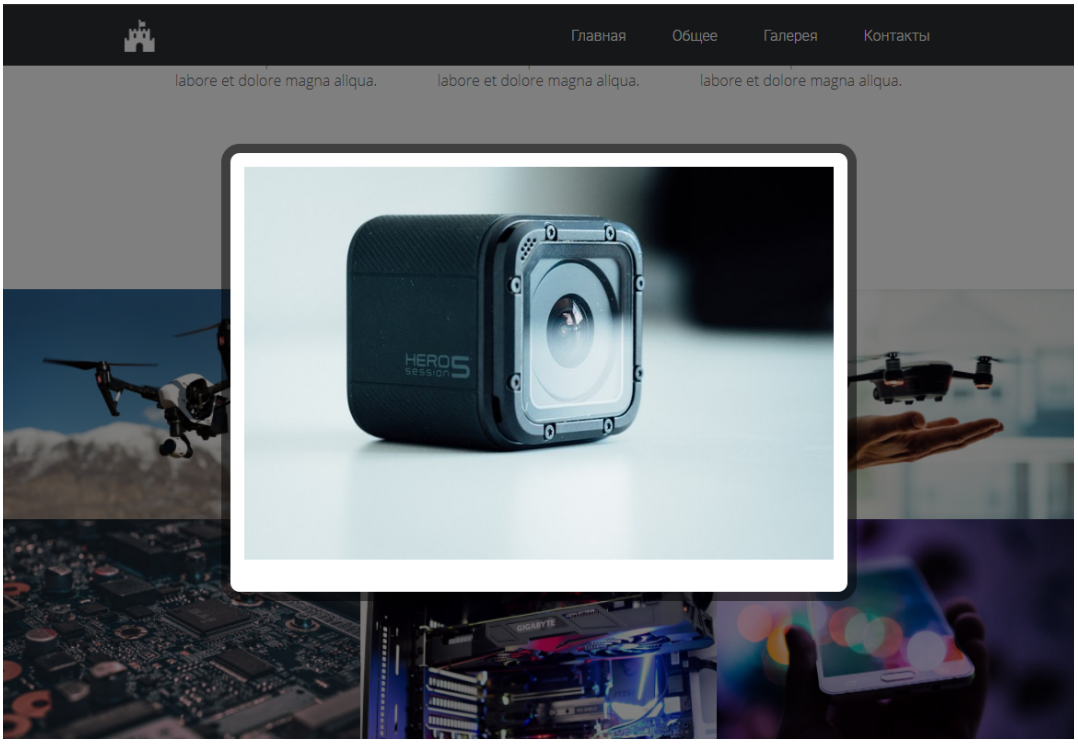

Рис. 71 – Модальное окно поверх страницы

Осталось лишь добавить закрытие. Мы добавили стилями курсор «pointer» для картинки в окне. Это популярный приём пользовательского интерфейса (UI), чтобы подсказать пользователю, что с элементом можно взаимодействовать. Добавьте такой же стиль и для картинок в портфолио, чтобы пользователь догадался на них кликнуть.

А вот по клику на картинку в окне мы сделаем его закрытие. Код выглядит просто, при клике на созданную картинку – удалить объект модального окна.

```
19
20    img.on('click', () => {
21      container.remove();
22    });
    }
```

Рис. 72 – Обработчик клика по картинке с методом удаления

Всё хорошо, но только не хватает немного плавности. Давайте доработаем код нашей функции задержкой. Пусть через время, после создания модального окна, ему будет добавлен класс. В этом поможет метод jQuery addClass(). Обратите внимание, что добавляя к элементу класс, мы пишем название класса, которое должно быть добавлено, а не ищем элемент по селектору, соответственно точку перед названием писать не нужно.

```
14    container.append(img);
15    $('body').append(container);
16    setTimeout(() => {
17      container.addClass('ready');
18    });
19
20    img.on('click', () => {
21      container.removeClass('ready');
22      setTimeout(() => {
          container.remove();
24      }, 250);
25    });
26  }
```

Рис. 73 – Установка задержек для плавного эффекта появления

При закрытии наоборот, сначала удаляем класс при помощи removeClass, а через 250ms полностью удаляем контейнер, когда анимация закончилась.

В стилях добавим эффект «выезжающей» картинки:

```
.popup-container {
  position: fixed;
  z-index: 2;
  background: rgba(0, 0, 0, 0);
  top: 0;
  right: 0;
  bottom: -1500px;
  left: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  transition: all 250ms ease;
}
.popup-container.ready {
  background: rgba(0, 0, 0, 0.5);
  bottom: 0;
}
```

Рис. 74 – Стили для плавного эффекта появления

Обратите внимание, что начальное состояние контейнера изменилось. Ещё одним частым элементом, встречающимся на страницах, является блок отзывов клиентов. Обычно представляет собой перелистываемую карусель с комментариями. Саму секцию с комментариями создать не сложно, однако для перелистывания снова понадобится помощь JavaScript.

Создадим новую секцию с типовым заголовком и подписью. Под подписью блоки-контейнеры для карточек с отзывами.

```
150         <section class="testimonials">
151           <h1 class="main-title">Отзывы</h1>
152           <div class="delimiter"></div>
153           <p class="main-title-description">Наши клиенты о нас</p>
154           <!--  внешний блок -->
155           <div class="testimonials-container">
156             <!-- обёртка -->
157             <div class="testimonials-inner">
158
159             </div>
160           </div>
161         </section>
```

Рис. 75 – Секция с отзывами

А внутри блока-обёртки несколько блоков-карточек с отзывами. Разметка одного блока изображена на рисунке 76. Допустим таких карточек будет тоже 6, по аналогии с другими секциями.

```
154         <!--  внешний блок -->
155         <div class="testimonials-container">
156           <!-- обёртка -->
157           <div class="testimonials-inner">
158             <!-- элемент -->
159             <div class="testimonials-card">
160               <div class="card-row">
161                 <img class="card-img" src="img/testimonials/01.jpg" alt="">
162                 <div class="card-head">
163                   <h4>Angelina Glory</h4>
164                   <span>25/02/2021</span>
165                 </div>
166               </div>
167               <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiu
168             </div>
```

Рис. 76 – Карточка отзыва внутри секции

Картинки для аватарок можете подобрать на свой вкус или взять из папки с дополнительными материалами к работе. Зададим блоку и карточкам стили:

```
.testimonials {
  margin: 80px;
}
.testimonials-container {
  max-width: 960px;
  margin: auto;
  overflow: hidden;
}
.testimonials-inner {
  display: flex;
  padding: 40px 0 25px;
  transition: transform 350ms ease-in-out;
}
.testimonials-card {
  box-shadow: 0 0 8px rgba(0, 0, 0, 0.2);
  width: 47%;
  flex-shrink: 0;
  margin: 15px;
  border-radius: 7px;
  padding: 15px;
  color: #454546;
}
.card-img {
  width: 60px;
  border-radius: 50%;
  margin-right: 30px;
}
.card-row {
  display: flex;
  margin-bottom: 15px;
  align-items: center;
}
.card-head span {
  line-height: 1.5em;
  color: #c1c1c1;
}
```

Рис. 77 – Стили для секции и карточек отзывов

Результат должен быть примерно, как на рисунке 78. То есть отображается лишь две карточки из шести. Дело в том, что мы задали карточкам ширину, не позволяющую всем, разместится внутри родительского блока. А родительскому блоку мы не разрешили переносить карточки, как раньше. Ну а для того, чтобы не было видно остальных, мы добавили классу testimonials-container стиль overflow: hidden, который скрывает всё, что не помещается в блок. Попробуйте удалить этот стиль и посмотреть, что получится.

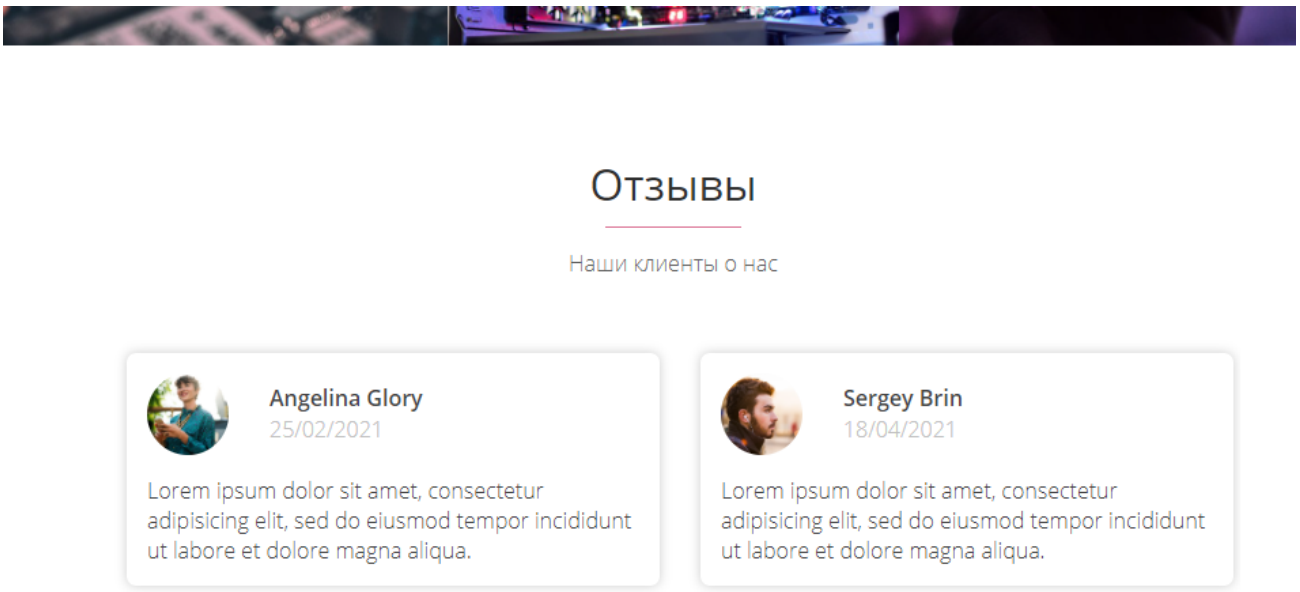

Рис. 78 – Внешний вид карточек отзывов

Добавим под карточки кнопки-подсказки. Они будут подсказывать, что комментариев несколько, а также будут отображать, какая страница комментариев сейчас активна. Добавим их под блок-обёртку:

```
156          <!-- обёртка -->
157 ›        <div class="testimonials-inner">…
222          <!-- кнопки -->
223          <div class="testimonials-controls">
224            <div class="control-item active"></div>
225            <div class="control-item"></div>
226            <div class="control-item"></div>
227          </div>
228        </div>
229      </section>
```

Рис. 79 – Кнопки переключения отзывов

Создадим их в 2 раза меньше, чем отзывов. Так как перелистывать будем по два отзыва за раз. Также зададим соответствующие стили.

```
.control-item {
  flex: 1;
  margin: 15px 10px;
  height: 6px;
  border-radius: 3px;
  transition: background-color 250ms ease;
  background: lightgray;
  cursor: pointer;
}
.control-item:hover {
  background-color: gray;
}
.control-item.active {
  cursor: default;
  background-color: #ce416e;
}
```

Рис. 80 – Стили для кнопок

Внешний вид секции изображён на рисунке ниже.

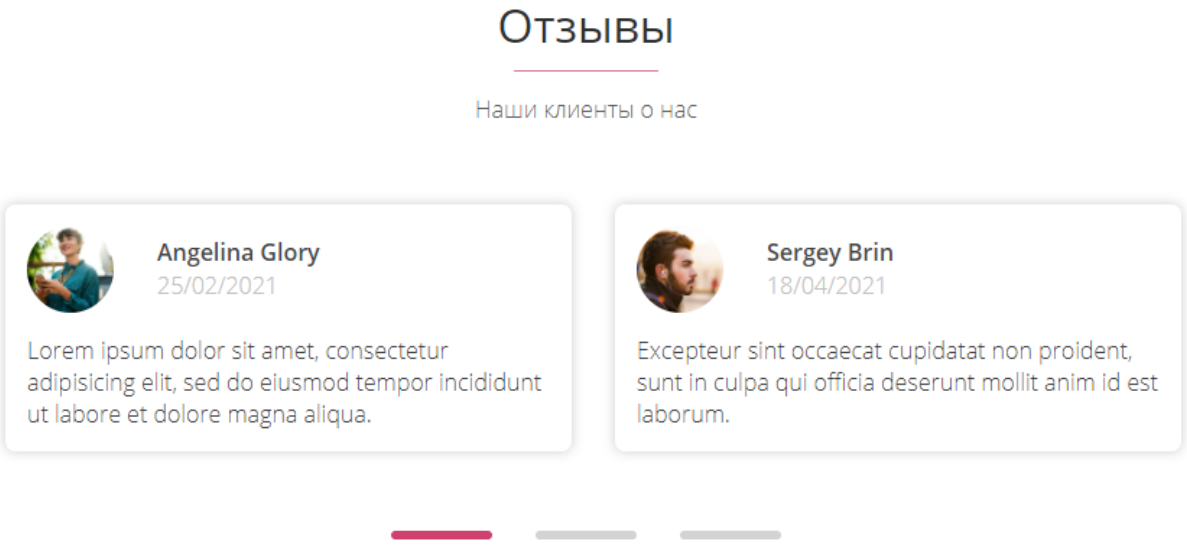

Рис. 81 – Внешний вид секции отзывов

Теперь мы готовы оживить этот раздел. Перейдём в файл popup.js и добавим в корневую функцию новый обработчик события «клик» по блоку с классом .control-item.

```
// ожидаем загрузки документа
$(document).ready(() => {
  // после загрузки ищём все элементы с классом .portfolio-item
  // и задаём событие щелчка
  $('.portfolio-item').on('click', (e) => {
    e.stopPropagation();
    createPopup(e.currentTarget);
  });

  // щелчок по кнопке отзывов
  $('.control-item').on('click', (e) => {
    e.stopPropagation();
    slideTestimonials(e.currentTarget);
  });
});
```

Рис. 82 – Создание обработчика событий

Внутри вызываем функцию slideTestimonials(), в которую также в качестве параметра передаём элемент, на который был произведён щелчок. Ниже опишем функцию. Здесь добавим условие, если элемент уже активен, то есть содержит класс active, то прерываем функцию, так как она не имеет смысла. Элемент уже в нужном состоянии.

```
// функция, выполняющаяся по клику на НЕактивный элемент
function slideTestimonials(item) {
  // передаём HTML в jQuery переменную
  const clicked = $(item);
  // проверяем что страница неактивна
  if (clicked.hasClass('active')) {
    // прекращаем работу программы, если уже активна
    return;
  }
  // получаем номер кнопки
  const index = $('.control-item').index(clicked);
  console.log(index);
}
```

Рис. 83 – Создание функции перелистывания

Следом за этим получаем индекс кнопки, по которой кликнули. Добавьте вывод в консоль, чтобы проверить работу функции. Обратите внимание, что индексы jQuery возвращает как и для массивов, начиная с 0.

И это удобно, так как нам необходимо пролистывать определённое количество раз блок с карточками. Соответственно если пользователь кликнул на первую кнопку, то необходимо пролистать контейнер 0 раз.

```
44    // получаем номер кнопки
45    const index = $('.control-item').index(clicked);
46    // измеряем ширину карточки вместе с margin - ключ true
47    const width = $('.testimonials-card').outerWidth(true);
48    // измеряем необходимое для пролистывания расстояние и применяем к блоку-обёртке
49    const scroll = width * 2 * index;
50    $('.testimonials-inner').css('transform', 'translateX(-'+ scroll +'px)');
51  }
```

Рис. 84 – Код, листающий блок с карточками

На рисунке 84 описаны шаги по измерению полной ширины карточки вместе с margin. Без ключа true функция outerWidth() возвращает ширину блока только с учётом padding и границы border. Умножаем на 2, так как функция листает по 2 отзыва и на индекс кнопки. Полученный результат приписываем стилем к блоку обёртке. Внимание на знак «минус» в 50 строчке.

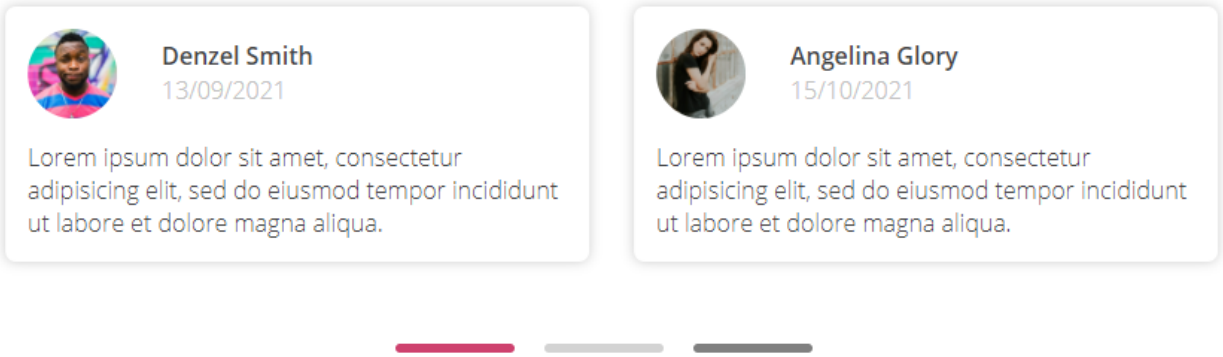

Рис. 85 – Перелистанные карточки

Если функция отрабатывает правильно, то у вас должно заработать перелистывание. Однако не меняется статус текущей активной кнопки. Из-за чего не получается перелистнуть в начало. Доработаем нашу функцию:

```
50    $('.testimonials-inner').css('transform', 'translateX(-'+ scroll +'px)');
51
52    // удаляем у всех кнопок класс active
53    $('.control-item').removeClass('active');
54    // задаём класс active кнопки с текущим индексом
55    $('.control-item').eq(index).addClass('active');
56  }
```

Рис. 86 – Стили для плавного эффекта появления

Как видно из текста комментариев. Мы удалили класс active где бы он не был среди кнопок. Затем, при помощи функции .eq() передали индекс элемента, которому мы устанавливаем класс active. Обратите внимание, что в строке 55 мы последовательно вызвали несколько функций к элементу.

На самом деле вызовы выше можно было сократить до строчки

```
53    $('.control-item').removeClass('active').eq(index).addClass('active');
```

И таких вызовов может быть сколь угодно много. Однако, такой стиль может вызвать затруднения при дальнейшей работе с программой. Потому лучше разделять вызовы функций, как сложные предложения, на части. Так будет легче читать вашему коллеге в будущем, когда необходимо будет изменить или доработать ваш код.

Задание на контроль

1. Повторить пример их практической части.

2. Для раздела «Портфолио» необходимо реализовать отображение изображений в модальном окне.

3. Для раздела «Отзывы» реализовать перелистывание карточек.

4. Объяснить, при помощи какой библиотеки было реализовано окно, как была подключена библиотека.

5. Объяснить, как вызываются методы библиотеки.

6. Объяснить, как передаются параметры в функции JS и jQuery

7. Лабораторная работа считается защищенной, если студент показал в окне браузера страницу из практической части со стилизованным меню и ответил правильно на все заданные контрольные вопросы (следующая лабораторная работа не подлежит защите до тех пор, пока не будет защищена предыдущая).