



Geekbrains

**Разработка и поддержка адаптивного сайта магазина гитар
на основе JavaScript.**

Программа: Программист
Frontend-разработчик

Замчалин Михаил Николаевич

Москва

2024

Содержание

Введение (стр.)

Глава 1. Основы разработки адаптивного сайта (стр.)

1.1 Кто такой frontend-разработчик? Какие задачи выполняет?

1.2 Основы работы веб - страницы

1.3 Технологии и инструменты Frontend - разработчика

1.4 Язык гипертекстовой разметки HTML.

1.5 HTML5 и его новые возможности.

1.6 Каскадная таблица стилей CSS.

1.7 CSS3 и его новые возможности

1.8 Git и почему эта технология настолько важна.

Глава 2. Как превратить красивую картинку в живой интернет организм
(стр.)

2.1 JavaScript основа веб-программирования

2.2 ECMAScript и его возможности

2.3 Node.js

2.4 REST Api

2.5 ООП на основе JavaScript

2.6 Основные принципы адаптивного сайта

2.6 Проверка сайта, его функциональности и отображения на разных устройствах

Глава 3. (стр.)

3.1 Разработка веб-страниц без использования макета

3.2 Подключение CSS к проекту и адаптивная верстка

3.3 Подключение JS к проекту. Разработка функциональности сайта.

3.4 Проверка сайта на валидность и функциональность.

Заключение (4 стр.)

Список используемой литературы

Приложения

Введение

В современном digital-мире, где доступ к информации возможен через множество устройств — от настольных компьютеров до смартфонов и планшетов, важность адаптивного веб-дизайна невозможно переоценить. Прежде всего, адаптивный веб-дизайн — это подход, который позволяет веб-сайтам изменять свою компоновку и содержимое в зависимости от размеров и разрешения экрана устройства. Это делает взаимодействие с сайтом более удобным и эффективным для пользователей.

Адаптивные сайты не только способны подстраиваться под различные размеры экранов, но и обеспечивают пользователю целостный и гладкий опыт взаимодействия. Frontend-разработка, которая охватывает создание визуальной части веб-приложений, играет ключевую роль в этом процессе. В рамках данной работы мы рассмотрим основные технологии и инструменты, используемые в адаптивной веб-разработке, включая HTML5, CSS3 и JavaScript, а также фреймворки и библиотеки, такие как Bootstrap и React.

Темы, которые будут рассмотрены в данном дипломном проекте, включают определение роли Frontend-разработчика, основные технологии и инструменты, используемые в адаптивной веб-разработке, современные тренды и вызовы, а также примеры успешной реализации адаптивного и функционального сайта.

Тема проекта: разработка и поддержка адаптивного сайта магазина гитар на основе нативного JavaScript.

Цель: изучить особенности верстки сайта эконо сегмента без макета. Возможности добавления функциональной части сайта без использования библиотек.

Какую проблему решает: случается, что заказчик из эконом - сегмента запрашивает быстро сделать небольшой сайт для маленького магазина без привлечения команды.

Задачи:

- Изучить литературу, касающуюся темы исследования.
- Рассмотреть основные виды и методы разработки функционального сайта.
- Ознакомиться с основными принципами работы таких технологий как HTML, CSS, JavaScript.
- Составить план разработки сайта.
- Разработать сайт по заранее разработанному плану
- Провести проверку сайта на валидность и функциональность.

Инструменты: Git, Html, CSS, JavaScript, SCSS.

Состав команды: Замчалин Михаил Николаевич (Frontend-разработчик)

ГЛАВА 1

Основы разработки адаптивного сайта

Адаптивная веб-разработка становится все более актуальной в условиях многообразия устройств и разрешений экранов, которые используют пользователи для доступа к информации. Эта глава посвящена основам работы в области фронтенд-разработки, задачам, стоящим перед разработчиками, а также основным технологиям, необходимым для создания адаптивных веб-сайтов.

1.1 Кто такой frontend-разработчик? Какие задачи выполняет?

Frontend-разработчик — это специалист, который создает визуальную часть веб-приложений, отвечающую за взаимодействие пользователя с сайтом. Основная цель фронтенд-разработчика заключается в том, чтобы предоставить пользователю удобный и интуитивно понятный интерфейс. В рамках своей работы разработчик выполняет следующие задачи:

Проектирование интерфейса: Разработка макетов и прототипов с помощью инструментов дизайна, таких как Figma, чтобы обеспечить удобство и функциональность пользовательского опыта.

Верстка страниц: Преобразование дизайна в код с использованием HTML, CSS и JavaScript, чтобы страницы отображались корректно на различных устройствах.

Оптимизация производительности: Обеспечение быстрой загрузки страниц путем минимизации кода, сжатия изображений и использования современных инструментов и методов.

Тестирование и отладка: Проверка работоспособности веб-приложений на различных браузерах и устройствах, а также устранение найденных ошибок.

Сотрудничество с командой: Работа в команде с бэкенд-разработчиками, дизайнерами и продуктовыми менеджерами для достижения единой цели и создания качественного продукта.

1.2 Основы работы веб-страницы

Веб-страница — это документ, создаваемый с использованием языков разметки, который отображается в интернет-браузере. Основными элементами структуры веб-страницы являются:

HTML (HyperText Markup Language): Язык, который обеспечивает разметку контента. Он определяет структуру документа и включает такие элементы, как заголовки, параграфы, изображения, ссылки, блоки, поля ввода и т.д.

CSS (Cascading Style Sheets): Язык, используемый для описания визуальной презентации HTML-документов. CSS позволяет изменять шрифты, цвета, размеры и позиционирование элементов на странице.

JavaScript: Скриптовый язык программирования, который добавляет интерактивность и динамичность на веб-страницы. С его помощью можно реализовать анимацию, управление событиями и взаимодействие с пользователем.

Страницы веб-сайта могут быть статическими или динамическими. Статические страницы отображают фиксированный контент, тогда как динамические генерируются на сервере и могут изменяться в зависимости от действий пользователя.

1.3 Технологии и инструменты Frontend-разработчика

Frontend-разработка включает в себя множество технологий и инструментов, необходимых для создания современных веб-приложений. Некоторые из них включают:

HTML/CSS: Основные технологии для создания структуры и стилей веб-страниц.

JavaScript: Основной язык программирования для создания интерактивности на веб-сайтах.

Фреймворки и библиотеки: Использование фреймворков, таких как React, Angular, Vue.js, помогает упростить разработку сложных интерфейсов, предоставляя готовые компоненты и решая задачи управления состоянием.

Инструменты сборки: Такие как Webpack и Gulp, которые позволяют автоматизировать задачи по сборке и оптимизации кода.

Системы контроля версий: Git становится стандартом для управления изменениями в коде, позволяя разработчикам работать над проектами одновременно.

1.4 Язык гипертекстовой разметки HTML

HTML, или язык гипертекстовой разметки, служит основой для создания веб-страниц. Он структурирует содержимое и делает его доступным для браузеров.

HTML-документ состоит из элементов, заключенных в теги.

Основные характеристики HTML:

Структура документа: Обязательные элементы включают ``<html>``, ``<head>`` и ``<body>``. В ``<head>`` размещаются метаданные, а в ``<body>`` — контент страницы.

Основные HTML-теги

Форматирование текста: ``, `<i>`, `<u>`, ``, `` и другие.

Заголовки: `<h1>` до `<h6>`

Абзацы и разрывы строк: `<p>`, `
`

Списки: ``, ``, ``, `<dl>`, `<dt>`, `<dd>`

Ссылки: `<a>`

Картинки: ``

Атрибуты и мета-теги

Глобальные атрибуты: `class`, `id`, `style`.

Атрибуты событий: `onclick`, `onload`.

Основные мета-теги: `charset`, `viewport`, `description`.

Формы

Формы: `<form>`

Поля ввода: `<input>`, `<textarea>`, `<button>`, `<select>`, `<option>`.

Валидация форм разными способами.

Таблицы

Основная структура таблицы: `<table>`, `<tr>`, `<td>`, `<th>`

1.5 HTML5 и его новые возможности

HTML5 стал революционным в плане возможностей, которые он предоставляет разработчикам. Он включает улучшения, которые упрощают создание мультимедийного контента и взаимодействия с пользователем, например:

Новые семантические теги: `<article>`, `<section>`, `<nav>`, `<header>`, `<footer>`, которые улучшает структуру документа и SEO.

Поддержка мультимедиа: Теги `<audio>` и `<video>` упрощают добавление медиаконтента без необходимости в сторонних плагинах.

API: HTML5 предоставляет различные API, такие как Canvas для графики, Geolocation для определения местоположения и Web Storage для хранения данных на стороне клиента.

1.6 Каскадная таблица стилей CSS

CSS (Cascading Style Sheets) — это язык стилей, который позволяет отделить содержимое документа от его представления. CSS обеспечивает гибкость в изменении внешнего вида страницы, не затрагивая HTML-код.

Основные возможности CSS:

Селекторы: позволяют применять стили к элементам на основе их тегов, классов, идентификаторов и атрибутов.

Модели блочной модели: CSS управляет расположением элементов на странице с помощью свойств, таких как `margin`, `border`, `padding` и `width/height`.

Медиа-запросы: Возможность адаптивного дизайна — изменять стили в зависимости от размеров экрана. Менять элементы местами и делать сайт более удобным вне зависимости от устройства, с которого им пользуются.

1.7 CSS3 и его новые возможности

CSS3 предлагает расширенные возможности по сравнению с предыдущими версиями. Некоторые из ключевых функций включают:

Градиенты и тени: CSS3 позволяет создавать градиенты и тени, добавляя глубину и стиль к элементам.

Анимации и переходы: Разработчики могут анимировать изменения свойств CSS, делая интерфейс более привлекательным.

Flexbox: это современная модель компоновки в CSS, разработанная для упрощения укладки элементов на веб-странице. Эта модель позволяет разработчикам легко выстраивать и выравнивать элементы в контейнерах, даже когда их размеры неизвестны или динамичны.

Преимущества FlexBox:

- Упрощение компоновки: Flexbox делает задачу расположения элементов более интуитивной и удобной.
- Адаптивность: легко настраивается под разные размеры экранов и устройства.
- Простой контроль выравнивания и распределения пространства: Flexbox предоставляет мощные инструменты для управления расположением элементов.

Grid Layout: это мощная технология компоновки, которая позволяет создавать сложные и адаптивные макеты на веб-страницах с помощью сетки. Вместо линейного расположения элементов, как в Flexbox, Grid предоставляет двумерную сетку, где элементы могут занимать разные размеры и позиции.

CSS Grid Layout позволяет вам с легкостью создавать сложные и адаптивные макеты, при этом вы можете легко управлять размерами и позициями элементов. С его помощью разработка современного и функционального интерфейса становится более быстрой и эффективно

1.8 Git и почему эта технология настолько важна

Git — это система контроля версий, которая позволяет разработчикам отслеживать изменения в коде и работать над проектами совместно.

Она важна по нескольким причинам:

История изменений: Git позволяет хранить историю всех изменений, что упрощает отслеживание ошибок и откат к предыдущим версиям

Совместная работа: Система разработки позволяет множеству разработчиков работать над одним проектом одновременно, избегая конфликтов.

Разграничение задач: Git предоставляет возможность создавать ветки (branches), что позволяет изолировать различные функции и исправления ошибок, прежде чем затем объединить их в основную ветку. Таким образом, первая глава предоставляет базовые знания о frontend-разработке, охватывая ключевые аспекты и технологии, которые необходимы для создания адаптивных веб-сайтов. Эти знания помогут в дальнейших разделах дипломного проекта, где будут рассмотрены практические примеры и лучшие практики адаптивной разработки.

Глава 2. Как превратить красивую картинку в живой интернет организм.

2.1 JavaScript основа веб-программирования

JavaScript — это высокоуровневый, интерпретируемый язык программирования, который изначально был создан для разработки динамических и интерактивных веб-страниц. Он стал неотъемлемой частью технологии веб-разработки и играет ключевую роль в создании современных веб-приложений. Давайте более подробно рассмотрим его особенности, функции и роль в веб-программировании.

1. Основные характеристики JavaScript

Кросс-платформенность: JavaScript выполняется на всех современных веб-браузерах и поддерживается различными операционными системами, что делает его универсальным инструментом для веб-разработки.

Динамическая типизация: В JavaScript тип данных переменной определяется во время выполнения. Это дает возможность легко изменять значение переменной без необходимости указывать ее тип заранее.

Прототипное наследование: JavaScript основан на объектно-ориентированном подходе. Он использует прототипное наследование, что позволяет создавать объекты, основанные на других объектах, а не на классах.

Функции как объекты первого класса: Функции в JavaScript являются объектами первого класса. Это означает, что функции могут передаваться как аргументы другим функциям, возвращаться из других функций и присваиваться переменным.

2. Разработка интерактивности

Одной из главных причин популярности JavaScript является его способность создавать интерактивные элементы на веб-страницах. Это включает в себя:

Обработка событий: JavaScript позволяет пользователям взаимодействовать с элементами веб-страницы через такие события, как клики мышью, наведение курсора, нажатие клавиш и многое другое. Например, можно создать кнопку, нажатие на которую будет вызывать определенную функцию.

Анимация: С помощью JavaScript можно создавать анимацию, которая улучшает пользовательский интерфейс. Библиотеки, такие как jQuery и GSAP, облегчают создание сложных анимаций.

Манипуляция DOM: Document Object Model (DOM) представляет собой структуру HTML-документа, и JavaScript позволяет динамически изменять содержимое и стиль страницы путем манипуляции с элементами. Это позволяет обновлять контент без перезагрузки страницы.

3. Асинхронное программирование

JavaScript поддерживает асинхронное программирование, что особенно полезно для веб-разработки, где запросы к серверу могут занять неопределенное время. Основные аспекты:

Промисы: В JavaScript можно использовать промисы для обработки асинхронных операций. Промис представляет собой объект, который может находиться в одном из трех состояний: ожидание, выполнен или отклонен. Это позволяет писать более читаемый и управляемый код.

async/await: Эта конструкция была введена для упрощения работы с промисами и асинхронными вызовами. Она позволяет писать асинхронный код, который выглядит и работает как синхронный, что делает его легче для понимания и отладки.

4. Взаимодействие с сервером

JavaScript позволяет взаимодействовать с сервером для получения данных без перезагрузки страницы через:

AJAX: Асинхронный JavaScript и XML (AJAX) позволяют отправлять и получать данные от сервера без обновления страницы. Это существенно улучшает работу с динамическим контентом.

Fetch API: Это современный способ выполнения сетевых запросов. Он предоставляет более удобный и мощный интерфейс по сравнению с XMLHttpRequest и поддерживает промисы, что делает его использование более простым и эффективным.

5. Разработка на стороне клиента и сервера

JavaScript можно использовать как на стороне клиента, так и на стороне сервера:

На стороне клиента: JavaScript обрабатывается в браузере пользователя и отвечает за взаимодействие с пользователем. Он управляет логикой веб-приложения, а также отображением данных.

На стороне сервера: с помощью платформы Node.js JavaScript можно запускать на сервере, что позволяет создавать серверные приложения. Это открыло новые возможности для веб-разработки, включая создание RESTful API, работа с базами данных и серверное управление аутентификацией.

JavaScript является основой веб-программирования благодаря своей универсальности, возможностям интерактивности, поддержки асинхронного программирования и взаимодействия с сервером. Он активно используется для разработки как клиентских, так и серверных приложений. Понимание JavaScript и его экосистемы является важным шагом для любого разработчика, стремящегося создавать современные веб-приложения.

2.2 ECMAScript

ECMAScript — это стандарт, на основе которого разработан язык программирования JavaScript. Он задает правила, как должен функционировать язык, определяет синтаксис, типы данных, операторы, объекты, уникальные методы и другие составляющие, необходимые для построения языка программирования. Стандарт разработан Комитетом по экранированию (TC39) и регулярно обновляется, чтобы вписывать в себя новые идеи и практики.

1. История ECMAScript

ECMAScript 1 (1997): Первая версия стандарта, которая положила начало языковому синтаксису.

ECMAScript 2 (1998): Включала небольшие изменения и исправления.

ECMAScript 3 (1999): Добавила многие функции, используемые и по сей день, такие как исключения, регулярные выражения и улучшенные строки.

ECMAScript 4: Этот проект был отменен из-за сложностей и разногласий среди разработчиков.

ECMAScript 5 (2009): Включила новые возможности, такие как строгий режим (strict mode), методы для массивов и JSON.

ECMAScript 2015 (ES6): Важное обновление, которое добавило поддержку классов, модулей, стрелочных функций, промисов и многих других возможностей.

ECMAScript 2016 (ES7) и последующие версии: Приносили небольшие изменения и улучшения, включая новый оператор `**` для возведения в степень и метод `Array.prototype.includes()`.

2. Возможности ECMAScript

ECMAScript предлагает множество функций, которые делают разработку более эффективной и гибкой:

Стрелочные функции: Позволяют писать функции более лаконичным способом, сохраняя значение `this` из контекста, в котором они были определены.

```
const sum = (a, b) => a + b;
```

Классы: ECMAScript 2015 ввел синтаксис для классов, что упростило создание объектов и наследование.

```
class Shape {  
  constructor(type) {  
    this.type = type;  
  }  
}  
  
class Circle extends Shape {  
  constructor(radius) {  
    super('Circle');  
    this.radius = radius;  
  }  
}
```

Модули: Теперь в ECMAScript можно использовать модули для организации кода, что упрощает его повторное использование.

```
// В файле math.js
export const pi = 3.14;

// В другом файле
import { pi } from './math.js';
```

Промисы и асинхронные функции: Промисы позволяют работать с асинхронным кодом более удобно, а `async/await` делает обработку асинхронных операций более читабельной.

```
async function fetchData() {
  const response = await
  fetch('https://jsonplaceholder.typicode.com/todos');
  const data = await response.json();
  return data;
}
```

Шаблонные строки: Упрощают работу со строками, позволяя встраивать выражения и использовать многострочные строки.

```
const name = 'World';
console.log(`Hello, ${name}!`);
```

Деструктуризация: Позволяет извлекать значения из массивов или объектов в переменные с помощью удобного синтаксиса.

```
const arr = [1, 2, 3];
const [one, two] = arr;
```

```
const obj = { x: 1, y: 2 };  
const { x, y } = obj;
```

3. Заключение

ECMAScript стал основой для разработки не только JavaScript, но и других языков и фреймворков. С каждой новой версией возможности стандарта расширяются, позволяя разработчикам создавать более комплексные и удобные веб-приложения. Важно следить за изменениями в стандарте и адаптировать свои навыки, чтобы оставаться актуальным в быстро развивающемся мире веб-разработки.

Благодаря своей гибкости и мощности, ECMAScript продолжает оставаться важной частью веб-технологий и основой для создания современных интерактивных приложений.

2.3 Node.js

Node.js — это мощная платформа для разработки серверных приложений на JavaScript, основанная на движке V8 от Google. Она позволяет разработчикам создавать масштабируемые сетевые приложения с высокой производительностью. Ниже приведены основы Node.js, включая его ключевые особенности, архитектуру и примеры использования.

1. Архитектура Node.js

Node.js построен на однопоточной модели с неблокирующим вводом-выводом (I/O), что позволяет обрабатывать множество операций одновременно. Основные компоненты архитектуры включают:

Событийный цикл (Event Loop): Сердце Node.js, которое управляет асинхронными операциями. Он обрабатывает все запросы и события, позволяя Node.js оставаться отзывчивым.

Модульная система: Каждый файл в Node.js считается модулем. Это облегчает организацию кода и его повторное использование.

NPM (Node Package Manager): Стандартный менеджер пакетов для Node.js, который позволяет легко устанавливать и управлять сторонними библиотеками.

2. Работа с модулями

Node.js обладает встроенной модульной системой. Можно использовать как встроенные модули (например, `'fs'` для работы с файловой системой), так и сторонние модули.

3. Асинхронные операции и промисы

Node.js использует асинхронные операции, чтобы не блокировать основной поток выполнения. Промисы и `async/await` позволяют работать с асинхронным кодом более удобно и читабельно.

4. Express.js

Express.js — это популярный веб-фреймворк для Node.js, который упрощает создание веб-приложений и API.

5. Заключение

Node.js является отличным инструментом для разработчиков, позволяя создавать высокопроизводительные и масштабируемые приложения. Благодаря своей асинхронной природе и поддержке модулей, Node.js подходит для разработки как простых веб-сайтов, так и сложных серверных приложений.

2.4 REST Api

REST (Representational State Transfer) — это архитектурный стиль для проектирования сетевых приложений, который используется для создания веб-сервисов. Вот основные принципы и технологии, лежащие в основе REST API:

Основные Условия REST API

1. Клиент-серверная архитектура:

- REST разделяет клиентскую и серверную части, что позволяет им развиваться независимо друг от друга. Клиент отвечает за пользовательский интерфейс и взаимодействие с пользователем, а сервер — за обработку запросов, управление данными и бизнес-логику.

2. Статус без сохранения состояния:

- Каждый запрос от клиента к серверу должен содержать всю необходимую информацию для его обработки, что упрощает масштабируемость и снижает нагрузку на сервер. Сервер не хранит информацию о состоянии клиента между запросами.

3. Кешируемость:

- HTTP-запросы могут быть кешируемыми или нет, что позволяет уменьшить нагрузки на сервер и улучшить производительность. Сервер может пометить определенные ответы как кешируемые, предоставляя клиентам возможность хранить их.

4.Единообразие интерфейса:

- REST API должен иметь единый интерфейс, который упрощает взаимодействие между клиентом и сервером. Это обычно достигается через использование стандартных HTTP методов.

5. Упрощение:

- Комплексные системы должны быть построены по принципу интеграции простых компонентов, что облегчает их понимание и использование.

6. Иерархия ресурсов:

- Всевозможные ресурсы (например, данные о пользователях, продуктах) могут быть представлены в виде URL, и эти ресурсы могут быть сгруппированы и организованы в иерархическую структуру.

Основные HTTP методы

REST API использует стандартные методы HTTP для выполнения операций:

GET: Получить данные с сервера.

POST: Создать новый ресурс на сервере.

PUT: Обновить существующий ресурс или создать его, если он не существует.

PATCH: Частично обновить ресурс.

DELETE: Удалить ресурс с сервера.

Форматы обмена данными

REST API чаще всего использует форматы JSON или XML для передачи данных. JSON (JavaScript Object Notation) является более легковесным и удобным для работы с современными веб-приложениями.

Пример структуры REST API

URL-структура

Предположим, у нас есть ресурс "пользователи":

GET /users: Получить список всех пользователей.

GET /users/:id: Получить пользователя с определенным ID.

-POST /users: Создать нового пользователя.

PUT /users/:id: Обновить данные пользователя с определенным ID.

DELETE /users/:id: Удалить пользователя с определенным ID.

Применение REST API

REST API находит широкое применение в различных областях, включая:

- Веб-приложения и мобильные приложения: Обмен данными между клиентом и сервером.

- Интеграция сервисов: Позволяет различным системам взаимодействовать друг с другом.

- Микросервисная архитектура: REST API часто используется для связи между микросервисами в рамках одной системы.

Заключение

REST API предоставляет удобный и гибкий способ для проектирования клиент-серверных приложений. Его архитектурные принципы и стандартизированный подход делают его идеальным выбором для разработки современных веб-сервисов.

2.5 ООП на основе JavaScript

Объектно-ориентированное программирование (ООП) — это парадигма программирования, которая основывается на концепции "объектов", которые могут содержать как данные, так и код: данные в виде полей (обычно известных как атрибуты или свойства), и код в виде процедур (часто известных как методы). JavaScript, как язык, поддерживающий ООП, предлагает несколько способов реализации объектов и их взаимодействия.

Основные концепции ООП на JavaScript

Объекты: Объект в JavaScript представляет собой коллекцию свойств, которые могут включать как данные, так и функции. Объекты можно создавать разными способами:

Конструкторы: Конструкторы — это специальные функции, которые создаются для создания объектов. Они обычно используют `this`` для установки свойств создаваемого объекта.

Наследование: Наследование в JavaScript можно реализовать через цепочку прототипов. Это позволяет одному объекту использовать свойства и методы другого.

ES6 Классы: С выходом ECMAScript 2015 (ES6) в JavaScript появились классы, которые делают синтаксис более понятным и приближают его к классическим языкам ООП.

Инкапсуляция : это практика скрытия внутреннего состояния объекта и предоставления публичного интерфейса. Хотя JavaScript не имеет строгой инкапсуляции, это можно реализовать с помощью замыкания или символов.

Преимущества ООП в JavaScript

- 1. Повторное использование кода:** Используя наследование, можно создавать новые объекты на основе существующих, минимизируя дублирование кода.
- 2. Упрощение разработки:** Объекты позволяют группировать данные и методы, что делает код более организованным.
- 3. Лучшая поддержка и расширяемость:** Новые функции можно добавлять к объектам, не затрагивая существующий код.

Заключение

ООП в JavaScript предоставляет мощные инструменты для работы с данными и логикой. С пониманием основных принципов, таких как создание объектов, наследование, инкапсуляция и использование классов, разработчики могут создавать более структурированные и поддерживаемые приложения. Использование современных возможностей языка, таких как ES6 классы, упрощает работу с объектами и делает код более читабельным и понятным.

2.6 Основные принципы адаптивного сайта

Адаптивный дизайн — это метод веб-разработки, позволяющий создавать сайты, которые автоматически адаптируются к различным размерам экранов и устройствам. Основной целью является обеспечение оптимального пользовательского опыта вне зависимости от устройства, с которого пользователи получают доступ к сайту. Вот основные принципы адаптивного сайта:

Гибкие сетки (Flexible Grid Layouts)

- Используются относительные единицы измерения, такие как проценты и `em`, вместо фиксированных значений в пикселях. Это позволяет элементам сайта изменять свои размеры в зависимости от ширины экрана.

- Например, вместо установки ширины элемента в 300 пикселей, можно указать ``width: 50%``, чтобы элемент занимал половину доступного пространства.

Медийные запросы (Media Queries)

- Это CSS-технология, позволяющая применять стили в зависимости от размеров устройства. С помощью медийных запросов можно изменять оформление элементов для различных экранов.

Гибкие изображения (Flexible Images)

- Изображения и медиа-контент должны быть восприимчивыми. Это достигается с помощью установки максимальной ширины изображения на 100%, чтобы они не превышали ширину родительского блока.

Удобство взаимодействия (Touch-Friendly)

- Элементы интерфейса должны быть удобными для взаимодействия как на сенсорных, так и на несенсорных устройствах. Это означает, что

кнопки, ссылки и другие интерактивные элементы должны быть достаточно большими и удобно расположенными, чтобы их можно было легко нажимать на экранах разных размеров.

Приоритет контента (Content Prioritization)

- На адаптивном сайте важно понять, какой контент является наиболее важным для пользователя. На мобильных устройствах лучше размещать наиболее критическую информацию вверху страницы, избегая перегрузки интерфейса.

Тестирование на разных устройствах

- Для обеспечения оптимального отображения и функциональности сайта необходимо проводить тестирование на различных устройствах и экранах. Это включает в себя использование инструментов для эмуляции разных размеров экрана и реальное тестирование на мобильных телефонах и планшетах.

Оптимизация загрузки (Load Optimization)

- Оптимизация кода, изображений и других ресурсов сайта важна для быстрого времени загрузки, особенно на мобильных устройствах с менее стабильным интернет-соединением. Использование небольших файлов изображений, минимизация CSS и JavaScript, а также использование кеширования могут значительно ускорить загрузку страниц.

Дизайн на основе узоров (Pattern-Based Design)

- Использование привычных узоров и элементов дизайна помогает пользователям интуитивно понять, как взаимодействовать с сайтом. Например, основные принципы навигации, шрифты, кнопки и другие элементы интерфейса должны быть хорошо известны и легко узнаваемы.

Заключение

Адаптивный сайт — это не просто эстетичный дизайн; это также функциональность и удобство использования. Следуя этим принципам, можно создать динамичный и отзывчивый веб-сайт, который обеспечивает положительный пользовательский опыт на всех устройствах.

\

2.6 Проверка сайта, его функциональности и отображения на разных устройствах

Проверка сайта на функциональность и отображение на разных устройствах — это важный этап в веб-разработке, который позволяет обеспечить высокое качество пользовательского опыта. Вот шаги и инструменты, которые можно использовать для этой проверки:

Проверка функциональности сайта

Функциональное тестирование включает в себя проверку всех функций сайта, чтобы убедиться, что они работают как задумано.

Проверка ссылок: убедиться, что все ссылки работают и ведут на правильные страницы.

Формы и входные данные: проверить работу всех форм (регистрация, вход, обратная связь). Убедиться, что валидация работает корректно и сообщения об ошибках отображаются.

Кнопки и интерактивные элементы: убедиться, что все кнопки и интерактивные элементы (меню, выпадающие списки) работают корректно.

Функциональность на JavaScript: проверить работу всех скриптов (например, AJAX-запросов), чтобы убедиться, что асинхронные операции также выполняются без ошибок.

Адаптивность сетевого соединения: проверить, как сайт работает при различных скоростях интернет-соединения.

Проверка отображения на разных устройствах

Важно тестировать сайт на различных устройствах и разрешениях экрана, чтобы убедиться, что он выглядит и функционирует хорошо.

Эмуляция устройств:

- Можно использовать инструменты разработчика в браузере (например, Chrome DevTools), чтобы эмулировать различные устройства и разрешения экрана.

- Включить возможность просмотра сайта в режимах мобильных и планшетных устройств.

Физические устройства: если возможно, нужно протестировать сайт на реальных устройствах (смартфоны, планшеты, ноутбуки) с различными операционными системами (iOS, Android, Windows, Mac).

Масштабирование и прокрутка: проверить, как сайт масштабируется и прокручивается. Убедиться, что элементы интерфейса не выходят за пределы видимой области экрана.

Кросс-браузерное тестирование: убедиться, что сайт корректно отображается во всех популярных браузерах (Chrome, Firefox, Safari, Edge). Каждый браузер может интерпретировать CSS и JavaScript по-разному.

Теоретическая и практическая главы

Основная часть составляет 80% от всего объёма написанного. В нём последовательно рассказывается, как были решены вопросы, перечисленные во вступлении, какие цели достигнуты и с каким результатом.

Содержит 2 главы — теоретическую и практическую, где автор выражает свою позицию относительно гипотез.

В практической части студент должен пошагово описать:

- как он делал проект;
- какие инструменты использовал в работе;
- на каком этапе использовал инструменты и как они помогли в достижении цели работы.

В основной части должны быть практические примеры, результаты и выводы после каждого элемента исследования.

Заключение

В заключение необходимо включить следующее:

1. Краткие и ёмкие теоретические и практические выводы, которые были получены во время анализа теоретической базы и практического исследования.
2. Оценка проведённого исследования, описание его результатов.
3. Практическая значимость работы, рекомендации и планы на дальнейшие исследования.
4. Общий итог — достижение цели, выполнение задач, доказательство гипотезы.
5. Предложения по совершенствованию объекта исследования.

Список используемой литературы

Здесь нужно будет указать список используемой литературы, ссылки на все ресурсы, которые нужны были для создания проектной работы.

Основные правила оформления использованной литературы и ресурсов:

1. Каждый источник упоминается единожды, независимо от того, насколько часто на него ссылаются.
2. Список литературы оформляется в алфавитном порядке по фамилии автора, сначала русскоязычная литература, затем иностранная, далее интернет-сайты.
3. Библиографическая запись обязательно включает:
 - Фамилию автора или фамилии их группы, инициалы (при наличии).
 - Название статьи, книги, справочника, закона, иного документа.
 - Населённый пункт, в котором был издан источник, наименование издательства.
 - Год публикации.
 - Число страниц.

Пример

- *Книга: Автор. Название книги. Город: Издательство, Год.*
- *Статья: Автор. "Заголовок статьи." Название журнала Том, номер (Год): страницы.*

Приложения

В **приложения** обычно входят артефакты, получившиеся в процессе создания проекта:

1. Объёмные графики и таблицы, которые не помещаются на лист А4.
2. Длинные математические формулы и расчёты по ним.
3. Характеристики аппаратуры, которая использовалась для проведения исследования.
4. Авторские методики.
5. Вспомогательный материал: тесты, карточки, схемы, рисунки.
6. Материалы, полученные на предприятии: отчёты, прочие документы.