

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ
Зав. каф. ЭВМ
_____ Б.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему
ВЕБ-ПРИЛОЖЕНИЕ ПО ПРОДАЖЕ МУЗЫКАЛЬНОГО ОБОРУДОВАНИЯ

БГУИР ДП 1–40 02 01 01 078 ПЗ

Студент	В.А. Петрикевич
Руководитель	В.В. Старовойтов
Консультанты:	
от кафедры ЭВМ	В.В. Старовойтов
по экономической части	О.А. Матяс
Нормоконтролер	Е.Е. Клинецвич
Рецензент	

МИНСК 2022

РЕФЕРАТ

Дипломный проект представлен следующим образом. Электронные носители: 1 компакт-диск. Чертежный материал: 6 листов формата А1. Пояснительная записка: 103 страницы, 26 рисунков, 29 таблиц, 10 литературных источников, 3 приложения.

Ключевые слова: веб-приложение, клиент, сервер, база данных, магазин, позиция, заказ, управление, покупатели.

Предметная область находится на пересечении музыкальных ритейлеров и онлайн-магазинов. Объектом разработки является веб-приложение.

Целью данного дипломного проекта является проектирование и реализация веб-приложения по продаже музыкального оборудования.

При разработке был использован язык Ruby, среда разработки JetBrains RubyMine. В качестве системы управления базами данных была выбрана PostgreSQL.

В результате разработки реализовано веб-приложение, предоставляющее возможность авторизованным пользователям заказывать и приобретать товары в магазине музыкального оборудования.

Данный продукт может быть использован для работы с каталогом оборудования магазина и осуществления работы по обработке заказов по продаже данного оборудования.

Разработанное приложение можно считать экономически эффективным. Оно упрощает взаимодействие потребителя и ритейлера.

Указанная в предполагаемом функционале часть задач дипломного проектирования реализована в полном объеме. При этом присутствует возможность дальнейшего расширения и улучшения функциональности проекта посредством добавления поддержки альтернативных способов ввода и обработки данных.

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет: ФКСиС. Кафедра: ЭВМ.

Специальность: 40 02 01 «Вычислительные машины, системы и сети».

Специализация: 40 02 01-01 «Проектирование и применение локальных компьютерных сетей».

УТВЕРЖДАЮ

Заведующий кафедрой ЭВМ

_____ Б.В.Никульшин

« ____ » _____ 2022 г.

ЗАДАНИЕ

по дипломному проекту студента
Петрикевича Валерия Александровича

1 Тема проекта: «Веб-приложение по продаже музыкального оборудования»
– утверждена приказом по университету от 1 апреля 2022 г. № 892-с.

2 Срок сдачи студентом законченного проекта: 1 июня 2022 г.

3 Исходные данные к проекту:

3.1 Протокол взаимодействия: HTTP, HTTPS.

3.2 Формат обмена данными: JSON.

3.3 Операционная система: macOS Monterey 12.3.

3.4 Среда разработки: JetBrains RubyMine.

3.5 Языки программирования: Ruby, JavaScript.

4 Содержание пояснительной записки (перечень подлежащих разработке вопросов):

Введение 1. Обзор литературы. 2. Системное проектирование.
3. Функциональное проектирование. 4. Разработка программных модулей.
5. Программа и методика испытаний. 6. Руководство пользователя.
7. Экономическое обоснование разработки. Заключение. Список использованных источников. Приложения.

5 Перечень графического материала (с точным указанием обязательных чертежей):

5.1 Вводный плакат. Плакат.

5.2 Веб-приложение по продаже музыкального оборудования.

Схема структурная.

5.3 Веб-приложение по продаже музыкального оборудования.

Диаграмма классов.

5.4 Веб-приложение по продаже музыкального оборудования.

Диаграмма последовательности.

5.5 Веб-приложение по продаже музыкального оборудования.

Модель данных.

5.6 Заключительный плакат. Плакат.

6 Содержание задания по экономической части: «Экономическое обоснование разработки и реализации программного модуля веб-приложения по продаже музыкального оборудования».

ЗАДАНИЕ ВЫДАЛ

О.А. Матяс

КАЛЕНДАРНЫЙ ПЛАН

Наименование этапов дипломного проекта	Объем этапа, %	Срок выполнения этапа	Примечания
Подбор и изучение литературы. Сравнение аналогов. Уточнение задания на ДП	10	23.03 – 30.03	
Структурное проектирование	15	30.03 – 08.04	
Функциональное проектирование	25	08.04 – 24.04	
Разработка программных модулей	20	24.04 – 8.05	
Программа и методика испытаний	10	8.05 – 15.05	
Расчет экономической эффективности	5	15.05 – 20.05	
Оформление пояснительной записки	15	20.05 – 30.05	

Дата выдачи задания: 23.03.2022

Руководитель

В.В. Старовойтов

ЗАДАНИЕ ПРИНЯЛ К ИСПОЛНЕНИЮ

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	7
1 ОБЗОР ЛИТЕРАТУРЫ.....	8
1.1 Обзор аналогов.....	8
1.1.1 Thomann.de.....	8
1.1.2 Guitarland.by.....	9
1.2 Обзор технологий.....	10
1.2.1 Ruby.....	11
1.2.2 JavaScript.....	12
1.2.3 Ruby on Rails.....	12
1.2.4 Шаблон проектирования MVC.....	13
1.2.5 Архитектура клиент-сервер.....	14
1.2.6 База данных.....	15
1.3 Постановка задачи.....	16
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....	17
2.1 Блок базы данных.....	17
2.2 Блок авторизации пользователя.....	18
2.3 Блок администрирования.....	19
2.4 Блок пользовательского интерфейса.....	19
2.5 Блок взаимодействия с корзиной.....	19
2.6 Блок веб-сервера.....	19
2.7 Блоки управления и взаимодействия с позициями.....	21
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	22
3.1 Описание модели данных.....	22
3.2 Описание структуры и взаимодействия между классами.....	28
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ.....	45
4.1 Алгоритм выбора бренда.....	45
4.2 Алгоритм оформления заказа.....	46
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ.....	51
5.1 Юнит-тестирование.....	54
5.2 Сквозное тестирование (end-to-end).....	56
5.3 Тестирование пользовательского интерфейса.....	56
6.1 Требования к аппаратному обеспечению.....	62
6.2 Руководство по развертыванию приложения.....	62
6.3 Руководство по использованию ПО.....	64
6.3.1 Главная страница.....	64
6.3.2 Страница корзины.....	65
6.3.3 Руководство по входу пользователя в систему.....	66
6.3.4 Страница оформления заказа.....	67
6.3.5 Страница администратора.....	69
6.3.6 Страница редактирования пользователей.....	69
6.3.7 Страница редактирования позиций.....	71
6.3.8 Страница редактирования заказов.....	73

7 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО МОДУЛЯ ВЕБ-ПРИЛОЖЕНИЯ ПО ПРОДАЖЕ МУЗЫКАЛЬНОГО ОБОРУДОВАНИЯ.....	75
7.1 Характеристика разработанного программного средства	75
7.2 Расчет цены программного модуля веб-приложения по продаже музыкального оборудования на основе затрат.....	75
7.2.1 Затраты на основную заработную плату команды разработчиков	76
7.2.2 Затраты на дополнительную заработную плату команды	77
7.2.3 Отчисления на социальные нужды	77
7.2.4 Прочие расходы.....	78
7.2.5 Общая сумма затрат на разработку	78
7.2.6 Плановая прибыль, включаемая в цену программного средства.....	79
7.2.7 Отпускная цена программного средства	79
7.3 Расчет результата от разработки и реализации программного модуля веб-приложения по продаже музыкального оборудования.....	80
7.4 Расчет показателей экономической эффективности разработки программного модуля веб-приложения по продаже музыкального оборудования	81
ЗАКЛЮЧЕНИЕ	82
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	83
ПРИЛОЖЕНИЕ А	84
ПРИЛОЖЕНИЕ Б.....	102
ПРИЛОЖЕНИЕ В	103

ВВЕДЕНИЕ

В последние годы наметилась тенденция к тому, что люди все больше и больше совершают покупки в онлайн магазинах. Миллионы транзакций ежеминутно совершают люди покупая товары онлайн. Онлайн магазины предоставляют нам доступ к широкому ассортименту товаров в различных формах и размерах, что значительно экономит время, которое человек проводит в очередях при поиске и подборе товара. Немаловажным преимуществом интернет-магазинов является возможность доставки приобретенного товара, что крайне актуально в нынешней непростой эпидемиологической ситуации, так как исключает надобность посещения человеком места торговли товаром. Также онлайн магазины позволяют экономить средства производителя за ненадобностью содержания физического магазина.

На данный момент в ряде крупных стран интернет-торговля занимает 20% товарооборота. Из вышеприведенных фактов следует, что данная сфера пользуется популярностью и будет в дальнейшем расти, и, следовательно, создание нового сервиса для онлайн-продаж является целесообразным. Интернет-аудитория увеличивается каждый день, ведущие сервисы по продаже товаров совершенствуются, однако это не является препятствием для создания нового сервиса. При подробном анализе рынка существующих сервисов, нетрудно определить преимущества и недостатки ведущих онлайн-магазинов. Наиболее важной проблемой при создании онлайн магазина является сохранность данных кредитных карт. Уровень безопасности транзакций крайне важен для всех сторон продажи товара. В целях достижения максимального уровня безопасности транзакций и хранения данных о платежных картах покупателей, разработчик должен предусмотреть множество уязвимостей в программном обеспечении интернет-магазина.

Задачей дипломного проекта является разработка веб-приложения по продаже музыкального оборудования. Для выполнения поставленной задачи необходимо произвести обзор аналогов и, на основе анализа и корректно выстроенного плана разработки, создать конкурентноспособный продукт. Для создания работоспособного продукта потребуются уделить внимание вопросу разработки клиент-серверных приложений и выполнить проектирование проекта.

В соответствии с поставленной целью были определены следующие задачи:

1. Выбор платформы создания системы.
2. Разработка пользовательского интерфейса.
3. Разработка протокола взаимодействия клиентского интерфейса с серверной частью программного модуля.
4. Тестирование программного модуля.
5. Расчет экономических затрат на создание проекта.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

В данном подразделе рассмотрим существующие аналоги реализуемого программного обеспечения.

1.1.1 Thomann.de

Thomann.de [1] (см. рисунок 1.1) – немецкий специализированный сайт по продаже музыкального оборудования. Главная страница представлена удобным интерфейсом с приятным оформлением, без лишнего нагромождения.

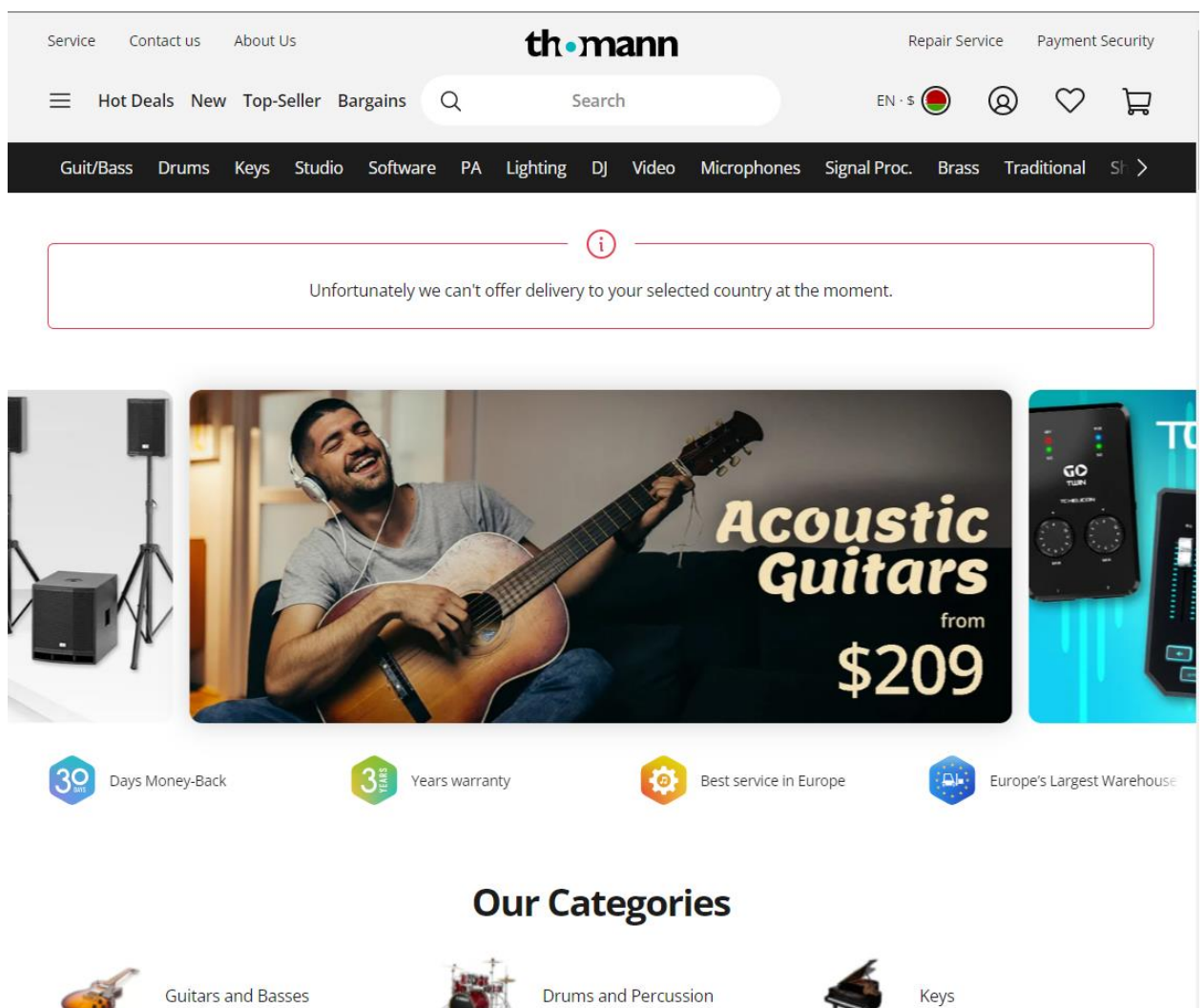


Рисунок 1.1 – Главная страница Thomann.de [1]

На сайте отсутствует явная реклама. Однако на сайте присутствует неуместный раздел – раздел новостей. В случае если пользователь захочет

узнать новости по тематике определенного звукового оборудования, то он воспользуется специализированным сайтом. В данном случае функционал раздела новостей является избыточным. На данном сайте можно найти большое количество разнообразной музыкальной техники: от бас-гитар до световых установок. Существенным недостатком является отсутствие локализации для русского языка, что может принести некотором пользователям дискомфорт при пользовании сайтом и оформлении заказов.

Резюмируя вышеперечисленное, можно выделить явные достоинства и недостатки.

Достоинства:

- широкий ассортимент товара;
- практичный интерфейс;
- удобная адаптация для мобильных устройств.

Недостатки:

- отсутствие русской локализации;
- невозможность в данный момент доставки в Республику Беларусь.

1.1.2 Guitarland.by

Guitarland.by [2] (см. рисунок 1.2) – сайт магазина музыкальных инструментов. Деятельность данного сайта направлена на более узкий спектр музыкального оборудования.

Здесь в ассортименте представлены инструменты для живого исполнения и записи. Однако кроме нужных разделов на сайте присутствует достаточное количество лишних разделов. Таковыми являются новости и события, которые можно прочесть на новостных сайтах. Вторым таким разделом является статьи Guitarland, который было бы целесообразнее вести в социальных сетях магазина, а на самом сайте размещать ссылки на страницы в социальных сетях. Существенным недостатком данного сайта является отсутствие прямой связи с продавцом напрямую через сайт. Также сайт не оптимизирован для использования в мобильной версии, что ухудшает удобство просмотра сайта.

Исходя из предложенных ранее фактов, резюмируем достоинства и недостатки данного ресурса.

Достоинства:

- узконаправленность (удобство для клиента, который знает чего хочет);
- наличие сезонных скидок на большое количество товаров;
- торговое помещение в центре Минска.

Недостатки:

- отсутствие адаптации для мобильных устройств;
- присутствие большого количества лишней информации на странице сайта.



Рисунок 1.2 – Главная страница Guitarland.by [2]

1.2 Обзор технологий

Разрабатываемая в рамках данного дипломного проекта программная система является веб-приложением. Под веб-приложением понимается клиент-серверное приложение, в котором роль клиента играет браузер, а сервером является веб-сервер, который может храниться в облачном хранилище. Логика веб-приложения распределена между сервером и клиентом, хранение данных осуществляется, преимущественно, на сервере, обмен информацией происходит по сети. Благодаря такому подходу клиент может не зависеть от конкретной операционной системы. Из этого следует, что веб-приложения являются кроссплатформенными сервисами.

Разрабатываемая программная система имеет монолитную архитектуру. Концепция монолитного программного обеспечения подразумевает, что различные компоненты приложения объединяются в одну программу на одной платформе. Обычно монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения. Все функции такого приложения управляются в одном месте.

Для разработки серверной части приложения использовался паттерн MVC. MVC представляет из себя архитектурный шаблон, который означает, что приложение MVC будет разделено на три части, а именно:

- модель;
- представление;
- контроллер.

Модель представляет из себя данные, передаваемые между представлениями и контроллерами, а также модель может содержать операции и преобразования для манипулирования этими данными.

Представления применяются для визуализации части модели в виде пользовательского интерфейса.

Контроллеры обрабатывают поступающие запросы, выполняют операции с моделью и выбирают представления для визуализации пользователю.

В большинстве случаев для разработки веб-сервера используются следующие языки программирования: Java, JavaScript, Python, PHP, C#, Ruby.

1.2.1 Ruby

Для написания веб-сервера данного дипломного проекта используется язык программирования Ruby [3]. Ruby – это интерпретируемый, высокоуровневый, динамичный, универсальный язык программирования с открытым исходным кодом, который фокусируется на простоте и производительности. Он был спроектирован и разработан в середине 1990-х годов Юкиhiro Мацумото в Японии.

Ruby – это язык динамического программирования со сложной, но выразительной грамматикой и базовой библиотекой классов с богатым и мощным API. Ruby - выбрал в себя черты таких языков, как Lisp, Smalltalk и Perl, но использует синтаксис, которым без особого труда смогут овладеть программисты, работающие на языках C и Java.

Ruby является абсолютным объектно-ориентированным языком, но в нем также неплохо уживаются процедурные и функциональные стили программирования. Приоритетом Ruby является удобство и минимизация затрат труда программиста при разработке программы, освобождение программиста от рутинной работы, которую компьютер может выполнять быстрее и качественнее. Особое внимание, в частности, уделено будничным рутинным занятиям (обработка текстов, администрирование), и для них язык настроен особенно хорошо. В противоположность машинно-

ориентированным языкам, работающим быстрее, Ruby – язык, наиболее понятный человеку, даже очень далекому от программирования. Любая работа с компьютером выполняется людьми и для людей, и необходимо заботиться в первую очередь о затрачиваемых усилиях людей [4].

1.2.2 JavaScript

Для разработки пользовательского интерфейса используется язык программирования JavaScript и библиотека jQuery.

JavaScript работает на клиентской стороне приложения и используется для определения того, как веб-страницы ведут себя при возникновении события [5]. JavaScript является простым в изучении, широко используемым, а также мощным языком для управления поведением веб-страниц.

jQuery – легковесная библиотека JavaScript, слоганом которой является “пиши меньше, делай больше”. Цель jQuery - сделать использование JavaScript на проекте намного проще. jQuery берет много общих задач, для выполнения которых требуется много строк JavaScript-кода, и обортывает их в методы, которые можно вызвать одной строкой. jQuery также упрощает многие сложные вещи из JavaScript, такие как вызовы AJAX и манипуляции с DOM элементами [6].

1.2.3 Ruby on Rails

Ruby on Rails – фреймворк, написанный на языке программирования Ruby с открытым исходным кодом, реализует архитектурный шаблон Model-View-Controller для веб-приложений [7]. Ruby on Rails включает в себя инструменты, которые облегчают общие задачи разработки «из коробки», такие как scaffolding, которые могут автоматически генерировать некоторые модели и представления, необходимые для базового сайта. В конфигурации по умолчанию модель отображает одну из таблицы в базе данных. Например, класс модели User обычно определяется в файле «user.rb» в каталоге app/models и привязывается к таблице «users» в базе данных.

Контроллер – это компонент Rails, который отвечает на внешние запросы от веб-сервера к приложению, определяя, какой файл отображения нужно отрисовать. Контроллеру также может потребоваться запросить одну или несколько моделей для получения информации и передать их в отображение.

Ruby on Rails также заслуживает внимания за широкое использование библиотек JavaScript для написания Ajax-запросов. Для этого прекрасно подойдет jQuery, который полностью поддерживается как замена Prototype и поддерживается по умолчанию в Rails с версии 3.1.

Основным преимуществом языка программирования Ruby и фреймворка Ruby on Rails является скорость разработки. На практике скорость разработки проектов на Ruby on Rails выше на 30-40 процентов по отношению

к любому другому языку программирования или фреймворку. Такой прирост к скорости разработки объясняется обширным набором готовых к работе штатных инструментов Ruby on Rails, возможностью использовать готовые решения других разработчиков и удобством программирования на Ruby.

Кроме того, в отличие от других фреймворков, в составе Ruby on Rails есть отличные средства автоматизированного тестирования, что ускоряет переход проекта от стадии «программа написана» к стадии «программа работает без ошибок». Зачастую именно этот переход отнимает больше всего времени при реализации практически любого проекта.

1.2.4 Шаблон проектирования MVC

Шаблон проектирования MVC [8] (см. рисунок 1.3) – это не столько шаблон, сколько метод или идея организации приложения, его структуры. Сама идея заключается в том, чтобы отделить бизнес логику приложения от его представления. Как следует из названия MVC (Model – View – Controller) состоит из трех основных компонентов: модель, представление и контроллер.

При этом модификация каждого из трех модулей происходит независимо. Данный подход описывает один из принципов SOLID – принцип единой ответственности. Это значит, что каждый объект (модуль) имеет одну ответственность и все поведение объекта должно быть направлено на выполнение только этой одной ответственности.

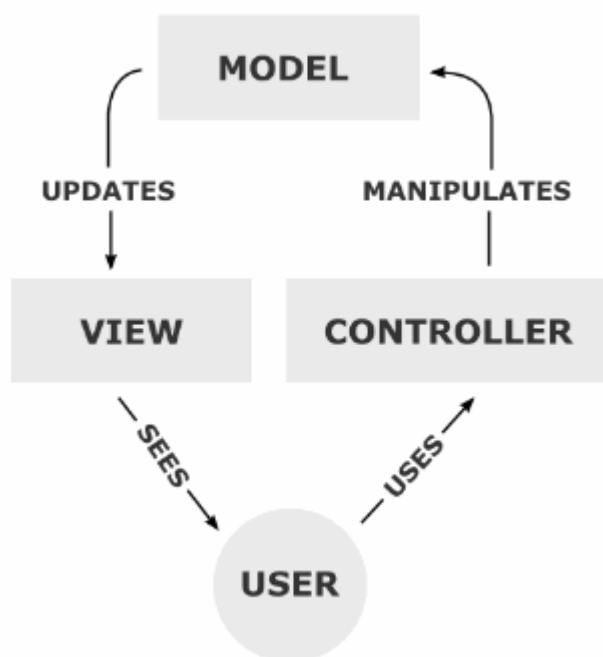


Рисунок 1.3 – Шаблон MVC [8]

Модель в шаблоне MVC – основная часть логики, которая отвечает за выборку данных из базы, изменение данных, расчеты. Вообще модель можно

представить, как набор каких-либо функций, некий ящик, который принимает в себя данные, обрабатывает их и возвращает обратно. Данные в модель приходят из контроллера, затем эти данные обрабатываются, будь то запросы к базе данных, либо же бизнес вычисления. Данные, пришедшие из контроллера, могут сохраняться в базе данных. После обработки данных моделью, результат работы возвращается обратно в контроллер. В больших приложениях, как правило, имеется много моделей, каждая из которых отвечает за конкретный модуль, будь то создание пользователей или вывод списка записей в таблицу. В модели определяется ограничения по обрабатываемым данным и правила манипуляции данными.

Представление отвечает за вывод данных для пользователя. Обычно представление содержит в себе мало логики и предназначено только для отображения результата работы модели. В современных сайтах большое внимание уделяется клиентской части приложения. Поэтому скрипты в представлениях порой достигают очень больших размеров, что замедляет работу приложения. Представление реагирует на действия пользователя и передает данные в контроллер. Обычно представление реализуется посредством шаблонов HTML-разметки, которые заполняются данными. Данные приходят из модели, контроллер заполняет представление и шаблон HTML документа отправляется клиенту.

Контроллер – блок в шаблоне MVC, который получает данные, нормализует их, производит валидацию данных, и после прохождения этих процедур передает данные дальше в нужную модель. С другой стороны, контроллер получает данные из модели, и выбирая нужное представление, отображает их. Контроллер не должен содержать в себе никакой бизнес логики. Вся основная бизнес-логика должна содержаться в слое модели. Контроллер выступает связующим звеном между моделью и представлением.

1.2.5 Архитектура клиент-сервер

Обычно компьютеры, входящие в какую-либо систему, не являются равноправными. Так же, как и в социальной группе есть лидер и подчиненные. Некоторые компьютеры несут в себе информацию, а другие же пользуются этой информацией посредством запросов к главному компьютеру. Сервером называется компьютер, который хранит в себе информацию, и который обрабатывает полученные от клиента запросы, а затем выдает ответ. Как правило, на сервере установлено специальное программное обеспечение для выполнения какой-либо сервисной задачи без участия человека. В свою очередь веб-сервером называют как программное обеспечение, которое выполняет функции сервера, так и сам компьютер, на котором это программное обеспечение работает. Клиентом же называют компьютеры, которые посылают запросы на сервер и получают ответ. Клиент и сервер могут находиться как в рамках одной сети, так и в рамках различных сетей. Предполагается, что приложение запущено и работает на сервере.

Суть клиент-серверной архитектуры заключается в разделении приложения на три группы:

- отображение и ввод данных для пользователя;
- функции связывающие отображения и бизнес-логику;
- функции управления ресурсами (работа с базой данных).

Из этого следует, что любое приложение можно разделить на три компонента:

- компонент отображения;
- прикладной компонент;
- компонент управления ресурсами.

Отправка данных от клиента серверу осуществляется с помощью приложения, называемого браузером. Браузер не имеет привязки к операционной системе, поэтому веб-приложения являются кроссплатформенными. Браузер в свою очередь интерпретирует HTML документ своим образом и отображается пользователю. Чтобы браузер мог отображать данные для пользователя, серверу необходимо посылать ответ в формате HTML.

Веб-сервер отвечает за обработку данных, полученных от клиента. Веб-сервер получает данные от клиента, активизируя действие по определенному пути, передает данные в логические модули программы. После этого приложение обрабатывает полученный запрос и веб-сервер формирует HTML страницу, которую отправляет назад клиенту. Сервер содержит ряд шаблонов HTML, на которые накладываются данные и отсылаются клиенту.

Отличительной особенностью веб клиент-серверной архитектуры является то, что в роли клиента выступает браузер, который посылает запросы на сервер, посредством HTTP/HTTPS протоколов. Веб-сервер принимает запросы от клиента, обрабатывает данные, и посылает ответ клиенту. А взаимодействие между клиентом и сервером осуществляется через интернет соединение.

1.2.6 База данных

В качестве системы управления базами данных будет использоваться PostgreSQL.

PostgreSQL — свободная объектно-реляционная система управления базами данных (СУБД). PostgreSQL является одной из наиболее популярных систем управления базами данных [9]. Сам проект postgresql эволюционировал из другого проекта, который назывался Ingres. Формально развитие postgresql началось еще в 1986 году. Тогда он назывался POSTGRES. А в 1996 году проект был переименован в PostgreSQL, что отражало больший акцент на SQL. И 8 июля 1996 года состоялся первый релиз продукта. С тех пор вышло множество версий postgresql. Текущей версией является версия 14.2. Однако регулярно также выходят подверсии. PostgreSQL поддерживается для всех основных операционных систем – Windows, Linux, MacOS.

Особенности PostgreSQL:

- высокая производительность;
- надежность;
- гибкость и простота в использовании;
- многоверсионность;
- репликация;
- масштабируемость.

1.3 Постановка задачи

Сегодня рынок программного обеспечения выставляет высокие требования ко всем разрабатываемым программным продуктам. Для современных программных средств важными требованиями являются мультиплатформенность, масштабируемость и переносимость. Использование вышеперечисленных технологий при разработке дипломного проекта позволяет сократить время на разработку, увеличить качество кода и засчет этого выполнить данные требования

Объектом исследования является процесс создания объектов и обработки данных, поступающих на сервер. Предметом исследования является создание программного комплекса, обеспечивающего полноценную работу веб-приложения.

Основными требованиями, которые были заложены в основу при разработке программного комплекса дипломного проекта стали: простота использования и расширяемость.

На основании вышесказанного, для разработки программного продукта были определены следующие задачи:

- разработка структуры БД;
- разработка серверной части;
- разработка клиентской части.

Веб-приложение будет представлять из себя сайт, который будет предоставлять следующие функции:

- регистрация пользователя в системе;
- изменение данных пользователя;
- создания, редактирование и удаление объявления о продаже;
- отправка письма на электронную почту о получении заказа;
- администрирование пользователей, товаров.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Проанализировав теоретическую часть разрабатываемой системы, был получен список требований, который необходим для эффективного, стабильного функционирования разрабатываемого веб-приложения. В ходе анализа требований было принято решение разделить программный продукт на функциональные блоки. Распределение функционала по блокам обеспечит удобство в разработке, а также предоставит возможность проверки основных характеристик блока без внешнего воздействия. Выделенные функциональные блоки:

- блок базы данных;
- блок авторизации пользователя;
- блок взаимодействия с корзиной;
- блок работы сервера;
- блок управления позициями;
- блок взаимодействия с позициями;
- блок администрирования;
- блок пользовательского интерфейса.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.078 С1.

Рассмотрим каждый структурный блок подробнее.

2.1 Блок базы данных

Блок базы данных включает данные, используемые веб-приложением. При реализации используется реляционная база данных PostgreSQL. От других СУБД PostgreSQL отличается поддержкой востребованного объектно-ориентированного и/или реляционного подхода к базам данных. Например, полная поддержка надежных транзакций, т.е. атомарность, последовательность, изоляционность, прочность (Atomicity, Consistency, Isolation, Durability (ACID)). Существует обширный список типов данных, которые поддерживает PostgreSQL. Кроме числовых, с плавающей точкой, текстовых, булевых и других ожидаемых типов данных (а также множества их вариаций), PostgreSQL имеет преимущество в качестве поддержки uuid, денежного, перечисляемого, геометрического, бинарного типов, сетевых адресов, битовых строк, текстового поиска, xml, json, массивов, композитных типов и диапазонов, а также некоторых внутренних типов для идентификации объектов и местоположения логов.

Основными достоинствами PostgreSQL являются:

- надежность (полное соответствие принципам ACID - атомарность, непротиворечивость, изолированность, сохранность данных);
- производительность (основывается на использовании индексов, интеллектуальном планировщике запросов, тонкой системы блокировок,

системе управления буферами памяти и кэширования, превосходной масштабируемости при конкурентной работе);

- расширяемость (означает, что пользователь может настраивать систему путем определения новых функций, агрегатов, типов, языков, индексов и операторов);

- поддержка SQL;

- поддержка JSON;

- богатый набор типов данных;

- простота использования.

2.2 Блок авторизации пользователя

Блок авторизации пользователя является частью системы, которая отвечает за проверку существования пользователя и в случае его существования в системе, генерирует авторизационный токен для этого пользователя. Для данных целей будет использоваться популярная в Rails среде библиотека Devise. Данная библиотека состоит из следующих модулей:

- Database Authenticatable – хэширует и сохраняет пароль в базе данных для проверки подлинности пользователя при входе в систему;

- Confirmable – отправляет электронные письма с инструкциями по подтверждению аккаунта и проверяет подтверждена учетная запись или нет;

- Recoverable – при необходимости сбрасывает пароль и отправляет инструкции по восстановлению;

- Registerable – предоставляет механизм регистрации нового пользователя;

- Trackable – запоминает количество входов, время и IP-адрес;

- Timeoutable – закрывает открытые сессии пользователя если они не были активны заданный период времени;

- Validatable – проверяет при входе корректность введенных данных так же можно определить свои валидации;

- Lockable – блокирует аккаунт после определенного количества неудачных попыток входа. Может быть разблокирован по электронной почте или по истечении указанного периода времени;

- Rememberable – добавляет возможность «запомнить аккаунт»;

- Omniauthable – поддержка авторизации через социальные сети.

В разрабатываемом программном продукте будут использоваться только некоторые из этих модулей, а именно: Database Authenticatable, Recoverable, Rememberable, Validatable, Registerable. Так же присутствуют методы, упрощающие работу с пользователями, которые будут подробно описаны в следующих главах.

2.3 Блок администрирования

Блок администрирования – это часть системы, отвечающая за управление всеми ресурсами сайта. В Rails для этого есть несколько решений, в данном проекте используется плагин Rolify. Rolify предоставляет возможность назначать зарегистрированным пользователям роли, с помощью которых каждому отдельному пользователю предоставляются доступные для данной роли действия. Для роли admin в пользовательском интерфейсе открываются дополнительные опции для доступа ко всем ресурсам приложения, а также появляется возможность для проведения любых манипуляций с данными. Кроме того, стоит отметить, что Rolify гибко настраивается в зависимости от потребностей проекта.

2.4 Блок пользовательского интерфейса

Блок пользовательского интерфейса является клиентской частью веб-приложения. Данный блок представляет собой совокупность средств, при помощи которых пользователь взаимодействует с приложением через браузер. Для построения интерфейса будет использоваться несколько технологий: JavaScript, JQuery. Также в Rails есть расширение HTML файлов «.erb», которое позволяет использовать Ruby-код непосредственно в HTML-разметке и предоставляет множество заготовленных хелперов для отрисовки элементов. JQuery – используется для Ajax-запросов на сервер. Ajax – это технология, которая позволяет обновлять данные не перезагружая страницу полностью.

2.5 Блок взаимодействия с корзиной

Блок взаимодействия с корзиной – это блок, основной задачей которого является хранение и помещение в него товаров пользователем. В дальнейшем из продуктов, которые хранятся в корзине, формируется заказ. Для реализации данного блока используется Redis. Redis – это быстрое хранилище данных типа «ключ-значение». Оно используется как для баз данных, так и для реализации кэша из-за быстрой скорости обработки операций (хранит базу данных в оперативной памяти).

2.6 Блок веб-сервера

Блок веб-сервера – это блок, который обрабатывает запросы, отправленные клиентом, в качестве серверной части будет использоваться фреймворк Ruby On Rails.

Фреймворк Ruby On Rails написан с помощью языка программирования Ruby.

Ruby – интерпретируемый язык программирования высокого уровня. Язык динамический, объектно-ориентированный, реализует

многопоточность, которая не зависит от операционной системы, имеет «сборщик мусора».

Ruby on Rails предоставляет из себя архитектуру MVC для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером базы данных. Ruby on Rails определяет следующие принципы разработки приложений, помогающие разработчику в создании элегантных программных решений, усвоенные сообществом разработчиков:

- предоставляет механизмы повторного использования, позволяющие минимизировать дублирование кода в приложениях (принцип Don't repeat your self);

- по умолчанию используются соглашения по конфигурации, типичные для большинства приложений (принцип Convention over configuration);

- основными компонентами приложений Ruby on Rails являются модель, представление и контроллер;

- Ruby on Rails использует REST-стиль построения веб-приложений.

Модель используется для отображения данных остальным компонентам приложения. Объекты модели производят загрузку и сохранение данных в базе данных, а также реализуют бизнес-логику.

Для хранения объектов модели в СУБД по умолчанию в Rails использована библиотека Active Record.

Представление создает пользовательский интерфейс с использованием полученных от контроллера данных. Представление также передает запросы пользователя на взаимодействие с данными в контроллер (само представление обычно не изменяет модель). Контроллер в Rails – это набор логики, запускаемой после получения HTTP-запроса сервером. Контроллер отвечает за вызов методов модели и запускает формирование представления.

Вокруг Rails сложилась большая экосистема плагинов – подключаемых «гемов» (англ. gem). «Гем» – это библиотека, составленная определенным образом. То есть это набор кода (модули, классы, представления и так далее) которые решают определенную задачу. Утилита gem занимается тем, что управляет этими библиотеками. Например 'gem install colorize' скачает из интернета библиотеку (далее "гем") в каталог, в который прежде был установлен Ruby. После чего в коде можно написать require 'colorize' и пользоваться методами, которые данный гем предоставляет. Гем может требовать для установки другие геммы. Для того чтобы не ставить/обновлять геммы по-одному каждый раз, был создан Bundler, который сам по себе является гемом. Работает это следующим образом: в Gemfile описываются требуемые в данном конкретном проекте геммы. После чего запускается команда bundle install. Bundler просматривает Gemfile проекта и устанавливает (с помощью утилиты gem) нужные геммы, а также создает файл Gemfile.lock, в котором описывает установленный гем и, если понадобились при установке, то дополнения к данному гемму.

2.7 Блоки управления и взаимодействия с позициями

Блоки управления объявлениями и взаимодействия с позициями призваны отвечать за работу с позициями. Через эти блоки проводятся базовые CRUD-операции с позициями. Аббревиатура CRUD в переводе с английского означает C(create) – создать, R(read) – читать, U(update) – обновлять, D(delete) – удалять. Это основные операции, используемые для реализации приложений с постоянным хранением данных и приложений реляционных баз данных.

CRUD постоянно используется для всего, что связано с базами данных и их проектированием. Разработчики программного обеспечения ничего не могут сделать без операций CRUD. Например, при разработке веб-сайтов используется REST (передача репрезентативного состояния), который является надмножеством CRUD, используемого для ресурсов HTTP.

С другой стороны, CRUD не менее важен для конечных пользователей. Без него были бы невозможны такие базовые операции как регистрация пользователя, размещение новых записей. Большинство приложений, которые мы используем, позволяют нам добавлять или создавать новые записи, искать существующие, вносить в них изменения или удалять их.

Create позволяет вносить новые строки в таблицу базы данных. Сделать это можно с помощью команды INSERT INTO. Команда начинается с ключевого слова INSERT INTO, за которым следует имя таблицы, имена столбцов и значения, которые нужно вставить.

Функция чтения соизмерима с функцией поиска, поскольку позволяет извлекать определенные записи и считывать их значения. Она относится к команде SELECT.

Обновление – это изменение существующей записи в таблице. Это используется для внесения изменений в существующие записи в таблице базы данных. При выполнении команды UPDATE определяется целевая таблица и столбцы, которые необходимо обновить.

Удаление используется для удаления записи из таблицы базы данных. SQL имеет встроенную функцию удаления для одновременного удаления одной или нескольких записей из базы данных. Некоторые приложения реляционных баз данных могут разрешать жесткое удаление (безвозвратное удаление) или мягкое удаление (обновление статуса строки).

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описывается функционирование и структура разрабатываемого программного средства.

Взаимоотношения между классами разрабатываемого программного обеспечения приведены на диаграмме классов ГУИР.400201.078 РР.1.

3.1 Описание модели данных

3.1.1 Таблица Users

Данная таблица служит для хранения основных данных об аккаунте пользователя. Поля таблицы представлены в таблице 3.1.

Таблица 3.1 – Описание полей таблицы users

Поле	Описание
id	Первичный ключ
name	Имя пользователя в приложении
email	Электронная почта пользователя
phone_number	Номер телефона пользователя в приложении
reset_password_token	Зашешированный пароль, который был выслан пользователю для восстановления аккаунта
reset_password_sent_at	Время и дата отправления пароля для восстановления аккаунта
created_at	Дата и время создания аккаунта
updated_at	Дата и время последних изменений аккаунта
encrypted_password	Захешированный пароль пользователя

При создании таблицы для колонок id, encrypted_password, email, created_at, updated_at был использован специальный атрибут «null: false», который задает условие создания и заполнения данных колонок так, что они не могут быть пустыми. В случае если данным колонкам не будет явно задано значение, будет выставлено значение по умолчанию. Для колонок encrypted_password, email это значение пустой строки. Поля id, created_at, updated_at будут автоматически заполнены Ruby on Rails. У различных колонок могут быть разные типы данных. У колонок name,

encrypted_password, email, reset_password_token тип данных «character varying». Этот тип данных является символьной строкой переменной длины. Колонки id, phone_number имеют тип «integer». Колонки reset_password_sent_at, created_at, updated_at имеют тип timestamp.

3.1.2 Таблица Roles

Данная таблица служит для хранения данных о созданных в системе ролях для пользователей, которые разграничивают доступный пользователям функционал. Поля таблицы представлены в таблице 3.2

Таблица 3.2 – Описание полей таблицы roles

Поле	Описание
id	Первичный ключ
name	Название роли в приложении
created_at	Дата и время создания роли
resource_type	Составная часть сложного индекса
resource_id	Составная часть сложного индекса
updated_at	Дата и время последних изменений роли

Колонки id, created_at, updated_at не могут быть пустыми. Колонка resource_id имеет тип «integer». Колонка name имеет тип «character varying» и заполняется при создании новой роли. Колонка id имеет тип «integer». Колонки created_at, updated_at имеют тип timestamp и заполняются автоматически Ruby on Rails.

3.1.3 Таблица UserRoles

Данная таблица является связующей таблицей между таблицами Users и Roles. В ней фиксируется принадлежность пользователя к некой роли, а, соответственно, и наличие у пользователя каких-либо предпочтений при использовании приложением. Поля таблицы представлены в таблице 3.3

Таблица 3.3 – Описание полей таблицы `users_roles`

Поле	Описание
<code>user_id</code>	Внешний ключ для связи с таблицей <code>users</code>
<code>role_id</code>	Внешний ключ для связи с таблицей <code>roles</code>

Колонки `user_id`, `role_id` имеют тип «integer».

3.1.4 Таблица `Versions`

Данная таблица служит для хранения версий пользователей при их изменениях. Поля таблицы представлены в таблице 3.4.

Таблица 3.4 – Описание полей таблицы `versions`

Поле	Описание
<code>id</code>	Первичный ключ
<code>item_type</code>	Составная часть сложного индекса
<code>item_id</code>	Составная часть сложного индекса
<code>event</code>	Событие изменения
<code>whodunnit</code>	Имя пользователя, который изменил состояние предыдущей версии
<code>object</code>	Объект изменения
<code>created_at</code>	Дата и время создания объекта

При создании таблицы для колонок `id`, `item_type`, `item_id`, `event` был использован специальный атрибут «`null: false`», который задает условие создания и заполнения данных колонок так, что они не могут быть пустыми. Для колонок `item_type`, `event`, в случае если они не будут заполнены, то они будут заполнены по умолчанию пустой строкой. Поля `id`, `created_at` будут автоматически заполнены Ruby on Rails. У колонок `item_type`, `event`, `whodunnit` тип данных «character varying». Колонки `id`, `item_id` имеют тип «integer». Колонка `created_at` имеет тип `timestamp`. Колонка `object` имеет тип `text`.

3.1.5 Таблица Products

Данная таблица хранит информацию о музыкальном оборудовании, которое выставлено на продажу и содержит его описание. Поля таблицы представлены в таблице 3.5.

Таблица 3.5 – Описание полей таблицы products

Поле	Описание
id	Первичный ключ
name	Название позиции в приложении
photo	Изображение позиции
price	Цена данной позиции оборудования
created_at	Дата и время создания позиции
updated_at	Дата и время последних изменений позиции
description	Описание данной позиции оборудования
brand_id	Внешний ключ для связи с таблицей Brands
category_id	Внешний ключ для связи с таблицей Categories

При создании таблицы для колонок id, created_at, updated_at, brand_id, category_id использован специальный атрибут «null: false», который задает условие создания и заполнения данных колонок так, что они не могут быть пустыми. Поля id, created_at, updated_at будут автоматически заполнены Ruby on Rails. Поле name имеет тип данных «character varying». Поле description имеет тип text. Поле price имеет тип «integer». Поле photo имеет тип «character varying». В колонке photo будут храниться ссылки на загруженные пользователем изображения. Поля created_at, updated_at имеют тип timestamp. Колонки id, brand_id, category_id имеют тип «integer».

3.1.6 Таблица Orders

Данная таблица тесно связана с таблицей users и хранит в себе информацию о созданном пользователем заказе позиций музыкального оборудования. Поля таблицы представлены в таблице 3.6.

Таблица 3.6 – Описание полей таблицы orders

Поле	Описание
id	Первичный ключ
name	Имя человека, оформляющего заказ позиций
order_price	Полная стоимость заказа
user_id	Внешний ключ для связи с таблицей Users
dest_address	Адрес, куда нужно доставить заказанные позиции
phone_number	Номер телефона заказчика
created_at	Дата и время создания заказа
updated_at	Дата и время последних изменений заказа

Колонки id, user_id, order_price, phone_number имеют тип «integer». Колонки dest_address, name имеют тип «character varying». Колонки created_at, updated_at имеют тип timestamp.

3.1.7 Таблица OrderProducts

Данная таблица является связующей таблицей между таблицами Orders и Products. В ней фиксируется принадлежность заказу позиций, которые пользователь добавил себе в корзину и оформил данный заказ. Поля таблицы представлены в таблице 3.7.

Таблица 3.7 – Описание полей таблицы order_products

Поле	Описание
id	Первичный ключ
product_id	Внешний ключ для связи с таблицей products
order_id	Внешний ключ для связи с таблицей orders
created_at	Дата и время создания связи заказа и позиции
updated_at	Дата и время последних изменений связи заказа и позиции

Колонки `id`, `order_id`, `product_id` имеют тип «integer». Колонки `created_at`, `updated_at` имеют тип `timestamp`.

3.1.8 Таблица **Brands**

Данная таблица служит для хранения данных о созданных в системе брендах товаров. Поля таблицы представлены в таблице 3.8.

Таблица 3.8 – Описание полей таблицы `brands`

Поле	Описание
<code>id</code>	Первичный ключ
<code>brand_name</code>	Название бренда в приложении
<code>created_at</code>	Дата и время создания бренда
<code>updated_at</code>	Дата и время последних изменений бренда

Колонки `id`, `created_at`, `updated_at` не могут быть пустыми. Колонка `brand_name` имеет тип «character varying» и заполняется при создании нового бренда. Колонка `id` имеет тип «integer». Колонки `created_at`, `updated_at` имеют тип `timestamp` и заполняются автоматически Ruby on Rails.

3.1.9 Таблица **Categories**

Данная таблица служит для хранения данных о созданных в системе категориях товаров. Поля таблицы представлены в таблице 3.9.

Таблица 3.9 – Описание полей таблицы `categories`

Поле	Описание
<code>id</code>	Первичный ключ
<code>category_name</code>	Название категории в приложении
<code>created_at</code>	Дата и время создания категории
<code>updated_at</code>	Дата и время последних изменений категории

Колонки `id`, `created_at`, `updated_at` не могут быть пустыми. Колонка `category_name` имеет тип «character varying» и заполняется при создании новой категории. Колонка `id` имеет тип «integer». Колонки

`created_at`, `updated_at` имеют тип `timestamp` и заполняется автоматически Ruby on Rails.

3.1.10 Таблица **BrandCategories**

Данная таблица является связующей таблицей между таблицами `Brands` и `Categories`. В ней фиксируется принадлежность определенных категорий товаров определенным брендам и наоборот: какие бренды содержат какие категории товаров. Поля таблицы представлены в таблице 3.10.

Таблица 3.10 – Описание полей таблицы `brand_categories`

Поле	Описание
<code>id</code>	Первичный ключ
<code>brand_id</code>	Внешний ключ для связи с таблицей <code>Brands</code>
<code>category_id</code>	Внешний ключ для связи с таблицей <code>Categories</code>
<code>created_at</code>	Дата и время создания связи бренда и категории
<code>updated_at</code>	Дата и время последних изменений связи бренда и категории

Колонки `id`, `brand_id`, `category_id` имеют тип «integer». Колонки `created_at`, `updated_at` имеют тип `timestamp`.

3.2 Описание структуры и взаимодействия между классами

При разработке приложения использовался паттерн MVC, поэтому оно имеет структуру, состоящую из контроллера, сервиса и репозитория. Также стоит отметить, что все проектируемые сервисы разработаны с соблюдением всех правил и норм REST архитектуры. Рассмотрим классы каждого из данных слоев приложения.

3.2.1 Класс **ApplicationController**

Данный класс-контроллер является главным контроллером приложения, от которого наследуются все остальные контроллеры приложения. `ApplicationController` в свою очередь наследуется от `ActionController::Base`, это базовый модуль Rails приложения, который предоставляет большое количество методов для работы с контроллерами. Методы класса описаны в таблице 3.11.

Таблица 3.11 – Описание методов класса ApplicationController

Метод	Описание
configure_permitted_params	protected метод экземпляра, добавляет параметры к объекту пользователя
current_user?	protected метод экземпляра, проверяет по идентификатору является ли данный пользователь текущим пользователем
token	protected метод экземпляра, записывает значение сессионного токена
authorize	protected метод экземпляра, выполняет перенаправление на страницу с авторизацией, если пользователь не является текущим пользователем
authorize?	protected метод экземпляра, возвращает токен, если пользователь не является текущим пользователем
basket	protected метод экземпляра, создает новую корзину для текущей сессии
redis_basket	protected метод экземпляра, создает новую redis-корзину с помощью текущей записи в redis
after_sign_in_path_for	protected метод экземпляра, возвращает корневой путь
after_sign_out_path_for	protected метод экземпляра, возвращает пользователя на предыдущую страницу

3.2.2 Класс ApplicationController

Данный класс-контроллер предназначен для авторизации пользователей. Методы класса описаны в таблице 3.12.

Таблица 3.12 – Описание методов класса AuthenticationController

Метод	Описание
new	Публичный метод экземпляра класса, используется для отображения шаблона, в котором отображаются все поля, которые необходимо заполнить при создании нового пользователя, после ввода отправляет информацию, которую ввел пользователь, для дальнейшей обработки в метод create
create	Публичный метод экземпляра класса, используется для создания пользователя, после получения данным методом информации от пользователя, он ее обрабатывает, проверяет соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их пользователю с описанием ошибки
destroy	Публичный метод экземпляра класса, используется для удаления пользователя
user_params	Приватный метод экземпляра класса, используется для отделения параметров о пользователе от других параметров, которые приходят в запросе

3.2.3 Класс BasketsController

Данный класс-контроллер предназначен для управления корзиной. Методы класса описаны в таблице 3.13.

Таблица 3.13 – Описание методов класса BasketsController

Метод	Описание
1	2
index	Публичный метод экземпляра класса, используется для получения всех позиций, находящихся в корзине и отображения их пользователю
add	Публичный метод экземпляра класса, используется для добавления новой корзины
remove	Публичный метод экземпляра класса, используется для удаления корзины

Продолжение таблицы 3.13

1	2
clear	Публичный метод экземпляра класса, используется для очищения корзины

3.2.4 Класс CheckoutController

Данный класс-контроллер предназначен для управления оформлением заказов. Методы класса представлены в таблице 3.14.

Таблица 3.14 – Описание методов класса CheckoutController

Метод	Описание
new	Публичный метод экземпляра класса, используется для отображения шаблона, в котором отображаются все поля, которые необходимо заполнить при создании нового заказа, после ввода отправляет информацию, которую ввел пользователь, для дальнейшей обработки в метод create
create	Публичный метод экземпляра класса, используется для создания заказа, после получения данным методом информации от пользователя, он ее обрабатывает, проверят соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их пользователю с описанием ошибки
order_params	Приватный метод экземпляра класса, используется для отделения параметров о заказе от других параметров, которые приходят в запросе

3.2.5 Модуль CheckoutsService

Данный модуль используется для создания заказов и включается в CheckoutController. Методы модуля представлены в таблице 3.15.

Таблица 3.15 – Описание методов модуля CheckoutService

Метод	Описание
initialize	Публичный метод модуля, используется для инициализации переменных класса в момент создания объекта
make_order	Публичный метод модуля, используется для построения структуры заказа
build_form_params	Приватный метод модуля, используется для установления параметров заказа
make_order_product	Приватный метод модуля, используется для записи собранных в корзине позиций в заказ

3.2.6 Модуль BasketsService

Данный модуль используется для управления корзиной, созданной с помощью Redis, и включается в BasketsController. Данный модуль используется в ApplicationController, ProductsController. Методы модуля представлены в таблице 3.16.

Таблица 3.16 – Описание методов модуля BasketsService

Метод	Описание
initialize	Публичный метод модуля, предназначен для инициализации переменных класса в момент создания объекта
add	Публичный метод модуля, предназначен для добавления позиции в корзину
remove	Публичный метод модуля, предназначен для удаления позиции из корзины
all	Публичный метод модуля, предназначен для отображения всего содержимого корзины, если таковая имеется
clear	Публичный метод модуля, предназначен для удаления корзины
size	Публичный метод модуля, предназначен для подсчета количества позиций, которые на данный момент находятся в корзине
sum	Публичный метод модуля, предназначен для подсчета суммы позиций, находящихся в данный момент в корзине
token	Приватный метод модуля, записывает значение сессионного токена

3.2.7 Класс ProductsController

Данный класс-контроллер используется для управления позициями, размещенными в приложении. Методы класса представлены в таблице 3.17.

Таблица 3.17 – Описание методов класса ProductsController

Метод	Описание
index	Публичный метод экземпляра класса, используется для получения всех позиций, находящихся в приложении и отображения их пользователю
show	Публичный метод экземпляра класса, используется для получения одной позиции и отображения ее пользователю, поиск данной позиции производится по идентификатору позиции
add_to_basket	Публичный метод экземпляра класса, используется для добавления позиции в корзину
delete_from_basket	Публичный метод экземпляра класса, используется для удаления позиции из корзины
product_params	Приватный метод экземпляра класса, используется для отделения параметров о позиции от других параметров, которые приходят в запросе

3.2.8 Класс SessionsController

Данный класс-контроллер используется для создания и удаления информации о текущей сессии, используя методы, встроенные в родительский класс SessionsController, встроенный в гем Devise. Методы класса представлены в таблице 3.18.

Таблица 3.18 – Описание методов класса SessionsController

Метод	Описание
1	2
create	Публичный метод экземпляра класса, предназначен для создания сессии, вызывая метод из родительского класса Devise::SessionsController, и переназначения корзины на созданный токен сессии

Продолжение таблицы 3.8

1	2
destroy	Публичный метод экземпляра класса, предназначен для переназначения корзины на другой созданный токен сессии и удаления корзины, вызывая метод из родительского класса <code>Devise::SessionsController</code>

3.2.9 Класс AdminsController

Данный класс-контроллер используется для авторизации пользователей с ролью admin. Он является главным в модуле Admin, который наследуется от ApplicationController, и отвечает за работу администрации в приложении. Методы класса представлены в таблице 3.19.

Таблица 3.19 – Описание методов класса AdminsController

Метод	Описание
admin?	Публичный метод экземпляра класса, предназначен для проверки является ли текущий пользователь администратором
authorize_admin	Публичный метод экземпляра класса, выполняет перенаправление на главную страницу приложения, если пользователь не является администратором
authenticate_user!	Коллбэк, который вызывается перед всеми публичными методами класса и используется для проверки аутентификации пользователя

3.2.10 Класс OrdersController

Данный класс-контроллер используется для того, чтобы администраторы имели возможность управлять заказами. Он является частью модуля Admin. Методы класса представлены в таблице 3.20.

Таблица 3.20 – Описание методов класса OrdersController

Метод	Описание
1	2
index	Публичный метод экземпляра класса, используется для получения всех заказов, находящихся в приложении и отображения их администратору

Продолжение таблицы 3.20

1	2
show	Публичный метод экземпляра класса, используется для получения заказа и отображения его администратору, поиск данного заказа производится по идентификатору заказа
new	Публичный метод экземпляра класса, используется для отображения шаблона, в котором отображаются все поля, которые необходимо заполнить при создании нового заказа, после ввода отправляет информацию, которую ввел пользователь для дальнейшей обработки в метод create
create	Публичный метод экземпляра класса, используется для создания заказа, после получения данным методом информации от пользователя, он ее обрабатывает, проверят соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их пользователю с описанием ошибки
edit	Публичный метод экземпляра класса, используется для изменения уже существующего заказа, отображает все поля заказа на шаблоне, которые администратор может изменить и после ввода передает методу update для обработки
update	Публичный метод экземпляра класса, используется для изменения существующего заказа, после получения данным методом информации от пользователя, он ее обрабатывает, проверят соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их пользователю с описанием ошибки

Окончание таблицы 3.20

1	2
destroy	Публичный метод экземпляра класса, используется для удаления существующего заказа. Заказ для удаления находится по идентификатору, передаваемому в параметрах при запросе
order_params	Приватный метод экземпляра класса, используется для отделения параметров о заказе от других параметров, которые приходят в запросе
make_products_array	Приватный метод экземпляра класса, используется для создания массива из позиций
authorize_admin	Коллбэк, который вызывается перед всеми публичными методами класса и используется для проверки авторизации администратора

3.2.11 Класс ProductsController

Данный класс-контроллер используется для того, чтобы администраторы имели возможность управлять позициями. Он является частью модуля Admin. Методы класса представлены в таблице 3.21.

Таблица 3.21 – Описание методов класса ProductsController

Метод	Описание
1	2
index	Публичный метод экземпляра класса, используется для получения всех позиций, находящихся в приложении и отображения их администратору
show	Публичный метод экземпляра класса, используется для получения позиции и отображения ее администратору, поиск данной позиции производится по идентификатору позиции

Продолжение таблицы 3.21

1	2
new	публичный метод экземпляра класса, используется для отображения шаблона, в котором отображаются все поля, которые необходимо заполнить при создании новой позиции, после ввода отправляет информацию, которую ввел администратор для дальнейшей обработки в метод create
create	Публичный метод экземпляра класса, используется для создания позиции, после получения данным методом информации от администратора, он ее обрабатывает, проверяет соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их администратору с описанием ошибки
edit	Публичный метод экземпляра класса, используется для изменения уже существующей позиции, отображает все поля позиции на шаблоне, которые администратор может изменить и после ввода передает методу update для обработки
update	Публичный метод экземпляра класса, используется для изменения существующей позиции, после получения данным методом информации от администратора, он ее обрабатывает, проверяет соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их администратору с описанием ошибки
destroy	Публичный метод экземпляра класса, используется для удаления существующей позиции. Позиция для удаления находится по идентификатору, передаваемому в параметрах при запросе
add_to_basket	Публичный метод экземпляра класса, используется для добавления позиции в корзину

Окончание таблицы 3.21

1	2
delete_from_basket	Публичный метод экземпляра класса, используется для удаления позиции из корзины
product_params	Приватный метод экземпляра класса, используется для отделения параметров о позиции от других параметров, которые приходят в запросе
authorize_admin	Коллбэк, который вызывается перед всеми публичными методами класса и используется для проверки авторизации администратора

3.2.12 Класс RolesController

Данный класс-контроллер используется для того, чтобы администраторы имели возможность управлять ролями. Он является частью модуля Admin. Методы класса представлены в таблице 3.22.

Таблица 3.22 – Описание методов класса RolesController

Метод	Описание
1	2
index	Публичный метод экземпляра класса, используется для получения всех ролей, находящихся в приложении и отображения их администратору
show	Публичный метод экземпляра класса, используется для получения роли и отображения ее администратору, поиск данной роли производится по идентификатору роли
new	Публичный метод экземпляра класса, используется для отображения шаблона, в котором отображаются все поля, которые необходимо заполнить при создании новой позиции, после ввода отправляет информацию, которую ввел администратор для дальнейшей обработки в метод create

Продолжение таблицы 3.22

1	2
create	Публичный метод экземпляра класса, используется для создания роли, после получения данным методом информации от администратора, он ее обрабатывает, проверяет соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их администратору с описанием ошибки
edit	Публичный метод экземпляра класса, используется для изменения уже существующей роли, отображает все поля роли на шаблоне, которые администратор может изменить и после ввода передает методу update для обработки
update	Публичный метод экземпляра класса, используется для изменения существующей роли, после получения данным методом информации от администратора, он ее обрабатывает, проверяет соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их администратору с описанием ошибки
destroy	Публичный метод экземпляра класса, используется для удаления существующей роли. Роль для удаления находится по идентификатору, передаваемому в параметрах при запросе
role_params	Приватный метод экземпляра класса, используется для отделения параметров о роли от других параметров, которые приходят в запросе
authorize_admin	Коллбэк, который вызывается перед всеми публичными методами класса и используется для проверки авторизации администратора

3.2.13 Класс UsersController

Данный класс-контроллер используется для того, чтобы администраторы имели возможность управлять пользователями. Он является частью модуля Admin. Методы класса представлены в таблице 3.23.

Таблица 3.23 – Описание методов класса UsersController

Метод	Описание
1	2
index	Публичный метод экземпляра класса, используется для получения всех пользователей, находящихся в приложении и отображения их администратору
show	Публичный метод экземпляра класса, используется для получения пользователя и отображения его администратору, поиск данного пользователя производится по идентификатору пользователя
new	Публичный метод экземпляра класса, используется для отображения шаблона, в котором отображаются все поля, которые необходимо заполнить при создании нового пользователя, после ввода отправляет информацию, которую ввел администратор для дальнейшей обработки в метод create
create	Публичный метод экземпляра класса, используется для создания пользователя, после получения данным методом информации от администратора, он ее обрабатывает, проверяет соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их администратору с описанием ошибки
edit	Публичный метод экземпляра класса, используется для изменения уже существующего пользователя, отображает все поля пользователя на шаблоне, которые администратор может изменить и после ввода передает методу update для обработки

Продолжение таблицы 3.23

1	2
update	Публичный метод экземпляра класса, используется для изменения существующего пользователя, после получения данным методом информации от администратора, он ее обрабатывает, проверяет соответствие валидаторам, заданным в модели, и, если все проверки пройдены, то сохраняет информацию, если есть какие-то ошибки возвращает их администратору с описанием ошибки
destroy	Публичный метод экземпляра класса, используется для удаления существующего пользователя. Пользователь для удаления находится по идентификатору, передаваемому в параметрах при запросе
user_params	Приватный метод экземпляра класса, используется для отделения параметров о пользователе от других параметров, которые приходят в запросе
authorize_admin	Коллбэк, который вызывается перед всеми публичными методами класса и используется для проверки авторизации администратора

3.2.14 Класс User

Данный класс является объектным отображением таблицы `users`. Данный класс используется для управления данными о пользователях в аккаунте. Данный класс связан с классом `Order`. Это реализовано с помощью связи один ко многим и определено в классе с помощью метода `has_many :orders`.

3.2.15 Класс Order

Данный класс является объектным отображением таблицы `orders`. Данный класс используется для управления данными о заказах. Он связан с классами `OrderProduct`, `Product`, `User`. Связь с классом `User` реализована с помощью связи один ко многим и определена в классе с помощью метода `belongs_to :user`. Связь с классом `Product` реализована с помощью связи многие-ко-многим и определена в классе с помощью метода `has_many :products` и с помощью промежуточной

таблицы `order_product`, связь с которой осуществляется с помощью метода `has_many :order_products`.

3.2.16 Класс `OrderProduct`

Данный класс является объектным отображением таблицы `order_products`. Данный класс используется для хранения данных о связи заказов и позиций. Данная таблица является промежуточной в связи многие-ко-многим между классами `Product` и `Order` и связана с ними методами `belongs_to :product` и `belongs_to :order` соответственно.

3.2.17 Класс `Product`

Данный класс является объектным отображением таблицы `products`. Данный класс используется для управления данными о позициях. Данный класс связан с классом `Order`. Связь реализована с помощью связи многие-ко-многим и определена в классе с помощью метода `has_many :orders` и с помощью промежуточной таблицы `order_product`, связь с которой осуществляется с помощью метода `has_many :order_products`. Также, данный класс связан с классами `Brand` и `Category`. Связи реализованы с помощью связи один ко многим и с помощью методов `belongs_to :brand` и `belongs_to :category`. Данный класс содержит метод `mount_uploader`, который указывает, какой загрузчик файлов будет использоваться в классе. В данном случае, используется `PhotoUploader`, что отображено в методе `mount_uploader :photo`.

3.2.18 Класс `Brand`

Данный класс является объектным отображением таблицы `brands`. Данный класс используется для управления данными о брендах. Он связан с классами `BrandCategory`, `Category`, `Product`. Связь с классом `Product` реализована с помощью связи один ко многим и определена в классе с помощью метода `has_many :products`. Связь с классом `Category` реализована с помощью связи многие-ко-многим и определена в классе с помощью метода `has_many :categories` и с помощью промежуточной таблицы `brand_categories`, связь с которой осуществляется с помощью метода `has_many :brand_categories`.

3.2.19 Класс `Category`

Данный класс является объектным отображением таблицы `categories`. Данный класс используется для управления данными о

категориях. Он связан с классами `BrandCategory`, `Brand`, `Product`. Связь с классом `Product` реализована с помощью связи один ко многим и определена в классе с помощью метода `has_many :products`. Связь с классом `Brand` реализована с помощью связи многие-ко-многим и определена в классе с помощью метода `has_many :brands` и с помощью промежуточной таблицы `brand_categories`, связь с которой осуществляется с помощью метода `has_many :brand_categories`.

3.2.20 Класс `BrandCategory`

Данный класс является объектным отображением таблицы `brand_categories`. Данный класс используется для хранения данных о связи брендов и категорий товаров. Данная таблица является промежуточной в связи многие-ко-многим между классами `Brand` и `Category` и связана с ними методами `belongs_to :brand` и `belongs_to :category` соответственно.

3.2.21 Класс `Role`

Данный класс является объектным отображением таблицы `roles`. Данный класс используется для управления данными о ролях. Данный класс связан с классом `User`. Связь реализована с помощью связи многие-ко-многим и определена в классе с помощью метода `has_and_belongs_to_many :users`, через промежуточную таблицу `users_roles`.

3.2.22 Класс `PhotoUploader`

Данный класс используется для загрузки изображений позиций товаров, используя методы, встроенные в родительский класс `CarrierWave::Uploader::Base`, встроенный в гем `CarrierWave`.

Методы класса:

- `store_dir` – публичный метод экземпляра класса, предназначен для установки папки хранения загруженных файлов изображений;
- `filename` – публичный метод экземпляра класса, предназначен для распознавания и задания имени загруженного файла изображения;
- `secure_token()` – `protected` метод экземпляра класса, предназначен для генерации токена безопасности.

3.2.23 Класс `ApplicationMailer`

Данный класс является стандартным классом отправки электронных писем в приложении. Он наследует методы от родительского класса `ActionMailer::Base`.

3.2.24 Класс `OrderReportMailer`

Данный класс предназначен для отправки электронного письма пользователю с уведомлением об оформлении заказа. Методы класса представлены в таблице 3.24.

Таблица 3.24 – Описание методов класса `OrderReportMailer`

Метод	Описание
<code>report</code>	Публичный метод экземпляра класса, предназначен для установления параметров адресата и отправки ему сообщения с уведомлением

3.2.25 Класс `EmailSenderWorker`

Данный класс-воркер предназначен для отправки электронного письма пользователю с уведомлением об оформлении заказа. Методы класса представлены в таблице 3.25.

Таблица 3.25 – Описание методов класса `EmailSenderWorker`

Метод	Описание
<code>perform(user_id, order_id)</code>	Публичный метод экземпляра класса, предназначен для установления параметров адресата в <code>OrderReportMailer</code> , помещения задачи по отправки электронного письма в асинхронную очередь и добавления и отправки ему сообщения с уведомлением

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

Листинг программы представлен в Приложении А.

4.1 Алгоритм выбора бренда

Данный алгоритм служит для создания и заполнения селектора выбора бренда товара. В этом алгоритме особого внимания заслуживает технология AJAX, которая является способом организации клиент-серверного взаимодействия. При помощи этой технологии происходит асинхронное взаимодействие сервера с клиентом. Выбор бренда доступен на страницах создания и редактирования товара. Необходимо выбрать категорию товара. Это действие сгенерирует запрос на AJAX-запрос на клиенте и отправит его на сервер.

```
document.addEventListener("turbolinks:load", function() {
    jQuery(function() {
        return $('#subcategory_select').change(function() {
            const brandSelect = $('#brand_select')[0];
            const categoryId =
$('#product_category_id')[0].value;
            $.ajax({
                type:"GET",
                url:"/categories/" + categoryId + "/brands",
                dataType:"json",
                success: function(response) {
                    const ajaxVariable = response.brands;
                    const oldSelector =
$('#category_select')[0];
                    if (oldSelector) {
                        oldSelector.remove();
                    }
                    const selectList =
document.createElement("select");
                    selectList.setAttribute("id",
"category_select");
                    selectList.setAttribute("name",
"product[brand_id]");
                    brandSelect.appendChild(selectList);
                    $.each(ajaxVariable, function(index, value)
{
                        $('#category_select')
                            .append("<option></option>")
                            .attr("value", value[0])
                            .text(value[1]));
                    });
                }
            });
        });
    });
});
```

```
});  
});
```

Данный запрос будет направлен по адресу `/categories/:category_id/brands` и соответствовать методу `index` контроллера `BrandsController`.

```
def index  
  @brands = Brand.joins(brand_categories:  
:category).where(categories: { id: params[:category_id]  
}).pluck(:id, :brand_name)  
  render json: { brands: @brands }  
end
```

Разберем данный метод подробнее. Первой строкой в переменную экземпляра `@brands` записываем с помощью запроса в базу данных массив, состоящий из массивов идентификаторов и названий брендов, которые содержит данная категория товара. Данный массив получается с помощью переданного через `json`-запрос идентификатора категории товаров `id: params[:category_id]`. Если категория с данным идентификатором не содержит никаких товаров, то будет передан пустой массив, а селектор не будет содержать в себе ничего. Далее отправляем на клиент полученный массив в формате `json`. После получения ответа клиент отобразит в селекторе `Brands` полученные из данной категории бренды. В случае отсутствия в данной категории брендов, селектор `Brands` будет пустым.

4.2 Алгоритм оформления заказа

После заполнения пользователем корзины и нажатием кнопки «CHECKOUT» происходит формирование корзины. В первую очередь вызывается контроллер `CheckoutController`, который начинает формирование заказа.

```
class CheckoutController < ApplicationController  
  
  def new  
    @products = @basket.all  
    @order = Order.new  
    @user = current_user || User.new  
  end  
  
  def create  
    @order = ::CheckoutsService.new(session,  
order_params).make_order  
    @basket.clear  
    redirect_to '/'  
  end  
end
```

```

private

def order_params
  params.require(:order).permit(:user_id, :order_price,
:dest_address, :name, :phone_number)
end
end

```

Рассмотрим данный класс подробнее. В первую очередь работает метод `new`

```

def new
  @products = @basket.all
  @order = Order.new
  @user = current_user || User.new
end

```

Данный метод создает и инициализирует переменные экземпляра для дальнейшего их использования. Переменная `@products` заполняется товарами из корзины, переменная `@order` инициализируется как новая переменная класса `Order`, в переменную `User` записывается информация о текущем пользователе, а при его отсутствии – происходит создание нового пользователя.

Следующим отработывает метод `create`.

```

def create
  @order = ::CheckoutsService.new(session,
order_params).make_order
  @basket.clear
  redirect_to '/'
end

```

Данный метод заполняет переменную экземпляра `@order`, в которой вызывается сервис создания заказа `CheckoutsService`, который инициализируется с помощью данных о текущей сессии и параметрами заказа, которые представлены в приватном методе экземпляра класса `order_params`.

```

private

def order_params
  params.require(:order).permit(:user_id, :order_price,
:dest_address, :name, :phone_number)
end

```

Далее переходим к обзору сервиса CheckoutService. Первым вызывается метод initialize при вызове сервиса CheckoutService в контроллере CheckoutController.

```
def initialize(session, order_params)
  @session = session
  @order_params = order_params
  @key = if session['warden.user.user.key']
          session['warden.user.user.key'][0][0]
        else
          session[:token]
        end
end
```

Данный метод инициализирует переменные, переданные при вызове сервиса при вызове сервиса CheckoutService в контроллере CheckoutController. В переменную @session записываются данные о текущей сессии, переменная @order_params получает параметры о заказе, в переменную @key записывается значение пользовательского ключа или сессионный токен.

Следующим выполняется метод make_order.

```
def make_order
  @basket = ::BasketsService.new(@session).all

  build_form_params
  order = Order.new(@order_params)
  return order.errors unless order.valid?

  order.save
  make_order_product

  ::EmailSenderWorker.perform_async(@session['warden.user.user.key']
  '[0][0]', @order.id) if @session['warden.user.user.key'][0][0]
end
```

Данный метод является основным в сервисе CheckoutService. В переменную @basket записываются данные о существующей в данной сессии корзине. Далее с помощью приватного метода экземпляра класса build_form_params устанавливаются параметры заказа, а именно: с помощью сервиса BasketsService полученной стоимости товаров из корзины и умножением на 1,2 с учетом НДС, а также устанавливается параметр идентификатора пользователя user_id с помощью значения ключа сессии @key.


```
private
```

```
def build_form_params
  @order_params[:order_price] =
  ::BasketsService.new(@session).sum * 1.2
  @order_params[:user_id] = @key
end
```

Далее в локальную переменную `order` заносятся данные созданного заказа, а затем заказ сохраняется при условии, что заказ валиден. В ином случае выводятся ошибки о создании заказа. Далее при помощи приватного метода экземпляра класса `make_order_product` составляется заказ из продуктов, содержащихся в корзине.

```
private
```

```
def make_order_product(order)
  @basket.map { |product| OrderProduct.create!(product_id:
product['id'], order_id: order.id) }
end
```

После составления заказа выполняется воркер `EmailSenderWorker`, который помещает задачу по отправке электронного письма в асинхронную очередь, и с чьей помощью происходит отправка письма о том, что заказ был принят.

```
class EmailSenderWorker
  include Sidekiq::Worker

  def perform(user_id, order_id)
    OrderReportMailer.with(user: User.find(user_id), order:
Order.find(order_id)).report.deliver_now
  end
end
```

Воркер `EmailSenderWorker` выполняет функцию `perform`, которая, передавая параметры `user_id` и `order_id` в `OrderReportMailer`, выполняет отправку электронного письма пользователю, который совершил данный заказ.

```
class OrderReportMailer < ApplicationMailer

  def report
    @user = params.fetch(:user)
    @order = params[:order]

    mail(to: @user.email, subject: 'Your order is still
accepted!')
```

end

end

Класс `OrderReportMailer` выполняет непосредственно отправку электронного письма адресату с помощью метода `report`. Параметры, полученные из метода `perform` воркера `EmailSenderWorker`, присваиваются непосредственно переменным экземпляра `@user` и `@order`. Далее с помощью функции `mail`, которую класс получил с помощью наследования от встроенного в Rails `ApplicationMailer`, совершает отправку электронного письма по электронному адресу пользователя, совершившего данный заказ. Результат получения письма об оформлении заказа представлен на рисунке 4.1.

Your order is still accepted!



valeronp.com@gmail.com Сегодня, 2:18

Кому: вам

Valery Petrikevich's order

Your order is accepted. Please, wait!

Order:

Name	Phone number	Destination Address	Total price
Valery Petrikevich	7981076	31, Kuntsevshchina Str, 220017, Minsk, Belarus	720 BYN

Рисунок 4.1 – Результат получения письма об оформлении заказа

5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Большинство создаваемых в данное время Rails-приложений разрабатывается с применением TDD (test-driven development) методологии. Данная методология разработки программного обеспечения предполагает повторение коротких циклов разработки: в первую очередь пишется тест, который покрывает изменение, а затем пишется код, который пройдет тест. Ближе к концу разработки проводится рефакторинг нового кода для приведения его к соответствующим стандартам. Такой подход является полной противоположностью разработке, при которой сначала разрабатывается программное обеспечение, а затем описываются тестовые ситуации.

Тест – это операция, с помощью которой определяется работоспособен ли данный код. При проверке написанного кода программист выполняет тестирование функционала вручную. В данной ситуации тестирование состоит из двух стадий: стимулирование кода и проверка выполнения его функционала. Автоматическое тестирование выполняется по-другому: вместо программиста стимулированием кода и проверкой его выполнения занимается компьютер, на дисплее которого отображается результат прохождения теста: код работоспособен или код неработоспособен. Таким образом происходит «инверсия ответственности»: от реализации тестов и их детальности зависит, будет ли код соответствовать техническому заданию. Методика разработки через тестирование заключается главным образом в реализации автоматических тестов.

Разработка через тестирование требует от разработчика написания автоматизированных модульных тестов, определяющих требования к коду непосредственно перед написанием самого кода. Тест содержит проверки условий, которые могут либо выполняться, либо нет. Когда условия выполняются, говорят, что тест пройден. Прохождение теста подтверждает поведение, предполагаемое программистом. Разработчики часто пользуются библиотеками для тестирования для создания и автоматизации запуска наборов тестов. На практике модульные тесты покрывают критические и нетривиальные участки кода. Это может быть код, который подвержен частым изменениям, код, от работы которого зависит работоспособность большого количества другого кода, или код с большим количеством зависимостей.

Среда разработки должна быстро реагировать на небольшие модификации кода. Архитектура программы должна базироваться на использовании множества сильно связанных компонентов, которые слабо соединены друг с другом, благодаря чему тестирование кода упрощается.

TDD не только предполагает проверку исправности работы, но и влияет на дизайн программы. Опираясь на тесты, разработчики могут подробнее представить, какая функциональность необходима пользователю. Таким образом, детали интерфейса появляются задолго до окончательной реализации решения.

К тестам применяются те же требования стандартов кодирования, что и к основному коду.

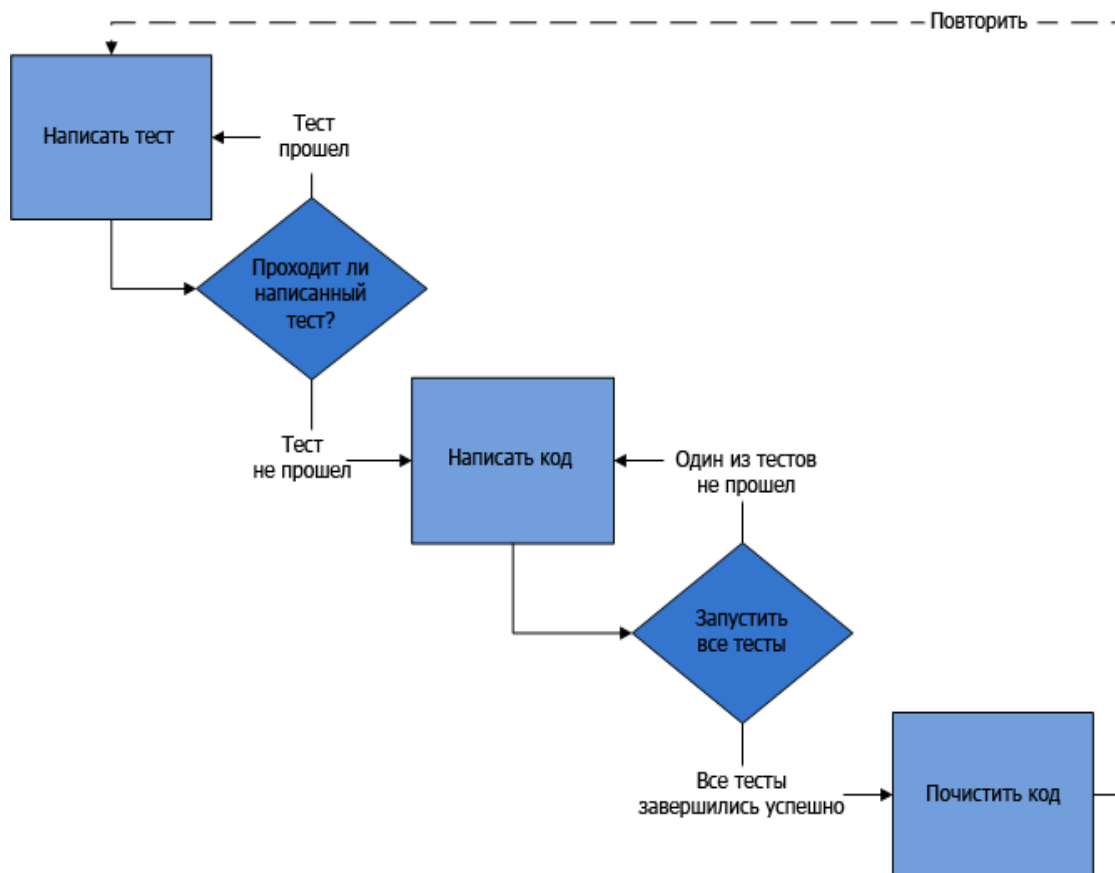


Рисунок 5.1 – Разработка с использованием TDD

Современные проекты предъявляют высокие требования к покрытию автоматическими тестами. Написание тестов является одним из главных требований при разработке и написании кода программного обеспечения в наше время. Все чаще мы слышим такие аббревиатуры, как TDD (Test Driven Development) и BDD (Behaviour Driven Development) и многие строго следуют этим подходам в разработке. BDD это одна из разновидностей TDD, и именно такая парадигма используется в системе RSpec.

RSpec – это фреймворк для тестирования написанный на языке программирования Ruby и предоставляющий специальный DSL (domain-specific programming language) для написания тестов – спецификаций. RSpec – инструмент для BDD. Спецификация (spec) – это обычно отдельный файл, который содержит описание какой-нибудь части программы, в контексте Rails, это может быть описание целого контроллера, модели, шаблона, партиала, хелпера и т.д. Файлы спецификаций принято хранить в поддиректории spec проекта, а имена файлов должны заканчиваться на `_spec.rb`.

Часто случаются ситуации, когда добавление нового функционала ведет к изменению поведения старого, и в такой ситуации разработчик может об этом даже не узнать. Такая ситуация имеет свое название – регрессия.

Регрессионное тестирование – это виды тестирования программного обеспечения, позволяющие обнаружить ошибки в уже протестированном ранее функционале. Регрессионное тестирование выполняется не для того, чтобы убедиться в отсутствии ошибок в имеющемся функционале, а для исправления регрессионных ошибок (ошибки, появляющиеся не при написании программы, а при добавлении в исходный код нового функционала или исправление ошибок, что и стало причиной возникновения новых ошибок в уже протестированной программе). Цикл регрессионного тестирования представлен на рисунке 5.2.



Рисунок 5.2 – Цикл регрессионного тестирования

Методы регрессионного тестирования включают в себя многократное выполнение предыдущих тестов и проверку того, что добавление нового функционала не вызвало создание новых регрессионных ошибок.

Регрессионное тестирование включает в себя три этапа:

- верификация устранения нового дефекта;
- верификация того, что дефект, который был проверен и исправлен ранее, не воспроизводится вновь;
- верификация того, что функциональность приложения не нарушилась после исправления дефектов и внесения нового кода.

Выделяют несколько видов регрессионных тестов:

- верификационные тесты (проводятся для проверки исправления обнаруженной ранее ошибки);

- тестирование верификации версии (проверка работоспособности основной функциональности программы).

При написании исходного кода веб-приложения тестирование проводилась в три этапа:

- тестирование отдельно каждого сервиса в процессе написания программного кода;
- тестирование взаимодействия нескольких сервисов между собой;
- полное тестирование программы после окончания процесса написания программного кода.

Данные этапы являются важными и связанными между собой, ни один из них невозможно исключить. Например, без модульного тестирования, при анализе работы программы в целом, будет происходить достаточное количество сбоев, выявить которые может оказаться достаточно сложным, в то время как при анализе работы одного модуля неисправность оказывается достаточно очевидной. И обратный случай, работоспособность каждого компонента в отдельности не гарантирует корректное поведение всей программы в целом.

При написании веб-приложения проводилось юнит-тестирование. Отдельно необходимо упомянуть сквозное тестирование.

5.1 Юнит-тестирование

Для тестирования сервисов серверной части использовалось юнит-тестирование. Под юнит-тестированием подразумевается процесс, позволяющий проверить на корректность отдельные модули исходного кода. Основная цель юнит-тестирования заключается в том, чтобы писать тесты для каждой функции или метода и проверять как позитивные, так и негативные сценарии. Это позволяет быстро проверить, не привело ли добавление нового функционала к регрессии. Тесты, предназначенные для проверки функционала сервисов, представлены в таблице 5.1.

Таблица 5.1 – Тестирование сервисов веб-приложения

Содержание теста	Фактический результат
1	2
Ввод в форму входа данных без указания электронного адреса пользователя и вызов метода отправки формы	Форма не отправлена на сервер, выведена ошибка валидации
Ввод в форму входа данных без указания пароля пользователя и вызов метода отправки формы	Форма не отправлена на сервер, выведена ошибка валидации

Продолжение таблицы 5.1

1	2
Ввод в форму входа валидных данных пользователя во все поля и вызов метода отправки формы	Форма валидна и инициирован вызов метода для отправки формы
Вызов метода для обработки успешного входа пользователя	Инициирован вызов метода для сохранения пользовательских данных
Создание новой позиции товара	Создана новая сущность товара в базе данных
Создание аккаунта пользователя без введения адреса электронной почты	Возвращена ошибка невозможности создания аккаунта без введения адреса электронной почты
Создание аккаунта пользователя без введения пароля пользователя	Возвращена ошибка невозможности создания аккаунта без введения пароля пользователя
Создание аккаунта пользователя без подтверждения пароля пользователя	Возвращена ошибка невозможности создания аккаунта без подтверждения пароля пользователя
Попытка создания аккаунта со введенными во все поля валидными данными	Запись о новом аккаунте внесена в базу данных и возвращен пользователь
Ввод в форму входа данных пользователя, записи о котором не существует в базе данных, и вызов метода отправки формы	Возвращена ошибка невозможности входа в данный аккаунт
Покупка пользователем позиции товара	Позиция добавлена в корзину пользователя
Удаление пользователем позиции товара из корзины	Позиция удалена из корзины пользователя
Очистка пользователем корзины	Все позиции, которые были добавлены пользователем в корзину, были удалены
Получение пользователем списка всех предложенных позиций	Возвращен список всех занесенных в базу данных записей о добавленных в каталог веб-приложения позиций
Попытка регистрации нового пользователя с данными уже существующего пользователя в базе данных	Возвращена ошибка о том, что пользователь с такими данными уже зарегистрирован в системе
Попытка редактирования данных существующего пользователя администратором	Данные о пользователе в базе данных изменены

Окончание таблицы 5.1

1	2
Попытка удаления администратором учетной записи существующего пользователя из базы данных	Учетная запись пользователя удалена из базы данных
Попытка создания администратором новой позиции продукта	Запись о новой позиции внесена в базу данных и произошел переход на главную страницу веб-приложения
Попытка редактирования администратором существующей позиции продукта	Данные о позиции продукта в базе данных изменены
Попытка удаления администратором существующей позиции продукта	Позиция продукта удалена из базы данных веб-приложения

5.2 Сквозное тестирование (end-to-end)

Сквозные тесты выполняют тестирование системы в целом, эмулируя реальную пользовательскую среду. Данные тесты позволяют протестировать полный цикл работы какого-либо сценария, начиная от пользовательского интерфейса и до отправки запроса на сервер. Сквозные тесты проверяют взаимодействие сервисов между собой. В интернете это тесты, запущенные в браузере, имитирующие щелчки мышью и нажатия клавиш.

Для сквозного тестирования в проектах, созданных с помощью фреймворка Ruby on Rails, используется фреймворк Capybara. Данный фреймворк имеет понятную и объемную документацию. На официальном сайте можно найти всю информацию по фреймворку, основные его плюсы и возможности.

5.3 Тестирование пользовательского интерфейса

Современное веб-приложение должно одинаково обрабатывать в различных браузерах. Задачей данного пункта тестирования является проверить корректность отображения страниц веб-приложения в различных браузерах, ибо в жизни пользователи могут пользоваться совершенно различными видами браузеров. Для точности и эффективности проведения тестирования были выбраны несколько популярных браузеров. В ходе тестирования страницы перезагружались по несколько раз за малый промежуток времени с целью нахождения ошибок. В результате ошибок обнаружено не было. Страницы перезагружались корректно, никаких артефактных изображений не было отображено. Для тестирования были выбраны браузеры Google Chrome, Safari, Opera. Пользовательский интерфейс в данных браузерах изображен на рисунках 5.3, 5.4, 5.5.

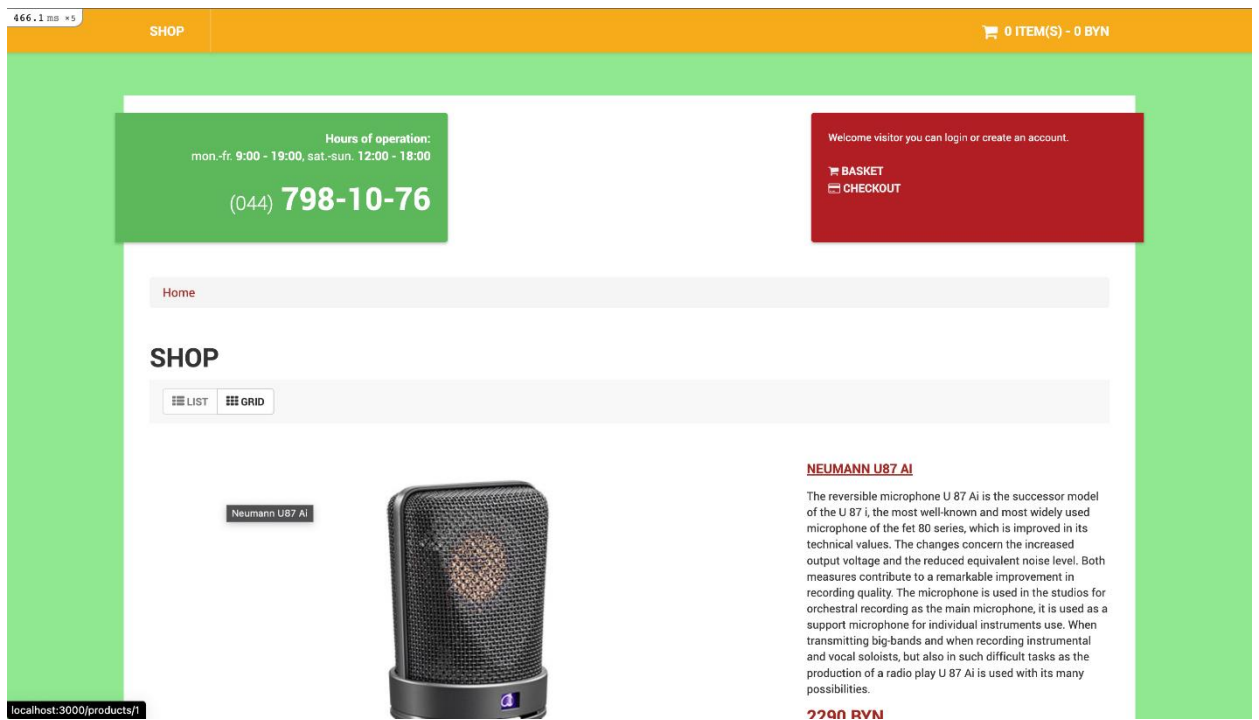


Рисунок 5.3 – Пользовательский интерфейс веб-приложения в браузере Google Chrome

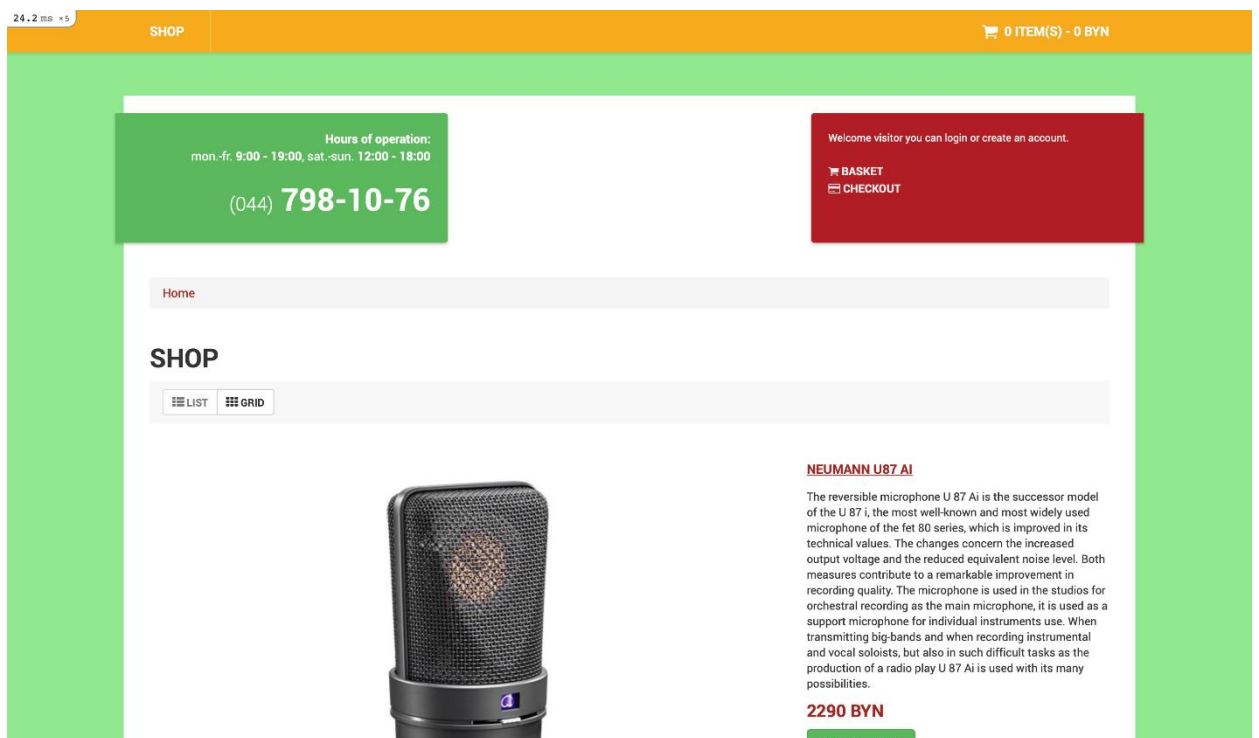


Рисунок 5.4 – Пользовательский интерфейс веб-приложения в браузере Safari

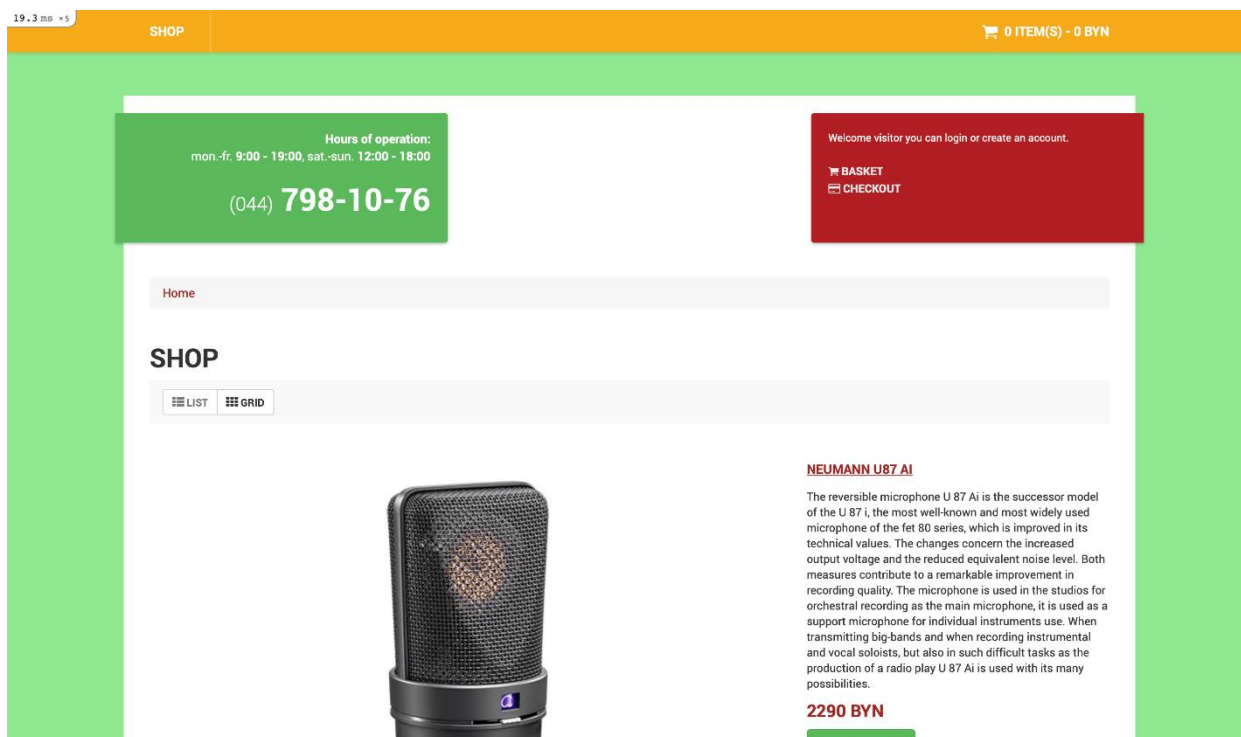


Рисунок 5.5 – Пользовательский интерфейс веб-приложения в браузере Opera

Аналогично проведем тестирование мобильной версии веб-приложения в различных браузерах и на различных мобильных устройствах с различными операционными системами. Тестирование проводилось аналогично тестированию десктопных версий веб-приложения: страницы перезагружались по несколько раз за малый промежуток времени с целью нахождения ошибок. В результате ошибок обнаружено не было. Страницы перезагружались корректно, никаких артефактных изображений не было отображено. Данные веб-приложения оставались неизменными. Все пользовательские данные сохранились во время тестирования. Мобильное устройство работало корректно, перегревов не было, ускоренной разрядки аккумуляторной батареи устройства замечено не было, при сворачивании-разворачивании браузера и множественной блокировке-разблокировке устройства страница веб-приложения визуально не отличалась от начального состояния при первом заходе на нее. Веб-приложение продолжало стабильную работу при открытии его в нескольких различных браузерах на одном устройстве, независимо от операционной системы устройства, при помощи которого проводится тестирование. Приложение во время тестирования ни разу не приостанавливало свою работу, корректно отображалось при множественной смене разрешения дисплея различных устройств, которые были использованы при тестировании веб-приложения. Для тестирования были выбраны браузеры Google Chrome, Safari, Opera. Пользовательский интерфейс в мобильных версиях данных браузеров изображен на рисунках 5.6, 5.7, 5.8.

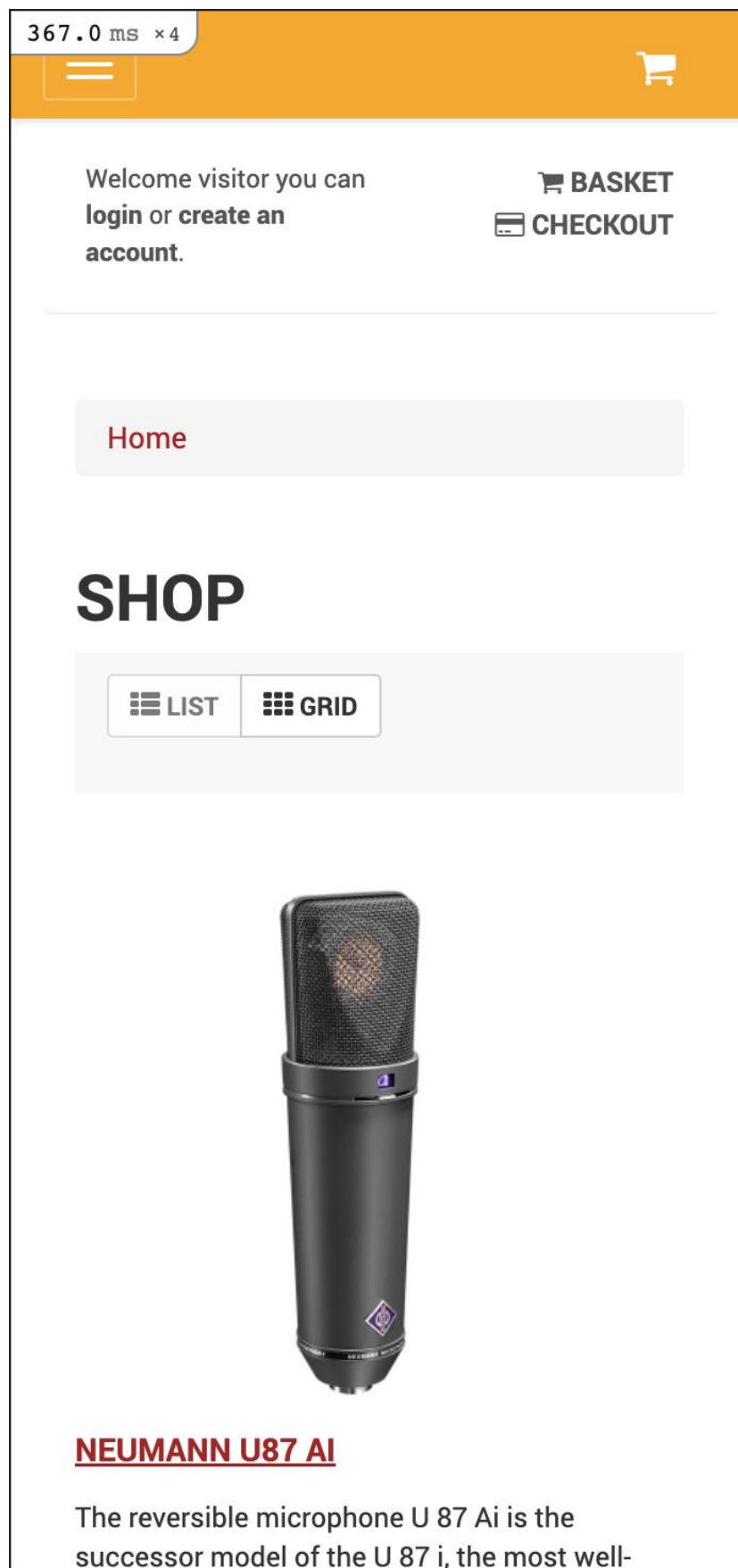


Рисунок 5.6 – Пользовательский интерфейс веб-приложения в мобильной версии браузера Google Chrome

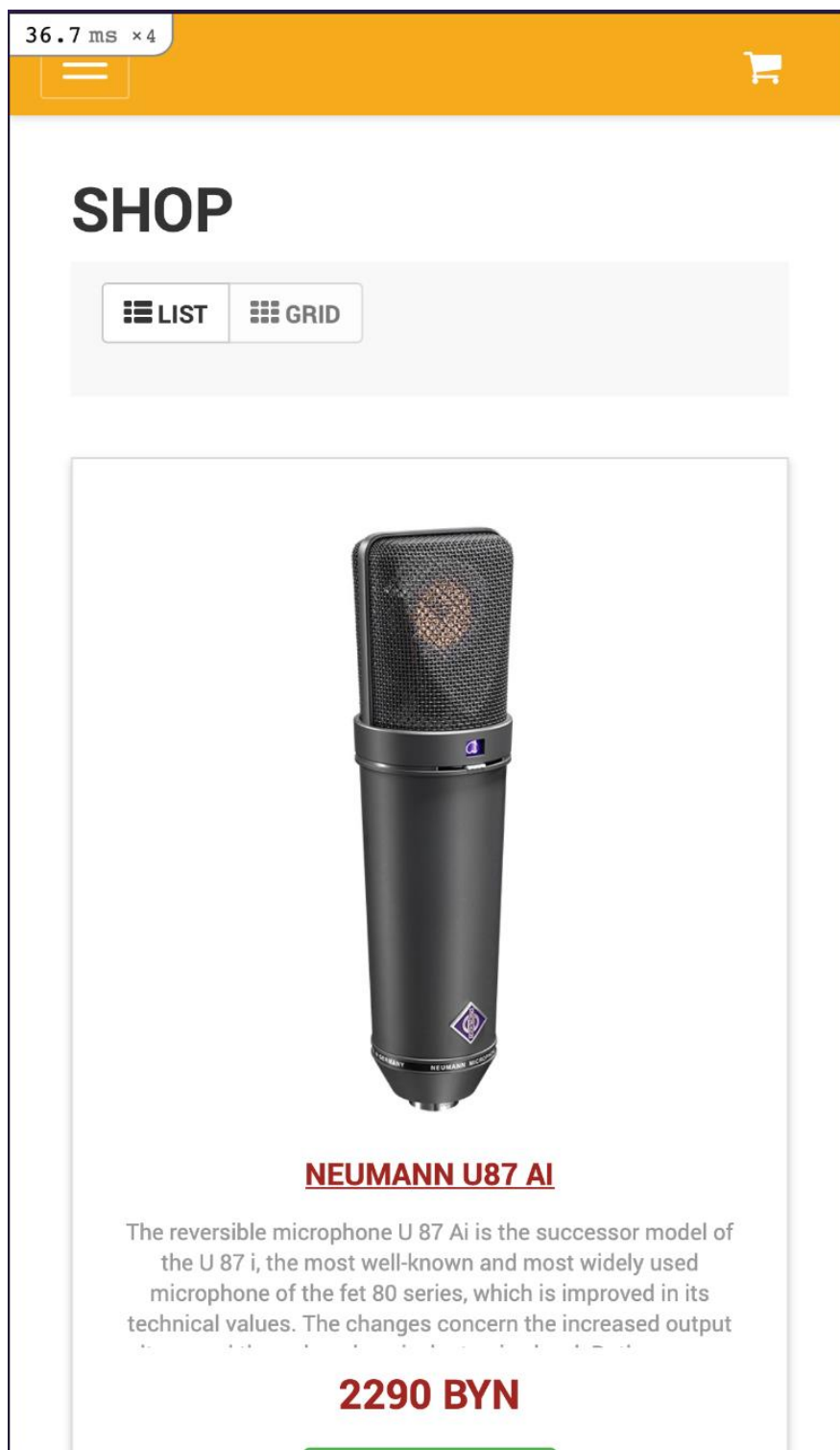



Рисунок 5.7 – Пользовательский интерфейс веб-приложения в мобильной версии браузера Safari

181.8 ms × 8

2290 BYN

ADD TO CART

DELETE FROM CART



SHURE SM 7 B

The Shure SM7B is a dynamic moving coil microphone with a fixed cardioid characteristic that is ideal for recording speech and vocals, but it also does a great job when miking instruments. A wide, linear frequency range ensures natural reproduction of voices and instruments. A pneumatic shock-absorbing mount and sophisticated shielding prevent mechanical and broadband electromagnetic interference of any kind from affecting the recordings, so you're all set to record without anything getting in the way!

300 BYN

Рисунок 5.8 – Пользовательский интерфейс веб-приложения в мобильной версии браузера Opera

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Требования к аппаратному обеспечению

Как минимальные требования для работы приложения можно обозначить минимальные требования для работы современного браузера. Для обозначения минимальных требований, возьмем требования к аппаратному обеспечению для браузера Google Chrome. Таким образом, требования к аппаратному обеспечению клиентской части приложения представлены в таблице 6.1.

Таблица 6.1 – Требования клиентской части к аппаратному обеспечению [10]

	Windows	Mac	Linux
Операционная система	Windows 7 Windows 8 Windows 8.1 Windows 10	Mac OS X 10.10	Ubuntu 14.4 Debian 8 OpenSUSE 13.3 Fedora Linux 14
Процессор	Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2		
Видеоадаптер	3D адаптер nVidia, Intel, AMD/ATI		
Видеопамять	64 Мб		
Свободное место на диске	350 Мб		
Оперативная память	512 Мб		

Минимальные требования к аппаратному обеспечению компьютера для развертывания серверной части:

1. Процессор Intel Core i5 7400.
2. Объем свободного пространства на постоянном запоминающем устройстве – пятьдесят ГБ файлового хранилища.
3. Объем оперативного запоминающего устройства – шестнадцать ГБ.
4. Наличие доступа к сети Internet.
5. Клавиатура.
6. Мышь.
7. Монитор.

6.2 Руководство по развертыванию приложения

При разработке программного средства использовался язык программирования Ruby. Использование данного языка программирования накладывает определенные ограничения при развертывании приложения. Для корректного развертывания приложения на компьютере должна быть

установлена любая UNIX OS. В случае данного дипломного проекта используется macOS Monterey 12.3.

Для запуска серверной части в первую очередь необходимо установить и настроить интерпретатор языка Ruby. Для выбранной операционной системы доступна установка средствами стандартного пакетного менеджера. Однако для установки наиболее актуальной версии интерпретатора и последующей корректной установки библиотек следует выполнять установку с использованием специальной утилиты RVM (Ruby Version Manager). Для установки данной утилиты требуется выполнить следующую команду:

```
curl -sSL https://get.rvm.io | bash -s stable
```

Далее устанавливаем стабильную версию языка Ruby. На момент написания данной работы таковой является версия 3.1.2. Для установки необходимо выполнить команду:

```
rvm install ruby
```

После установки языка Ruby нам понадобится установить библиотеки bundler и rails:

```
gem install bundler rails
```

Далее надо настроить базу данных. Для базы данных будет использоваться PostgreSQL. Для установки серверной и клиентской части PostgreSQL необходимо выполнить следующую команду:

```
sudo apt-get install postgresql postgresql-contrib
```

Для запуска клиентской части приложения необходимо скачать и установить последнюю версию Node.js. Для этого необходимо выполнить следующую команду:

```
brew install nodejs
```

Далее надо установить все сторонние пакеты зависимостей, перечисленные в файле package.json, необходимые для работы приложения. Для этого необходимо выполнить следующие команды:

```
sudo npm install -g yarn
```

Для запуска отправления почтовых уведомлений необходимо установить планировщик заданий Sidekiq и хранилище данных Redis. Для этого необходимо выполнить следующие команды:

```
brew install redis
```

```
bundle add sidekiq
```

Основная настройка сервера завершена. Далее, для локального запуска приложения необходимо установить используемые приложением библиотеки командой `bundle install`, выполнить настройку с помощью команды `bin/setup`, запустить хранилище данных Redis с помощью команды `brew services start redis` и запустить планировщик заданий Sidekiq командой `bundle exec sidekiq`. И затем для запуска приложения выполнить команду `rails s`.

6.3 Руководство по использованию ПО

6.3.1 Главная страница

Перейдя по ссылке приложения впервые пользователь будет направлен на главную страницу сайта. Здесь пользователь может увидеть список позиций, которые предлагает данный магазин музыкального оборудования, кнопки переходов на страницы корзины и оформления заказов. Для более удобного просмотра списка позиций пользователь может нажать кнопку «GRID» и отобразить список в виде сетки. Главная страница показана на рисунке 6.1.

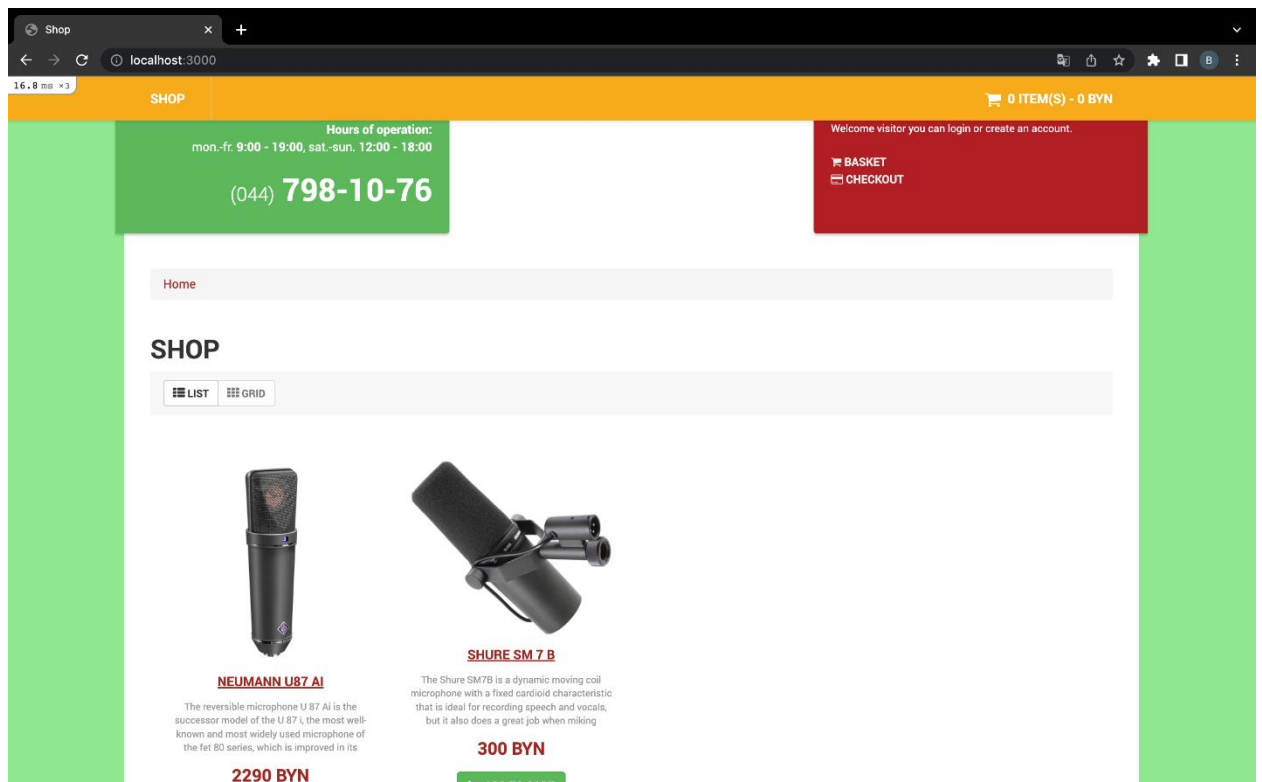


Рисунок 6.1 – Главная страница

Далее пользователь имеет возможность просмотреть выбрать необходимый ему товар. Для этого пользователю необходимо нажать на название товара, при этом произойдет переход на страницу данной позиции оборудования.

При нажатии кнопки «BASKET» пользователь попадает на страницу корзины с товаром, который пользователь выбрал, нажав на кнопку «ADD TO CART», которая представлена на рисунке 6.2.

При нажатии кнопки «CHECKOUT» пользователь попадает на страницу оформления заказа, где пользователем производится оформление заказа на покупку выбранных позиций. Страница оформления заказа представлена на рисунках 6.5 и 6.6.

Для оформления заказа пользователю необходимо создать собственный аккаунт в данном интернет-магазине. Для этого необходимо на главной странице в разделе «My Account» нажать кнопку «Register», после чего произойдет переход на страницу регистрации. При наличии уже существующего аккаунта в разделе «My Account» необходимо нажать на кнопку «Login», которая при нажатии произведет перенаправление на страницу ввода данных, необходимых для входа в аккаунт. При нажатии на кнопку «Login» при отсутствии учетной записи присутствует возможность зарегистрировать учетную запись, нажав на кнопку «CONTINUE» на странице входа в учетную запись произойдет перенаправление на страницу регистрации учетной записи пользователя.

6.3.2 Страница корзины

На данной странице пользователю отображен список выбранных им позиций музыкального оборудования в веб-приложении, заголовок с названием позиции данного оборудования, при нажатии на который можно перейти на страницу самой позиции оборудования для изучения более подробной информации, краткое описание позиции музыкального оборудования. С помощью кнопки «CLEAR BASKET» пользователь может удалить из корзины весь список выбранных им товаров и затем приступить к заказу снова, перейдя на главную страницу веб-приложения. При желании добавить еще одну такую же позицию товара пользователь может нажать кнопку «ADD ANOTHER», которая расположена рядом с изображением позиции, и в корзину добавится еще одна такая же позиция. Для удаления одной единицы позиции пользователь может нажать расположенную рядом с изображением кнопку «DELETE ONE» и удалить один элемент товара из корзины. При манипуляциях пользователя с товаром происходит изменение конечной стоимости товаров в корзине, которое отображается в верхнем меню веб-приложения и у каждого отдельного товара музыкального оборудования по отдельности. Страница корзины представлена на рисунке 6.2.

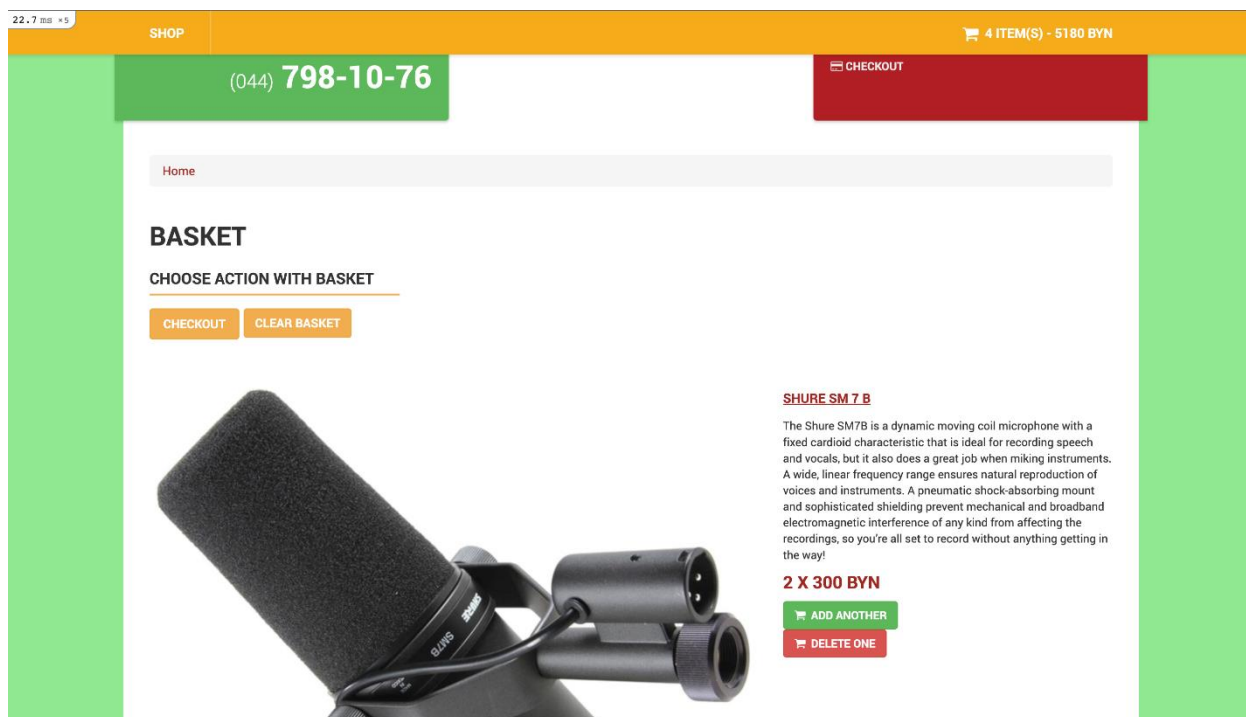


Рисунок 6.2 – Страница корзины

6.3.3 Руководство по входу пользователя в систему

Для оформления заказа позиции товара музыкального оборудования с помощью разработанного веб-приложения, пользователю необходимо быть зарегистрированным в системе веб-приложения и быть записанным в базе данных данного веб-приложения. Страница регистрации учетной записи пользователя представлена на рисунке 6.3. Для регистрации пользователю в системе веб-приложения необходимо указать в соответствующих формах свое имя и адрес электронного ящика, которые затем будут использованы при входе в аккаунт в веб-приложении, и номер телефона в семизначном виде, а также придумать пароль для входа в учетную запись. После ввода данные пользователя будут записаны в базу данных разработанного веб-приложения и будут использованы в таких действиях как вход пользователя в систему веб-приложения, оформление пользователем заказа позиций музыкального оборудования.

Для успешного входа в созданный в системе аккаунт, пользователю необходимо обязательно указать логин и пароль своей учетной записи на странице входа в соответствующих формах, которая изображена на рисунке 6.4. При попытке пользователем входа без ввода пароля, ему будет выведено сообщение о невозможности входа без введения пароля. Если пользователь введет неверные данные логина или пароля, то ему будет выведено сообщение о том, что один или оба введенных параметра не являются корректными и вход в систему веб-приложения невозможен.

Рисунок 6.3 – Страница регистрации

Рисунок 6.4 – Страница входа в учетную запись

6.3.4 Страница оформления заказа

Для оформления заказа пользователю необходимо нажать из главного меню на кнопку «CHECKOUT» после чего произойдет перенаправление пользователя на страницу оформления заказа. Возможность попасть на страницу регистрации присутствует на странице корзины, а также в разделе

«My Account». Страница оформления заказа представлена на рисунках 6.5 и 6.6.

26.3

SHOP

3 ITEM(S) - 6870 BYN

[Home](#) / [Basket](#) / [Checkout](#)

CHECKOUT

Step 1: Name Options

* Add Name To Your Order

Valery Petrikevich

Step 2: Phone Options

* Add Phone To Your Order

7981076

Step 3: Address Options

* Add Destination Address To Your Order

Step 4: Confirm Order



Product Name	Model	Quantity	Price	Total
Neumann U87 Ai		3	2290	6870

Рисунок 6.5 – Страница оформления заказа

26.3

SHOP

3 ITEM(S) - 6870 BYN



Sub-Total:	6870
НДС 20%:	1374.0
Total:	8244.0

CONFIRM ORDER

Рисунок 6.6 – Продолжение страницы оформления заказа

Здесь, на странице оформления заказа, пользователю необходимо ввести имя человека, на которого придет заказ, контактный номер телефона и адрес доставки. По умолчанию все параметры, кроме адреса доставки, будут заполнены по аналогии с введенными пользователем при регистрации данными.

Ниже на странице, после введения пользователем данных для заказа, пользователю предоставляется цена заказа с учетом налога на добавочную стоимость, а также пользователь может ознакомиться с конечной стоимостью оборудования и с размером налога.

Для окончания оформления заказа пользователь должен нажать кнопку «CONFIRM ORDER», по нажатии которой пользователю на указанный при регистрации электронный ящик придет письмо с подтверждением заказа, в котором будут отображены указанные пользователем данные при заказе.

6.3.5 Страница администратора

Для взаимодействия с содержимым веб-приложения существует роль администратора в приложении. Для этого пользователь с определенными данными назначается администратором, который получает права на редактирование приложения. Для взаимодействия с приложением администратор должен ввести свои данные на странице входа. После успешного входа, администратору станет доступен функционал во вкладке «Admin». Доступные администратору разделы представлены на рисунке 6.7.

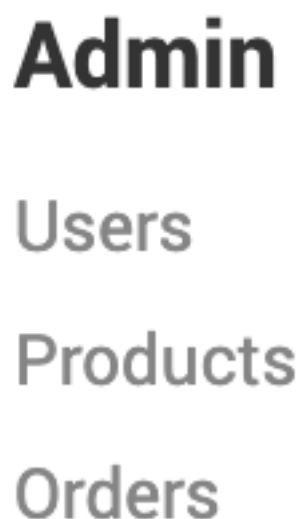


Рисунок 6.7 – Доступные администратору разделы

6.3.6 Страница редактирования пользователей

Администратор обладает возможностью редактировать данные пользователей, информация о которых была записана в базу данных

разработанного веб-приложения. Для этого на вкладке в нижней части страницы «Admin» администратор должен нажать на кнопку «Users» и перейти на страницу редактирования пользователей. На данной странице отображается список пользователей. На записи каждого пользователя присутствуют кнопки «EDIT» и «DELETE», с помощью которых администратор может перейти на страницу редактирования данных существующего пользователя или удалить данного пользователя из базы данных веб приложения. Страница редактирования пользователей представлена на рисунке 6.8.

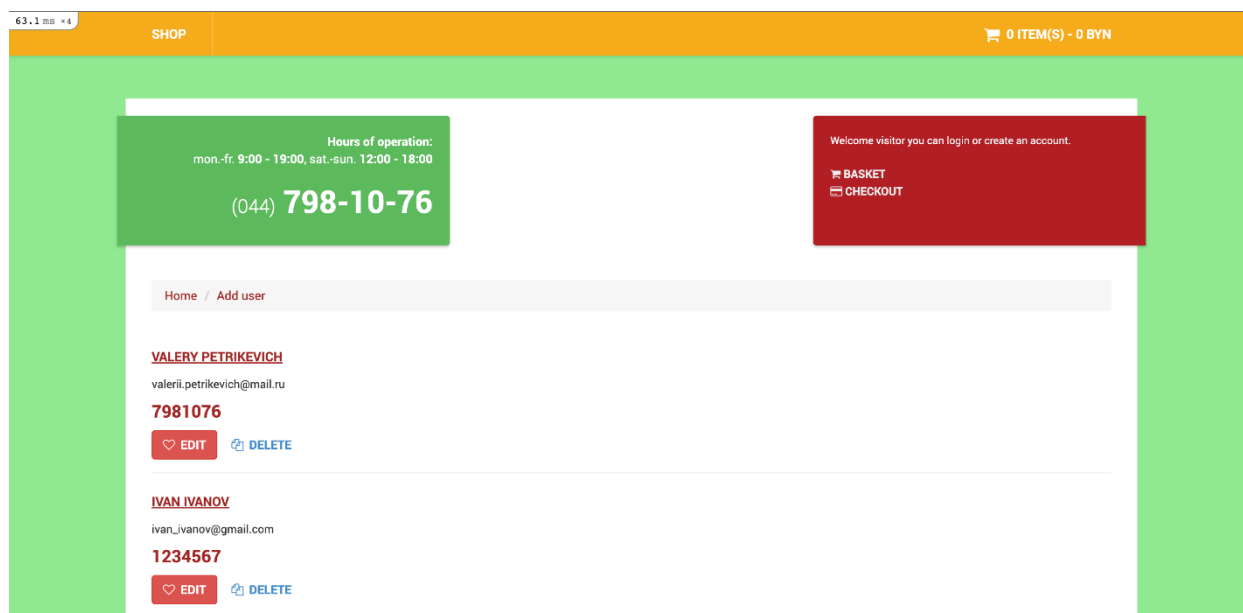


Рисунок 6.8 – Страница редактирования пользователей

При нажатии администратором кнопки «Edit», происходит перенаправление пользователя-администратора на страницу редактирования данных пользователя, занесенного в базу данных разработанного веб-приложения, которая представлена на рисунке 6.9. На данной странице администратор имеет возможность изменить все данные пользователя, записанные в базу данных веб-приложения, а именно: его имя пользователя в системе, адрес электронной почты и пароль пользователя. При смене пароля пользователя администратору будет необходимо после введения нового пароля в форму ввести этот же новый пароль в поле формы «Confirm password» для подтверждения нового пароля пользователя. Для сохранения результатов изменений данных пользователя в базе данных разработанного веб-приложения, пользователь-администратор должен нажать кнопку «CONTINUE» после этого обновленные данные о пользователе магазина будут записаны и сохранены в базе данных разработанного веб-приложения.

34.8 ms

SHOP

0 ITEM(S) - 0 BYN

Hours of operation:
mon.-fr. 9:00 - 19:00, sat.-sun. 12:00 - 18:00

(044) 798-10-76

Welcome visitor you can login or create an account.

BASKET

CHECKOUT

Home / Users

UPDATE USER

Your Personal Details

Name:

Email:

Telephone:

Your Password

Password:

Confirm password:

CONTINUE

Рисунок 6.9 – Страница редактирования данных пользователя

6.3.7 Страница редактирования позиций

Администратор обладает возможностью добавлять новые позиции музыкального оборудования в каталог магазина, а также изменять данные существующих позиций. Для этого на вкладке «Admin» администратору необходимо нажать на кнопку «Products» и произойдет перенаправление на страницу с позициями товаров. Для создания новой позиции администратору необходимо нажать на кнопку «Add product» и перейти на страницу создания позиции. Страница создания позиции представлена на рисунке 6.10. При создании новой позиции товара, администратору необходимо указать наименование позиции, ее описание, цену и загрузить изображение товара, нажав кнопку «Выберите файл» и выбрав файл изображения товара из существующих на жестком диске компьютера изображений, которые будут доступны после нажатия кнопки «Выберите файл». Для сохранения созданного товара администратору необходимо нажать кнопку «CONTINUE», после этого произойдет запись о создании нового товара в базу данных веб-приложения.

Для редактирования информации о позиции администратору необходимо на странице с позициями товаров нажать на сам товар и перейти на страницу с информацией о данном товаре. На странице товара администратору доступны кнопки «EDIT» и «DELETE», с помощью которых администратор может изменить данные о позиции или удалить данную позицию из каталога интернет-магазина. Страница товара представлена на рисунке 6.11.

49.5

SHOP

0 ITEM(S) - 0 BYN

Hours of operation:
mon.-fr. 9:00 - 19:00, sat.-sun. 12:00 - 18:00

(044) 798-10-76

Welcome visitor you can login or create an account.

BASKET

CHECKOUT

Home

NEW PRODUCT

Product's Details

Name:

Description:

Price:

Product's photo

Photo:

Выберите файл

Файл не выбран

CONTINUE

Рисунок 6.10 – Страница создания позиции

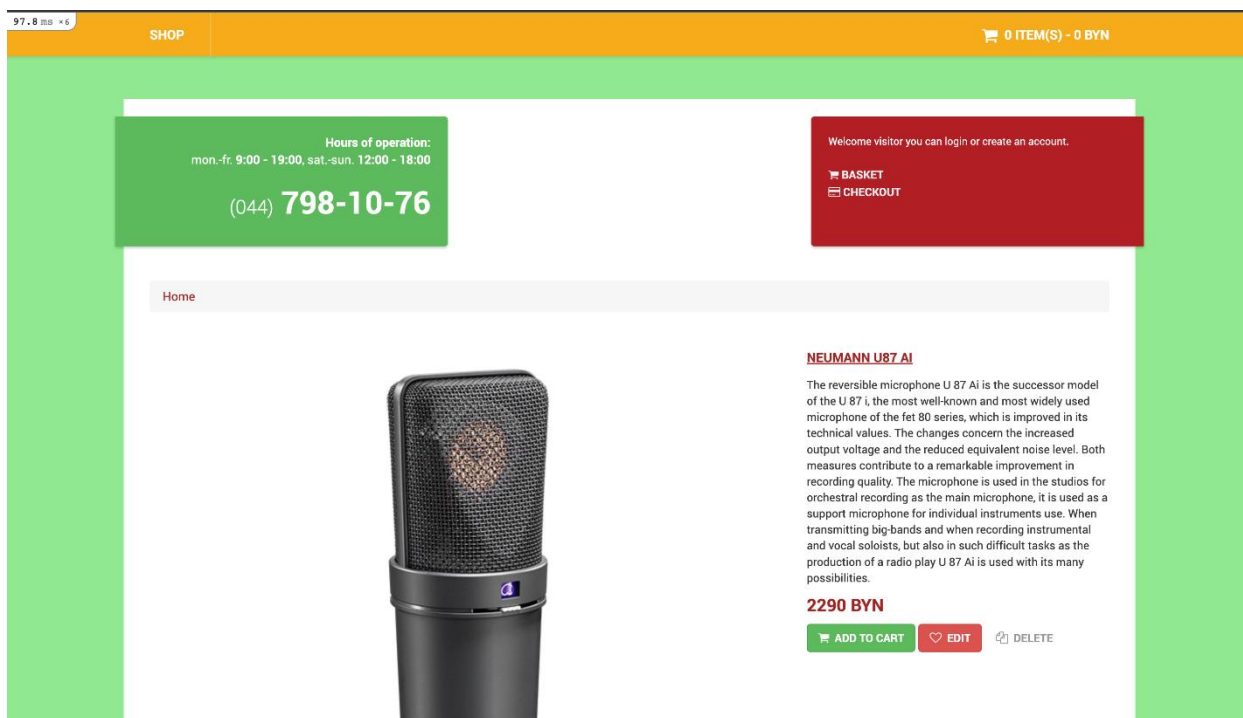


Рисунок 6.11 – Страница товара

Для редактирования информации о товаре администратору необходимо нажать на кнопку «EDIT» и перейти на страницу редактирования позиции. На данной странице администратор имеет возможность изменить все данные позиции и изменить изображение позиции в каталоге. Для сохранения

результатов изменений, администратор должен нажать кнопку «CONTINUE» после этого обновленные данные о позиции будут записаны и сохранены в базе данных. Страница редактирования данных позиции представлена на рисунке 6.12.

Рисунок 6.12 – Страница редактирования данных позиции

6.3.8 Страница редактирования заказов

Администратор обладает возможностью изменять данные заказов пользователей. Для доступа к изменению заказов администратору необходимо на вкладке «Admin» в нижнем меню страницы нажать на кнопку «Orders» и произойдет перенаправление пользователя-администратора на страницу редактирования заказов. Страница редактирования заказов представлена на рисунке 6.13. На странице редактирования заказов отображен список заказов и их стоимость. Пользователь-администратор может удалить заказ из базы данных веб-приложения, нажав кнопку «DELETE», или перейти на страницу редактирования заказа, нажав кнопку «EDIT».

На данной странице администратор имеет возможность изменить все данные заказа, занесенные в базу данных веб-приложения. Для сохранения результатов изменений, администратор должен нажать кнопку «EDIT ORDER» после этого обновленные данные о заказе будут записаны и сохранены в базе данных. Страница редактирования заказа представлена на рисунке 6.14.

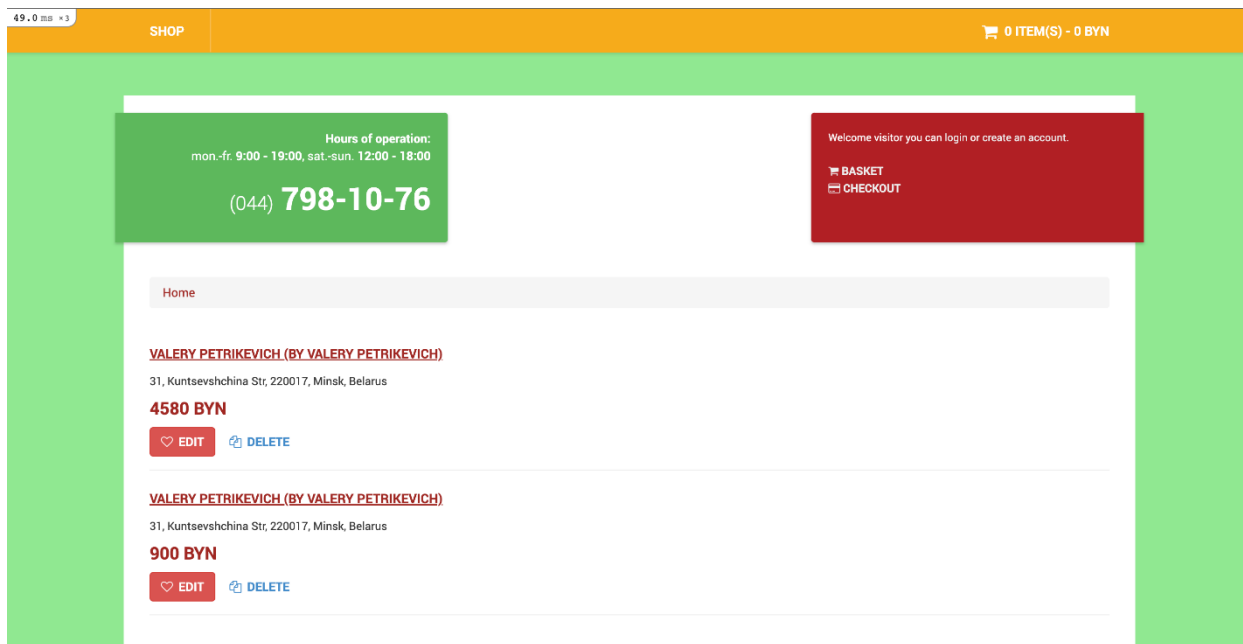


Рисунок 6.13 – Страница редактирования заказов

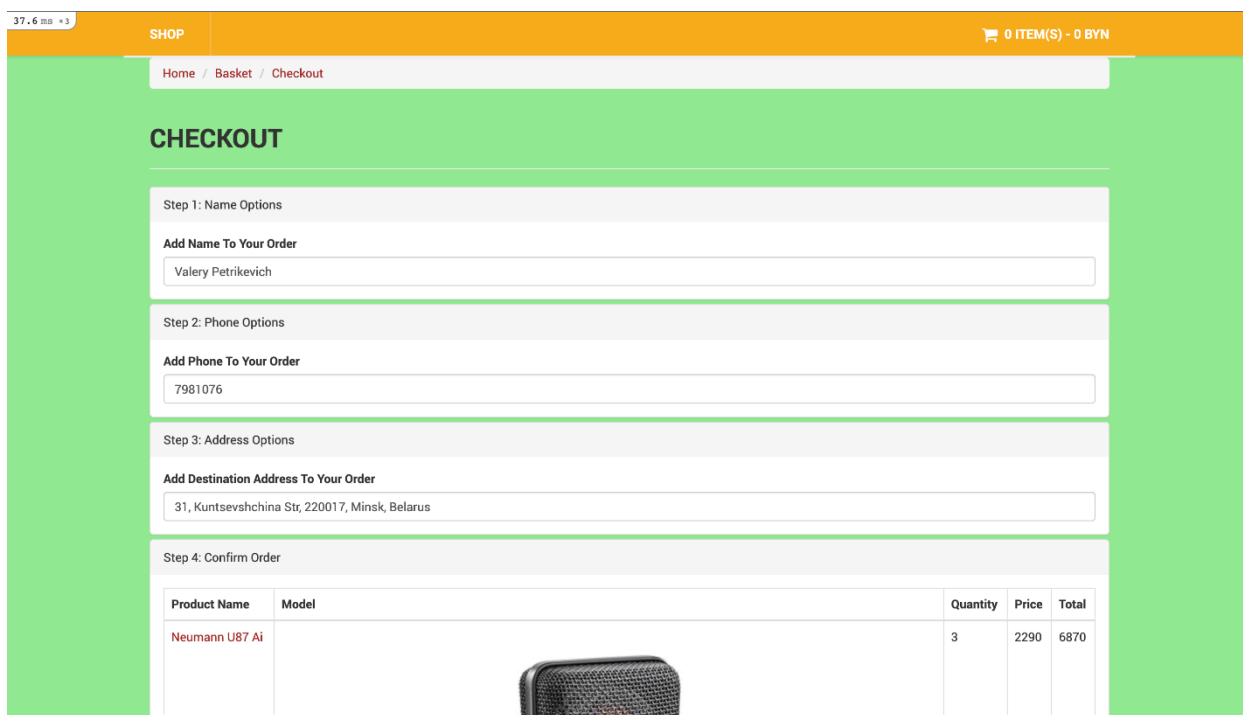


Рисунок 6.14 – Страница редактирования заказа

7 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И РЕАЛИЗАЦИИ ПРОГРАММНОГО МОДУЛЯ ВЕБ-ПРИЛОЖЕНИЯ ПО ПРОДАЖЕ МУЗЫКАЛЬНОГО ОБОРУДОВАНИЯ

7.1 Характеристика разработанного программного средства

Целью данного дипломного проекта является разработка веб-приложения по продаже музыкального оборудования. В ходе разработки программной части будет получен программный комплекс. Данное веб-приложение реализуется как набор программ, выполняющих функции проекта, позволяющего изменять и вносить необходимые данные, а также предоставляющего доступ клиентам. Веб-приложение реализуется по заказу магазина музыкального оборудования. Главными требованиями, положенными в основу при разработке комплекса, стали легкое использование и расширяемость.

Задачей данного программного продукта является привлечение потенциальных клиентов, которые совершат выбор необходимых им позиций и, при необходимости, приобретут их. Визуальная концепция приложения должна быть информационно-коммерческой, то есть ориентирована на удобное предоставление всей информации, необходимой потенциальному клиенту для выбора товара.

Разработанное программное средство позволит обеспечить более тесное взаимодействие магазина с пользователями через Интернет. С помощью данного программного продукта клиенты могут получить информацию о нужных для них позициях без необходимости выхода из дома, а также совершить заказ интересующих их товаров. Разработанный программный комплекс соответствует требованиям пользователей, не нуждается в большом количестве аппаратных ресурсов, основан на технологии, работа которой не зависит от выбора платформы.

7.2 Расчет цены программного модуля веб-приложения по продаже музыкального оборудования на основе затрат

Для реализации данного проекта компания-разработчик заключила соглашение с компанией-заказчиком на разработку веб-приложения. В соглашении определены требования к программному средству и установлена цена. Цена программного средства будет определена на основе полных затрат на разработку организацией-разработчиком ($C_{отп}$) и включает в себя следующие статьи затрат: основная заработная плата разработчиков, дополнительная заработная плата разработчиков, отчисления на социальные нужды, прочие расходы, общая сумма затрат на разработку, плановая прибыль (включаемая в цену программного средства), отпускная цена программного средства.

7.2.1 Затраты на основную заработную плату команды разработчиков

Для расчета затрат на разработку программного средства в первую очередь необходимо рассчитать основную заработную плату команды разработчиков.

Расчет осуществляется исходя из состава и численности команды, размера месячной заработной платы каждого участника команды, а также трудоемкости работ, выполняемых при разработке программного средства отдельными исполнителями по формуле:

$$З_o = K_{\text{пр}} \sum_{i=1}^n З_{\text{чи}} \cdot t_i, \quad (7.1)$$

где $K_{\text{пр}}$ – коэффициент премий (равный 1,3);

n – категории исполнителей, занятых разработкой программного средства;

$З_{\text{чи}}$ – часовая заработная плата исполнителя i -й категории, р.;

t_i – трудоемкость работ, выполняемых исполнителем i -й категории, определяется исходя из сложности разработки программного обеспечения и объема выполняемых им функций, ч.

Проведя анализ предметной области и переговоры с ведущими специалистами-разработчиками, были решены следующие вопросы: состав команды разработчиков и оценена трудоемкость работ. Трудоемкость работ оценена специалистами сроком в 2 месяца, что составляет 336 рабочих часов (по данным Министерства труда и социальной защиты населения, количество рабочих часов в месяце равно 168ч.), а состав команды разработчиков включает в себя бизнес-аналитика, системного архитектора, программиста, тестировщика и дизайнера. Размеры заработных плат сотрудников указаны по состоянию на 28.04.2022. Так как заработная плата считается в долларах США, все показатели были пересчитаны по настоящему валютному курсу Национального Банка, равного 2,68BYN. Часовая заработная плата каждого исполнителя определялась путем деления его месячной заработной платы (оклад плюс надбавки) на количество рабочих часов в месяце. Все данные представлены в таблице 7.1.

Таблица 7.1 – Расчет затрат на основную заработную плату разработчиков

Категория исполнителя	Месячная заработная плата, руб.	Часовая заработная плата, руб.	Трудоемкость работ, ч	Итого, руб.
1	2	3	4	5
Разработчик ПО	2100,00	12,50	250	3125,00
Тестировщик	1540,00	9,17	100	917,00

Продолжение таблицы 7.1

1	2	3	4	5
Ведущий разработчик ПО	3000,00	17,86	120	2142,86
Дизайнер	1500,00	8,93	90	803,70
Итого				6988,56
Премия (30% от основной заработной платы)				2096,57
Всего затрат на заработную плату разработчиков				9085,13

7.2.2 Затраты на дополнительную заработную плату команды

Дополнительная заработная плата – это оплата за сверхурочный труд, различные трудовые успехи и надбавки за особые условия труда команды и включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах (составляет 15%) к основной заработной плате по следующей формуле:

$$З_д = \frac{З_о \cdot Н_д}{100\%} \quad (7.2)$$

где $З_о$ – затраты на основную заработную плату;

$Н_д$ – норматив дополнительной заработной платы, 15%.

Подставим имеющиеся значения в формулу 7.2 и получим расчет:

$$З_д = \frac{9085,13 \cdot 15\%}{100\%} = 1362,77 \text{ руб.}$$

Согласно расчетам, затраты на дополнительную заработную плату команды составит 1362,77 рублей.

7.2.3 Отчисления на социальные нужды

В статье отчисления на социальные нужды отражаются обязательные отчисления по установленным законодательством тарифам в фонд социальной защиты населения, а также расходы предприятия на обязательное медицинское страхование некоторых категорий работников в соответствии с законодательством. Расчет размера отчислений в фонд социальной защиты населения и на обязательное страхование определяется в соответствии с действующими законодательными актами Республики Беларусь и вычисляются по формуле:

$$P_{\text{соц}} = \frac{(Z_o + Z_d) \cdot H_{\text{соц}}}{100\%}, \quad (7.3)$$

где $H_{\text{соц}}$ – норматив отчислений на социальные нужды, %.

Согласно законодательству Республики Беларусь, отчисления на социальные нужды составляют 34% в фонд социальной защиты и 0,6% на обязательное страхование. Подставим имеющиеся значения в формулу 7.3 и получим результат:

$$P_{\text{соц}} = \frac{(9085,13 + 1362,77) \cdot 34,6\%}{100\%} = 3614,97 \text{ руб.}$$

Согласно расчетам, размер отчислений в фонд социальной защиты и на обязательное страхование составляет 3614,97 рублей.

7.2.4 Прочие расходы

Прочие расходы связаны с функционированием организации-разработчика в целом, например: затраты на аренду офисных помещений, отопление, освещение, амортизацию основных производственных фондов и т.д. При расчете данной статьи затрат учитывается норматив прочих затрат в целом по организации. В данном случае 30%. Расходы по данной статье осуществляется в процентах от затрат на основную заработную плату команды разработчиков и рассчитывается по формуле:

$$P_{\text{пр}} = \frac{Z_o \cdot H_{\text{пз}}}{100\%}, \quad (7.4)$$

где $H_{\text{пз}}$ – норматив прочих затрат в целом по организации, 30%;

Подставим имеющиеся значения в формулу 7.4 и произведем расчет.

$$P_{\text{пр}} = \frac{9085,13 \cdot 30\%}{100\%} = 2725,54 \text{ руб.}$$

Таким образом, размер прочих расходов составляет 2725,54 рублей.

7.2.5 Общая сумма затрат на разработку

Общая сумма затрат на разработку рассчитывается путем суммирования основной заработной платы, дополнительной заработной платы, отчислений на социальные нужды, прочих затрат. Представим в виде формулы:

$$Z_p = Z_o + Z_d + P_{\text{соц}} + P_{\text{пр}}. \quad (7.5)$$

Подставим имеющиеся значения в формулу 7.5 и произведем расчет.

$$З_p = 9085,13 + 1362,77 + 3614,97 + 2725,54 = 16788,41 \text{ руб.}$$

Согласно расчетам, общая сумма затрат на разработку составляет 16788,41 рублей.

7.2.6 Плановая прибыль, включаемая в цену программного средства

Плановая прибыль, включаемая в цену программного средства, рассчитывается по формуле:

$$П_{п.с} = \frac{З_p \cdot P_{п.с}}{100}, \quad (7.6)$$

где $P_{п.с}$ – рентабельность затрат на разработку программного средства, 40%

В данном случае рентабельность установили на уровне 40%. Подставим имеющиеся значения в формулу 7.6 и произведем расчет.

$$П_{п.с} = \frac{16788,41 \cdot 40\%}{100\%} = 6715,36 \text{ руб.}$$

Исходя из расчетов, плановая прибыль, включаемая в цену программного средства, составляет 6715,36 рублей.

7.2.7 Отпускная цена программного средства

Отпускная цена программного продукта представляет собой сумму затрат на заработную плату и плановой прибыли. Рассмотрим основную формулу:

$$Ц_{п.с} = З_p + П_{п.с} \quad (7.7)$$

Подставим имеющиеся значения в формулу 7.7, произведем расчеты.

$$Ц_{п.с} = 16788,41 + 6715,36 = 23503,77 \text{ руб.}$$

Таким образом, отпускная цена программного средства составляет 23503,77 рублей.

Расчет цены на разработку программного средства представлен в итоговой таблице 7.2.

Таблица 7.2 – Результаты расчета цены на разработку программного средства.

Наименование статьи затрат	Сумма, р.
1. Основная заработная плата разработчиков	9085,13
2. Дополнительная заработная плата разработчиков	1362,77
3. Отчисления на социальные нужды	3614,97
4. Прочие расходы	2725,54
5. Всего затраты на разработку	16788,41
6. Плановая прибыль	6715,36
7. Цена программного средства	23503,77

7.3 Расчет результата от разработки и реализации программного модуля веб-приложения по продаже музыкального оборудования

В данном случае организация выступает в лице разработчика программного средства по индивидуальному заказу. Для организации-разработчика экономическим эффектом является прирост чистой прибыли, полученной от разработки и реализации программного средства заказчику. Так как программное средство будет реализовываться организацией-разработчиком по отпускной цене, сформированной на основе затрат на разработку, то экономический эффект, полученный организацией-разработчиком, в виде прироста чистой прибыли от его разработки, определяется по формуле:

$$\Delta\Pi_{\text{ч}} = \Pi_{\text{п.с}} \left(1 - \frac{H_{\text{п}}}{100}\right), \quad (7.8)$$

где $\Pi_{\text{п.с}}$ – прибыль, включаемая в цену программного средства, р;

$H_{\text{п}}$ – ставка налога на прибыль согласно действующему законодательству, (по состоянию на 01.01.2022 г. – 18 %).

Подставим имеющиеся данные в формулу 7.8 и произведем расчет:

$$\Delta\Pi_{\text{ч}} = 6715,36 \left(1 - \frac{18\%}{100\%}\right) = 5506,60 \text{ руб.}$$

Экономический эффект равен 5506,60 рублей.

7.4 Расчет показателей экономической эффективности разработки программного модуля веб-приложения по продаже музыкального оборудования

Для организации-разработчика программного средства оценка экономической эффективности разработки осуществляется с помощью расчета рентабельности затрат на разработку программного средства. Рентабельность является одним из основных показателей эффективности предприятия с точки зрения использования привлеченных средств. Она представляет собой отношение суммы чистой приведенной прибыли, полученной за весь расчетный период, к суммарным приведенным затратам за этот же период и определяется по формуле:

$$P_z = \frac{\Delta\Pi_{\text{ч}}}{Z_p} \cdot 100 \%, \quad (7.9)$$

где $\Delta\Pi_{\text{ч}}$ – прирост чистой прибыли, полученной от разработки программного средства организацией-разработчиком по индивидуальному заказу, руб.;

Z_p – затраты на разработку программного средства организацией-разработчиком, руб.

Подставим имеющиеся данные в формулу 7.9 и произведем расчет.

$$P_z = \frac{5506,60}{16788,41} \cdot 100 \% = 32,8\%.$$

Рассчитанный показатель отображает, сколько чистой прибыли компания-разработчик получит от вложенных денег в разработку программного средства.

В результате проведения расчетов была определена необходимость разработки программного обеспечения, а также получен экономический эффект от использования данного программного продукта. По результатам проведенного экономического обоснования были получены следующие результаты:

1. Стоимость заказа на разработку веб-приложения по продаже музыкального оборудования составила 23503,77 рублей.

2. Прирост чистой прибыли составил 5506,60 рублей.

3. Данная разработка имеет положительный экономический эффект в размере 32,8%.

Таким образом, разработка и реализация по индивидуальному заказу программного модуля веб-приложения по продаже музыкального оборудования с экономической точки зрения целесообразна.

ЗАКЛЮЧЕНИЕ

Во время работы над дипломным проектом была произведена работа по разработке клиентской и серверной частей веб-приложения по продаже музыкального оборудования.

Для написания данного веб-приложения использовался язык программирования Ruby. Данный язык был выбран ввиду минимизации затрат при использовании, быстрого написания кода, а также простоты разработки.

Для реализации пользовательского интерфейса был использован язык программирования JavaScript, а также библиотека jQuery.

Используемый в работе над проектом фреймворк Ruby on Rails – фреймворк, написанный на языке программирования Ruby с открытым исходным кодом, реализует архитектурный шаблон для веб-приложений. Ruby on Rails представляет собой архитектуру MVC для веб-приложений, а также обеспечивает их интеграцию с веб-сервером и сервером базы данных.

В сервисе веб-сервера была реализована функциональность, связанная с REST, заложена гибкая и расширяемая архитектура.

Таким образом результатами выполнения данного дипломного проекта стали:

1. Разработана гибкая архитектура, с помощью которой легко сопровождать и расширять разработанное приложение.
2. Разработан протокол взаимодействия между клиентской и серверной частями приложения.

Проведя расчет экономической эффективности, можно сделать вывод, что проектирование и разработка данного приложения являются целесообразными, принесут выгоду как компании-разработчику, так и покупателю данного приложения.

Дипломный проект является завершенным. Все поставленные задачи решены, решения реализованы программно. Заложена гибкая и расширяемая архитектура, которая при необходимости позволит решать задачи по расширению и дополнению веб-приложения дополнительными компонентами. При этом возможно дальнейшее улучшение и расширение приложения посредством добавления поддержки альтернативных способов ввода и обработки данных.

Заключительный плакат представлен на чертеже ГУИР.400201.078 ПЛ.2.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Сайт продажи музыкального оборудования [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.thomann.de/intl/by/index.html>. – Дата доступа: 24.03.2022.
- [2] Сайт продажи музыкального оборудования [Электронный ресурс]. – Электронные данные. – Режим доступа: <http://guitarland.by/>. – Дата доступа: 24.03.2022.
- [3] О Ruby [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.ruby-lang.org/ru/about/>. – Дата доступа: 27.03.22.
- [4] Язык программирования Руби. / D. Flanagan Y. Matsumoto – СПб. Издательство Питер, 2011. – 496с.
- [5] JavaScript [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://developer.mozilla.org/ru/docs/Learn/JavaScript>. – Дата доступа: 28.03.2022.
- [6] Официальный сайт JQuery [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://jquery.com/>. – Дата доступа: 01.04.2022.
- [7] Ruby on Rails [Электронный ресурс]. – Электронные данные. – Режим доступа: https://web-creator.ru/articles/why_ruby_on_rails. – Дата доступа: 01.04.22.
- [8] Шаблон MVC [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://web-creator.ru/articles/mvc>. – Дата доступа: 01.04.22.
- [9] PostgreSQL [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://www.postgresql.org/>. – Дата доступа: 02.04.2022.
- [10] Google Help [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://support.google.com/>. – Дата доступа: 10.05.2022.

ПРИЛОЖЕНИЕ А

(обязательное)

Листинг кода

/shop/app/models/application_record.rb

```
# frozen_string_literal: true

class ApplicationRecord < ActiveRecord::Base
  self.abstract_class = true
end
```

/shop/app/models/order.rb

```
# frozen_string_literal: true

class Order < ApplicationRecord
  has_many :order_products, dependent: :destroy
  has_many :products, through: :order_products

  belongs_to :user
end
```

/shop/app/models/order_product.rb

```
# frozen_string_literal: true

class OrderProduct < ApplicationRecord
  belongs_to :order
  belongs_to :product
end
```

/shop/app/models/product.rb

```
# frozen_string_literal: true

class Product < ApplicationRecord
  has_many :order_products, dependent: :destroy
  has_many :orders, through: :order_products

  belongs_to :brand
  belongs_to :category
  mount_uploader :photo, PhotoUploader
end
```

/shop/app/models/role.rb

```
# frozen_string_literal: true

class Role < ApplicationRecord
  has_and_belongs_to_many :users, join_table: :users_roles

  belongs_to :resource,
    polymorphic: true,
    optional: true

  validates :resource_type,
    inclusion: { in: Rolify.resource_types },
    allow_nil: true

  scopify
end
```

/shop/app/models/user.rb

```
# frozen_string_literal: true

class User < ApplicationRecord
  # Include default devise modules. Others available are:
  # :confirmable, :lockable, :timeoutable, :trackable and :omniauthable
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable

  has_many :orders, dependent: :destroy

  rolify

  has_paper_trail
end
```

/shop/app/models/brand.rb

```
# frozen_string_literal: true

class Brand < ApplicationRecord
  has_many :brand_categories
  has_many :categories, through: :brand_categories
  has_many :products
end
```

/shop/app/models/categories.rb

```
# frozen_string_literal: true

class Category < ApplicationRecord
  has_many :brand_categories
  has_many :brands, through: :brand_categories
  has_many :products
end
```

/shop/app/models/brand_categories.rb

```
# frozen_string_literal: true

class BrandCategory < ApplicationRecord
  belongs_to :brand
  belongs_to :category
end
```

/shop/app/controllers/application_controller.rb

```
# frozen_string_literal: true

require_dependency '../services/baskets_service.rb'
require_dependency '../services/checkouts_service.rb'

class ApplicationController < ActionController::Base
  before_action :basket
  before_action :authorize?

  before_action :configure_permitted_parameters, if: :devise_controller?

  protected

  def configure_permitted_parameters
    devise_parameter_sanitizer.permit(:sign_up) do |u|
      u.permit(:name, :email, :phone_number, :password, :password_confirmation)
    end
  end
end
```

```

def current_user?(user)
  current_user.id == user.id
end

def token
  @token ||= session[:token] || SecureRandom.hex(8)
  session[:token] = @token
end

def authorize
  redirect_to '/login' unless current_user
end

def authorize?
  token unless current_user
end

def basket
  @basket ||= ::BasketsService.new(session)
end

def redis_basket
  @redis_basket ||= Redis.current
end

def after_sign_in_path_for(_resource)
  root_path
end

def after_sign_out_path_for(_resource_or_scope)
  request.referrer
end
end

```

/shop/app/controllers/authentication_controller.rb

```

# frozen_string_literal: true

class AuthenticationController < ApplicationController
  def new
    @user = User.new
  end

  def create
    @user = User.find_by(email: user_params[:email])

    if @user&.authenticate(user_params[:password])
      session['warden.user.user.key'][0][0] = @user.id
      redis_basket.rename(token, session['warden.user.user.key'][0][0]) if
redis_basket.exists?(token)

      redirect_to admin_user_path(@user.id)
    else
      @error_message = 'Invalid email/password combination'
      @user = User.new
      render :new, status: :unprocessable_entity
    end
  end

  def destroy
    if redis_basket.exists?(session['warden.user.user.key'][0][0])
      redis_basket.rename(session['warden.user.user.key'][0][0],
                           token)
    end
    session['warden.user.user.key'][0][0] = nil
    redirect_to '/login'
  end

  private

  def user_params
    params.require(:user).permit(:email, :password)
  end
end

```

/shop/app/controllers/baskets_controller.rb

```
# frozen_string_literal: true

class BasketsController < ApplicationController
  def index
    @products = basket.all
  end

  def add
    ::BasketsService.new(session).add(params[:id])
    @products = basket.all
    redirect_to basket_path
  end

  def remove
    ::BasketsService.new(session).remove(params[:id])
    @products = basket.all
    redirect_to basket_path
  end

  def clear
    ::BasketsService.new(session).clear
    @products = basket.all
    redirect_to basket_path
  end
end
```

/shop/app/controllers/checkout_controller.rb

```
# frozen_string_literal: true

class CheckoutController < ApplicationController
  def new
    @products = @basket.all
    @order = Order.new
    @user = current_user || User.new
  end

  def create
    @order = ::CheckoutsService.new(session, order_params).make_order
    @basket.clear
    redirect_to '/'
  end

  private

  def order_params
    params.require(:order).permit(:user_id, :order_price, :dest_address, :name, :phone_number)
  end
end
```

/shop/app/controllers/products_controller.rb

```
# frozen_string_literal: true

class ProductsController < ApplicationController
  def index
    @products = Product.all
    @products_in_basket = basket.all
  end

  def show
    @product = Product.find(params[:id])
  end

  def add_to_basket
    ::BasketsService.new(session).add(params[:id])
    @products = Product.all
    @products_in_basket = basket.all
    redirect_to root_path
  end
end
```

```

end

def delete_from_basket
  ::BasketsService.new(session).remove(params[:id])
  @products = Product.all
  @products_in_basket = basket.all
  redirect_to root_path
end

private

def product_params
  params.require(:product).permit(:name, :description, :photo, :price)
end
end

```

/shop/app/controllers/sessions_controller.rb

```

# frozen_string_literal: true

class SessionsController < Devise::SessionsController
  def create
    super
    redis_basket.rename(token, session['warden.user.user.key'][0][0]) if
    redis_basket.exists?(token)
  end

  def destroy
    if redis_basket.exists?(session['warden.user.user.key'][0][0])
      redis_basket.rename(session['warden.user.user.key'][0][0], token)
    end
    super
  end
end

```

/shop/app/controllers/admin/admins_controller.rb

```

# frozen_string_literal: true

module Admin
  class AdminsController < ApplicationController
    before_action :authenticate_user!

    def admin?
      user = current_user
      user.has_role? :admin
    end

    def authorize_admin
      redirect_to '/' unless admin?
    end
  end
end

```

/shop/app/controllers/admin/orders_controller.rb

```

# frozen_string_literal: true

module Admin
  class OrdersController < AdminsController
    before_action :authorize_admin

    def index
      @orders = Order.all
    end

    def show
      @order = Order.find(params[:id])
      make_products_array
    end
  end
end

```



```

def new
  @products = @basket.all
  @order = Order.new
  @user = User.find(session['warden.user.user.key'][0][0])
end

def create
  @order = Order.new(order_params)

  if @order.valid?
    @order.save!
    redirect_to admin_order_path(@order.id)
  else
    @products = @basket.all
    @user = User.find(session['warden.user.user.key'][0][0])
    render :new, status: :unprocessable_entity
  end
end

def edit
  @order = Order.find(params[:id])
  make_products_array
end

def update
  @order = Order.find(params[:id])

  if @order.update(order_params)
    redirect_to admin_order_path(params[:id])
  else
    render :edit, status: :unprocessable_entity
  end
end

def destroy
  @order_products = OrderProduct.find_by(order_id: params[:id])
  @order = Order.find_by_id(params[:id])
  @order&.destroy

  redirect_to admin_orders_path
end

private

def order_params
  params.require(:order).permit(:user_id, :order_price, :dest_address, :name, :phone_number)
end

def make_products_array
  @products = Product.joins(:order_products).where(order_products: { order_id: @order.id })
end
end
end

```

/shop/app/controllers/admin/products_controller.rb

```

# frozen_string_literal: true

module Admin
  class ProductsController < AdminsController
    before_action :authorize_admin

    def index
      @products = Product.all
      @products_in_basket = @basket.all
    end

    def show
      @product = Product.find(params[:id])
    end

    def new
      @product = Product.new
    end
  end
end

```

```

def create
  @product = Product.new(product_params)

  if @product.valid?
    @product.save!
    redirect_to admin_product_path(@product.id)
  else
    render :new, status: :unprocessable_entity
  end
end

def edit
  @product = Product.find(params[:id])
end

def update
  @product = Product.find(params[:id])

  if @product.update(product_params)
    redirect_to admin_product_path(params[:id])
  else
    render :edit, status: :unprocessable_entity
  end
end

def destroy
  @product = Product.find_by_id(params[:id])
  @product&.destroy
  redirect_to admin_products_path
end

def add_to_basket
  ::BasketsService.new(session).add(params[:id])
  @products = Product.all
  @products_in_basket = @basket.all
  redirect_to admin_products_path
end

def delete_from_basket
  ::BasketsService.new(session).remove(params[:id])
  @products = Product.all
  @products_in_basket = @basket.all
  redirect_to admin_products_path
end

private

def product_params
  params.require(:product).permit(:name, :description, :photo, :price)
end
end
end

```

/shop/app/controllers/admin/roles_controller.rb

```

# frozen_string_literal: true

module Admin
  class RolesController < AdminsController
    before_action :authorize_admin

    def index
      @roles = Role.all
    end

    def show
      @role = Role.find(params[:id])
    end

    def new
      @role = Role.new
    end

    def create
      @role = Role.new(role_params)
    end
  end
end

```

```

    if @role.valid?
      @role.save!
      redirect_to admin_role_path(@role.id)
    else
      render :new, status: :unprocessable_entity
    end
  end

  def edit
    @role = Role.find(params[:id])
  end

  def update
    @role = Role.find(params[:id])

    if @role.update(role_params)
      redirect_to admin_role_path(params[:id])
    else
      @role = Role.find(params[:id])
      render :edit, status: :unprocessable_entity
    end
  end

  def destroy
    @role = Role.find_by_id(params[:id])
    @role&.destroy
    redirect_to admin_roles_path
  end

  private

  def role_params
    params.require(:role).permit(:name)
  end
end
end

```

/shop/app/controllers/admin/users_controller.rb

```

# frozen_string_literal: true

module Admin
  class UsersController < AdminsController
    before_action :authorize_admin

    def index
      @users = User.all
    end

    def show
      @user = User.find(params[:id])
    end

    def new
      @user = User.new
    end

    def create
      @user = User.new(user_params)

      if @user.valid?
        @user.save!
        redirect_to admin_user_path(@user.id)
      else
        render :new, status: :unprocessable_entity
      end
    end

    def edit
      @user = User.find(params[:id])
    end

    def update
      @user = User.find(params[:id])
    end
  end
end

```

```

    @user.paper_trail_event = 'update'
    if @user.update(user_params)
      redirect_to admin_user_path(params[:id])
    else
      render :edit, status: :unprocessable_entity
    end
  end

  def destroy
    @user = User.find_by_id(params[:id])
    @user&.destroy
    redirect_to admin_users_path
  end

  private

  def user_params
    params.require(:user).permit(:name, :password, :email, :password_confirmation,
:phone_number)
  end
end
end

```

/shop/app/controllers/categories/brands_controller.rb

```

# frozen_string_literal: true

module Categories
  class BrandsController < ApplicationController
    def index
      @brands = Brand.joins(:brand_categories).where(products: { id:
params[:category_id] }).pluck(:id, :brand_name)
      render json: { brands: @brands }
    end
  end
end

```

/shop/app/mailers/application_mailer.rb

```

# frozen_string_literal: true

class ApplicationMailer < ActionMailer::Base
  default from: 'from@example.com'
  layout 'mailer'
end

```

/shop/app/mailers/order_report_mailer.rb

```

# frozen_string_literal: true

class OrderReportMailer < ApplicationMailer
  def report
    @user = params.fetch(:user)
    @order = params[:order]

    mail(to: @user.email, subject: 'Your order is still accepted!')
  end
end

```

/shop/app/services/baskets_service.rb

```

# frozen_string_literal: true

class BasketsService
  def initialize(session)
    @session = session
    @key = if session['warden.user.user.key']

```

```

        session['warden.user.user.key'][0][0]
      else
        token
      end
    @redis_basket = Redis.current
  end

  def add(id)
    product = Product.find(id)
    products = all

    products << product
    @redis_basket.set(@key, products.to_json)
  end

  def remove(id)
    product = Product.find(id)
    products = all

    products.delete_at(products.index(product.as_json)) if products.include?(product.as_json)
    @redis_basket.set(@key, products.to_json)
  end

  def all
    if @redis_basket.exists?(@key)
      JSON.parse(@redis_basket.get(@key))
    else
      []
    end
  end

  def clear
    @redis_basket.del(@key)
  end

  def size
    if @redis_basket.exists?(@key)
      products = JSON.parse(@redis_basket.get(@key))
      @size = products.length
    else
      @size = 0
    end
  end

  def sum
    products = JSON.parse(@redis_basket.get(@key)) if @redis_basket.exists?(@key)
    sum = 0
    products&.each { |x| sum += x['price'] }
    sum
  end

  private

  attr_reader :session

  def token
    @token ||= @session[:token] || SecureRandom.hex(8)
    @session[:token] = @token
  end
end

```

/shop/app/services/checkouts_service.rb

```

# frozen_string_literal: true

class CheckoutsService
  def initialize(session, order_params)
    @session = session
    @order_params = order_params
    @key = if session['warden.user.user.key']
      session['warden.user.user.key'][0][0]
    else
      session[:token]
    end
  end

  end

```

```

def make_order
  @basket = ::BasketsService.new(@session).all

  build_form_params
  order = Order.create!(@order_params)
  return order.errors unless order.valid?

  order.save
  make_order_product(order)
  if @session['warden.user.user.key'][0][0]
    ::EmailSenderWorker.perform_async(@session['warden.user.user.key'][0][0],
                                      order.id)
  end
end

private

def build_form_params
  @order_params[:order_price] = ::BasketsService.new(@session).sum * 1.2
  @order_params[:user_id] = @key
end

def make_order_product(order)
  @basket.map { |product| OrderProduct.create!(product_id: product['id'], order_id: order.id) }
end
end

```

/shop/app/uploaders/photo_uploader.rb

```

# frozen_string_literal: true

class PhotoUploader < CarrierWave::Uploader::Base
  # Include RMagick or MiniMagick support:
  # include CarrierWave::RMagick
  # include CarrierWave::MiniMagick

  # Choose what kind of storage to use for this uploader:
  storage :file
  # storage :fog

  # Override the directory where uploaded files will be stored.
  # This is a sensible default for uploaders that are meant to be mounted:
  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end

  # Provide a default URL as a default if there hasn't been a file uploaded:
  # def default_url(*args)
  #   # For Rails 3.1+ asset pipeline compatibility:
  #   # ActionController::Base.helpers.asset_path("fallback/" + [version_name,
  "default.png"].compact.join('_'))
  #
  #   "/images/fallback/" + [version_name, "default.png"].compact.join('_')
  # end

  # Process files as they are uploaded:
  # process scale: [200, 300]
  #
  # def scale(width, height)
  #   # do something
  # end

  # Create different versions of your uploaded files:
  # version :thumb do
  #   process resize_to_fit: [50, 50]
  # end

  # Add an allowlist of extensions which are allowed to be uploaded.
  # For images you might use something like this:
  # def extension_allowlist
  #   %w(jpg jpeg gif png)
  # end

  # Override the filename of the uploaded files:

```

```

# Avoid using model.id or version_name here, see uploader/store.rb for details.
# def filename
#   "something.jpg" if original_filename
# end
def filename
  "#{secure_token(10)}.#{file.extension}" if original_filename.present?
end

protected

def secure_token(length = 16)
  var = :@#{mounted_as}_secure_token"
  model.instance_variable_get(var) or model.instance_variable_set(var, SecureRandom.hex(length
/ 2))
end
end

```

/shop/app/workers/email_sender_worker.rb

```

# frozen_string_literal: true

class EmailSenderWorker
  include Sidekiq::Worker

  def perform(user_id, order_id)
    OrderReportMailer.with(user: User.find(user_id), order:
Order.find(order_id)).report.deliver_now
  end
end

```

/shop/app/javascript/utilities/filter.js

```

document.addEventListener("turbolinks:load", function() {
  jQuery(function() {
    return $('#subcategory_select').change(function() {
      const brandSelect = $('#brand_select')[0];
      const categoryId = $('#product_category_id')[0].value;
      $.ajax({
        type: "GET",
        url: "/categories/" + categoryId + "/brands",
        dataType: "json",
        success: function(response) {
          const ajaxVariable = response.brands;
          const oldSelector = $('#category_select')[0];
          if (oldSelector) {
            oldSelector.remove();
          }
          const selectList = document.createElement("select");
          selectList.setAttribute("id", "category_select");
          selectList.setAttribute("name", "product[brand_id]");
          brandSelect.appendChild(selectList);
          $.each(ajaxVariable, function(index, value) {
            $('#category_select')
              .append($('</option>')
                .attr("value", value[0])
                .text(value[1]));
          });
        }
      });
    });
  });
});

```

/shop/config/environments/development.rb

```

# frozen_string_literal: true

require 'active_support/core_ext/integer/time'

Rails.application.configure do
  # Settings specified here will take precedence over those in config/application.rb.

```

```

# In the development environment your application's code is reloaded any time
# it changes. This slows down response time but is perfect for development
# since you don't have to restart the web server when you make code changes.
config.cache_classes = false

# Do not eager load code on boot.
config.eager_load = false

# Show full error reports.
config.consider_all_requests_local = true

# Enable/disable caching. By default caching is disabled.
# Run rails dev:cache to toggle caching.
if Rails.root.join('tmp', 'caching-dev.txt').exist?
  config.action_controller.perform_caching = true
  config.action_controller.enable_fragment_cache_logging = true

  config.cache_store = :redis_cache_store, { url: ENV['REDIS_URL'] }
  config.public_file_server.headers = {
    'Cache-Control' => "public, max-age=#{2.days.to_i}"
  }
else
  config.action_controller.perform_caching = false

  config.cache_store = :redis_cache_store, { url: ENV['REDIS_URL'] }
end

# Store uploaded files on the local file system (see config/storage.yml for options).
config.active_storage.service = :local

# Don't care if the mailer can't send.
config.action_mailer.raise_delivery_errors = false

config.action_mailer.perform_caching = false

config.action_mailer.delivery_method = :smtp
config.action_mailer.perform_deliveries = true

config.action_mailer.smtp_settings = {
  address: 'smtp.gmail.com',
  port: 587,
  domain: 'gmail.com',
  user_name: 'valeronp.com@gmail.com',
  password: 'pass',
  authentication: 'plain',
  enable_starttls_auto: true
}

config.action_mailer.default_url_options = { host: 'localhost:3000' }

# Print deprecation notices to the Rails logger.
config.active_support.deprecation = :log

# Raise exceptions for disallowed deprecations.
config.active_support.disallowed_deprecation = :raise

# Tell Active Support which deprecation messages to disallow.
config.active_support.disallowed_deprecation_warnings = []

# Raise an error on page load if there are pending migrations.
config.active_record.migration_error = :page_load

# Highlight code that triggered database queries in logs.
config.active_record.verbose_query_logs = true

# Debug mode disables concatenation and preprocessing of assets.
# This option may cause significant delays in view rendering with a large
# number of complex assets.
config.assets.debug = true

# Suppress logger output for asset requests.
config.assets.quiet = true

# Raises error for missing translations.
# config.i18n.raise_on_missing_translations = true

# Annotate rendered view with file names.

```



```

# config.action_view.annotate_rendered_view_with_filenames = true

# Use an evented file watcher to asynchronously detect changes in source code,
# routes, locales, etc. This feature depends on the listen gem.
config.file_watcher = ActiveSupport::EventedFileUpdateChecker

# Uncomment if you wish to allow Action Cable access from any origin.
# config.action_cable.disable_request_forgery_protection = true
end

```

/shop/config/routes.rb

```

# frozen_string_literal: true

require 'sidekiq/web'

Rails.application.routes.draw do
  # devise_for :users, controllers: {registrations: 'registrations', sessions: 'sessions'}
  devise_for :users, controllers: { sessions: 'sessions' }
  mount Sidekiq::Web => '/sidekiq'

  namespace :admin do
    resources :users do
      collection do
        end
      end

    resources :products do
      collection do
        post 'add/:id', to: 'products#add_to_basket'
        post 'delete/:id', to: 'products#delete_from_basket'
        get 'add/:id', to: 'products#index'
        get 'delete/:id', to: 'products#index'
      end
    end

    resources :orders do
      collection do
        end
      end

    resources :roles do
      collection do
        end
      end
    end

  get '/registration', to: 'registration#new'
  post '/registration', to: 'registration#registration'
  get '/login', to: 'authentication#new'
  post '/login', to: 'authentication#create'
  get '/logout', to: 'authentication#destroy'

  get '/basket', to: 'baskets#index'
  post '/basket/add/:id', to: 'baskets#add'
  post '/basket/delete/:id', to: 'baskets#remove'
  get '/basket/add/:id', to: 'baskets#index'
  get '/basket/delete/:id', to: 'baskets#index'
  post '/basket/clear', to: 'baskets#clear'

  get '/checkout', to: 'checkout#new'
  post '/checkout', to: 'checkout#create'

  get '/products/:id', to: 'products#show'
  post 'add/:id', to: 'products#add_to_basket'
  post 'delete/:id', to: 'products#delete_from_basket'
  get 'add/:id', to: 'products#index'
  get 'delete/:id', to: 'products#index'

  root 'products#index'
end

```

/shop/config/puma.rb

```
# frozen_string_literal: true

# Puma can serve each request in a thread from an internal thread pool.
# The `threads` method setting takes two numbers: a minimum and maximum.
# Any libraries that use thread pools should be configured to match
# the maximum value specified for Puma. Default is set to 5 threads for minimum
# and maximum; this matches the default thread size of Active Record.
#
max_threads_count = ENV.fetch('RAILS_MAX_THREADS', 5)
min_threads_count = ENV.fetch('RAILS_MIN_THREADS') { max_threads_count }
threads min_threads_count, max_threads_count

# Specifies the `worker_timeout` threshold that Puma will use to wait before
# terminating a worker in development environments.
#
worker_timeout 3600 if ENV.fetch('RAILS_ENV', 'development') == 'development'

# Specifies the `port` that Puma will listen on to receive requests; default is 3000.
#
port ENV.fetch('PORT', 3000)

# Specifies the `environment` that Puma will run in.
#
environment ENV.fetch('RAILS_ENV', 'development')

# Specifies the `pidfile` that Puma will use.
pidfile ENV.fetch('PIDFILE', 'tmp/pids/server.pid')

# Specifies the number of `workers` to boot in clustered mode.
# Workers are forked web server processes. If using threads and workers together
# the concurrency of the application would be max `threads` * `workers`.
# Workers do not work on JRuby or Windows (both of which do not support
# processes).
#
# workers ENV.fetch("WEB_CONCURRENCY") { 2 }

# Use the `preload_app!` method when specifying a `workers` number.
# This directive tells Puma to first boot the application and load code
# before forking the application. This takes advantage of Copy On Write
# process behavior so workers use less memory.
#
# preload_app!

# Allow puma to be restarted by `rails restart` command.
plugin :tmp_restart
```

/shop/config/database.yml

```
# SQLite. Versions 3.8.0 and up are supported.
#   gem install sqlite3
#
#   Ensure the SQLite 3 gem is defined in your Gemfile
#   gem 'sqlite3'
#
default: &default
  adapter: postgresql
  pool: <%= ENV.fetch("RAILS_MAX_THREADS") { 5 } %>
  username: <%= 'postgres' %>
  password: <%= '12345678' %>
  timeout: 5000

development:
  <<: *default
  database: postgres

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  <<: *default
  database: postgres
```

```
production:
  <<: *default
  database: postgres
```

/shop/config/application.rb

```
# frozen_string_literal: true

require_relative 'boot'

require 'rails/all'

# Require the gems listed in Gemfile, including any gems
# you've limited to :test, :development, or :production.
Bundler.require(*Rails.groups)

module Shop
  class Application < Rails::Application
    # Initialize configuration defaults for originally generated Rails version.
    config.load_defaults 6.1

    # Configuration for the application, engines, and railties goes here.
    #
    # These settings can be overridden in specific environments using the files
    # in config/environments, which are processed later.
    #
    # config.time_zone = "Central Time (US & Canada)"
    # config.eager_load_paths << Rails.root.join("extras")
  end
end
```

/shop/Gemfile

```
# frozen_string_literal: true

source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }

ruby '2.6.3'

# Bundle edge Rails instead: gem 'rails', github: 'rails/rails', branch: 'main'
gem 'rails', '~> 6.1.3', '>= 6.1.3.2'
# Use sqlite3 as the database for Active Record
gem 'sqlite3', '~> 1.4'
# Use Puma as the app server
gem 'puma', '~> 5.0'
# Use SCSS for stylesheets
gem 'sass-rails', '>= 6'
# Transpile app-like JavaScript. Read more: https://github.com/rails/webpacker
gem 'webpacker', '~> 5.0'
# Turbolinks makes navigating your web application faster. Read more:
https://github.com/turbolinks/turbolinks
gem 'turbolinks', '~> 5'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilder
gem 'jbuilder', '~> 2.7'
# Use Redis adapter to run Action Cable in production
# gem 'redis', '~> 4.0'
# Use Active Model has_secure_password
gem 'bcrypt', '~> 3.1.7'

# Use Active Storage variant
# gem 'image_processing', '~> 1.2'

# Reduces boot times through caching; required in config/boot.rb
gem 'bootsnap', '>= 1.4.4', require: false

group :development, :test do
  # Call 'byebug' anywhere in the code to stop execution and get a debugger console
  gem 'byebug', platforms: %i[mri mingw x64_mingw]
end

group :development do
  # Access an interactive console on exception pages or by calling 'console' anywhere in the
```

```

code.
  gem 'web-console', '>= 4.1.0'
  # Display performance information such as SQL time and flame graphs for each request in your
  browser.
  # Can be configured to work on production as well see: https://github.com/MiniProfiler/rack-
  mini-profiler/blob/master/README.md
  gem 'listen', '~> 3.3'
  gem 'rack-mini-profiler', '~> 2.0'
  # Spring speeds up development by keeping your application running in the background. Read
  more: https://github.com/rails/spring
  gem 'spring'
end

group :test do
  # Adds support for Capybara system testing and selenium driver
  gem 'capybara', '>= 3.26'
  gem 'selenium-webdriver'
  # Easy installation and use of web drivers to run system tests with browsers
  gem 'webdrivers'
end

# Windows does not include zoneinfo files, so bundle the tzinfo-data gem
gem 'pg', '~> 1.2', '>= 1.2.3'
gem 'simple_form'
gem 'tzinfo-data', platforms: %i[mingw mswin x64_mingw jruby]

gem 'rolify'

gem 'carrierwave', '~> 2.0'

gem 'pry', '~> 0.13.1'

gem 'hiredis'
gem 'redis', '~> 4.3.1'

gem 'letter_opener', group: :development
gem 'sidekiq'

group :development, :test do
  gem 'rspec-rails', '~> 5.0.0'
  gem 'rubocop', require: false
end

gem 'rails-controller-testing'

gem 'devise'

gem 'paper_trail'

```

/shop/db/schema.rb

```

# frozen_string_literal: true

# This file is auto-generated from the current state of the database. Instead
# of editing this file, please use the migrations feature of Active Record to
# incrementally modify your database, and then regenerate this schema definition.
#
# This file is the source Rails uses to define your schema when running `bin/rails
# db:schema:load`. When creating a new database, `bin/rails db:schema:load` tends to
# be faster and is potentially less error prone than running all of your
# migrations from scratch. Old migrations may fail to apply correctly if those
# migrations use external dependencies or application code.
#
# It's strongly recommended that you check this file into your version control system.

ActiveRecord::Schema.define(version: 20_210_708_140_916) do
  # These are extensions that must be enabled in order to support this database
  enable_extension 'plpgsql'

  create_table 'order_products', force: :cascade do |t|
    t.bigint 'product_id'
    t.bigint 'order_id'
    t.datetime 'created_at', precision: 6, null: false
    t.datetime 'updated_at', precision: 6, null: false
    t.index ['order_id'], name: 'index_order_products_on_order_id'
  end
end

```

```

    t.index ['product_id'], name: 'index_order_products_on_product_id'
  end

  create_table 'orders', force: :cascade do |t|
    t.bigint 'user_id'
    t.integer 'order_price'
    t.string 'dest_address'
    t.string 'name'
    t.integer 'phone_number'
    t.datetime 'created_at', precision: 6, null: false
    t.datetime 'updated_at', precision: 6, null: false
    t.index ['user_id'], name: 'index_orders_on_user_id'
  end

  create_table 'products', force: :cascade do |t|
    t.string 'name'
    t.text 'description'
    t.string 'photo'
    t.integer 'price'
    t.datetime 'created_at', precision: 6, null: false
    t.datetime 'updated_at', precision: 6, null: false
  end

  create_table 'roles', force: :cascade do |t|
    t.string 'name'
    t.string 'resource_type'
    t.bigint 'resource_id'
    t.datetime 'created_at', precision: 6, null: false
    t.datetime 'updated_at', precision: 6, null: false
    t.index %w[name resource_type resource_id], name:
'index_roles_on_name_and_resource_type_and_resource_id'
    t.index %w[resource_type resource_id], name: 'index_roles_on_resource'
  end

  create_table 'users', force: :cascade do |t|
    t.string 'name'
    t.integer 'phone_number'
    t.datetime 'created_at', precision: 6, null: false
    t.datetime 'updated_at', precision: 6, null: false
    t.string 'email', default: '', null: false
    t.string 'encrypted_password', default: '', null: false
    t.string 'reset_password_token'
    t.datetime 'reset_password_sent_at'
    t.datetime 'remember_created_at'
    t.index ['email'], name: 'index_users_on_email', unique: true
    t.index ['reset_password_token'], name: 'index_users_on_reset_password_token', unique: true
  end

  create_table 'users_roles', id: false, force: :cascade do |t|
    t.bigint 'user_id'
    t.bigint 'role_id'
    t.index ['role_id'], name: 'index_users_roles_on_role_id'
    t.index %w[user_id role_id], name: 'index_users_roles_on_user_id_and_role_id'
    t.index ['user_id'], name: 'index_users_roles_on_user_id'
  end

  create_table 'versions', force: :cascade do |t|
    t.string 'item_type', null: false
    t.bigint 'item_id', null: false
    t.string 'event', null: false
    t.string 'whodunnit'
    t.text 'object'
    t.datetime 'created_at'
    t.index %w[item_type item_id], name: 'index_versions_on_item_type_and_item_id'
  end

  add_foreign_key 'order_products', 'orders'
end

```

ПРИЛОЖЕНИЕ Б
(обязательное)

Спецификация программного дипломного проекта

ПРИЛОЖЕНИЕ В
(обязательное)

Ведомость документов

[illegible]