

Содержание

Введение.....	3
1 Постановка задачи.....	4
1.1 Обзор прототипов	4
1.1.1 Веб приложение МОЙЭКОДВОР.РФ.....	4
1.1.2 Веб приложение RSBOR.RU.....	5
1.1.3 Веб приложение TAGRET99.BY	5
1.2 Обзор прототипов	6
2 Проектирование программного средства	8
2.1 Проектирование архитектуры программного средства	8
2.2 Проектирование серверной части	9
2.3 Проектирование клиентской части	10
2.4 Проектирование базы данных	11
3 Разработка программного средства.....	13
3.1 Разработка серверной части.....	13
3.1.1 Файловая структура, файл server.js	13
3.1.2 Контроллеры и маршрутизаторы	14
3.1.3 Взаимодействие с базой данных	16
3.2 Разработка клиентской части.....	17
3.2.1 Настройка React JS приложения, файл App.js	17
3.2.2 Компоненты	18
3.3 Взаимодействие серверной и клиентской частей.....	19
3.4 Использование Docker.....	20
4 Тестирование веб-приложения	21
4.1 Тестирование формы входа	21
4.2 Тестирование формы регистрация	23
5 Руководство пользователя.....	26
5.1 Методика установки	26
5.2 Руководство пользователя.....	27
5.2.1 Руководство пользователя для роли «гость»	27
5.2.2 Руководство пользователя для роли «пользователь».....	28
5.2.3 Руководство пользователя для роли «администратор»	31
Заключение	33
Список используемых источников.....	34
ПРИЛОЖЕНИЕ А	35
ПРИЛОЖЕНИЕ Б.....	36
ПРИЛОЖЕНИЕ В	37
ПРИЛОЖЕНИЕ Г.....	38
ПРИЛОЖЕНИЕ Д	39

Введение

Сегодня каждый белорус выбрасывает примерно 350 кг бытовых отходов за год. Это в два раза больше, чем 15 лет назад. К тому же состав мусора кардинально изменился. Еще совсем недавно домашняя мусорная корзина содержала в основном пищевые и бумажные отходы, старая одежда была из натуральных тканей, а техника в мусоре почти не оказывалась.

Современный бытовой мусор - это разнообразный пластик, искусственные материалы, из которых сделана упаковка, одежда, домашние вещи, батарейки, электроника. При захоронении эти вещи разлагаются сотни лет, а может, и больше, выделяя токсичные вещества в почву, воду и воздух. Полиэтиленовый пакет, который послужил вам 20 минут, останется на мусорном полигоне и через 100 лет. При этом многие бытовые отходы можно переработать, пустить в дело и благодаря этому сократить объемы захоронения мусора. И к тому же сэкономить природные ресурсы. Переработка отходов - это необходимость сегодня.

На производство товаров тратятся ресурсы: нефть, древесина, чистая вода, топливо для перевозки и электричества. Раздельный сбор даёт возможность этим ресурсам не пропадать. Для системы раздельного сбора почти все отходы — это вторичное сырьё, которое можно ещё не раз использовать для производства новой продукции.

Раздельный сбор положительно влияет на образ жизни людей. Вы:

- адекватно оцениваете свои расходы;
- меньше подвержены спонтанным покупкам;
- экономите деньги на том, что на самом деле вам не нужно, но что навязано быстрой модой;
- четко видите, что вы покупаете, сколько тратится на упаковку.

Тема раздельного сбора бытовых отходов в последние года становится все более актуальной. Но, к большому сожалению, в нашей стране данной проблеме уделяется недостаточно внимания. В западных странах в каждом доме и квартире имеются необходимые резервуары для раздельного сбора мусора.

Цель разработки курсового проекта – создание сервиса, способствующего популяризации сортировки бытовых отходов.

Задачи, решение которых поможет достичь цели:

- определить обязательный функционал приложения;
- согласно требованиям спроектировать составляющие приложения;
- разработать приложение, согласно требованиям;
- протестировать функционал приложения, на наличие ошибок;
- составить руководство пользования для приложения.

1 Постановка задачи

Главная задача программного средства состоит в том, чтобы заинтересовать пользователя начать сортировать бытовые отходы и сдавать их в специальные пункты приема, помимо этого пользователи могут публиковать статьи с советами по сбору и сортировке отходов. Администратор может контролировать процессы, происходящие в приложении, может редактировать и удалять любые статьи. Приложения такого плана присутствуют на просторах Интернета. Наличие аналогов в данной сфере предоставляет пользователям возможность узнавать нужную информацию из разных источников.

1.1 Обзор прототипов

На сегодняшний день с легкостью можно найти ресурсы где так или иначе занимаются информированием или привлечением к раздельному сбору мусора. Для обзора схожих по тематике ресурсов были выбраны следующие веб-приложения:

- МОЙЭКОДВОР.РФ [1];
- *RSBOR.RU* [2];
- *TAGRET99.BY* [3];

1.1.1 Веб приложение МОЙЭКОДВОР.РФ

Рассматриваемый интернет-ресурс представляет собой сайт, главная страница которого отображена на рисунке 1.1.

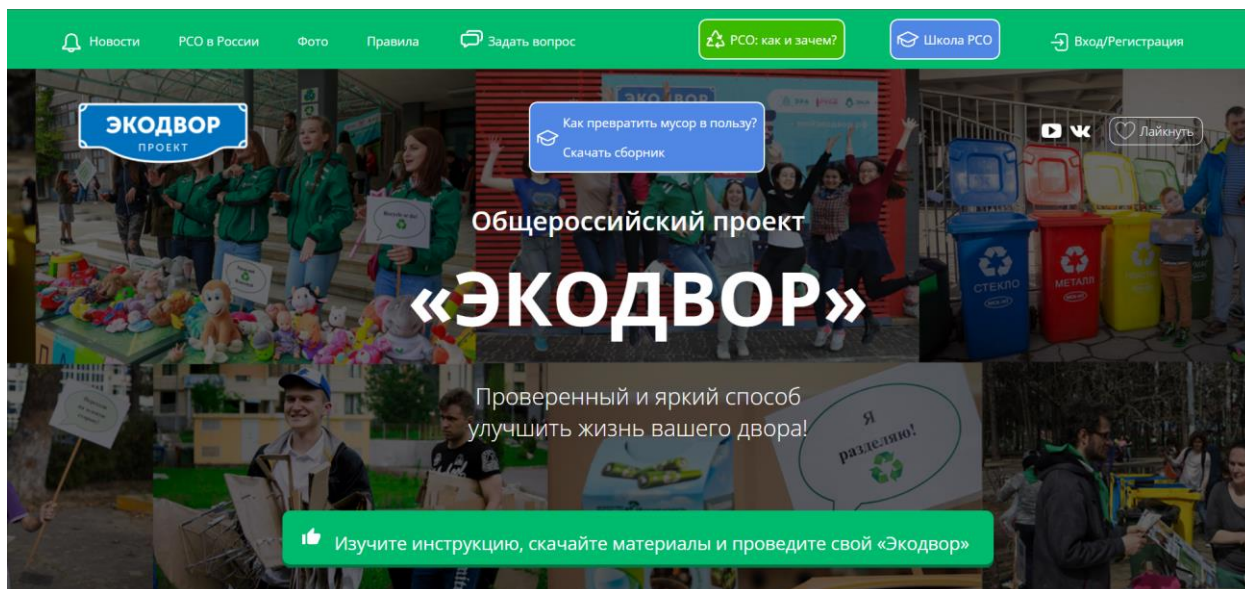


Рисунок 1.1 – Главная страница веб-приложения МОЙЭКОДВОР.РФ

Анализируемой веб-приложение позволяет пользователям ознакомиться с рекомендациями по раздельному сбору отходов, просмотреть видео уроки под названием «Школа PCO». Просмотреть фото с событий, проводимых

участниками «Экодвор». И так же можно стать частью этого сообщества. И иметь возможность проводить свой «Экодвор».

1.1.2 Веб приложение RSBOR.RU

Следующим результатом поиска аналогов стал сайт RSBOR.RU, главная страница которого представлена на рисунке 1.2.

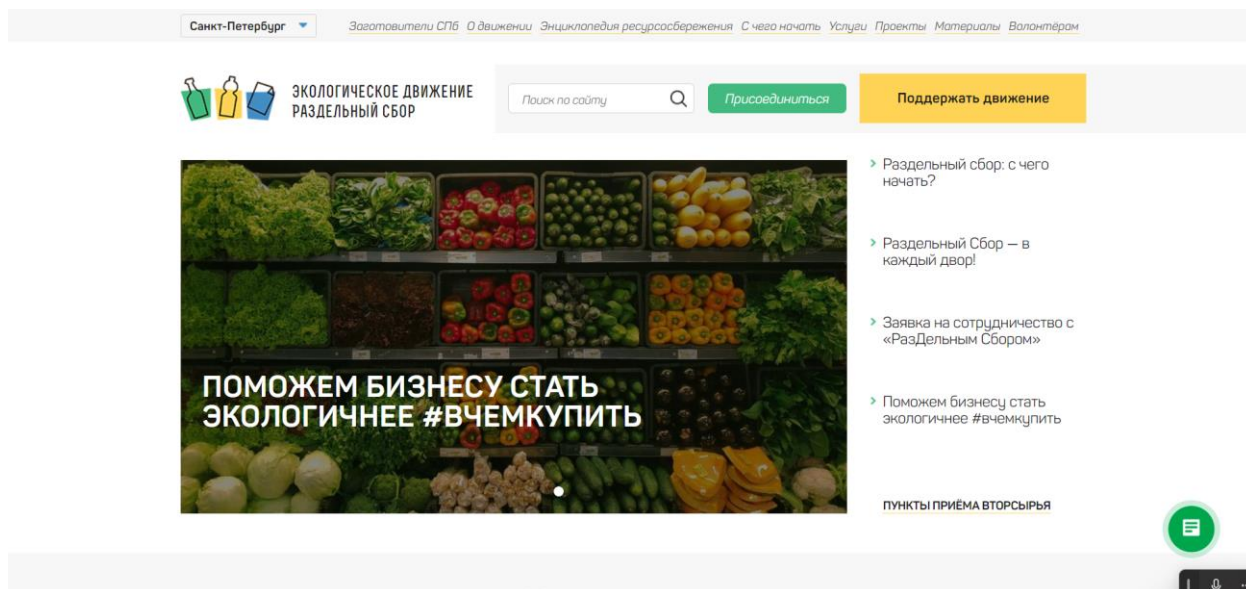


Рисунок 1.2 – Главная страница сайта RSBOR.RU

На данном интернет-ресурсе представлены последние новости в сфере сбора, разделения и переработки отходов и вторсырья в России. Здесь же можно ознакомиться с зарубежным опытом переработки отходов и ресурсосбережении. Также на сайте имеется возможность увидеть проекты, которые проводит Экологическое Движение Раздельный Сбор.

Для тех, кто желает поддержать движение представлена возможность сделать пожертвования для развития движения.

Так же есть возможность скачать различные материалы для пропаганды заботы об окружающей среде.

Еще одним важным шагом в развитии РазДельного Сбора является предоставленная возможность стать волонтером. Для этого нужно нажать кнопку “Присоединиться” и заполнить анкету.

1.1.3 Веб приложение TAGRET99.BY

После первых двух приложений я обратилась к белорусскому ресурсу государственного учреждения «Оператор вторичных материальных ресурсов», уполномоченного министерством жилищно-коммунального хозяйства Республики Беларусь. Для большего понимания какое развитие имеет ситуация с вторсырьем в нашей стране. Данный веб-ресурс представлен на рисунке 1.2.

Следить за движением Цель 99 можно через многие социальные сети такие как: *Telegram*, *Instagram*, ВКонтакте, *Facebook* и на *YouTube* канале. Для связи есть многоканальный номер телефона.

Основная информация, представленная на главной странице сайта – это правила сортировки вторичных ресурсов. Для удобства правила разделены по категориям.

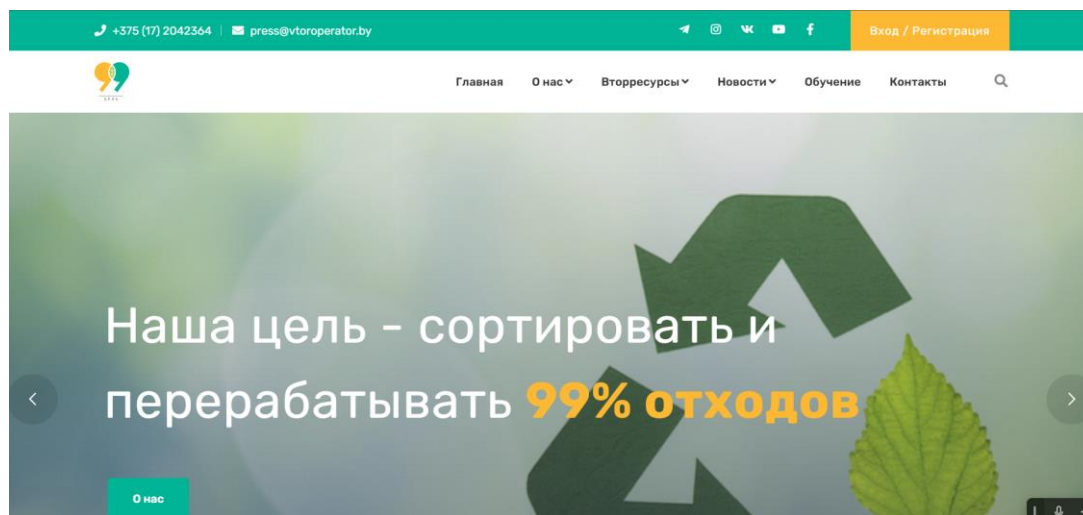


Рисунок 1.3 – Страница сайта TAGRET99.BY

Важный момент для развития отдельного выброса мусора – это подробная карта пунктов приема вторичного сырья. В Беларуси сегодня более 1700 приемных пунктов и постоянно пополняется новыми объектами.

Калькулятор экологического эффекта – еще один полезный раздел для тех, кто хочет знать о пользе переработки в цифрах.

Так же на ресурсе можно ознакомиться с последними новостями о переработке отходов и экологичном стиле жизни.

1.2 Обзор прототипов

Рассмотренные выше аналоги, программного средства помогли в составлении и определении средств и технологий, благодаря которым разработка приложения была оптимизирована и сконцентрирована на решении поставленной задачи.

Технологии, используемые при разработке курсового проекта, представлены в таблице 1.1.

Таблица 1.1 – Описание технологий

Название технологии	Описание
<i>MySQL</i> [4]	система управления реляционными базами данных (<i>RDBMS</i>), разработанная <i>Oracle</i> и основанная на языке структурированных запросов (<i>SQL</i>). <i>SQL</i> — это универсальный язык, который поддерживают все системы управлением базами данных.

Окончание таблицы 1.1

Название технологии	Описание
<i>Sequelize</i> [5]	<i>ORM-библиотека</i> (англ. <i>Object-Relational Mapping</i> , рус. объектно-реляционное отображение, или преобразование) для приложений на <i>Node.js</i> , которая осуществляет сопоставление таблиц в базе данных и отношений между ними с классами. При использовании <i>Sequelize</i> мы можем не писать <i>SQL</i> -запросы, а работать с данными как с обычными объектами.
<i>Express JS</i> [6]	Базовая платформа веб-приложений. Является структурой веб-приложения для <i>Node JS</i> . <i>Express JS</i> упрощает написание кода сервера, так как имеет в себе ряд готовых функций. Предназначена для создания надежных веб-приложений и <i>API</i> . Известна своей быстрой скоростью и минималистской структурой, многие функции доступны в виде плагинов.
<i>React JS</i> [7]	Декларативная, высокоэффективная и гибкая <i>JavaScript</i> библиотека для создания пользовательских интерфейсов. Она позволяет собирать сложный <i>UI</i> из маленьких изолированных кусочков кода, называемых «компонентами». Предоставляет высокую скорость, простоту и масштабируемость. К особенностям можно отнести однонаправленную модель передачи данных, виртуальный <i>DOM</i> , <i>JSX</i> , <i>React Hooks</i> , динамическую верстку.
<i>JavaScript</i> [8]	Текстовый язык программирования, который позволяет создавать интерактивные элементы на веб-странице.
<i>Node JS</i> [9]	Программная платформа, основанная на движке <i>Chrome V8</i> . <i>Node JS</i> асинхронен и событийно-ориентирован. Данная платформа позволяет <i>JavaScript</i> взаимодействовать с устройствами ввода-вывода через свой <i>API</i> , подключать другие внешние библиотеки. <i>Node JS</i> предназначен для построения масштабируемых сетевых приложений, преимущественно серверов.
<i>NPM</i> [10]	Менеджер пакетов, входящий в состав <i>Node JS</i> , позволяющий расширить возможности приложения.
<i>Body-parser</i> [11]	Библиотека, которая позволяет парсить тела запросов.
<i>JsonWebToken</i> [12]	Библиотека для создания токенов доступа, основанных на формате <i>JSON</i> .
<i>HTML</i> [13]	Стандартизированный язык разметки документов, на котором представлены большинство веб-страниц в сети <i>Internet</i> .
<i>CSS</i> [14]	Язык, отвечающий за визуальное представление документов пользователю.

Разрабатываемое программное средство строится на клиент-серверной архитектуре. В качестве клиента используется приложение написанное на *React JS*, которое отображается в браузере. Для сервера использовалась платформа *Node JS*, благодаря своей кроссплатформенности, простому разворачиванию сервера и гибкости работы за счет менеджера пакетов *NPM*.

2 Проектирование программного средства

Разработка архитектуры проекта – важная задача в процессе работы над приложением, потому что в зависимости от неё определяется уровень связности между компонентами приложения, и насколько легко можно будет это приложение расширить. Также необходимо учитывать множество факторов, влияющих на работоспособность разрабатываемого веб-приложения. Современные приложения должны иметь интуитивно понятный и удобный интерфейс, хорошую функциональность; должны быть просты и гибки в использовании, оптимизированы и в большинстве случаев автоматизированы.

Для создания программного средства была выбрана среда разработки *Visual Studio Code* – это один из наиболее популярных редакторов кода, разработанный корпорацией *Microsoft*. Он распространяется в бесплатном доступе и поддерживается всеми актуальными операционными системами: *Windows*, *Linux* и *macOS*. *VS Code* представляет собой обычный текстовый редактор с возможностью подключения различных плагинов, что дает возможность работать со всевозможными языками программирования для разработки любого ИТ-продукта.

Основная задача в ходе реализации курсового проекта является создание сервиса, позволяющего пользователям узнать больше и заинтересоваться сортировкой своих же бытовых отходов, а администратору – управлять внутренними процессами данного программного средства. Таким образом, было принято решение создать три роли: гость, пользователь, администратор. Каждая из этих ролей имеет ряд функций:

- гость – может только просмотреть статьи, опубликованные пользователями и прочитать комментарии;
- пользователь – имеет возможность просматривать все статьи, публиковать, удалять и редактировать свои статьи, оценивать статьи, комментировать, отправлять в базу данных приложения количество сданных отходов и получать скидку от партнеров приложения, а так же получать информацию сколько новой продукции будет произведено из переработанных отходов;
- администратор – также имеет возможность публиковать, просматривать, оценивать и комментировать все статьи, но так же имеет доступ удалить и изменить любую статью, добавить новую точку сбора, добавить новые скидки и просмотреть все скидки.

Функциональные возможности ролей в разработанном программном средстве представлены в виде диаграммы вариантов использования в приложении А.

2.1 Проектирование архитектуры программного средства

Программное средство, разработанное в ходе выполнения курсовой работы, является представителем приложений с архитектурой «клиент-сервер». Логика, отвечающая за обработку данных, а именно создание, удаление,

обновление и получение записей из базы данных расположена на сервере в виде файлов контроллеров и их методов, написанных на языке *JavaScript*.

Клиент реализован при помощи библиотеки *React JS*, которая позволяет собирать сложный *UI* из изолированных кусочков кода – называемых компонентами. Компоненты представляют собой *HTML* разметку, которая применяет *CSS*. Цель *CSS* – предоставить высокую скорость, простоту и масштабируемость. К особенностям можно отнести однонаправленную модель передачи данных, виртуальный *DOM*, *JSX*, *React Hooks*, динамическую верстку.

Взаимодействие клиента с сервером происходит посредством использования протокола *HTTP*[20]. *HTTP* – это протокол прикладного уровня передачи данных различного формата. Клиент инициирует *GET* запрос нажатием на какой-либо пункт в навигационном меню и сервер *Node JS* генерирует соответствующий ответ, основываясь на *URI* запроса. Межкомпонентное взаимодействие представлено в виде *MVC*, где *model* – это объекты *MS SQL*, *controller* – контроллеры сервера *Node JS*, а *view* – компоненты *React JS*.

Как только клиент инициирует какой-либо сценарий, то посредством использования *HTTP* происходит передача данных контроллеру на стороне сервера, который осуществляет взаимодействие с базой данных. После завершения работы управление возвращается клиенту.

Для наглядного понимания спроектированных частей проекта была построена диаграмма компонентов и развертывания в приложении Б.

2.2 Проектирование серверной части

При проектировании структуры сервера важно сделать так, чтоб она была читабельной и легко расширяемой. Для данных целей подходит архитектура *MVC*.

Controller выполняет функцию обработчика запросов. Каждый файл контроллера представляет собой *JS*-модуль, содержащий функции, которые обрабатывают запрос. В таблице 2.1 представлены и описаны все контроллеры, разработанные в курсовой работе.

Таблица 2.1 – Описание контроллеров

Название контроллеров	Описание
<i>ArticlesController.js</i>	Предназначен для функций, обеспечивающих работу со статьями.
<i>AuthController.js</i>	Функции предназначены для аутентификации, регистрации пользователей и при выполнении этих функций выполняется формирование токенов.
<i>DiscountsController.js</i>	Предназначен для функций, обеспечивающих работу со скидками, а в частности создание, удаление, изменение и вывод их для пользователя.

Окончание таблицы 2.1

Название контроллера	Описание
----------------------	----------

<i>MarksController.js</i>	Предназначен для управления данным по количеству начисления баллов за сдачу отходов и произведенной новой продукции.
<i>PointsController.js</i>	Предназначен для вывода данных по точкам сбора отходов.
<i>RatingsController.js</i>	Используется для управления комментариями.
<i>ReceptionController.js</i>	Функция применяется для начисления баллов и вывода количества произведенной новой продукции из сданных отходов. Данная функция представлена в приложении В.

Маршрутизация в приложении организована с помощью роутеров, которые используют технологию *Express JS*. Маршрутизация определяет, как приложение отвечает на клиентский запрос к конечной точке, которым является *URI*, и определенному методу запроса *HTTP (GET, POST, DELETE)*. Для каждого из контроллеров был разработан собственный маршрутизатор. Он представляет собой *JS*-модуль, который в зависимости от контроллера будет передаваться в *middleware* сервера с указанием первого *URL*-параметра.

В таблице 2.2 представлены и описаны все разработанные маршрутизаторы.

Таблица 2.2 – Описание маршрутизаторов

Название маршрутизатора	Описание
<i>ArticlesRouter.js</i>	Роутер, который перенаправляет запрос, приходящий по пути <i>/Articles</i> .
<i>AuthRouter.js</i>	Перенаправляет запросы, приходящие по пути <i>/login</i> , <i>/register</i> , <i>/logout</i> , <i>/me</i> .
<i>DiscountsRouter.js</i>	Маршрутизатор обрабатывает запросы по <i>URL /Discounts</i> .
<i>MarksRouter.js</i>	Роутер, перенаправляет запрос приходящий с <i>/Marks</i> .
<i>PointsRouter.js</i>	Перенаправляет запрос, приходящий по пути <i>/Points</i> .
<i>RatingsRouter.js</i>	Маршрутизатор обрабатывает запросы по <i>URL /Ratings</i> .
<i>ReceptionsRouter.js</i>	Перенаправляет запросы, приходящие по пути <i>/Receptions</i>

Вся серверная часть представляет собой веб-приложение, предоставляющее интерфейс для взаимодействия с ним клиентского приложения. Для корректной работы сервера используются контроллеры и маршрутизаторы, которые вместе предоставляют полную обработку и перенаправление приходящих запросов от клиента.

2.3 Проектирование клиентской части

Клиентская часть веб-приложения реализована с помощью *React JS*. Для разработки пользовательских интерфейсов *React JS* часто используется с другими библиотеками.

Структура клиентского приложения представлена на рисунке 2.1

Директорий *components* содержит все компоненты, используемые в приложении, например, компоненты форм, модальных окон;

Директорий *pages* содержит все страницы, используемые в приложении;

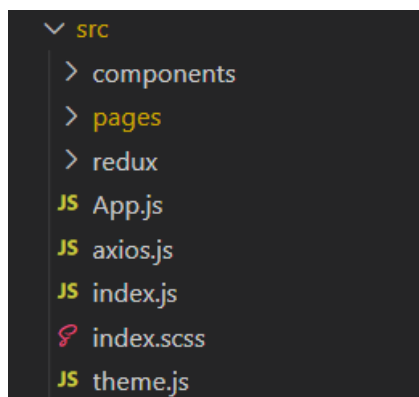


Рисунок 2.2 – Структура клиентского приложения

Директорий *redux* содержит все объекты *Redux*, используемые в приложении;

Остальные файлы, представленные на рисунке выше, являются базовыми и создаются при инсталляции проекта *React JS*.

2.4 Проектирование базы данных

В качестве базы данных в проекте используется MySQL. Для реализации функционала приложения была разработана база данных. Для создания базы данных была выбрана система управления базами данных MySQL. Текущая база данных состоит из 7 таблиц, схема базы данных продемонстрирована в приложении Г.

Таблица *users* хранит информацию о пользователях, содержит следующие поля:

- *id*, уникальное поле в котором указывается идентификатор пользователя;
- *username*, хранится имя пользователя;
- *email*, хранится почта пользователя;
- *passwordHash*, хранится пароль пользователя в захешированном виде
- *points*, хранятся количество начисленных баллов;
- *avatarUrl*, хранится адрес на фото пользователя;
- *role*, хранится роль пользователя.

Таблица *articles* хранит в себе статьи и информацию, предназначенную для них, содержит следующие поля:

- *id*, уникальное поле в котором указывается идентификатор статьи;
- *Title*, название статьи;
- *Text*, текст статьи;
- *DatePub*, дата публикации статьи;
- *ImageU*, картинка, относящаяся к статье;
- *Author*, автор статьи;
- *Like*, количество лайков у статьи.

Таблица *ratings* хранит в себе комментарии к статьям, содержит следующие поля:

- *id*, уникальное поле в котором указывается идентификатор комментария;
- *Item*, идентификатор статьи к которой относится комментарий;
- *Commentator*, идентификатор пользователя который оставил комментарий;
- *Comment*, текст комментария.

Таблица *points* хранит в себе пункты сдачи отходов, содержит следующие поля:

- *id*, уникальное поле в котором указывается идентификатор точки сдачи отходов;
- *Address*, адрес точки сдачи отходов;
- *SecretKey*, хранится секретный ключ относящийся к точке сдачи отходов в захешированном виде.

Таблица *receptions* хранит информацию по сдаче отходов, содержит следующие поля:

- *id*, уникальное поле в котором указывается идентификатор приема;
- *idUser*, идентификатор пользователя, который сдал отходы;
- *Weight*, сколько по весу было сдано отходов в кг;
- *Accrued*, сколько начислилось баллов;
- *NewKg*, сколько новой продукции будет произведено из сданных отходов в кг;
- *TypeWaste*, тип сданных отходов;
- *StationKey*, секретный ключ в захешированном виде, предоставленный пользователем.

Таблица *discounts* хранит в себе скидки, содержит следующие поля:

- *id*, уникальное поле в котором указывается идентификатор скидки;
- *Discount*, скидка;
- *PointD*, сколько нужно баллов для получения скидки.

Таблица *marks* хранит в данные по «цене» и количеству произведенной новой продукции, содержит следующие поля:

- *id*, уникальное поле в котором указывается идентификатор;
- *Rubbish*, вид отхода
- *PointsOKg*, сколько баллов начисляется за один кг сданных отходов;
- *NewOKg*, сколько новой продукции будет произведено из одного кг сданных отходов.

3 Разработка программного средства

Разрабатываемое приложение включает в себя две части: серверную и клиентскую. Для разработки каждой из них использован определенный набор технологий, включающий различные библиотеки, механизмы, необходимые для полноценной работы проекта.

3.1 Разработка серверной части

Серверная часть приложения написана на языке JavaScript, использует такие технологии как Node JS, Express JS, Sequelize, NPM, Body-parser, Crypto, JWT. Все указанные технологии описаны в первой главе пояснительной записки. Они помогают в быстром и эффективном написании серверной части проекта.

3.1.1 Файловая структура, файл server.js

Для начала опишем файловую структуру разрабатываемого сервера. На рисунке 3.1 отображены все составляющие серверной части, которые взаимодействуют друг с другом и в целом представляют собой хорошо разработанный сервер, который имеет главный файл server.js, ряд контроллеров и маршрутизаторов, выполняющих основную функциональность программного средства.

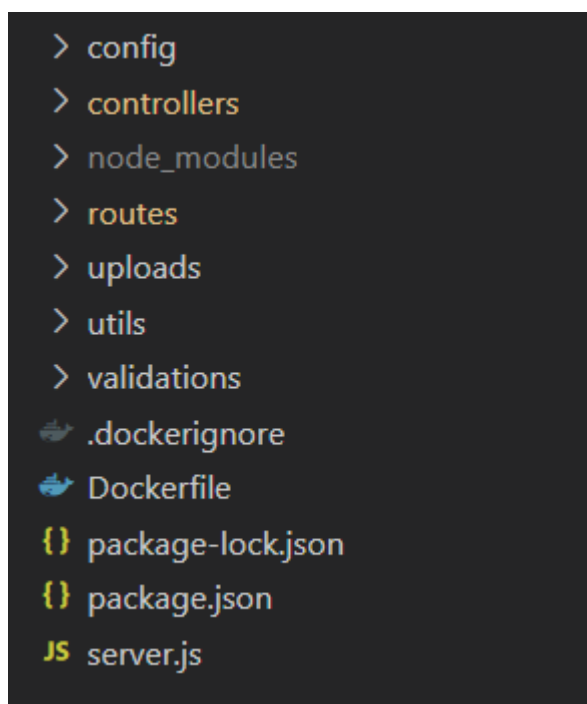


Рисунок 3.1 – Структура серверной части

Проект сервера состоит из следующих директорий:

- *config*, содержит подключение к базе данных и модели базы данных, которые с использованием *Sequelize* упрощают работу с *MySQL*;
- *controllers*, содержит все разработанные контроллеры;
- *node_modules*, содержит модули или библиотеки, которые используются при разработке серверной части;
- *route*, хранит все разработанные файлы маршрутизаторов;
- *uploads*, хранит загруженные картинки;
- *utils*, хранит функции проверки доступа;
- *validations*, хранит функции валидации.

На рисунке 3.1 также отображены два файла: *package.json* и *index.js*. Данные файлы являются главными при разработке сервера.

Файл *package.json* тесно связан с такой технологией как *Node JS*. Он хранит информацию о проекте: название, версию, список пакетов, необходимых для корректной работы проекта. Любая директория, в которой есть этот файл, интерпретируется как *Node JS* пакет.

Файл *server.js* является главным в серверной части. В нем происходит подключение всех модулей и библиотек, которые используются в приложении. Запуск сервера происходит при вызове функции *listen*. В этом файле присутствует код, представленный на листинге 3.1.

```
let AuthRouter = require('./routes/AuthRouter');
let ArticlesRouter = require('./routes/ArticlesRouter');
let RatingsRouter = require('./routes/RatingsRouter');
let PointsRouter = require('./routes/PointsRouter');
let DiscountsRouter = require('./routes/DiscountsRouter');
let MarksRouter = require('./routes/MarksRouter');
let ReceptionsRouter = require('./routes/ReceptionsRouter');

app.use(AuthRouter);
app.use(ArticlesRouter);
app.use(RatingsRouter);
app.use(PointsRouter);
app.use(DiscountsRouter);
app.use(MarksRouter);
app.use(ReceptionsRouter);
```

Листинг 3.1 – Маршруты для роутов

Здесь описаны маршруты для всех роутов, используемых в программном средстве.

3.1.2 Контроллеры и маршрутизаторы

Контроллеры и маршрутизаторы являются основой элемента *controllers* архитектурного паттерна MVC, который используется в разработке дипломного проекта. Благодаря ним происходит общение между *model* и *view* –

элементами *MVC*. Все разработанные файлы маршрутизаторов и контроллеров были описаны ранее во второй главе пояснительной записки.

Рассмотрим более наглядно взаимодействие между маршрутизатором и контроллером на примере файлов *ArticlesRouter.js* и *ArticlesController.js* соответственно. Объявление всех маршрутизатора представлено ниже в листинге 3.2.

```
let ArticlesRouter = require('./routes/ArticlesRouter');
app.use(ArticlesRouter);
```

Листинг 3.2 – Маршрут для роута *ArticlesRouter*

Программный код *ArticlesRouter.js* представлен в листинге 3.3

```
router.delete('/Articles/:id', chekAuth, ArticlesController.deleteArticles);
```

Листинг 3.3 - Маршрут файла *ArticlesRouter.js*

Представленный выше маршрут нужен для того, чтобы удалить статью с переданным идентификатором. Метод *delete* определяет http-метод запроса, который будет отправлен по пути, указанному в кавычках. В данном случае по пути */Articles/:id*. В функции *chekAuth* происходит проверка на то зарегистрирован ли пользователь, из заголовка *Authorization* берем токен *JWT*, он декодируется и сравнивается идентификатор пользователя и с тем, что хранится в токене, таким образом мы можем проверить прошол ли пользователь авторизацию. Рассматриваемый листинг маршрутизатора переопределяет запрос к функции *deleteArticles* контроллера *ArticlesController.js*, представлен на листинге 3.4.

```
deleteArticles: (req, res, next) => {
    db.models.Articles.destroy({where: {id:
req.params.id}}).then(()=>{res.send();})
    .catch((err) => {
        console.log(err);
        res.status(500).json({message: 'Не удалось удалить
статью',});})
    .then((res) => {
        console.log(res);});},
```

Листинг 3.4 – Фрагмент файла *ArticlesController.js*

Рассматриваемый листинг контроллера представляет собой экспортируемую функцию *deleteArticles*, о которой говорилось выше, в ней выполняется обработка запроса и формирование ответы

На основе такой коммуникации между роутером и контроллером построено все веб-приложение. Клиент отправляет запрос на сервер, который перенаправляется маршрутизатором на контроллер для его обработки. После обработки формируется ответ, который отправляется на *view*, клиенту.

3.1.3 Взаимодействие с базой данных

Работа с базой данных выполняется с использованием *ORM* библиотеки *Sequelize*. *Sequelize* обязывает разработчиков создавать схемы данных, то есть описание будущей таблицы для дальнейшего взаимодействия с базой данных посредством использования программных объектов. Программные объекты значительно упрощают и делают процесс работы с данными комфортным и удобным программисту. Благодаря *Sequelize* создается слой между базой данных и сервером, что делает их взаимодействие более надежным и безопасным.

Все модели проекта описаны во второй главе, посвященной проектированию.

Для лучшего понимания работы *Sequelize* рассмотрим схему *Articals*. Исходный код модели представлен в листинге 3.5.

```
const Model = Sequelize.Model;
class Articles extends Model{}
const {Users} = require('../config/Users')
Articles.init (    {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique:
true, autoIncrementIdentity: true, required: true},
    Title:{type: Sequelize.STRING, allowNull: false,
unique: true, required: true},
    Text: {type: Sequelize.STRING, allowNull: false,
unique: true, required: true},
    DatePub: {type: Sequelize.DATE, allowNull: false, re-
quired: true},
    ImageU: {type: Sequelize.STRING},
    Author: {type: Sequelize.INTEGER, allowNull: false, re-
quired: true},
    Like: {type: Sequelize.INTEGER, required: true},
    },    { sequelize, modelName:'Articles', tableName:'arti-
cles', timestamps: false});

Users.hasMany(Articles, {foreignKey:'Author'});
Articles.belongsTo(Users, {foreignKey: 'Author'});
module.exports = {Articles};
```

Листинг 3.5 – Модель Articals

Первая строка кода — это подключение библиотеки *Sequelize.Model*. Это позволяет разработчику использовать функции, описанные в модуле. К примеру, для того чтобы создать модель достаточно вызвать функцию *init*, в параметрах которой указывается название будущей модели. Модель следует

сделать экспортируемой для ее же использования в других частях проекта, например, в методах контроллеров.

Аналогично создаются и другие модели проекта. С помощью *Sequelize* взаимодействие сервера и базы данных значительно упрощается и становится легким и удобным.

3.2 Разработка клиентской части

Клиентская часть приложения – это отдельный проект, основанный на библиотеке React JS. React JS позволяет создавать сложные UI из небольших и изолированных частей кода, называемых компонентами. Компоненты позволяют разделить UI на независимые, повторно используемые части и работать с каждой из них отдельно.

3.2.1 Настройка React JS приложения, файл App.js

Создание приложения *React JS* предусматривает знание технологий *HTML*, *CSS*. Основные требования по созданию *React JS* проекта – это наличие последней версии *Node JS*, *IDE*, менеджера пакетов *NPM*, доступа к сети интернет.

Чтобы создать *React JS* проект, следует самостоятельно создать новую директорию, находясь в которой выполнить консольную команду `npm create-react-app`.

После выполнения указанной команды в созданной директории отобразятся новые файлы, папки, среди которых будет файл *App.js*.

Как и в серверной части файл *App.js* является главным. В нем подключаются модули, все разработанные компоненты, прописываются роутеры, рендерится основная страница. То есть рассматриваемый файл является связующим между всеми элементами *React JS* приложения. Программный код файла *App.js* представлен в листинге 3.6.

```
return (
  <>
    <Header/>
    <Container maxWidth="lg">
      <Routes>
        <Route path="*" element={<Home/>}/>
        <Route path="/posts/:id" element={<FullPost />}/>
        <Route path="/posts/:id/edit" element={<AddPost />}/>
        <Route path="/addpost" element={<AddPost/>}/>
      </Routes>
    </Container>
  </>
)
```

Листинг 3.6 – Фрагмент файла App.js

На этом фрагменте кода представлены роуты клиентского приложения.

3.2.2 Компоненты

Компоненты – основные элементы клиентской части. Они представляют собой обособленную группу кода, которая является самодостаточной, и направлена на решение какой-либо одной задачи. Компоненты позволяют расширить базовый набор *HTML*-элементов, путем добавления новых, инкапсулируя при это их внутреннюю структуру. То есть, мы можем взять группу базовых *HTML*-тегов, добавить к ним необходимые *CSS* стили и логику на *JavaScript*, и упаковать все это в новый *HTML*-компонент для повторного использования в любом месте приложения.

Рассмотрим фрагмент кода компонента, *Post*. Исходный код данного компонента представлен в листинге 3.7.

```
{imageUrl && (
  <img className={clsx(styles.image, { [styles.imageFull]:
isFullPost })}
    src={imageUrl}
    alt={title}
  />
)}
<div className={styles.wrapper}>
  <UserInfo {...user} additionalText={createdAt} />
  <div className={styles.indentation}>
    <h2
      className={clsx(styles.title, { [styles.titleFull]: isFullPost })}
    >
      {isFullPost ? title : <Link
to={`/posts/${id}`}>{title}</Link>}
    </h2>
    {children && <div className={styles.content}>{children}</div>}
```

Листинг 3.7 – Фрагмент кода компонента *Post*

Рассмотренный функциональный компонент нужен, чтобы вывести на главную страницу статьи, которые могут видеть как авторизованные так и не авторизованные пользователи. Для работы с *JSX* – необходимо импортировать класс *React*. *JSX* – это расширение языка *JavaScript* для работы с *React JS*. *JSX* преобразует весь написанный *HTML*-код в код *React JS*. Например строка `<h2 className={clsx(styles.title, { [styles.titleFull]: isFullPost })}>` с использованием *JSX* преобразуется в `React.createElement("h2", null, " title ")`. Это необходимо для того, чтобы *React JS* понимал, как должен выглядеть конечный UI элемент.

Все разработанные компоненты *React JS* имеют практически схожий принцип работы. С помощью *HTML*-тегов происходит описание компонента, которое в дальнейшем будет рендериться. Разработчику главное правильно

описать компонент, получить данные, внедрить их в рендер и, конечно, верно связать компоненты, если это необходимо.

3.3 Взаимодействие серверной и клиентской частей

Серверная часть веб-приложения предоставляет собой *API* для клиентской части. По этой причине возникает вопрос: как правильно связать два абсолютно разных приложения? Существует несколько способов сделать это: *Fetch*, *Proxy*, *Axios*. Но наиболее популярным решением на сегодняшний день является технология *Axios*.

Axios — это широко известная *JavaScript*-библиотека. Она представляет собой HTTP-клиент, основанный на *Promise* и предназначенный для веб-браузеров и для *Node JS*. Использование *Axios* позволяет избежать написания больших объемов шаблонного кода и сделать код чище и понятнее. При работе с *Axios* есть возможность прямого доступа к *JSON*-результату объекта *response*. Рассмотрим пример работы с *Axios*. В листинге 3.8 продемонстрирован пример отправки запроса.

```
import {createSlice, createAsyncThunk} from '@re-
duxjs/toolkit';
import axios from '../..//axios';
export const fetchDiscounts = createAsyncThunk('dis-
counts/fetchDiscounts', async() => {
  const {data} = await axios.get('/DiscountU');
  return data;});
const initialState = {
  discounts: {
    items: [],
    status: 'loading', }};
const discountsSlice = createSlice({
  name: 'discounts',
  initialState,
  reducers:{},
  extraReducers:{
    [fetchDiscounts.pending]: (state) =>{
      state.discounts.items = [];
      state.discounts.status = 'loading';    },
    [fetchDiscounts.fulfilled]: (state, action) =>{
      state.discounts.items = action.payload;
      state.discounts.status = 'loaded';    },
    [fetchDiscounts.rejected]: (state, action) =>{
      state.discounts.items = [];
      state.discounts.status = 'error'; }, },});
export const discountsReducer = discountsSlice.reducer
```

Листинг 3.8 – Пример отправки запроса

В данном примере показан запрос на получение скидки. Механизм отправки запроса на сервер, следующий: клиент, который использует React-приложение, вызвав какое-либо событие, к примеру, нажатие на кнопку, отправляет с помощью Axios запрос на сервер. Сервер, получая запрос по пути, который существует в его таблице маршрутизации, перенаправляет запрос на контроллер, который и производит его обработку, формирует ответ. Клиент через некоторое время получает ответ и получаем данные, декодируем токен и заносим в локальное хранилище необходимую информацию. По такому принципу происходит связь между серверной и клиентской частями приложения.

3.4 Использование Docker

Докер — это открытая платформа для разработки, доставки и эксплуатации приложений. Docker разработан для более быстрой публикации ваших приложений. С помощью docker вы можете отделить ваше приложение от вашей инфраструктуры и обращаться с инфраструктурой как управляемым приложением. Docker помогает выкладывать ваш код быстрее, быстрее тестировать, быстрее выкладывать приложения и уменьшить время между написанием кода и запуском кода. Docker делает это с помощью легковесной платформы контейнерной виртуализации, используя процессы и утилиты, которые помогают управлять и выкладывать ваши приложения.

В своем ядре docker позволяет запускать практически любое приложение, безопасно изолированное в контейнере. Безопасная изоляция позволяет вам запускать на одном хосте много контейнеров одновременно. Легковесная природа контейнера, который запускается без дополнительной нагрузки гипервизора, позволяет вам добиваться больше от вашего железа.

Самым удобным способом, благодаря которому можно осуществить поддержку «контейнеризации» приложения, является возможность создания докер-файлов, содержащих в себе базовую конфигурацию. Листинг dockerfile для запуска контейнера сервера и клиента приложения представлен в приложении Д.

Важно отметить, что необходимо поменять в строке подключения 'localhost' на 'host.docker.internal', это позволит нашему приложению подключиться к базе данных, которая находится на хосте.

4 Тестирование веб-приложения

Тестирование программного обеспечения – это процесс проверки соответствия заявленных к продукту требований и реально реализованной функциональности, осуществляемый путем наблюдения за его работой в искусственно созданных ситуациях и на ограниченном наборе тестов, выбранных определенным образом.

Данный курсовой проект будет тестироваться вручную – производится тестировщиком без использования программных средств, для проверки программы или сайта путём моделирования действий пользователя.

4.1 Тестирование формы входа

Приступив к мануальному тестированию проекта в первую очередь, проверим форму входа в приложение. На рисунке 4.1 показан результат при вводе неверных данных в форме авторизации.

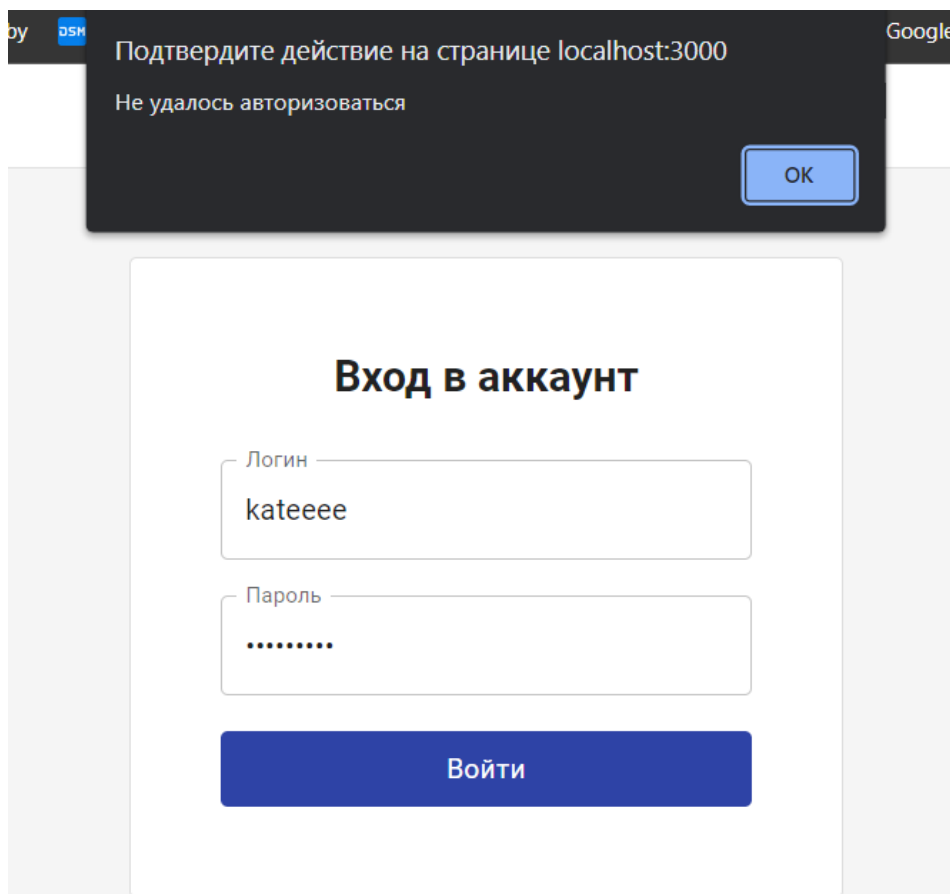
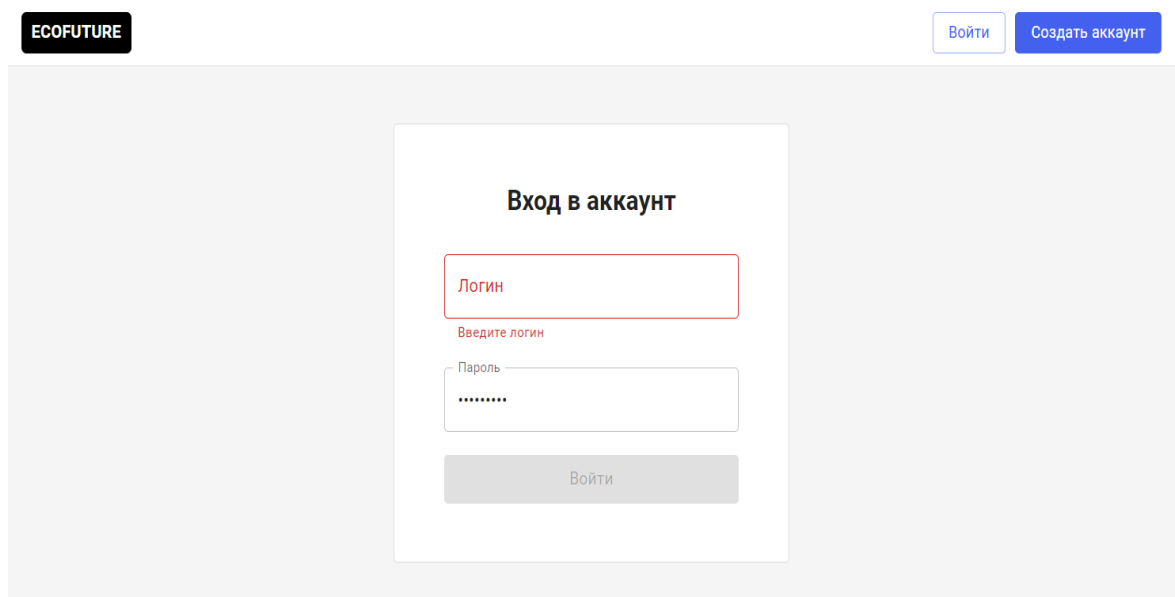


Рисунок 4.1 – Пример ввода некорректных данных при входе в систему

Как видно на изображении выше при вводе неверных данных выводится сообщение об ошибке авторизации.

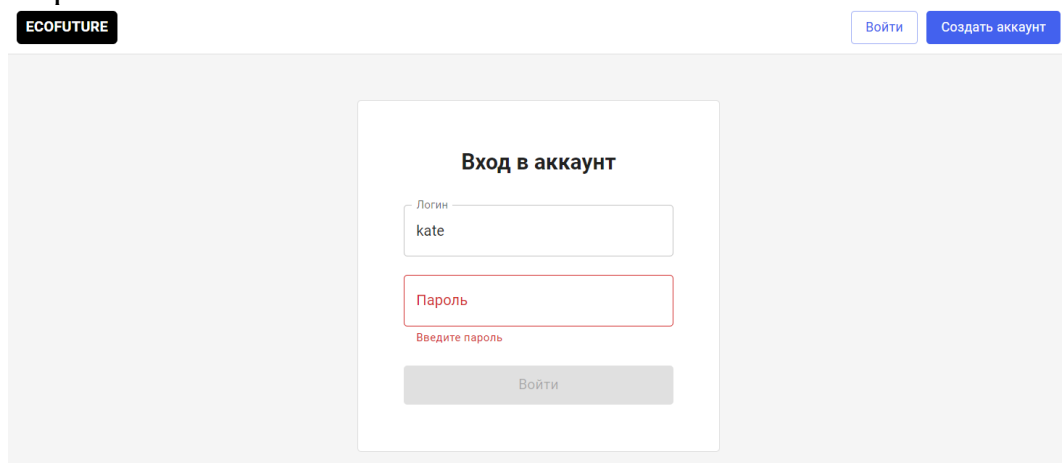
Далее проверим отсутствие данных в одном из полей формы авторизации. Рисунок 4.2 показывает результат при попытке авторизации без ввода логина.



The screenshot shows a web interface for 'ECOFUTURE'. At the top right, there are two buttons: 'Войти' (Login) and 'Создать аккаунт' (Create account). The main content area features a white box titled 'Вход в аккаунт' (Login to account). Inside this box, there are two input fields. The first field, labeled 'Логин' (Login), is empty and has a red border. Below it, the text 'Введите логин' (Enter login) is displayed in red. The second field, labeled 'Пароль' (Password), contains several dots representing masked characters. Below the password field is a grey button labeled 'Войти' (Login).

Рисунок 4.2 – Пример не заполнения поля логин при авторизации

Так же проверим случай, если в форме авторизации отсутствуют данные в поле пароль. На рисунке 4.3 показан результат попытки авторизации без ввода пароля.



The screenshot shows the same 'ECOFUTURE' login interface. In this instance, the 'Логин' (Login) field contains the text 'kate'. The 'Пароль' (Password) field is empty and has a red border. Below it, the text 'Введите пароль' (Enter password) is displayed in red. The 'Войти' (Login) button remains grey.

Рисунок 4.3 – Попытка входа с пропуском поля для ввода пароль при авторизации

После проверки формы авторизации можно перейти к тестированию формы регистрации.

4.2 Тестирование формы регистрации

При попытке ввести пустые значения, на любое поле, будет выведена ошибка, а в частности, при пустом поле логина выведется ошибка «Введите логин», при пустом поле *email* выведется ошибка «Введите почту», при пустом поле пароля выведется ошибка «Введите пароль», помимо этого если, возникшая какая-либо из эти ошибок кнопка «Зарегистрироваться» блокируется и ей нельзя воспользоваться пока не исправится ошибка. Один из таких примеров отображен на рисунке 4.4.

Создание аккаунта

Логин

Введите логин

Почта

kate@gmail.com

Пароль

.....

Зарегистрироваться

Рисунок 4.4 – Сообщение об ошибке при пустом поле логин

В случае если ввести некорректные данные выведется сообщение «Не удалось зарегистрироваться». Что можно увидеть на рисунке 4.5.

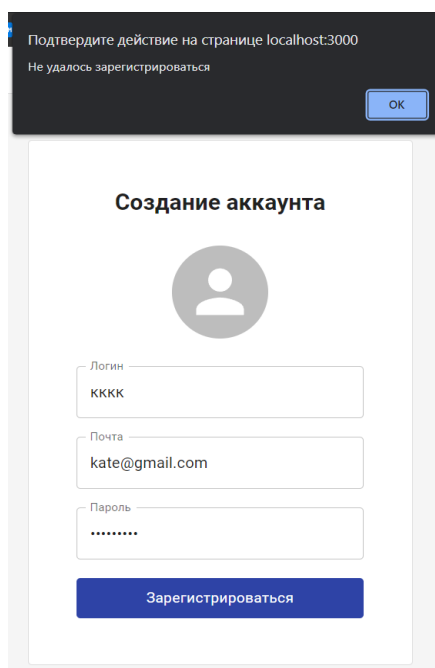


Рисунок 4.5 – Сообщение о неудачной регистрации

Если ввести неверный формат почты нам также выведется сообщение об ошибке «Адрес электронной почты должен содержать символ @. В указанном адресе отсутствует символ. Данный пример показан на рисунок 4.6

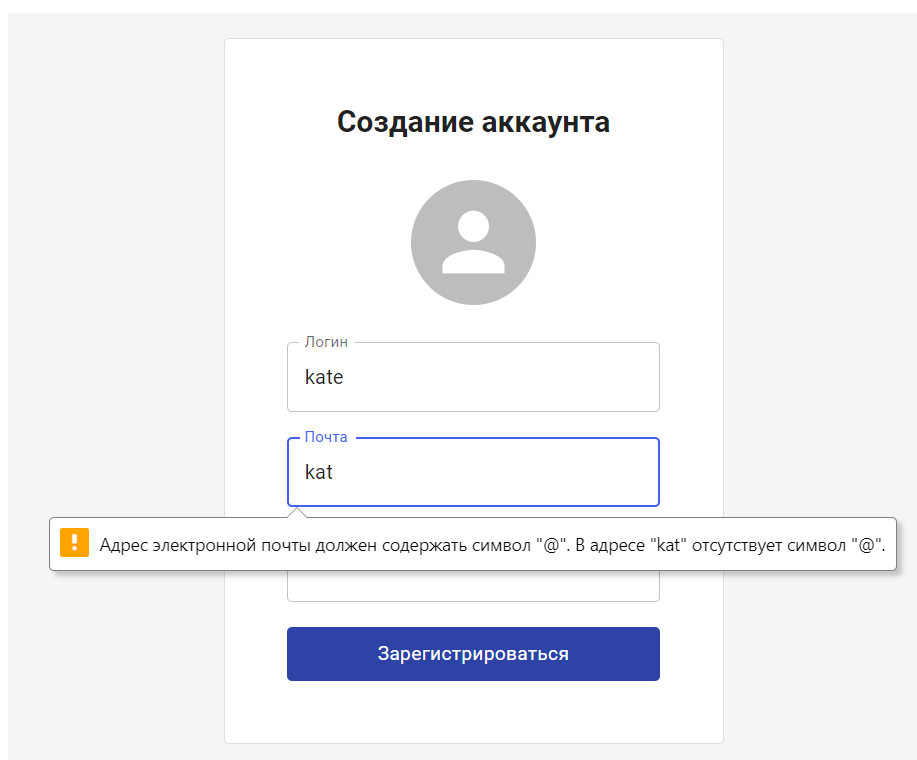


Рисунок 4.6 – Неверный формат почты

Во всех остальных местах вывести ошибочную ситуацию не удалось.

Тестирование, проводимое в данном разделе, относится к мануальному или ручному. Данный вид тестирования является оптимальным и наиболее быстрым способом проверки корректной работы как серверной части разработанного приложения, так и клиентской части. Оно полностью имитирует фактическое использование системы конечным пользователем, легко адаптируется к динамичным изменениям тестируемого приложения.

Разработанное приложение прошло функциональное тестирование без ошибок. Проверки с корректно введенными данными и нет прошли успешно. В случае введения ошибочных данных отображались информационные сообщения об ошибке ввода.

На основании этого был сделан вывод о готовности использования приложения в реальных условиях.

5 Руководство пользователя

В данном разделе будет кратко описан способ установки веб-приложения, включающий разработанные программные модули. А также будет представлено описание использования, в котором будет присутствовать весь функционал разработанного дипломного проекта.

В процессе разработки дипломного проекта использовались следующие технические и программные средства.

Технические средства:

- ноутбук *Asus VivoBook*;
- процессор *AMD Ryzen 5 5500U*.

Программные средства:

- *Windows 11*;
- *Visual Studio Code (Version: 1.74)*;
- браузеры *Google Chrome*;
- *Docker Desktop*;
- локальная база данных *MySQL*.

5.1 Методика установки

Разработанное веб-приложение состоит из серверной и клиентской частей. Соответственно, чтобы запустить готовое веб-приложение необходимо настроить и серверную часть, и клиентскую. А также следует настроить базу данных.

Заранее отметим то, что веб-приложение медицинского центра представляет собой стек технологий, главные составляющие которого являются кроссплатформенными. Таким образом, разработанное веб-приложение можно без каких-либо проблем запустить, как и на операционной системе *Windows*, так и на *Linux*, *Mac*.

Перед тем как запустить сервер, необходимо подключить базу данных, как хранилище, к которому сервер будет подключаться при запуске. В случае отсутствия такого подключения, запуск сервера будет неудачным, в результате чего программным продуктом не получится пользоваться.

Для этого необходимо зайти в *MySQL Workbench* под авторизованным пользователем, создать пустую базу данных с названием *ecofuture*.

Открыть в любом редакторе файл *DB.js* из папки *server*, проверить, чтобы в параметре хоста стояло *host.docker.internal*, далее ввести свои имя пользователя и пароль для подключения к *MySQL*.

Далее необходимо запустить *Docker Desktop*.

В проводнике заходим в папку клиента(*front*) и вызываем командную строку. По очереди запускаем следующие команды:

- `docker build -t front .;`
- `docker run -dp 3000:3000 --name front front.`

После успешного выполнения, переходим в папку сервера (*server*) и снова же вызываем командную строку. По очереди запускаем следующие команды:

- `docker build -t server .;`
- `docker run -dp 8082:8082 --name server server.`

После успешного выполнения команд в Docker Desktop в контейнерах должны появиться два компонента, вариант успешной загрузки отображен на рисунке 5.1

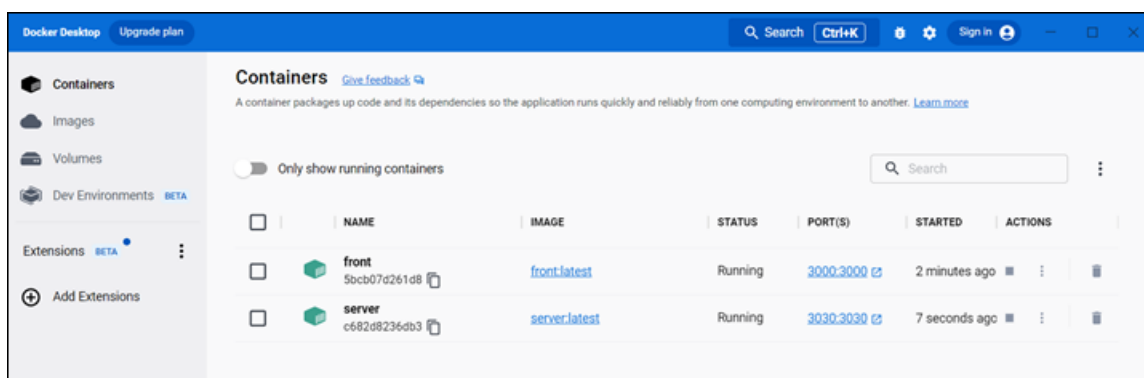


Рисунок 5.1 – Пример успешно созданных контейнеров

На этом установка приложения заканчивается. Настройка базы данных, запуск серверной и клиентской частей веб-приложения позволяют пользователям начать работу с веб-приложением.

5.2 Руководство пользователя

В руководстве пользователя мы рассмотрим отдельно функционал каждой роли, а именно: гость, пользователь, администратор.

5.2.1 Руководство пользователя для роли «гость»

При входе на сайт, гость видит главную страницу со всеми статьями опубликованными пользователями.

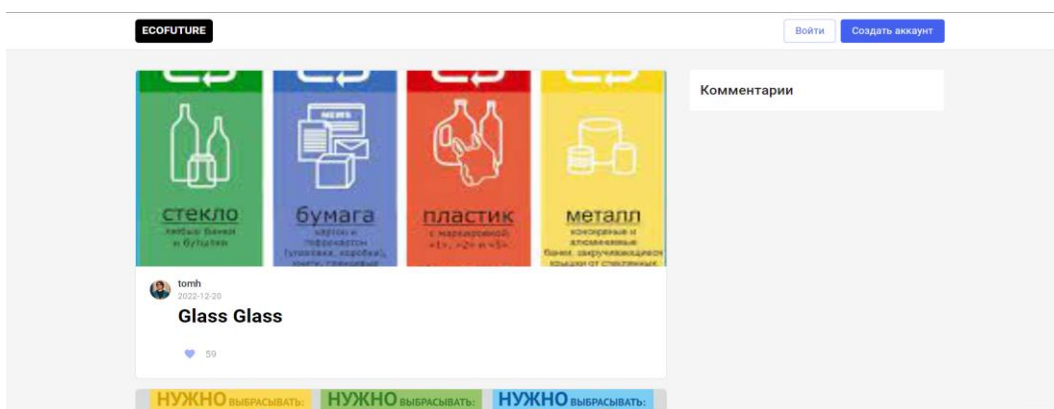


Рисунок 5.2 – Главная страница сайта

Так же доступен функционал «Войти» и «Зарегистрироваться», т.к., профиля нет, необходимо зарегистрироваться. На рисунке 5.3 изображена форма регистрации.

Создание аккаунта

Логин
kate

Почта
kate@gmail.com

Пароль
.....

Зарегистрироваться

Рисунок 5.3 – Страница регистрации с успешно заполненными данными

После успешной регистрации у нас доступно новое меню, мы стали пользователем.

5.2.2 Руководство пользователя для роли «пользователь»

Окно авторизации было рассмотрено ранее в главе 4, поэтому не будем к нему возвращаться, перейдем сразу к функционалу.

В разделе «Написать статью», пользователь может добавить свою статью, воспользовавшись формой рисунок 5.4.

ECOFUTURE

Сдать отходы Мои скидки Точки сбора Написать статью Выйти

Загрузить превью

Заголовок статьи...

В I Н “ ” ≡ % 🖼️ 📺 ✖️

Введите текст...

Рисунок 5.4 – Добавление статьи

На странице конкретной статьи можно оставить комментарий и поставить лайк это показано на рисунке 5.5.

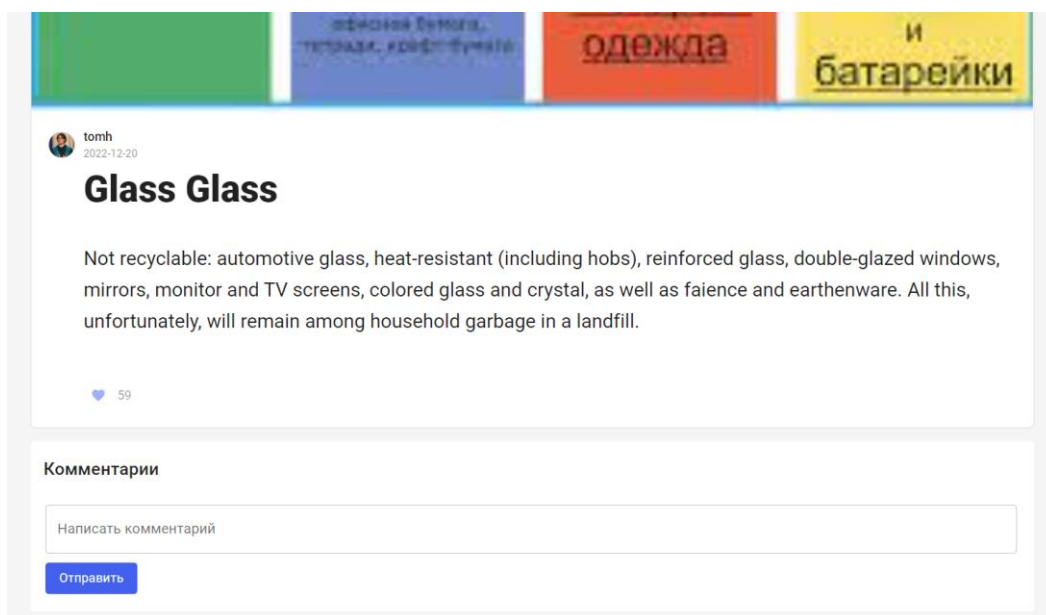


Рисунок 5.5. – Добавление комментария и возможность поставить лайк

На странице точки сбора пользователь может посмотреть адреса, где он может сдать свои отходы, это продемонстрировано на рисунке 5.6.

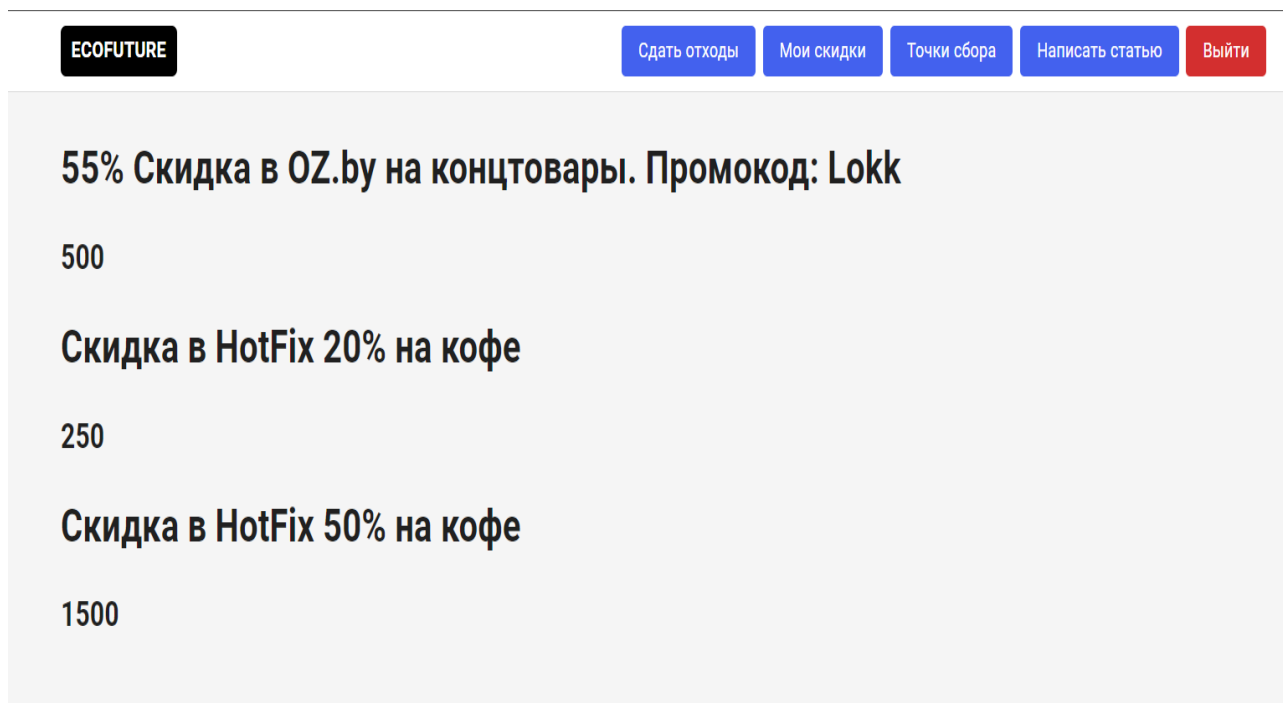


Рисунок 5.6 – Просмотр своих скидок

На странице сдать отходы можно ввести вес сколько отходов сдал пользователь, тип сдаваемых отходов и секретный ключ, который сообщают на

станции сбора отходов, для подтверждения подлинности сдачи отходов. Рисунок 5.7

Рисунок 5.7 – Сдача отходов

После успешной сдачи выводится количество начисленных баллов и количество новой продукции, которая будет сделана из сданных пользователем отходов.

Также пользователь может изменять и удалять свои статьи. Это продемонстрировано на рисунке 5.8.



Рисунок 5.8 – Возможность удалить и изменить статью

На этом функционал пользователя полностью рассмотрен.

5.2.3 Руководство пользователя для роли «администратор»

При входе администратору открываются дополнительные кнопки в меню. Помимо того, что может выполнять пользователь администратор может удалить и изменить статьи других пользователей. Это продемонстрировано на рисунке 5.9.



Рисунок 5.9 – Удаление и изменение администратором статей других пользователей

На вкладке «Все скидки» администратор может просмотреть все скидки которые предоставляет программное средство рисунок 5.10.

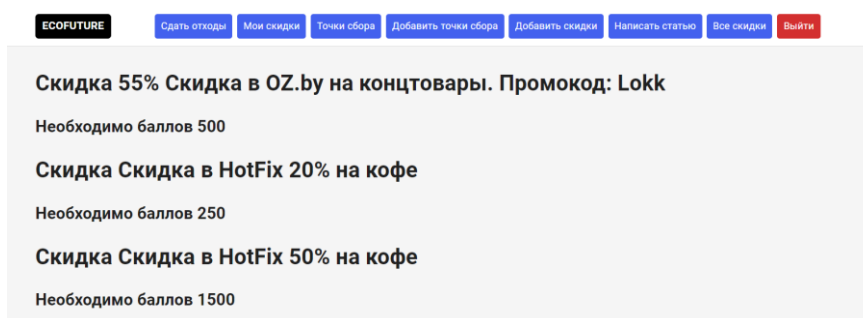
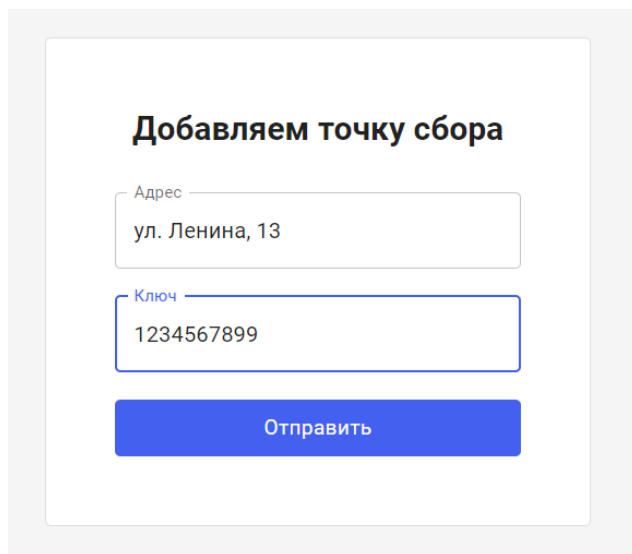


Рисунок 5.10 – Все скидки

Так же администратор может добавить новые скидки это продемонстрировано на рисунке 5.11.

Рисунок 5.11 – Добавление новой скидки

И еще администратор может добавлять новые адреса для сдачи отходов рисунок 5.12.



Добавляем точку сбора

Адрес
ул. Ленина, 13

Ключ
1234567899

Отправить

Рисунок 5.12 – Новый адрес

На этом весь функционал администратора мы рассмотрели.

Заключение

В ходе выполнения курсового проекта была проанализирована предметная область, рассмотрены основные аналоги – их достоинства и недостатки, изучены возможности рассматриваемых программных средств. Были проанализированы и выбраны основные технологии и средства для разработки курсового проекта. На основе выбранных технологий и рассмотренных аналогов был разработан основной функционал приложения.

В качестве инструмента разработки серверной части была использована платформа *Node JS*, библиотеки *Express JS*, менеджер пакетов *NPM*. Для реализации клиентского приложения использовалась библиотека *React JS*, а также технологии *HTML*, *CSS*, *Axios*.

Для хранения, использования и манипулирования данными была спроектирована и реализована база данных *MS SQL*. Для удобной работы с данными была выбрана *ORM*-библиотека *Sequelize*. В результате проектирования было разработано 7 коллекций: *Articles*, *Discounts*, *Marks*, *Points*, *Ratings*, *Receptions*, *Users*.

В результате выполнения курсового проекта была достигнута поставленная цель: создание простого и удобного в использовании сервиса, позволяющего пользователям узнать больше о раздельном сборе мусора и прибегнуть к этому.

Для функционирования системы был разработан пользовательский интерфейс и руководство пользователя, в котором описаны принципы взаимодействия с приложением.

Для проверки правильности работы всех разработанных модулей было проведено тестирование. Результаты тестирования успешны.

В результате работы, все поставленные задачи были выполнены в полной мере, цель курсового проекта достигнута.

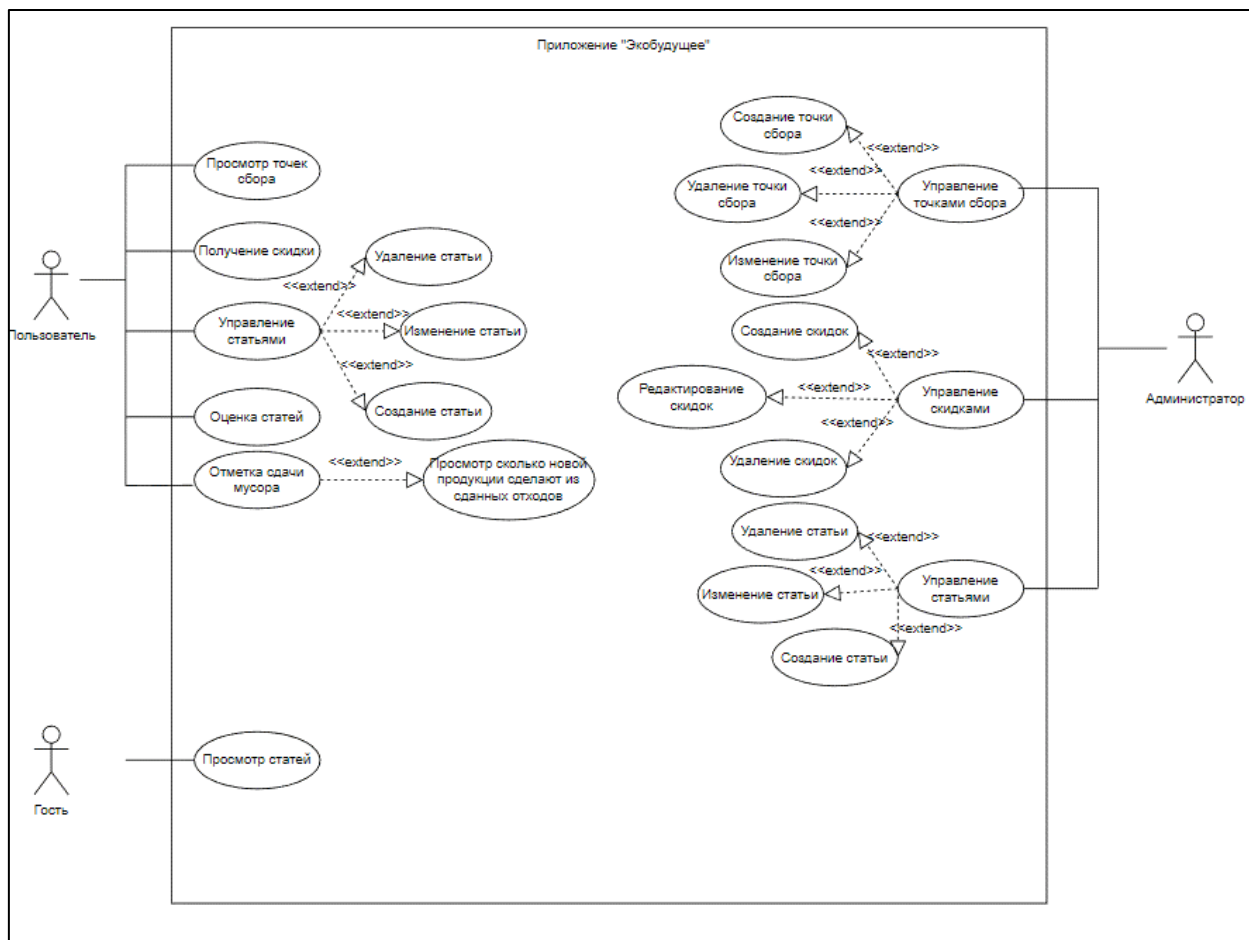
Курсовой проект является завершенным, присутствует возможность дальнейшего расширения веб-приложения и увеличение его функционала.

Список используемых источников

- 1 МОЙЭКОДВОР.РФ [Электронный ресурс] / Главная страница. – Режим доступа: <https://xn--b1adoeimccs8j.xn--p1ai/>. – Дата доступа: 10.11.2022.
- 2 RSBOR.RU [Электронный ресурс] / Главная страница. – Режим доступа: <https://rsbor.ru/>. – Дата доступа: 10.11.2022.
- 3 TAGRET99.BY [Электронный ресурс] / Главная страница. – Режим доступа: <http://www.target99.by/>. – Дата доступа: 10.11.2022.
- 4 Что такое MySQL [Электронный ресурс]. – Режим доступа: <https://mchost.ru/articles/что-такое-mysql/>. – Дата доступа: 10.11.2022.
- 5 Sequelize [Электронный ресурс]. – Режим доступа: <https://metanit.com/web/nodejs/9.1.php>. – Дата доступа: 24.11.2022.
- 6 Express JS guide [Электронный ресурс]. – Режим доступа: <https://expressjs.com/en/guide/routing.html>. – Дата доступа: 10.12.2022.
- 7 React JS create app [Электронный ресурс]. – Режим доступа: <https://ru.reactjs.org/docs/create-a-new-react-app.html>. – Дата доступа: 26.11.2022.
- 8 Основы JavaScript [Электронный ресурс]. – Режим доступа: https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/JavaScript_basics. – Дата доступа: 05.12.2022.
- 9 Node JS downloads [Электронный ресурс]. – Режим доступа: <https://nodejs.org/en/>. – Дата доступа: 28.11.2022.
- 10 NPM Package manager [Электронный ресурс]. – Режим доступа: https://www.w3schools.com/nodejs/nodejs_npm.asp. – Дата доступа: 29.11.2022.
- 11 Body-parser package [Электронный ресурс]. – Режим доступа: <https://github.com/expressjs/body-parser>. – Дата доступа: 30.10.2022.
- 12 JWT parsing [Электронный ресурс]. – Режим доступа: <https://jwt.io/>. – Дата доступа: 25.11.2022.
- 13 Справочник по HTML [Электронный ресурс]. – Режим доступа: <http://htmlbook.ru/html>. – Дата доступа: 25.11.2022.
- 14 Справочник CSS [Электронный ресурс]. – Режим доступа: <http://htmlbook.ru/css>. – Дата доступа: 25.11.2022.

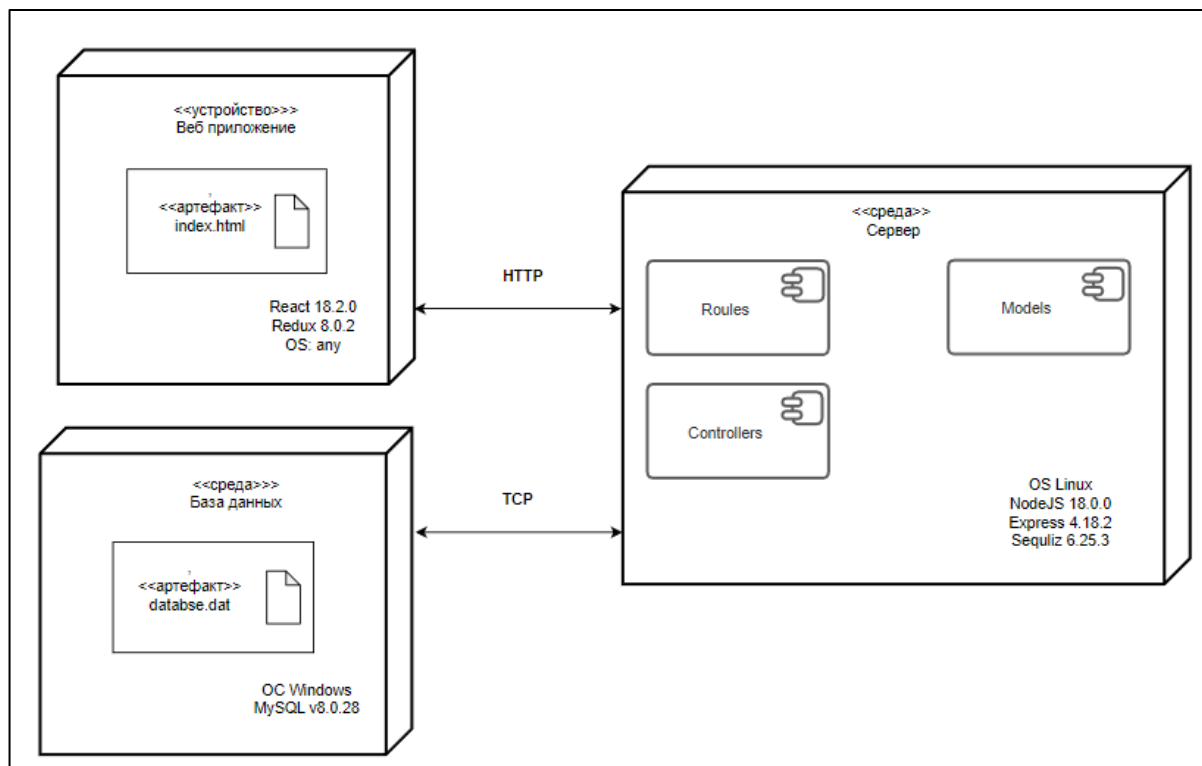
ПРИЛОЖЕНИЕ А

Диаграмма вариантов использования



ПРИЛОЖЕНИЕ Б

Диаграмма компонентов и развертывания



ПРИЛОЖЕНИЕ В

Функция для начисления баллов и отображения сколько новой продукции будет произведено из сданных отходов

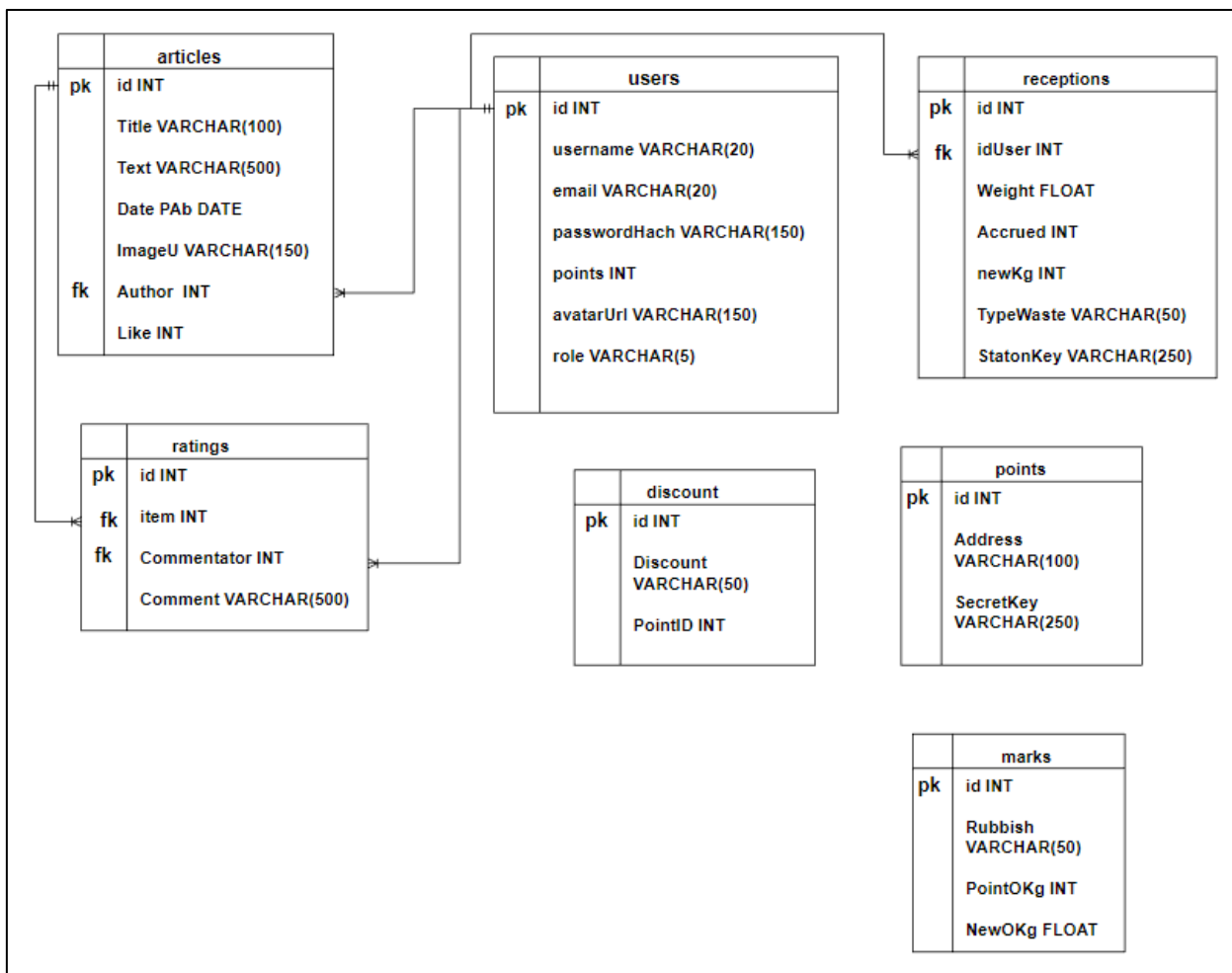
```

Receptions: async( req, res, next) => {
  try {
    const StationKey = req.body.StationKey;
    const salt = '$2b$10$qNuSSupDD53DkQfO8wqpf.';
    const StKey = await bcrypt.hash(StationKey, salt)
    const SecKey = await db.models.Points.findOne({
      attributes: ["SecretKey"],
      where: {SecretKey: StKey}
    })
    console.log("Ключ из таблицы "+SecKey.SecretKey);
    console.log("Ключ который ввел пользователь "+StKey)
    const Rub = req.body.TypeWaste
    const Mark = await db.models.Marks.findOne({
      attributes: ["PointsOKg", "NewOKg"],
      where: {Rubbish: Rub}
    })
    console.log(Mark);
    // Начисляем баллы и считаем сколько новой продукции будет
    сделано из сданных отходов
    const WeightReq = req.body.Weight;
    const AccruedN = WeightReq*Mark.PointsOKg;
    const NewKgR = WeightReq*Mark.NewOKg;
    console.log("Былы за сданое "+AccruedN);
    console.log("Новая продукция из сданого "+NewKgR);
    const Upoint = await db.models.Users.findOne({
      attributes: ["points"],
      where: {id: req.userId}})
    const NewPointU = AccruedN+Upoint.points
    console.log("Баллы для пользователя "+NewPointU);
    db.models.Users.update({
      points: NewPointU
    }, {where: {id: req.userId }})
    db.models.Receptions.create({
      idUser: req.userId,
      Accrued: AccruedN,
      NewKg: NewKgR,
      Weight: WeightReq,
      TypeWaste: Rub,
      StationKey: StKey
    })
    res.status(200).json({
      message: 'Ваши былы начислены',
    });
  } catch (error) {
    console.log(error);
    res.status(500).json({
      message: 'Не удалось начислить баллы'
    });
  }
}

```


ПРИЛОЖЕНИЕ Г

Схема базы данных



ПРИЛОЖЕНИЕ Д

Листинг *dockerfile* для запуска контейнера сервера и клиента

```
//server
FROM node:18.0.0

WORKDIR /vesuviusserver

COPY package.json package.json
COPY package-lock.json package-lock.json

RUN npm install

COPY . .

CMD ["node", "server.js"]

//client
FROM node:18.0.0

WORKDIR /vesuviusfront

COPY package.json package.json
COPY package-lock.json package-lock.json

RUN npm i --save --legacy-peer-deps

COPY . .

CMD ["npm", "start"]
```