МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 «Программное обеспечение информационных технологий»

Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий (программирование интернет-приложений)»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту на тему:**

Web-приложение «Интернет-магазин для заказа акустических гитар»

Выполнил студент Сивак Михаил Николаевич

(Ф.И.О.)

Руководитель проекта преп.-стаж. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-стаж. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-стаж. Дубовик М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2020

Реферат

Пояснительная записка курсового проекта содержит 30 страниц пояснительной записки, 29 иллюстраций, 6 источников литературы, 1 приложение.

nODE, EXPRESS, sequelize, MSSQL, Bootstrap, css, handlebars, GUitars, orders, authorization.

На сегодняшний день всё больше людей перестают пользоваться обычными магазинами и предпочитают приобретать покупки онлайн, так как это более комфортно для человека, выбор товара соответственно намного больше, а также можно найти определённый товар дешевле, что не мало важно.

В соответствии с заданием курсового проекта следует создать веб-приложение архитектуры клиент-сервер с базой данных (БД) для хранения данных о заказах, товарах и пользователей и системой администрирования с помощью кроссплатформенной разработкой через Node.js

В первой главе проводится построение задачи и аналитический обзор литературы по тематике курсового проекта

Вторая глава посвящена процессу проектирования системы и описание технологий, использованных во время выполнения проекта.

В третьей главе описывается процесс разработки, принципы функционирования и назначение созданных компонент проекта.

В четвёртой главе описано руководство пользователя либо тестирование, позволяющее подробно понять интерфейс программного средства.

В пятой главе приведены результаты тестирования проделанной работы.

Оглавление

[Реферат.....................................................................................................................2](#_Toc41866027)

[Введение...................................................................................................................4](#_Toc41866028)

[1 Постановка задачи и обзор аналогов 5](#_Toc41866029)

[1.1 Постановка задачи 5](#_Toc41866030)

[1.2 Обзор аналогов 6](#_Toc41866031)

[2 Проектирование 9](#_Toc41866032)

[2.1 Проектирование серверной и клиентской части 9](#_Toc41866033)

[2.2 Выбор СУБД (системы управления базами данных) 12](#_Toc41866034)

[2.3 Диаграммы UML 15](#_Toc41866035)

[2.4. Структура моделей и классов 16](#_Toc41866036)

[3 Разработка функциональной модели и модели данных ПС 17](#_Toc41866037)

[3.1 Выполняемые функции 17](#_Toc41866038)

[3.2 Модель базы данных 19](#_Toc41866039)

[4 Руководство пользователя 21](#_Toc41866040)

[5 Тестирование 24](#_Toc41866041)

[Заключение 27](#_Toc41866042)

[Список литературы 28](#_Toc41866043)

[Приложение A 29](#_Toc41866044)

# Введение

В данной записке приведено описание веб-приложения и базы данных, разработанных в соответствии с заданием на курсовое проектирование по теме «Интернет-магазин для заказа акустических гитар» по дисциплине «Программирование серверных кроссплатформенных приложений».

Разработанное приложение помогает людям не выходя из дома выбрать акустическую гитару, того производителя, которого они захотят по доступной цене, а не идти в магазин гитар в поисках того, что осталось на прилавках.

Разработанное приложение представляет собой интернет-магазин, который позволяет заказывать гитары онлайн по определённой цене. Также присутствует удобный поиск по производите и цене, для более быстрого поиска. Система также позволяет со стороны администратора редактировать список продуктов, отображаемых на складах, а также позволяет просматривать и редактировать пользователей, зарегистрировавшихся на данной платформе.

Перед разработкой были проанализированы аналогичные приложения в веб и мобильной среде, подход которых использовался в разрабатываемом приложение. Яркими примерами можно считать Musicmarket, Musicart. Данные приложения были описаны в первой главе.

# 1 Постановка задачи и обзор аналогов

## 1.1 Постановка задачи

Постановка задачи – это процесс формулировки назначения программного обеспечения и основных требований к нему. Описываются функциональные требования, определяющие функции, которые должно выполнять программное обеспечение, и эксплуатационные требования, определяющие характеристики его функционирования. Один из этапов построение задачи это формирование диаграммы использования продукта. Диаграмма вариантов использования веб-ресурса – диаграмма, отражающая отношения между актором (пользователем) и прецедент (компонентом системы). Диаграммы ВИ применяются при бизнес-анализе для моделирования видов работ, выполняемых организацией, и для моделирования функциональных требований к ПС при ее проектировании и разработке. Построение модели требований при необходимости дополняется их текстовым описанием. Спецификация затем станет основой для тестирования и документации, а на следующих этапах проектирования она дополняется и оформляется в виде диаграммы (в рамках ICONIX используется диаграмма последовательности, но в UML для этого имеются также диаграммы деятельности).

На диаграмме использования изображаются:

* user – группы лиц, взаимодействующих с нашей системой;
* admin – лицо, которое администрирует нашу систему;
* отношения между элементами диаграммы.

Правила при разработке диаграммы:

* каждый прецедент относится как минимум к одному действующему лицу;
* каждый прецедент имеет инициатора;
* каждый прецедент приводит к соответствующему результату.

В результате анализа требований, сформулированных на этапе планирования, была разработана диаграмма вариантов использования. С ее использованием будет проходить дальнейшая разработка функциональных возможностей веб-приложения. Если подводить итог, то диаграммы ВИ предназначены:

* чётко отделить систему от её окружения;
* определить действующих лиц (актёров), их взаимодействие с системой и ожидаемую функциональность системы;
* определить в глоссарии предметной области понятия, относящиеся к детальному описанию функциональности системы (то есть прецедентов).

Диаграмма вариантов использования представлена на рисунке 1.1.

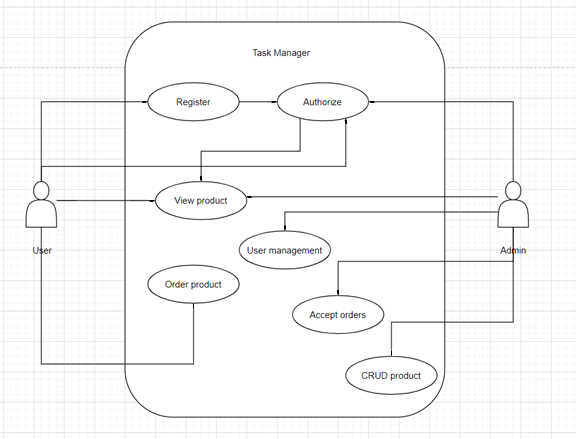


Рисунок 1.1 – Диаграмма вариантов использования

Из диаграммы вариантов использования видно действующих лиц, их взаимодействие с системой и ожидаемую функциональность системы. С ее использованием будет проходить дальнейшая разработка веб-приложения.

## 1.2 Обзор аналогов

Из открытых источников (интернет) были найдены аналоги веб приложения для управления задачами. Начнем с веб-приложения Musicmarket, изображённом на рисунке 1.2.

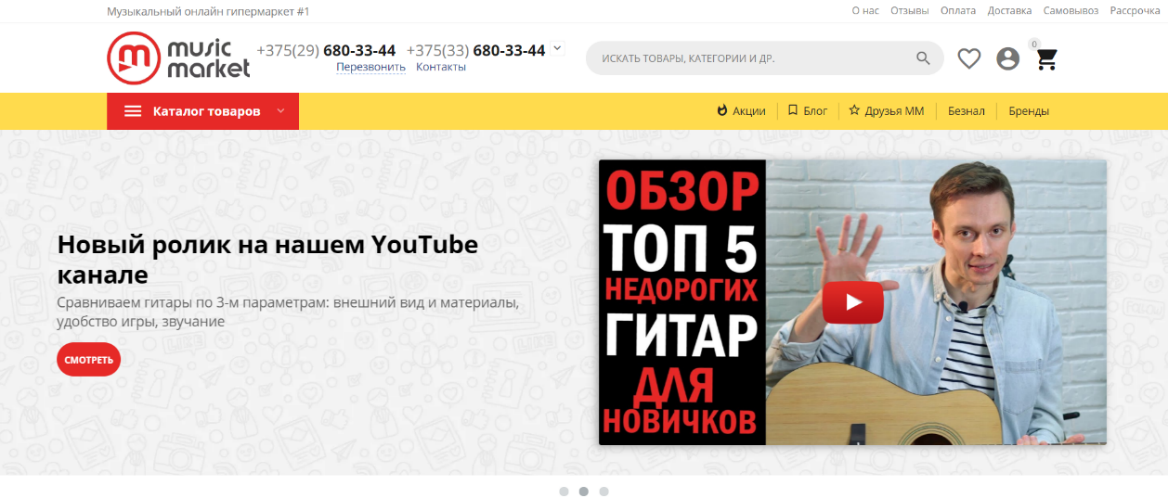


Рисунок 1.2. – Главная страница Musicmarket.

Функционал приложения включает в себя:

* просмотр каталога товаров;
* обратная связь;
* лайки;
* отзывы;
* корзина пользователя;
* быстрый поиск по сайту;
* просмотр действующих акций.

Следующее найденное приложение – это веб-приложение Muz, где главная страница показана на рисунке 1.3

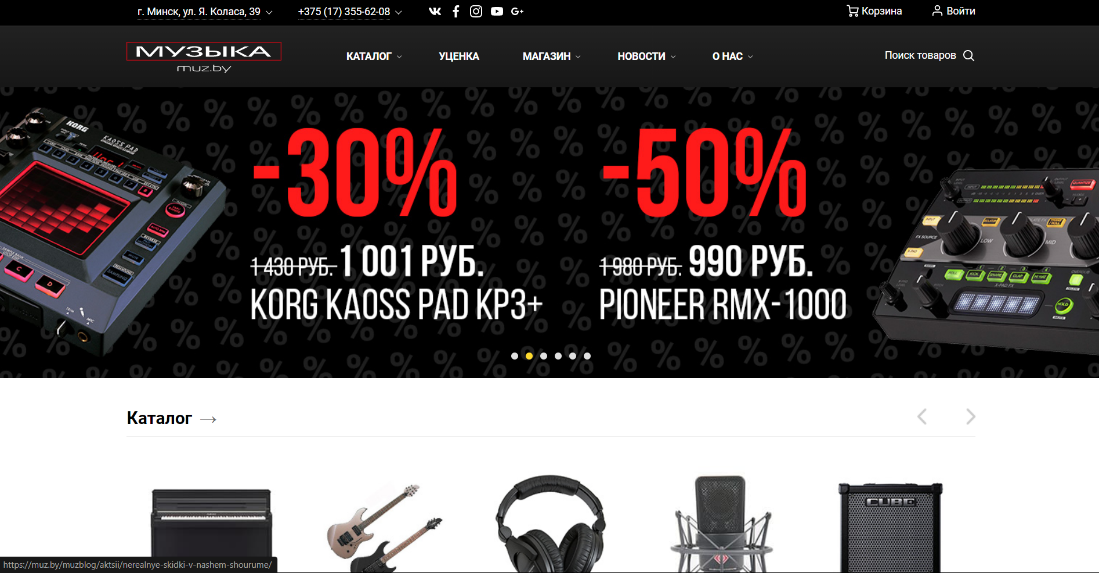


Рисунок 1.3 – Главная страница Muz.

Приложение дает следующие возможности:

* просмотр каталога товаров;
* обратная связь при помощи сторонних социальных сетей;
* лайки;
* отзывы;
* корзина пользователя;
* быстрый поиск по сайту;
* просмотр действующих акций;
* предоставление возможности сотрудничества.

В дальнейшем на рисунке 1.4 представлен процесс заказа товара.

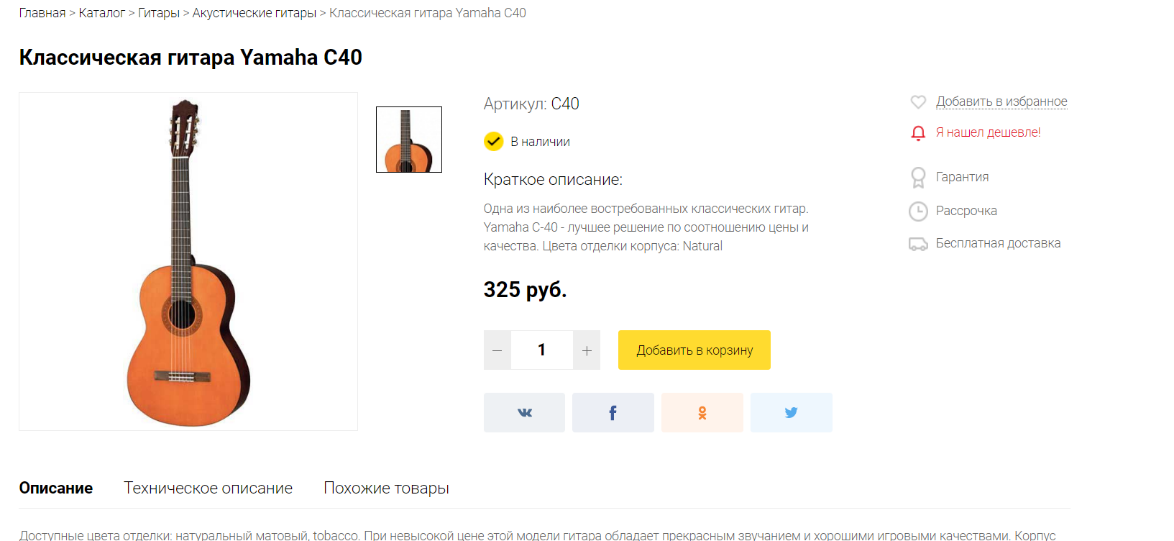


Рисунок 1.4 – Интерфейс заказа товара.

Дальше на рисунке 1.5 предоставлен интерфейс, где мы можем воспользоваться быстрым поиском по сайту.

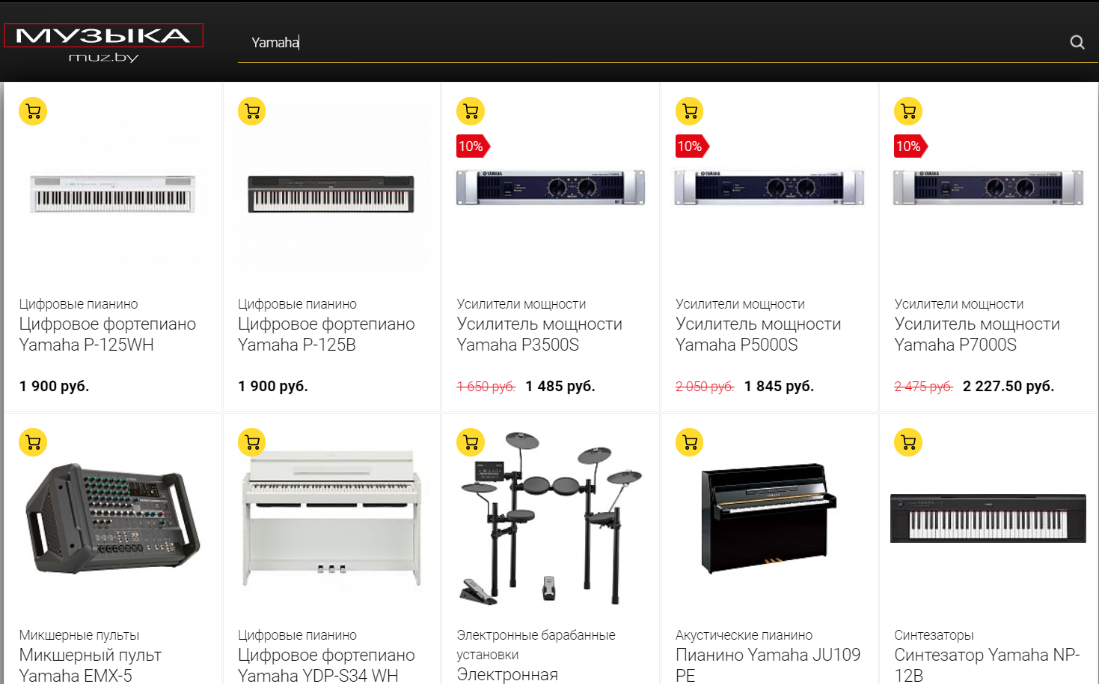


Рисунок 1.5 – Интерфейс быстрого поиска

# 2 Проектирование

## 2.1 Проектирование серверной и клиентской части

Серверная часть написана на NodeJS c помощью Express, Sequelize. Node.js – это кроссплатформенная среда исполнения JavaScript с открытым исходным кодом, которая выполняет код JavaScript вне веб-браузера. Node.js позволяет разработчикам использовать JavaScript для написания инструментов командной строки и для сценариев на стороне сервера - запуск сценариев на стороне сервера для создания динамического содержимого веб-страницы перед ее отправкой в ​​веб-браузер пользователя. Следовательно, Node.js представляет собой парадигму «JavaScript везде», объединяющую разработку веб-приложений вокруг единого языка программирования, а не разных языков для серверных и клиентских сценариев. Express - это минимальная и гибкая среда веб-приложений Node.js, предоставляющая надежный набор функций для веб-приложений и мобильных приложений. Sequelize - это основанная на promise ORM Node.js для Postgres, MySQL, MariaDB, SQLite и Microsoft SQL Server. Socket.io – JavaScript-библиотека для веб-приложений и обмена данными в реальном времени поверх протокола WebSocket. Архитектура программного средства представляет себя Клиент – Сервер. Она изображена на рисунке 2.1.

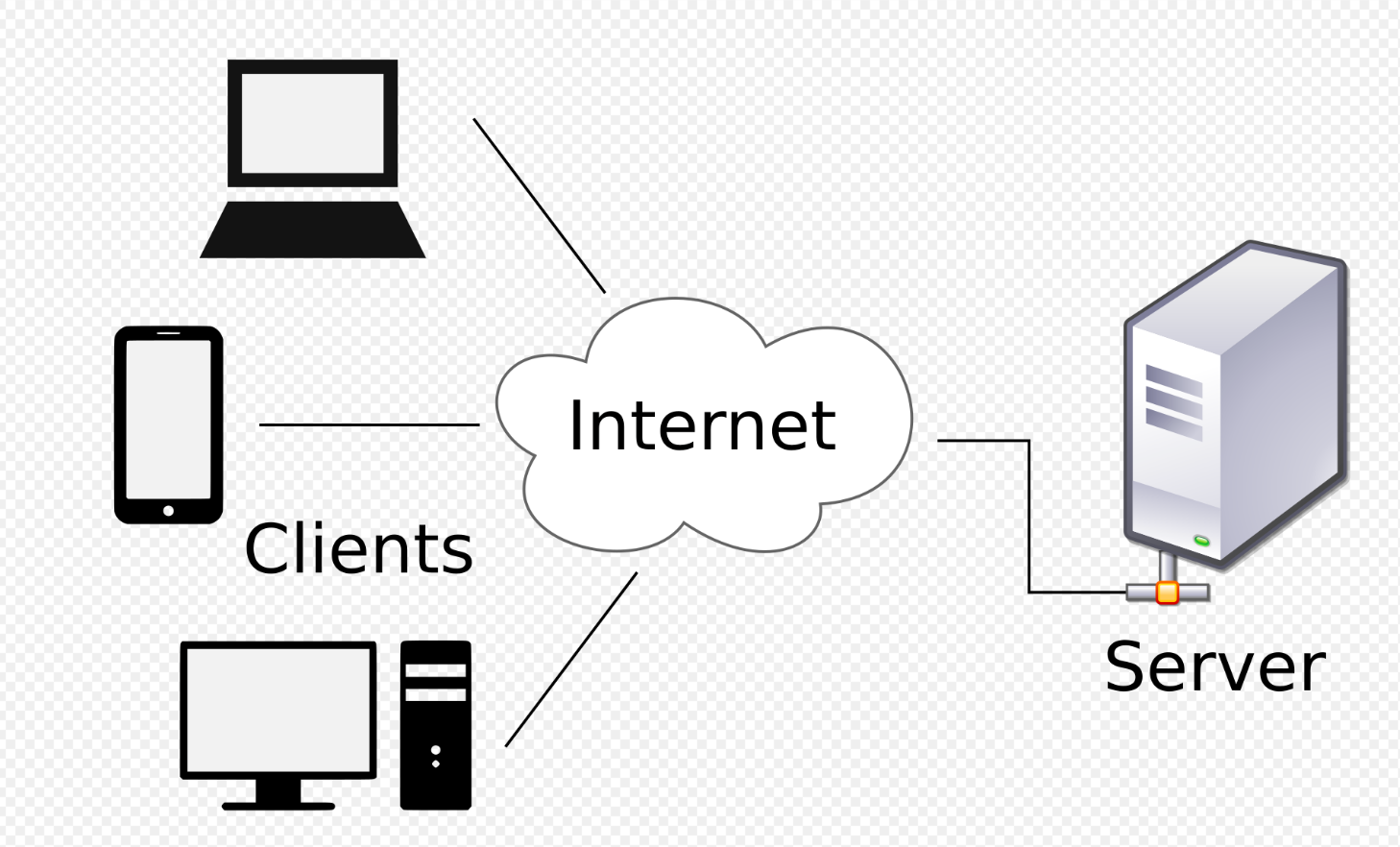


Рисунок 2.1 – Архитектура клиент-сервер.

Клиент предоставляет себя браузерным приложением, которое посылает запросы (request) по сети с помощью HTTP протокола, а сервер должен, принимать запросы от клиента и давать ответ (response). В нашем случае приложения поделено на две части. Это frond-end (клиентская) и back-end(серверная). Клиентская часть разрабатывалась с помощью библиотеки Bootstrap, css, html, js.

Так же клиентская часть приложения имеет архитектуру TP т. е. Traditional Page. Traditional Page Lifecycle можно посмотреть на рисунке 2.2.

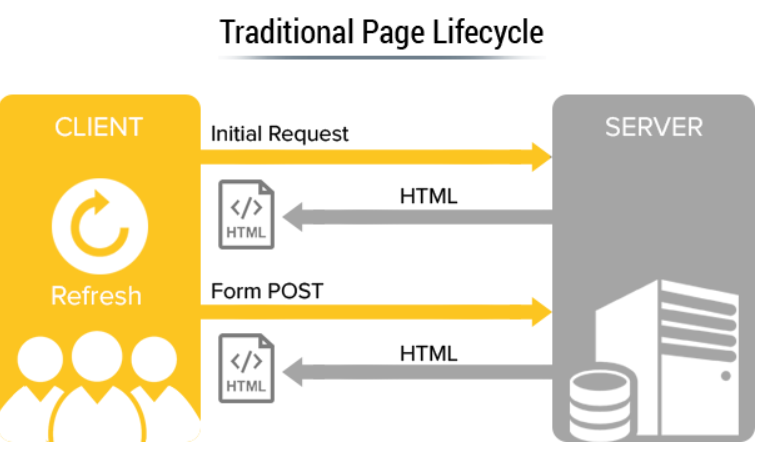


Рисунок 2.2 – Архитектура Traditional Page Lifecycle.

В традиционных приложениях сервер нам всегда отсылает HTML страницу после каждой акции. Также при разработке приложения учитывался MVC шаблон. Модель предоставляет остальным компонентам приложения объектно-ориентированное отображение данных (таких как каталог продуктов или список заказов). Объекты модели могут осуществлять загрузку и сохранение данных в реляционной базе данных, а также реализуют бизнес-логику. На рисунке 2.1 показано взаимодействие между слоями в MVC. Шаблон [Model-View-Controller](https://ru.wikipedia.org/wiki/Model-View-Controller) для [веб-приложений](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), а также обеспечивает их интеграцию с [веб-сервером](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80) и сервером баз данных. Модель предоставляет остальным компонентам приложения объектно-ориентированное отображение данных (таких как каталог продуктов или список заказов). Объекты модели могут осуществлять загрузку и сохранение данных в реляционной базе данных, а также реализуют бизнес-логику.

На рисунке 2.3 показано взаимодействие между слоями в MVC.

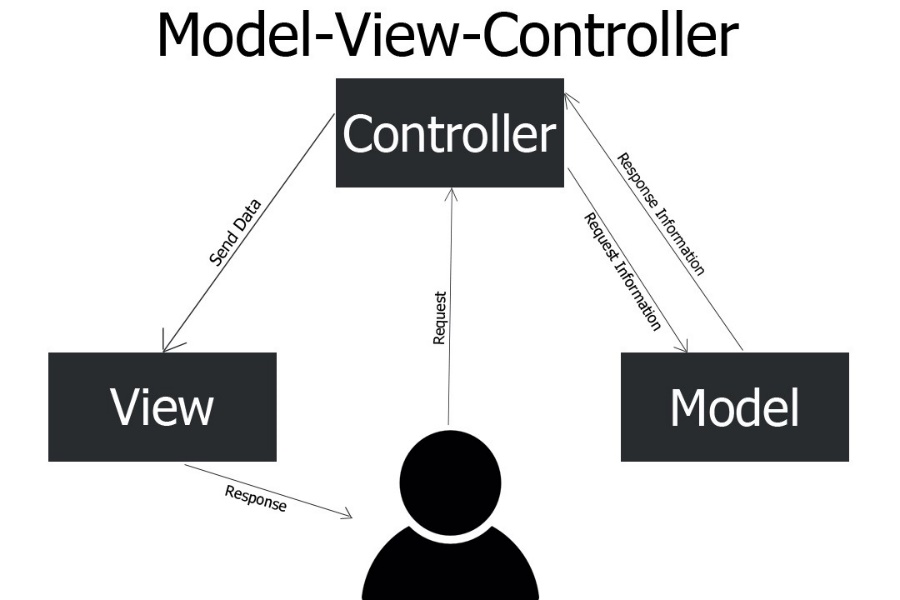


Рисунок 2.3 – Взаимодействия между слоями.

Для физического представления архитектуры MVC, ниже будет приведена структура разработанного приложения в виде директорий проекта на рисунке 2.4.

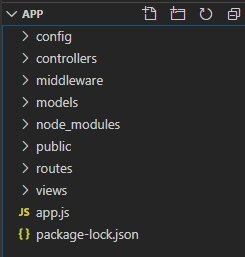


Рисунок 2.4 – Структура разработанного приложения.

В config находиться файл для подключения к БД, controllers – контроллер, routes – маршрутизация, models – модели в БД, routes – настройка маршрутизации, public – статические файлы, views– клиентская часть, middleware - серверная часть pipeline обработки. Листинг компонентов сервера и клиента предоставлен в приложении А. Клиентская часть написана при помощи Bootstrap и JS. Приложение-клиент представляет собой веб-приложение, написанное на языках JavaScript, HTML, CSS придерживаясь технологий Bootstrap. React – JavaScript-библиотека для создания пользовательских интерфейсов. Bootstrap – это бесплатная CSS-инфраструктура с открытым исходным кодом, ориентированная на адаптивную мобильную веб-разработку. Он содержит шаблоны дизайна на основе CSS и JavaScript для типографики, форм, кнопок, навигации и других компонентов интерфейса. Также было использовано множество ресурсов Bootstrap, выраженных в иконках, header для сайта, а также некоторые заготовки для отображения контента. Ниже на рисунке 2.5 приведены использованные компоненты Bootstrap.

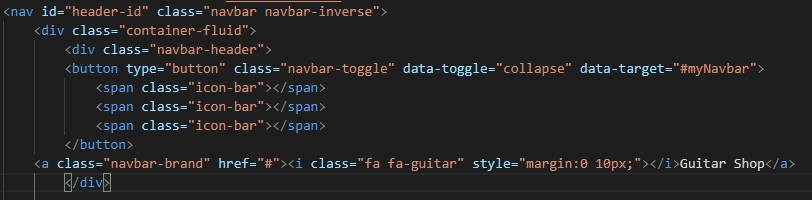




Рисунок 2.5 – Компонент Bootstrap

Основная логика взаимодействия клиента с сервером, заключается в использовании так называемых end points. End points это конечные точки взаимодействия пользователя с сервером, выраженные в rout в самом проекте. Первоначально для проверки end point было использовано вспомогательно программное обеспечение Postman, про него чуть ниже. На рисунке 2.6 предоставлены некоторые end point, а именно end point для доступа к контенту и списку товаров для администратора, чтобы наше приложение могло общаться с сервером.



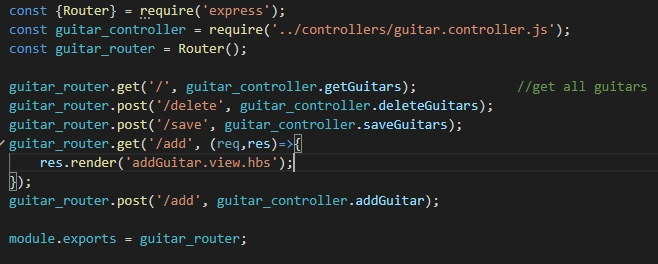


Рисунок 2.6 – Конечные точки приложения.

## 2.2 Выбор СУБД (системы управления базами данных)

Стоит отметить, что в качестве системы управления базами данных был выбран MSSQL, ниже проведём основные аспекты сравнения СУБД, таких как MSSQL, MongoDB. Сравнения будут проходить между этими СУБД, потому что выбор был именно между ними. Для нашего приложения первоначально была разработана концепция хранения данных в json, так как он довольно гибок в использовании и легко ложиться на объектно-ориентированную модель сущностей, но для большей гибкости мы решили не останавливаться только на json формате и начали смотреть реляционные базы данных.

Для начала начнём с того, что же такое базы данных и зачем их вообще используют. База данных - это организованная коллекция структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе. База данных обычно контролируется системой управления базами данных (СУБД). Данные и СУБД вместе с приложениями, которые с ними связаны, называются системой баз данных, часто сокращаемой до просто базы данных.

Наконец давайте рассмотрим типы баз данных. Существует много разных типов баз данных, но среди них выделяют два основных типа:

* Реляционные базы данных (SQL);
* Нереляционные базы данных (NoSQL).

Различия между ними заключаются в том, как они спроектированы, какие типы данных поддерживают, как хранят информацию, что можно увидеть на рисунке 2.7.

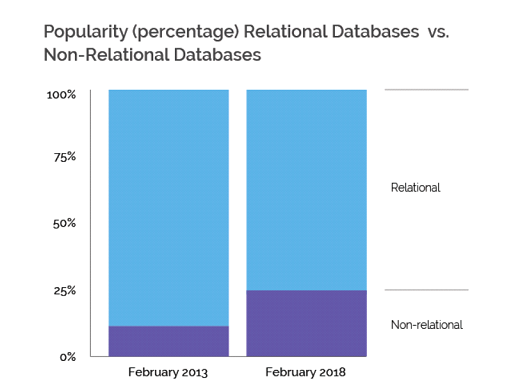


Рисунок 2.7 – Соотношение использования SQL и NoSQL баз данных

Реляционные БД (SQL)

СУБД более известны и понятны, чем их двоюродные братья по NoSQL. Реляционные базы данных появились в 70-х годах для хранения данных в соответствии со схемой, позволяющей отображать данные в виде таблиц со строками и столбцами. Рассматривайте реляционную базу данных как набор таблиц, каждая из которых имеет схему, которая представляет фиксированные атрибуты и типы данных, которые будут иметь элементы в таблице. Все СУБД предоставляют функции для чтения, создания, обновления и удаления данных, как правило, с помощью операторов языка структурированных запросов (SQL).

Таблицы в реляционной базе данных имеют связанные с ними ключи, которые используются для идентификации определенных столбцов или строк таблицы и ускорения доступа к конкретной таблице, строке или интересующему столбцу.

Целостность данных особенно важна в реляционных базах данных, и СУБД используют ряд ограничений для обеспечения надежности и точности данных, содержащихся в ваших таблицах. Хотя существует много реляционных баз данных, которые со временем стали самыми популярными.

MySQL – это СУБД с открытым исходным кодом, основанная на языке структурированных запросов (SQL). MySQL работает практически на всех платформах, включая Linux, UNIX и Windows.

Oracle – это многомодельная система управления базами данных, производимая и продаваемая корпорацией Oracle.

Microsoft SQL Server – это СУБД, которая поддерживает широкий спектр приложений для обработки транзакций, бизнес-аналитики и аналитики в корпоративных ИТ-средах.

PostgreSQL – PostgreSQL, часто просто Postgres, является объектно-реляционной системой управления базами данных (ORDBMS) с акцентом на расширяемость и соответствие стандартам.

Преимущества:

* реляционные базы данных являются хорошо документированными и зрелыми технологиями, а СУБД продаются и обслуживаются рядом известных корпораций.
* стандарты SQL четко определены и общеприняты.
* большой пул квалифицированных разработчиков имеет опыт работы с SQL и RDBMS.
* все RDBMS совместимы с ACID, что означает, что они удовлетворяют требованиям атомарности, согласованности, изоляции и долговечности.

Недостатки:

* + СУБД плохо работают или не работают вообще с неструктурированными или полуструктурированными данными из-за ограничений схемы и типа. Это делает их неподходящими для больших аналитических или IoT-событий.
* таблицы в вашей реляционной базе данных не обязательно будут отображаться один на один с объектом или классом, представляющим те же данные.
* при переносе одной РСУБД в другую схемы и типы должны, как правило, быть идентичными в исходной и целевой таблицах для перехода на работу (ограничение схемы). По многим из тех же причин чрезвычайно сложные наборы данных или те, которые содержат записи переменной длины, как правило, трудно обрабатывать с помощью схемы RDBMS.

Базы данных NoSQL стали популярной альтернативой реляционным базам данных, поскольку веб-приложения становились все более сложными. NoSQL / Нереляционные базы данных могут принимать различные формы. Однако критическое различие между NoSQL и реляционными базами данных состоит в том, что схемы RDBMS жестко определяют, как все данные, вставляемые в базу данных, должны быть типизированы и составлены, тогда как базы данных NoSQL могут быть независимыми от схемы, позволяя хранить и манипулировать неструктурированными и полуструктурированными данными.

Существует множество типов нереляционных баз данных, затронем только некоторые из них:

* Хранилища ключей-значений (Redis) – представляют собой чрезвычайно простые системы управления базами данных, которые хранят только пары ключ-значение
* Хранилища документов (MongoDb) – представляют собой системы без схем, которые хранят данные в форме документов JSON. Имя документа - это ключ, а содержимое документа, каким бы оно ни было, является значением.
* База данных графов (Neo4j) – представляют данные в виде сети связанных узлов или объектов для облегчения визуализации данных и анализа графиков.
* Поскольку существует так много типов и разнообразных приложений баз данных NoSQL, их сложно зафиксировать, но обычно:
* Модели данных без схемы более гибки и просты в администрировании.
* Базы данных NoSQL, как правило, более масштабируемы по горизонтали и отказоустойчивые.

Данные могут быть легко распределены по разным узлам. Чтобы повысить доступность и или допуск раздела, вы можете выбрать, чтобы данные на некоторых узлах были «в конечном итоге согласованными».

Недостатки:

* Они также зависят от типа базы данных.
* Базы данных NoSQL, как правило, менее широко распространены и зрелы, чем решения RDBMS, поэтому часто требуются специальные знания.
* Существует ряд форматов и ограничений, специфичных для каждого типа базы данных.

Microsoft SQL Server - одна из наиболее мощных систем работы с базами данных в архитектуре "клиент-сервер". Особенность системы - работа сервера только в операционных системах ряда Microsoft Windows NT - NT Server 4.0, 2000 Server, Server 2003, при этом клиентская часть может взаимо-действовать с сервером из Microsoft Windows 98 и других операционных систем. Рекомендуемая файловая система для SQL Server - NTFS, хотя возможна работа и в системе FAT.

В своем составе система имеет средства создания баз данных, работы с информацией баз данных, перенесения данных из других систем и в другие системы, резервного копирования и восстановления данных, развитую систему транзакций, систему репликации данных, реляционную подсистему для анализа, оптимизации и выполнения запросов клиентов, систему безопасности для управления правами доступа к объектам базы данных и пр.

## 2.3 Диаграммы UML

UML – это унифицированный графический язык моделирования для описания, визуализации, проектирования и документирования объектно-ориентированных систем. UML призван поддерживать процесс моделирования ПС на основе объектно-ориентированного подхода, организовывать взаимосвязь концептуальных и программных понятий, отражать проблемы масштабирования сложных систем. Модели на UML используются на всех этапах жизненного цикла ПС, начиная с бизнес-анализа и заканчивая сопровождением системы. Мы уже использовали поведенческую UML диаграмму вариантов использования для описания всех возможных возможностей для двух видом пользователей (зарегистрированного и незарегистрированного) в системе. Далее с помощью структурной диаграммы будет описана структура моделей классов.

## 2.4. Структура моделей и классов

Как известно, база данных – это хранилище структурированной информации, пассивное по своей сути. Бизнес-логика приложения реализуется где-то вне базы, в виде «набора действий для достижения требуемого результата». В случае внесения изменений в хранимый набор данных результатом должно стать новое состояние базы. Если предметная область автоматизации представляет собой систему взаимодействующих значений, то ее можно описать ORM-моделью. Диаграмма классов является ключевым элементом редактора UML-диаграмм, поскольку зачастую приложения генерируются именно с диаграммы классов. Диаграмма классов – это набор статических, декларативных элементов модели. Диаграммы классов могут применяться и при прямом проектировании, то есть в процессе разработки новой системы, и при обратном проектировании – описании существующих и используемых систем. Информация с диаграммы классов напрямую отображается в исходный код приложения. Таким образом, диаграмма классов – конечный результат проектирования и отправная точка процесса разработки. В соответствии с моделью базы данных и поставленными задачами была разработана модель классов, представленная на рисунке 2.8.

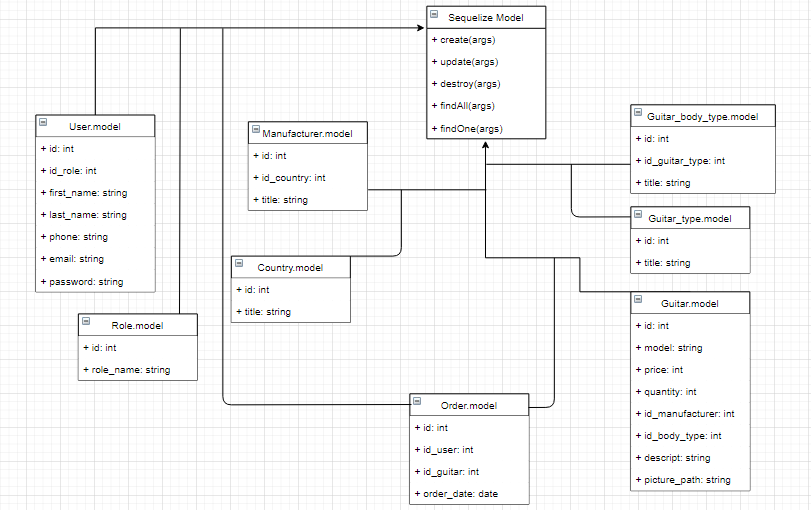


Рисунок 2.8 – UML диаграммы моделей

Такая модель не только образует логическую структуру базы данных, но обладает всеми свойствами программы, каковой, по сути, и является – в вычислительной среде, образованной методами упомянутых абстрактных сущностей.

# 3 Разработка функциональной модели и модели данных ПС

## 3.1 Выполняемые функции

Разрабатываемое веб-приложение представляет собой, интернет-магазин, где главное цель — это дать лёгкую возможность пользователю заказать интересующий его продукт. В результате была создана база данных, которая удовлетворяет поставленным задачам. Также была создана удобная работа с контентом, включая регистрацию и авторизацию пользователя. Данная реализация достаточно проста и универсальна, с ее помощью в дальнейшем можно изменять и дополнять структуру базы данных. Функционально программное средство должно выполнять следующие задачи:

* Аутентификация пользователей.
* Создание, удаление товара, редактирование.
* Просмотр товара.
* Заказ товара.
* Быстрая сортировка по необходимым критериям (цена, производитель).

Помимо вышеуказанного, в ходе разработки был добавлен функционал для регистрации. Во время разработки RESTful API для данного проекта возникла необходимость реализовать аутентификацию пользователя. Одним из принципов REST является независимость от состояния (stateless). Это значит, что клиент должен сам позаботиться о своей аутентификации при каждом запросе. JSON Web Token или просто JWT. JSON JWT имеет некоторые преимущества – он самодостаточен, все необходимые для аутентификации данные можно хранить в самом токене. Последовательно рассмотрим устройство токена. JWT состоит из трех основных частей: заголовка (header), нагрузки (payload) и подписи (signature). Заголовок и нагрузка формируются отдельно, а затем на их основе вычисляется подпись. После первого логина, клиенту возвращается сгенерированный сервером JWT. При каждом следующем запросе клиент должен передавать JWT установленным API способом

Результатом авторизации является JWT token — некий ключ (обычно просто набор символов), предъявление которого является пропуском к защищенным ресурсам. Обращение к ним в самом простом случае происходит по HTTPS с указанием в заголовках или в качестве одного из параметров полученного token'а.

Сервер декодирует header и payload и проверяет зарезервированные поля. Если все в порядке, по указанному в header алгоритму составляется подпись. Если полученная подпись совпадает с переданной, пользователя авторизуют

В протоколе описано несколько вариантов авторизации, подходящих для различных ситуаций:

* авторизация для приложений, имеющих серверную часть (чаще всего, это сайты и веб-приложения);
* авторизация для полностью клиентских приложений (мобильные и desktop-приложения);
* авторизация по логину и паролю;
* восстановление предыдущей авторизации.

В нашем проекте было решено использовать авторизацию через серверную часть.

Перенаправление на страницу авторизации.

На странице авторизации у пользователя запрашивается подтверждение выдачи прав

В случае согласия пользователя, браузер перенаправляется на URL, указанный при открытии страницы авторизации, с добавлением в GET-параметры специального ключа – authorization code

Сервер приложения выполняет POST-запрос с полученным authorization code в качестве параметра. В результате этого запроса возвращается JWTtoken

Это самый сложный вариант авторизации, но только он позволяет сервису однозначно установить приложение, обращающееся за авторизацией (это происходит при коммуникации между серверами на последнем шаге). Во всех остальных вариантах авторизация происходит полностью на клиенте и по понятным причинам возможна маскировка одного приложения под другое.

Для получения доступа к данным пользователя в стороннем сервисе, необходимо зарегистрировать свое приложение в системе данного сервиса. После регистрации приложения сервис создаст учётные данные клиента - идентификатор клиента (client ID) и секрет клиента (client secret). Идентификатор клиента представляет собой публично доступную строку, которая используется API сервиса для идентификации приложения, а также используется для создания авторизационных URL для пользователей. Секрет клиента используется для аутентификации подлинности приложения для API сервиса, когда приложение запрашивает доступ к аккаунту пользователя. Секрет клиента должен быть известен только приложению и API.

При запросе кода авторизации (authorization code), необходимо передать на сторонний сервис идентификатор приложения (client\_id) и адрес перенаправления (redirect\_uri). После того как, пользователь разрешит приложению доступ к свои данным, и при условии, что приложение с указанным client\_id зарегистрировано и имеет указанный адрес перенаправления в списке разрешённых, браузер перенаправляется на URL, указанный в redirect\_uri, с добавлением в GET-параметры специального ключа – authorization code. После чего приложение отправляет новый запрос на сторонний сервис, содержащий поля client\_id, secret\_key, и authorization code, если все поля являются актуальными сервер отправит в ответ JWT token. В дальнейшем для получения доступа к ресурсам пользователя достаточно отправить на сторонний сервис JWT token.

На рисунке 3.1 приведён код реализации промежуточного конвейера обработки для проверки и осуществления аутентификации пользователей.



Рисунок 3.1 – JWT token пример

## 3.2 Модель базы данных

Диаграмма связей таблиц для необходимой базы данных представлена на рисунке 3.2. Всего в рамках курсового проекта было разработано 8 таблиц.

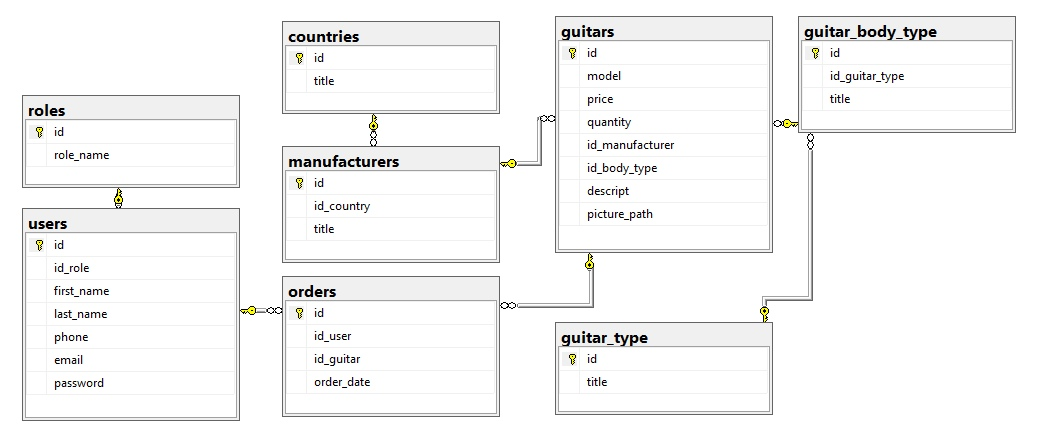


Рисунок 3.2 – Структура модели базы данных.

Описание всех таблиц приведено ниже.

Таблица users хранит в себе информацию о пользователях базы данных:

* id – идентификатора пользователя;
* id\_role – роль пользователя;
* first\_name – имя пользователя;
* last\_name – фамилию пользователя;
* phone – телефон пользователя;
* email – почту ползователя;
* password – пароль пользователя;

Таблица roles хранит идентификатор роли и имя роли:

* id – идентификатор;
* role\_name – название роли;

Таблица orders служит для хранения следующей информации о заказах:

* id – идентификатор;
* id\_user – идентификатор пользователя;
* id\_guitar – идентификатор товара;
* order\_date – дата заказа;

Таблица manufacturer служит для хранения следующей информации о производителях:

* id – идентификатор;
* id\_country – идентификатор страны;
* title – имя производителя;

Таблица guitars служит для хранения следующей информации о товарах:

* id – идентификатор;
* model – название модели;
* price – цена;
* quantity – количество;
* id\_manufacturer – идентификатор производителя;
* id\_body\_type – идентификатор типа корпуса гитары;
* descript – описание товара;
* picture\_path – путь к изображениям гитары.

Таблица guitar\_type служит для хранения следующей информации о типе гитар:

* id – идентификатор;
* title – название типа;

Таблица guitar\_body\_type служит для хранения следующей информации о типе корпуса гитар:

* id – идентификатор;
* id\_guitar\_type – идентификатор типа гитары;
* title – название типа;

Таблица countries служит для хранения следующей информации о странах:

* id – идентификатор;
* title – название страны;

Для отображения данных с БД в объектное представление был использован sequelize. Sequelize – ORM основана на обещаниях Node.js. Во второй главе была продемонстрирована UML схема моделей (таблиц) БД.

# 4 Руководство пользователя

При входе на главную страницу, если пользователь не авторизирован или не зарегистрирован, то необходимо пройти процедуру аутентификация или регистрации. Форма для аутентификации изображена на рисунке 4.1.

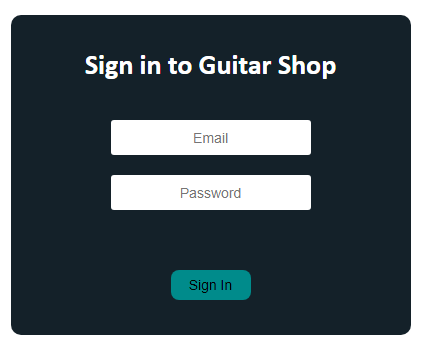


Рисунок 4.1 – Форма для логина

Процедура регистрации изображена на рисунке 4.2, расположенном ниже.

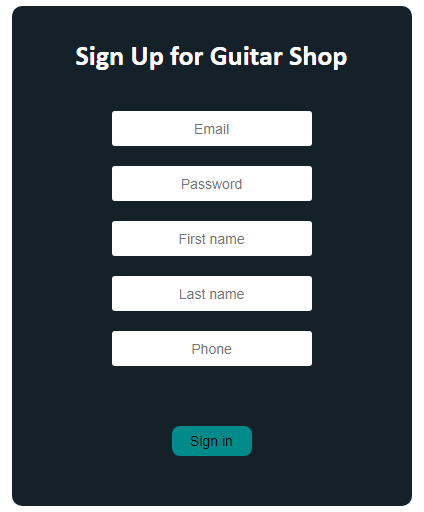


Рисунок 4.2 – Форма для регистрации

После регистрации пользователь может осуществлять заказы перейдя по ссылке Guitars, расположенной в заголовке страницы, изображённом на рисунке 4.3.

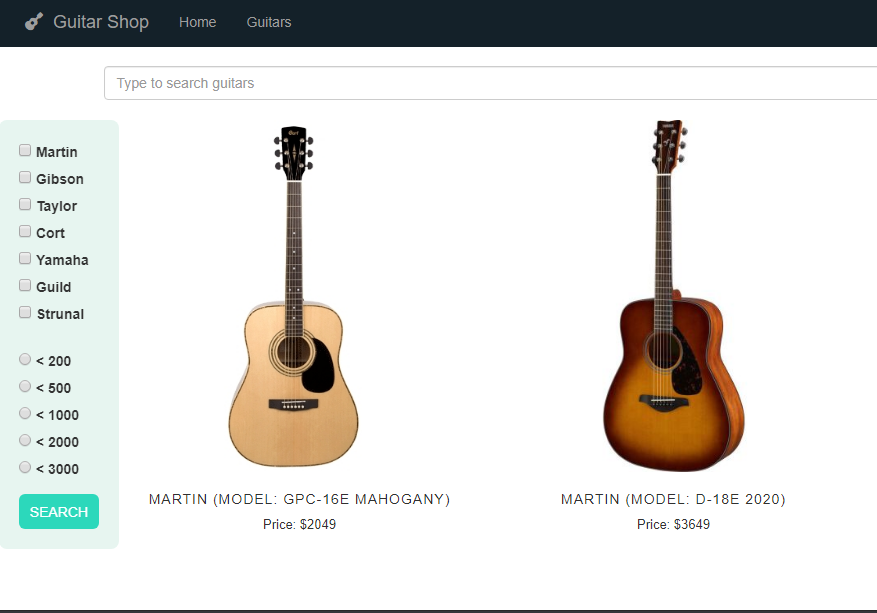


Рисунок 4.3 – Отображение контента для пользователей

Далее пользователь может перейти по ссылке, расположенной под изображением товара, расположенном на рисунке 4.4.



Рисунок 4.4 – Ссылка на товар

После того, как пользователь перешёл по ссылке, открывается описание выбранного товара, как видно на рисунке 4.5

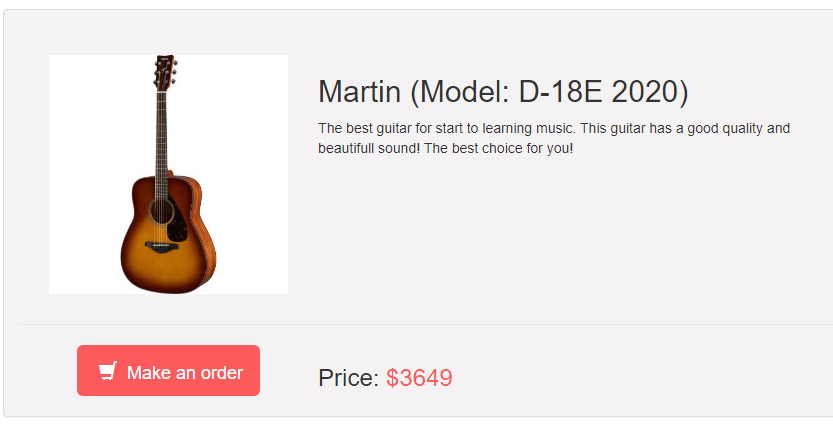


Рисунок 4.5 – Описание товара.

При нажатии на кнопку Make an order осуществляется заказ товара и идёт перенаправление на страницу с надписью о успешно заказанном товаре , как видно на рисунке 4.6.

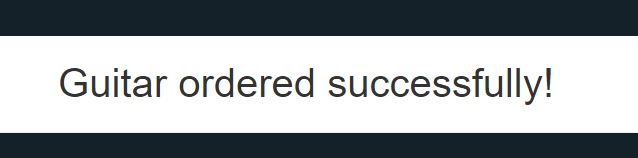


Рисунок 4.6 – Страница с уведомлением о успешном заказе товара

Также пользователь может выполнить logout и очистить данные авторизации, нажав на ссылку Logout, находящейся в заголовке страницы, после чего он не сможет осуществлять заказы.

# 5 Тестирование

Первоначально, при создании голого API, для тестирования приложения я использовал программное средство POSTMAN, которое облегчило построение правильно работающего интерфейса.

Пройдёмся по тестированию моего программного средства с помощью браузера и покажем результаты.

Первым делом протестируем проверку на полях при логине на корректность введённых данных, что показано на рисунке 5.1.

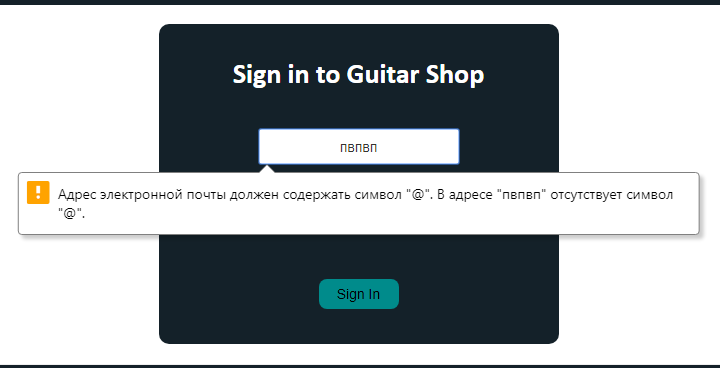


Рисунок 5.1 – Проверка на корректность ввода для логина

Далее протестируем защиту на ввод запросов в URL авторизированным клиентам для получения контента, принадлежащего только администратору, что можно увидеть на рисунке 5.2.



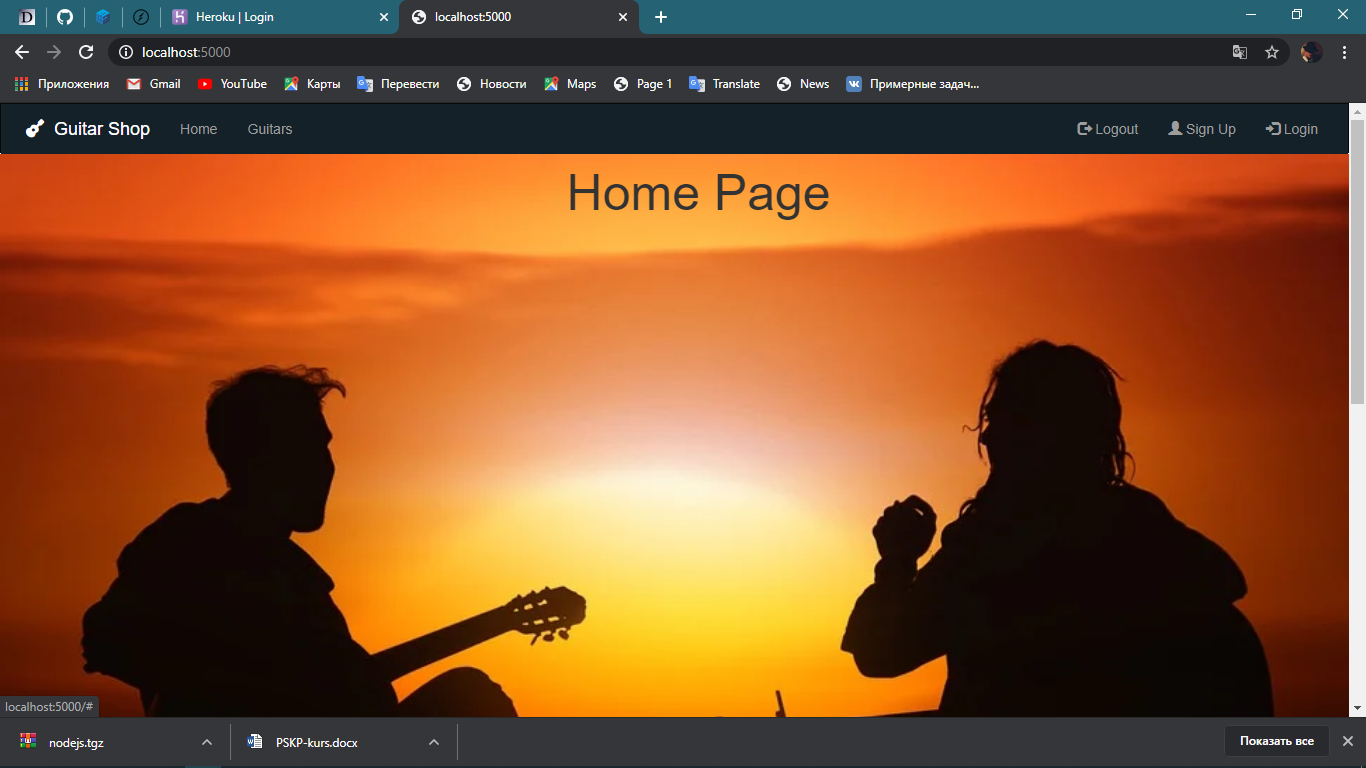


Рисунок 5.2 – Проверка на get-запросы в url строке браузера

Как мы видим, если пользователь пытается получить доступ к ресурсам, принадлежащим клиенту, то сервер сразу же перенаправляет клиента на главную страницу сайта.

Теперь проверим поведение сайта при переходе не авторизированного или не зарегистрированного пользователя через url к ресурсам администратора, что изображено на рисунке 5.3.



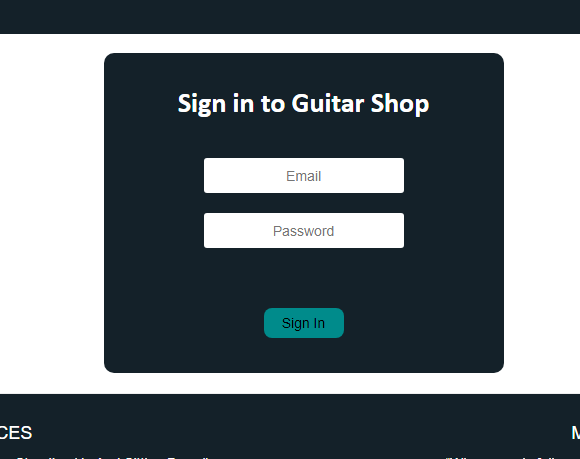


Рисунок 5.3 – Проверка не авторизированного пользователя.

И наконец проверим возможность осуществлять заказы товаров не авторизированным или не зарегистрированным пользователям, что можно увидеть ниже на рисунках 5.4 – 5.5 соответственно.

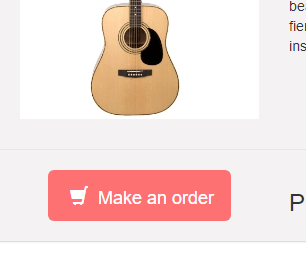


Рисунок 5.4 – Кнопка для заказа товара

И вот снова мы можем видеть, что не авторизированные пользователи не могут выполнять никаких операций кроме просмотра контента сайта и набора товаров.

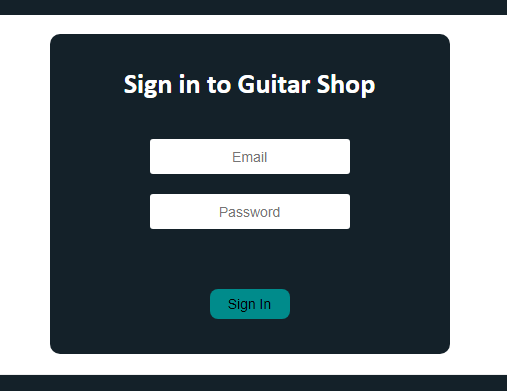


Рисунок 5.5 – Перенаправление пользователя на окно логина

Как мы видим, данное приложение защищено достаточно хорошо от нежелательного доступа клиентов к информации, предназначенной для администратора сайта. В общем, можно заметить, что программа довольно неплохо прошла часть тестирования.

# Заключение

В курсовом проекте рассмотрены основные аналоги, их преимущества и недостатки, а также изучены возможности рассматриваемых программных средств. Произведен обзор теоретического материала по теме курсового проекта.

Разработано веб-приложение, которое было поделено на две части. Это клиентская и серверная часть, где использовался язык разработки – JavaScript, в котором есть преимущества кроссплатформенной разработки на платформе NodeJS. В качестве СУБД была использована MSSQL Management Studio. Клиентская часть была реализована с помощью библиотеки Handlebars, которая использовала API сервера.

При разработке программного средства была спроектирована структура базы данных, спроектирован и разработан интерфейс с возможностью расширения, реализована форма работы с пользовательскими задачами, реализованы формы поиска.

Отображение, бизнес логика и хранилище данных максимально независимы друг от друга для возможности расширения.

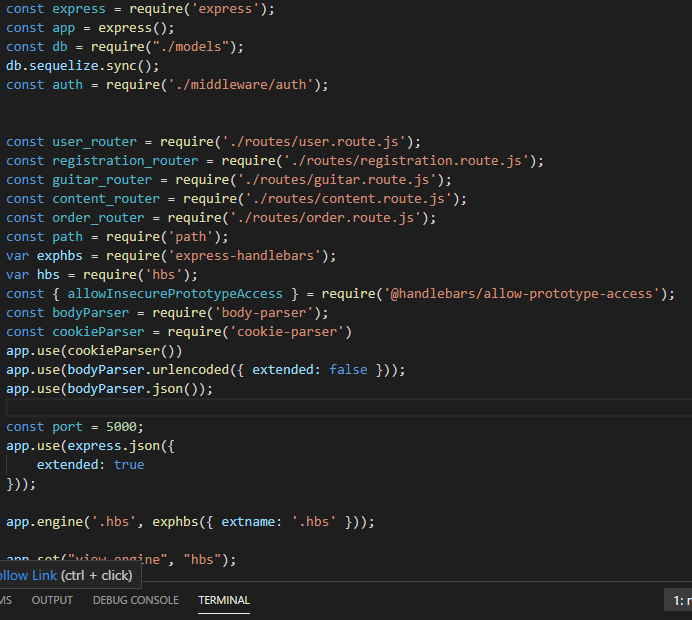
При составлении руководства пользователя была подробно описана работа с программным средством. Так же было проведено тестирование, показавшее что продукт соблюдает все ранее установленные требования.

# Список литературы

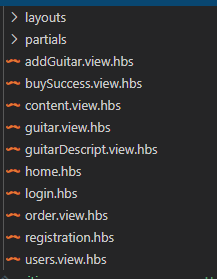
1. Handlebars Documentation – <https://handlebarsjs.com/>
2. Node.js Documentation – <https://nodejs.org/docs/latest-v12.x/api/>
3. Microsoft Documentation – Режим доступа: <https://docs.microsoft.com/en-us/>
4. SQL.RU [Электронный ресурс] – Режим доступа: <https://sql.ru>.
5. Sequelize Manual – <https://sequelize.org/master/index.html>
6. Finding solutions to problems – <https://stackoverflow.com/>

# Приложение A

Листинг инициализация запуска сервера.



Структура клиентской части



Листинг одного из компонентов клиентской части.

