МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРОГРАММНАЯ РЕАЛИЗАЦИЯ WEB-ПРИЛОЖЕНИЯ «МУЗЫКАЛЬНЫЙ МАГАЗИН»

Студент _____ _Сергеев Александр Сергеевич_
(подпись, дата) (ФИО полностью)

Руководитель _____ _____Кольчугина Е.А. _____
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ _____Такташкин Д.В._____
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ _____Козлов А.Ю._____
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ ____Попова Н.А._____
(подпись) (фамилия, инициалы)

Пенза, 2023

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет Вычислительной техники

Кафедра МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«__»_____2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Сергеев Александр Сергеевич гр.19ВП1 факультета ВТ
2.Тема работы «Программная реализация web-приложения «Музыкальный магазин»»
Тема утверждена приказом ПГУ №_____от "_____"_____20_____
3. Руководитель работы Кольчугина Е.А.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение:
Предоставление программного обеспечения web-приложения музыкального магазина
Цель программной реализации веб-приложения:
Предоставление пользователям приложения для поддержки работы музыкального магазина
Основные функции веб-приложения:
а) создание и наполнение товаров
б) добавление товаров в корзину
в) воспроизведение аудио звучания музыкальных инструментов
Структуры данных: пользователь, изображение, страница,
Технология разработки: ООП, MVC
Языки программирования: JavaScript, PostgreSQL, HTML5, CCS3

5. Перечень подлежащих разработке вопросов
   1) Анализ предметной области
   2) Анализ функциональных требований
   3) Планирование разработки и оценка бюджета
   4) Проектирование веб-приложения
   5) Реализация
   6) Контроль качества
      - тестирование
      - метрики кода
   7) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания "___"_____20_____

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан факультета _____
(подпись, дата)

Реферат

Пояснительная записка содержит 83 листа, 21 рисунок, 21 таблица, 18 источников, 9 приложений.

АНАЛИЗ, ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА, ТЕСТИРОВАНИЕ, ПЛАНИРОВАНИЕ, ОЦЕНКА БЮДЖЕТА, ВЕБ-ПРИЛОЖЕНИЕ, МУЗЫКАЛЬНЫЙ МАГАЗИН, МЕТОДОЛОГИИ РАЗРАБОТКИ.

Объект исследования – web-приложения музыкального магазина.

Цель работы – осуществление программной реализации web-приложения музыкального магазина

Технология разработки: ООП, JavaScript, React bootstrap, React JS, Axios, React-router-dom, MobX.

В результате выполнения работы было спроектировано и создано веб-приложение интернет магазина музыкальных инструментов. Главным преимуществом продукта является простой интерфейс и функция воспроизведения звучания музыкального инструмента.

Основные технико-эксплуатационные характеристики:

- многофункциональность;
- универсальность;
- совместимость с многими операционными системами;
- удобство пользования.

Данный программный продукт предназначен для использования в качестве вспомогательного инструмента при создании интернет магазина музыкальных инструментов.

Веб-приложение может быть использовано пользователями в качестве инструмента создания музыкального магазина.

Содержание

Введение 7
1 Постановка задачи и анализ требований 8
1.1 Постановка задачи 8
1.2 Обзор web-приложений музыкальных магазинов 9
1.3 Сравнительный анализ аналогов 13
1.4 Анализ функциональных требований 15
1.5 Анализ требований к пользовательскому интерфейсу 24
2 Проектирование web-приложения музыкального магазина 25
2.1 Проектирование пользовательского интерфейса 25
2.2 Организация процесса обработки информации 28
2.3 Структуры хранения данных 30
2.4 Выбор программно-аппаратных средств 33
3 Реализация web-приложения «Музыкальный магазин» 36
3.1 Кодирование web-приложения 36
3.2 Развертывание web-приложения 39
4 Тестирование программного обеспечения 41
4.1 Обоснование выбора тестирования 41
4.2 Тестирование 42
4.3 Метрики кода 47
5 Планирование разработки и оценка бюджета 50
5.1 Календарное и ресурсное планирование 50
5.2 Расчет бюджета 51
Заключение 57
Список использованных источников 59
Приложение А. Подробное описание таблиц базы данных 61
Приложение Б. Листинг кода файла index.js 65
Приложение В. Листинг кода файла deviceController.js 67
Приложение Г. Листинг кода файла userController.js 70
Приложение Д. Листинг кода файла models.js 72

Приложение Е. Листинг кода файла userAPI.js 75
Приложение Ж. Листинг кода файла DeviceAPI.js 77
Приложение И. Распределение трудовых ресурсов 79
Приложение К. Диаграмма Ганта..... 82

Введение

В современном мире интернет активно развивается, создается большое количество различных веб-приложений. Развитие технологий и появление новых устройств, таких как смартфоны и планшеты, ускорило этот процесс и позволило создать огромное количество различных веб-сервисов и приложений, выполняющих разнообразные задачи. На данный момент можно найти множество сервисов, выполняющие любые задачи.

Все современные компании сейчас стараются не отставать от технологий и внедрять свой бизнес в Интернет, чтобы занять лидирующие позиции в бизнесе. Одним из самых важных аспектов конкурентной компании в наше время является наличие web-приложения.

Веб-приложение – это программное обеспечение, которое запускается в веб-браузере. Компании должны обмениваться информацией и оказывать услуги удаленно. Они используют веб-приложения для удобной и безопасной связи с клиентами [1].

Несмотря на многообразие видов веб-приложений, одним из самых актуальных в сфере интернет-торговли является интернет-магазин. Основными преимуществами интернет-магазина в сравнении с обычными является то, что онлайн магазин может включать в себя большее количество товаров и обеспечить потребителей большим объемом информации.

К главным плюсам интернет-магазина так же можно отнести низкую стоимость создания и запуска. Разработка и запуск магазина в Интернете обходится гораздо дешевле, чем открытие обычного магазина.

С каждым годом количество охват пользователей увеличивается, а с ним растет и количество интернет-магазинов. Популярность такого рода торговли обуславливается удобством пользования как со стороны потребителя, так и со стороны владельца бизнеса.

1 Постановка задачи и анализ требований

1.1 Постановка задачи

Целью выпускной квалификационной работы является проектирование и разработка веб-приложения музыкального магазина.

Для достижения поставленной цели были определены следующие задачи:

– подобрать и изучить литературу по проектированию и разработке веб-приложений;

– провести анализ существующих web-приложений музыкальных магазинов;

– определить структуру и содержание веб-приложения музыкального магазина;

– осуществить проектирование пользовательского интерфейса, программных средств и структур данных разрабатываемой системы;

– реализовать веб-приложение музыкального магазина;

– провести тестирование разработанного приложения;

– провести планирование и расчёт бюджета разработки.

Разрабатываемое приложение музыкального магазина обязательно должно иметь следующие характеристики и функции:

– наличие разделения степени доступа пользователей на клиента и администратора;

– наличие административной панели для добавления нового содержимого на сайт (добавление типов, брендов, новых товаров);

– возможность добавление на сайт фото и аудио файлов;

– возможность сортировки товаров по типу и бренду;

– возможность добавления товара в корзину, сохранение информации о содержимом корзин для каждого пользователя;

– возможность воспроизведения аудио файлов на сайте.

1.2 Обзор web-приложений музыкальных магазинов

Для анализа предметной области необходимо определить основные аналоги.

Определение основных аналогов в предметной области помогает понять, какие популярные сервисы существуют, какие возможности уже доступны пользователям, какие функции уже реализованы в других приложениях, а также выявить проблемы и недостатки, которые можно избежать при разработке нового продукта.

В интернете существует множество web-приложений музыкальных магазинов. Самыми популярными из них являются:

– Музторг;
– Jazz-shop;
– Мир музыки.

Рассмотрим подробно их особенности ниже.

Начнем с интернет магазина Музторг.

Музторг – лидер рынка розничной продажи музыкальных инструментов и оборудования в России [2]. Данный сервис является самым популярным среди всех интернет магазинов музыкальных инструментов, обладает наибольшим ассортиментом и функционалом.

Рассмотрим интерфейс сайта Музторг.

При открытии главной страницы сайта можно заметить, что на нем расположено большое количество различных баннеров, мелких надписей и интерактивных элементов.

Каталог представлен в виде большого списка видов музыкальных инструментов, при наведении открывается список типов данного вида. При нажатии на конкретный тип музыкального инструмента открывается отдельная страница с определенными товарами.

Интерфейс главной страницы сайта Музторг представлен на рисунке 1.

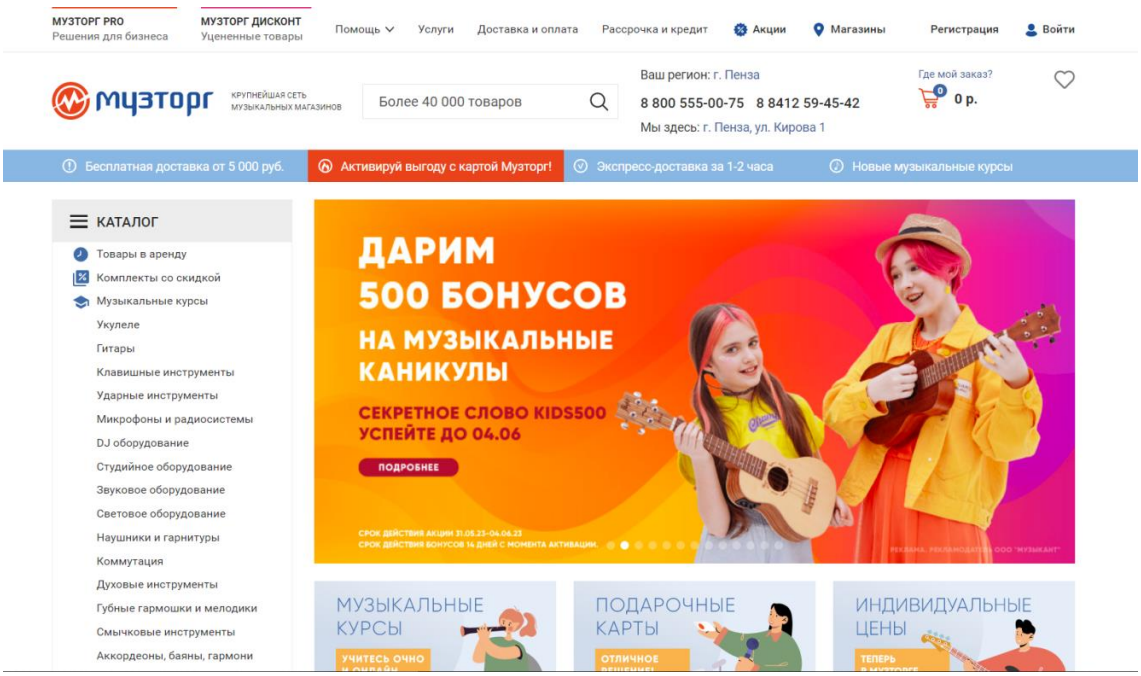


Рисунок 1 – Интерфейс главной страницы Музторг

Jazz-shop – еще один популярный интернет магазин музыкальных инструментов. В интернет-магазине Jazz-shop представлен огромный выбор продукции. В зависимости от пожеланий заказчика, продукция может выдаваться в офисах, розничных магазинах, пунктах выдачи сети Jazz-shop, которые расположены в крупных городах России [3].

Рассмотрим интерфейс сайта Jazz-shop.

Интерфейс сайта Jazz-shop имеет важное значение для привлечения и удержания клиентов, поэтому внимание к деталям и удобству использования играют ключевую роль в его разработке. Интерфейс должен давать пользователю возможность пользоваться функциями приложения.

На главной странице расположены различные баннеры, рекламирующие музыкальные инструменты. Чуть ниже представлен каталог в виде интерактивных картинок, при переходе на которые пользователь попадает в определенный раздел товаров.

Интерфейс главной страницы и каталога сайта Jazz-shop представлены на рисунках 2, 3.

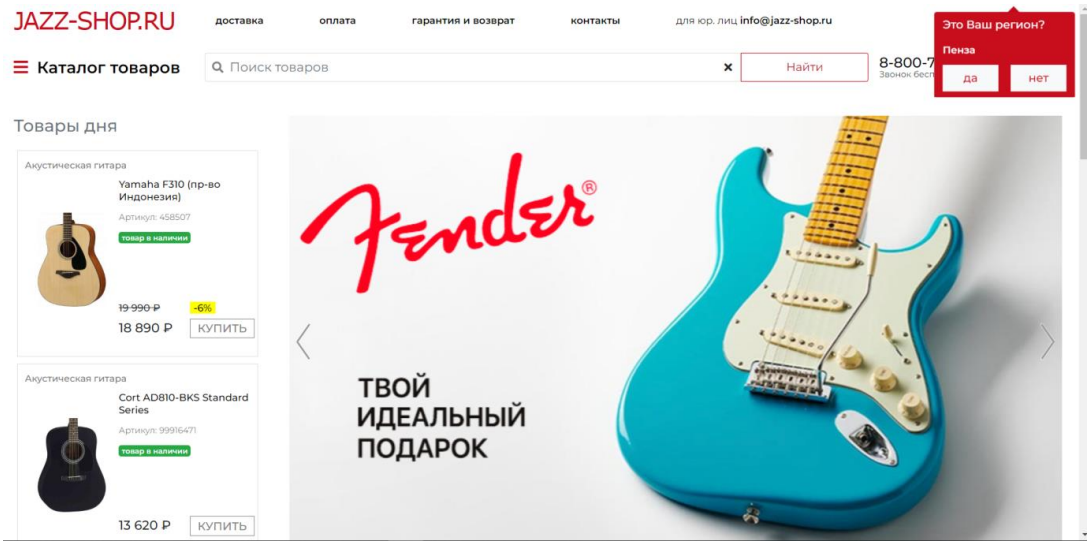


Рисунок 2 – Интерфейс главной страницы Jazz-shop

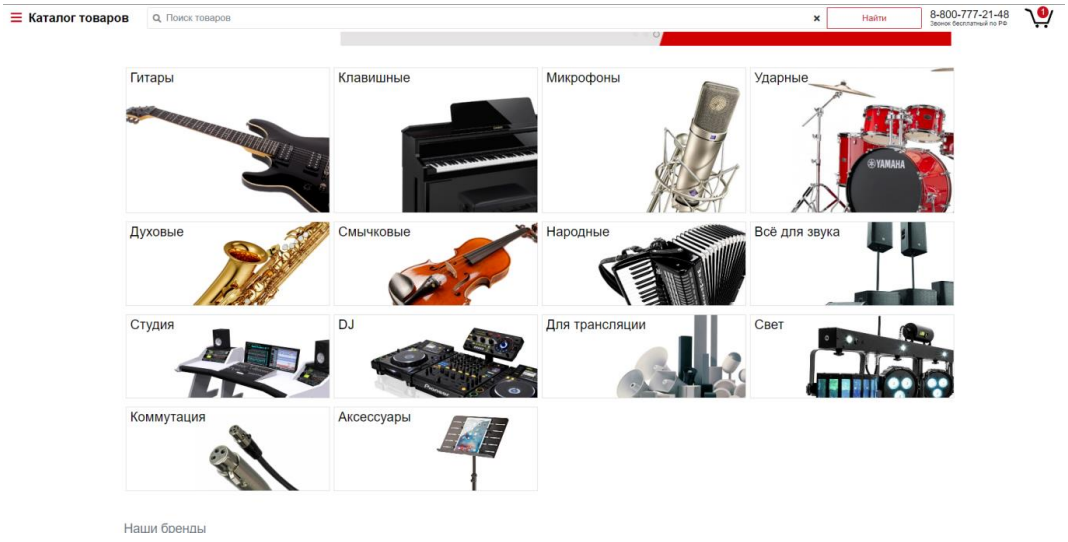


Рисунок 3 – Интерфейс каталога Jazz-shop

Мир музыки – это сеть крупнейших в России магазинов музыкальных инструментов, световой техники, звукового, студийного и торгового оборудования для музыкальных магазинов [4].

Рассмотрим интерфейс сайта Мир музыки.

На главной странице сайта представлен каталог с типами музыкальных инструментов, большой рекламный баннер и карточки с брендами музыкальных инструментов, при переходе на которые открывается отдельная страница с определенными товарами.

Ниже представлена главная страница и раздел с популярными категориями (рисунки 4, 5).

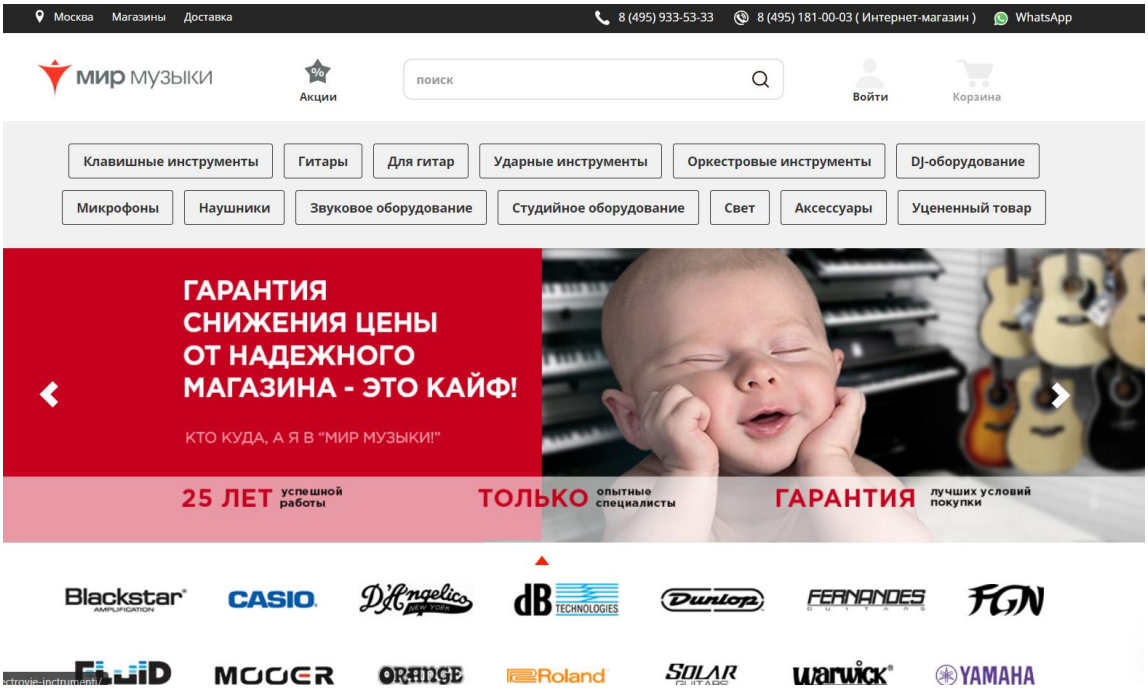


Рисунок 4 – Интерфейс главной страницы Мир музыки

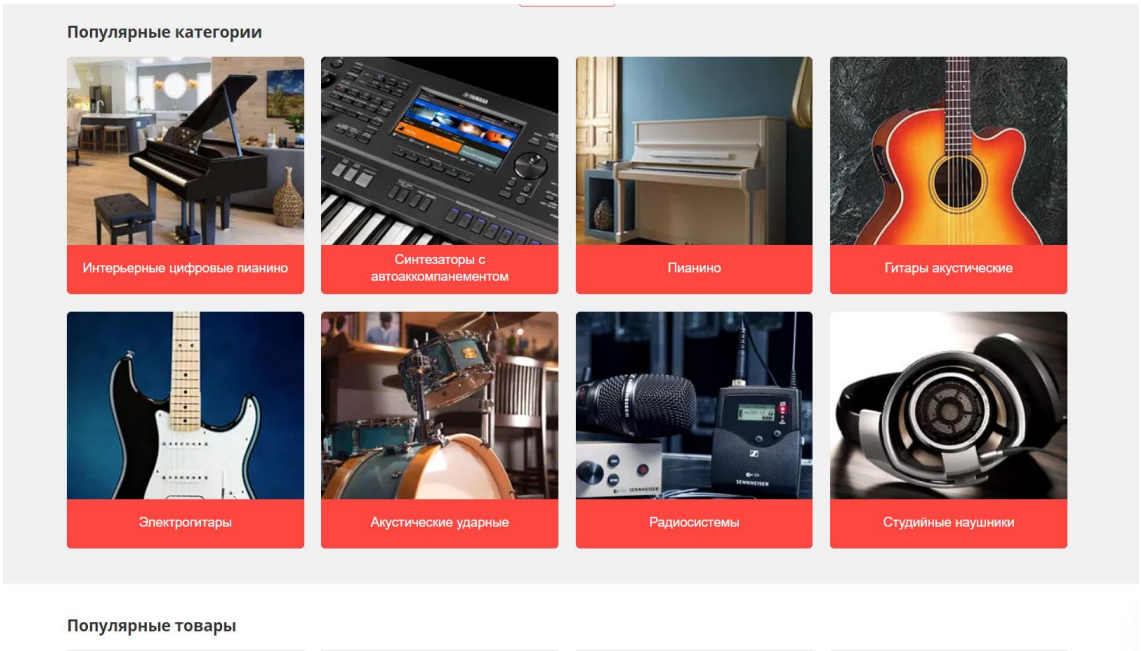


Рисунок 5 – Интерфейс раздела популярные категории

В результате обзора web-приложений музыкального магазина были рассмотрены интерфейсы сайтов.

1.3 Сравнительный анализ аналогов

После того, как были рассмотрены приведенные выше аналоги приложения, необходимо подвести итоги. Определим основные критерии для сравнения аналогов приложения.

При первичном анализе приложений аналогов было выявлено, что одними из ключевых недостатков является наличие рекламных баннеров в приложении, загруженность интерфейса и отсутствие звукового представления товаров. Так же преимуществами разрабатываемого продукта может являться доступность программного обеспечения для заказчика.

Перегруженный интерфейс сайта может стать причиной ухудшения пользовательского опыта и привести к снижению эффективности сайта. Ниже приведены некоторые критерии, которые могут указывать на перегруженность интерфейса сайта.

Слишком много элементов интерфейса. Если на странице много элементов, таких как кнопки, ссылки, изображения, видео и т. д., это может привести к перегруженности интерфейса и снижению его эффективности.

Наличие рекламных баннеров. Рекламные баннеры являются раздражителями для пользователя и негативно влияют на восприятие интерфейса.

Качество контента. Наличие качественного и интересного контента на сайте повышает эргономичность интерфейса, так как пользователи могут оставаться на сайте и возвращаться к нему снова и снова.

Проведем анализ загруженности интерфейса аналогов. Для этого подсчитаем количество элементов интерфейса на главной странице, количество рекламных баннеров и процент качественного контента на странице.

На главной странице сайта Музторг расположено меню сайта, состоящее из 23 элементов. Слева находится каталог, содержащий виды музыкальных инструментов, различные акции и предложения. Всего каталог содержит 28 элементов. Основной раздел главной страницы содержит 51 элемент. Нижняя часть сайта содержит 40 элементов. Итого на главной странице Музторга

расположено 142 элемента, среди которых только 45 являются семантически значимыми для пользователя. Так же на сайте расположено 15 рекламных баннера. Расчет доли семантически значимого контента на сайте, производится по формуле:

$$P = \frac{A_1}{A_2} * 100,$$

где P – доля семантически значимого контента (%);

A_1 – количество необходимых элементов (шт.);

A_2 – количество всех элементов (шт.).

Подсчитаем долю семантически значимого контента на главной странице сайта Музторг:

P = 45 / 142 * 100 = 31,7 %.

Jazz-shop имеет более простой интерфейс. Главное меню содержит шапку сайта, содержащую 11 интерактивных элементов, основной раздел, включающий 28 элементов и нижнюю часть сайта, содержащую 7 элементов. Итого на странице Jazz-shop 46 элементов, среди которых 24 являются семантически значимыми для пользователя. Так же на сайте расположено 3 рекламных баннера. Подсчитаем долю семантически значимого контента на главной странице сайта:

P = 24 / 46 * 100 = 52,2 %.

Мир музыки содержит меню из 11 интерактивных элементов. Основной раздел содержит 74 элементов. Нижняя часть сайта содержит 35 элементов. Итого главная страница включает в себя 120 элементов, среди которых лишь 41 элемент является семантически значимым для пользователя. На сайте расположено 5 рекламных обьявления. Подсчитаем долю семантически значимого контента на главной странице сайта:

P = 41 / 120 * 100 = 34,2 %.

В результате проведения подсчетов, можно сделать вывод о том, что интерфейсы сайтов Музторг и Мир музыки являются перегруженными. Интерфейс сайта Jazz-shop в сравнении с ними является более чистым. Все приложения содержат рекламные баннеры и не содержат аудио представления музыкальных товаров.

Приведем таблицу, содержащую основную информацию о web-приложениях музыкальных магазинов, отображающую общие черты, присутствующие либо отсутствующие в каждом приложении: количество элементов на главной странице сайта, количество рекламных баннеров, доля семантически значимого контента и наличие аудио представления товаров (таблица 1).

Таблица 1 – Сравнение аналогов

Web-приложения музыкального магазина	Количество элементов	Количество рекламных баннеров	Доля семантически значимого контента, %	Наличие аудио представления
Музторг	142	15	31.7%	Нет
Jazz-shop	46	3	52.2%.	Нет
Мир музыки	120	5	34.2%.	Нет

В результате анализа аналогов web-приложения музыкального магазина была получена более точная информация о возможных требованиях и потребностях пользователей, а также выявлены основные недостатки приложений, такие как загруженность интерфейса, наличие рекламных баннеров, отсутствие звукового представления товаров. Было принято решение о разработке собственного приложения, лишенного недостатков, выявленных у рассмотренных выше аналогов.

1.4 Анализ функциональных требований

Для того чтобы определить протекающие процессы в web-приложении музыкального магазина необходимо провести анализ наиболее вероятных вариантов использования.

Приложение должно включать в себя страницу авторизации и регистрации, чтобы иметь возможность создавать новых пользователей и сохранять различные данные о пользователе.

Так же web-приложение должно иметь следующие типы пользователей:

– администратор;

– пользователь.

«Администратор» должен иметь право добавлять новые типы, бренды товаров, а так же сами товары через «Админ панель». При добавлении товара Администратор будет указывать тип и бренд музыкального инструмента из существующих, загружать картинку с изображением товара, а так же аудио-файл, показывающий звучание данного музыкального инструмента

У обычного пользователя возможность добавления товара должна отсутствовать.

Приложение должно включать в себя главную страницу. На главной странице будет отображаться шапка сайта, список карточек музыкальных инструментов, фильтры выбора типа и бренда музыкального инструмента.

Карточка музыкального инструмента должна включать основную информацию о товаре:

– название музыкального инструмента;

– фото музыкального инструмента;

– стоимость музыкального инструмента.

Так же на карточках музыкальных инструментов будет расположена кнопка, при нажатии которой на сайте будет воспроизводиться аудио-файл, имитирующий звучание данного музыкального инструмента.

Пользователь должен иметь возможность выбрать интересующий тип или бренд музыкального инструмента, посмотреть подробную информацию конкретного музыкального инструмента, послушать звучание музыкального инструмента, добавить музыкальный инструмент в корзину, посмотреть содержимое корзины.

Проанализировав особенности, можно перейти к определению вариантов использования автоматизированной информационной системы web-приложения музыкального магазина, обслуживающей предполагаемых пользователей.

Информационные процессы, которые возникают во время работы с сайтом музыкального магазина у обычного пользователя достаточно простые и сводятся к небольшому набору основных действий:

– регистрация нового пользователя;
– авторизация пользователя;
– просмотр подробной информации о музыкальном инструменте;
– прослушивание звучания музыкального инструмента;
– добавление товара в корзину;
– просмотр содержимого корзины;
– выход из учетной записи.

У пользователя с правами администратора появляются следующие варианты использования:

– добавление вида музыкального инструмента;
– добавление типа музыкального инструмента;
– добавление бренда музыкального инструмента;
– добавление музыкального инструмента;

Рассмотрим варианты использования подробнее.

Регистрация нового пользователя. При переходе на страницу авторизации, если у пользователя нет аккаунта, он может выбрать соответствующее действие и перейти на страницу регистрации. Здесь он должен указать адрес своей электронной почты и пароль, после чего нажать кнопку регистрации. После этого пользователь попадает на главную страницу сайта под своей учетной записью, а в базе данных сохраняется email нового пользователя и зашифрованный пароль.

Аутентификация и авторизация пользователя. На странице авторизации пользователь должен ввести данные своего аккаунта, а именно электронную почту и пароль. Система получает данные, проверяет их, сравнивая с данными, хранящимися в базе данных. Если полученные данные совпадают с данными из БД, то пользователь успешно переходит на главную страницу сайта под своей учетной записью. Если же данные не совпали, то пользователь получает соответствующую ошибку.

Просмотр подробной информации о музыкальном инструменте. Доступ к подробной информации об инструменте должен быть у всех пользователей web-приложения, в том числе не авторизованным. У музыкального инструмента должно быть указано его название, цена, фото инструмента, общие характеристики, а так же аудио информация о звучании инструмента.

Прослушивание звучания музыкального инструмента. Пользователь должен иметь возможность прослушать звучание музыкального инструмента. Это может послужить хорошим плюсом, так как пользователь получает не только визуальную информацию, но и звуковую, благодаря которой он может оценить специфику звучания того или иного музыкального инструмента.

Добавление товара в корзину и просмотр содержимого корзины. Доступ к корзине должен быть только у авторизованных пользователей. Товары, добавленные в корзину должны сохраняться в базе данных для каждого пользователя отдельно.

Добавление музыкального инструмента Добавление музыкального инструмента должно быть доступно только администратору. При добавлении должны указываться тип и бренд музыкального инструмента, его характеристики, а так же администратор должен загрузить фотографию музыкального инструмента и аудио-файл, содержащий звук данного инструмента.

При добавлении нового музыкального инструмента администратор должен выбрать тип, бренд, название и цену музыкального инструмента. Так же администратор выбирает изображения и аудио-файлы, а так же определяет характеристики для товара.

Составим UML-диаграмму вариантов использования для обычного пользователя и для администратора [6].

Диаграмма вариантов использования представлена на рисунке 6.

Рисунок 6 – Диаграмма вариантов использования

Далее представлены примеры спецификация для некоторых прецедентов.

На рисунке 7 представлена спецификация прецедента «Просмотр информации о музыкальном инструменте»

Прецедент: Просмотр информации о музыкальном инструменте
ID:1
Краткое описание: Пользователь просматривает информацию о музыкальном инструменте
Главные актеры: Пользователь
Второстепенные актеры: СУБД
Предусловия: 1. Пользователь открыл главную страницу магазина.
Основной поток: 1. Пользователь переходит на страницу музыкального инструмента 2. Если сервер работает 2.1. Система выдает информацию о музыкальном инструменте 2.2. Пользователь просматривает информацию о музыкальном инструменте 3. Иначе 3.1 Система выдает сообщение об ошибке
Постусловия: Пользователь просмотрел информацию о музыкальном инструменте
Альтернативные потоки: Нет.

Рисунок 7 – Спецификация прецедента «Просмотр информации о музыкальном инструменте»

На рисунке 8 представлена спецификация прецедента «Добавление товара в корзину», на котором описана главная информация о прецеденте и последовательность действий при добавлении товара в корзину.

Прецедент: Добавление товара в корзину
ID:2
Краткое описание: Пользователь добавляет музыкальный инструмент в корзину
Главные актеры: Пользователь
Второстепенные актеры: СУБД
Предусловия: 1. Пользователь авторизован в системе
Основной поток: 1. Пользователь переходит на страницу музыкального инструмента 2. Пользователь нажимает кнопку добавления товара в корзину 3. Если сервер работает 3.1. Товар добавляется в корзину пользователя 4. Иначе 4.1 Система выдает сообщение об ошибке
Постусловия: Товар добавлен в корзину пользователя
Альтернативные потоки: Нет.

Рисунок 8 – Спецификация прецедента «Добавление товара в корзину»

На рисунке 9 представлена спецификация прецедента «Добавление нового музыкального инструмента»

Прецедент: Добавление нового музыкального инструмента
ID:3
Краткое описание: Администратор добавляет новый музыкальный инструмент
Главные актеры: Администратор
Второстепенные актеры: СУБД
Предусловия: 1. Пользователь авторизован в системе как администратор
Основной поток: 1. Администратор переходит на страницу админ панели 2. Администратор выбирает тип и вид музыкального инструмента 3. Администратор вводит название и цену музыкального инструмента 4. Администратор выбирает изображение и аудио-файл 5. Администратор добавляет характеристики музыкального инструмента 6. Если сервер работает 6.1. Добавляется новый музыкальный инструмент 7. Иначе 7.1 Система выдает сообщение об ошибке
Постусловия: Новый музыкальный инструмент добавлен
Альтернативные потоки: Нет.

Рисунок 9 – Спецификация прецедента «Добавление нового музыкального инструмента»

На основе вариантов использования, была сформирована последовательность действий для некоторых прецедентов.

Диаграмма последовательности действия для прецедента добавления товара в корзину представлена на рисунке 10.

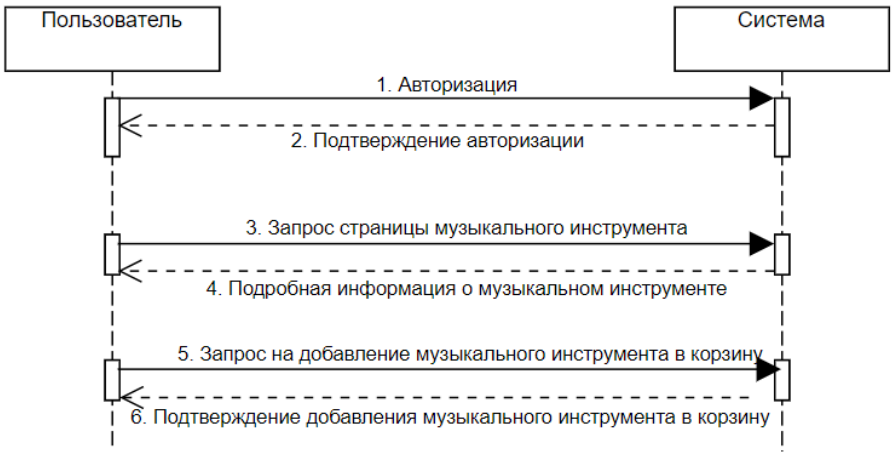


Рисунок 10 – Диаграмма последовательности действий для прецедента «Добавить товар в корзину»

Диаграмма последовательности действия для прецедента добавления нового музыкального инструмента представлена на рисунке 11.

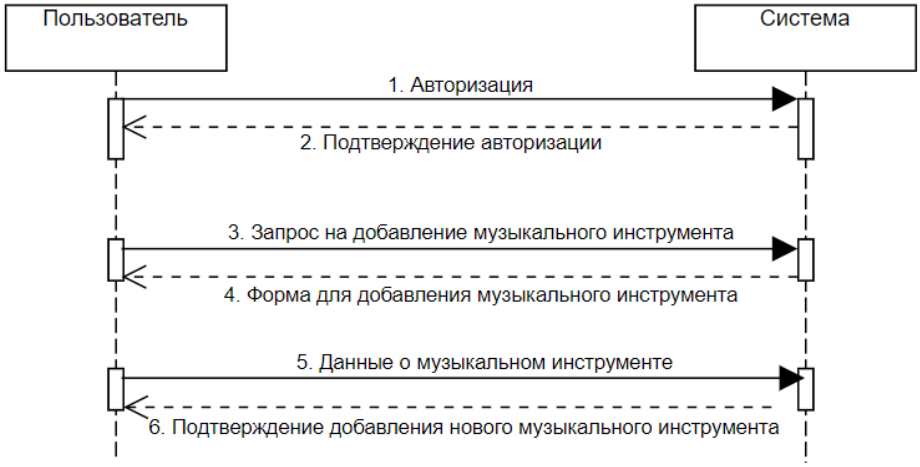


Рисунок 11 – Диаграмма последовательности действий для прецедента «Добавить новый музыкальный инструмент»

После проведения анализа вариантов использования web-приложения музыкального магазина, можно приступить к определению его структурной модели.

1.5 Анализ требований к пользовательскому интерфейсу

Интерфейс web-приложения должен быть простым и интуитивно понятным. При разработке дизайна сайта должны использоваться в основном светлые цвета. Недопустимо использование токсичных цветов, таких как неоновый, ядовито зеленый и т. д.

На сайте не должно быть ненужной информации и баннеров. Текст должен быть легко читаем и не должен сливаться с фоновым цветом. Все блоки и кнопки должны располагаться на правильных местах, чтобы обеспечить пользователю интуитивно понятный интерфейс.

На всех страницах сверху должно располагаться меню, содержащее основные разделы сайта. На главной странице по бокам должны быть расположены блоки фильтрации компонентов, а в центре должен быть расположен блок с товарами.

Количество товаров должно ограничиваться лимитом, чтобы не создавалось излишней загруженности интерфейса, должна быть реализована пагинация блока с товарами. Интерфейс сайта должен быть адаптивным и нормально отображать содержимое на всех популярных размерах экрана.

Верстка сайта должна быть кроссбраузерной, сам сайт должен быть доступен для полнофункционального просмотра при помощи следующих браузеров:

- Microsoft Edge 80 и выше;
- Opera 65 и выше;
- Firefox 70 и выше;
- Google Chrome 80 и выше.

На сайте должны быть расположены все возможные разделы, кнопки и описания на русском языке. Текстовое содержание приложения должно быть понятно для пользователя, названия кнопок должны соответствовать их содержанию.

2 Проектирование web-приложения музыкального магазина

2.1 Проектирование пользовательского интерфейса

Составим общую структуру сайта, схема которой представлена на рисунке 12.

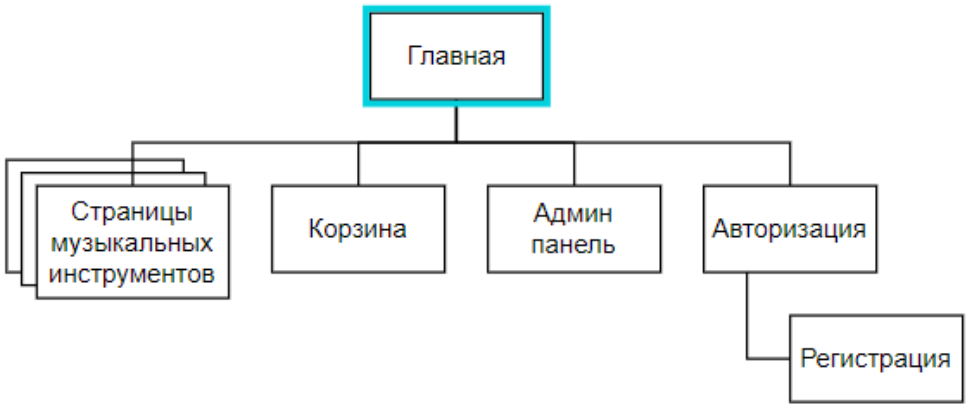


Рисунок 12 – Общая структура сайта

Смоделировав структуру сайта, можно перейти к проектированию пользовательского интерфейса.

Проектирование пользовательского интерфейса является одним из ключевых этапов создания веб-приложения. Цель проектирования – создание интерфейса, который был бы удобен и понятен для пользователей, а также обеспечил бы эффективное взаимодействие между пользователем и системой.

При разработке интерфейса следует учитывать несколько основных принципов. В первую очередь, интерфейс должен быть простым и интуитивно понятным для пользователей. Кроме того, он должен обладать достаточной информативностью и функциональностью, чтобы пользователи могли легко находить нужную им информацию и выполнять требуемые действия.

Интерфейс должен быть построен таким образом, чтобы пользователи могли быстро ориентироваться и находить нужную информацию. Важно учитывать, что большинство пользователей не желают тратить много времени на изучение интерфейса, поэтому он должен быть максимально простым и понятным.

Важным аспектом при создании интерфейса является выбор цветовой гаммы, шрифтов и графических элементов. Они должны гармонично сочетаться между собой и соответствовать общему стилю сайта.

Также необходимо обеспечить адаптивность интерфейса под различные устройства и экраны, чтобы пользователи могли удобно пользоваться сайтом с любого устройства.

Главная страница является основной страницей на сайте и должна содержать основные элементы сайта.

На каждой странице должно быть верхнее меню, содержащее логотип и кнопки перехода на основные разделы сайта.

Главная страница должна быть разделена на следующие основные блоки:

– типы музыкальных инструментов;

– бренды музыкальных инструментов;

– основной раздел, содержащий карточки музыкальных инструментов.

Макет главной страницы представлен на рисунке 13.

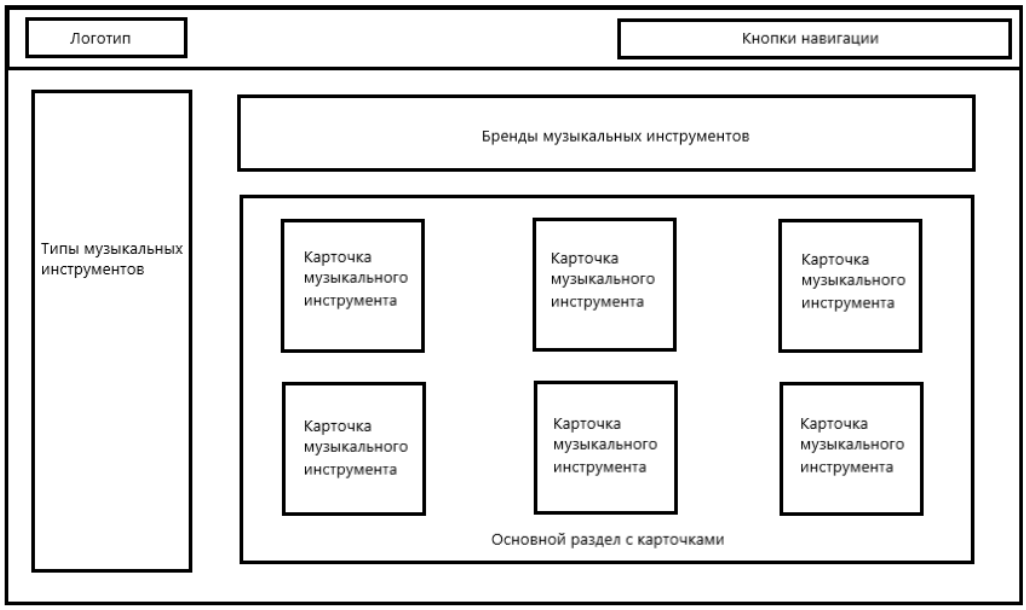


Рисунок 13 – Макет главной страницы

Страницы регистрации и авторизации должна содержать следующие элементы:

– поле ввода Email;
– поле ввода пароля;
– кнопка перехода на страницу регистрации/авторизации;
– кнопка входа.

Макет страницы авторизации представлен на рисунке 14.

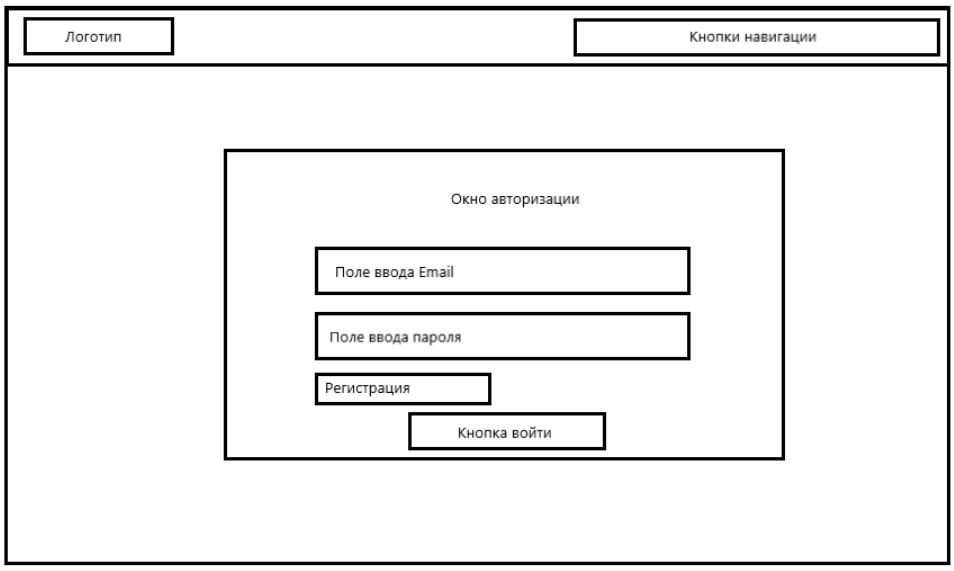


Рисунок 14 – Макет страницы авторизации

Страница отдельного музыкального инструмента должна отображать основную информацию о музыкальном инструменте, так же интерфейс должен предусматривать возможность добавления музыкального инструмента в корзину.

Страница музыкального инструмента должна быть разделена на следующие основные блоки:

– изображение музыкального инструмента;
– описание музыкального инструмента;
– раздел, содержащий цену товара и кнопку добавления в корзину;
– аудио проигрыватель;
– блок характеристик товара.

Макет страницы музыкального инструмента представлен на рисунке 15.

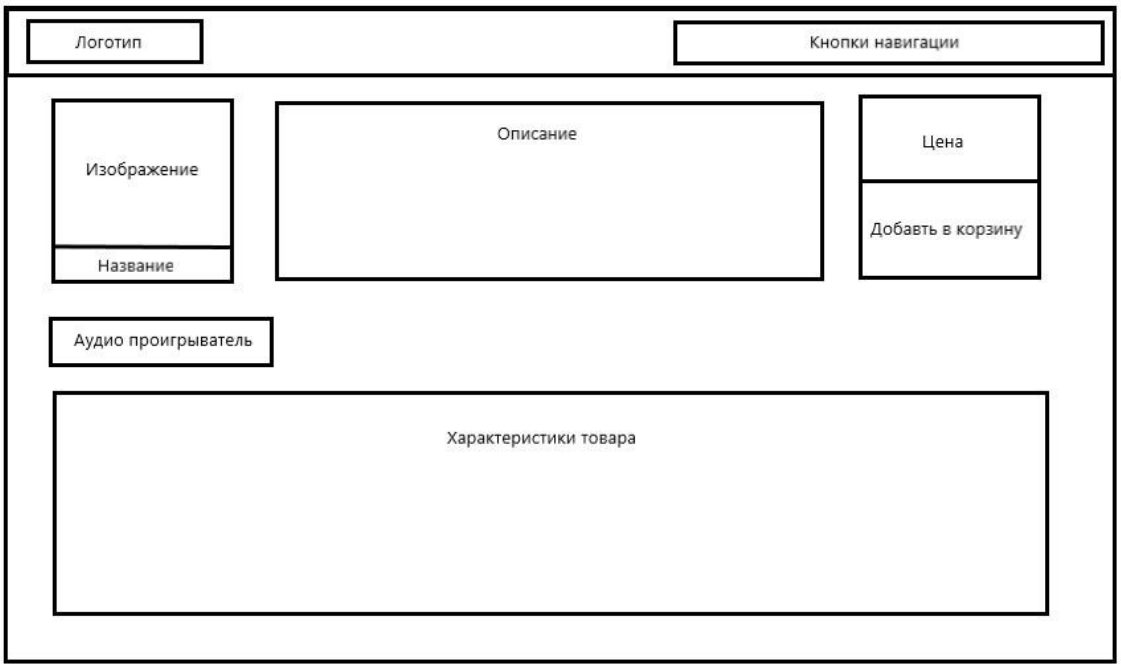


Рисунок 15 – Макет страницы музыкального инструмента

Страница корзины должна содержать следующие элементы интерфейса:

– вывод итоговой стоимости товаров;
– блок, содержащий карточки добавленных товаров.

2.2 Организация процесса обработки информации

Чтобы обеспечить эффективную работу web-приложения, необходимо правильно организовать процесс обработки информации.

Процесс обработки информации в web-приложении включает в себя сбор, хранение, обработку и вывод информации, которая передается между пользователем и сервером. Это включает в себя обработку пользовательских запросов, работу с базами данных, передачу данных между сервером и клиентом, и многое другое.

Организация процесса обработки информации в web-приложении имеет решающее значение для его функциональности, производительности и безопасности. Определим все компоненты нашего приложения и их

доступность для каждого вида пользователя. Для этого разработаем диаграмму компонентов.

Диаграмма компонентов представлена на рисунке 16.

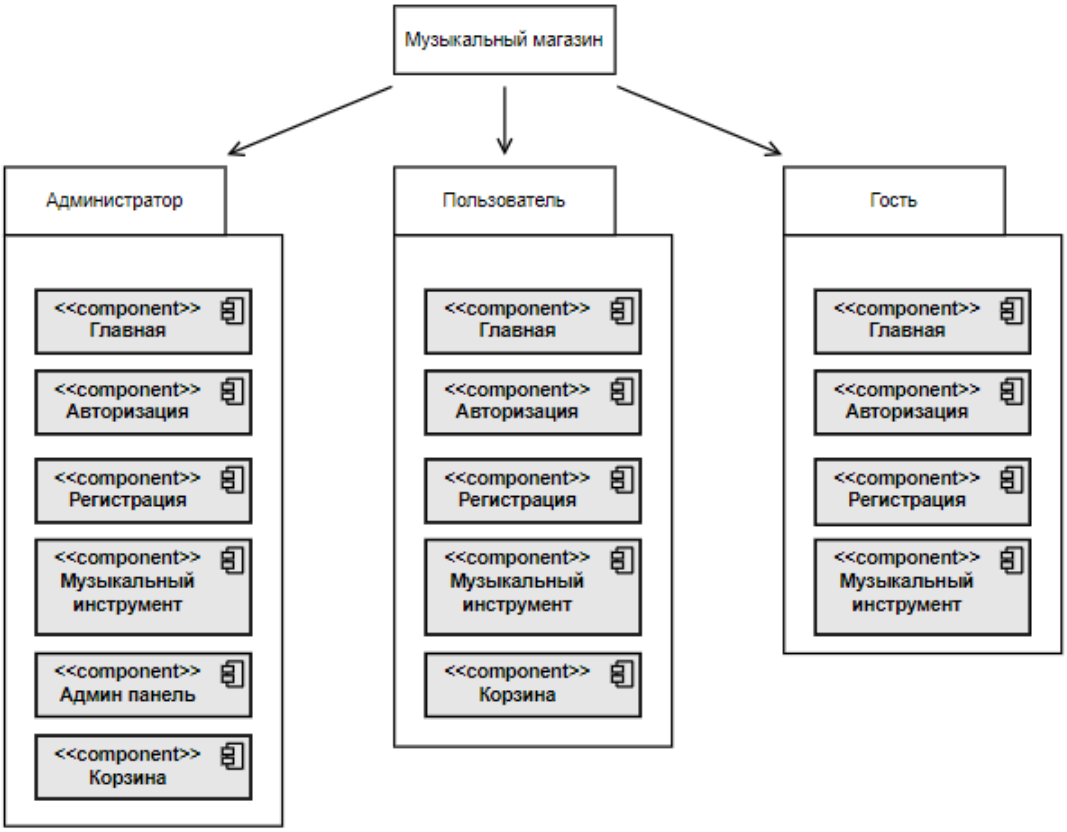


Рисунок 16 – Диаграмма компонентов

Чтобы реализовать систему обработки запросов, используется так называемый контроллер. Он получает адрес, запрошенный пользователем, и обрабатывает запрос.

После обработки запроса сервисы возвращают необходимые данные обратно в контроллер. Контроллер формирует ответ для клиента в формате JSON, который содержит запрошенные данные.

Таким образом, контроллер является ключевым элементом в обработке запросов веб-приложения. Он обеспечивает связь между пользователем и системой, обрабатывает запросы и возвращает необходимую информацию обратно клиенту. Важно убедиться, что контроллеры веб-приложения эффективно работают во всех сценариях использования, чтобы обеспечить

быстродействие и удобство работы приложения для пользователей.

Для реализации организации взаимодействия программного кода с базой данных используем инструмент Sequelize. С помощью него взаимодействие с базой данных происходит не благодаря использованию специального языка запросов SQL, а при помощи встроенных функций, что позволяет обеспечить удобное динамическое взаимодействие базы данных с сервером.

Для обеспечения безопасности аутентификации пользователей в web-приложениях часто используется JWT или JSON Web Token. Этот метод заключается в том, что при успешной авторизации пользователь получает уникальный токен, который затем используется для подтверждения его личности при последующих запросах к системе. JWT является удобным и надежным методом для авторизации пользователей в web-приложениях. Он обеспечивает безопасность передаваемых данных и позволяет избежать многих проблем, связанных с аутентификацией и авторизацией пользователей в сети.

2.3 Структуры хранения данных

Структура хранения данных является важным аспектом при проектировании и разработке информационных систем. Она определяет, как данные будут храниться, организовываться. Кроме того, структура хранения данных определяет структуру доступа для использования. Правильно спроектированная структура хранения данных обеспечивает эффективное использование ресурсов и позволяет получать быстрый доступ к нужной информации. В данном разделе мы рассмотрим основные аспекты структуры хранения данных, включая модели данных, типы баз данных, а также методы организации и обработки данных.

В ходе проектирования баз данных была определена структура и содержание классов БД. Так же были определены типы данных и их ограничения.

Диаграмма классов базы данных представлена на рисунке 17

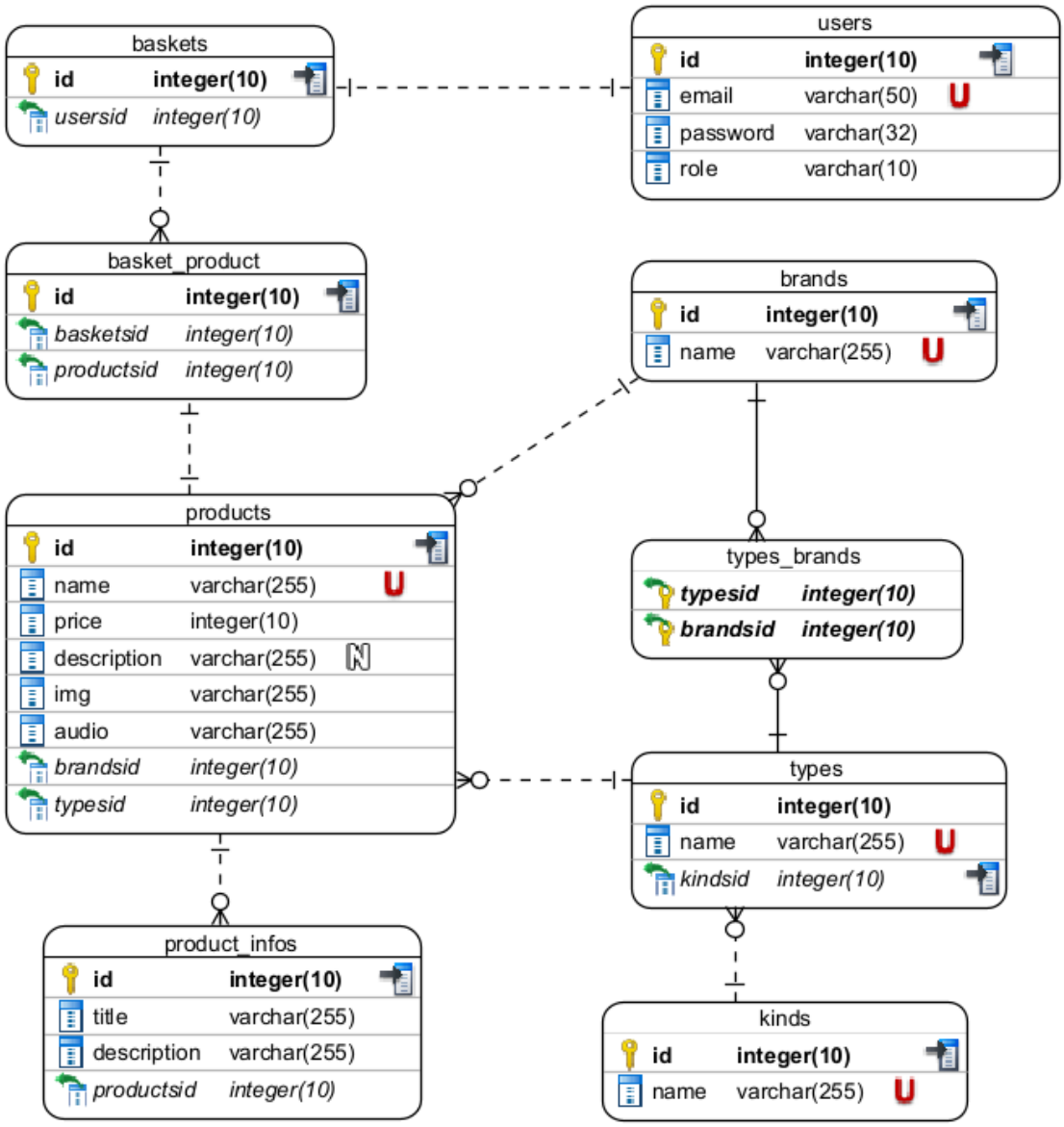


Рисунок 17 – Диаграмма классов базы данных

Для хранения данных используется мощная и надежная система управления реляционными базами данных, которая имеет множество преимуществ перед другими СУБД PostgreSQL. Непосредственное взаимодействие с базами данных осуществляется при помощи программы pgAdmin 4. База данных музыкального магазина состоит из 9 связанных классов.

Назначение таблиц, представленных на схеме, представлено в таблице 2.

Таблица 2 – описание таблиц БД

Название сущности	Идентификатор сущности	Назначение сущности
Пользователи	users	Содержит сведения о пользователях
Товары	products	Содержит сведения о товарах
Информация о товаре	product_infos	Содержит характеристики товара
Типы	types	Содержит названия типов товара
Виды	kinds	Содержит названия видов товара
Бренды	brands	Содержит названия брендов товара
Корзина	baskets	Содержит сведения о корзине пользователя
Корзина_товар	basket_products	Связывает корзину и товары
Тип_бренд	type_brands	Связывает типы и бренды

Определим связи таблиц исходя из логики. Таблицы «users» и «baskets» связаны отношением «один-к-одному». Таблицы «baskets» и «basket_products» связаны отношением «один-ко-многим». Таблицы «basket_products» и «products» связаны отношением «один-к-одному». Таблицы «products» и «product_infos» связаны отношением «один-ко-многим». Таблицы «brands» и «products» и таблицы «types» и «products» связаны отношением «один-ко-многим». Таблицы «users» и «baskets» связаны отношением «один-к-одному». Таблицы «types» и «brands» связаны отношением «многие-ко-многим», поэтому образуется вспомогательная таблица «type_brands». Таблицы «kinds» и «types» связаны отношением «один-ко многим».

Связь программного кода и реляционной базы данных будет осуществляться с помощью ORM-библиотеки Sequelize. С её помощью осуществляются различные изменения данных в БД с использованием встроенных в библиотеку функций. Описание таблиц базы данных представлено в приложении А.

2.4 Выбор программно-аппаратных средств

Среди всех возможных вариантов программного обеспечения для музыкального магазина самым подходящим вариантом является web-приложение. Главным преимуществом web-приложения является доступность пользования с любого устройства, имеющего доступ в Интернет. Так же пользователю не нужно устанавливать стороннее ПО, а все основные действия происходят на стороне сервера.

Веб-приложение представляет собой приложение, в котором все взаимодействие происходит в браузере [7]. Веб-приложение использует клиент-серверный подход. Хранение различных данных внутри приложения осуществляется на стороне сервера. При открытии клиентом приложения в браузере или при каком-либо взаимодействии с интерфейсом web-приложения, должен отправляться запрос на сервер. Сервер в свою очередь взаимодействует с базой данных. После чего сервер эти данные возвращает клиенту и пользователь получает их в виде какой то информации. Работоспособность веб-приложений следует тестировать на различных браузерах с целью предотвращения ошибок.

Сайт – это информационный ресурс в Интернете, состоящий из одной или более страниц, связанных между собой и имеющий уникальный адрес. Доступ к сайту осуществляется с помощью браузера [8, 9].

Как упоминалось выше, приложение будет состоять из серверной и клиентской части. Одной из самых лучших технологий разработки серверной части является Node.js. Использовать данную технологию мы будем в связке с фреймворком Express.

Node.js – это платформа с открытым исходным кодом для работы с языком программирования JavaScript. Она позволяет писать серверный код для веб-приложений, динамических страниц, а так же программ командной строки [10].

Express – популярный веб-фреймворк, написанный на JavaScript и работающий внутри среды исполнения node.js. Express это гибкая платформа веб-приложений, которая предоставляет набор полезных функций для реализации приложения. Данный фреймворк является достаточно производительным и удобным, благодаря чему пользуется высоким спросом [11].

В качестве системы управления базами данных будем использовать PostgreSQL. Чтобы связывать программный код с базой данных, будем использовать ORM технологию для реляционных баз данных на node.js Sequelize.

PostgreSQL – это объектно-реляционная система управления базами данных с открытым исходным кодом, которая поддерживает большую часть стандарта SQL и предоставляет множество современных функций для работы с данными [12].

Sequelize – это ORM-библиотека для приложений на Node.js, которая осуществляет сопоставление таблиц в базе данных и отношений между ними с классами. При использовании данной технологии взаимодействие с базой данных происходит без использования специального языка запросов SQL. Работа с данными реализована как с обычными объектами [13].

Для реализации клиентской части web-приложения будет использована технология React. Для удобства вёрстки будет использован React Bootstrap. Для реализации работы запросов к серверу будет использована библиотека Axios. Для навигации по страницам используем React-router-dom, а в качестве стейт менеджмента будем использовать MobX.

React – JavaScript-библиотека для создания пользовательских интерфейсов с открытым исходным кодом. Данная технология используется для разработки интерфейсов в клиентской части web-приложений [14].

React Bootstrap – это CSS-фреймворк с открытым исходным кодом, предназначенный для верстки адаптивных интерфейсов веб-приложений. React Bootstrap реализует компоненты Bootstrap с использованием React и включает в себя множество шаблонов компонентов для быстрой вёрстки [15].

Axios – открытая JavaScript библиотека для поддержки HTTP-запросов. Данная библиотека изоморфная и может работать в браузере и node.js с той же базой кодов [16].

React Router DOM – это модуль узла, который предназначен для маршрутизации в веб-приложениях. Данная библиотека позволяет создавать маршруты и переключения по ним для React страниц [17].

MobX – это автономная библиотека, реализующая масштабируемое управление состоянием приложения. Библиотека MobX работает в связке с React и имеет очень большой функционал наблюдения за изменениями модели. Она позволяет определять свойства состояния и реализовывать функции, которые будут исполняться при изменении модели [18].

3 Реализация web-приложения «Музыкальный магазин»

3.1 Кодирование web-приложения

Клиент-серверный подход разработки web-приложений – это архитектурный подход, в котором веб-приложение разбито на две основные части: клиентскую и серверную. Клиентская часть отвечает за интерфейс и взаимодействие с пользователем, а серверная – за бизнес-логику и работу с данными.

Серверная часть имеет следующую структуру файлов:

– controllers – папка содержащая классы контроллеры, описывающий логику серверной части приложения;

– error – папка, содержащая класс, отвечающий за обработку универсальных ошибок;

– middleware – папка с классами, описывающие обработку ошибок;

– models – папка содержащий файл, описывающий модель базы данных;

– routes – папка с файлами, описывающими маршрутизацию;

– static – папка с изображениями и аудиофайлами музыкальных инструментов;

– .env – файл конфигурации, описывающий переменные окружения;

– db.js – файл, конфигурирующий подключение к базе данных;

– index.js – файл, отвечающий за запуск серверной части приложения.

Рассмотрим назначение файлов в приведенных выше папках.

Controllers – классы контроллеров:

– basketController.js – класс контроллера для запросов, связанных с корзиной;

– brandController.js – класс контроллера для запросов, связанных с брендами музыкальных инструментов;

– deviceController.js – класс контроллера для запросов, связанных с

музыкальными инструментами;

– kindController.js – класс контроллера для запросов, связанных с видами музыкальных инструментов;

– typeController.js – класс контроллера для запросов, связанных с типами музыкальных инструментов;

– userController.js – класс контроллера для запросов, связанных с пользователями приложения.

error – класс обработки универсальных ошибок.

apiError.js – класс, описывающий стандартные ошибки.

Middleware – классы, описывабщие конкретные ошибки:

– authMiddleware.js – класс, описывающий ошибки, связанные с авторизацией;

– checkRoleMiddleware.js – класс, описывающий ошибки, связанные с доступом пользователя;

– errorHandlingMiddleware.js – класс, описывающий непредвиденные ошибки.

Models – класс, описывабщий модель базы данных.

models.js – класс, описывающий модель базы данных и её ограничения.

Routes – содержит файлы, описывающие маршрутизацию сайта:

– index.js – главный класс, описывающий маршрутизацию основных элементов сервера;

– basketRouter.js – класс, описывающий маршрутизацию для корзины;

– brandRouter.js – класс, описывающий маршрутизацию брендов;

– deviceRouter.js – класс, описывающий маршрутизацию товаров;

– kindRouter.js – класс, описывающий маршрутизацию видов

музыкальных инструментов;

– typeRouter.js – класс, описывающий маршрутизацию типов музыкальных инструментов;

– userRouter.js – описывающий маршрутизацию пользователей.

Клиентское приложение является SPA – Single Page Application (одностраничное приложение), реализованное с помощью React. Особенность React – это использование компонентной архитектуры. Компоненты в React – это небольшие, независимые и переиспользуемые блоки кода, которые могут иметь свои собственные свойства и состояния. Это делает код более организованным, удобочитаемым и легко поддерживаемым. Рассмотрим компоненты разрабатываемого приложения:

– appRouter.js – описывает логику навигации по страницам;

– brandBar.js – компонент с брендами музыкальных инструментов;

– deviceItem.js – карточка музыкального инструмента;

– deviceList.js – список карточек музыкальных инструментов;

– navBar.js – верхняя панель сайта;

– pages.js – компонент, описывающий пагинацию музыкальных инструментов;

– typeBar.js – компонент с типами музыкальных инструментов;

– createBrand.js – компонент добавления брендов музыкальных инструментов;

– createDevice.js – компонент добавления музыкального инструмента;

– createKind.js – компонент добавления видов музыкальных инструментов;

– createType.js – компонент добавления типов музыкальных инструментов;

– admin.js – страница админ панели;

– auth.js – страница авторизации;

– basket.js – страница корзины;

– devicePage.js – страница музыкального инструмента;

– shop.js – главная страница магазина.

Помимо компонентов клиентская часть приложения содержит следующие файлы:

– app.js – самый внешний файл, описывающий структуру страниц;

– routes.js – файл, описывающий маршруты навигации по страницам;

– consts.js – файл, содержащий названия всех путей;

– deviceStore.js, userStore.js – файлы, описывающие глобальное хранилище музыкальных инструментов и пользователей;

– deviceAPI.js, index.js, userAPI.js – файлы, соединяющие клиентскую часть приложения с серверной.

3.2 Развертывание web-приложения

Для работы веб-приложения музыкального магазина необходимы компоненты, такие как веб-сервер, база данных и клиентское приложение.

В качестве веб-сервера для этого приложения используется Express, который является быстрым и гибким веб-фреймворком для Node.js. Express облегчает создание маршрутов и обработку запросов от клиентов, что делает его отличным выбором для создания веб-приложений.

Для хранения данных в приложении используется реляционная база данных, которая поддерживает язык запросов SQL. В данном случае, рекомендуется использовать систему управления базами данных pgAdmin.

Для клиентской части приложения используется любой современный веб-браузер для любых платформ.

Для установки и использования приложения необходимо поместить все файлы веб-приложения в папку домена веб-сервера и настроить подключение к базе данных. Это позволит приложению получать и хранить необходимую информацию, а также обрабатывать запросы от клиентов и отправлять им соответствующие ответы.

Диаграмма развертывания представлена на рисунке 18.

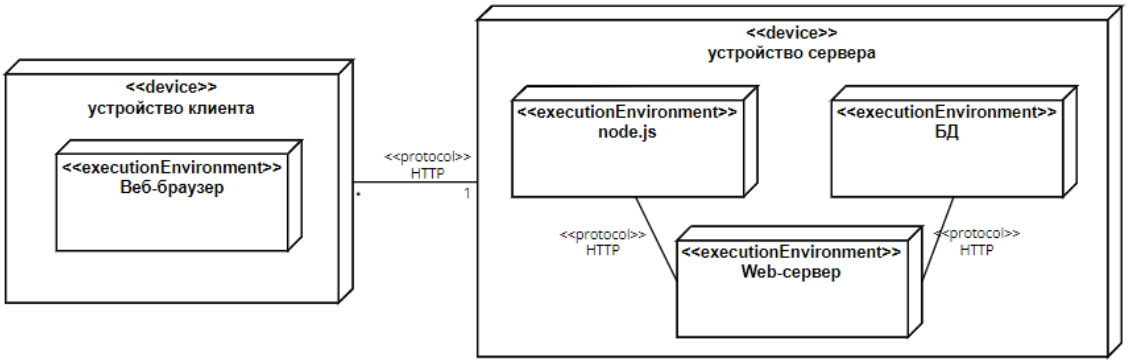


Рисунок 18 – Диаграмма развертывания

Диаграмма развертывания показывает структуру развертывания приложения.

4 Тестирование программного обеспечения

4.1 Обоснование выбора тестирования

Тестирование приложения является важным этапом в разработке любого программного продукта. Оно позволяет выявить и исправить ошибки, улучшить качество и надежность приложения, а также повысить удовлетворенность пользователей.

Выбор методов тестирования зависит от целей тестирования и характеристик приложения. Тестирование должно проводиться на всех этапах жизненного цикла приложения, начиная с тестирования отдельных модулей и заканчивая тестированием в реальных условиях использования.

В ходе анализа приложения было выяснено, что для наиболее полного и эффективного тестирования рекомендуется использовать функциональное, структурное и модульное тестирование.

Функциональное тестирование поможет проверить, соответствует ли приложение требованиям и спецификациям, определенным для него. Также оно позволит проверить, правильно ли обрабатываются входные данные, как работает интерфейс приложения, и другие функциональные аспекты.

Структурное тестирование включает в себя проверку внутренней структуры приложения, которая поможет выявить потенциальные уязвимости и ошибки в логике работы. Это в свою очередь позволит убедиться, что приложение работает корректно и эффективно.

Модульное тестирование позволит проверить отдельные модули приложения на соответствие требованиям и правильность работы. Это позволит быстро выявлять и исправлять ошибки на ранних этапах разработки, что уменьшит затраты на разработку и повысит качество приложения.

Таким образом, использование функционального, структурного и модульного тестирования позволит обеспечить более полное и эффективное тестирование приложения, выявить и исправить ошибки и повысить качество приложения.

4.2 Тестирование

После анализа требований и вариантов использования были выделены самые часто возникающие сценарии использования web-приложения музыкального магазина, которые необходимо покрыть тестами для обеспечения максимальной надежности работы и минимизации количества ошибок на этапе кодирования:

- загрузка страницы;
- переход на главную при обращении к несуществующему ресурсу;
- регистрация;
- авторизация;
- фильтрация музыкального магазина по бренду;
- фильтрация музыкального магазина по типу;
- прослушивание звучания музыкального инструмента;
- добавление в корзину;
- добавление нового вида;
- добавление нового типа;
- добавление нового музыкального инструмента.

На основе приведенных выше сценариев был составлен план тестирования и проведено функциональное тестирование. Результат тестирования приведен в таблице 3.

Таблица 3 – План тестирования

№	Названия тестового случая	Тест	Результат
1	Загрузка страницы	Будет отображена запрашиваемая страница	Страница успешно загружена
2	Переход на главную при обращении к несуществующему ресурсу	Пользователь переходит на несуществующий ресурс	Пользователь переходит на главную страницу

Продолжение таблицы 3

№	Названия тестового случая	Тест	Результат
3	Регистрация	При вводе корректных данных пользователь успешно регистрируется и авторизуется	Новый пользователь создан и авторизован в системе
4	Авторизация	При вводе существующих в базе и корректных данных пользователь успешно авторизуется	Пользователь успешно авторизуется в системе
5	Фильтрация музыкального инструмента по бренду	Пользователь выбирает определенный бренд	Отображен список музыкальных инструментов определенного бренда
6	Фильтрация музыкального инструмента по типу	Пользователь выбирает определенный тип	Отображен список музыкальных инструментов определенного типа
7	Прослушивание звучания музыкального инструмента	Пользователь наводит курсор на карточку музыкального инструмента	На сайте воспроизводится аудио-файл, отображающий звучание данного инструмента
8	Добавление в корзину	Пользователь добавляет товар в корзину	Товар успешно добавлен и отображается в корзине у пользователя
9	Добавление нового вида	Администратор добавляет новый вид музыкального инструмента	Новый вид музыкального инструмента успешно добавлен
10	Добавление нового типа	Администратор добавляет новый тип музыкального инструмента	Новый тип музыкального инструмента успешно добавлен
11	Добавление нового музыкального инструмента	Администратор добавляет новый музыкальный инструмент	Новый музыкальный инструмента успешно добавлен

Для авторизации было проведено тестирование методом выделения классов эквивалентности.

Результаты тестирования приведены в таблице 4.

Таблица 4 – Результаты тестирования

№	Входное условие	Пример	Правильность	Тест пройден
1	Корректный адрес почты	admin@mail.ru	+	+
2	Некорректный адрес почты	examplemailcom	-	+
3	Пустая строка в поле для ввода почты		-	+
4	Корректный пароль	Test1234	+	+
5	Пароль короче 8 символов	ww	-	+
6	Пароль длиннее 32 символов	nvhfYurhFJu57svcbxsA574636373728TrsdsddsdssaDFDAadsadsadssadsadaassasasadsdsasdasdasdasdasdasdasdas	-	+
7	Пустая строка в поле для ввода пароля		-	+

При тестировании веб-приложения, состоящего из клиентской и серверной частей, необходимо проводить тестирование каждой части, учитывая особенности каждой технологии. Серверная часть приложения разработана на языке JavaScript с использованием фреймворка для веб-приложений Express и библиотеки для работы с базами данных Sequelize. Клиентская часть приложения разработана с использованием React.

Для тестирования серверной части приложения используются средства тестирования, встроенные в интегрированную среду разработки WebStorm, а также библиотека Mock для проведения тестирования отдельных компонентов без учета влияния внешних сущностей. Методы API (application programming interface) тестируются с использованием специального ПО для работы с API – Postman, которое позволяет сохранять коллекции запросов к серверу и проводить автоматизированное тестирование.

Клиентскую часть приложения, написанную на JavaScript с использованием React, эффективнее и быстрее тестировать встроенными средствами React.

На основе функциональных требований были выделены следующие основные модули для тестирования:

– подсистема авторизации;
– модуль работы с пользователями;
– модуль работы с музыкальными инструментами.

На основе вариантов использования для модуля работы с музыкальными инструментами был составлен список тестовых случаев:

– создание вида музыкального инструмента;
– создание типа музыкального инструмента;
– создание бренда музыкального инструмента;
– создание музыкального инструмента;
– получение вида музыкального инструмента;
– получение типа музыкального инструмента;
– получение бренда музыкального инструмента;
– получение всех музыкальных инструментов;
– получение одного музыкальных инструментов;
– добавление музыкального инструмента в корзину;
– получение корзины.

Используя метод покрытия условий, был составлен список тестов для вариантов использования этого модуля. Список тестов представлен в таблице 5.

Таблица 5 – Тесты модуля работы с музыкальными инструментами

№	Метод класса	Тест	Результат	Тест пройден
1	createType	Создание типа инструмента	Тип успешно создан	+
2	fetchTypes	Получение типов инмтрументов	Возвращены типы инструментов	+
3	createKind	Создание вида инмтрумента	Вид успешно создан	+
4	fetchKinds	Получение видов инмтрументов	Возвращены виды инструментов	+
5	createBrand	Создание бренда инмтрумента	Бренд успешно создан	+
6	fetchBrands	Получение брендов инмтрументов	Возвращены бренды инструментов	+
7	createDevice	Создание музыкального инмтрумента	Музыкальный инструмент успешно создан	+
8	fetchDevices	Получение музыкальных инструментов	Возвращены музыкальные инструменты	+
9	fetchOneDevices	Получение одного музыкального инструмента	Возвращен музыкальный инструмент	+
10	addToBasket	Добавление инструмента в корзину	Инструмент успешно добавлен в корзину	+
11	getBasket	Получение содержимого корзины	Возвращает содержимое корзины	+

На основе составленных тестов были созданы юнит-тесты.

После проведения тестов было выявлено, что все методы класса покрыты тестами, а все функциональные требования выполнены, что подтверждает отсутствие ошибок. Аналогично, юнит-тесты других классов также были проведены и прошли успешно.

Отметим, что тестирование клиентской части приложения проводится вручную, поэтому использование встроенных средств тестирования React не требуется.

4.3 Метрики кода

В качестве основной метрики используется Maintainability Index – комплексный показатель качества кода, который автоматически рассчитывается средствами WebStorm. Данная метрика зависит от других метрик, среди которых: объем программы (Halstead Volume, HV), цикломатическая сложность (Cyclomatic Complexity, CC) и количество строк кода (Lines of Code, LOC). Вычисляется данная метрика по формуле:

$$MI = (0, (171 - 5.2 * ln(HV) - 0.23 * CC - 16.2 * ln(LoC)) * 100 / 171).$$

Данный показатель принимает значения от 0 до 100, чем выше значение, тем более простой считается поддержка кода. Хорошим показателем считается значение от 20 до 100, приемлемым – от 10 до 20, плохим – менее 10.

После применения автоматизированного программного средства, встроенного в WebStorm, для анализа и расчёта метрик были получены следующие показатели, представленные в таблице 6.

Таблица 6 – Метрики кода для разработанного веб-приложения

Метрика	Значение
Размерно-ориентированные метрики	
Количество строк кода	6212
Логических строк кода	2838
Количество строк комментариев	184
Объектно-ориентированные метрики	
Количество классов	53
Количество методов	126
Сложность кода	
Средняя цикломатическая сложность по классу	14.1

Показатели Maintainability Index приведены в таблице 7.

Таблица 7 – Показатель качества кода основных компонентов

№	Компонент	MI
1	MusicShop.client.src.App	69
2	MusicShop.client.src.index	65
3	MusicShop.client.src.routes	63
4	MusicShop.client.src.components.AppRouter	64
5	MusicShop.client.src.components.DeviceItem	58
6	MusicShop.client.src.components.DeviceList	66
7	MusicShop.client.src.components.NavBar	67
8	MusicShop.client.src.components.Pages	63
9	MusicShop.client.src.components.TypeBar	72
10	MusicShop.client.src.components.models.CreateBrand	81
11	MusicShop.client.src.components.models.CreateDevice	100
12	MusicShop.client.src.components.models.CreateKind	100
13	MusicShop.client.src.components.models.CreateType	67
14	MusicShop.client.src.http.DeviceAPI	67
15	MusicShop.client.src.http.index	100
16	MusicShop.client.src.http.userAPI	100
17	MusicShop.client.src.pages.Admin	100
18	MusicShop.client.src.pages.Auth	100
19	MusicShop.client.src.pages.Basket	100
20	MusicShop.client.src.pages.DevicePage	100
21	MusicShop.client.src.pages.Shop	100
22	MusicShop.client.src.store.DeviceStore	100
23	MusicShop.client.src.store.UserStore	93
24	MusicShop.client.src.utils.consts	100
25	MusicShop.server.db	100
26	MusicShop.server.index	100
27	MusicShop.server.controllers.basketController	100
28	MusicShop.server.controllers.brandController	100
29	MusicShop.server.controllers.deviceController	48
30	MusicShop.server.controllers.kindController	60

Продолжение таблицы 7

№	Компонент	MI
31	MusicShop.server.controllers.typeController	75
32	MusicShop.server.controllers.userController	67
33	MusicShop.server.error.ApiError	100
34	MusicShop.server.middleware.authMiddleware	100
35	MusicShop.server. middleware.checkRoleMiddleware	100
36	MusicShop.server. middleware.ErrorHandlingMiddleware	100
37	MusicShop.server.models.models	100
38	MusicShop.server.error.ApiError	100
39	TaskForceX.Api.Helpers.PaginationHeader	93
	Средний показатель качества кода проекта	84,86

В результате проведения расчетов был получен средний показатель качества кода проекта. Подводя итог, можно сказать, что проект имеет высокий показатель качества кода.

5 Планирование разработки и оценка бюджета

5.1 Календарное и ресурсное планирование

Неотъемлемой частью проектирования и разработки автоматизированной системы является календарное и ресурсное планирование проекта, которое позволяет определить, какие ресурсы будут задействованы в проекте, и каким образом они будут использоваться. Планирование позволяет осуществлять управление ходом работ по созданию информационной системы.

Календарный план проекта содержит определение длительности работ и их взаимосвязей. Диаграмма Ганта позволяет в удобной графической форме представить сроки работ по проекту, продолжительность этапов, список требуемых ресурсов и т. д.

Кроме перечня работ в календарное планирования входит составление ресурсной модели проекта. Среди всех ресурсов можно выделить несколько основных категорий: трудовые, материальные и невозобновляемые. Ресурсы по-своему влияют на их использование и затраты, которые и составляют бюджет проекта.

В целом, календарное и ресурсное планирование являются важными элементами проектирования и разработки автоматизированной системы. Они позволяют определить необходимые ресурсы, распределить их и управлять ходом работ, что в свою очередь способствует успешной реализации проекта в соответствии с поставленными целями и требованиями.

Для получения плана-графика проекта с оптимальным использованием ресурсов и минимальными сроками проведем календарное планирование.

Составим план стадий и этапов создания веб-приложения. План, содержащий подробную информацию о наименовании и длительности стадий и этапов разработки, а также лиц, выполняющих их, представлен в приложении И.

Для составления плана будет использоваться программа Microsoft Project. Создадим диаграмму Ганта (приложение К).

Для наглядности составим информацию о ресурсах, используемых при разработки проекта.

Информация о ресурсах представлена на рисунке 19.

	Название ресурса	Тип	Единицы измерения материалов	Краткое название	Группа	Макс. единиц	Стандартная ставка	Ставка сверхурочных
1	Руководитель проекта	Трудовой		РП		100%	50 000,00 ₽/мес	350,00 ₽/ч
2	Ведущий программист	Трудовой		ВП1		100%	40 000,00 ₽/мес	300,00 ₽/ч
3	Программист	Трудовой		П1		100%	28 000,00 ₽/мес	240,00 ₽/ч
4	Тестировщик	Трудовой		Т1		100%	21 500,00 ₽/мес	180,00 ₽/ч
5	Интернет	Затраты		И				
6	Компьютер	Материальный		К			110 000,00 ₽	
7	Принтер	Материальный		П			25 000,00 ₽	
8	Бумага	Материальный		Б			600,00 ₽	

Рисунок 19 – Лист ресурсов проекта

Проанализировав этапы и их продолжительность можно сделать вывод, что на проектирование и разработку рассматриваемого программного продукта потребуется 312 часов.

5.2 Расчет бюджета

Проведем расчет стоимости проектирования и разработки проекта. Для расчета затрат необходимо учесть следующие статьи расходов:

– амортизация;
– материалы;
– фонды основной и дополнительной заработной платы;
– социальные выплаты;
– прочие затраты.

Произведем расчет амортизации.

Амортизация – это процесс перенесения по частям стоимости изношенной части основных производственных фондов (ОПФ) на стоимость готовой продукции.

Стоимость ОПФ и расчет амортизационных отчислений приведены в таблице 8.

Таблица 8 – Амортизационные отчисления

Оборудование	Стоимость, руб.	СПИ, лет	НА, %	А, год, руб.
Компьютер с ПО: MS Office 2019; MS Windows 10	110 000	5	20	22 000
Принтер Hp	25 000	5	20	5 000
ТЗР (1%)	1350			
Итого	136350			27 000

Обозначения в таблице:

– СПИ: срок полезного использования;

– НА: норма амортизации;

– А: амортизация;

– ТЗР: транспортно-заготовительные расходы.

Для разработки программного продукта планируется использовать программное обеспечение в свободном доступе: WebStorm 2023.1.2, PgAdmin 4, PostgreSQL, веб-фреймворк Express [6].

Амортизационные отчисления рассчитываются по следующей формуле:

$$А = С_{пер} * НА,$$

где А – амортизационные начисления (руб.);

$С_{пер}$ – первоначальная стоимость основных средств (руб.);

НА – норма амортизации, которая находится по формуле (руб.).

$$НА = 1 / СПИ,$$

где СПИ – срок полезного использования основных средств.

Срок полезного использования компьютера = 5 лет.

$$НА_{комп} = 1 / 5 = 0,2.$$

Срок полезного использования принтера = 5 лет.

$$НА_{принт} = 1 / 5 = 0,2.$$

$$А_{комп} = 110\,000 * 0,2 = 22\,000 \text{ (руб.)}.$$

$$А_{принт} = 25\,000 * 0,2 = 5000 \text{ (руб.)}.$$

$$А_{г} = 22\,000 + 5000 + 1350 = 28\,350 \text{ (руб.)}.$$

$$А_{м} = 28\,350 / 12 = 2362,5 \text{ (руб/мес)}.$$

$$А_{итог} = 2362,5 * 1 = 2362,5 \text{ (руб.)}.$$

Работа по созданию программного продукта предполагается вести 312 часов, расчёт ведётся за 1 месяц.

Произведем расчет материалов.

В данной статье приведены расчеты оборотных средств, использовавшиеся для разработки программного продукта, они приведены в таблице 9.

Таблица 9 – Затраты по статье «Материалы»

Наименование	Количество, шт.	Сумма, руб.
Бумага	2	1200
ТЗР (1%)		12
ИТОГО:		1212

Фонд основной и дополнительной заработной платы.

Зарплата – вознаграждение за труд в зависимости от квалификации работника, сложности, количества, качества и условий выполняемой работы.

Составим таблицу расчета основной и дополнительной заработной платы, а так же определим трудоемкость для каждого специалиста, участвующего в разработке проекта.

Трудоемкость всех видов работ, связанных с разработкой рассматриваемого программного продукта представлена в таблице 10.

Таблица 10 – Расчет основной и дополнительной заработной платы

Исполнитель	Трудоемкость, часов (Тр)	Ставка, руб./час	ОЗП, руб.	ДЗП, руб.	Соц. выплаты
Руководитель проекта	88	670	58 958,24	5 895,82	19 456,22
Ведущий программист	104	580	60 320,44	6 032,04	19 905,74
Программист	64	490	31360,20	3 136,02	10 348,87
Тестировщик	56	450	25 200,55	2 520,05	8 316,18
Итого:	312		175 839,43	17 583,93	58027,01

Всего на разработку и тестирование программного продукта затрачено 312 часов.

Рассмотрим оплату труда на примере расчета заработной платы руководителя проекта.

Оплата за труд рассчитывается следующим образом:

$$ОЗП = Сч * Тр,$$

где Тр – трудоемкость;

Сч – тарифная ставка, которая рассчитывается по формуле:

$$Сч = (МРОТ * Тк)/160,$$

где МРОТ – минимальный размер оплаты труды = 16 242 (руб.);

Тк – тарифный коэффициент;

160 – количество часов в месяц по ТК РФ.

$$Ткрук = 6,6.$$

$$Счрук = (16\ 242 * 6,6)/160 = 669,98.$$

$$ОЗПрук = 669,98 * 88 = 58\ 958,24 \text{ руб.}$$

Мотивация труда оценивается через дополнительную заработную плату, которая рассчитывается по формуле:

$$ДЗП = 10\% ОЗП.$$

$$ДЗП рук = 10\% * 58\,958{,}24 = 5\,895{,}82 руб.$$

Произведем расчет социальных выплат.

Социальные выплаты состоят из:

– фонд социального страхования (ФСС) – 5,9%;

– пенсионный фонд РФ (ПФР) – 22%;

– фонд обязательного медицинского страхования (ФОМС) – 2,1%.

$$СВ = 30\% (ОЗП + ДЗП).$$

$$СВ рук = 30\% * (58\,958{,}24 + 5\,895{,}82) = 19\,456{,}22 руб.$$

Расчитаем затраты на доступ в Интернет и электроэнергию (ЭЭ).

ЭЭ рассчитывается по формуле:

$$ЭЭ = Тс * W * Траб,$$

где Тс = стоимость 1 кв/ч = 4,87 (руб.);

W – энергомощность компьютера = 0,4 (кВ/ч);

Траб – время создания (ч).

Так как программный продукт разрабатывался 312 часов, рассчитаем затраты на электроэнергию:

$$ЭЭ = 4{,}87 *0{,}4*312 = 607{,}77 руб.$$

Произведем расчет стоимости услуги доступа в Интернет.

Работа над программным продуктом велась 312 часов, тариф за услугу доступ в Интернет составляет 900 руб/мес. Однако, провайдер не предоставляет почасовой доступ в интернет, поэтому расчет ведется за 1 месяца.

$$Интернет = 900 * 1 = 900 руб.$$

$$Прочие затраты = ЭЭ + Интернет = 607{,}77 + 900 = 1507{,}77 руб.$$

Структура затрат представлена в таблице 11.

Таблица 11 – Структура затрат

Наименование статей	Затраты, руб.	%
Амортизация	2362,5	0,92%
Материалы	1212	0,47%
Фонд основной и дополнительной ЗП	193 423,36	74,4%
Социальные выплаты	58027,01	22,62%
Прочие	1507,77	0,59%
Затраты на проект	256 532,64	

Диаграмма структуры затрат представлена на рисунке 20.

Рисунок 20 – Структура затрат

Проводя анализ структуры расходов можно сделать вывод, что основными статьями расходов является фонд оплаты труда и социальные выплаты. Затраты на проектирование и разработку программного продукта «Веб-приложение музыкального магазина» составят 256 532,64 рублей.

Заключение

В процессе выполнения выпускной квалификационной работы были изучены особенности, принципы разработки и использования веб-приложений. Были проанализированы аналоги приложений музыкальных магазинов, а также рассмотрены актуальные технические материалы по проектированию и разработке информационных систем для Интернета.

Проведенный анализ существующих web-приложений музыкальных магазинов показал наличие недостатков, таких как перегруженный интерфейс, рекламные баннеры, недоступность для заказчика и отсутствие аудио-представления товаров. Целью работы было создание более доступного и удобного как для пользователя, так и для заказчика приложения, в котором были учтены все аспекты работы с ним и реализованы основные функции, такие как добавление и наполнения товаров, добавление товаров в корзину и воспроизведение звучания музыкальных инструментов.

В процессе проектирования были определены структуры хранения данных и пользовательского интерфейса, а также составлены диаграммы вариантов использования, подробное описание прецедентов и последовательности действий для некоторых вариантов использования. Были выбраны средства и инструменты разработки веб-приложений и разработаны макеты интерфейса клиентской части приложения. Также была спроектирована структура хранения данных и описаны таблицы базы данных.

После проектирования было создано полноценное Rest API-приложение музыкального магазина, которое позволяет пользователям сортировать музыкальные инструменты по типу и бренду, слушать звучание музыкальных инструментов, добавлять товары в корзину, добавлять новые типы, виды, бренды и товары. Также была реализована возможность добавления музыкального инструмента с аудио-файлом, имитирующим звучание инструмента. Доступ к функционалу добавления новых элементов имеют только пользователи с ролью администратора.

Для обеспечения безопасности было осуществлено шифрование пароля пользователей.

Для проверки качества были применены методологии тестирования, такие как функциональное, структурное и модульное тестирование. Главные классы приложения были покрыты модульными тестами и показали отсутствие ошибок. В результате можно сделать вывод о том, что приложение работает без ошибок и соответствует заявленным требованиям.

Также было проведено планирование проекта с выделением длительности работ и взаимосвязей между ними. Составлен план стадий и этапов проектирования и разработки веб-приложения, а на его основе была построена диаграмма Ганта. Анализ этапов и их продолжительности показал, что на проектирование и разработку системы понадобится 312 часов. Также была произведена оценка стоимости проекта, учитывая все расходы на амортизацию, зарплату, социальные выплаты и материальные расходы. В результате расчета стоимость проекта составила 256 532,64 рублей.

Подводя итог, можно сказать, что цель работы была достигнута, было создано полноценное веб-приложение музыкального магазина, соответствующее заявленным требованиям и учитывающее все аспекты работы с приложением. Были проведены исследования особенностей работы, принципов разработки и использования веб-приложений, проанализированы самые популярные приложения музыкальных магазинов и выявлены их недостатки. В результате проектирования была определена структура приложения, спроектированы программные средства, структуры хранения данных и пользовательского интерфейса, а также разработаны макеты интерфейса клиентской части приложения.

Таким образом, проект был успешно завершен, и созданное приложение соответствует заявленным требованиям.

Список использованных источников

1. Amazon.com [Электронный ресурс]. URL: https://aws.amazon.com/ru/what-is/web-application/ (дата обращения: 19.04.2023).

2. Muztorg.ru [Электронный ресурс]. URL: https://www.muztorg.ru/ (дата обращения: 21.04.2023).

3. Jazz-shop.ru [Электронный ресурс]. URL: https://jazz-shop.ru/ (дата обращения: 21.04.2023).

4. Mirm.ru [Электронный ресурс]. URL: https://mirm.ru/ (дата обращения: 21.04.2023).

5. Буч, Г. Язык UML. Руководство пользователя. 2-е изд.: Пер. с англ. Мухин Н. / Г. Буч, Д. Рамбо, И. Якобсон. – М.: ДМК Пресс, 2006. – 496 с.: ил.

6. Алекс, Бэнкс React и Redux. Функциональная веб-разработка. Руководство / Бэнкс Алекс. – М.: Питер, 2018. – 458 с.800 с.

7. Пьюривал, С. Основы разработки веб-приложений / С. Пьюривал. – Спб.: Питер, 2015. – 272 с.

8. Брайан, П. Хоган HTML5 и CSS3. Веб-разработка по стандартам нового поколения / Брайан П. Хоган. – М.: Питер, 2013. – 128 с.

9. Дакетт, Д. HTML и CSS. Разработка и дизайн веб-сайтов / Д. Дакетт. – М: Эксмо, 2013. – 480 с.

10. Дэвид, Хэррон Node.js. Разработка серверных веб-приложений на JavaScript / Хэррон Дэвид. – М.: ДМК Пресс, 2016. – 667 с.

11. Expressjs.com [Электронный ресурс]. URL: https://expressjs.com/ (дата обращения: 25.04.2023).

12. Стоунз PostgreSQL. Основы / Стоунз, Мэттью Ричард; , Нейл. – М.: СПб: Символ-Плюс, 2002. – 640 с.

13. Никсон Робин Создаем динамические веб-сайты с помощью PHP, MySQL, JavaScript, CSS и HTML5; Питер – М., 2016. – 768 с.

14. Reactjs.org [Электронный ресурс]. URL: https://ru.legacy.reactjs.org (дата обращения: 26.04.2023).

15. Runebook.dev [Электронный ресурс]. URL: https://runebook.dev/ru/docs/react_bootstrap/getting-started/why-react-bootstrap/index/ (дата обращения: 30.04.2023).

16. Дакетт, Джон Javascript и jQuery. Интерактивная веб-разработка / Джон Дакетт. – М.: Эксмо, 2014. – 146 с.

17. Webformyself.com [Электронный ресурс]. URL: https://webformyself.com/ponimanie-react-router-na-primere-prostogo-bloga/ (дата обращения: 11.05.2023).

18. Jazzteam.ru [Электронный ресурс]. URL: https://jazzteam.ru/technical-articles/introduction-to-mobx-5/ (дата обращения: 27.04.2023).

Приложение А

Подробное описание таблиц базы данных

(рекомендуемое)

Таблица А.1 – Таблица «users» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	12
email	Электронная почта	varchar(50)	нет		drm@mail.ru
password	Пароль	varchar(32)	нет		Qwerty123
role	Роль	varchar(10)	нет		USER

Таблица А.2 – Таблица «products» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	16
name	Название	varchar(255)	нет		SIGMA DM-ST
price	Цена	integer(10)	нет		13299
description	Описание	varchar(255)	да		Состоит из дерева
img	Изображение	varchar(255)	нет		pic.img
audio	Аудио файл	varchar(255)	нет		music.mp3
brandsid	Идентификатор бренда	integer(10)	нет	FK	12
typesid	Идентификатор типа	integer(10)	нет	FK	3

Таблица А.3 – Таблица «product_infos» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	7
title	Название характеристики	varchar(255)	нет		Вес:
description	Значение характеристики	varchar(255)	нет		700гр.
productsid	Идентификатор товара	integer(10)	нет	FK	12

Таблица А.4 – Таблица «types» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	1
name	Название типа	varchar(255)	нет		Акустическая гитара
kindsid	Идентификатор вида	integer(10)	нет	FK	1

Таблица А.5 – Таблица «kinds» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	1
name	Название вида	varchar(255)	нет		Гитара

Таблица А.6 – Таблица «brands» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	1
name	Название бренда	varchar(255)	нет		Aria

Таблица А.7 – Таблица «baskets» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	3
usersid	Идентификатор пользователя	integer(10)	нет	FK	14

Таблица А.8 – Таблица «basket_product» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	7
basketsid	Идентификатор корзины	integer(10)	нет	FK	11
productsid	Идентификатор товара	integer(10)	нет	FK	15

Таблица А.9 – Таблица «types_brands» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
typesid	Идентификатор типа	integer(10)	нет	FK	3
brandsid	Идентификатор бренда	integer(10)	нет	FK	12

Приложение Б

Листинг кода файла index.js

(рекомендуемое)

```
require('dotenv').config()

const express = require('express')
const sequelize = require('./db')
const PORT = process.env.PORT || 5000
const models = require('./models/models')
const cors = require('cors')
const fileUpload = require('express-fileupload')
const router = require('./routes/index')
const errorHandler =
require('./middleware/ErrorHandlingMiddleware')
const path = require('path')

const app = express()
app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload({}))
app.use('/api', router)

app.use(errorHandler)//обработка ошибок, последний Middleware

app.get('/', (req, res) => {
    res.status(200).json({message: "good work"})
})

const start = async () => {
    try {
        await sequelize.authenticate()
        await sequelize.sync()
        app.listen(PORT, () => console.log(`Server started on port
${PORT}`))
    } catch (e) {
        console.log(e)
    }
}

start()
```

Приложение В

Листинг кода файла deviceController.js

(рекомендуемое)

```
const uuid = require('uuid')
const path = require('path')
const {Device, DeviceInfo, Type} = require('../models/models')
const ApiError = require('../error/ApiError')

class DeviceController {
    async create(req,res,next){
        try {
        let {name, price, description, brandId, typeId, info} =
req.body
        const {img} = req.files
        const {audio} = req.files
        let imgName = uuid.v4() + ".jpeg"
        let audioName = uuid.v4() + ".mp3"
        img.mv(path.resolve(__dirname, '..', 'static', imgName))
            audio.mv(path.resolve(__dirname, '..', 'static',
audioName))

            const device = await Device.create({name, price,
description, brandId, typeId, img:imgName, audio:audioName})

            if (info) {
                info = JSON.parse(info)
                info.forEach( i =>
                    DeviceInfo.create({
                        title: i.title,
                        description: i.description,
                        deviceId: device.id
                    })
                )
            }


        return res.json(device)
        } catch (e) {
            next(ApiError.badRequest(e.message))
        }

    }

    async getAll(req,res){
        try {
            let {brandId, typeId, limit, page} = req.query
            page = page || 1
            limit = limit || 9
            let offset = page * limit - limit
            let devices;
            if (!brandId && !typeId) {
                devices = await Device.findAndCountAll({limit,
offset})
            }
            if (brandId && !typeId) {
                devices = await Device.findAndCountAll({where:
```

```
{brandId}, limit, offset})
            }
            if (!brandId && typeId) {
                devices = await Device.findAndCountAll({where:
{typeId}, limit, offset})
            }
            if (brandId && typeId) {
                devices = await Device.findAndCountAll({where:
{typeId, brandId}, limit, offset})
            }
            return res.json(devices)
        } catch(e) {
            console.log(e)
        }
    }

    async getOne(req,res){
        const {id} = req.params
        const device = await Device.findOne(
            {
                where: {id},
                include: [{model: DeviceInfo, as: 'info'}]
            },
        )
        return res.json(device)
    }

}

module.exports = new DeviceController()
```

Приложение Г

Листинг кода файла userController.js

(рекомендуемое)

```
const ApiError = require('../error/ApiError')
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const {User, Basket} = require('../models/models')
const generateJwt = (id, email, role) => {
    return jwt.sign(
        {id, email, role},
        process.env.SECRET_KEY,
        {expiresIn: '24h'}
    )
}
class UserController {
    async registration(req,res,next){
        const {email, password, role} = req.body
        if (!email || !password) {
            return next(ApiError.badRequest('Некорректный email
или password'))
        }
        const candidate = await User.findOne({where: {email}})
        if (candidate) {
            return next(ApiError.badRequest('Пользователь с таким
email уже существует'))
        }
        const hashPassword = await bcrypt.hash(password, 5)
        const user = await User.create({email, role, password:
hashPassword})
        const basket = await Basket.create({userId: user.id})
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }
    async login(req,res, next ){
        const {email, password} = req.body
        const user = await User.findOne({where: {email}})
        if (!user) {
            return next(ApiError.internal('Пользователь не
найден'))
        }
        let comparePassword = bcrypt.compareSync(password,
user.password)
        if (!comparePassword) {
            return next(ApiError.internal('Указан неверный
пароль'))
        }
        const token = generateJwt(user.id, user.email, user.role)
        return res.json({token})
    }
    async check(req,res,next){
        const token = generateJwt(req.user.id, req.user.email,
req.user.role)
        return res.json({token})
    }
}
module.exports = new UserController()
```

Приложение Д

Листинг кода файла models.js

(рекомендуемое)

```
const sequelize = require('../db')
const {DataTypes} = require('sequelize')

const User = sequelize.define('user', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    email: {type: DataTypes.STRING, unique: true,},
    password: {type: DataTypes.STRING},
    role: {type: DataTypes.STRING, defaultValue: "USER"},
})

const Basket = sequelize.define('basket', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
})

const BasketDevice = sequelize.define('basket_device', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
})

const Device = sequelize.define('device', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    name: {type: DataTypes.STRING, unique: true, allowNull:
false},
    price: {type: DataTypes.INTEGER, allowNull: false},
    description: {type: DataTypes.STRING, allowNull: true},
    img: {type: DataTypes.STRING, allowNull: false},
    audio: {type: DataTypes.STRING, allowNull: false},
})

const Type = sequelize.define('type', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    name: {type: DataTypes.STRING, unique: true, allowNull:
false},
})

const Kind = sequelize.define('kind', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    name: {type: DataTypes.STRING, unique: true, allowNull:
false},
})

const Brand = sequelize.define('brand', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    name: {type: DataTypes.STRING, unique: true, allowNull:
false},
})
```

```
const DeviceInfo = sequelize.define('device_info', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
    title: {type: DataTypes.STRING, allowNull: false},
    description: {type: DataTypes.STRING, allowNull: false},
})

const TypeBrand = sequelize.define('type_brand', {
    id: {type: DataTypes.INTEGER, primaryKey: true, autoIncrement:
true},
})


User.hasOne(Basket)
Basket.belongsTo(User)

Basket.hasMany(BasketDevice)
BasketDevice.belongsTo(Basket)

Kind.hasMany(Type)
Type.belongsTo(Kind)

Type.hasMany(Device)
Device.belongsTo(Type)

Brand.hasMany(Device)
Device.belongsTo(Brand)

Device.hasMany(BasketDevice)
BasketDevice.belongsTo(Device)

Device.hasMany(DeviceInfo, {as: 'info'});
DeviceInfo.belongsTo(Device)

Type.belongsToMany(Brand, {through: TypeBrand })
Brand.belongsToMany(Type, {through: TypeBrand })

module.exports = {
    User,
    Basket,
    BasketDevice,
    Device,
    Type,
    Kind,
    Brand,
    TypeBrand,
    DeviceInfo
}
```

Приложение Е

Листинг кода файла userAPI.js

(рекомендуемое)

```
import {$authHost, $host} from "./index";
import jwt_decode from "jwt-decode"
export const registration = async (email, password) => {
    const {data} = await $host.post('api/user/registration',
{email, password, role: 'USER'})
    localStorage.setItem('token', data.token)
    return jwt_decode(data.token)
}
export const login = async (email, password) => {
    const {data} = await $host.post('api/user/login', {email,
password})
    localStorage.setItem('token', data.token)
    return jwt_decode(data.token)
}
export const check = async () => {
    const {data} = await $authHost.get('api/user/auth',)
    localStorage.setItem('token', data.token)
    return jwt_decode(data.token)
}
```

Приложение Ж

Листинг кода файла DeviceAPI.js

(рекомендуемое)

```
import {$authHost, $host} from "./index";
export const createType = async (type) => {
    const {data} = await $authHost.post('api/type', type)
    return data
}
export const fetchTypes = async () => {
    const {data} = await $host.get('api/type')
    return data
}
export const createKind = async (type) => {
    const {data} = await $authHost.post('api/kind', type)
    return data
}
export const fetchKinds = async () => {
    const {data} = await $host.get('api/kind')
    return data
}
export const createBrand = async (brand) => {
    const {data} = await $authHost.post('api/brand', brand)
    return data
}
export const fetchBrands = async () => {
    const {data} = await $host.get('api/brand')
    return data
}
export const createDevice = async (device) => {
    const {data} = await $authHost.post('api/device', device)
    return data
}
export const fetchDevices = async (typeId, brandId, page, limit =
5) => {
    const {data} = await $host.get('api/device',{params:{
            typeId, brandId, page, limit
        }})
    return data
}
export const fetchOneDevice = async (id) => {
    const {data} = await $host.get('api/device/' + id)
    return data
}
export const addToBasket = async (deviceId) => {
    const {response} = await $authHost.post('api/basket',
deviceId)
    return response
}
export const getBasket = async () => {
    const {data} = await $authHost.get('api/basket')
    return data
}
```

Приложение И

Распределение трудовых ресурсов

(рекомендуемое)

Таблица И.1 – Распределение трудовых ресурсов

Название задачи	Затраты времени
Анализ предметной области	56 ч
Исследование вариантов использования	16 ч
– Руководитель проекта	16 ч
Обзор существующих решений	8 ч
– Руководитель проекта	8 ч
Выбор программно-аппаратных средств	32 ч
– Руководитель проекта	8 ч
– Ведущий программист	8 ч
– Программист	8 ч
– Тестировщик	8 ч
Техническое задание	64 ч
Анализ требований к системе	16 ч
– Руководитель проекта	16 ч
Составление технического задания	48 ч
– Руководитель проекта	16 ч
– Ведущий программист	16 ч
– Программист	16 ч
Программирование	112 ч
Разработка серверной части приложения	64 ч
Постановка задачи	24 ч
– Руководитель проекта	8 ч
– Ведущий программист	8 ч
– Программист	8 ч
Разработка структуры базы данных	24 ч
– Ведущий программист	24 ч
Заполнение базы данных	16 ч
– Ведущий программист	16 ч
Разработка клиентской части приложения	48 ч
Постановка задачи	16 ч

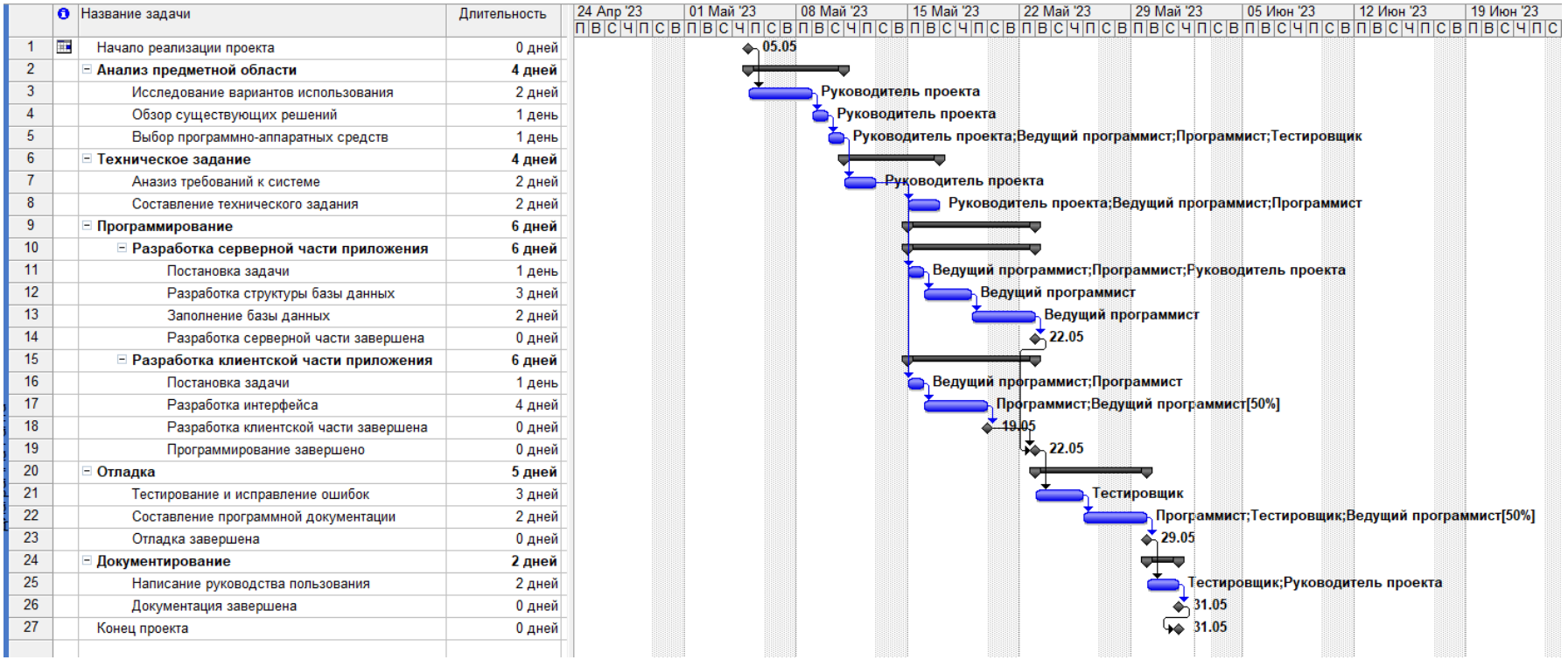
Продолжение таблицы И.1

Название задачи	Затраты времени
– Ведущий программист	8 ч
– Программист	8 ч
Разработка интерфейса	32 ч
– Ведущий программист	16 ч
– Программист	16 ч
Отладка	48 ч
Тестирование и исправление ошибок	24 ч
– Тестировщик	24 ч
Составление программной документации	24 ч
– Ведущий программист	8 ч
– Программист	8 ч
– Тестировщик	8 ч
Документирование	32 ч
Написание руководства пользования	32 ч
– Руководитель проекта	16 ч
Итого	312 ч

Приложение К

Диаграмма Ганта

(рекомендуемое)



	Название задачи	Длительность
1	Начало реализации проекта	0 дней
2	**Анализ предметной области**	**4 дней**
3	Исследование вариантов использования	2 дней
4	Обзор существующих решений	1 день
5	Выбор программно-аппаратных средств	1 день
6	**Техническое задание**	**4 дней**
7	Анализ требований к системе	2 дней
8	Составление технического задания	2 дней
9	**Программирование**	**6 дней**
10	**Разработка серверной части приложения**	**6 дней**
11	Постановка задачи	1 день
12	Разработка структуры базы данных	3 дней
13	Заполнение базы данных	2 дней
14	Разработка серверной части завершена	0 дней
15	**Разработка клиентской части приложения**	**6 дней**
16	Постановка задачи	1 день
17	Разработка интерфейса	4 дней
18	Разработка клиентской части завершена	0 дней
19	Программирование завершено	0 дней
20	**Отладка**	**5 дней**
21	Тестирование и исправление ошибок	3 дней
22	Составление программной документации	2 дней
23	Отладка завершена	0 дней
24	**Документирование**	**2 дней**
25	Написание руководства пользования	2 дней
26	Документация завершена	0 дней
27	Конец проекта	0 дней

Рисунок К.1 – Диаграмма Ганта