Тема 12. Архитектура и организация проекта.

хекслет колледж

Цель занятия:

Сформировать у студентов представление о правильной организации файлов и структуры фронтенд-проекта, принципах разделения ответственности и базовой модульности кода при разработке многостраничного сайта без серверной части.

Учебные вопросы:

1. Понятие архитектуры фронтенд-проекта
2. Структура папок фронтенд-проекта. Разделение ответственности
3. Подключение файлов в проекте
4. Организация JavaScript-кода. Модульная система ES6

1. Понятие архитектуры фронтенд-проекта

Архитектура фронтенд-проекта — это способ организации структуры проекта и кода, который определяет:

- как разложены файлы и папки;
- где хранится HTML, CSS и JavaScript;
- как взаимодействуют части приложения между собой;
- как проект развивается и поддерживается со временем.

Проще говоря, архитектура отвечает на вопрос:
«Как правильно организовать проект, чтобы с ним было удобно работать сейчас и в будущем».

Зачем нужна архитектура?

Грамотная архитектура позволяет:

- быстрее ориентироваться в проекте;
- упрощать добавление новых страниц и функций;
- избегать дублирования кода;
- легче находить и исправлять ошибки;
- работать над проектом нескольким разработчикам.

Даже в учебных и небольших проектах архитектура важна, так как формирует профессиональные привычки разработки.

Проблемы при отсутствии архитектуры

Если архитектура проекта не продумана, возникают типичные проблемы:
все файлы находятся в одной папке;

- JavaScript-код написан в одном большом файле;
- сложно понять, какой код за что отвечает;
- любое изменение приводит к ошибкам в других частях проекта;
- проект тяжело дорабатывать и проверять.

Такие проекты называют «хаотичными» или «неподдерживаемыми».

Архитектура и масштабируемость

Правильная архитектура:

- позволяет легко добавлять новые страницы;
- упрощает подключение новой логики;
- делает проект устойчивым к изменениям.

Даже если проект изначально небольшой (несколько страниц), он должен быть организован так, как будто он будет расти.

Архитектура в рамках лекции

В рамках данного курса под архитектурой понимается:

- понятная структура папок;
- разделение HTML, CSS и JavaScript;
- логическая организация JavaScript-кода;
- использование относительных путей и корректного подключения файлов;
- подготовка проекта к курсовой работе.

Речь не идёт о сложных фреймворках и сборщиках, а о базовых архитектурных принципах, обязательных для начинающего фронтенд-разработчика.

Вывод:

Архитектура фронтенд-проекта — это фундамент любого сайта.

Хорошо организованный проект:

- проще писать;
- проще проверять;
- проще защищать и дорабатывать.

2. Структура папок проекта. Разделение ответственности.

Грамотная структура папок и чёткое разделение ответственности между HTML, CSS и JavaScript — основа удобного и поддерживаемого фронтенд-проекта.

Для учебных и курсовых работ рекомендуется использовать простую и понятную структуру:

Типовая структура фронтенд-проекта:

```
project/
├── index.html
├── pages/
│   ├── about.html
│   └── contacts.html
├── css/
│   └── style.css
├── js/
│   ├── main.js
│   ├── menu.js
│   └── modal.js
├── images/
│   └── logo.png
└── assets/
    └── fonts/
```

Такая структура:

- сразу показывает назначение файлов;
- облегчает навигацию по проекту;
- упрощает сопровождение и проверку работы.

Назначение основных папок

HTML-файлы

- index.html — главная страница сайта;
- папка pages/ — дополнительные страницы проекта;
- каждый HTML-файл отвечает только за структуру и содержание страницы.

CSS (css/)

- содержит стили сайта;
- отвечает за внешний вид, адаптивность и оформление;
- CSS не должен содержать логику и поведение.

JavaScript (js/)

- содержит логику сайта;
- управляет интерактивностью и поведением элементов;
- может быть разделён на несколько файлов по задачам.

Изображения и ресурсы

- images/ — картинки, иконки;
- assets/ — дополнительные ресурсы (шрифты, иконки).

Разделение ответственности

HTML

Отвечает за:

- структуру страницы;
- семантику;
- текстовое содержимое.

HTML не должен:

- содержать стили оформления;
- содержать сложную JavaScript-логику.

CSS

Отвечает за:

- внешний вид;
- расположение элементов;
- адаптацию под разные экраны.

CSS не должен:

- управлять логикой приложения;
- заменять JavaScript.

JavaScript

Отвечает за:

- поведение элементов;
- обработку пользовательских действий;
- изменение содержимого страницы.

JavaScript не должен:

- содержать разметку страницы целиком;
- дублировать стили оформления.

Разделение HTML, CSS и JavaScript позволяет:

- легче находить ошибки;
- безопасно вносить изменения;
- повторно использовать код;
- упростить командную работу.

Типичные ошибки начинающих:

- стили прямо в HTML (style="");
- JavaScript-код внутри HTML-документа без необходимости;
- один файл для всего проекта;
- хаотичное именование файлов.

Вывод:

- Структура папок и разделение ответственности — фундамент качественной фронтенд-разработки.
- Даже простой сайт должен быть организован так, чтобы любой разработчик мог быстро понять, что где находится и за что отвечает.

3. Подключение файлов в проекте

Правильное подключение файлов — важный элемент архитектуры фронтенд-проекта.
От того, как и где подключаются CSS и JavaScript, зависят корректность работы сайта, скорость загрузки и удобство поддержки проекта.

Подключение CSS-файлов

CSS-файлы подключаются в секции <head> HTML-документа с помощью тега <link>.

Пример:

```
<head>
    <link rel="stylesheet" href="css/style.css">
</head>
```

Основные правила:

- стили подключаются до загрузки страницы, чтобы избежать «мигания» оформления;
- путь к файлу должен быть корректным и относительным;
- порядок подключения стилей имеет значение (поздние стили могут переопределять предыдущие).

Подключение JavaScript-файлов

JavaScript-файлы подключаются с помощью тега <script>.
Рекомендуемый способ:

```
<script src="js/main.js" defer></script>
```

Почему используется defer?

- HTML загружаетсяи отображается без задержек;
- скрипт выполняется после загрузки DOM;
- уменьшается вероятность ошибок при работе с элементами страницы.

Где располагать тег <script>?

Существует два безопасных варианта:

- В конце тега <body>
- В <head> с атрибутом defer (предпочтительный вариант)

Подключение нескольких JavaScript-файлов

В проектах часто используется несколько скриптов:

```
<script src="js/menu.js" defer></script>
<script src="js/modal.js" defer></script>
<script src="js/main.js" defer></script>
```

Важно:

- файлы подключаются в нужном порядке;
- main.js часто подключается последним, если он использует другие скрипты.

Использование относительных путей

Во фронтенд-проектах используются относительные пути, зависящие от структуры папок проекта.
Пример:

```
<script src="../js/main.js" defer></script>
```

Это означает переход на один уровень выше относительно текущего HTML-файла.

Типичные ошибки при подключении файлов:

- неправильные пути к файлам;
- подключение скриптов в <head> без атрибутов;
- дублирование подключений одного и того же файла.

Проверка подключения

Для проверки корректного подключения JavaScript используется:

console.log("Скрипт подключён");

Если сообщение отображается в консоли - файл подключён правильно.

Вывод:

Корректное подключение файлов:

- обеспечивает стабильную работу сайта;
- повышает производительность;
- упрощает поддержку и развитие проекта.

Это обязательный навык для разработки качественного многостраничного фронтенд-проекта

4. Организация JavaScript-кода. Модульная система ES6.

По мере роста проекта объём JavaScript-кода увеличивается.
Чтобы код оставался понятным, управляемым и переиспользуемым, его необходимо правильно организовывать.
Для этого используется разбиение кода на файлы и модульный подход, реализованный в стандарте ES6.
ES6 — это одна из версий стандарта языка JavaScript.

Зачем организовывать JavaScript-код?

Плохо организованный код:

- сложно читать и понимать;
- трудно поддерживать и расширять;
- приводит к дублированию функций;
- повышает вероятность ошибок.

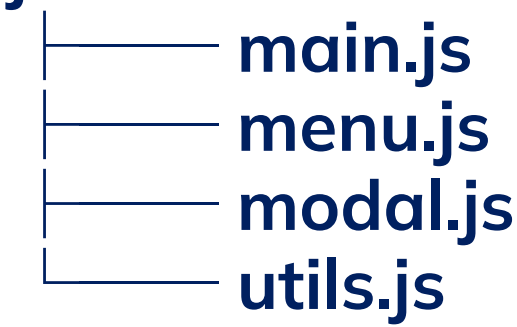
Хорошо организованный код:

- разделён по задачам;
- легко масштабируется;
- понятен не только автору, но и другим разработчикам.

Разделение JavaScript-кода по файлам

Вместо одного большого файла используется несколько файлов, каждый из которых отвечает за свою задачу:

Пример структуры:



Примеры назначения файлов:

- menu.js — логика меню;
- modal.js — модальные окна;
- utils.js — вспомогательные функции;
- main.js — точка входа, объединяющая логику.

Проблемы проектов без модульного подхода

Без модульного подхода:

- все переменные и функции находятся в глобальной области видимости;
- возможны конфликты имён;
- сложно контролировать зависимости между частями кода.

Модульная система ES6
ES6-модули позволяют:

- делить код на независимые части (модули);
- явно указывать, какие элементы экспортируются и импортируются;
- избегать загрязнения глобальной области видимости.

Экспорт из модуля

Для передачи данных из одного файла в другой используется export.
Пример (utils.js):

```
export function sum(a, b) {
    return a + b;
}
```

Также можно экспортировать переменные:
export const PI = 3.14;

Импорт модулей

В файле, где модуль используется, применяется import.

Пример (main.js):

```
import { sum, PI } from './utils.js';

console.log(sum(2, 3));
console.log(PI);
```

Подключение модулей в HTML

Для работы ES6-модулей необходимо указать атрибут type="module":

```
<script type="module" src="js/main.js"></script>
```

Особенности модулей:

- код выполняется в собственном пространстве имён;
- defer применяется автоматически;
- модули работают только при загрузке через HTTP (локальный сервер).

Основные правила работы с модулями:

- всегда указывать расширение .js при импорте;
- использовать относительные пути;
- логически разделять функциональность;
- избегать циклических зависимостей.

Когда использовать модули:

- Модули рекомендуется использовать:
- начиная с проектов средней сложности;
- в курсовых работах;
- при повторном использовании кода.

Для самых простых учебных проектов допускается один файл, но с ростом проекта модульность становится необходимой.

Вывод:

Модульная система ES6:

- делает код структурированным и читабельным;
- предотвращает конфликты переменных;
- упрощает развитие проекта.

Грамотная организация JavaScript-кода — важный шаг от учебного скрипта к полноценному фронтенд-проекту.

Выводы по теме лекции:

- архитектура проекта — это осмысленная организация файлов, папок и кода;
- даже небольшой фронтенд-проект требует понятной структуры;
- корректная структура папок облегчает навигацию по проекту и его сопровождение;
- важно строгое разделение ответственности между HTML, CSS и JavaScript;
- HTML отвечает за структуру, CSS — за оформление, JavaScript — за логику и интерактивность;
- файлы стилей и скриптов должны подключаться корректно и с использованием относительных путей;
- JavaScript-код рекомендуется разделять на несколько файлов по функциональности;
- модульная система ES6 позволяет структурировать код и избегать конфликтов в глобальной области видимости;
- грамотная организация проекта повышает качество кода и упрощает подготовку курсовой работы.

Контрольные вопросы:

Что понимается под архитектурой фронтенд-проекта?
Зачем необходима продуманная структура проекта даже для небольшого сайта?
Какие основные папки используются в типовом фронтенд-проекте и для чего они предназначены?
Какую роль выполняют HTML, CSS и JavaScript в проекте?
Почему не рекомендуется смешивать разметку, стили и логику в одном файле?
Как правильно подключать CSS-файлы к HTML-документу?
Зачем разделять JavaScript-код на несколько файлов?
Что такое модульный подход в JavaScript?
Как подключается модульный JavaScript-файл к HTML-странице?

Домашнее задание:

https://ru.hexlet.io/courses/js-basics

хекслет колледж

@HEXLY.KZ