

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

Программная реализация web-приложения

«МУЗЫКАЛЬНАЯ МАСТЕРСКАЯ»

Студент

(подпись, дата) Румбенс Артём Дмитриевич
(ФИО полностью)

Руководитель

(подпись, дата) Кольчугина Е.А.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата) Такташкин Д.В.
(фамилия, инициалы)

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

Заведующий кафедрой

(подпись) Козлов А.Ю.
(фамилия, инициалы)

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

Секретарь ГЭК

(подпись) Попова Н.А.
(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 Румбенс Артём Дмитриевич гр. 19ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программная реализация web-приложения «Музыкальная мастерская»

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

3. Руководитель работы Кольчугина Е.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Назначение: программа предназначена для регистрации заявок на ремонт музыкальных инструментов

2) Средства разработки: язык программирования JavaScript, фреймворк React, Node.js интегрированная среда разработки VisualStudio code

3) Основные функции программы:

а) Регистрация заявок на ремонт инструментов

б) Просмотр заявок;

в) Изменение статуса заявок

г) Авторизация и регистрация пользователей

д) Просмотр инструментов, подлежащих ремонту

5. Перечень подлежащих разработке вопросов

Анализ предметной области и постановка задачи

Анализ требований на разработку

Проектирование приложения
Реализация приложения
Тестирование и контроль качества приложения
Планирование разработки и оценка бюджета приложения
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	5%	01.04.2023 – 08.04.2023	
2	Анализ требований на разработку	5%	08.04.2023 – 17.04.2023	
3	Проектирование приложения	20%	17.04.2023 – 26.04.2023	
4	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
5	Тестирование и контроль качества приложения	10%	12.05.2023 – 19.05.2023	
6	Планирование разработки и оценки бюджета	10%	19.05.2023 – 2.05.2023	
7	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 ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 71 лист, 13 таблиц, 21 рисунок, 16 использованных источников и 6 приложений.

WEB – ПРИЛОЖЕНИЕ, ПРОЕКТИРОВАНИЕ, РАЗРАБОТКА, ТЕСТИРОВАНИЕ, ПЛАНИРОВАНИЕ, ОЦЕНКА БЮДЖЕТА,

Объект разработки – Web-приложение «Музыкальная мастерская».

Цель работы – провести анализ предметной области и поставить задачу на разработку, провести сравнительный анализ конкурирующего программного обеспечения, определиться с инструментальными средствами, спроектировать диаграммы UML и пользовательский интерфейс, разработать программное обеспечение, провести его тестирование и анализ кода с помощью различных метрик, спланировать бюджет разработки.

Средства разработки: язык программирования JavaScript, фреймворки React, Node.js интегрированная среда разработки VisualStudio code.

					ПГУ 09.03.04 – 08БР191.10 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Румбенс А.Д.			Программная реализация web-приложения «Музыкальная мастерская»	Лит.	Лист	Листов
Провер.		Кольчугина Е.А.					4	71
Н. Контр.		Такташкин Д.В.			Пояснительная записка	Группа 19ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Содержание	5
Введение.....	7
1 Постановка задачи и анализ требований	8
1.1 Постановка задачи	8
1.2 Анализ программного обеспечения аналогичного назначения	9
1.3 Сравнение аналогов	12
1.4 Анализ функциональных требований	13
1.5 Анализ требований к пользовательскому интерфейсу.....	19
2 Проектирование web-приложения музыкальной мастерской.....	20
2.1 Проектирование пользовательского интерфейса	20
2.2 Выбор программно-аппаратных средств	25
2.4 Структуры хранения данных.....	30
3 Реализация web-приложения «Музыкальная мастерская»	34
3.1 Кодирование web-приложения	34
3.2 Развертывание web-приложения	39
4 Тестирование программного обеспечения.....	40
4.1 Обоснование выбора тестирования	40
4.2 Тестирование	41
4.3 Метрики кода.....	45
4.3.2 Метрики Холстеда	46
4.3.3 Комплексный показатель качества кода	47
5 Расчет бюджета	48
5.1 Планирование и оценка бюджета проекта	48
Заключение.....	52
Список использованных источников	53
Приложение А	55
Листинг кода файла index.js.....	55
Приложение Б	57
Листинг кода файла RequestController.js.....	57
Приложение В	60
Листинг кода файла userController.js.....	60
Приложение Г	63
Листинг кода файла models.js	63
Приложение Д	67

Листинг кода файла userAPI.js.....	67
Приложение Е.....	69
Листинг кода файла RequestAPI.js.....	69

Введение

В современном мире музыка стала неотъемлемой частью большинства людей. Почти у каждого человека имеется какой-либо музыкальный инструмент. Однако инструменты недолговечны, периодически они ломаются.

Для решения подобной проблемы создаются музыкальные мастерские, в которых производится ремонт инструментов. Но любая современная компания обязана иметь веб-приложение, иначе занять лидирующие позиции в бизнесе будет невозможно.

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 наличие спроса клиентов на такие приложения и большое количество музыкальных мастерских, веб-приложений с такой тематикой не много. Данный факт делает наличие у компаний веб-приложений востребованным.

Помимо всего прочего, это направление является весьма перспективным. Компания, имея подобное web-приложение может получить большую прибыль и приток клиентов.

Основными плюсами веб-приложения являются дешевое производство и простое взаимодействие с клиентом.

Задачами данного проекта являются:

- анализ требований на разработку системы;
- анализ приложений схожего назначения;
- разработка информационного и программного приложения;
- тестирование программного обеспечения;
- планирование и расчет бюджета.

1 Постановка задачи и анализ требований

1.1 Постановка задачи

Веб-приложения по ремонту музыкальных инструментов – это программные решения, которые помогают музыкантам и ремонтным мастерским упростить и автоматизировать процесс обслуживания и ремонта музыкальных инструментов.

Веб-приложения предоставляют услуги онлайн-заказа ремонта, позволяя клиентам быстро и удобно найти мастерскую для ремонта своего инструмента.

Онлайн-заказ ремонта музыкальных инструментов имеет ряд преимуществ перед традиционным способом обращения в ремонтную мастерскую.

Удобство: клиенты могут заказать ремонт своего инструмента онлайн, не выходя из дома или офиса. Это экономит время и усилия, которые были бы потрачены на личный визит в мастерскую;

Доступность: онлайн-заказ ремонта доступен 24/7, что позволяет клиентам заказывать ремонт в удобное для них время, в том числе в выходные и праздничные дни;

Быстрота: в отличие от традиционных мастерских, где может быть длинная очередь на ремонт, онлайн-заказ позволяет быстро оформить заявку и начать процесс ремонта.

В целом, веб-приложения по ремонту музыкальных инструментов помогают сделать процесс обслуживания и ремонта более доступным, удобным и эффективным для всех участников этого процесса.

1.2 Анализ программного обеспечения аналогичного назначения

Для анализа необходимо определить аналоги и провести сравнение.

Были выбраны следующие веб-приложения:

- Гитарная лаборатория;
- Techno Guitars;
- Dr. Sound.

Рассмотрим каждый из этих сервисов ниже.

Гитарная лаборатория – отличная мастерская, потрясающие профессионалы своего дела, способные починить любую гитару [2].

Преимущества:

- понятный интерфейс;
- отсутствие рекламы;
- служба поддержки.

Недостатки:

- перегруженный интерфейс;
- узкая направленность.

На рисунке 1 представлен интерфейс данного программного обеспечения.

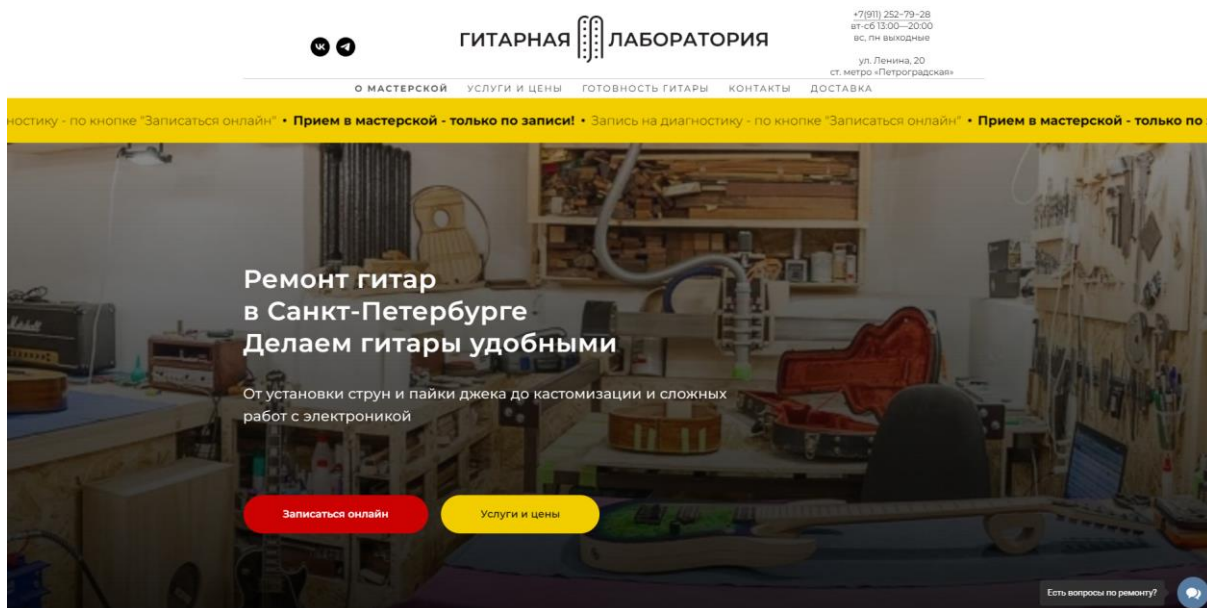


Рисунок 1 – Интерфейс веб-приложения Гитарная лаборатория

Основным достоинством данного веб-приложения является понятный интерфейс. Сразу видна вся необходимая информация и весь функционал.

Главным недостатком же является узкая направленность. Данная компания занимается только ремонтом гитар, остальные инструменты не обслуживаются.

Techno Guitars – мастера, занимающиеся отстройкой, реставрацией, ремонтом акустических, электро и бас гитар, шлифовкой и заменой ладов, регулировкой прогиба грифа и многим другим [3].

Преимущества:

- отсутствие рекламы;
- круглосуточная поддержка.

Недостатки:

- непонятный интерфейс;
- узкая направленность;
- мало информации;
- долгая загрузка приложения.

На рисунке 2 представлен интерфейс данного веб-приложения.

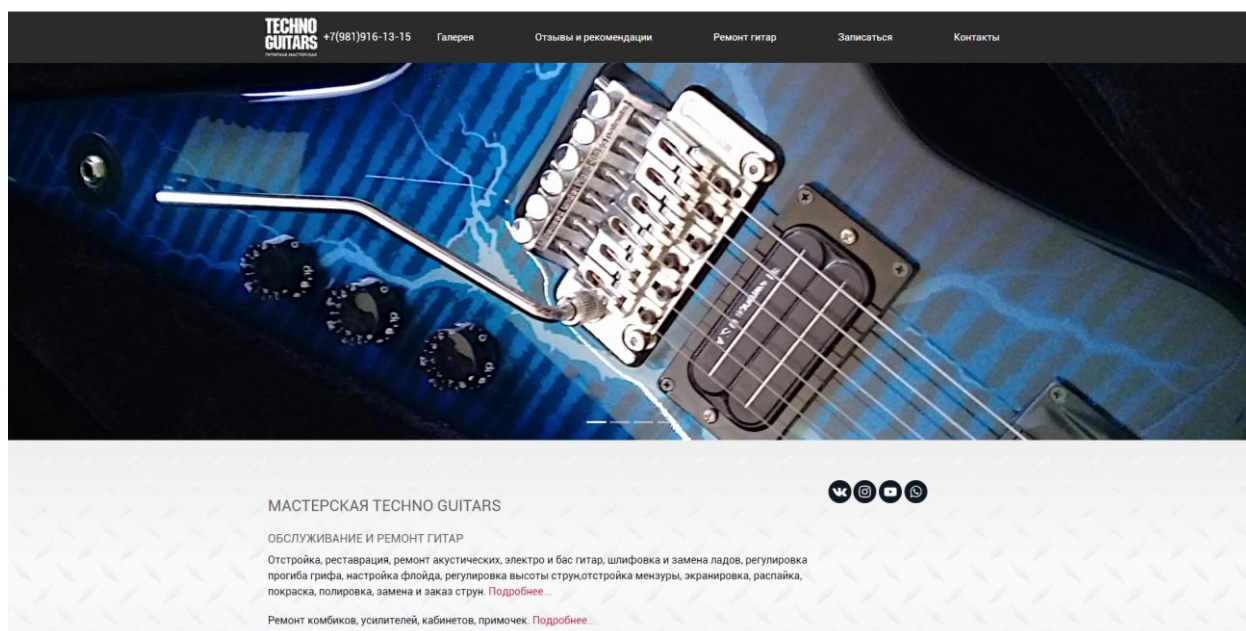


Рисунок 2 – Интерфейс веб-приложения Techno Guitars

Главным достоинством приложения является отсутствие рекламы, что делает интерфейс менее нагруженным и более приятным глазу.

Главным же недостатком, как и в примере выше, является узкая направленность на ремонт только гитар. Так же приложение очень долго открывается, что свидетельствует о плохо написанном коде.

Сервисный центр «Доктор Саунд» представляет собой место, где можно провести обслуживание звукового и светового оборудования или ремонтные работы.

Преимущества:

- широкая направленность на ремонт инструментов;
- низкие цены.

Недостатки:

- неудобный интерфейс;
- служба поддержки.

На рисунке 3 представлен интерфейс данного ПО.

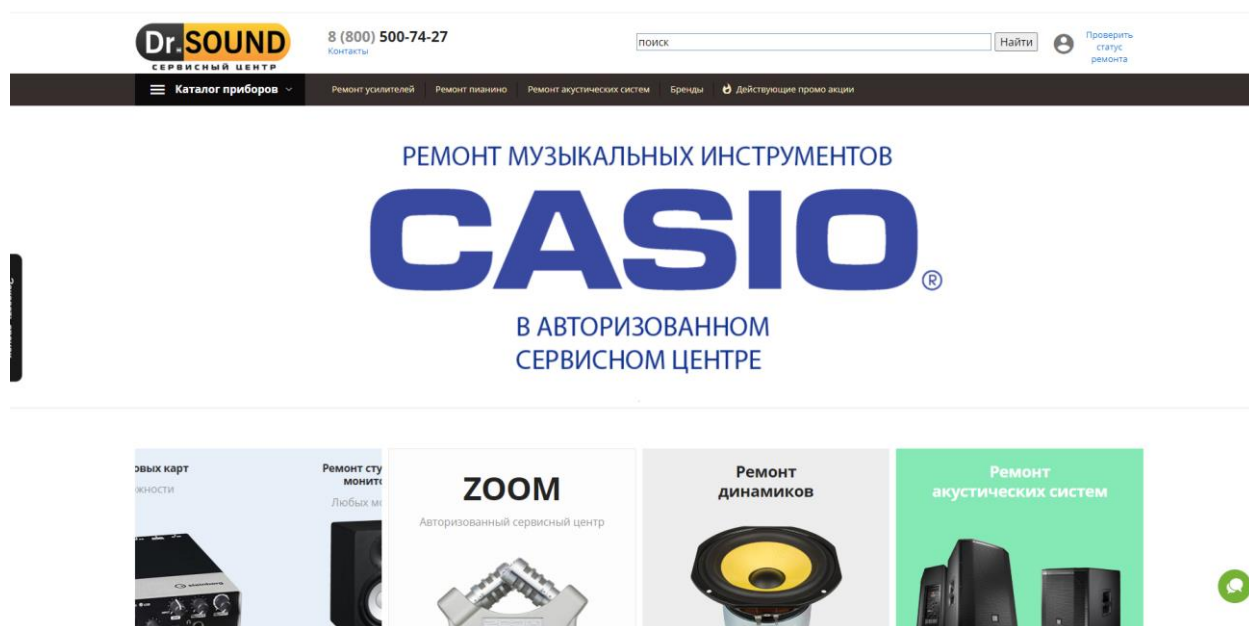


Рисунок 3 – Интерфейс веб-приложения Dr. Sound

Основным достоинством веб-приложения является его универсальность. Компания предоставляет услуги по ремонту многих инструментов.

Основным недостатком является интерфейс. Он выглядит устаревшим и нуждается в обновлении. Так же следует выделить службу поддержки, которая является слишком навязчивой. При загрузке страницы сразу всплывает модальное окно с предложением помочь, что является неудачным решением.

1.3 Сравнение аналогов

Для представленных выше аналогов был проведен сравнительный анализ.

Основными критериями сравнения являются:

- интерфейс;
- поддержка;
- кроссплатформенность;
- широкая направленность;
- реклама.

Результаты приведены в таблице 1.

Таблица 1– Сравнение аналогов

Веб-приложение	Удобство интерфейса	Кроссплатформенность	Широкая направленность	Реклама	Поддержка
Гитарная лаборатория	Удобный	Да	Нет	Отсутствует	Есть
Techno Guitars	Неудобный	Да	Нет	Отсутствует	Есть
Dr. Sound	Неудобный	Да	Да	Присутствует	Есть
Разрабатываемое приложение	Удобный	Да	Да	Отсутствует	Есть

В результате анализа аналогов web-приложения музыкальной мастерской была получена более точная информация о возможных требованиях и потребностях пользователей, а также выявлены основные недостатки приложений, такие как неудобный интерфейс, направленность только на одном типе инструмента. 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 иметь 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 возможность просматривать все заявки пользователей и менять их статус «выполнена»/ «не выполнена».

У обычного пользователя возможность такая возможность должна отсутствовать. Пользователь должен иметь возможность заполнять новую заявку и отправлять ее на сервер.

Приложение должно включать в себя главную страницу. На главной странице будет отображаться шапка сайта, список музыкальных инструментов, по которым осуществляется ремонт.

Также в приложении будет содержаться страница с заявками. К ней доступ будет иметь только админ.

Проанализировав особенности, можно перейти к определению вариантов использования автоматизированной информационной системы, обслуживающей предполагаемых пользователей.

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

- регистрация нового пользователя;
- аутентификация и авторизация пользователя;
- регистрация новой заявки.

У пользователя с правами администратора появляются следующие варианты использования:

- просмотр заявок;
- изменение статуса заявки.

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 логин и пароль, после чего нажать кнопку регистрации. После этого пользователь попадает на главную страницу сайта под своей учетной записью, а в базе данных сохраняется логин нового пользователя и зашифрованный пароль.

Аутентификация и авторизация пользователя. На странице авторизации пользователь должен ввести данные своего аккаунта, а именно логин 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.

Регистрация новой заявки. Пользователь, предварительно войдя в систему, может нажать на кнопку «оставить заявку», после чего попадет на страницу регистрации заявки. На данной странице пользователь заполняет поля инструмент, номер телефона, поломка, после чего нажимает кнопку отправить заявку, и та сохраняется в базе.

Просмотр заявок. Пользователь, войдя под статусом админа, может перейти в админпанель, где отображаются все заявки, находящиеся в базе.

Изменение статуса заявки. Администратор, перейдя в админпанель и увидев списки заявок, может изменить ее статус, нажав на определенную кнопку.

Составим UML-диаграмму [6] вариантов использования для обычного пользователя и для администратора (рисунки 4, 5).

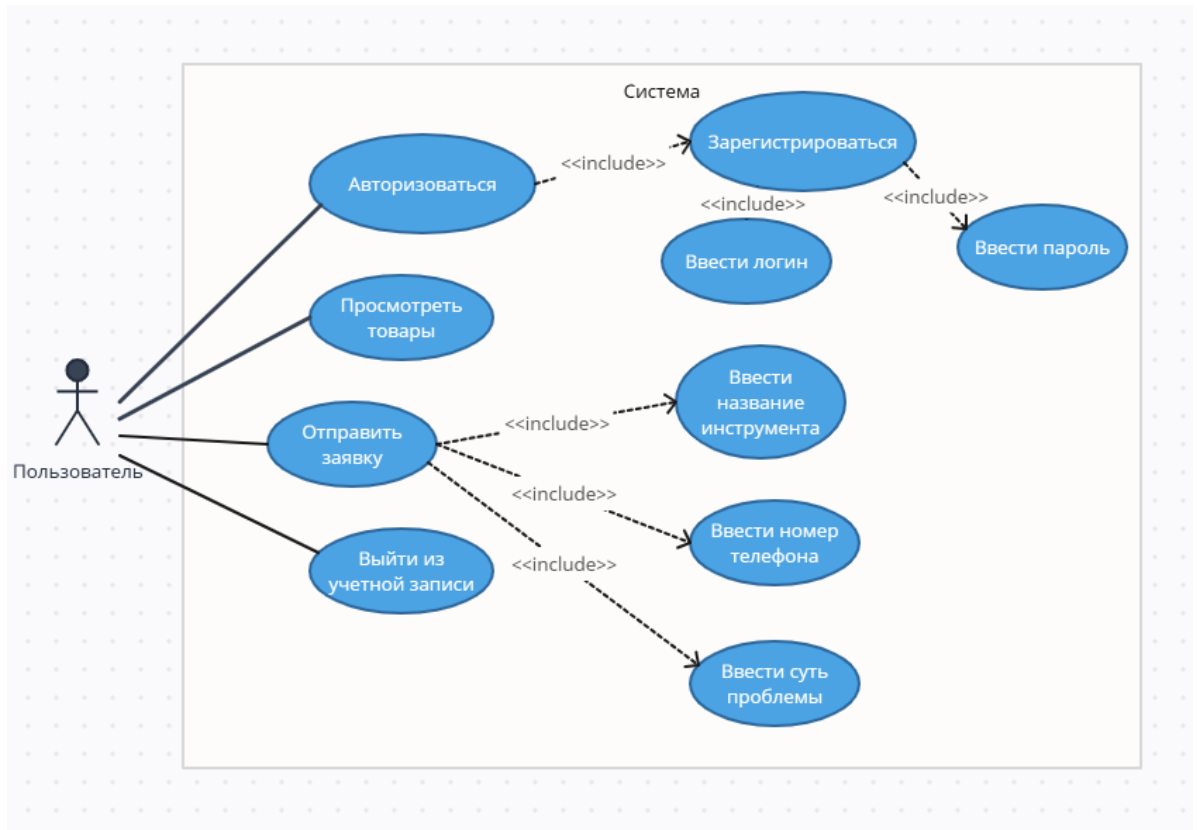


Рисунок 4 – Диаграмма вариантов использования пользователя

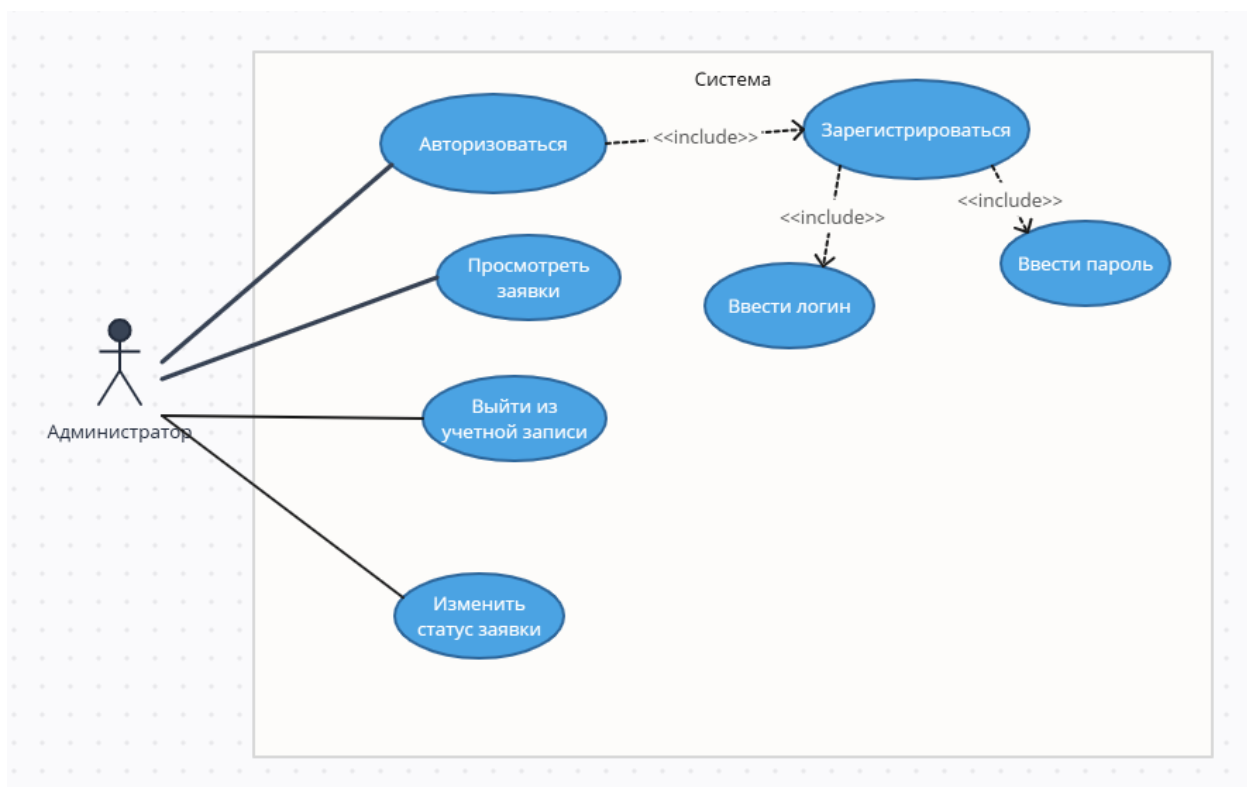


Рисунок 5 – Диаграмма вариантов использования администратора

Далее приводятся примеры спецификаций для некоторых прецедентов.

На рисунке 6 описывается прецедент авторизации клиента.

Прецедент: Авторизация клиента
ID:1
Краткое описание: Клиент авторизируется на ресурсе
Главные актеры: Пользователь.
Второстепенные актеры: Нет
Предусловия: 1. Пользователь зашел на сайт.
Основной поток: 1. Пользователь нажимает на кнопку авторизоваться 2. Если пользователь зарегистрирован, то 2.1 Пользователь вводит свои данные 2.2 Пользователь нажимает на кнопку войти 3. Иначе 3.1 Система перенаправляет пользователя на страницу регистрации. 3.2 Пользователь вводит свои данные. 3.3 Пользователь нажимает на кнопку зарегистрироваться
Постусловия: 1. Пользователь вошел в систему
Альтернативные потоки: Нет

Рисунок 6 – Спецификация прецедента «Авторизоваться»

На рисунке 7 описывается прецедент отправить заявку.

Прецедент: Отправить заявку
ID:2
<p>Краткое описание:</p> <p>Пользователь отправляет заявку на ремонт инструмента</p>
<p>Главные актеры:</p> <p>Пользователь.</p>
<p>Второстепенные актеры:</p> <p>Нет</p>
<p>Предусловия:</p> <p>1. Пользователь вошел в систему.</p>
<p>Основной поток:</p> <ol style="list-style-type: none"> 1. Пользователь нажимает на кнопку оставить заявку 2. Система перенаправляет пользователя на страницу заполнения заявки 3. Пользователь заполняет поля заявки 4. Пользователь нажимает на кнопку отправить заявку
<p>Постусловия:</p> <ol style="list-style-type: none"> 1. Заявка сохранена в БД
<p>Альтернативные потоки:</p> <p>Нет.</p>

Рисунок 7 – Спецификация прецедента «Отправить заявку»

На рисунке 8 описывается прецедент просмотр заявок.

Прецедент: Просмотр заявок
ID:3
Краткое описание: Администратор просматривает заявки
Главные актеры: Администратор
Второстепенные актеры: Нет
Предусловия: 1.Администратор вошел в систему
Основной поток: 1. Администратор нажимает на кнопку админпанель. 2. Система перенаправляет администратора на страницу админпанель. 3. Система выдает администратору список заявок
Постусловия: 1. Администратор видит заявки
Альтернативные потоки: Нет.

Рисунок 8 – Спецификация прецедента «Просмотреть заявки»

Для реализации приложения за основу были взяты представленные варианты использования и актеры.

В результате анализа функциональных требований были успешно выявлены основные функции разрабатываемого программного продукта. Для разрабатываемого программного продукта была построена диаграмма вариантов

использования, а также для некоторых прецедентов были составлены спецификации.

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. В центре должен быть расположен блок с инструментами.

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 музыкальной мастерской

2.1 Проектирование пользовательского интерфейса

Составим общую структуру сайта, схема которой представлена на рисунке

9.

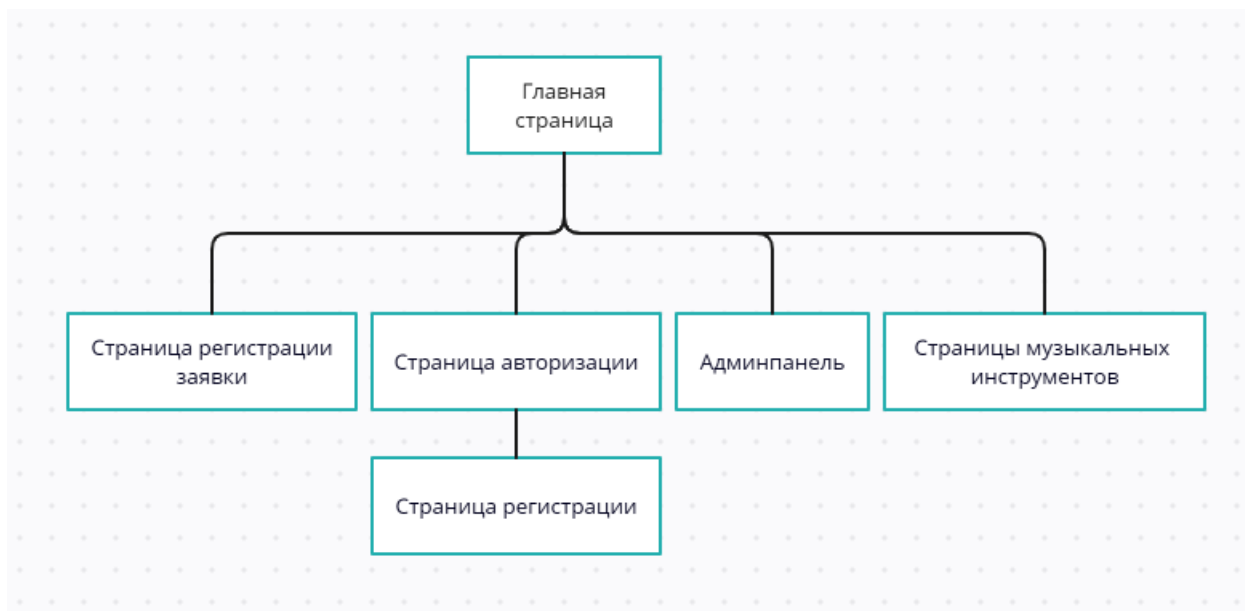


Рисунок 9 – Общая структура сайта

Смоделировав структуру сайта, можно перейти к проектированию пользовательского интерфейса.

Была создана базовая обертка над нужными компонентами для взаимодействия с пользователем. Схема архитектуры ПО представлена на рисунке 10.

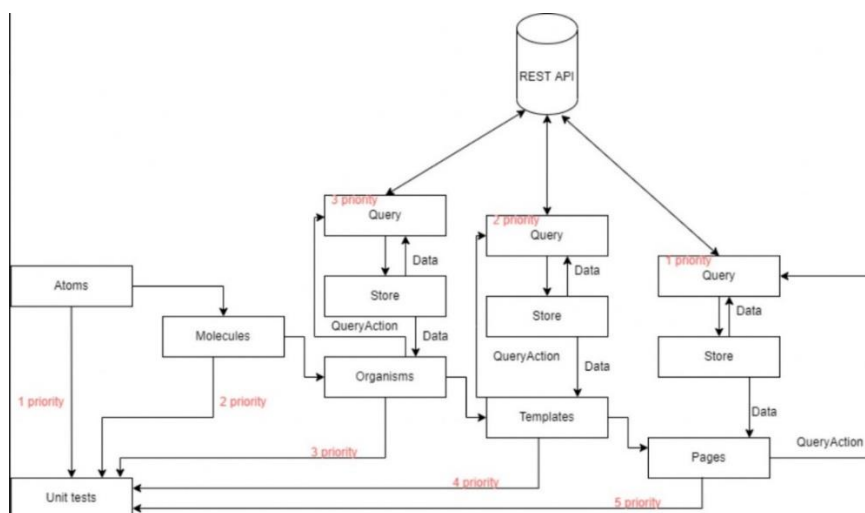


Рисунок 10 – Схема архитектуры пользовательского ПО

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.

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

Главная страница должна содержать карточки музыкальных инструментов, по которым осуществляется ремонт.

Карточки должны быть однотипными. Основное различие будет в названии и в изображениях инструмента.

Интерфейс должен иметь спокойные, сочетающиеся цвета.

Макет главной страницы представлен на рисунке 11.

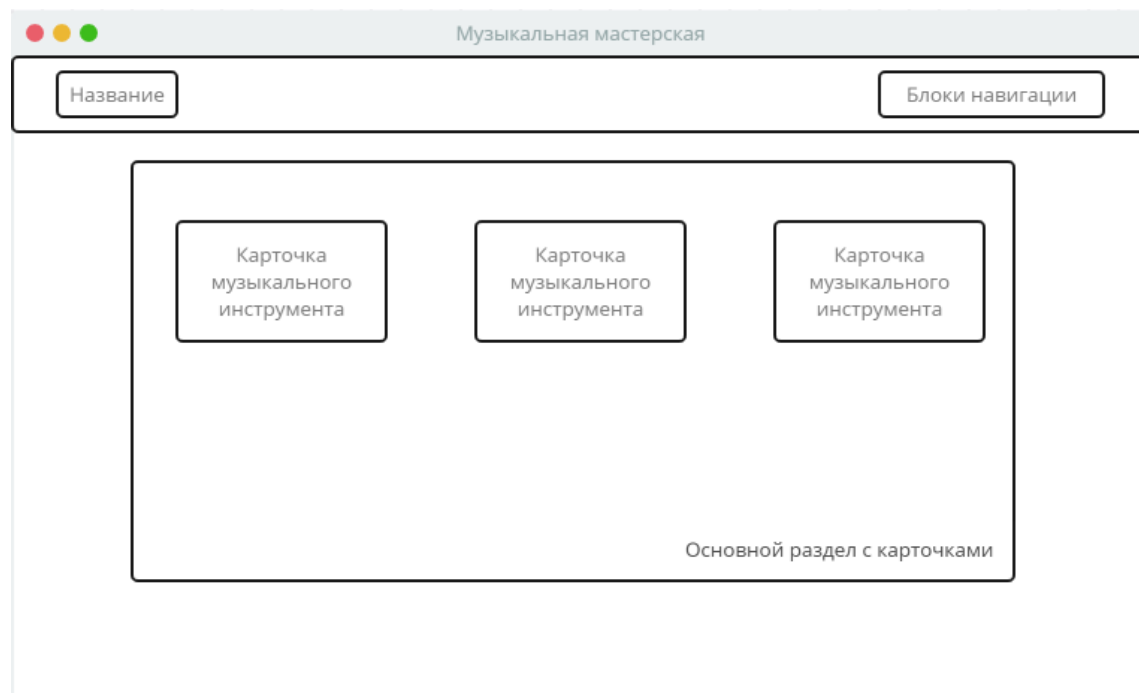


Рисунок 11 – Макет главной страницы

Страницы регистрации и авторизации должны быть максимально простыми и интуитивно понятными. Не должно быть слишком много полей, перегружающих интерфейс и отпугивающих пользователя. Между страницей регистрации и авторизации должны быть переходы, то есть они должны ссылаться друг на друга. Должен осуществляться редирект со страницы авторизации на страницу регистрации и наоборот.

При соблюдении этих правил получим оптимальные страницы регистрации и авторизации.

Страницы регистрации и авторизации должна содержать следующие элементы:

- поле ввода логина;
- поле ввода пароля;
- кнопка перехода на страницу регистрации/авторизации;
- кнопка входа.

Макет страницы авторизации представлен на рисунке 12.

The wireframe shows a window titled "Музыкальная мастерская" (Musical Workshop). It features a header bar with two elements: "Название" (Name) on the left and "Блоки навигации" (Navigation blocks) on the right. The main content area is centered and contains a box labeled "Окно авторизации" (Authorization window). Inside this box, there are four elements arranged vertically: "Поле ввода логина" (Login input field), "Поле ввода пароля" (Password input field), "Регистрация" (Registration button), and "Кнопка войти" (Login button).

Рисунок 12 – Макет страницы авторизации

Страница отдельного музыкального инструмента должна отображать основную информацию о музыкальном инструменте, его картинку, а также кнопку создания заявки на ремонт этого инструмента.

Она также должна быть простой и интуитивно понятной, не должно быть излишней загруженности и наполненности ненужными элементами. Блок с характеристиками инструмента должен быть лаконичен, и в то же время в нем должна быть полная информация о инструменте.

Вся страница должна иметь спокойные и не броские цвета, сочетающиеся между собой.

При соблюдении всех условий, мы получим простую, но в то же время функциональную страницу, отвечающую всем требованиям пользователя.

Макет страницы музыкального инструмента представлен на рисунке 13.

The wireframe shows a web page layout for a musical instrument. At the top is a header bar with the title 'Музыкальная мастерская' in the center. Below the header, there are two main sections. The left section contains a large rectangular box labeled 'Изображение' (Image). To the right of this box is a smaller rectangular button labeled 'Кнопка создать заявку' (Create request button). Below the image box is a wide rectangular box labeled 'Характеристика инструмента' (Instrument characteristics). At the top of the main content area, there are two smaller boxes: 'Название' (Name) on the left and 'Блоки навигации' (Navigation blocks) on the right.

Рисунок 13 – Макет страницы музыкального инструмента

Страница заявки должна содержать следующие элементы:

- поле ввода инструмента;
- поле ввода номера телефона;
- поле ввода описания проблемы;
- кнопка отправить заявку.

Страница заявки также не должна быть перегружена. Пользователь должен вводить основные поля, предоставляя основную информацию и не нагружая систему лишней или не нужной информацией.

При соблюдении всех условий, получится компактная и функционально эффективная форма, не требующая от клиента чрезмерных усилий для заполнения.

Макет страницы заявки на ремонт представлен на рисунке 14.

The wireframe shows a web browser window titled 'Музыкальная мастерская'. The header bar contains two elements: a text input field labeled 'Название' and a button labeled 'Блоки навигации'. The main content area features a central box titled 'Форма заявки'. Inside this box, there are three stacked text input fields labeled 'Поле ввода инструмента', 'Поле ввода номера телефона', and 'Поле ввода описания проблемы'. At the bottom of the box is a button labeled 'Кнопка отправить заявку'.

Рисунок 14 – Макет страницы заявки на ремонт

Таким образом, проанализировав все требования, был составлен наиболее оптимальный интерфейс для разрабатываемого приложения.

2.2 Выбор программно-аппаратных средств

Среди всех возможных вариантов программного обеспечения для музыкальной мастерской 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].

2.3 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 15.

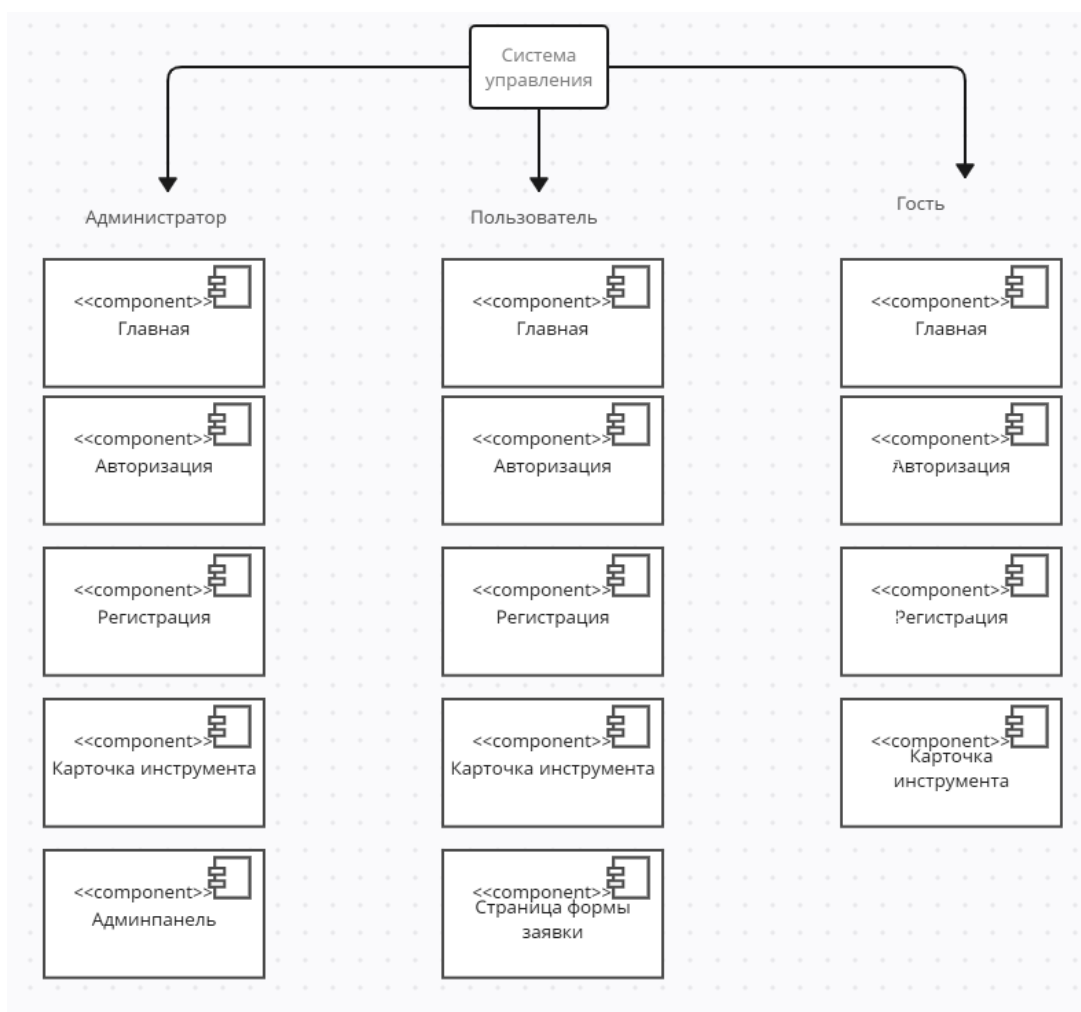


Рисунок 15 – Диаграмма компонентов

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 быстрое действие 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.

2.4 Структуры хранения 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 16.

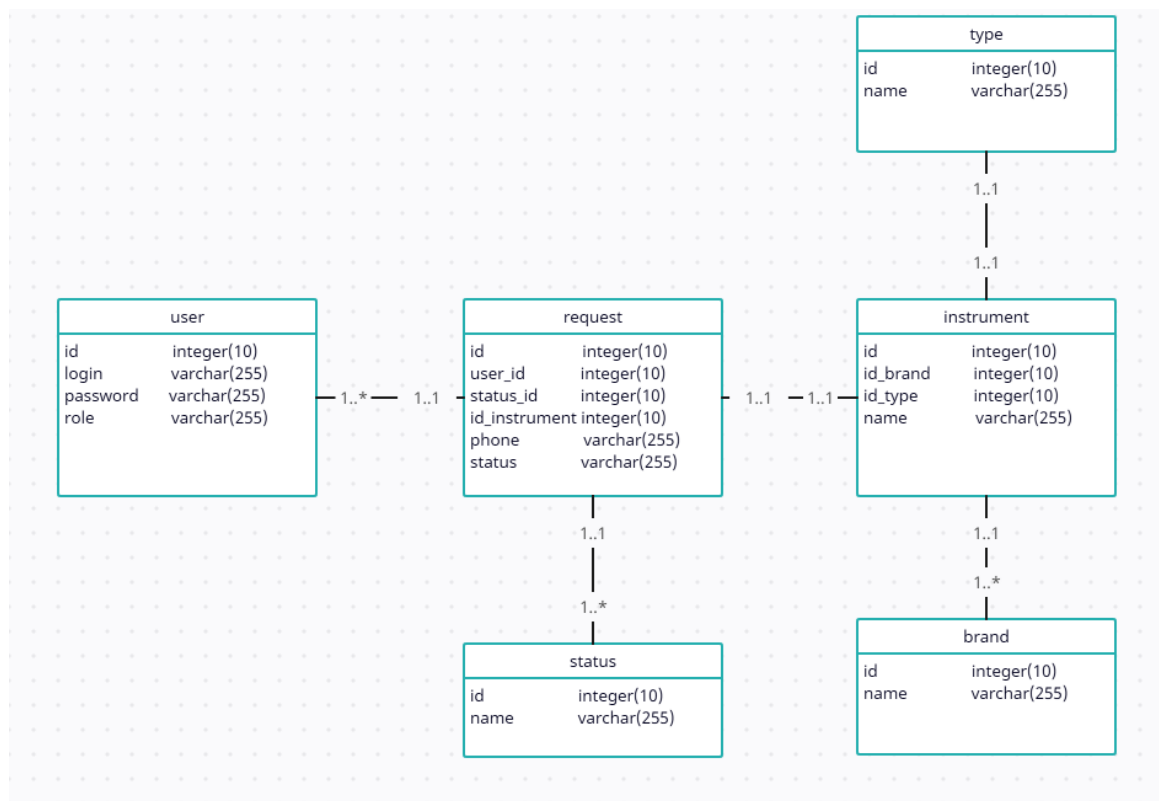


Рисунок 16 – Диаграмма классов базы данных

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 pgAdmin 4. База данных музыкальной мастерской состоит из 6 связанных классов. Каждый класс логически связан с другими.

При исключении одного класса, вся система перестанет функционировать и выдаст ошибку.

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
Заявки	requests	Содержит сведения о заявках
Статус	status	Содержит о статусе заявки
Тип	type	Содержит названия типов инструмента
Инструмент	instrument	Содержит сведения о инструменте
Бренды	brand	Содержит названия брендов инструмента

Определим связи таблиц исходя из логики. Таблицы «users» и «requests» связаны отношением «один-ко-многим». Таблицы «status» и «requests» связаны отношением «один-ко-многим». Таблицы «instruments» и «request» связаны отношением «один-к-одному». Таблицы «type» и «instrument» связаны отношением «один-к-одному». Таблицы «brand» и «instruments» связаны отношением «один-ко-многим».

В таблицах ниже приведены характеристики и ограничения полей таблиц базы данных.

Описание «users» описано в таблице 3

Таблица 3 – Таблица «users» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	12
login	логин	varchar(50)	нет		alex
password	Пароль	varchar(32)	нет		Qwerty123

role	Роль	varchar(10)	нет		USER
------	------	-------------	-----	--	------

Описание «requests» описано в таблице 4

Таблица 4 – Таблица «requests» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	16
User_id	Id пользователя	integer(10)	нет	FK	4
Status_id	Id статуса	integer(10)	нет	FK	1
Id_instrument	Id инструмента	integer(10)	да	FK	5
phone	Телефон	varchar(255)	нет		89567843766
status	Статус	varchar(255)	нет		Выполнена

Описание «instrument» описано в таблице 5

Таблица 5 – Таблица «instrument» базы данных

Идентификатор	Описание	Тип данных, длина	Возможность значения Null	Роль	Пример
id	Идентификатор	integer(10)	нет	PK	7
Id_brand	Id бренда	integer(10)	нет	FK	5
Id_type	Id типа	integer(10)	нет	FK	3
name	имя	varchar(255)	нет		Herington

Описание «status» описано в таблице 6

Таблица 6 – Таблица «status» 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		Выполнена

Описание «type» описано в таблице 7

Таблица 7 – Таблица «type» 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		Гитара

Описание «brand» описано в таблице 8

Таблица 8 – Таблица «brand» 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

Связь программного кода и реляционной базы данных будет осуществляться с помощью ORM-библиотеки Sequelize. С её помощью осуществляются различные изменения данных в БД благодаря встроенным JavaScript функций.

3 Реализация web-приложения «Музыкальная мастерская»

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, описанную ниже.

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

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

middleware – папка с классами DTO (Data Transfer Object) для передачи и приема данных клиенту и от клиента;

models – папка содержащий файл, описывающий модель базы данных;

routes – папка с файлами, описывающими маршрутизацию;

static – папка с моделями (сущностями), которые отображаются в базе данных;

.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:

AuthController.cs – класс контроллера для запросов, связанных с аутентификацией;

MessagesController.cs – класс контроллера для запросов, связанных с системой личных сообщений;

PhotosController.cs – класс контроллера для запросов, связанных с фотографиями;

UserController.cs – класс контроллера для запросов, связанных с действиями с профилями пользователей;

TasksControllers.cs – класс контроллера для запросов, связанных с действиями с задачами и спринтами;

Data – классы БД:

AuthRepository.cs – класс для добавления и получения данных об аутентификации пользователей, включающий методы хеширования паролей;

DataContext.cs – главный класс, определяющий сущности и их свойства, которые переносятся в базу данных при запуске процесса миграции;

IAuthRepository.cs – интерфейс для класса аутентификации;

ITfxRepository.cs – главный интерфейс, определяющий методы доступа к данным в БД;

Seed.cs – класс для создания и заполнения базы данных при ее отсутствии (используется только при разработке);

TfxRepository.cs – класс, содержащий методы доступа к записям в базе данных;

UserSeedData.json – файл, данные из которого использует класс Seed для заполнения новой БД;

Dtos – классы для передачи данных:

MessageForCreationDto.cs, MessageToReturnDto.cs – классы для передачи и приема данных о личных сообщениях;

PhotoForCreationDto.cs, PhotoForReturnDto.cs, PhotosForDetailedDto.cs – классы для передачи и приема данных об изображениях;

UserForDetailedDto.cs, UserForListDto.cs, UserForLoginDto.cs, UserForRegisterDto.cs, UserForUpdateDto.cs – классы для передачи и приема данных о пользователях;

TaskForDetailedDto.cs, TaskForListDto.cs, TaskForCreationDto.cs, TaskForUpdateDto.cs – классы для передачи и приема данных о задачах;

SprintForCreationDto.cs, SprintForDetailedDto.cs – класс для создания и просмотра спринтов;

Helpers – вспомогательные классы:

AutoMapperProfiles.cs – класс для добавления новых возможностей для класса AutoMapper, позволяющего автоматически сопоставлять объекты разных классов;

CloudinarySettings.cs – класс для настройки облачного сервиса Cloudinary для хранения файлов, загружаемых в систему;

Extensions.cs – класс, содержащий методы расширения;

LogUserActivity.cs – класс, регистрирующий обращения пользователей к API и обновляющий их время последней активности;

MessageParams.cs – класс с параметрами для получения страницы сообщений;

PagedList.cs – класс, позволяющий получать страницы данных вместо всех сохраненных данных;

PaginationHeader.cs – класс, добавляющий в ответ на запрос заголовки с информацией о полученной странице, если данные получаются не целиком, а отдельными страницами;

UserParams.cs – класс с параметрами для получения страницы пользователей;

TaskParams.cs – класс с параметрами для получения страницы задач;

SprintParams.cs – класс с параметрами для получения страницы спринтов;

Models – отображение данных из базы данных;

Like.cs – информация о пользователях в избранном;

Message.cs – информация о личных сообщениях;

Photo.cs – информация о сохраненных в облаке изображениях;

User.cs – информация о пользователях;

Клиентское приложение является SPA – Single Page Application (одностраничное приложение), реализованное с помощью фреймворка Angular [29]. Особенность Angular заключается в том, что приложение состоит из изолированных компонентов, которые могут взаимодействовать друг с другом с помощью специальных конструкций. Каждый компонент состоит из

трех файлов: шаблон, класс и таблица стилей. Шаблон является HTML-страницей, которая определяет структуру компонента. Класс является файлом с кодом на языке TypeScript, который определяет поведение компонента. Таблица стилей же является CSS-файлом, который применяется только внутри компонента, к которому относится.

Рассмотрим компоненты разрабатываемого приложения:

`home.component` – главная страница приложения;

`lists.component` – страница пользователей, добавленных в избранное;

`member-card.component` – компонент карточки пользователя для отображения в списках;

`member-detail.component` – компонент страницы профиля пользователей;

`member-edit.component` – компонент для редактирования профиля пользователя;

`member-list.component` – компонент для отображения всех пользователей в системе;

`member-messages.component` – компонент для отправки и получения личных сообщений;

`photo-editor.component` – компонент для загрузки и удаления изображений;

`messages.component` – компонент, отображающий список полученных и отправленных сообщений;

`nav.component` – компонент навигации, располагающийся в верхней части приложения на любом экране;

`register.component` – компонент для создания нового пользователя;

`task-detail.component` – компонент для создания, просмотра и редактирования задач;

`task-list.component` – компонент, содержащий элементы для отображения списка задач;

`task-comments.component` – компонент, содержащий комментарии к задаче и позволяющий добавлять новые комментарии;

`sprint-detail.component` – компонент для создания и просмотра спринтов;

`sprint-list.component` – компонент, содержащий элементы для отображения списка спринтов.

Помимо компонентов в папке с проектом SPA присутствуют следующие файлы:

`auth.guard.ts` – запрещает навигацию неавторизованным пользователям;

модели: `messages.ts`, `paginations.ts`, `photo.ts`, `task.ts`, `task-comment.ts`, `user.ts`, `sprint.ts` – отображают данные, хранящиеся в базе данных;

резолверы: `list.resolver.ts`, `member-detail.resolver.ts`, `member-edit.resolver.ts`, `member-list.resolver.ts`, `messages.resolver.ts`, `task-list.resolver.ts`, `task.resolver.ts`, `sprint.resolver.ts`, `sprint-list.resolver.ts` – классы, используемые маршрутизатором Angular для вызова компонентов при переходе на определенные ссылки;

`alertify.service.ts` – сервис, дающий удобный доступ к подключаемому компоненту Alertify для отображения уведомлений на странице;

`auth.service.ts` – сервис, содержащий методы, связанные с аутентификацией;

`user.service.ts` – сервис, содержащий методы для работы с данными о пользователях;

`task.service.ts` – сервис, содержащий методы для работы с данными о задачах;

`sprint.service.ts` – сервис, содержащий методы для работы с данными о спринтах;

`photo.component` – компонент для загрузки и удаления изображений;

`requests.component` – компонент, отображающий список полученных и отправленных заявок.

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.

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;
- добавление новой заявки;
- просмотр заявок;
- изменение статуса заявки.

На основе приведенных выше сценариев был составлен план тестирования и проведено функциональное тестирование. Результат тестирования приведен в таблице 8.

Таблица 8 – 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
3	Регистрация	При вводе корректных данных пользователь успешно регистрируется и авторизуется	Новый пользователь создан и авторизован в системе

Продолжение таблицы 8

№	Названия тестового случая	Тест	Результат
4	Авторизация	При вводе существующих в базе и корректных данных пользователь успешно авторизуется	Пользователь успешно авторизуется в системе
5	Добавление новой заявки	При вводе корректных данных, создается новая заявка	Заявка создана и добавлена в базу данных
6	Просмотр заявок	При нажатии на кнопку админпанель, система предоставит администратору список всех заявок	Список заявок успешно отображен
7	Изменение статуса заявки	Администратор меняет статус заявки	Статус изменен, изменения сохранены в системе

Для авторизации было проведено тестирование методом выделения классов эквивалентности. Результаты тестирования приведены в таблице 9.

Из тестирования видно, что приложение работает корректно и возвращает ожидаемый результат.

Таблица 9 – Результаты тестирования

№	Входное условие	Пример	Правильность
1	Корректный логин	Alexy1978	+
2	Логин короче 6 символов	zzz	-
3	Пустая строка в поле для ввода логина		-

Продолжение таблицы 9

№	Входное условие	Пример	Правильность
4	Корректный пароль	Test1234	+
5	Пароль короче 8 символов	ww	-
6	Пароль длиннее 32 символов	nvhfYurhFJu57svcbxsA574636373728Trsdsddsds saDFDAadsadsadssadsadaassasasadsdsasdasdasda sdasdasdasdas	-
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 заявками на ремонт.

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

- создание заявки;
- создание бренда музыкального инструмента;
- получение музыкального инструмента;
- получение заявки;
- получение бренда музыкального инструмента;
- получение всех заявок.

Используя метод покрытия условий, был составлен список тестов для вариантов использования этого модуля. Список тестов представлен в таблице 10.

Таблица 10 – Тесты модуля работы с музыкальными инструментами

№	Метод класса	Тест	Результат	Тест пройден
1	createRequest	Создание заявки	Заявка успешно создана	+
2	fetchRequests	Получение заявки	Возвращена заявка	+
3	createBrand	Создание бренда инструмента	Бренд успешно создан	+
4	fetchBrands	Получение брендов инструментов	Возвращены бренды инструментов	+
5	createInstrument	Создание музыкального инструмента	Музыкальный инструмент успешно создан	+
6	fetchInstruments	Получение музыкальных инструментов	Возвращены музыкальные инструменты	+
7	fetchAllRequests	Получение всех заявок	Возвращены все заявки	+

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

Мера представляет собой числовой показатель, который отражает степень, количество или размер определенных атрибутов продукта или процесса. Например, количество ошибок может быть мерой.

Метрика программного кода является количественной мерой, которая позволяет оценить, насколько система, компоненты или процесс обладают определенным атрибутом. Примером метрики может быть количество обнаруженных ошибок в отношении человек/час, затраченных на разработку.

Метрики программного кода могут быть разделены на следующие группы:

- Метрики размера программы (например, метрика строк исходного кода – SLOC);
- Метрики сложности управления потоком программы (например, метрики Холстеда, цикломатическая сложность, метрики Чепина).

Давайте рассчитаем соответствующие метрики кода для варианта использования «Регистрация пользователя».

4.3.1 Метрика SLOC

Мерой называется количественный показатель, который позволяет оценить степень, количество или размеры кода. SLOC-метрика используется для подсчета физических и логических строк кода. Физические строки включают все непустые строки в исходном файле программы, а логические строки кода отражают количество команд, используемых в программе. Кроме того, определение команды зависит от конкретного языка программирования. [9]

На рисунке 17 приведен код для регистрации пользователя.

```
async registration(req, res, next) {
  const {login, password, role} = req.body
  if (!login || !password) {
    return next(ApiError.badRequest('Некорректный login или password'))
  }
  const candidate = await User.findOne({where: {login}})
  if (candidate) {
    return next(ApiError.badRequest('Пользователь с таким login уже существует'))
  }
  const hashPassword = await bcrypt.hash(password, 5)
  const user = await User.create({login, password: hashPassword, role})
  const token = generateJwt(user.id, user.login, user.role)
  return res.json({token})
}
```

Рисунок 17 – код регистрации пользователя

Применим SLOC-метрику к нашему программному коду. Результаты данной метрики представлены в таблице 11.

Таблица 11 – SLOC-метрики

Число физических строк	15
Число логических строк	6
Число пустых строк	3
Число строк-комментариев	0

4.3.2 Метрики Холстеда

Метрики Холстеда измеряются следующими характеристиками программы:

- количество уникальных операторов, включая символы-разделители, имена процедур и знаки операций.
- количество уникальных операндов.
- общее количество операторов в программе.
- общее количество операндов в программе.

На основе этих метрик можно получить следующие оценки:

- словарь программы: сумма количества уникальных операторов и уникальных операндов.
- длина программы: сумма общего количества операторов и операндов.
- объем программы: произведение длины программы на логарифм по основанию 2 от словаря программы.
- сложность программы:
- показатель усилий программиста: произведение сложности программы на объем программы.

Применим метрики Холстеда к нашему программному коду. Результаты представлены в таблице 12.

Таблица 12 – метрики Холстеда

Число операторов (уникальных)	Число операндов (уникальных)
4 (2)	28 (17)

Получим следующие оценки:

- Словарь программы = $2 + 17 = 19$;
- Длина программы = $4 + 28 = 32$;
- Объем программы = $32 \log_2 21 = 162,5$;
- Сложность программы = $(2/1) * (28/17) = 2,6$;
- Показатель усилий программиста = 389,6

4.3.3 Комплексный показатель качества кода

Рассмотрим подсчет данной метрики на примере метода получения всех заявок. Данный метод находится в классе `getAllRequests`.

$HV = 8$.

$CC = 4$.

$LoC = 7$.

$MI = 93,14$.

Данное значение показывает, что метод легко поддерживать.

5 Расчет бюджета

5.1 Планирование и оценка бюджета проекта

Для выполнения поставленных задач проекта и достижения цели необходимо учитывать время, бюджет, ресурсы, а также все риски данного проекта. Именно контроль этих факторов называется управлением проектом.

Управление проектом – деятельность по решению задач и достижению поставленных целей проект. Управление проектом характеризуется как организация процесса по успешному достижению поставленной цели (за отведенное время, выделенный бюджет, имеющиеся в наличии ресурсы) и нахождение путей развития проекта в условиях кризиса или непредвиденных ситуаций. При осуществлении управления нужно учитывать правило треугольника проекта, которое характеризует связь между параметрами ВРЕМЯ-КАЧЕСТВО-БЮДЖЕТ и гласит, что изменение любой из этих составляющих влияет на две другие.

При планировании разработки было выделено 22 задачи. Для каждой задачи было выделено время и порядок следования задач. Это является очень важным этапом проектирования. Без правильного планирования и оценки невозможно корректно поставить задачу и реализовать дальнейшее написание проекта.

В таблице 13 представлены наименование, предшественник и длительность каждой из задач.

Таблица 13 – Перечень задач

№	Наименование	Длительность	Предшественник
1	Начало реализации проекта	-	
2	Программирование	9 дней	
3	Разработка серверной части приложения	9 дней	

Продолжение таблицы 13

№	Наименование	Длительность	Предшественник
4	Постановка задачи	1 день	1
5	Разработка структуры базы данных	3 дня	4
6	Заполнение базы данных	2 дня	5
7	Разработка триггеров	3 дня	6
8	Разработка серверной части завершена	-	7
9	Разработка клиентской части приложения	5 дней	
10	Постановка задачи	1 день	1
11	Разработка интерфейса	4 дня	10
12	Разработка клиентской части завершена	-	11
13	Программирование завершено	-	8;12
14	Отладка	5 дней	
15	Тестирование и исправление ошибок	3 дня	13
16	Составление программной документации	2 дня	15
17	Отладка завершена	-	16
18	Документирование	2 дня	
19	Создание установочного файла	1 день	17
20	Написание руководства пользования	1 день	19
21	Документация завершена	-	20
22	Конец проекта	-	21

Планирование разработки и оценка бюджета разрабатываемого проекта были выполнены в программе Microsoft Project 2007. Данная программа обеспечивает разработку планов, распределение ресурсов по задачам, отслеживание прогресса и анализ объёмов работ, а также позволяет генерировать расписания критического пути.

Ниже представлены две части представления диаграммы Ганта: табличная часть, изображенная на рисунке 18, и графическая часть, проиллюстрированная на рисунке 19. Табличное представление диаграммы Ганта содержит подробное описание задач. Графическое представление диаграммы Ганта отображает график выполнения работ по проекту [10].

№	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	Начало реализации проекта	0 дней	Ср 15.03.23	Ср 15.03.23		
2	Программирование	9 дней?	Ср 15.03.23	Пн 27.03.23		Интернет[800,00 Р]
3	Разработка серверной части приложения	9 дней	Ср 15.03.23	Пн 27.03.23		
4	Постановка задачи	1 день	Ср 15.03.23	Ср 15.03.23	1	Ведущий программист; Программист
5	Разработка структуры базы данных	3 дней	Чт 16.03.23	Пн 20.03.23	4	Ведущий программист
6	Заполнение базы данных	2 дней	Вт 21.03.23	Ср 22.03.23	5	Ведущий программист
7	Разработка триггеров	3 дней	Чт 23.03.23	Пн 27.03.23	6	Ведущий программист
8	Разработка серверной части завершена	0 дней	Пн 27.03.23	Пн 27.03.23	7	
9	Разработка клиентской части приложения	5 дней?	Ср 15.03.23	Вт 21.03.23		
10	Постановка задачи	1 день?	Ср 15.03.23	Ср 15.03.23	1	Ведущий программист; Программист
11	Разработка интерфейса	4 дней	Вт 21.03.23	Вт 21.03.23	10	Программист; Ведущий программист[50%]
12	Разработка клиентской части завершена	0 дней	Вт 21.03.23	Вт 21.03.23	11	
13	Программирование завершено	0 дней	Пн 27.03.23	Пн 27.03.23	8;12	
14	Отладка	5 дней	Вт 28.03.23	Пн 03.04.23		Интернет[800,00 Р]
15	Тестирование и исправление ошибок	3 дней	Вт 28.03.23	Чт 30.03.23	13	Тестировщик
16	Составление программной документации	2 дней	Пт 31.03.23	Пн 03.04.23	15	Программист; Тестировщик; Ведущий программист[50%]
17	Отладка завершена	0 дней	Пн 03.04.23	Пн 03.04.23	16	
18	Документирование	2 дней	Вт 04.04.23	Ср 05.04.23		Интернет[800,00 Р]
19	Создание установочного файла	1 день	Вт 04.04.23	Вт 04.04.23	17	Тестировщик
20	Написание руководства пользователя	1 день	Ср 05.04.23	Ср 05.04.23	19	Тестировщик
21	Документация завершена	0 дней	Ср 05.04.23	Ср 05.04.23	20	
22	Конец проекта	0 дней	Ср 05.04.23	Ср 05.04.23	21	

Рисунок 18 – Табличное представление диаграммы Ганта

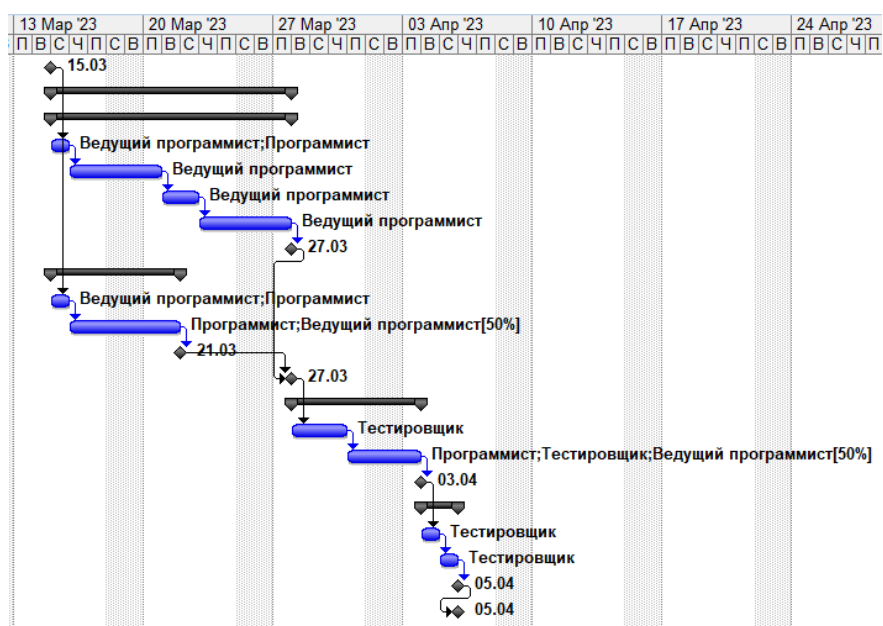


Рисунок 19 – Графическое представление диаграммы Ганта

На рисунке 20 представлен график ресурсов. Синим цветом – отображены нормы рабочего времени, красным – переработки.

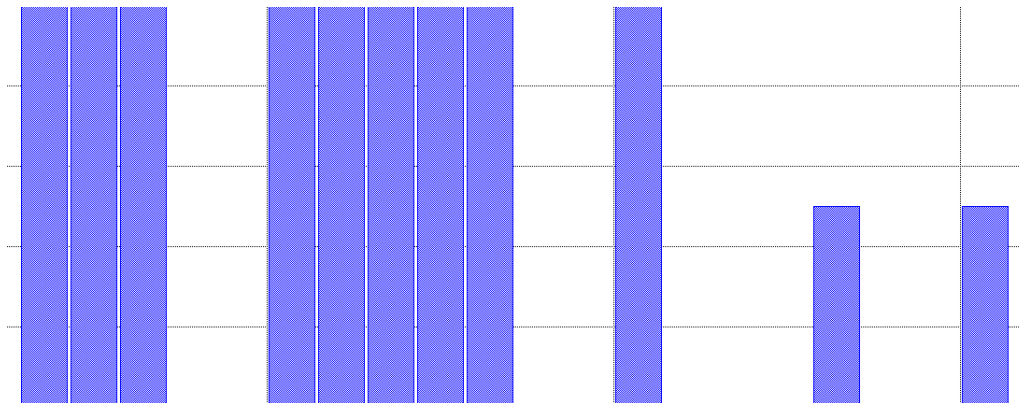


Рисунок 20 – График ресурсов

Исходя из рисунка, можем сделать вывод о том, что переработок нет. Кроме того, в некоторые дни, программист занят только на 50% времени. Соответственно данный план можно считать корректным.

Итоговая стоимость проекта составила: 41025 рублей. В неё входят заработные платы работников, а также траты на электричество и интернет.

Затраты проекта изображены на рисунке 21.

Затраты			
Текущий план:	41 025,00 Р	Оставшиеся:	41 025,00 Р
Базовый план:	0,00 Р	Фактические:	0,00 Р
Отклонение:	41 025,00 Р		

Рисунок 21 – Затраты проекта

В результате планирования разработки и оценки бюджета была построена диаграмма Ганта и посчитана итоговая стоимость проекта.

Заключение

В данном проекте были выполнены поставленные задачи на анализ требований, разработку системы и тестирование продукта.

Был проведен анализ приложений схожего назначения, выявлены критерии, по которым проводилось сравнение.

Разработка приложения выполнялась в VisualStudio code на языке программирования JavaScript с использованием фреймворков React и Node.js.

Также, на основе представленной архитектуры и готового решения можно сделать вывод о том, что приложение имеет потенциал, и при правильной дальнейшей поддержке может стать востребованным в своей области.

В будущем разработанное приложение может быть улучшено и введено в продакшн.

Список использованных источников

1. Профи.ru[Электронный ресурс]. URL:
<https://penza.profi.ru/remont/remont-musicalnih-instrumentov/> (дата обращения: 18.04.2023).
2. GuitarLiabrury.ru[Электронный ресурс]. URL:
<https://guitarlaboratory.ru/> (дата обращения: 26.04.2023).
3. Technoguitars.ru [Электронный ресурс]. URL:
<http://technoguitars.ru/repairs/> (дата обращения: 26.04.2023).
4. DRsound.ru [Электронный ресурс]. URL:
<https://www.drsound.ru/remont-muzykalnogo-oborudovaniya/> (дата обращения: 26.04.2023).
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: 20.04.2023).
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: 17.04.2023).
15. Кэти Сьерра и Берт Бейтс. Изучаем Java. / Эксмо, 720. – 720 с.
16. Дакетт, Джон Javascript и jQuery. Интерактивная веб-разработка / Джон Дакетт. – М.: Эксмо, 2014. – 146 с.

Приложение А
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/ErrorHandlerMiddleware')
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 Б
Листинг кода файла RequestController.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 В
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 Г
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 Д
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 Е
Листинг кода файла RequestAPI.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  
  }  
}
```