Geekbrains

Разработка и поддержка адаптивного сайта магазина гитар на основе JavaScript.

Программа: Программист Frontend-разработчик

Замчалин Михаил Николаевич

Москва

2024

Содержание

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 JavaScript.

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

Какую проблему решает: случается, что заказчик из эконом - сегмента запрашивает 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: <b>, <i>, <u>, , и другие.
Заголовки: <h1> до <h6>
Абзацы и разрывы строк: <p>,
Списки: <ul>, <ol>, <li>, <dl>, <dt>, <dd>
Ссылки: <a>
Картинки: <img>

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.

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.

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, облегчает 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 дом-элементами. Это позволяет обновлять контент без перезагрузки страницы.

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.

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.

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;
```

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 важною 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.

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.

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.

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.

Express.js

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

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: убедится, что все ссылки работают и ведут на правильные страницы.

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

Кнопки и интерактивные элементы: убедится, 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.

Глава 3

Практическая часть проекта

3.1 Разработка веб-страниц без использования макета

В этой секции будет описан процесс создания веб-страниц для магазина гитар. В начале необходимо определить структуру сайта. Для магазина могут быть предусмотрены следующие страницы:

Главная страница (Приложение 1)

Разработка веб-страницы для интернет-магазина гитар — это многогранный процесс, включающий проектирование структуры, выбор технологий, написание кода и последующую тестировку. В данном случае, я опишу этапы, которые были проработаны для создания предоставленного HTML-кода главной страницы.

Определение структуры страницы

Перед написанием кода было определено, какие элементы должны присутствовать на главной странице магазина:

- Заголовок (header), который включает в себя логотип и навигационное меню.
- Основной контент (main), который содержит заголовок и изображения с кратким описанием доступных категорий гитар.
- Футер (footer), где размещены copyright, дополнительные ссылки и социальные иконки.

Выбор технологий

В качестве технологии разработки была выбрана HTML для разметки контента. Для стилей применялся CSS, который подключается через

внешний файл (`style.css`). Также добавлены ссылки на шрифты Google, что обеспечивает более разнообразный и современный внешний вид текста.

Написание HTML-кода

Заголовок страницы

В начале кода указан стандартный DOCTYPE и языковые настройки для страницы. В секции `<head>` добавлены метатеги для обеспечения совместимости и настройки отображения на мобильных устройствах. Далее указаны ссылки на необходимые шрифты и файл стилей.

Структура заголовка

В секции `<header>` был добавлен логотип и навигационное меню <nav>. Каждый элемент меню представляет собой ссылку, что позволяет пользователю легко перемещаться по различным разделам магазина.

Основной контент

Основной контент страницы размещается в секции `<main>`. Здесь добавлены изображения и текст, которые представляют различные категории гитар: электрогитары, бас-гитары и акустические гитары. Каждый раздел содержит заголовок и изображение с соответствующим описанием.

Футер

В футере размещена информация о праве собственности, ссылки на остальные страницы сайта и иконки социальных сетей. Это помогает пользователям находить нужную информацию и подключаться к социальным сетям магазина.

Так, был получен функциональный и структурированный HTML-код для главной страницы интернет-магазина, который позволяет пользователям легко навигировать по контенту и знакомиться с предложениями магазина.

Каталог товаров (Приложение 2)

На первом этапе была определена основная цель страницы: предоставить пользователям доступ к каталогу товаров (гитар) с удобным интерфейсом для навигации и фильтрации. Основные элементы, которые необходимо было включить на страницу, включали: навигационное меню, заголовок каталога, фильтры для сортировки и отображения товаров, а также футер с контактной информацией и ссылками на социальные сети.

Создание структуры HTML

Первым шагом в написании кода стало создание базовой структуры HTML-документа. Использование элементов `<header>`, `<main>`, и `<footer>` служит важной практикой для логического разделения контента и улучшения восприятия страницы как пользователями, так и поисковыми системами.

Формирование основного контента (Main)

В основной области контента был создан раздел каталога, содержащий заголовок и блоки фильтров. Для удобства использования были использованы элементы `<details>` и `<summary>`, что дает пользователям возможность раскрывать содержимое фильтров. Каждый фильтр представлен ссылками, которые ведут к различным

категориям и способам сортировки товаров. Так же выделен блок для размещения карточек товаров из базы данных.

Создание фильтров и сортировки

Каждый фильтр для выбора товаров был оформлен с использованием SVG-иконок для повышения визуального восприятия. Это не только улучшает внешний вид страницы, но и делает интерфейс более интуитивно понятным.

Корзина (Приложение 3)

Создание HTML-кода для страницы корзины интернет-магазина гитар подразумевает шаги, направленные на построение понятного и функционального интерфейса для пользователей. Ниже выделены основные этапы и описания ключевых элементов кода.

Проектирование структуры страницы

Этап начальной разработки заключается в определении структуры страницы. Основной акцент делается на логичное распределение информации и обеспечении удобной навигации. В центре внимания находятся три основные части веб-страницы:

- Заголовок (header)
- Основной контент (main)
- Футер (footer)

Эти секции помогают поддерживать ясное и интуитивно понятное расположение элементов на странице.

Основной контент страницы

Разработка основной части страницы сосредоточена на корзине покупок, где пользователи могут видеть и управлять выбором товаров:

- Использование элемента для определения основной области контента.
- Создание заголовка для идентификации пользователя, что он находится на странице "Корзина".
- Включение секции для отображения товаров в корзине и их управления через блок
- Добавление функциональности для отображения общей суммы к оплате и кнопки "Оплатить" в блоке

Карточка продукта (Приложение 4)

Создание HTML-кода для страницы продукта интернет-магазина гитар подразумевает шаги, направленные на построение понятного и функционального интерфейса для пользователей. Главные части сайта header и footer остаются неизменными. А вот над блоком main была проведена работа. Была добавлена структура распределения информации о продукте, загружаемой из базы данных. Так же добавлена кнопка добавления товара в корзинк.

О нас (Приложение 5)

Создание HTML-кода для страницы о нас состояло из основных блоков header и footer. В блок main было добавлено 2 блока. Первый блок рассказывает немного о магазине. Второй блок содержит блок с контактной информацией. Так же была подключена карта через тег <script> с помощью Api сервиса Яндекс карты, что является удобным вариантом для разработчика и облегчающим возможность найти локацию магазина для клиента.

Заключение

После написания кода была проведена тестировка веб-страницы, а так же всех ссылок присутствующих на сайте.

Так, был получен функциональный и структурированный HTML-код интернет-магазина, который позволяет пользователям легко навигировать по контенту и знакомиться с предложениями магазина.

3.2 Подключение CSS к проекту и адаптивная верстка

Для разработки внешнего вида был выбран препроцессор SCSS, что позволило сделать CSS код более структурированным и читабельным. Были использованы функции препроцессора, такие как:

-Импортирования всех файлов в основной.

-Создание переменных с основными цветами и шрифтами для однообразного стиля для всех страниц.

-Вложенность позволяющая читать и править код более комфортно.

Для позиционирования контента на страницах были использованы современные технологии, такие как:

-Flexbox который позволил быстро и удобно расставлять блоки на странице, а также упростить изменение расположения контента на странице в зависимости от размеров экрана, на котором отображается страница

-С помощью технологии Grid были выполнены блоки содержащие карточки товаров на страницах корзина и каталог товаров. Эта технология позволяет лаконично размещать карточки, вне зависимости от количества товаров, которые были добавлены из базы данных.

-Медиа-запросы позволяют менять структуру и стили страницы в зависимости от размера экрана. Ниже приведу некоторые изменения, которые позволили сделать сайт адаптивным для разных девайсов.

. Рассмотрим изменения, внесённые через медиа-запросы:

@media (max-width: 1024px)

- `.fixed` элемент изменяет позицию с `fixed` на `static`.
- Изменяются отступы в `center`, чтобы уменьшить их (до 367px).
- Элементы в `.main__title` уменьшают свои начальные параметры:
  - `_img` ширина: 400px.
  - `_title` шрифт: 40px, жирность: 800.

- `_subtitle` шрифт: 18px, жирность: 700.
- `.collection` уменьшается сетка и отступы:
- Грид: `repeat(3, 1fr)` с меньшим `gap`.
- Ширина `.collection__type`: 200px.

2@media (max-width: 769px)

- `.header` навигация сокращает `gap` до 30px, а размер шрифта `.header__nav_link` до 16px.
- `center` более ужимает отступы до 330px.

3@media (max-width: 450px)

- `center` уменьшает отступы до 171px.
- `.header`:
- `gap` в `__nav` до 20px.
- Размер лого уменьшается до 40px.
- Навигационные ссылки (`__nav_link`) используют размер шрифта в 14px.
- `.main__title` переключается на колонную компоновку:
- `_img` уменьшены до 200px.
- `_text` меняет границу с левой на верхнюю и центрирует текст.
- `_title` и `_subtitle` уменьшают размеры шрифтов.
- `.collection`:
- Грид становится `repeat(1, 1fr)`, изменяя все карточки на одинаковую ширину (200px).
- `.footer` изменяет направление на колонну и размеры шрифтов уменьшаются.

Медиа-запросы в этой SCSS увеличивают отзывчивость сайта, изменяя его компоненты, чтобы они лучше подходили для различных размеров

экранов. Важно отметить, что эти изменения позволяют сайту оставаться функциональным и удобным для пользователя, независимо от того, с какого устройства он посещается.

3.3 Подключение JS к проекту. Разработка функциональности сайта.

Разработка JavaScript-приложения для загрузки и отображения списка продуктов в каталоге.

Разработка JavaScript-приложения для загрузки и отображения списка продуктов в каталоге включала ряд этапов, начиная с планирования и заканчивая реализацией и отладкой кода. Вот подробное описание процесса разработки данного проекта:

Этап 1: Планирование

На этапе планирования были определены основные цели и задачи:

1. Чтение данных. Необходимо было загрузить данные о продуктах из файла `data.json`, используя `fetch`, и обрабатывать возможные ошибки запроса.
2. Отображение продуктов. Данные должны отображаться на веб-странице в удобном для пользователя формате.
3. Добавление в корзину. Пользователи должны иметь возможность добавлять товары в корзину покупок.
4. Фильтрация и сортировка. Реализовать возможность фильтрации продуктов по типу и сортировки по цене.
5. Хранение состояния. Использование `localStorage` для хранения состояния корзины товаров.

Этап 2: Реализация

Код начинается с функции `loadProducts`, которая отвечает за загрузку данных:

- Используется `fetch` для получения данных из файла `data.json`.
- При успешной загрузке данные сохраняются и отображаются, а также инициализируются фильтры и сортировка.
- В случае ошибки выводится сообщение в консоль.

Функция `displayProducts` отвечает за визуальное представление списка товаров:
- Создается HTML-разметка для каждого продукта.
- Добавляются кнопки для добавления товаров в корзину.
- Используются обработчики событий, чтобы добавлять товары в корзину при нажатии на кнопку.

Работа с корзиной
Для управления корзиной реализована функция `addToCart`, которая:
- Проверяет наличие товара в `localStorage`.
- Добавляет товар в корзину, если его там нет.
- Обновляет интерфейс кнопки, чтобы пользователь знал, что товар добавлен.

Фильтрация и сортировка
- Фильтрация осуществляется через функцию `filterProductsByType`, которая отбирает продукты по выбранному типу.

- В `initializeFilters` задаются обработчики событий, чтобы при выборе фильтра обновлялся список товаров на странице.
- Сортировка осуществляется через функции `sortProductPlus` и `sortProductMinus`, которые изменяют порядок отображения товаров по возрастанию и убыванию цены соответственно.
- `initializeSort` содержит логику для обработки выбора сортировки и обновления интерфейса.

Этап 3: Тестирование и отладка

- Был проведен ряд тестов, чтобы убедиться, что данные загружаются корректно и отображаются в соответствии с выбранными настройками фильтрации и сортировки.
- Проверена совместимость с различными браузерами и исправлены возможные баги.
- Сделаны улучшения пользовательского интерфейса, такие как изменение текста кнопки на "Товар в корзине" для лучшей UX.

Разработка кода для загрузки и отображения деталей товара

Разработка кода для загрузки и отображения деталей товара включает несколько важных этапов, начиная от планирования до реализации и отладки. Здесь описан процесс, который следился на каждом из этапов для создания функций `loadProductDetails`, `displayProductDetails`, и `addToCart`.

Загрузка деталей товара

Функция `loadProductDetails` начинается с извлечения параметров из URL. Использование `URLSearchParams` позволяет получить ID товара, который передается через адресную строку.

Если ID не найден, выводится информативное сообщение в консоли, и функция прекращает выполнение. Далее следует запрос к файлу `data.json` с использованием `fetch`.
Если ответ не удачен (например, плохое соединение или файл не найден), генерируется ошибка. При успешном получении данных происходит поиск товара по ID:
Если продукт найден, вызывается функция для его отображения.

Отображение деталей товара

В функции `displayProductDetails` происходит создание и представление HTML-разметки для выбранного товара:
В разметке выводится изображение, название и описание товара, а также кнопка для добавления его в корзину.
После этого происходит проверка состояния товара в корзине. Если товар уже находится в корзине, изменяется текст кнопки и она блокируется:

Если товар ещё не добавлен в корзину, прикрепляется обработчик событий для кнопки. При щелчке на кнопку вызывается функция `addToCart`, и состояние кнопки обновляется:

Добавление товара в корзину

Функция `addToCart` отвечает за логику добавления товара в корзину, используя `localStorage` для сохранения состояния корзины. Курсор обращается к локальному хранилищу и проверяет, существует ли уже товар:

Если товары уже есть в корзине, увеличивается их количество. Если нет, товар добавляется в корзину с количеством 1:

После завершения обновления корзины она сохраняется обратно в `localStorage`.

Разработка кода для управления корзиной

Разработка кода для управления корзиной товаров на веб-сайте предполагает несколько ключевых шагов и принципов проектирования, которые были соблюдены в процессе написания данного кода. Ниже описан этот процесс и обоснования принятых решений.

Для хранения информации о корзине использовался объект `cart`, который получает данные из локального хранилища браузера с использованием `localStorage`. Это позволяет сохранять состояние корзины между сессиями пользователя. За основу был взят JSON-формат, чтобы упростить управление данными.

Функция отображения продуктов

Следующий шаг заключался в создании функции `displayProducts`, которая принимает массив с товарами и отображает их на странице.

Очистка существующего содержимого: перед тем как начать добавлять новые карточки с продуктами, функция очищает контейнер для продуктов.

Создание карточек: для каждого товара создается `div` с изображением, названием, описанием, ценой и кнопкой удаления. Внутри созданного элемента используется шаблонная строка для динамического наполнения данных.

Обновление цены: после отображения продуктов вызывается функция `updatePrice`, которая рассчитывает и отображает общую стоимость товаров в корзине.

Подсчет общей стоимости

Функция `updatePrice` принимает массив товаров из корзины и вычисляет общую стоимость с помощью метода `reduce`, который суммирует цену каждого товара в массиве.

Удаление товара

Функция `deleteProduct` отвечает за логику удаления товара из корзины. Она фильтрует массив товаров, исключая тот, который необходимо удалить, и обновляет локальное хранилище и отображение.

В итоге был создан интерактивный интерфейс корзины, который позволяет пользователям легко видеть и управлять своими продуктами. Код был организован таким образом, что делает его зрительно понятным и легко поддерживаемым. Каждая функция разбита на отдельные задачи, что упрощает процесс отладки и тестирования.

Заключение

Разработка была завершена успешно, предоставив пользователям возможность эффективного просмотра, фильтрации, сортировки и добавления товаров в корзину. Реализация данного проекта позволила глубже понять принципы работы с JavaScript, DOM и локальным хранилищем.

3.4 Проверка сайта на валидность и функциональность.

Проверка сайта на валидность и функциональность — это важный этап разработки веб-приложений, который позволяет убедиться, что сайт соответствует установленным стандартам и отвечает требованиям пользователей. Этот процесс можно разбить на несколько ключевых этапов, каждый из которых включает в себя определенные методы и инструменты.

Определение требований и стандартов

Перед тем как начинать проверку, важно определить, какие именно требования и стандарты необходимо учитывать. Это могут быть:

- Веб-стандарты: HTML, CSS, JavaScript, доступность, SEO и т.д.
-Функциональные требования: ожидаемое поведение функциональности сайта, например, работа форм, взаимодействие с пользователями и т.д.
-Нефункциональные требования: производительность, безопасность, кросс-браузерная совместимость.

Валидация кода

HTML и CSS

-Проверка HTML: Использовался [W3C Markup Validation Service](https://validator.w3.org/) для проверки HTML-кода на наличие ошибок. Это помогло удостовериться, что код соответствует стандартам.
-Проверка CSS: Аналогично, использовался [W3C CSS Validation Service](https://jigsaw.w3.org/css-validator/) для валидации CSS.

Функциональное тестирование

-Ручное тестирование: Просмотрены все страницы сайта, проверяны кнопки, формы и другие интерактивные элементы, чтобы убедиться, что все работает так, как задумано.

Примеры функциональных тестов:

- Проверка, корректно ли обрабатываются формы (например, в случае неправильного ввода).
- Проверка, что все ссылки работают и ведут на правильные страницы.
- Проверка, что динамические элементы (например, карусели, фильтры) корректно работают.

Тестирование на совместимость

- Проверка сайт на разных браузерах (Chrome, Firefox, Safari, Edge) и на различных устройствах (мобильные телефоны, планшеты, настольные компьютеры). Это позволило удостовериться, что сайт выглядит и работает одинаково на всех платформах.

Проверка сайта на валидность и функциональность — это многоступенчатый процесс, который требует внимательности и тщательности. Последовательное выполнение каждого из этапов от валидации кода до функционального тестирования, проверки совместимости и безопасности гарантирует, что ваш сайт будет работать корректно, будет доступен для всех пользователей и будет соответствовать современным стандартам и требованиям.

Заключение

В ходе работы над дипломной проектом на тему "Разработка и поддержка адаптивного сайта магазина гитар на основе JavaScript" была успешно реализована концепция создания современного веб-ресурса, который отвечает требованиям пользователей и внедряет передовые технологии веб-разработки.

Основное внимание в проекте было уделено созданию адаптивного дизайна, что позволяет обеспечить корректное отображение и удобное взаимодействие с сайтом на различных устройствах и экранах. Использование JavaScript как основного инструмента для реализации интерактивных элементов позволило значительно улучшить пользовательский опыт, повысив скорость загрузки страниц и отзывчивость интерфейса.

В ходе разработки были применены современные технологии, что не только ускорило процесс разработки, но и обеспечило гибкость в дальнейшем сопровождении и улучшении проекта. Реализованные функции поиска и фильтрации товаров делают процесс покупок более удобным и интуитивно понятным для пользователей.

В результате практического применения полученных знаний и навыков в разработке доступного, функционального и привлекательного сайта были достигнуты поставленные цели. Этот проект является основой для дальнейшего развития и реализации новых функций, таких как интеграция с системами доставки, внедрение системы отзывов и рейтингов товаров, а также расширение функционала личного кабинета пользователя.

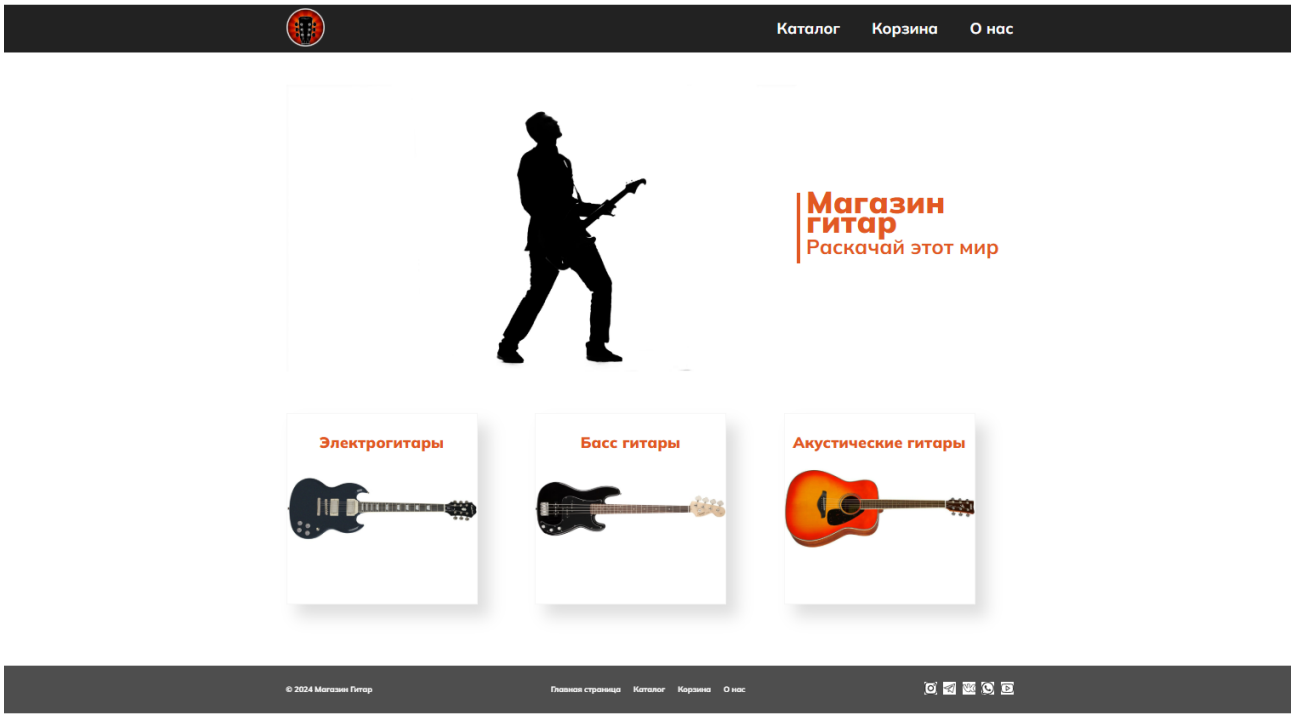
Являясь завершением работы, данный дипломный проект демонстрирует важность создания качественных веб-приложений, адаптированных под нужды пользователя, что, безусловно, имеет большое значение в условиях современного рынка электронной коммерции. Надеемся, что данный сайт станет не только успешным коммерческим проектом, но и полезным ресурсом для всех любителей гитар.

Список используемой литературы

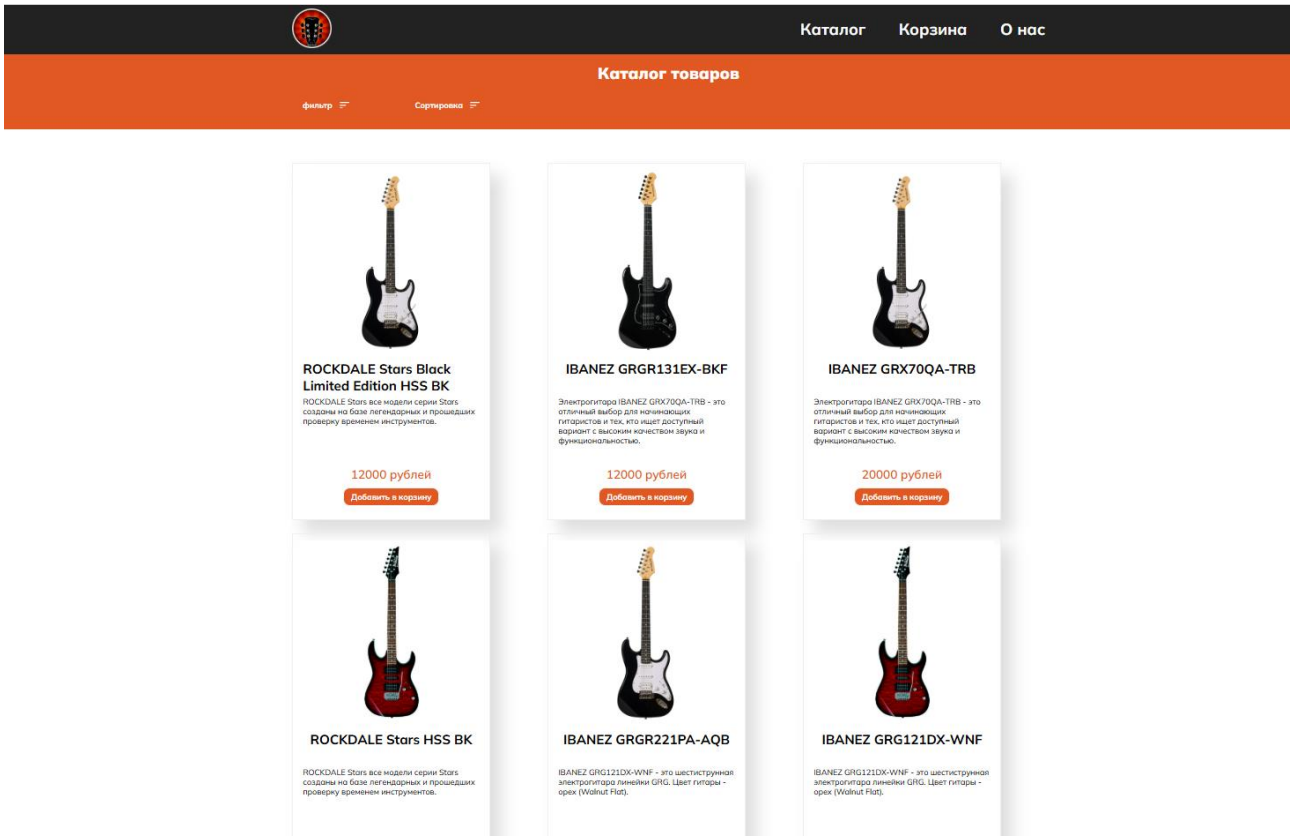
1. Драганов, Д. И. (2021). Основы веб-разработки и адаптивного дизайна. Москва: Издательство.
2. Котлер, Ф. (2020). Основы маркетинга. Санкт-Петербург: Издательство.
3. Ребекка Мерл (2019). JavaScript для веб-дизайнеров. Москва: Издательство.
4. Сайт официальной документации SCSS: https://sass-lang.com/
5. Документация HTML: https://developer.mozilla.org/ru/docs/Web/HTML
6. Документация JavaScript: https://developer.mozilla.org/ru/docs/Web/JavaScript
7. Обучающие материалы портала GeekBrains: https://gb.ru/

Приложения

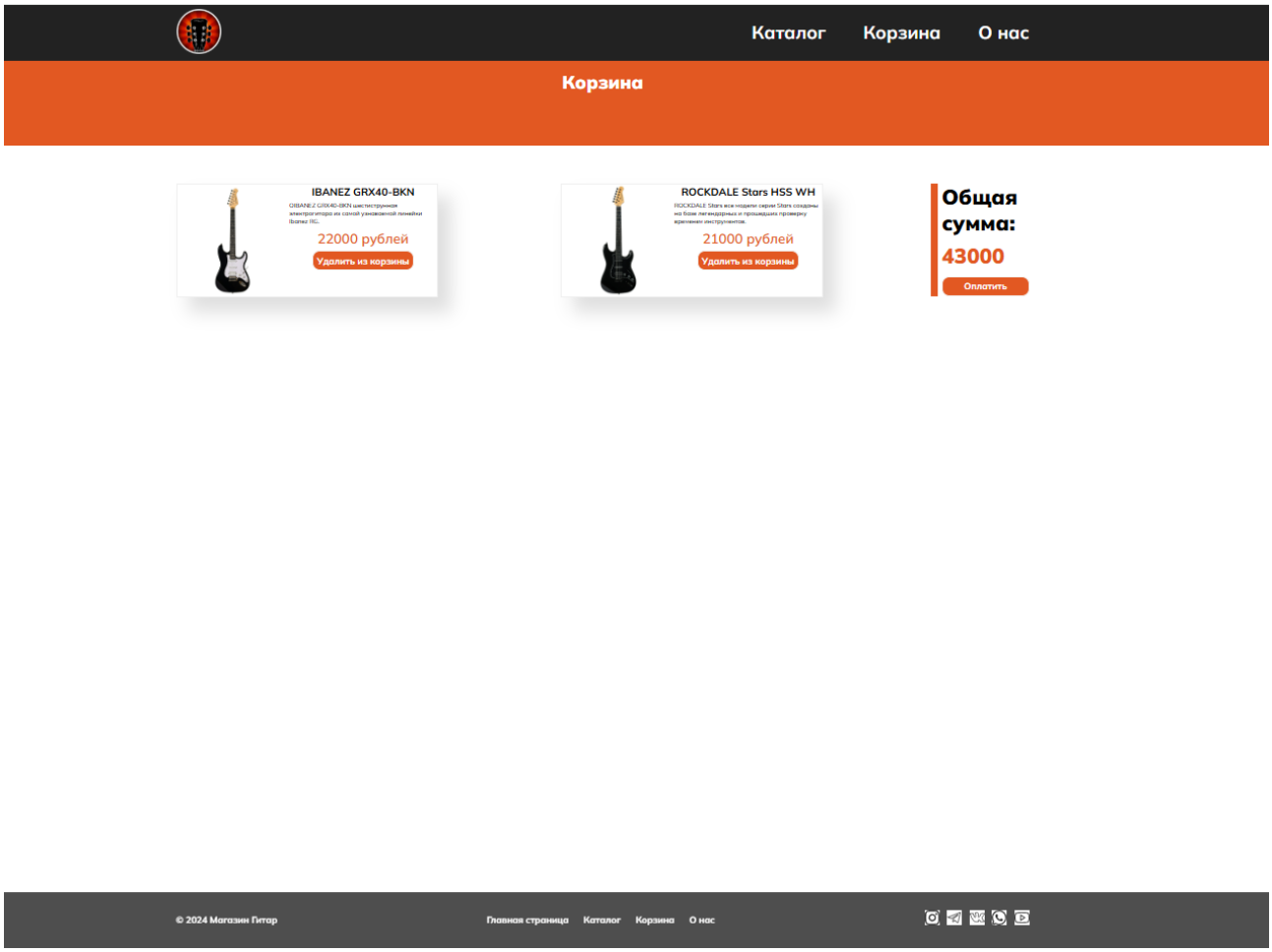
Приложение 1



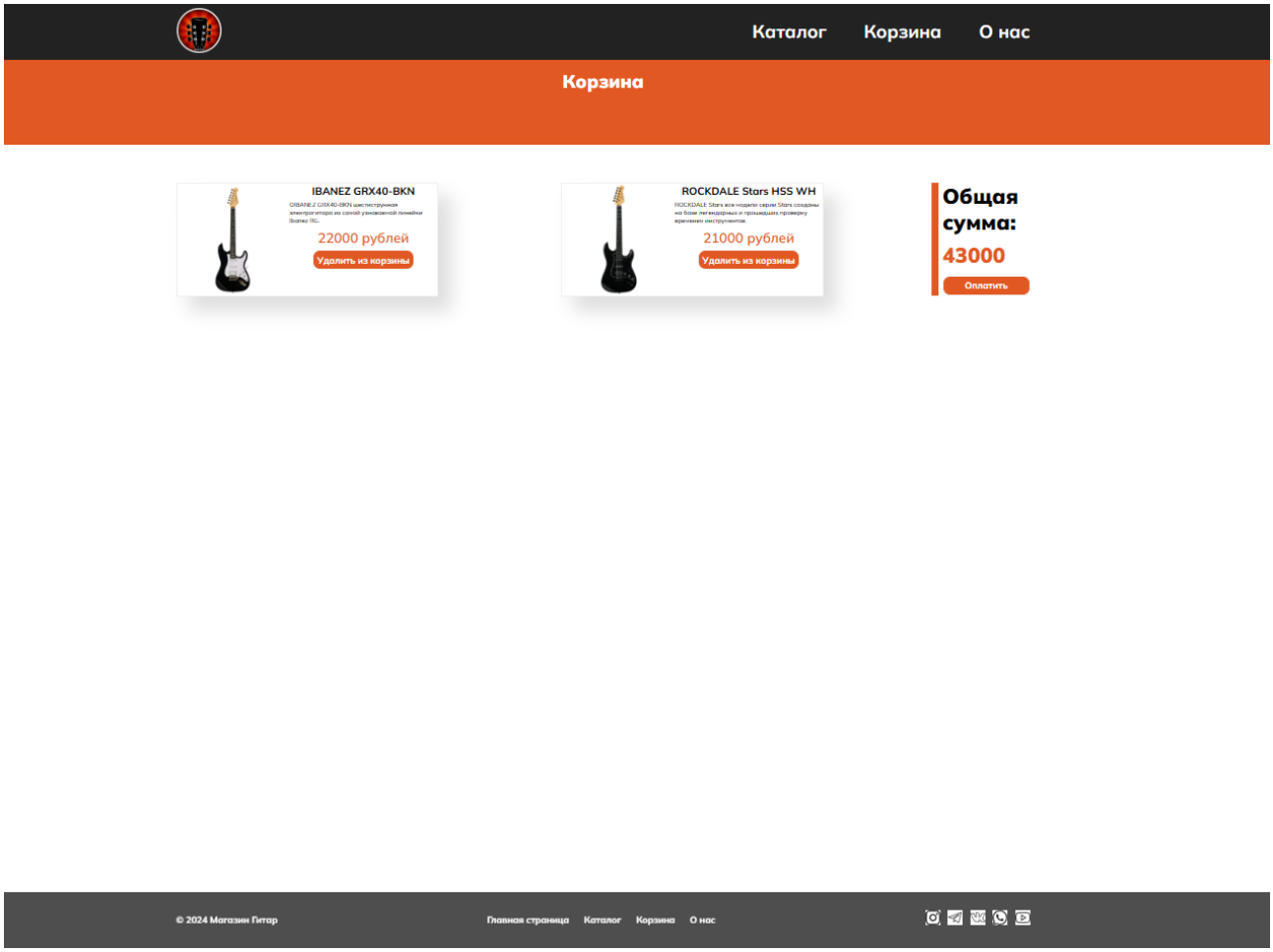
Приложение 2



Приложение 3



Приложение 4



Приложение 5