

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Программной инженерии
Специальность 1-40 01 01 «Программное обеспечение информационных технологий»
Специализация 1-40 01 01 10 «Программное обеспечение информационных технологий
(программирование интернет-приложений)»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ НА ТЕМУ:

«Web-приложение для отдельного сбора бытовых отходов»

Выполнил студент Трубац Дмитрий Сергеевич
(Ф.И.О.)

Руководитель проекта преп.-ст. Некрасова Анастасия Павловна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доцент Смелов Владимир Владиславович
(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты преп.-ст. Некрасова Анастасия Павловна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер преп.-ст. Некрасова Анастасия Павловна
(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовая работа защищена с оценкой _____

Минск 2024

Содержание

Введение	7
1 Постановка задачи и обзор аналогичных решений	8
1.1 Постановка задач	8
1.2 Аналитический обзор аналогов	8
1.2.1 Сайт greenmap.by	8
1.2.2 Сайт rsbor.ru	9
1.2.3 Сайт target99.by	10
1.3 Выводы по разделу	11
2 Проектирование веб-приложения	12
2.1 Функциональные возможности веб-приложения	12
2.2 Логическая схема базы данных	15
2.3 Архитектура веб-приложения	21
2.3 Выводы по разделу	23
3 Реализация веб-приложения	24
3.1 Программная платформа Node.js	24
3.2 Система управления базами данных MySQL	24
3.3 Object-Relational Mapping Sequelize	24
3.4 Программные библиотеки	32
3.5 Сторонние сервисы	33
3.6 Структура серверной части	33
3.7 Реализация функций для пользователя с ролью «Гость»	36
3.7.1 Функция регистрации	36
3.7.2 Функция авторизации	37
3.7.3 Функция просмотра статей	37
3.8 Реализация функций для пользователя с ролью «Пользователь»	38
3.8.1 Функция просмотра пунктов приема	38
3.8.2 Функция добавления статьи	39
3.8.3 Функция изменения статьи	39
3.8.4 Функция удаления статьи	40
3.8.5 Функция добавления комментария	41
3.8.6 Функция удаления комментария	42
3.8.7 Функция отметки статьи	43
3.8.8 Функция отметки сдачи вторсырья	43
3.8.9 Функция обмена баллов	43
3.9 Реализация функций для пользователя с ролью «Администратор»	44
3.9.1 Функция добавления ключей	44
3.9.2 Функция изменения ключа	45
3.9.3 Функция добавления пункта приема	45
3.9.4 Функция изменения времени работы пункта приема	46
3.9.5 Функция удаления пункта приема	47
3.9.6 Функция добавления вида вторсырья	48
3.9.7 Функция изменения вида вторсырья	49
3.9.8 Функция удаления вида вторсырья	49

3.9.9	Функция добавления проверки веса	50
3.9.10	Функция изменения проверки веса.....	51
3.9.11	Функция добавления скидки	51
3.9.12	Функция изменения скидки	52
3.9.13	Функция удаления скидки	53
3.9.14	Редактирование любых статей	54
3.9.15	Удаление любых статей	54
3.9.16	Удаление комментариев пользователей.....	55
3.10	Структура клиентской части	55
3.10.1	Реализация структуры проекта.....	55
3.10.2	Реализация компонент.....	56
3.10.3	Реализация хранилища при помощи библиотеки Redux ToolKit	57
3.11	Выводы по разделу	58
4	Тестирование веб-приложения	59
4.1	Функциональное тестирование	59
4.2	Выводы по разделу	62
5	Руководство пользователя	63
5.1	Регистрация	63
5.2	Авторизация	64
5.3	Просмотр статей.....	64
5.4	Просмотр пунктов приема	65
5.5	Просмотр скидок.....	66
5.6	Обмен баллов на скидки.....	66
5.7	Добавление статьи	67
5.8	Изменение статьи.....	68
5.9	Удаление статьи	69
5.10	Добавление комментария.....	69
5.11	Удаление комментария.....	70
5.12	Добавление отметки “Нравится”	70
5.13	Отметка сдачи вторсырья.....	71
5.14	Добавление ключей	72
5.15	Изменение ключей	72
5.16	Добавление пункта приема	73
5.17	Изменение времени работы пункта	74
5.18	Удаление пункта приема.....	74
5.19	Добавление вида вторсырья.....	75
5.20	Изменение вида вторсырья	76
5.21	Удаление вида вторсырья	76
5.22	Добавление проверки веса	77
5.23	Изменение проверки веса.....	78
5.24	Добавление скидки	78
5.25	Изменение скидки.....	79
5.26	Удаление скидки	80
5.27	Редактирование любых статей	80
5.28	Удаление любых статей	81

5.29 Удаление комментариев пользователей	81
5.30 Выводы по разделу	82
Заключение	83
Список используемых источников.....	84
ПРИЛОЖЕНИЕ А.....	87
ПРИЛОЖЕНИЕ Б	91

Введение

Веб-приложение – это клиент-серверное приложение, в котором клиент взаимодействует с сервером по протоколу HTTP [1]. Раздельный сбор отходов — это сортировка мусора с учётом его происхождения и пригодности к переработке или вторичному использованию [2].

Целью данного проекта является создание веб-приложения, который предложит пользователям удобную и эффективную систему для сортировки отходов, с информированием об утилизации и местах приема.

Для достижения цели были поставлены следующие задачи:

1. Провести анализ существующих веб-приложений для раздельного сбора отходов и определить ключевые требования к разрабатываемому веб-приложению (раздел 1);
2. Разработать архитектуру веб-приложения (раздел 2);
3. Реализовать функционал веб-приложения с учетом поставленных требований (раздел 3);
4. Провести тестирование для выявления и устранения ошибок, а также для проверки соответствия требованиям (раздел 4);
5. Разработать руководство пользователя веб-приложения (раздел 5)

Целевая аудитория веб-приложения включает широкий круг пользователей, заинтересованных в улучшении экологии и соблюдении стандартов переработки отходов. Это могут быть как частные лица, так и малые компании, которым важно иметь под рукой информацию для экологически чистого управления ресурсами.

В качестве программной платформы было решено использовать Node.js [3].

1 Постановка задачи и обзор аналогичных решений

1.1 Постановка задач

Задача заключается в разработке веб-приложения для раздельного сбора бытовых отходов, которое предоставит пользователям удобный и функциональный интерфейс для взаимодействия с системой утилизации мусора.

Для пользователя приложение должно реализовывать функции поиска и отображения ближайших пунктов сбора отходов, получения информации о сортировке мусора. Администратор должен иметь возможность управлять базой данных мест сбора, добавлять и удалять пункты, следить за системой выдачей промокодов.

1.2 Аналитический обзор аналогов

Были проанализированы цели и задачи, сформированные в данном курсовом проекте, а также рассмотрены аналогичные примеры их решений. На основании анализа всех достоинств и недостатков данных альтернативных решений были сформулированы требования к данному программному средству.

1.2.1 Сайт greenmap.by

Для анализа был изучен один из популярных экологических ресурсов, белорусский сайт «Зеленая карта» — greenmap.by [4].

На рисунке 1.1 показана страница данного сайта.

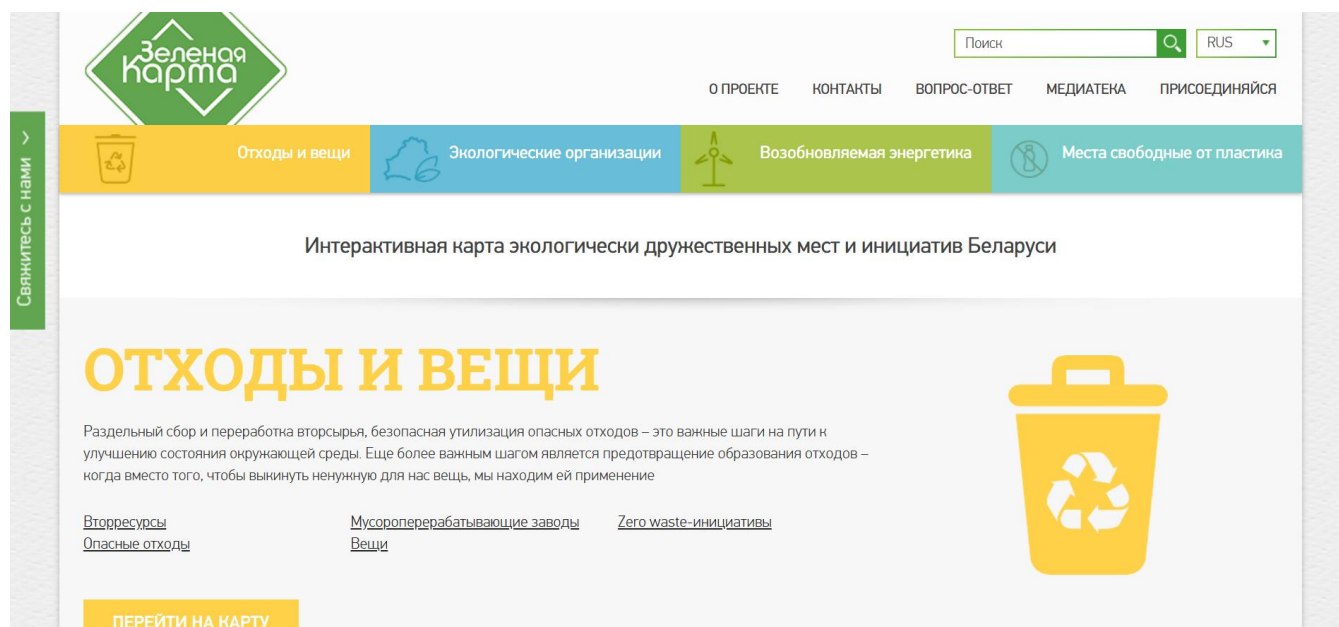


Рисунок 1.1 – Страница сайта greenmap.by

Кроме информации о раздельном сборе отходов, на сайте также можно найти информацию о различных экологических организациях, включая государственные организации, научные лаборатории и учебные заведения. Кроме того, на сайте есть

отдельный пункт меню, который посвящен возобновляемой энергетике, и на котором можно найти информацию о солнечных панелях, солнечных водонагревателях, ветрогенераторах, биогазовых установках и гидроэлектростанциях. На сайте также есть пункт меню, где представлены места, свободные от пластика, такие как школы, церковные приходы, кафе и офисы.

В данном проекте не было найдено никаких недостатков, ресурс полностью соответствует своему названию «Зеленая карта», и все возможные места, связанные с экологическими объектами, расположены на карте в понятном и доступном виде. Этот ресурс является отличным источником информации для тех, кто заботится об окружающей среде и хочет внести свой вклад в экологическую инициативу. Благодаря «Зеленой карте» люди могут быть в курсе всех экологических объектов и ресурсов в своем городе, что помогает им принимать осознанные решения и принимать активное участие в раздельном сборе отходов, защите природы и пропаганде экологического образа жизни.

1.2.2 Сайт rsbor.ru

Следующим результатом поиска аналогов стал сайт rsbor.ru [5], главная страница которого представлена на рисунке 1.2.

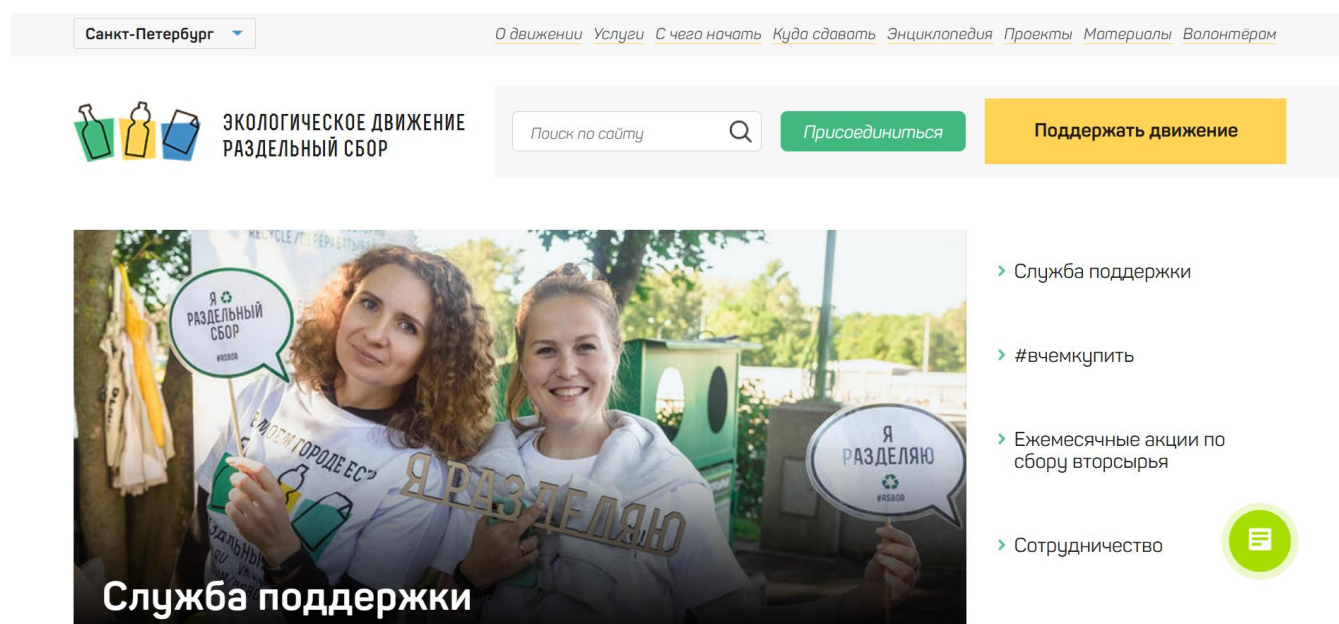


Рисунок 1.2 – Страница сайта rsbor.ru

Сайт rsbor.ru представляет собой информационный ресурс, основной упор которого сделан на новостную составляющую. На главной странице пользователи могут ознакомиться с последними новостями, связанными со сбором, разделением и переработкой отходов и вторсырья в Санкт-Петербурге или выбранном городе из выпадающего списка. Кроме того, на других страницах представлен зарубежный опыт в области переработки отходов и ресурсосбережения, позволяя пользователям расширить свои знания в этой сфере.

Одним из важных разделов сайта является раздел, посвященный проектам, проводимым Экологическим Движением «Раздельный Сбор». Здесь пользователи могут ознакомиться с текущими и предстоящими проектами, которые организует движение. Также на сайте предусмотрена возможность сделать пожертвования для поддержки и развития этого движения, позволяя желающим внести свой вклад.

Одной из удобных функций сайта является просмотр пунктов приема, где отходы могут быть сданы по видам вторсырья. Эта информация предоставляет удобный способ для пользователей найти ближайшие пункты приема и правильно сортировать свои отходы.

1.2.3 Сайт target99.by

После первых двух приложений обратилось внимание к белорусскому ресурсу государственного учреждения «Оператор вторичных материальных ресурсов» [6], 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 1.3.

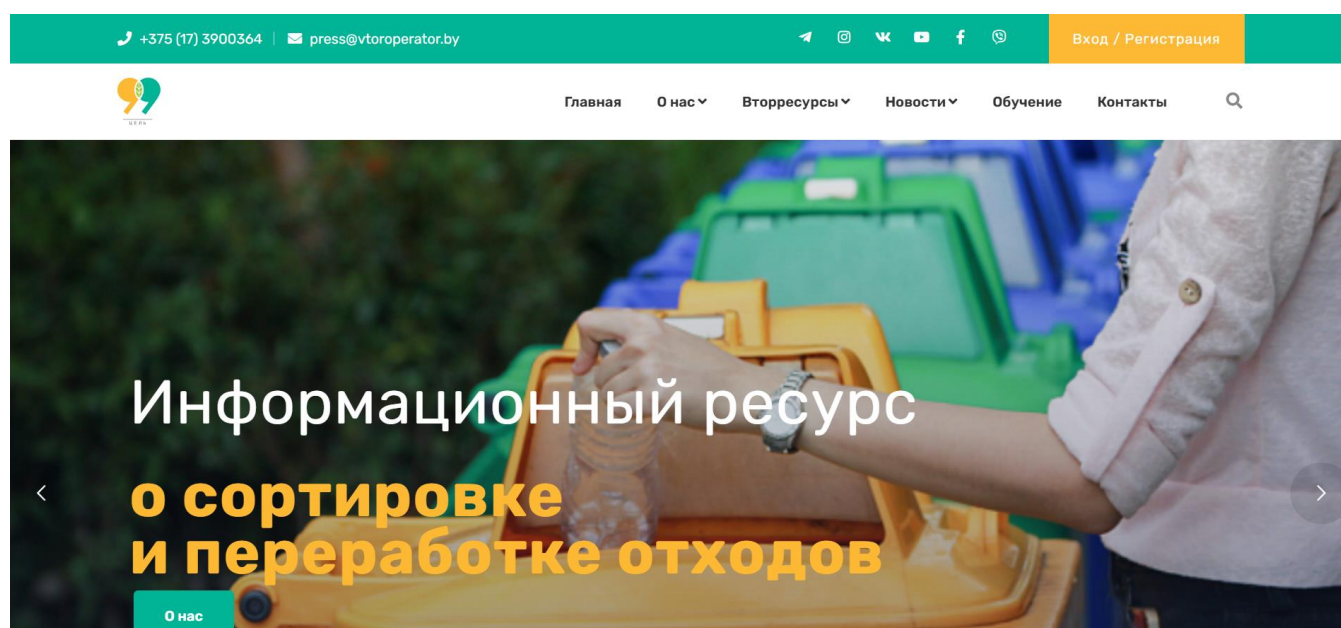


Рисунок 1.3 – Страница сайта target99.by

Основная информация, представленная на главной странице сайта, сосредоточена на правилах сортировки вторичных ресурсов. Чтобы облегчить поиск нужной информации, правила разделены по категориям, что позволяет пользователям быстро найти необходимую информацию в соответствии с их потребностями и видами отходов.

Одним из важных аспектов развития раздельного сбора мусора является подробная карта пунктов приема вторичного сырья. В Беларуси сегодня насчитывается более 1700 пунктов приема, и их число постоянно увеличивается. Эта карта предоставляет пользователю удобный инструмент для поиска ближайших пунктов приема и утилизации отходов, позволяя эффективно управлять своими ресурсами.

В целом, после рассмотрения всех указанных ресурсов на данном сайте не было выявлено никаких недостатков. Это говорит о том, что ресурс предоставляет полезную информацию и инструменты, необходимые для активного участия в раздельном сборе и переработке отходов, а также для продвижения экологичного образа жизни.

1.3 Выводы по разделу

1. Поставленные задачи требуют разработки веб-приложения для раздельного сбора бытовых отходов с поддержкой трех основных ролей: гостя, пользователя и администратора. Гость должен иметь возможность регистрации нового аккаунта и авторизации в системе. Для пользователей приложение должно предоставлять функционал поиска и отображения ближайших пунктов сбора отходов, информации о сортировке мусора и отслеживании статистики личного вклада в переработку отходов. Администратор должен иметь доступ к управлению базой данных мест сбора, добавлению и удалению пунктов, а также к контролю системы выдачи промокодов за вклад в сбор вторсырья.

2. Анализ существующих решений в сфере раздельного сбора отходов, таких как сайты «Зеленая карта» (greenmap.by), rsbor.ru и target99.by, показал, как их преимущества, так и недостатки. Эти ресурсы предоставляют базовую информацию о пунктах сбора отходов, экологических инициативах и правилах сортировки мусора, а также предлагают удобные инструменты, такие как карты пунктов приема и эко-калькуляторы. Однако ключевые недостатки включают ограниченный функционал взаимодействия с пользователями, отсутствие персонализированных функций и аналитики. Эти аспекты были учтены при формировании требований для разрабатываемого веб-приложения, чтобы предложить пользователям удобный и современный инструмент для экологически ответственного поведения.

2 Проектирование веб-приложения

2.1 Функциональные возможности веб-приложения

Функциональные возможности веб-приложения отображены в диаграмме вариантов использования (рисунк 2.1).

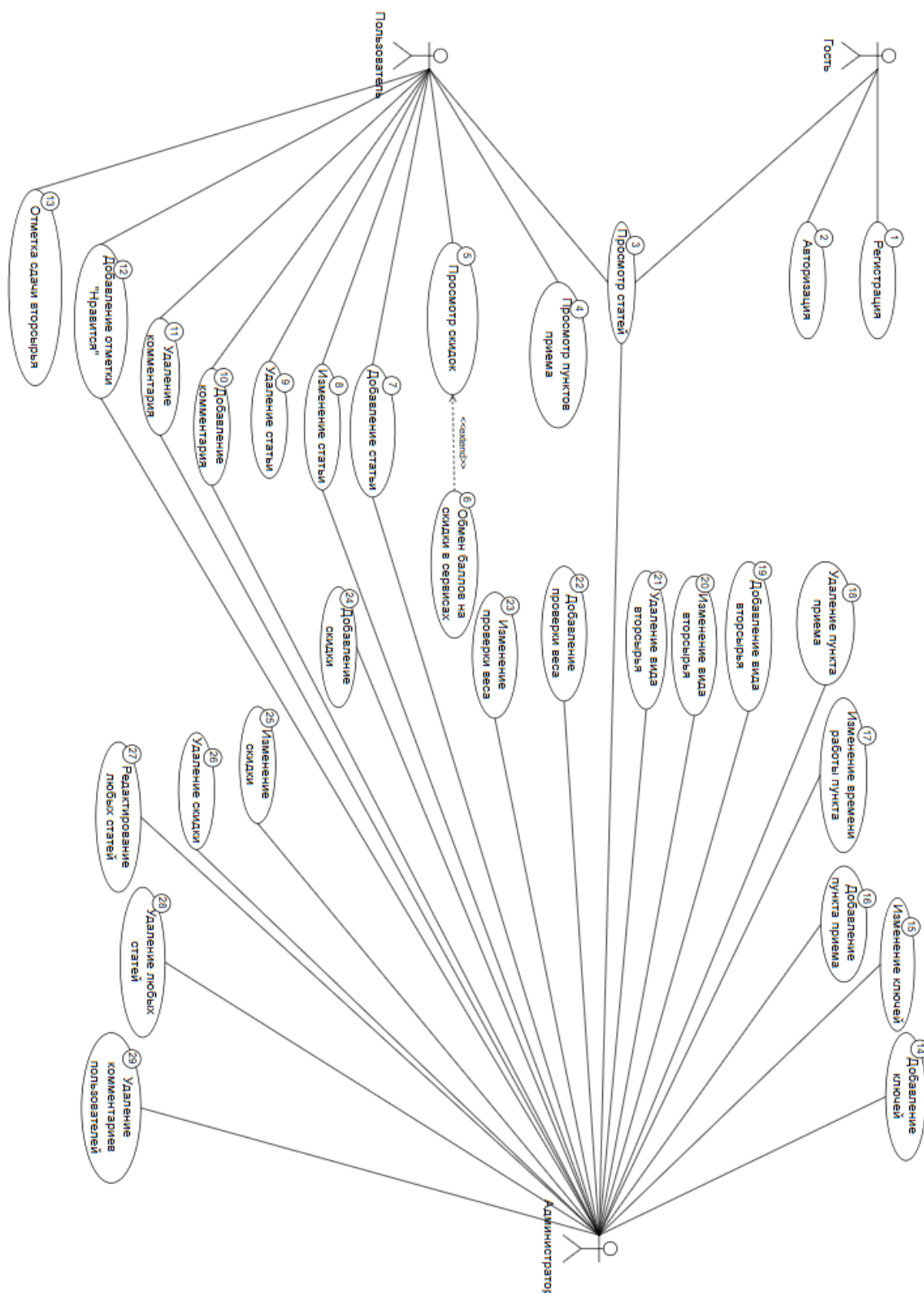


Рисунок 2.1 – Диаграмма вариантов использования

На диаграмме отражены все действующие в приложении роли и функции, которые доступны этим ролям. Описание ролей представлено в таблице 2.1.

Таблица 2.1 – Описание ролей

Роль	Описание
Гость	Может регистрироваться в системе и авторизоваться. Имеет доступ к просмотру статей. Не имеет доступа к основному функционалу.
Пользователь	Может просматривать пункты приема вторсырья, обменивать накопленные баллы на скидки в различных сервисах, создавать, изменять и удалять свои статьи, оценивать статьи, а также получать информацию о том, сколько новой продукции будет произведено из сданных отходов.
Администратор	Обладает правами для администрирования системы: управление видами вторсырья, пунктами приема, управление промокодами.

Функциональные возможности пользователя с ролью «Гость» представлены в таблице 2.2.

Таблица 2.2 – Функциональные возможности пользователя с ролью «Гость»

№	Вариант использования	Пояснение
1	Регистрация	Позволяет создать учетную запись, указав email и пароль.
2	Авторизация	Позволяет войти в систему с помощью email и пароля или по коду.
3	Просмотр статей	Обеспечивает возможность читать информационные материалы.

Далее рассмотрим функциональные возможности для пользователя, которые представлены в таблице 2.3.

Таблица 2.3 – Функциональные возможности пользователя с ролью «Пользователь»

№	Вариант использования	Пояснение
3	Просмотр статей	Обеспечивает возможность читать информационные материалы.
4	Просмотр пунктов приема	Предоставляет информацию о пунктах приема вторсырья.
5	Просмотр скидок	Отображает список доступных скидок за сдачу вторсырья.
6	Обмен баллов на скидки в сервисах	Позволяет пользователям получить скидку за сдачу вторсырья.
7	Добавление статьи	Позволяет пользователям добавлять новые статьи для публикации.

Продолжение таблицы 2.3

8	Изменение статьи	Позволяет пользователям редактировать существующие статьи.
9	Удаление статьи	Позволяет пользователям удалять статьи.
10	Добавление комментария	Позволяет комментировать статьи для обсуждения материалов.
11	Удаление комментария	Удаляет комментарии из статьи.
12	Добавление отметки “Нравится”	Позволяет отметить понравившиеся статьи.
13	Отметка сдачи вторсырья	Фиксирует факт сдачи вторсырья.

Функциональные возможности пользователя с ролью «администратор» представлены в таблице 2.4.

Таблица 2.4 – Функциональные возможности пользователя с ролью «Администратор»

№	Вариант использования	Пояснение
3	Просмотр статей	Обеспечивает возможность читать информационные материалы.
7	Добавление статьи	Позволяет администратору добавлять новые статьи для публикации.
8	Изменение статьи	Позволяет администратору редактировать существующие статьи.
10	Добавление комментария	Позволяет комментировать статьи для обсуждения материалов.
11	Удаление комментария	Удаляет комментарии из статьи.
12	Добавление отметки “Нравится”	Позволяет отметить понравившиеся статьи.
13	Добавление ключей	Позволяет администратору добавлять ключи для доступа к пункту приема.
14	Изменение ключей	Позволяет администратору изменять ключи для доступа к пункту приема.
15	Добавление пункта приема	Позволяет администратору добавлять новые пункты приема вторсырья.
17	Изменение времени работы	Позволяет администратору изменять время работы существующих пунктов приема.
18	Удаление пункта приема	Позволяет администратору удалять пункты приема вторсырья.
19	Добавление вторсырья	Вносит новые виды вторсырья в классификацию.
20	Изменение вторсырья	Корректирует описание существующих видов вторсырья.
21	Удаление вторсырья	Удаляет ненужные категории или виды из базы данных.

Продолжение таблицы 2.4

22	Добавление проверки веса	Позволяет администратору добавлять записи о проверке веса сданного вторсырья.
23	Изменение проверки веса	Редактирует настройки и параметры проверки веса.
24	Добавление скидки	Создает новые скидки, доступные за сдачу вторсырья.
25	Изменение скидки	Обновляет информацию о доступных скидках.
26	Удаление скидки	Удаляет устаревшие или неактуальные скидки.
27	Редактирование любых статей	Позволяет администратору редактировать любые статьи, включая пользовательские.
28	Удаление любых статей	Позволяет администратору удалять любые статьи, включая пользовательские.
29	Удаление комментариев пользователей	Позволяет администратору удалять комментарии, оставленные пользователями.

Такое распределение функций позволяет обеспечить четкое разграничение прав доступа, что способствует удобству использования и безопасности системы.

2.2 Логическая схема базы данных

Диаграмма базы данных таблиц (Database Table Diagram) – это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных.

В проекте была выбрана система управления базами данных MySQL [7] в качестве основной базы данных. Для реализации функциональности приложения была разработана структура базы данных, которая состоит из 12 таблиц. Диаграмма логической схемы базы данных представлена на рисунке 2.2.

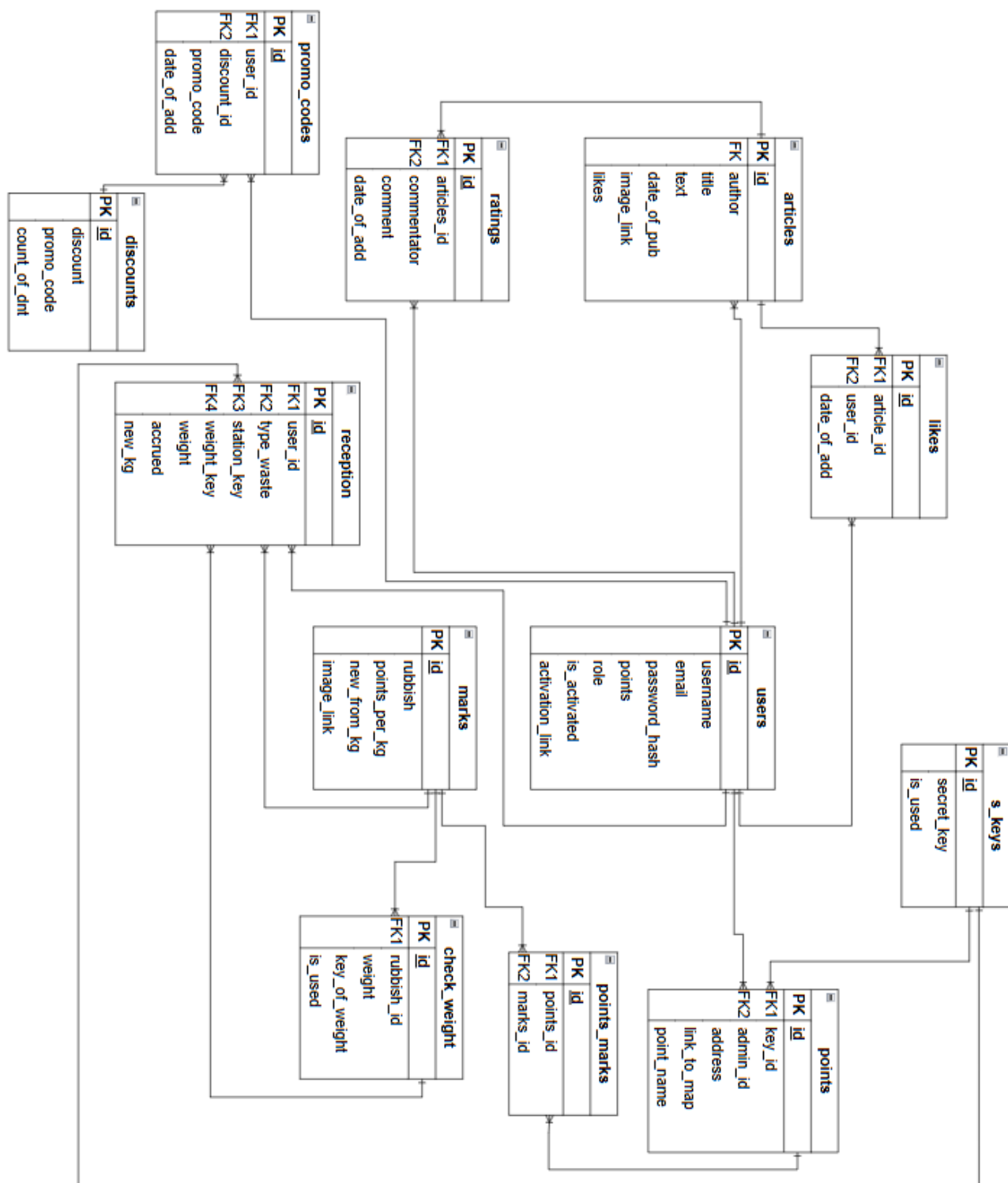


Рисунок 2.2 – Логическая схема базы данных

Для удобного взаимодействия с базой данных был выбран Sequelize [8] – ORM (Object-Relational Mapping) [9] для Node.js, который предоставляет удобный интерфейс для работы с базой данных через объекты и модели.

В таблице 2.4 содержится описание всех таблиц, которые есть в базе данных.

Таблица 2.4 – Описание таблиц базы данных

Название таблицы	Описание таблицы
users	Содержит информацию о пользователях приложения.
article	Хранит опубликованные статьи.
ratings	Хранит комментарии к опубликованным статьям.
likes	Содержит информацию о лайках, которые пользователи поставили статьям, и из этой таблицы они удаляются, при повторном вызове функции лайки.
points	Представляет пункты приема вторсырья.
marks	Содержит информацию о видах вторсырья.
points_marks	Реализует связь «многие-ко-многим» между таблицами points и marks.
check_weight	Хранит коды для проверки веса сданного вторсырья и его вида.
s_keys	Содержит секретные ключи для пунктов приема вторсырья.
discounts	Хранит информацию о скидках.
promo_codes	Содержит промокоды, которые открыты пользователями
receptions	Является статистической таблицей, где хранится информация о том, какой пользователь сдал какое вторсырье.

В таблице 2.5 показано описание полей таблицы «users».

Таблица 2.5 – Описание структуры таблицы «users»

Название	Тип данных	Описание
id	Int	Идентификатор пользователя (первичный ключ)
username	varchar(20)	Имя пользователя
email	varchar(100)	Почта
password_hash	varchar(100)	Захешированный пароль
points	int	Количество накопленных баллов
role	varchar(5)	Роль пользователя
is_activated	tinyint(1)	Активирована почта или нет
activation_link	varchar(150)	Ссылка активации, которая была отправлена на почту

В таблице 2.6 показано описание полей таблицы «articles».

Таблица 2.6 – Описание структуры таблицы «articles»

Название	Тип данных	Описание
id	int	Идентификатор статьи (первичный ключ)
title	varchar(100)	Название статьи
text	text	Текст статьи
date_of_pub	date	Дата публикации
image_url	varchar(150)	Ссылка на изображение
author	int	Автор публикации (внешний ключ)
likes	int	Количество лайков у данной статьи

В таблице 2.7 показано описание полей таблицы «ratings».

Таблица 2.7 – Описание структуры таблицы «ratings»

Название	Тип данных	Описание
id	int	Идентификатор комментария (первичный ключ)
article_id	int	Идентификатор статьи (внешний ключ)
commentator	int	Идентификатор пользователя (внешний ключ)
comment	Varchar(500)	Текст комментария
date_of_add	datetime	Дата добавления комментария

В таблице 2.8 показано описание полей таблицы «likes».

Таблица 2.8 – Описание структуры таблицы «likes»

Название	Тип данных	Описание
id	int	Идентификатор лайка (первичный ключ)
user_id	int	Идентификатор пользователя (внешний ключ)
article_id	int	Идентификатор статьи (внешний ключ)
date_of_add	datetime	Время добавления отметки

В таблице 2.9 представлена структура таблицы «s_keys».

Таблица 2.9 – Описание структуры таблицы «s_keys»

Название	Тип данных	Описание
id	int	Идентификатор секретного ключа (первичный ключ)
secret_key	varchar(100)	Секретный ключ в хешированном виде
is_used	int	Содержит значения 1 – если ключ использован и 0 – не использован

В таблице 2.10 представлена структура таблицы «points».

Таблицы 2.10 – Описание структуры таблицы «points»

Название	Тип данных	Описание
id	int	Идентификатор пункта приема (первичный ключ)
address	varchar(100)	Адрес пункта приема
time_of_work	varchar(100)	Время работы пункта приема
key_id	Int	Идентификатор на секретный ключ (внешний ключ)
admin_id	int	Идентификатор администратора, который добавил этот пункт (внешний ключ)
link_to_map	text	Ссылка на Яндекс карту
point_name	varchar(100)	Имя пункта приема

В таблице 2.11 представлена структура таблицы «marks».

Таблица 2.11 – Описание структуры таблицы «marks»

Название	Тип данных	Описание
id	int	Идентификатор вида вторсырья (первичный ключ)
rubbish	varchar(50)	Вид вторсырья
points_per_kg	int	Начисляемые баллы за один килограмм
new_from_kg	float	Сколько новой продукции будет произведено из 1 кг сданного вторсырья
image_link	varchar(255)	Ссылка на картинку

В таблице 2.12 представлено описание структура таблицы «points_marks».

Таблица 2.12 – Описание структуры таблицы «points_marks».

Название	Тип данных	Описание
id	int	Идентификатор (первичный ключ)
points_id	int	Идентификатор пункта приема (внешний ключ)
marks_id	int	Идентификатор вторсырья (внешний ключ)

В таблице 2.13 представлено описание структуры таблицы «check_weighth».

Таблица 2.13 – Описание структуры таблицы «check_weighth»

Название	Тип данных	Описание
id	int	Идентификатор (первичный ключ)
rubbish_id	int	Идентификатор на вид вторсырья (внешний ключ)
weight	int	Вес
key_of_weight	varchar(100)	Ключ для проверки в захешированном виде
is_used	int	Содержит значения 1 – если ключ использован и 0 – не использован

В таблице 2.14 представлено описание структуры таблицы «receptions».

Таблица 2.14 – Описание структуры таблицы «receptions»

Название	Тип данных	Описание
id	int	Идентификатор приема (первичный ключ)
user_id	int	Идентификатор пользователя (внешний ключ)
weight	float	Вес сколько было сдано вторсырья
accrued	Int	Начисленные баллы
new_kg	int	Вес новой продукции
type_waste	int	Вид вторсырья
station_key	int	Идентификатор секретного ключа (внешний ключ)
weight_key	int	Идентификатор ключа для проверки веса (внешний ключ)

В таблице 2.15 представлено описание структуры таблицы «discounts».

Таблица 2.15 – Описание структуры таблицы «discounts»

Название	Тип данных	Описание
id	int	Идентификатор скидки (первичный ключ)
discounts	varchar(50)	Название скидки
promo_code	varchar(50)	Промокод для применения скидки
count_for_dnt	int	Стоимость скидки в баллах

В таблице 2.16 представлено описание структуры таблицы «promo_codes».

Таблица 2.16 – Описание структуры таблицы «promo_codes»

Название	Тип данных	Описание
id	int	Идентификатор промокода (первичный ключ)
promo_code	varchar(50)	Текст промокода
user_id	int	Идентификатор пользователя, который открыл скидку (внешний ключ)
discount_id	int	Идентификатор скидки (внешний ключ)
date_of_add	date	Дата, когда был открыт промокод

В таблице 2.17 описаны связи таблиц друг с другом.

Таблица 2.17 – Описание связей сущностей

Таблица источник	Связанная таблица	Тип связи	Пояснение
users	articles	Один ко многим	Один пользователь может быть автором множества статей
users	ratings	Один ко многим	Один пользователь может оставить множество комментариев
users	likes	Один ко многим	Один пользователь может поставить множество лайков
users	points	Один ко многим	Один пользователь может быть администратором множества пунктов приема
users	receptions	Один ко многим	Один пользователь может сдать вторсырье в множество пунктов приема
users	promo_codes	Один ко многим	Один пользователь может иметь множество промокодов
articles	ratings	Один ко многим	Одна статья может иметь множество комментариев
articles	likes	Один ко многим	Одна статья может иметь множество лайков
s_keys	points	Один к одному	Один ключ используется для одного пункта приема

Продолжение таблицы 2.17

s_keys	receptions	Один ко многим	Один ключ может быть использован для множества сдач
marks	points_marks	Один ко многим	Один вид вторсырья может быть связан с множеством пунктов приема
marks	check_weight	Один ко многим	Один вид вторсырья может иметь множество записей о проверке веса
marks	receptions	Один ко многим	Один вид вторсырья может быть сдан множество раз
points	points_marks	Один ко многим	Один пункт приема может принимать множество видов вторсырья
points	receptions	Один ко многим	Один пункт приема может принять множество сдач
check_weight	receptions	Один ко многим	Одна запись о проверке веса может быть связана с множеством сдач
discounts	promo_codes	Один ко многим	Одна скидка может быть связана с множеством промокодов

Скрипт создания баз данных представлен в приложении А.

2.3 Архитектура веб-приложения

Архитектура приложения представляет собой распределённую систему, развернутую с использованием современных технологий, обеспечивающих производительность, масштабируемость и удобство управления. Архитектура веб-приложения представлена на рисунке 2.3.

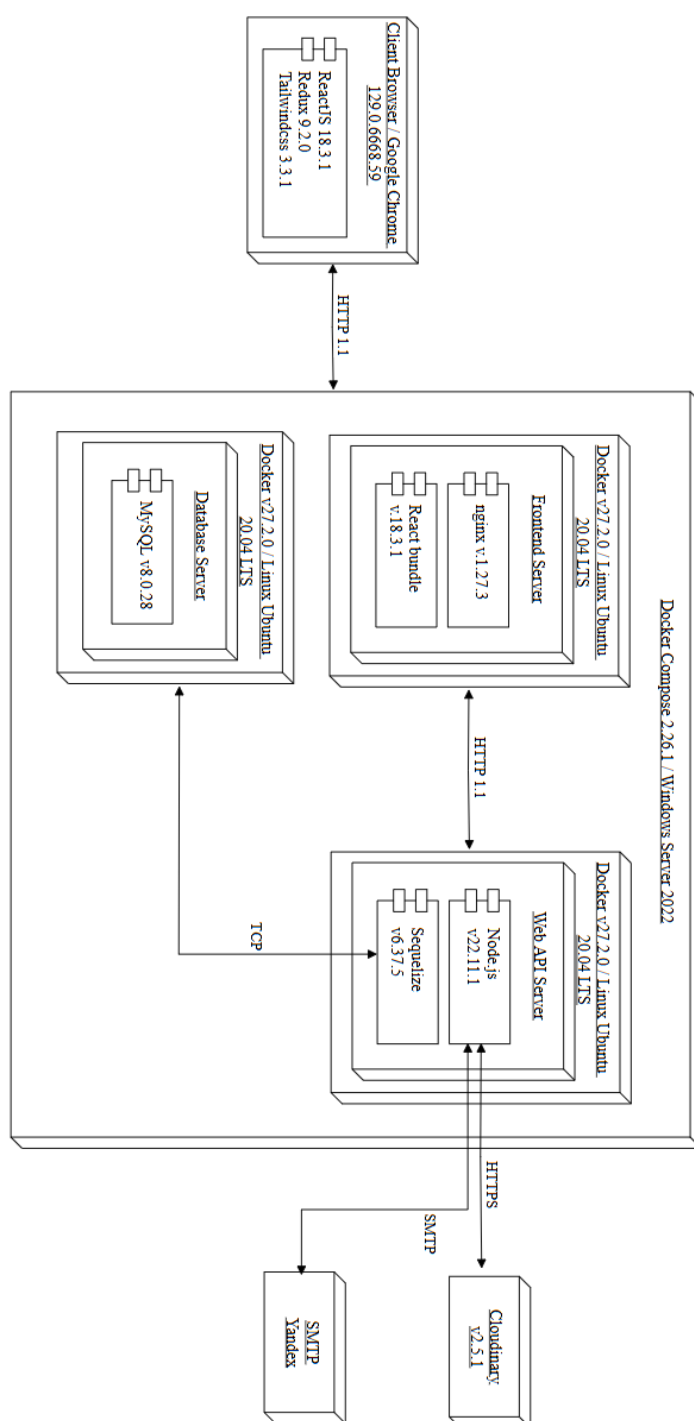


Рисунок 2.3 – Диаграмма развертывания

Пояснение назначения каждого элемента веб-приложения представлено в таблице 2.18.

Таблица 2.18 – Назначение элементов архитектурной схемы веб-приложения

Элемент	Назначение
Frontend Server (nginx [10])	Предоставляет доступ к статическим ресурсам клиентской части веб-приложения

Продолжение таблицы 2.18

Database Server (MySQL)	Используется для хранения и предоставления доступа к данным, которые необходимы для работы веб-приложения.
Backend Server (Node.js, Sequelize)	Обрабатывает запросы пользователя, запрашивает данные из базы данных с помощью ORM Sequelize.
Client Browser	Отображает клиентскую часть веб-приложения, отправляет запросы пользователя, отображает ответы сервера.
Cloudinary [11]	Облачный сервис, предназначенный для хранения изображений.
SMTP Yandex [12]	Используется для отправки электронных писем, например, для уведомлений пользователей или восстановления пароля.

Описание протоколов, используемых при работе веб-приложений, представлено в таблице 2.19.

Таблица 2.19 – Описание используемых протоколов

Протокол	Назначение
HTTP	Обмен данными между Client Browser и Frontend Server, Frontend Server и Backend Server.
HTTPS [13]	Обмен данными между Backend Server и Cloudinary. Обеспечивает безопасную передачу данных путём использования криптографического протокола TLS
SMTP [14]	Протокол для отправки электронных писем, используемый для передачи сообщений между почтовыми серверами и клиентами.
TCP [15]	Обмен данными между Database Server и Web API Server.

2.3 Выводы по разделу

1. Поддержка трех ролей с четко разграниченными правами доступа и функциональными возможностями: гость, клиент, администратор. Гость может зарегистрироваться и авторизоваться, Пользователь имеет доступ к основным функциям приложения, а Администратор управляет системой.

2. Количество функций: 29. Функционал приложения охватывает регистрацию и авторизацию, просмотр пунктов приема, вторсырья, создание, редактирование и удаление статей.

3. Количество таблиц в базе данных: 12. Таблицы охватывают основные сущности приложения, такие как пользователи, статьи, вторсырье, пункты приема, скидки и промокоды.

4. Веб-приложение имеет монолитную архитектуру с применением Nginx в качестве веб-сервера, MySQL для хранения данных, Node.js для серверной части и Docker Compose [16] для запуска многоконтейнерных Docker-приложений.

3 Реализация веб-приложения

3.1 Программная платформа Node.js

Для серверной части проекта была выбрана платформа Node.js v22.11.0, которая характеризуется событийно-ориентированной архитектурой и моделью однопоточного выполнения. В рамках проекта Node.js используется совместно с фреймворком Express.js [17], который упрощает настройку серверной части и разработку API.

3.2 Система управления базами данных MySQL

Для хранения и управления данными в проекте была выбрана система управления базами данных MySQL v8.0.28, которая является одной из самых популярных реляционных СУБД, которая обеспечивает высокую производительность, надежность и масштабируемость.

3.3 Object-Relational Mapping Sequelize

Для взаимодействия с базой данных MySQL в проекте используется ORM Sequelize v6.37.5. Sequelize — это мощная и гибкая библиотека для Node.js, которая предоставляет удобный интерфейс для работы с реляционными базами данных. Она поддерживает MySQL, PostgreSQL, SQLite и Microsoft SQL Server.

Сопоставление моделей, используемых в Sequelize, с их реальными структурами представлено в таблице 3.1.

Таблица 3.1 – Сопоставление моделей с их реальной структурой

Название модели в ORM Sequelize	Название таблицы в СУБД MySQL
Users	users
Articles	articles
Ratings	ratings
Likes	likes
Keys	s_keys
Points	points
Marks	marks
Points_marks	points_marks
Check_weight	check_weight
Receptions	receptions
Discounts	discounts
Promo_codes	promo_codes

Код, описывающий модель Users, приведён в листинге 3.1.

```
const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Users extends Model{}
Users.init (
```

```

    {
      id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
      username:{type: Sequelize.STRING, allowNull: false, unique:
true, required: true},
      email: {type: Sequelize.STRING, allowNull: false, unique:
true, required: true},
      password_hash: {type: Sequelize.STRING, allowNull: false,
required: true},
      points: {type: Sequelize.INTEGER},
      role: {type: Sequelize.STRING, validate: {isIn:[['user',
'admin']] }},
      is_activated:{type: Sequelize.BOOLEAN, default: false},
      activation_link: {type: Sequelize.STRING}
    },
    { sequelize, modelName:'Users', tableName:'users', timestamps:
false}
  );
module.exports = {Users};

```

Листинг 3.1 – Модель «Users»

Код, описывающий модель Articles, приведён в листинге 3.2.

```

const Sequelize = require('sequelize')
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.TEXT, allowNull: false,
required: true },
    date_of_pub: { type: Sequelize.DATE, allowNull: false,
required: true },
    image_url: { type: Sequelize.STRING },
    author: { type: Sequelize.INTEGER, allowNull: false,
required: true },
    likes: { type: Sequelize.INTEGER, required: true },
  },
  { sequelize, modelName: 'Articles', tableName: 'articles',
timestamps: false }
);
Users.hasMany(Articles, { foreignKey: 'author' });
Articles.belongsTo(Users, { foreignKey: 'author' });
module.exports = { Articles };

```

Листинг 3.2 – Модель «Articles»

Код, описывающий модель Ratings, приведён в листинге 3.3.

```
const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Ratings extends Model{}
const {Users} = require('../config/Users')
const {Articles} = require('../config/Articles')

Ratings.init (
  {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
    article_id: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    commentator: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    comment: {type: Sequelize.STRING, required: true},
    date_of_add: {type: Sequelize.DATE, allowNull: false,
required: true}
  },
  { sequelize, modelName: 'Ratings', tableName: 'ratings',
timestamps: false }
);
Users.hasMany(Ratings, {foreignKey: 'commentator'});
Ratings.belongsTo(Users, {foreignKey: 'commentator'});
Articles.hasMany(Ratings, {foreignKey: 'article_id', onDelete:
'cascade'});
Ratings.belongsTo(Articles, {foreignKey: 'article_id'});

module.exports = {Ratings};
```

Листинг 3.3 – Модель «Ratings»

Код, описывающий модель Likes, приведён в листинге 3.4.

```
const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Likes extends Model{}
const {Users} = require('../config/Users')
const {Articles} = require('../config/Articles')

Likes.init (
  {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
    article_id: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    user_id: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    date_of_add: {
      type: Sequelize.DATE,
      defaultValue: Sequelize.NOW
    }
  },
```



```

    },
    { sequelize, modelName: 'Likes', tableName: 'likes', timestamps:
false }
);

Users.hasMany(Likes, {foreignKey: 'user_id'});
Likes.belongsTo(Users, {foreignKey: 'user_id'});
Articles.hasMany(Likes, {foreignKey: 'article_id'});
Likes.belongsTo(Articles, {foreignKey: 'article_id'});

module.exports = {Likes};

```

Листинг 3.4 – Модель «Likes»

Код, описывающий модель Keys, приведён в листинге 3.5.

```

const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Keys extends Model{}

Keys.init({
  id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
  secret_key: {type: Sequelize.STRING, allowNull: false,
unique: true, required: true},
  is_used: {type: Sequelize.INTEGER, allowNull: false,
defaultValue:0 } ,
},
{sequelize, modelName: 'Keys', tableName: 's_keys', timestamps:
false}
);

module.exports = {Keys};

```

Листинг 3.5 – Модель «Keys»

Код, описывающий модель Points, приведён в листинге 3.6.

```

const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Points extends Model{}
const {Users} = require('../config/Users')
const {Keys} = require('./Keys')

Points.init (
  {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique:
true, autoIncrementIdentity: true, required: true},
    address: {type: Sequelize.STRING, allowNull: false,
unique: true, required: true},
    point_name: {type: Sequelize.STRING, allowNull: false,
unique: true, required: true},

```

```

        time_of_work: {type: Sequelize.STRING, allowNull: false,
required: true},
        key_id: {type: Sequelize.INTEGER, allowNull: false,
unique: true, required: true},
        admin_id: {type: Sequelize.INTEGER, allowNull: true,
required: true },
        link_to_map: {type: Sequelize.TEXT, required:true},
    },
    {sequelize, modelName: 'Points', tableName: 'points',
timestamps: false}
);

Users.hasMany(Points, {foreignKey:'admin_id'});
Points.belongsTo(Users, {foreignKey: 'admin_id'});
Keys.hasMany(Points, {foreignKey:'key_id'});
Points.belongsTo(Keys, {foreignKey: 'key_id'});

module.exports = {Points};

```

Листинг 3.6 – Модель «Points»

Код, описывающий модель Marks, приведён в листинге 3.7.

```

const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Marks extends Model{}

Marks.init (
    {
        id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
        rubbish: {type: Sequelize.STRING, allowNull: false,
required: true},
        points_per_kg: {type: Sequelize.INTEGER, allowNull: false,
required: true},
        new_from_kg: {type: Sequelize.FLOAT, allowNull: false,
required: true},
        image_link: { type: Sequelize.STRING },
    },
    { sequelize, modelName: 'Marks', tableName: 'marks', timestamps:
false }
);

module.exports = {Marks};

```

Листинг 3.7 – Модель «Marks»

Код, описывающий модель Points_marks, приведён в листинге 3.8.

```

const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Points_marks extends Model{}

```

```

const {Marks} = require('../config/Marks')
const {Points} = require('../config/Points')

Points_marks.init(
  {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
    points_id: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    marks_id: {type: Sequelize.INTEGER, allowNull: false,
required: true},
  },
  {sequelize, modelName: 'Points_marks', tableName:
'points_marks', timestamps: false}
);

Points.hasMany(Points_marks, {foreignKey:'points_id'});
Points_marks.belongsTo(Points, {foreignKey: 'points_id'});
Marks.hasMany(Points_marks, {foreignKey:'marks_id'});
Points_marks.belongsTo(Marks, {foreignKey: 'marks_id'});

module.exports = {Points_marks};

```

Листинг 3.8 – Модель «Points_marks»

Код, описывающий модель Check_weight, приведён в листинге 3.9.

```

const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Check_weights extends Model{}
const {Marks} = require('../config/Marks')

Check_weights.init(
  {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
    rubbish_id: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    weight: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    key_of_weight: {type: Sequelize.STRING, allowNull: false,
required: true}
  },
  {sequelize, modelName: 'Check_weights', tableName:
'check_weight', timestamps: false}
);
Marks.hasMany(Check_weights, {foreignKey:'rubbish_id'});
Check_weights.belongsTo(Marks, {foreignKey: 'rubbish_id'});
module.exports = {Check_weights};

```

Листинг 3.9 – Модель «Check_weight»

Код, описывающий модель **Receptions**, приведён в листинге 3.10.

```
const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Receptions extends Model{}
const {Users} = require('./Users')
const {Marks} = require('./Marks')
const {Keys} = require('./Keys')
const {Check_weights} = require('./Check_weights')

Receptions.init (
  {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique:
true, autoIncrementIdentity: true, required: true},
    user_id: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    accrued: {type: Sequelize.INTEGER, required: true},
    new_kg: {type: Sequelize.INTEGER, required: true},
    weight: {type: Sequelize.FLOAT, allowNull: false,
required: true},
    type_waste: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    station_key: {type: Sequelize.INTEGER, allowNull: false,
required: true},
    weight_key: {type: Sequelize.INTEGER, allowNull: false,
required: true},
  },
  { sequelize, modelName: 'Receptions', tableName: 'receptions',
timestamps: false }
);
Users.hasMany(Receptions, {foreignKey: 'user_id'});
Receptions.belongsTo(Users, {foreignKey: 'user_id'});
Marks.hasMany(Receptions, {foreignKey: 'type_waste'});
Receptions.belongsTo(Marks, {foreignKey: 'type_waste'});
Keys.hasMany(Receptions, {foreignKey: 'station_key'});
Receptions.belongsTo(Keys, {foreignKey: 'station_key'});
Check_weights.hasMany(Receptions, {foreignKey: 'weight_key'});
Receptions.belongsTo(Check_weights, {foreignKey: 'weight_key'});

module.exports = {Receptions};
```

Листинг 3.10 – Модель «Receptions»

Код, описывающий модель **Discounts**, приведён в листинге 3.11.

```
const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Discounts extends Model{}

Discounts.init (
  {
    id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
```

```

        discount: {type: Sequelize.STRING, allowNull: false, unique:
true, required: true},
        count_for_dnt: {type: Sequelize.INTEGER, allowNull: false,
required: true},
        promo_code: {type: Sequelize.STRING, allowNull: false,
unique: true, required: true},
    },
    { sequelize, modelName: 'Discounts', tableName: 'discounts',
timestamps: false }
);

module.exports = {Discounts};

```

Листинг 3.11 – Модель «Discounts»

Код, описывающий модель Promo_codes, приведён в листинге 3.12.

```

const Sequelize = require('sequelize')
const Model = Sequelize.Model;
class Promo_codes extends Model{}
const {Users} = require('../config/Users')
const {Discounts} = require('../config/Discounts')

Promo_codes.init(
    {
        id: {type: Sequelize.INTEGER, primaryKey:true, unique: true,
autoIncrementIdentity: true, required: true},
        promo_code: {type: Sequelize.STRING, allowNull: false,
required: true},
        user_id: {type: Sequelize.INTEGER, allowNull:
false, required: true},
        discount_id: {type: Sequelize.INTEGER, allowNull:
false, required: true},
        date_of_add: {type: Sequelize.DATE, allowNull: false,
required: true}
    },
    { sequelize, modelName: 'Promo_codes', tableName: 'promo_codes',
timestamps: false }
);

Users.hasMany(Promo_codes, { foreignKey: 'user_id' });
Promo_codes.belongsTo(Users, { foreignKey: 'user_id' });
Discounts.hasMany(Promo_codes, { foreignKey: 'discount_id' });
Promo_codes.belongsTo(Discounts, { foreignKey: 'discount_id' });

module.exports = {Promo_codes};

```

Листинг 3.12 – Модель «Promo_codes»

3.4 Программные библиотеки

В процессе разработки серверной части веб-приложения для обеспечения её функциональности и повышения эффективности работы системы были использованы программные библиотеки, представленные в таблице 3.2.

Таблица 3.2 – Программные библиотеки серверной части

Библиотека	Версия	Назначение
bcrypt [18]	5.1.1	Библиотека для хеширования паролей.
joi [19]	17.13.3	Библиотека для валидации данных.
jsonwebtoken [20]	9.0.2	Библиотека для работы с JSON Web Tokens (JWT).
mysql2 [21]	3.11.5	Драйвер для работы с MySQL в Node.js.
nodemailer [22]	6.9.16	Библиотека для отправки электронной почты.
nodemon [23]	3.1.9	Библиотека, следящая за обновлениями файлов и перезапускающая сервер.
swagger-jsdoc [24]	6.2.8	Библиотека для генерации документации API на основе JSDoc комментариев.
swagger-ui-express [25]	5.0.1	Middleware для интеграции Swagger UI с Express.
body-parser [26]	1.20.3	Middleware для парсинга входящих запросов в Express.
chalk [27]	4.1.2	Библиотека для стилизации строк в консоли.
cookie-parser [28]	1.4.7	Middleware для парсинга cookies в Express.
cors [29]	2.8.5	Middleware для включения CORS в Express.
express	4.21.2	Веб-фреймворк для Node.js.
express-validator [30]	7.2.0	Middleware для валидации и санитизации данных в Express.
fs [31]	0.0.1-security	Модуль для работы с файловой системой в Node.js.
http	0.0.1-security	Модуль для создания HTTP-серверов в Node.js.
https	1.0.0	Модуль для создания HTTPS-серверов в Node.js.
multer [32]	1.4.5-lts.1	Middleware для обработки multipart/form-data в Express
cloudinary	2.1.0	Библиотека для интеграции с Cloudinary, сервисом для хранения и обработки изображений в облаке.
sequelize	6.37.5	ORM для работы с базами данных в Node.js.
uuid [33]	11.0.3	Библиотека для генерации UUID.

В процессе разработки клиентской части веб-приложения были задействованы программные библиотеки, представленные в таблице 3.3.

Таблица 3.3 – Программные библиотеки клиентской части

Библиотека	Версия	Назначение
axios [34]	1.7.9	Библиотека для выполнения HTTP-запросов.
react-redux [35]	9.2.0	Библиотека для интеграции Redux с React-приложениями, позволяет использовать глобальное состояние.
react-toastify [36]	11.0.2	Библиотека для отображения уведомлений в React.
moment [37]	1.1.3	Библиотека для разбора и форматирования дат
@mui/material [38]	6.2.1	Библиотека компонентов Material-UI для React.
@tensorflow/tfjs [39]	4.22.0	Библиотека TensorFlow для JavaScript.
@reduxjs/toolkit [40]	2.5.0	Библиотека для управления состоянием в React с использованием Redux.
@testing-library/react [41]	16.1.0	Утилиты для тестирования React компонентов.
tailwindcss [42]	3.3.1	Фреймворк для быстрого создания пользовательских интерфейсов.
@emotion/react [43]	11.14.0	Библиотека для работы с CSS-in-JS в React.
@emotion/styled [44]	11.14.0	Библиотека для стилизации компонентов в React.
react [45]	18.3.1	Библиотека для создания пользовательских интерфейсов.
react-router-dom [46]	7.1.0	Библиотека для маршрутизации в React.
react-scripts [47]	5.0.1	Набор скриптов для создания React-приложений.

Программные библиотеки позволяют упростить реализацию веб-приложения.

3.5 Сторонние сервисы

Для успешной реализации функционала веб-приложения были использованы следующие сторонние сервисы:

1. SMTP — интеграция с этим сервисом позволила реализовать отправку электронных писем из приложения.
2. Cloudinary — используется для хранения, обработки и доставки изображений.

3.6 Структура серверной части

Основные компоненты структуры серверной части включают в себя несколько ключевых элементов, которые обеспечивают эффективную работу приложения:

1. Маршрутизаторы — управляют маршрутами и направляют запросы к соответствующим контроллерам.

2. Контроллеры — обрабатывают запросы от клиента, выполняют бизнес-логику через сервисы и возвращают ответы.

3. Middleware — промежуточные обработчики, используемые для валидации данных и обеспечения безопасности.

В таблице 3.4 приведён список директорий серверной части проекта.

Таблица 3.4 – Директории серверной части проекта

Директория	Назначение
certificates	Содержит SSL-сертификаты и ключи для обеспечения безопасного соединения.
config	Содержит конфигурационные файлы для моделей базы данных.
controllers	Содержит контроллеры, которые обрабатывают логику приложения и взаимодействие с моделями.
middleware	Содержит промежуточные обработчики (middleware) для обработки запросов, аутентификации, логирования и обработки ошибок.
routes	Содержит маршруты (роутеры), которые определяют URL-пути и соответствующие контроллеры для обработки запросов.
swagger	Содержит конфигурационные файлы для Swagger, используемого для генерации документации API.
uploads	Хранит загруженные файлы, такие как изображения и другие медиа-файлы.
utils	Содержит утилиты и вспомогательные функции, используемые в различных частях приложения.
validations	Содержит файлы валидации для проверки входных данных.

Таблица соответствия маршрутов контроллерам в исходном коде представлена в таблице 3.5.

Таблица 3.5 – Контроллеры и функции маршрутов

URL	Метод	Контроллер	Метод контроллера	№ ф-ции	Описание
/login	POST	AuthController	LoginUser	2	Авторизация пользователя
/register	POST	AuthController	RegisterUser	1	Регистрация пользователя
/articles	GET	ArticlesController	getArticles	3	Получение всех статей
/articles	POST	ArticlesController	addArticles	7	Добавление статьи
/articles/{id}	DELETE	ArticlesController	deleteArticles	9	Удаление статьи

Продолжение таблицы 3.5

/articles/\${updatedArticles.id}	PUT	ArticlesController	updateArticles	8	Изменение статьи
/like/\${id}	PUT	ArticlesController	like	12	Лайк к статье
/weight	POST	Check_weightsController	addWeight	22	Добавление проверки веса
/edit-Weight	PUT	Check_weightsController	editWeight	23	Изменение проверки веса
/discounts/\${id}	DELETE	DiscountsController	deleteDiscounts	26	Удаление скидки
/discounts	POST	DiscountsController	addDiscounts	24	Добавление скидки
/discounts/\${updatedDiscount.id}	PUT	DiscountsController	editDiscounts	25	Изменение скидки
/user/viewDiscounts	GET	DiscountsController	viewUserDiscounts	5	Получение скидок доступных пользователю
/used/discounts/\${id}	PUT	DiscountsController	usedMyDiscounts	6	Использование скидки
/marks/\${id}	DELETE	MarksController	deleteMarks	21	Удаление вида вторсырья
/marks	POST	MarksController	addMarks	19	Добавления вида вторсырья
/marks/\${updatedMark.id}	PUT	MarksController	editMarks	20	Изменение вида вторсырья
/points	GET	PointsController	getPoints	4	Получение всех пунктов приема
/points/\${id}	DELETE	PointsController	deletePoints	18	Удаление пункта приема вторсырья

Продолжение таблицы 3.5

/points	POST	PointsController	addPoints	16	Добавление нового пункта приема
/points/\${update-Point.id}	PUT	PointsController	editPoints	17	Изменение существующего пункта приема вторсырья
/keys	POST	KeysController	addKeys	14	Добавление секретного ключа для пункта приема
/points/key/\${updatedPointKeyId}	PUT	PointsController	editPointsKey	15	Изменение секретного ключа для пункта приема
/ratings/\${articles_id}	POST	RatingsController	addRatings	10	Добавление комментария
/ratings/\${id}	DELETE	RatingsController	deleteRatings	11	Удаление комментариев
/receptions	POST	ReceptionsController	Receptions	13	Сдача вторсырья

Функции с номерами 27, 28 и 29 реализуются не через контроллеры, как остальные функции, а с помощью проверки прав доступа, встроенной непосредственно в интерфейс веб-приложения (`user?.id === articles.User.id || user?.role === "admin"`). Данное условие определяет возможность взаимодействия с функционалом (например, редактирование или удаление статей) в зависимости от роли текущего пользователя. Если пользователь является автором статьи (`user?.id === articles.User.id`) или обладает правами администратора (`user?.role === "admin"`), то соответствующие элементы управления (например, кнопка редактирования или удаления) становятся видимыми и активными в интерфейсе.

При передаче данных между клиентом и сервером используется формат JSON (JavaScript Object Notation).

3.7 Реализация функций для пользователя с ролью «Гость»

3.7.1 Функция регистрации

Функция `RegisterUser` предназначена для регистрации новых пользователей в системе. Это асинхронный метод, который обрабатывает HTTP POST-запросы. Его задача — принять данные, отправленные клиентом, обработать их, создать нового

пользователя в базе данных, сгенерировать ссылку активации и отправить её на указанный пользователем email. Функция представлена в приложении Б.

Вначале пароль шифруется с использованием библиотеки `bcrypt`. Затем выполняется проверка уникальности имени и email пользователя в базе данных. Если пользователь с таким именем или email уже существует, возвращается сообщение об ошибке. Если проверка проходит успешно, создаётся новая запись пользователя с ролями и статусом неактивного аккаунта. Одновременно генерируется уникальная ссылка активации, которая отправляется на указанный email. Для отправки письма используется SMTP-сервер Yandex. Если все операции проходят успешно, сервер возвращает сообщение с уведомлением о необходимости подтвердить регистрацию. В случае возникновения ошибок отправляется соответствующее сообщение об их типе.

3.7.2 Функция авторизации

Функция `LoginUser` отвечает за процесс авторизации пользователя в системе. Она обрабатывает HTTP POST-запросы, принимая email и пароль пользователя. В первую очередь выполняется поиск пользователя в базе данных, проверяя указанный email и подтверждённый статус аккаунта. Если пользователь не найден или его аккаунт не активирован, возвращается сообщение об ошибке с указанием недействительных данных. Функция представлена в приложении Б.

Если пользователь найден, осуществляется проверка правильности введённого пароля с использованием библиотеки `bcrypt`. В случае некорректного пароля возвращается сообщение об ошибке. При успешной проверке генерируются токены `Access Token` и `Refresh Token`, которые упаковываются в cookies для последующей работы клиента с системой.

Успешная авторизация возвращает объект, содержащий сообщение о результате, сгенерированный `Access Token`, а также информацию о пользователе, включая его идентификатор, имя, роль и дополнительные данные. В случае сбоев в процессе работы сервера или некорректных данных отправляется общее сообщение об ошибке.

3.7.3 Функция просмотра статей

Функция `getArticles` реализует процесс извлечения списка статей из базы данных и их предоставления гостю. Она обрабатывает HTTP GET-запросы и возвращает данные о статьях, включая идентификатор, заголовок, текст, дату публикации, URL изображения, количество лайков, а также информацию об авторе. Функция представлена в листинге 3.13.

```
getArticles: async (req, res) => {
  try {
    const article = await db.models.Articles.findAll({
      attributes: ["id", "title", "text", "date_of_pub",
"image_url", "likes"],
      order: [['id', 'DESC']],
      include: [{
        model: db.models.Users,
        required: true,
```

```

        attributes: ["id", "username"]
    })
});

if (article.length === 0) {
    logger.warning('No articles found in the
database.');
```

```

        return res.json({ message: 'Статей нет' });
    } else {
        logger.success(`Fetched ${article.length} article(s)
successfully.`);
        res.json({ articles: article });
    }
} catch (error) {
    logger.error(`Error fetching articles:
${error.message}`);
    res.json({
        message: 'Не удалось найти статьи',
    });
}
},
```

Листинг 3.13 – Функция просмотра статей

С помощью Sequelize метод `findAll` извлекает статьи из таблицы `Articles`. При выборке указываются параметры: поля, которые должны быть включены в ответ (например, `id`, `title`, `text` и другие), сортировка по убыванию идентификаторов и вложение данных о пользователе-авторе из связанной таблицы `Users`. Если в базе данных отсутствуют статьи, функция возвращает JSON с сообщением "Статей нет". В случае успешного извлечения статей их список передается клиенту. Если в процессе выполнения происходит ошибка, гостю возвращается сообщение о невозможности найти статьи.

3.8 Реализация функций для пользователя с ролью «Пользователь»

3.8.1 Функция просмотра пунктов приема

Функция `getArticles` предназначена для извлечения списка статей из базы данных и формирования ответа в формате JSON. В начале выполняется запрос к модели `Articles` для получения данных о статьях, включая их идентификатор, заголовок, текст, дату публикации, ссылку на изображение и количество лайков. Данные сортируются в порядке убывания идентификатора, а также извлекается информация о пользователе, связанном с каждой статьей, включая его идентификатор и имя. Функция представлена в листинге 3.14.

```

getArticles: async (req, res) => {
    try {
        const article = await db.models.Articles.findAll({
            attributes: ["id", "title", "text", "date_of_pub",
"image_url", "likes"],
            order: [['id', 'DESC']],
```

```

        include: [{
            model: db.models.Users,
            required: true,
            attributes: ["id", "username"]
        }]
    });

    if (article.length === 0) {
        logger.warning('No articles found in the
database.');
```

```

        return res.json({ message: 'Статей нет' });
    } else {
        logger.success(`Fetched ${article.length} article(s)
successfully.`);
        res.json({ articles: article });
    }
} catch (error) {
    logger.error(`Error fetching articles:
${error.message}`);
    res.json({
        message: 'Не удалось найти статьи',
    });
}},
```

Листинг 3.14 – Метод для просмотра пунктов приема

Если в результате выполнения запроса статьи не найдены, записывается предупреждение в лог о пустом списке статей, и возвращается сообщение «Статей нет». В случае наличия статей информация о них передается клиенту, а успех операции фиксируется в логах с указанием количества найденных статей.

3.8.2 Функция добавления статьи

Функция `addArticles` предназначена для добавления новой статьи в базу данных. Сначала проверяется наличие статьи с таким же заголовком в базе данных. Если заголовок уже существует, в логах фиксируется предупреждение, и клиенту возвращается сообщение об этом. Функция представлена в приложении Б.

Если заголовок уникален, начинается процесс добавления статьи. Проверяется наличие URL изображения в запросе. Если изображение указано, статья создаётся с полем `image_url`, содержащим значение из запроса. Если изображение отсутствует, поле `image_url` заполняется пустой строкой.

3.8.3 Функция изменения статьи

Функция `updateArticles` используется для обновления данных статьи в системе. Она принимает HTTP PUT-запрос, в котором содержатся обновленные данные статьи, включая заголовок, текст, дату публикации и, при необходимости, ссылку на изображение. Функция представлена в приложении Б.

Сначала проверяется наличие статьи с переданным заголовком. Если URL изображения отсутствует, статья обновляется без изменения этого поля. В противном случае в обновленные данные также включается новый URL изображения.

При успешном обновлении возвращается сообщение о результате и обновленная информация о статье. В случае возникновения ошибки клиенту отправляется сообщение об ошибке.

3.8.4 Функция удаления статьи

Функция `deleteArticles` используется для удаления статьи из системы. Она обрабатывает HTTP DELETE-запрос, принимая идентификатор статьи через параметры маршрута. Реализация функции в листинг 3.15.

```
deleteArticles: async (req, res) => {
  try {
    const v_check_id_articles = await
db.models.Articles.findOne({
      where: { id: req.params.id },
    })
    if (v_check_id_articles != null) {
      const article = await db.models.Articles.destroy({
where: { id: req.params.id } })
      logger.success(`Article with ID ${req.params.id}
deleted successfully.`);
      res.json({
        message: 'Статья удалена'
      });
    }
    else {
      logger.warning(`No article found with ID
${req.params.id}.`);
      res.json({
        message: 'Не удалось удалить статью',
      });
    }
  } catch (error) {
    logger.error(`Error deleting article with ID
${req.params.id}: ${error.message}`); console.log(error);
    res.json({
      message: 'Не удалось удалить статью',
    });
  }
},
```

Листинг 3.15 – Реализация функции удаления статьи

В функции сначала выполняется проверка существования статьи с указанным идентификатором в базе данных. Если статья найдена, она удаляется из базы данных с помощью метода `destroy`, после чего сервер возвращает сообщение об успешном удалении статьи. Если статья не найдена, сервер возвращает сообщение о том, что

удаление невозможно. При возникновении ошибки возвращается сообщение об общей проблеме с удалением статьи.

3.8.5 Функция добавления комментария

Функция `addRatings` предназначена для добавления комментариев к статьям. Она обрабатывает HTTP POST-запрос, принимая идентификатор статьи и текст комментария, предоставленный пользователем, через тело запроса. Реализация функции в листинге 3.16.

```
addRatings: async (req, res) => {
  try {
    const v_check_body = req.body.comment
    if (!v_check_body) {
      res.json({
        message: 'Введите текст комментария'
      });
    }
    else {
      const v_check_a_article = await
db.models.Articles.findOne({
        where: { id: req.params.article_id }
      })
      if (v_check_a_article != null) {
        const comment = await db.models.Ratings.create({
          article_id: req.params.article_id,
          commentator: req.userId,
          comment: req.body.comment,
          date_of_add: Date.now(),
        })
        res.json({
          comment,
          message: 'Комментарий добавлен'
        });
      }
      else {
        res.json({
          message: 'Статья не существует'
        });
      }
    }
  } catch (err) {
    console.log(err);
    res.

      json({
        message: 'Не удалось добавить комментарий'
      });
  },
}
```

Листинг 3.16 – Функция добавления комментария

Функция проверяет, что текст комментария передан в запросе. Если текст отсутствует, возвращается сообщение с просьбой ввести комментарий. Затем функция проверяет, существует ли статья с указанным идентификатором в базе данных. Если статья найдена, создаётся новая запись в таблице Ratings с информацией о статье, пользователе, который добавил комментарий, и его содержанием. При успешной операции сервер возвращает сообщение о добавлении комментария. Если статья не существует или возникает ошибка, возвращается соответствующее сообщение о проблеме.

3.8.6 Функция удаления комментария

Функция `deleteRatings` предназначена для удаления комментариев из системы. Она обрабатывает HTTP DELETE-запрос, принимая идентификатор комментария через параметры маршрута. Реализация функции в листинге 3.17.

```
deleteRatings: async (req, res) => {
  try {
    const v_check_comment = await
db.models.Ratings.findOne({
      where: { id: req.params.id },
    })
    if (v_check_comment != null) {
      db.models.Ratings.destroy({ where: { id:
req.params.id } })
      res.json({
        message: 'Комментарий удален'
      });
    }
    else {
      res.json({
        message: 'Не удалось удалить комментарий',
      });
    }
  } catch (error) {
    console.log(error);
    res.json({
      message: 'Не удалось удалить комментарий',
    });
  }
},
```

Листинг 3.17 – Функция удаления комментария

Функция сначала проверяет существование комментария в базе данных с указанным идентификатором. Если комментарий найден, он удаляется из таблицы Ratings, и сервер возвращает сообщение об успешном удалении. Если комментарий отсутствует, отправляется сообщение о невозможности удаления. В случае возникновения ошибки выводится сообщение о её характере.

3.8.7 Функция отметки статьи

Функция like предназначена для добавления или удаления лайков к статье. Она принимает идентификаторы пользователя и статьи из запроса и проверяет, был ли уже поставлен лайк на статью текущим пользователем. Если лайк еще не был поставлен, он добавляется, и количество лайков обновляется. Если лайк уже существует, он удаляется, и количество лайков также обновляется. Функция представлена в приложении Б.

Функция сначала проверяет, поставил ли текущий пользователь лайк на статью. Если нет, то добавляет его в базу данных и обновляет количество лайков. Если лайк уже был поставлен, он удаляется, и количество лайков обновляется. После изменения данных отправляется обновленная информация о статье в ответ клиенту.

Если все операции успешны, возвращается сообщение с новым состоянием лайков и деталями статьи, иначе отправляется ошибка.

3.8.8 Функция отметки сдачи вторсырья

Функция Receptionс предназначена для начисления баллов пользователю за сдачу вторсырья. Она проверяет действительность ключей станции и веса, рассчитывает количество баллов на основе веса сдаваемого материала, обновляет баллы пользователя, а затем создает запись об операции в базе данных. Функция представлена в приложении Б.

В начале функции происходит проверка и верификация ключа станции. Сначала введенный ключ станции хешируется и проверяется в базе данных. Если ключ недействителен или уже использовался, функция возвращает ошибку. Далее, на основе найденного ключа, определяется пункт приёма, с которым связан этот ключ. Если пункт не найден, также возвращается ошибка.

После этого осуществляется проверка ключа веса, который также хешируется и проверяется в базе. Если ключ веса не существует или был использован ранее, возвращается ошибка. Если ключ действителен, то происходит получение информации о вторсырье, которое связано с этим ключом, и расчет начисляемых баллов.

С помощью полученных данных рассчитывается количество баллов, которое будет начислено пользователю, исходя из веса сдаваемого материала. Затем обновляются баллы пользователя в базе данных, и ключ веса помечается как использованный.

Наконец, создается запись в таблице Receptionс, где сохраняются информация о пользователе, начисленных баллах, весе и типе вторсырья. В ответе отправляется сообщение с обновленными данными о баллах пользователя и сообщением об успешной операции.

3.8.9 Функция обмена баллов

Функция usedMyDiscountс предназначена для того, чтобы позволить пользователю использовать скидку, уменьшив количество баллов, связанных с этой скидкой, и применив соответствующий промокод. После использования скидки, баллы

пользователя обновляются, а промокод генерируется для будущих операций с этим пользователем и скидкой. Функция представлена в приложении Б.

Функция начинается с получения текущего количества баллов пользователя и данных о скидке, которую он пытается использовать, используя ID, переданный в параметрах запроса. Затем рассчитывается количество новых баллов пользователя, вычитаемых за скидку, и эти баллы обновляются в базе данных.

После этого проверяется наличие ранее созданного промокода для пользователя и скидки. Если промокод для этой скидки уже существует, его обновляют в базе данных; если нет — создают новый.

Затем генерируется случайный строковый код для нового промокода с помощью функции `generateString`, которая генерирует строку заданной длины, состоящую из случайных символов. Полученный код обновляет существующую скидку в базе данных.

В завершение функция отправляет ответ с обновленными баллами пользователя и сообщением о том, что скидка была успешно использована.

3.9 Реализация функций для пользователя с ролью «Администратор»

3.9.1 Функция добавления ключей

Функция `addKeys` предназначена для добавления нового секретного ключа в базу данных. Она хэширует предоставленный ключ, проверяет его уникальность и добавляет в базу, если ключ уникален. Реализация функции в листинге 3.18.

```
addKeys: async (req, res) => {
  try {
    const v_secret_key = req.body.secret_key;
    const i_sk_salt = '$2b$10$qNuSSupDD53DkQfO8wqpf.';
    const o_secret_key = await bcrypt.hash(v_secret_key,
i_sk_salt);

    const v_check_keys = await db.models.Keys.findOne({
      where: { secret_key: o_secret_key },
    })

    if (v_check_keys == null) {
      db.models.Keys.create({
        // id: o_new_id,
        secret_key: o_secret_key,
        is_used: req.body.is_used,
      })
      res.json({
        message: 'Ключ добавлен'
      });
    }
    else {
      res.json({
        message: 'Такой ключ уже существует, введите
другой'
```

```

        });
    }

    } catch (err) {
        console.log(err);
        res.json({
            message: 'Не удалось добавить ключ'
        });
    }
}

```

Листинг 3.18 – Функция добавления ключей

В коде сначала принимается секретный ключ из тела запроса `req.body.secret_key`. Затем ключ хэшируется с использованием заданной соли `i_sk_salt`. После этого выполняется поиск в базе данных, чтобы проверить, существует ли уже такой хэшированный ключ. Если ключ уникален, он добавляется в базу данных с указанием его текущего статуса (`is_used`). В случае, если такой ключ уже существует, пользователю возвращается сообщение с просьбой ввести другой ключ. Если во время выполнения возникает ошибка, отправляется сообщение об ошибке.

3.9.2 Функция изменения ключа

Функция `editPointsKey` отвечает за обновление секретного ключа для существующего пункта сдачи отходов. Она проверяет наличие нового ключа в базе данных и обновляет его, если он уникален. Функция представлена в приложении Б.

Сначала происходит поиск текущего ключа, связанного с указанным пунктом, по его идентификатору. Затем из таблицы ключей определяется последний идентификатор, чтобы сгенерировать уникальный `id` для нового ключа. Новый секретный ключ, переданный через тело запроса, хэшируется с использованием заданной соли. После этого выполняется проверка уникальности ключа в базе данных. Если такого ключа ещё не существует, создаётся новая запись в таблице ключей. После успешного создания идентификатора для ключа таблица пунктов обновляется, чтобы отразить связь с новым ключом. Если ключ уже существует, возвращается сообщение о невозможности использования дублирующего кода. В случае любой ошибки выводится сообщение об ошибке, и выполнение функции завершается.

3.9.3 Функция добавления пункта приема

Функция `addPoints` предназначена для добавления нового пункта приема отходов. Она проверяет наличие адреса и названия пункта на уникальность, создает запись для нового пункта в базе данных, связывает его с соответствующими типами отходов, а также помечает связанный ключ как использованный. Функция представлена в приложении В.

Код функции начинается с преобразования строки отходов, переданной через `req.body.rubbish`, в массив, используя разделитель (запятую). Далее идет проверка

уникальности адреса и имени пункта. Если пункт с данным адресом или именем уже существует, возвращается соответствующее сообщение об ошибке.

Если имя и адрес уникальны, функция ищет доступный ключ, который не используется (is_used: 0). Если ключ найден, создается запись нового пункта, ключ помечается как использованный, а идентификатор нового пункта (v_point_id) сохраняется.

Далее функция связывает новый пункт с каждым типом отходов из массива. Она проверяет существование каждого вида отходов в базе, и если найденный вид отсутствует, возвращает сообщение о необходимости предварительного добавления этого отхода.

После всех успешных операций функция возвращает сообщение об успешном добавлении пункта сдачи отходов.

3.9.4 Функция изменения времени работы пункта приема

Функция editPoints предназначена для изменения времени работы существующего пункта сдачи отходов. Она проверяет наличие пункта по указанному идентификатору и обновляет запись в базе данных, если пункт существует. Реализация функции представлена в листинге 3.19.

```
editPoints: async (req, res) => {
  try {
    const v_point_id = await db.models.Points.findOne({
      where: { id: req.params.id }
    })
    if (v_point_id != null) {
      const o_points_up = await db.models.Points.update({
        time_of_work: req.body.time_of_work,
      },
      { where: { id: req.params.id } })

      res.json({
        message: 'Время работы изменено'
      });
    } else {
      res.json({
        message: 'Точки сбора отходов не существует'
      });
    }
  } catch (error) {
    console.log(error);
    res.json({
      message: 'Не удалось изменить точку сбора отходов'
    });
  }
},
```

Листинг 3.19 – Метод для изменения времени работы пункта приема

В коде функция сначала ищет в базе данных запись, соответствующую идентификатору `req.params.id`. Если такой пункт найден, обновляется поле `time_of_work` указанным значением из `req.body.time_of_work`, и пользователю возвращается сообщение об успешном изменении. Если пункта с таким идентификатором не существует, возвращается сообщение о его отсутствии.

Если в процессе выполнения кода возникает ошибка, она записывается в консоль, а пользователю отправляется сообщение о неудачной попытке изменения точки сбора.

3.9.5 Функция удаления пункта приема

Функция `deletePoints` предназначена для удаления точки сбора отходов по указанному идентификатору. Она проверяет наличие точки в базе данных и удаляет её, если такая точка существует. Реализация функции представлена в листинге 3.20.

```
deletePoints: async (req, res) => {
  try {
    const v_check_id_points = await
db.models.Points.findOne({
      where: { id: req.params.id },
    })

    if (v_check_id_points != null) {
      await db.models.Points.destroy({ where: { id:
req.params.id } })
      res.json({
        message: 'Точка сбора отходов удалена'
      });
    }
    else {
      res.json({
        message: 'Не удалось удалить точка сбора
отходов',
      });
    }
  } catch (error) {
    console.log(error);
    res.json({
      message: 'Не удалось удалить точку сбора отходов',
    });
  }
}
```

Листинг 3.20 – Метод для удаления пункта приема

В коде функция выполняет поиск точки сбора отходов в базе данных по идентификатору `req.params.id`. Если запись найдена, вызывается метод `destroy` для её удаления, после чего отправляется сообщение об успешном удалении. Если такой записи нет, пользователю возвращается сообщение о невозможности удалить несуществующую точку.

3.9.6 Функция добавления вида вторсырья

Функция `addMarks` отвечает за добавление нового вида вторсырья в базу данных. Она проверяет наличие указанного вида вторсырья в базе данных и добавляет его, если он уникален. Функция представлена в листинге 3.21.

```
addMarks: async (req, res) => {
  try {
    const v_check_marks = await db.models.Marks.findOne({
      where: { rubbish: req.body.rubbish },
    })

    if (v_check_marks == null) {
      await db.models.Marks.create({
        rubbish: req.body.rubbish,
        points_per_kg: req.body.points_per_kg,
        new_from_kg: req.body.new_from_kg,
        image_link: req.body.image_link,
      })
      res.json({
        message: 'Вторсырье добавлено'
      });
    }
    else {
      res.json({
        message: 'Такое вторсырье уже есть, введите
новое',
      });
    }
  } catch (err) {
    console.log(err);
    res.json({
      message: 'Не удалось добавить вторсырье'
    });
  }
},
```

Листинг 3.21 – Функция добавления вида вторсырья

Сначала происходит поиск указанного вида вторсырья в базе данных по его названию. Если такой вид вторсырья не найден, создается новая запись в таблице `Marks` с переданными через тело запроса данными: название вторсырья (`rubbish`), количество баллов за килограмм (`points_per_kg`), количество баллов за новый килограмм (`new_from_kg`) и ссылка на изображение (`image_link`). После успешного создания записи возвращается сообщение о добавлении вторсырья. Если такой вид вторсырья уже существует, возвращается сообщение о необходимости ввести новое название. В случае любой ошибки выводится сообщение об ошибке, и выполнение функции завершается.

3.9.7 Функция изменения вида вторсырья

Функция `editMarks` отвечает за обновление существующего вида вторсырья в базе данных. Она обновляет информацию о вторсырье на основе переданных данных. Функция представлена в листинге 3.22.

```
editMarks: async (req, res) => {
  try {
    await db.models.Marks.update({
      rubbish: req.body.rubbish,
      points_per_kg: req.body.points_per_kg,
      new_from_kg: req.body.new_from_kg,
      image_link: req.body.image_link,
    }, { where: { id: req.params.id } })

    res.json({
      message: 'Вторсырье обновлено'
    });
  } catch (error) {
    console.log(error);
    res.json({
      message: 'Не удалось обновить вторсырье'
    });
  }
},
```

Листинг 3.22 – Функция изменения вида вторсырья

Функция начинает с попытки обновления записи в таблице `marks`. Обновление включает в себя изменение названия вторсырья, количества баллов за килограмм, количества баллов за новый килограмм и ссылки на изображение. Обновление происходит для записи, идентификатор которой передан в параметрах запроса.

Если обновление проходит успешно, функция возвращает сообщение об успешном обновлении вторсырья. В случае возникновения ошибки, функция выводит сообщение об ошибке и завершает выполнение.

3.9.8 Функция удаления вида вторсырья

Функция `deleteMarks` предназначена для удаления существующего вида вторсырья из базы данных. Она проверяет наличие записи с указанным идентификатором и удаляет её, если она существует. Реализация функции представлена в листинге 3.23.

```
deleteMarks: async (req, res) => {
  try {
    const v_check_id_marks = await db.models.Marks.findOne({
      where: { id: req.params.id },
    })

    if (v_check_id_marks != null) {
```

```

        await db.models.Marks.destroy({ where: { id:
req.params.id } })
        res.json({
            message: 'Вротсырье удалено'
        });
    }
    else {
        res.json({
            message: 'Не удалось удалить вротсырье',
        });
    }

} catch (error) {
    console.log(error);
    res.json({
        message: 'Не удалось удалить вротсырье',
    });
}
}

```

Листинг 3.23 – Функция удаления вида вторсырья

Функция начинает с поиска записи в таблице marks по указанному идентификатору. Если запись найдена, она удаляется из базы данных. После успешного удаления записи возвращается сообщение об удалении вторсырья. Если запись не найдена, возвращается сообщение о невозможности удаления. В случае возникновения ошибки выводится сообщение об ошибке, и выполнение функции завершается.

3.9.9 Функция добавления проверки веса

Функция addWeight предназначена для добавления новой записи веса вторсырья в базу данных. Она начинает свою работу с извлечения данных из запроса, включая ключ для веса, тип вторсырья и вес. Затем функция проверяет наличие указанного типа вторсырья в таблице Marks. Если тип вторсырья не найден, возвращается сообщение об ошибке с предложением сначала добавить этот тип вторсырья. Функция представлена в приложении Б.

Если тип вторсырья найден, функция проверяет наличие записи с указанным ключом для веса в таблице Check_weights. Если такая запись уже существует, возвращается сообщение об ошибке с предложением ввести новый ключ.

Если запись с указанным ключом не найдена, функция проверяет наличие записи с указанным типом вторсырья и весом в таблице Check_weights. Если такая запись уже существует, возвращается сообщение об ошибке, указывающее, что ключ уже добавлен для этого типа вторсырья и веса.

Если все проверки пройдены успешно, функция создает новую запись в таблице Check_weights с указанными данными и возвращает сообщение об успешном добавлении ключа.

3.9.10 Функция изменения проверки веса

Функция `editWeight` предназначена для обновления существующей записи веса вторсырья в базе данных. Функция представлена в приложении Б.

Функция начинает свою работу с извлечения данных из запроса, включая идентификатор записи, тип вторсырья, вес и ключ веса. Затем ключ веса хешируется с использованием библиотеки `bcrypt` для обеспечения безопасности.

Далее функция проверяет наличие указанного типа вторсырья в таблице `Marks`. Если такой тип вторсырья не найден, возвращается сообщение об ошибке с предложением сначала добавить этот тип вторсырья.

Если тип вторсырья существует, функция проверяет наличие записи с указанным идентификатором в таблице `Check_weights`. Если запись найдена, она обновляется с новыми данными: идентификатором вторсырья, весом и хешированным ключом веса. После успешного обновления записи возвращается сообщение об успешном обновлении.

Если запись с указанным идентификатором не найдена, возвращается сообщение о том, что запись не найдена. В случае возникновения ошибки во время выполнения функции, выводится сообщение об ошибке, и выполнение функции завершается.

3.9.11 Функция добавления скидки

Функция `addDiscounts` предназначена для добавления новой скидки в базу данных. Реализация функции представлена в листинге 3.24.

```
addDiscounts: async (req, res) => {
  try {
    const v_check_discount = await
db.models.Discounts.findOne({
      where: { discount: req.body.discount }
    })
    if (v_check_discount == null) {
      db.models.Discounts.create({
        discount: req.body.discount,
        count_for_dnt: req.body.count_for_dnt,
        promo_code: req.body.promo_code
      })
      res.json({
        message: 'Скидка добавлена'
      });
    }
    else {
      res.json({
        message: 'Такая скидка уже существует'
      });
    }
  } catch (err) {
    console.log(err);
    res.json({
```

```

        message: 'Не удалось добавить Скидку'
    });
}
},

```

Листинг 3.24 – Функция добавления скидки

Функция начинает с проверки наличия скидки с указанным значением в таблице Discounts. Если такая скидка не найдена, функция создает новую запись в таблице Discounts с переданными данными: значением скидки, количеством для скидки и промокодом. После успешного добавления скидки возвращается сообщение об успешном добавлении.

Если скидка с указанным значением уже существует, возвращается сообщение о том, что такая скидка уже существует. В случае возникновения ошибки во время выполнения функции, выводится сообщение об ошибке, и выполнение функции завершается.

3.9.12 Функция изменения скидки

Функция editDiscounts предназначена для обновления существующей скидки в базе данных. Реализация функции представлена в листинге 3.25.

```

editDiscounts: async (req, res) => {
    try {
        const v_discount_up = await db.models.Discounts.update({
            discount: req.body.discount,
            count_for_dnt: req.body.count_for_dnt,
            promo_code: req.body.promo_code,
        }, {
            where: { id: req.params.id }
        })
        res.json({
            message: 'Скидка изменена'
        });
    } catch (error) {
        console.log(error);
        res.json({
            message: 'Не удалось обновить скидку'
        });
    }
},

```

Листинг 3.25 – Функция изменения скидки

Функция начинает свою работу с попытки обновить запись скидки в таблице Discounts. Она использует данные из тела запроса для обновления значений скидки, количества для скидки и промокода. Обновление происходит для записи с идентификатором, переданным в параметрах запроса.

Если обновление проходит успешно, функция возвращает сообщение об успешном изменении скидки. В случае возникновения ошибки во время выполнения функции, выводится сообщение об ошибке, и выполнение функции завершается.

3.9.13 Функция удаления скидки

Функция `deleteDiscounts` предназначена для удаления существующей скидки из базы данных. Реализация функции представлена в листинге 3.26.

```
deleteDiscounts: async (req, res) => {
  try {
    const v_check_id_discounts = await
db.models.Discounts.findOne({
      where: { id: req.params.id },
    })
    if (v_check_id_discounts != null) {
      await db.models.Discounts.destroy({where: {id:
req.params.id}})
      res.json({
        message: 'Скидка удалена'
      });
    }
    else {
      res.json({
        message: 'Не удалось удалить скидку',
      });
    }
  } catch (error) {
    console.log(error);
    res.json({
      message: 'Не удалось удалить скидку',
    });
  }
},
```

Листинг 3.26 – Функция удаления скидки

Функция начинает свою работу с проверки наличия скидки с указанным идентификатором в таблице `Discounts`. Если скидка с таким идентификатором существует, функция удаляет эту запись из базы данных. После успешного удаления скидки возвращается сообщение об успешном удалении.

Если скидка с указанным идентификатором не найдена, функция возвращает сообщение о том, что не удалось удалить скидку. В случае возникновения ошибки во время выполнения функции, выводится сообщение об ошибке, и выполнение функции завершается.

3.9.14 Редактирование любых статей

Участок кода, представленный в листинге 3.27, отвечает за отображение кнопки редактирования статьи на основе прав доступа текущего пользователя.

```
{ (user?.id === articles.User.id || user?.role === "admin") && (
    <div className={'flex gap-3 mt-4'}>
      <button className={'flex items-center justify-
center gap-2 text-xl text-lime-900 opacity-50'}>
        <Link to={`/${params.id}/edit`}>
          <AiTwotoneEdit />
        </Link>
      </button>
    </div>
  ) }
```

Листинг 3.27 – Редактирование любой статьи администратором

Условие `user?.id === articles.User.id || user?.role === "admin"` проверяет права текущего пользователя. Если пользователь является автором статьи или обладает ролью администратора, кнопка редактирования становится видимой.

Автор статьи может редактировать только свой контент, в то время как администратор обладает правом редактирования любых статей, что позволяет ему модерировать и обновлять контент на платформе. Если условие не выполняется, кнопка не отображается, предотвращая возможность несанкционированного изменения.

3.9.15 Удаление любых статей

Участок кода, представленный в листинге 3.28, отвечает за отображение кнопки удаления статьи на основе прав доступа текущего пользователя.

```
{user?.id === articles.User.id || user?.role === "admin" ? (
    <div className={'flex gap-3 mt-4'}>
      <button
        onClick={removeArticleHandler}
        className={'flex items-center justify-center
gap-2 text-xl text-lime-900 opacity-50'}>
        <AiFillDelete />
      </button>
    </div>
  ) : null
}
```

Листинг 3.28 – Удаление любой статьи администратором

Условие `user?.id === articles.User.id || user?.role === "admin"` проверяет права текущего пользователя. Если пользователь является автором статьи или обладает ролью администратора, кнопка удаления становится видимой.

Автор статьи может удалять только свой контент, в то время как администратор обладает правом удаления любых статей, что позволяет ему модерировать платформу. При нажатии на кнопку вызывается обработчик `removeArticleHandler`, который удаляет статью из базы данных. Если условие не выполняется, кнопка не отображается, предотвращая возможность несанкционированного удаления.

3.9.16 Удаление комментариев пользователей

Участок кода, представленный в листинге 3.29, отвечает за отображение кнопки удаления комментария на основе прав доступа текущего пользователя.

```
{user?.id === cmt.commentator || user?.role === "admin" ? (
    <button
      onClick={removeCommentHandler}
      className={'flex flex-wrap justify-center
gap-2 ml-5 mt-1.5 text-xl text-lime-900 opacity-50'}>
      <AiFillDelete />
    </button>
  ): null
}
```

Листинг 3.29 – Удаление любых комментариев администратором

Условие `{user?.id === cmt.commentator || user?.role === "admin"}` проверяет права текущего пользователя. Если пользователь является автором комментария или обладает ролью администратора, кнопка удаления становится доступной.

Автор комментария может удалять только свои записи, тогда как администратор обладает правом удаления любых комментариев. Это предоставляет администрации возможность модерировать дискуссии, удаляя неподобающие или ненужные записи.

При выполнении условия кнопка отображается. При нажатии на нее вызывается обработчик `removeCommentHandler`, который удаляет комментарий из базы данных. Если условие не выполняется, кнопка не отображается, исключая возможность несанкционированного удаления.

3.10 Структура клиентской части

3.10.1 Реализация структуры проекта

Клиентская часть приложения реализована с использованием компонентного подхода. Основная логика и элементы пользовательского интерфейса размещены в директории `src`. Директории представлены в таблице 3.6.

Таблица 3.6 – Директории проекта в папке src и их описание

Директория	Описание
components	Содержит компоненты приложения, такие как кнопки, элементы навигации, элементы управления, а также отдельные структурные элементы. Также включает элементы для отображения различных сущностей, таких как статьи или комментарии.
style	Содержит CSS-стили, используемые для стилизации отдельных компонентов.
image	Содержит статические изображения и SVG-иконки, используемые в интерфейсе.
pages	Содержит страницы приложения, соответствующие различным маршрутам, такие как главная страница, страницы для добавления, обновления и просмотра статей, скидок, пунктов приема и других сущностей. Эти страницы обрабатывают основные пользовательские действия.
redux	Реализует управление состоянием приложения. Включает директорию для различных фич приложения, таких как управление статьями, комментариями, скидками, пунктами приема и другими сущностями. Содержит отдельные slice для обработки состояния и store.js для конфигурации хранилища.
utils	Содержит вспомогательные модули и утилиты.

3.10.2 Реализация компонент

Компоненты в проекте играют центральную роль, предоставляя гибкие и переиспользуемые элементы пользовательского интерфейса. Они включают как небольшие элементы (например, кнопки), так и более сложные структуры (например, формы или меню). В таблица 3.7 представлено описание компонент.

Таблица 3.7 – Описание компонент

Компонента	Описание
AllDiscountItem	Отображает отдельный элемент скидки, предоставляя информацию о доступной скидке для пользователей.
ArticleItem	Отображает информацию о статье, включая её содержимое и автора. Используется в списке статей.
Button	Универсальная кнопка, стилизованная для использования в различных частях приложения.
CommentItem	Отображает отдельный комментарий, включающий текст и данные о пользователе, оставившем комментарий.
DiscountItem	Отображает данные о скидке, включая её название, описание и количество требуемых баллов.

Продолжение таблицы 3.7

Header	Реализует верхний колонтитул приложения, отображая название и основные навигационные элементы.
Layout	Компонент для управления общей структурой страниц, объединяющий шапку, меню и контент.
MarksItem	Отображает данные о виде вторсырья, включая название, категорию и другие детали.
MenuItem	Отображает отдельный пункт меню для навигации по страницам приложения.
MobileMenu	Реализует адаптивное меню для мобильной версии приложения.
Navbar	Реализует навигационную панель с основными ссылками и элементами управления.
NavItam	Компонент для создания отдельного элемента в навигации.
NavMenu	Отображает выпадающее меню с основными разделами приложения.
PointItem	Отображает данные о пункте приема, включая название, адрес и контактную информацию.
PointMarkItem	Реализует отображение данных о связи между пунктом приема и видом вторсырья.
PromoCodeItem	Отображает отдельный промокод с описанием и связанными скидками.
RecycleCamera	Реализует функциональность камеры для распознавания и отправки информации о вторсырье.
Wrapper	Обеспечивает структурирование компонентов, добавляя базовую разметку или стили для детей.

3.10.3 Реализация хранилища при помощи библиотеки Redux ToolKit

В приложении данные хранятся и управляются с использованием библиотеки Redux Toolkit (RTK). Конфигурация хранилища объединяет стандартные редьюсеры и редьюсеры RTK Query. Основной механизм хранения базируется на configureStore, который подключает все редьюсеры, middleware для обработки запросов и расширяет функциональность RTK Query.

Настроенные редьюсеры обрабатывают состояние и запросы следующим образом:

- стандартные редьюсеры (authSlice, userSlice и т.д.) управляют локальным состоянием приложения, связанным с конкретными сущностями;
- RTK Query редьюсеры (например, receptionSlice) обеспечивают автоматическое кэширование, инвалидацию данных и управление состоянием загрузки.

Описание всех редьюсеров представлено в таблице 3.8.

Таблица 3.8 – Описание редьюсеров

Редьюсер	Описание
authSlice	Управляет состоянием аутентификации пользователя (вход, выход, токены).
userSlice	Сохраняет данные о пользователях (профили, настройки).
articleSlice	Обрабатывает статьи (создание, получение, обновление, удаление).
commentSlice	Управляет состоянием комментариев (добавление, удаление).
discountSlice	Обеспечивает управление скидками (создание, изменение, получение данных).
markSlice	Управляет данными о видах вторсырья (создание, обновление, удаление).
pointSlice	Обрабатывает пункты приема (добавление, изменение, получение информации).
point_markSlice	Управляет связью между пунктами приема и видами вторсырья.
promo_codeSlice	Обеспечивает управление промокодами (создание, применение)
receptionSlice	Работает с данными о сдаче вторсырья (добавление, получение, изменение).
secretkeySlice	Управляет секретными ключами для пунктов приема.
weightSlice	Обрабатывает записи о проверке веса видов вторсырья.

3.11 Выводы по разделу

1. Сервер разработан на платформе Node.js с использованием фреймворка Express.js, в совокупности это обеспечивает гибкость и производительность приложению.

2. В качестве СУБД использована MySQL для хранения данных о пользователях, домах, квартирах, бронированиях и токенах.

3. Рассмотрены сторонние сервисы для расширения функциональности веб-приложения: SMTP-сервер, работа с загрузкой изображений.

4. Структура приложения основана на модульном подходе с применением современных библиотек для клиентской и серверной частей.

5. Для передачи данных используется JSON для текстовой информации. Для обработки мультимедиа, таких как изображения, используется формат FormData [42], который применяется при создании или изменении видов вторсырья.

6. Реализованы все функции для всех ролей: гостя, пользователя, администратора. Количество функций веб-приложения составило 29.

4 Тестирование веб-приложения

4.1 Функциональное тестирование

Для проведения корректной работы функций разработанного веб-приложения было произведено ручное тестирование, описание и итоги представлены в таблице 4.1.

Таблица 4.1 – Описание функциональных тестов

Номер теста	Функция веб-приложения	Описание теста	Ожидаемый результат	Статус теста
1	Регистрация	Действие: Ввести корректные данные (username, email, password) и отправить форму.	Данные пользователя заносятся в базу данных.	Пройден
2	Авторизация	Действие: Ввести корректные данные для входа (username, password) и нажать "Войти".	Пользователь успешно авторизуется и попадает на главную страницу.	Пройден
3	Просмотр статей	Действие: Перейти на страницу с категориями статей.	Статьи отображаются на странице.	Пройден
4	Просмотр пунктов приёма	Действие: Открыть страницу с определенным видом вторсырья.	Все доступные пункты приёма для данного вида вторсырья отображаются на странице.	Пройден
5	Просмотр скидок	Действие: Открыть страницу с актуальными скидками.	Все скидки отображаются с актуальной информацией.	Пройден
6	Обмен баллов на скидки в сервисах	Действие: Выбрать подходящую скидку и нажать на кнопку "Промокод".	Баллы обменены на скидку, статус обновлён.	Пройден
7	Добавление статьи	Действие: Перейти в раздел создания статьи, ввести все необходимые данные и отправить.	Статья добавляется и отображается в списке статей.	Пройден

Продолжение таблицы 4.1

8	Изменение статьи	Действие: Перейти на страницу редактирования статьи, изменить её содержание и сохранить изменения.	Статья изменяется, изменения отображаются на странице.	Пройден
9	Удаление статьи	Действие: Нажать кнопку "Удалить" рядом со статьёй и подтвердить действие.	Статья удаляется из списка и больше не отображается.	Пройден
10	Добавление комментария	Действие: Написать комментарий и отправить его под статьёй.	Комментарий добавляется и отображается на странице статьи.	Пройден
11	Удаление комментария	Действие: Нажать кнопку "Удалить" возле комментария.	Комментарий удаляется из статьи.	Пройден
12	Добавление отметки "Нравится"	Действие: Нажать кнопку "Нравится" под статьёй.	Отметка "Нравится" появляется, и количество лайков увеличивается.	Пройден
13	Отметка сдачи вторсырья	Действие: В разделе сдачи вторсырья отметить сдачу.	Результат сдачи вторсырья отображается на странице.	Пройден
14	Добавление пункта приёма	Действие: Перейти в форму добавления пункта приёма, заполнить поля и отправить.	Пункт приёма добавляется и отображается на карте/странице.	Пройден
15	Изменение времени работы	Действие: Перейти в раздел редактирования пункта приёма и изменить время работы.	Время работы пункта приёма изменяется.	Пройден
16	Удаление пункта приёма	Действие: Нажать кнопку "Удалить" на странице редактирования пункта приёма.	Пункт приёма удаляется из системы.	Пройден
17	Добавление ключей	Действие: Ввести ключи на странице добавления ключей и сохранить.	Ключи успешно добавляются в систему.	Пройден

Продолжение таблицы 4.1

18	Изменение ключей	Действие: Изменить данные ключей на странице редактирования и сохранить.	Изменённые ключи сохраняются и отображаются в списке.	Пройден
19	Добавление вторсырья	Действие: Ввести информацию о новом виде вторсырья и сохранить.	Новое вторсырье добавляется в систему.	Пройден
20	Изменение вторсырья	Действие: Изменить данные существующего вторсырья и сохранить.	Изменения вторсырья отображаются в системе.	Пройден
21	Удаление вторсырья	Действие: Удалить запись о вторсырье через интерфейс.	Вторсырьё удаляется из базы данных.	Пройден
22	Добавление проверки веса	Действие: Добавить информацию о весе вторсырья и подтвердить.	Проверка веса сохраняется и отображается.	Пройден
23	Изменение проверки веса	Действие: Изменить введённый вес для вторсырья.	Изменённый вес сохраняется в системе.	Пройден
24	Добавление скидки	Действие: Ввести параметры скидки и добавить её.	Скидка успешно добавляется и отображается в списке.	Пройден
25	Изменение скидки	Действие: Изменить параметры скидки и сохранить изменения.	Изменения скидки отображаются на странице.	Пройден
26	Удаление скидки	Действие: Удалить скидку через интерфейс управления.	Скидка удаляется из списка активных предложений.	Пройден
27	Редактирование любых статей	Действие: Перейти в раздел редактирования, изменить содержание статьи и сохранить.	Статья обновляется, и изменения отображаются на странице.	Пройден
28	Удаление любых статей	Действие: Нажать кнопку удаления на странице статьи.	Статья удаляется.	Пройден

Продолжение таблицы 4.1

29	Удаление комментариев пользователей	Действие: Удалить комментарий от пользователя через учетную запись администратора.	Комментарий удален.	Пройден
----	-------------------------------------	--	---------------------	---------

4.2 Выводы по разделу

1. Разработано 29 тестов, охватывающих все функции веб-приложения.
2. Проверена корректность обработки данных при выполнении пользовательских и администраторских операций взаимодействия с базой данных и сторонними сервисами.
3. Все тесты завершены успешно, что подтверждает готовность приложения к эксплуатации.
4. Покрытие кода тестами составило 100%.

5 Руководство пользователя

При запуске веб-приложения неавторизованный пользователь будет автоматически перенаправлен на главную страницу приложения. Главная страница является информационной точкой входа, где пользователь может получить краткую информацию о проекте.

Пользователь может ознакомиться с описанием проекта, его значимостью и преимуществами использования данного приложения. Для наглядного представления главной страницы веб-приложения, на рисунке 5.1 отображается внешний вид и расположение информационных элементов на странице.

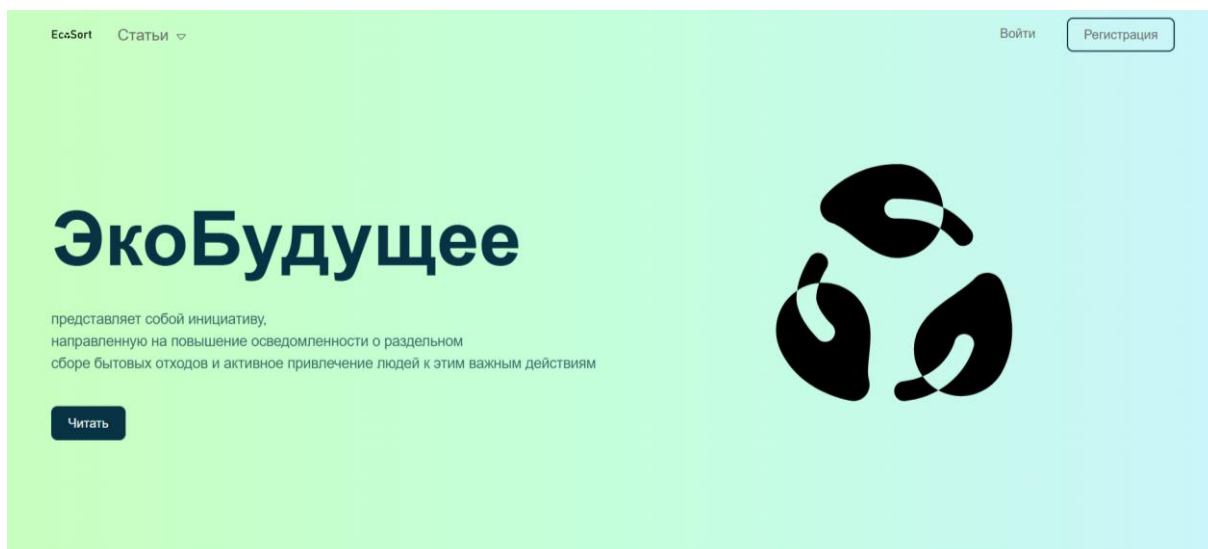


Рисунок 5.1 – Главная страница

5.1 Регистрация

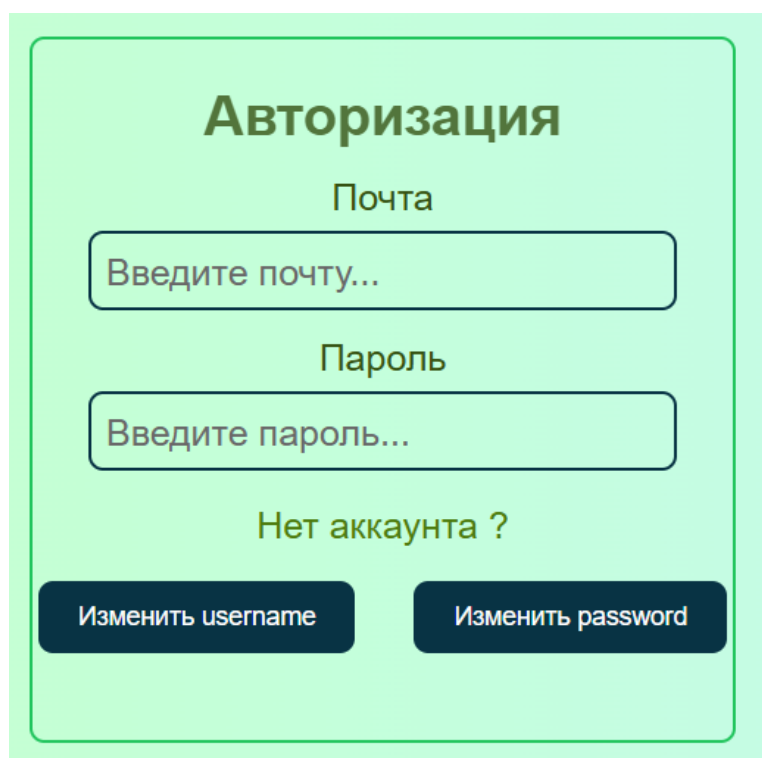
Если гость хочет получить более широкий функционал, он может нажать на кнопку «Регистрация» в навигационном меню. Форма регистрации представлена на рисунке 5.2.

Рисунок 5.2 – Форма регистрации

Здесь необходимо заполнить форму, указав свои данные. После заполнения формы, гость нажимает кнопку «Зарегистрироваться», которая инициирует процесс проверки введённых данных и после этого, если все успешно, пользователь перенаправляется на страницу входа.

5.2 Авторизация

После успешной регистрации требуется выполнить вход на сайт, необходимо корректно заполнить данные email и пароль. Форма входа в аккаунт представлена на рисунке 5.3.



The image shows a login form titled "Авторизация" (Authorization) in a light blue box. It contains two input fields: "Почта" (Email) with the placeholder "Введите почту..." and "Пароль" (Password) with the placeholder "Введите пароль...". Below the password field is a link "Нет аккаунта ?" (No account?). At the bottom are two dark blue buttons: "Изменить username" (Change username) and "Изменить password" (Change password).

Рисунок 5.3 – Форма авторизации

После успешного входа пользователь будет отправлен на главную страницу.

5.3 Просмотр статей

После успешного входа на сайт пользователь может перейти к просмотру статей. Для этого необходимо выбрать соответствующий раздел в меню или на главной странице. Страница с просмотром статей представлена на рисунке 5.4.

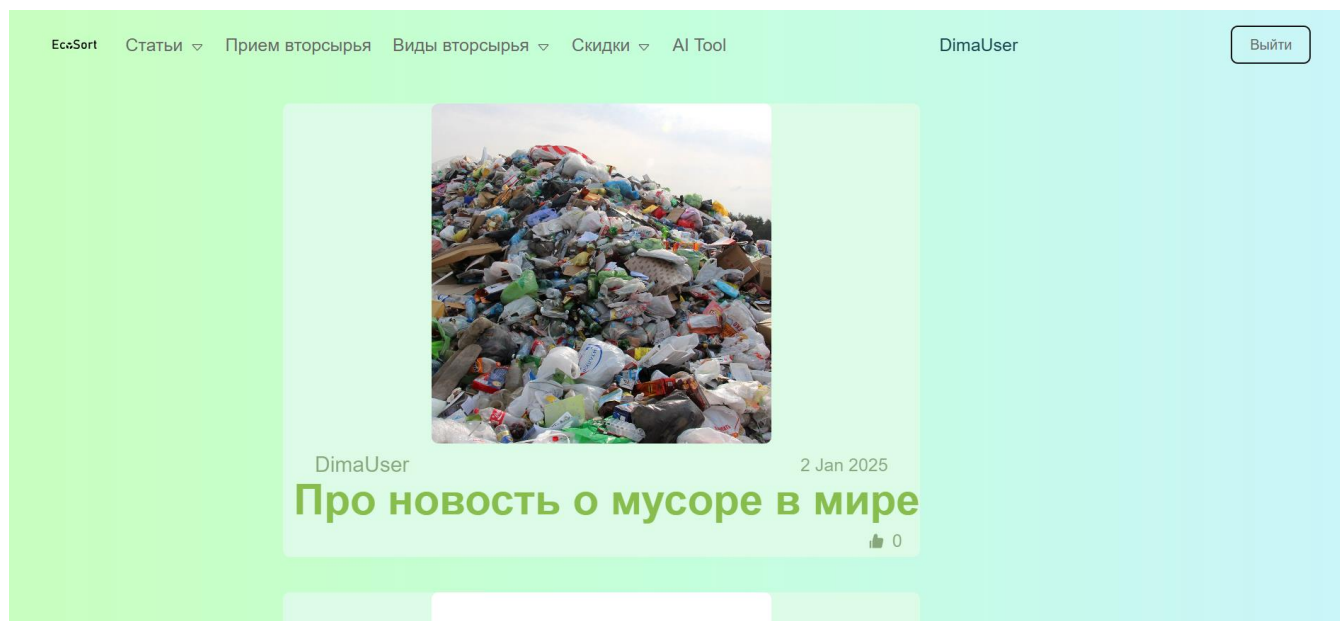


Рисунок 5.4 – Просмотр списка статей

Пользователь может выбрать интересующую его статью из списка доступных материалов. После выбора статьи откроется страница с полным текстом статьи, где можно ознакомиться с содержанием (рисунок 5.5).

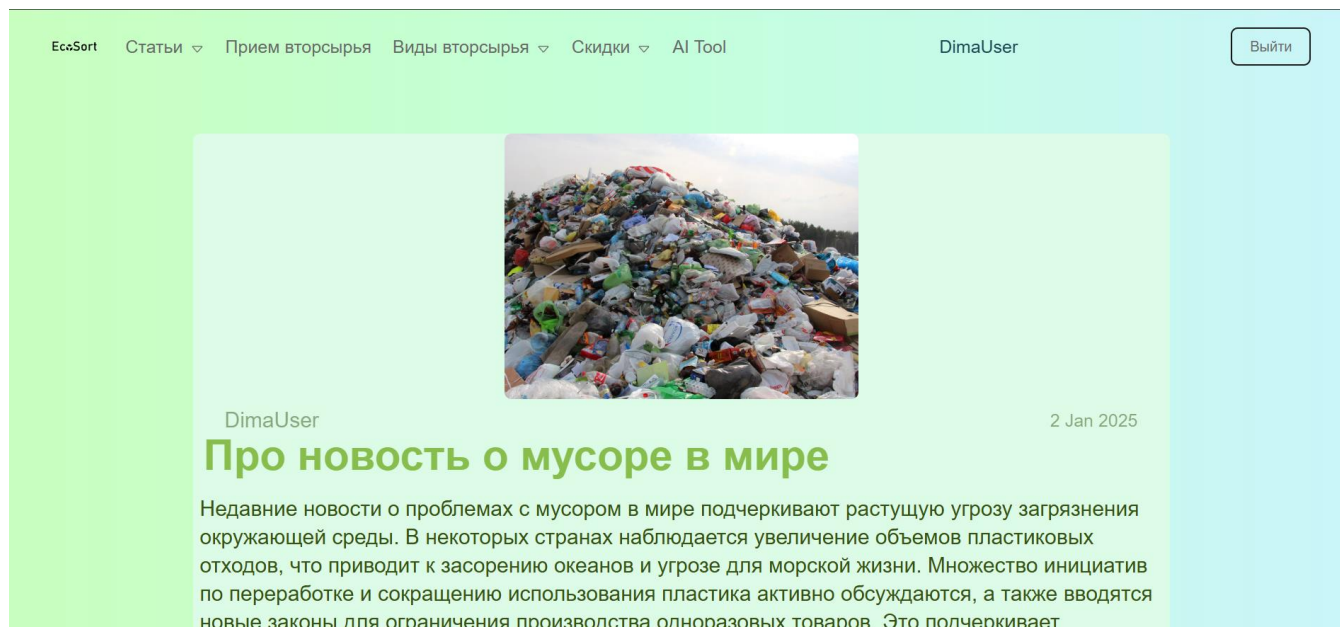


Рисунок 5.5 – Просмотр статьи

На странице просмотра статьи пользователь может увидеть заголовок статьи, дату публикации, автора и полный текст статьи.

5.4 Просмотр пунктов приема

Пользователь также может просматривать пункты приема вторсырья. Для этого необходимо выбрать соответствующий вид вторсырья, и он увидит все

пункты, которые относятся к данному виду. Страница с просмотром пунктов приема представлена на рисунке 5.6.

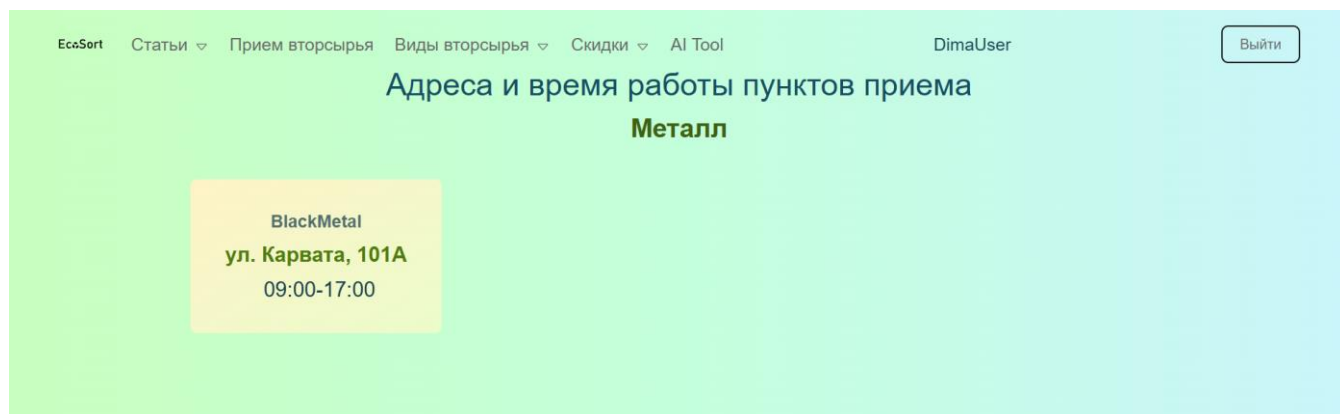


Рисунок 5.6 – Просмотр статьи

На странице просмотра пунктов приема пользователь может увидеть список доступных пунктов приема вторсырья. Каждый пункт приема отображается с указанием его названия, адреса, времени работы и видов принимаемого вторсырья.

5.5 Просмотр скидок

Пользователь также может просматривать доступные скидки. Для этого необходимо выбрать соответствующий раздел в меню или на главной странице. Страница с просмотром скидок представлена на рисунке 5.7.

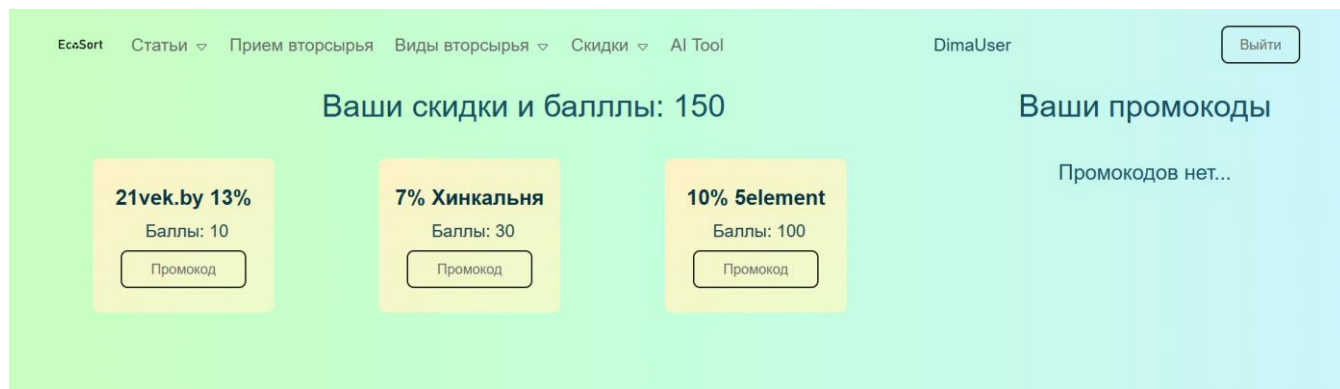


Рисунок 5.7 – Просмотр скидок

На странице просмотра скидок пользователь может увидеть список доступных скидок и акций.

5.6 Обмен баллов на скидки

Пользователь может обменивать свои накопленные баллы на скидки. Для этого необходимо выбрать нужную вам скидку и нажать на кнопку “Промокод” (рисунок 5.7).

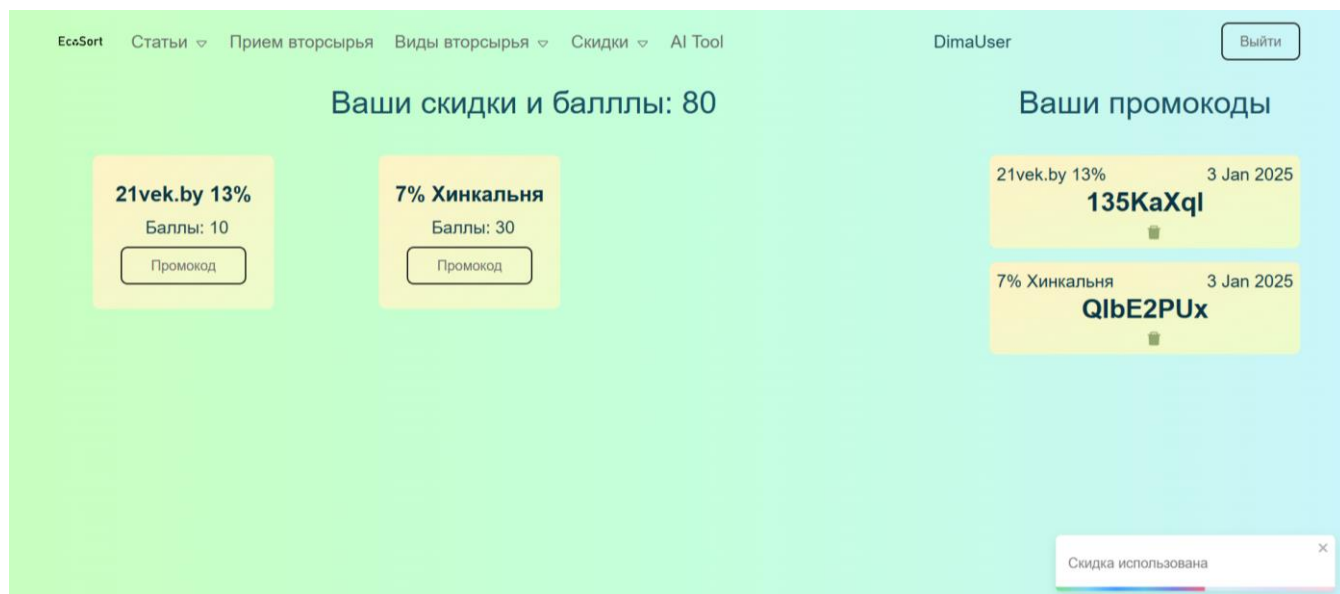


Рисунок 5.7 – Обмен баллов

Как можно видеть, обмен баллов произошел успешно и промокод отображается в правой части интерфейса.

5.7 Добавление статьи

Пользователь может добавлять новые статьи через специальный интерфейс. Для этого необходимо выбрать соответствующий раздел в меню или на главной странице. Интерфейс добавления статьи представлен на рисунке 5.8.

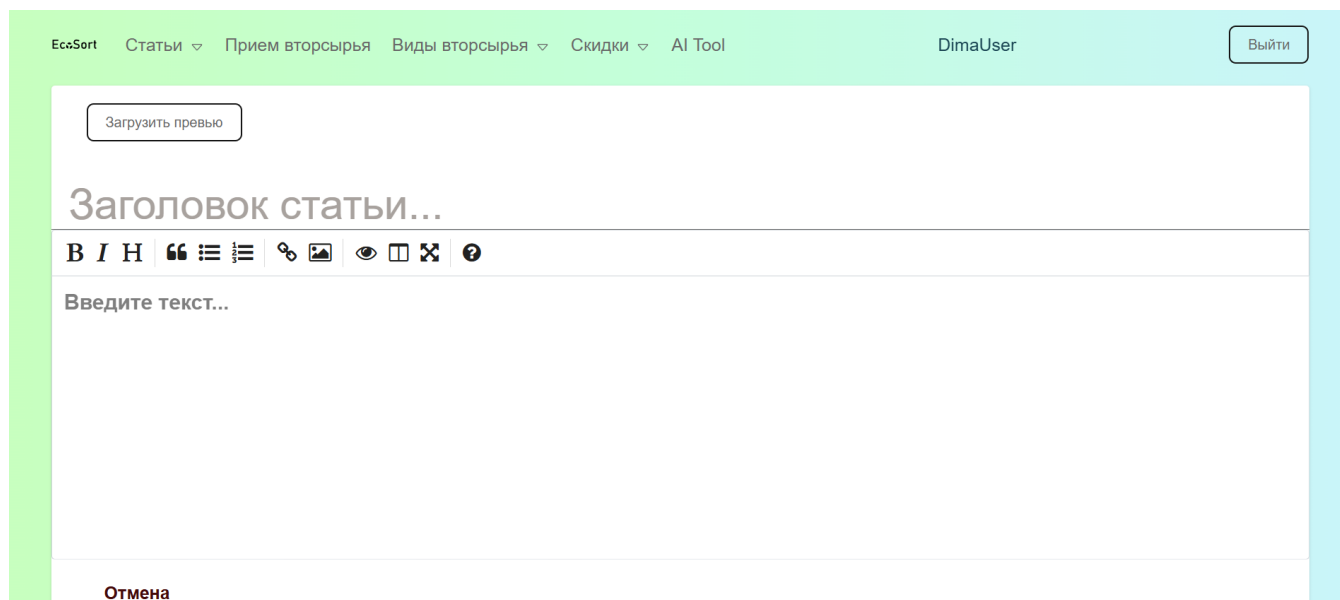


Рисунок 5.8 – Интерфейс добавления статьи

На странице добавления статьи пользователь может загрузить превью статьи, ввести заголовок и текст статьи. Превью статьи можно загрузить, нажав на кнопку

"Загрузить превью" и выбрав файл изображения. После загрузки превью отображается на странице, и пользователь может удалить его, нажав на кнопку "Удалить".

Пользователь также может ввести заголовок статьи в соответствующее поле и текст статьи в редактор. Редактор поддерживает форматирование текста, такие как жирный шрифт, курсив, списки и ссылки. После заполнения всех полей пользователь может сохранить статью, нажав на кнопку "Сохранить". Если все поля заполнены корректно, статья будет добавлена, и пользователь будет перенаправлен на страницу со списком статей.

Если пользователь хочет отменить добавление статьи, он может нажать на кнопку "Отмена", чтобы вернуться на главную страницу.

5.8 Изменение статьи

Пользователь может редактировать существующие статьи через специальный интерфейс. Интерфейс изменения статьи представлен на рисунке 5.9.

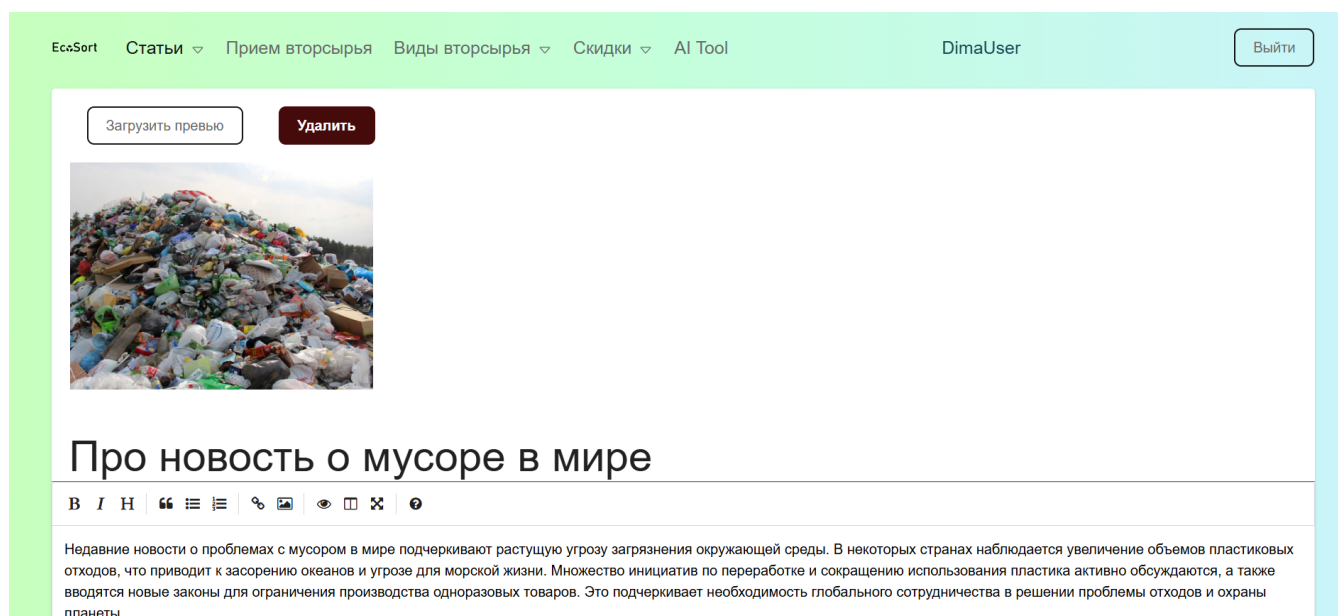


Рисунок 5.9 – Интерфейс изменения статьи

На странице изменения статьи пользователь может загрузить новое превью статьи, изменить заголовок и текст статьи. Превью статьи можно загрузить, нажав на кнопку "Загрузить превью" и выбрав файл изображения. После загрузки превью отображается на странице, и пользователь может удалить его, нажав на кнопку "Удалить".

Пользователь также может изменить заголовок статьи в соответствующем поле и текст статьи в редакторе. Редактор поддерживает форматирование текста, такие как жирный шрифт, курсив, списки и ссылки. После внесения всех изменений пользователь может сохранить статью, нажав на кнопку "Сохранить". Если все поля заполнены корректно, статья будет обновлена, и пользователь будет перенаправлен на страницу со списком статей.

Если пользователь хочет отменить изменения, он может нажать на кнопку "Отмена", чтобы вернуться на страницу со списком статей.

5.9 Удаление статьи

Пользователь может удалить статью, если он является автором статьи или имеет роль администратора. Для этого необходимо выбрать соответствующий раздел в меню или на главной странице и перейти на страницу статьи. На странице статьи будет доступна кнопка для удаления статьи (рисунок 5.10).



Рисунок 5.10 – Удаление статьи

5.10 Добавление комментария

Пользователь может добавлять комментарии к статьям, чтобы выразить свое мнение или задать вопросы. Для этого необходимо перейти на страницу статьи. На странице статьи будет доступна форма для добавления комментария (рисунок 5.11).



Рисунок 5.11 – Добавление комментария

После добавления комментария пользователь увидит уведомление о том, что комментарий был успешно добавлен.

5.11 Удаление комментария

Пользователь может удалить свои комментарии или комментарии других пользователей, если он является автором комментария или имеет роль администратора. Это позволяет поддерживать чистоту и актуальность обсуждений под статьями. Для удаления комментария необходимо нажать на соответствующую иконку удаление (рисунок 5.12).

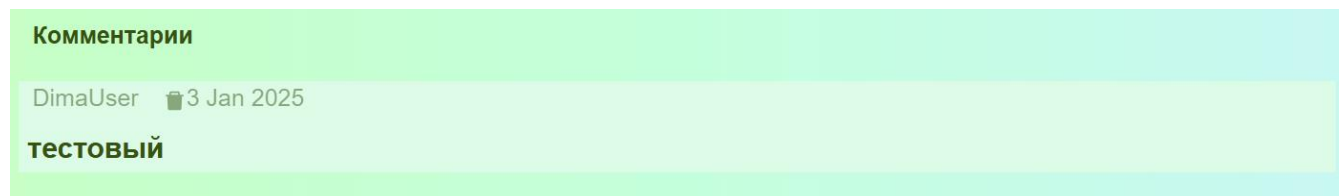


Рисунок 5.12 – Удаление комментария

После подтверждения удаления комментарий будет удален из системы, и пользователь увидит уведомление о том, что комментарий был успешно удален.

5.12 Добавление отметки “Нравится”

Пользователь может выразить свое одобрение статье, добавив отметку "Нравится". Чтобы добавить отметку "Нравится", пользователь должен нажать на кнопку с иконкой лайка (рисунок 5.13).



Рисунок 5.13 – Добавление отметки “Нравится”

Если пользователь не авторизован, он увидит уведомление с просьбой авторизоваться для оценки статьи.

После нажатия на кнопку отметка "Нравится" будет добавлена к статье, и количество лайков увеличится.

5.13 Отметка сдачи вторсырья

Пользователь может отметить сдачу вторсырья на пункте приема и получить баллы за это. Для этого необходимо выбрать соответствующий раздел в меню или на главной странице и перейти к форме сдачи вторсырья. Форма сдачи вторсырья представлена на рисунке 5.14.

Рисунок 5.14 – Форма сдачи вторсырья

На странице сдачи вторсырья пользователь может ввести ключ пункта приема и ключ веса. После ввода ключей система проверяет их корректность и соответствие пункту приема. Если ключи действительны и соответствуют пункту приема, система проверяет вид вторсырья и его вес.

Результат сдачи вторсырья отображается на рисунке 5.15.

Рисунок 5.15 – Результат сдачи

На странице результата сдачи пользователь увидит количество начисленных баллов, количество килограммов нового вторсырья и общее количество баллов на его счету.

5.14 Добавление ключей

Администратор может добавлять новые секретные ключи через специальный интерфейс для добавления пункта приема. Форма добавления секретного ключа представлена на рисунке 5.16.

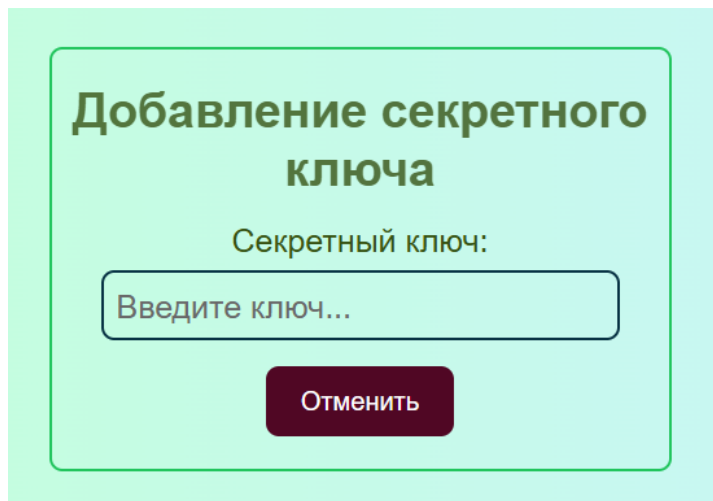
The image shows a web form titled "Добавление секретного ключа" (Add Secret Key) in a bold, dark green font. Below the title, the text "Секретный ключ:" (Secret key:) is displayed. Underneath is a light blue input field with the placeholder text "Введите ключ..." (Enter key...). At the bottom of the form is a dark red button with the white text "Отменить" (Cancel).

Рисунок 5.16 – Добавление секретного ключа

После нажатия на кнопку "Добавить" система проверяет, существует ли уже такой ключ в базе данных. Если ключ уникален, он будет добавлен в систему, и пользователь увидит уведомление о том, что ключ был успешно добавлен. Если ключ уже существует, пользователь увидит сообщение об ошибке и сможет ввести другой ключ.

5.15 Изменение ключей

Администратор может изменять секретные ключи, связанные с пунктами приема вторсырья. Форма изменения секретного ключа представлена на рисунке 5.17.

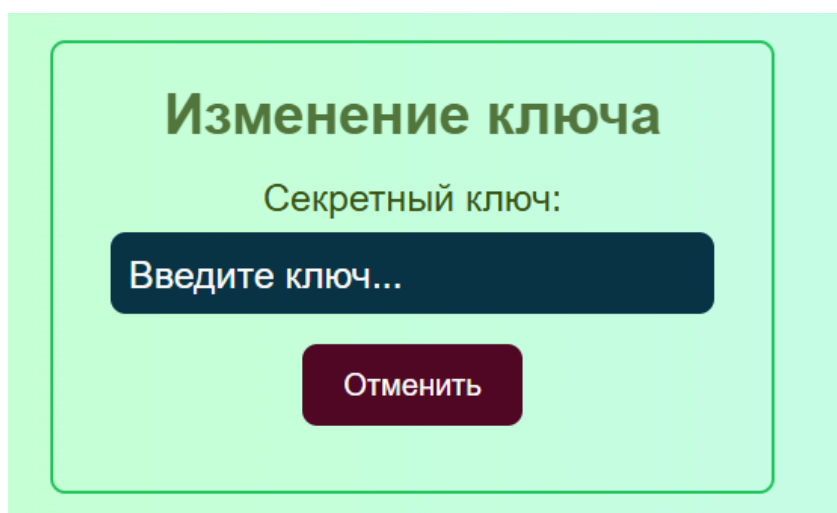
The image shows a web form titled "Изменение ключа" (Change Key) in a bold, dark green font. Below the title, the text "Секретный ключ:" (Secret key:) is displayed. Underneath is a dark blue input field with the placeholder text "Введите ключ..." (Enter key...). At the bottom of the form is a dark red button with the white text "Отменить" (Cancel).

Рисунок 5.17 – Изменение секретного ключа

На странице изменения секретного ключа пользователь может ввести новый ключ для выбранного пункта приема. После ввода нового ключа пользователь может нажать на кнопку "Сохранить", чтобы обновить ключ в системе.

Если ключ уникален, он будет обновлен в системе, и пользователь увидит уведомление о том, что ключ был успешно обновлен. Если ключ уже используется, пользователь увидит сообщение об ошибке и сможет ввести другой ключ.

5.16 Добавление пункта приема

Администратор может добавлять новые пункты приема вторсырья через специальный интерфейс. Форма добавления пункта приема представлена на рисунке 5.18.

The form is titled "Добавление пункта приема" (Adding a reception point) in bold black text. It contains several input fields and a button:

- Имя:** (Name) - A text input field with the placeholder "Введите имя..." (Enter name...).
- Адрес:** (Address) - A text input field with the placeholder "Введите адрес..." (Enter address...).
- Виды вторсырья:** (Types of waste) - A text input field with the placeholder "Введите вторсырье..." (Enter waste...).
- Время работы:** (Working hours) - A time selection interface consisting of two time pickers (each with "--:--" and a clock icon) separated by a minus sign.
- Ссылка на карту:** (Map link) - A text input field with the placeholder "Введите ссылку..." (Enter link...).
- Отменить** (Cancel) - A red button with white text.

Рисунок 5.18 – Форма добавления пункта приема

На странице добавления пункта приема администратор может ввести необходимую информацию о новом пункте приема. Форма включает несколько полей для заполнения: имя пункта приема, адрес, виды принимаемого вторсырья, время работы и ссылка на карту. Пользователь может заполнить эти поля, чтобы создать новый пункт приема.

Если администратор хочет отменить добавление пункта приема, он может нажать на кнопку "Отменить", чтобы очистить форму и вернуться к предыдущему состоянию.

5.17 Изменение времени работы пункта

Администратор может изменять время работы пунктов приема вторсырья. На странице с пунктами приема будет доступна кнопка для изменения времени работы. После нажатия на кнопку редактирования пользователь будет перенаправлен на страницу изменения времени работы пункта (рисунок 5.19).

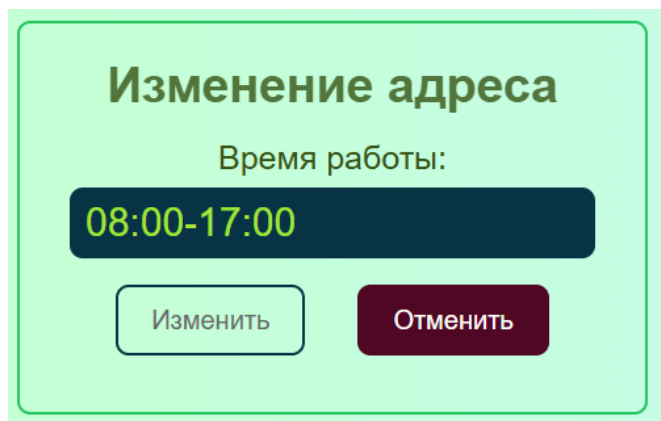


Рисунок 5.19 – Изменение времени работы пункта

На этой странице пользователь может ввести новое время работы пункта и сохранить изменения. Если все поля заполнены корректно, время работы пункта будет обновлено, и пользователь увидит уведомление о успешном изменении.

5.18 Удаление пункта приема

Администратор может удалять пункты приема вторсырья. На странице с пунктами приема будет доступна кнопка для удаления пункта (рисунок 5.20).

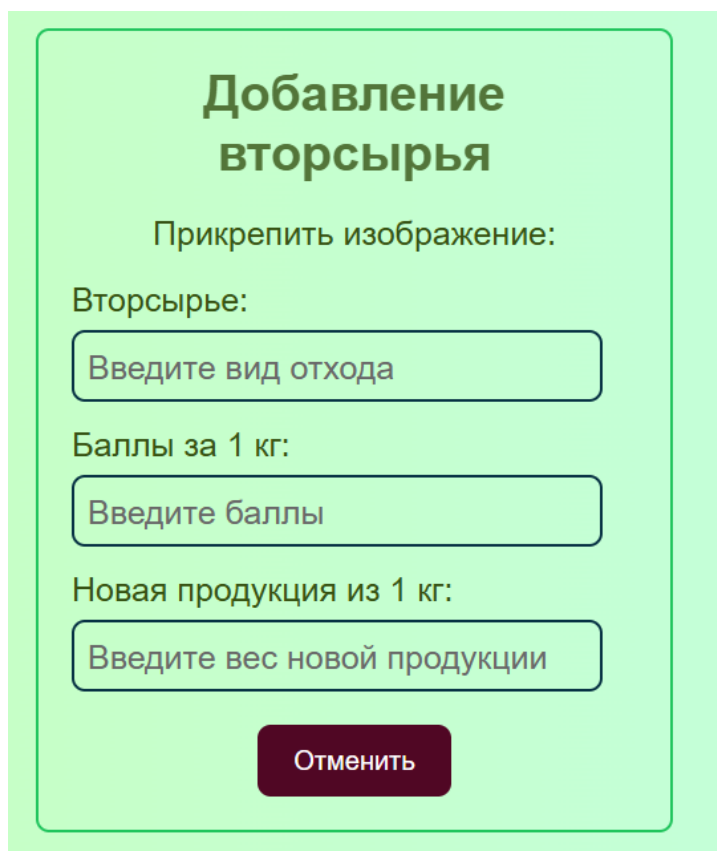


Рисунок 5.20 – Удаление пункта приема

После нажатия на кнопку удаления пункт приема будет удален из системы, и пользователь увидит уведомление о том, что пункт был успешно удален. Это позволяет администратору легко управлять списком пунктов приема и поддерживать его актуальность.

5.19 Добавление вида вторсырья

Администратор может добавлять новые виды вторсырья через специальный интерфейс. Форма добавления вида вторсырья представлена на рисунке 5.21.



**Добавление
вторсырья**

Прикрепить изображение:

Вторсырье:

Введите вид отхода

Баллы за 1 кг:

Введите баллы

Новая продукция из 1 кг:

Введите вес новой продукции

Отменить

Рисунок 5.21 – Форма добавления вида вторсырья

На странице добавления вида вторсырья пользователь может ввести информацию о новом виде вторсырья. Форма включает несколько полей для заполнения: название вторсырья, количество баллов за килограмм, количество новой продукции из килограмма и ссылка на изображение. Пользователь может заполнить эти поля, чтобы создать новый вид вторсырья.

После заполнения всех полей пользователь может нажать на кнопку "Добавить", чтобы сохранить информацию о новом виде вторсырья. Если все поля заполнены корректно, вид вторсырья будет добавлен в систему, и пользователь увидит уведомление о том, что вид вторсырья был успешно добавлен. Если такой вид вторсырья уже существует, пользователь увидит сообщение об ошибке и сможет ввести другой вид вторсырья.

Если пользователь хочет отменить добавление вида вторсырья, он может нажать на кнопку "Отменить", чтобы очистить форму и вернуться к предыдущему состоянию.

5.20 Изменение вида вторсырья

Администратор может изменять информацию о существующих видах вторсырья через специальный интерфейс. Форма изменения вида вторсырья представлена на рисунке 5.22.



Изменение вторсырья

Прикрепить изображение:

Удалить



Вторсырье:

Бумага

Баллы начисляемые за 1 кг:

100

Новая продукция из 1 кг:

1

Изменить Отменить

Рисунок 5.22 – Форма изменения вида вторсырья

После внесения всех изменений пользователь может нажать на кнопку "Сохранить", чтобы обновить информацию о виде вторсырья в системе. Если все поля заполнены корректно, вид вторсырья будет обновлен, и пользователь увидит уведомление о том, что вид вторсырья был успешно обновлен. Если возникнут ошибки валидации, пользователь увидит сообщения об ошибках и сможет исправить их.

5.21 Удаление вида вторсырья

Пользователь может удалять существующие виды вторсырья через специальный интерфейс. Для этого необходимо выбрать соответствующий раздел в меню или на главной странице и перейти к списку видов вторсырья. Форма удаления вида вторсырья представлена на рисунке 5.23.

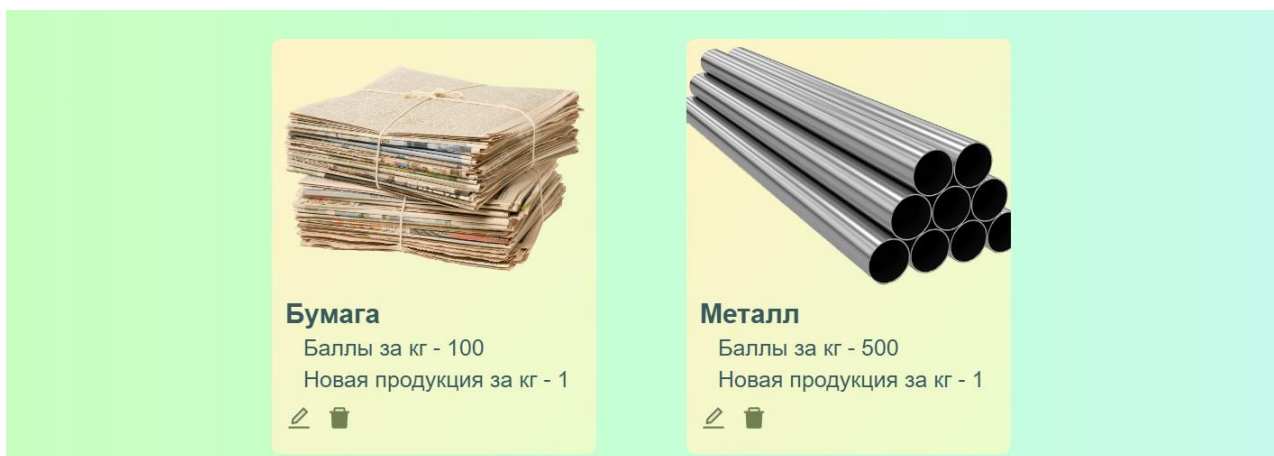


Рисунок 5.23 – Удаления вида вторсырья

На странице списка видов вторсырья каждый вид отображается с указанием его названия, количества баллов за килограмм, количества новой продукции из килограмма и ссылки на изображение. Если пользователь имеет права администратора, рядом с каждым видом вторсырья будет отображаться кнопка для удаления.

Для удаления вида вторсырья пользователь должен нажать на кнопку удаления. После подтверждения вид вторсырья будет удален из системы, и администратор увидит уведомление о том, что вид вторсырья был успешно удален.

5.22 Добавление проверки веса

Пользователь может добавлять новые ключи для проверки веса вторсырья через специальный интерфейс. Форма добавления проверки веса представлена на рисунке 5.24.

The image shows a form titled 'Добавление проверки веса' (Adding weight check). It contains three input fields: 'Вид отхода:' (Waste type), 'Вес:' (Weight), and 'Ключ:' (Key). Each field has a placeholder text 'Введите вид отхода', 'Введите вес', and 'Введите ключ' respectively. At the bottom of the form is a dark red button labeled 'Отменить' (Cancel).

Рисунок 5.24 – Добавление проверки веса

Если все поля заполнены корректно, ключ для проверки веса будет добавлен в систему, и администратор увидит уведомление о том, что ключ был успешно добавлен. Если такой ключ уже существует, администратор увидит сообщение об ошибке и сможет ввести новый ключ.

5.23 Изменение проверки веса

Пользователь может изменять существующие записи проверки веса вторсырья через специальный интерфейс. Форма изменения проверки веса представлена на рисунке 5.25.

The screenshot displays a web interface for managing scrap weight checks. A central modal window titled "Редактировать данные" (Edit data) is open, allowing the user to update an existing record. The modal contains three input fields: "Вид вторсырья:" (Type of scrap) with the value "Бумага" (Paper), "Вес:" (Weight) with the value "10", and "Ключ:" (Key) with the value "firstkey". Below these fields are two buttons: "Сохранить" (Save) and "Отмена" (Cancel). In the background, there are two main sections. On the left, a form for adding a new scrap type with fields for "Вторсырье:" (Scrap type), "Введите вид отхода" (Enter waste type), "Баллы за 1 кг:" (Points for 1 kg), "Введите баллы" (Enter points), "Новая продукция из 1 кг:" (New product from 1 kg), and "Введите вес новой продукции" (Enter weight of new product), along with an "Отменить" (Cancel) button. On the right, a form for adding a new weight check with fields for "Введите вид отхода" (Enter waste type), "Вес:" (Weight), "Введите вес" (Enter weight), "Ключ:" (Key), "Введите ключ" (Enter key), and an "Отменить" (Cancel) button. At the bottom, a table titled "Данные веса" (Weight data) lists existing records.

ID	Вид вторсырья	Вес	Ключ	Действия
1	Бумага	10	firstkey	Редактировать

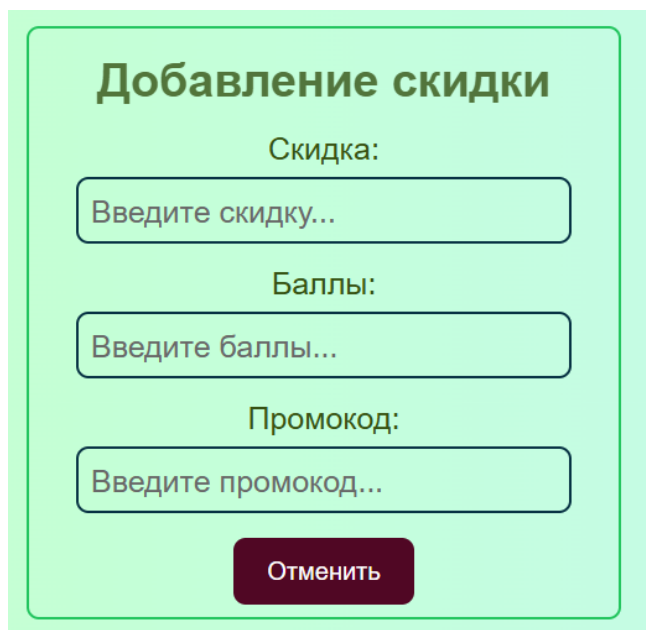
Рисунок 5.25 – Добавление проверки вторсырья

Для изменения записи проверки веса пользователь должен нажать на кнопку "Редактировать". После этого откроется модальное окно, где пользователь может ввести новые данные для вида вторсырья, веса и ключа. После внесения всех изменений пользователь может нажать на кнопку "Сохранить", чтобы обновить запись в системе. Если все поля заполнены корректно, запись будет обновлена.

5.24 Добавление скидки

Пользователь может добавлять новые скидки через специальный интерфейс. Для этого необходимо выбрать соответствующий раздел в меню или на главной странице. Форма добавления скидки представлена на рисунке 5.26.

На странице добавления скидки администратор может ввести информацию о новой скидке. Форма включает несколько полей для заполнения: размер скидки, количество баллов, необходимых для получения скидки, и промокод.



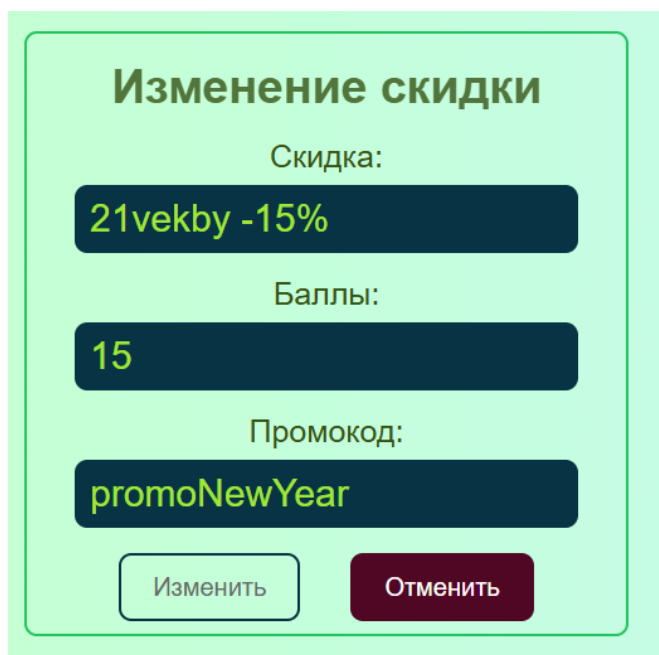
The form is titled "Добавление скидки" (Adding discount) in bold green text. It contains three input fields with green borders and light green backgrounds. The first field is labeled "Скидка:" (Discount:) and contains the placeholder text "Введите скидку..." (Enter discount...). The second field is labeled "Баллы:" (Points:) and contains the placeholder text "Введите баллы..." (Enter points...). The third field is labeled "Промокод:" (Promo code:) and contains the placeholder text "Введите промокод..." (Enter promo code...). Below the input fields is a dark red button with white text labeled "Отменить" (Cancel).

Рисунок 5.26 – Форма добавления скидки

После заполнения всех полей администратор может нажать на кнопку "Добавить", чтобы сохранить информацию о новой скидке.

5.25 Изменение скидки

Пользователь может изменять существующие скидки через специальный интерфейс. Форма изменения скидки представлена на рисунке 5.27.



The form is titled "Изменение скидки" (Changing discount) in bold green text. It contains three input fields with dark blue backgrounds and green borders. The first field is labeled "Скидка:" (Discount:) and contains the text "21vekby -15%". The second field is labeled "Баллы:" (Points:) and contains the text "15". The third field is labeled "Промокод:" (Promo code:) and contains the text "promoNewYear". Below the input fields are two buttons: a light green button with a green border labeled "Изменить" (Change) and a dark red button with white text labeled "Отменить" (Cancel).

Рисунок 5.27 – Форма добавления скидки

После внесения всех изменений пользователь может нажать на кнопку "Сохранить", чтобы обновить информацию о скидке в системе.

5.26 Удаление скидки

Администратор может удалять существующие скидки через специальный интерфейс. Для этого необходимо выбрать соответствующий раздел в меню управления скидкой (рисунок 5.28).

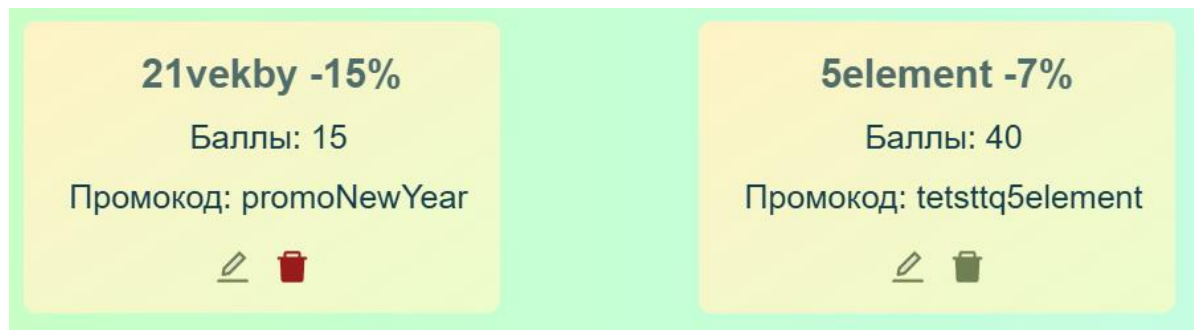


Рисунок 5.28 – Удаление скидки

Для удаления скидки пользователь должен нажать на кнопку удаления. После подтверждения скидка будет удалена из системы, и администратор увидит уведомление о том, что скидка была успешно удалена.

5.27 Редактирование любых статей

Администраторы имеют возможность редактировать любые статьи на платформе, независимо от того, кто является автором статьи (рисунок 5.29).

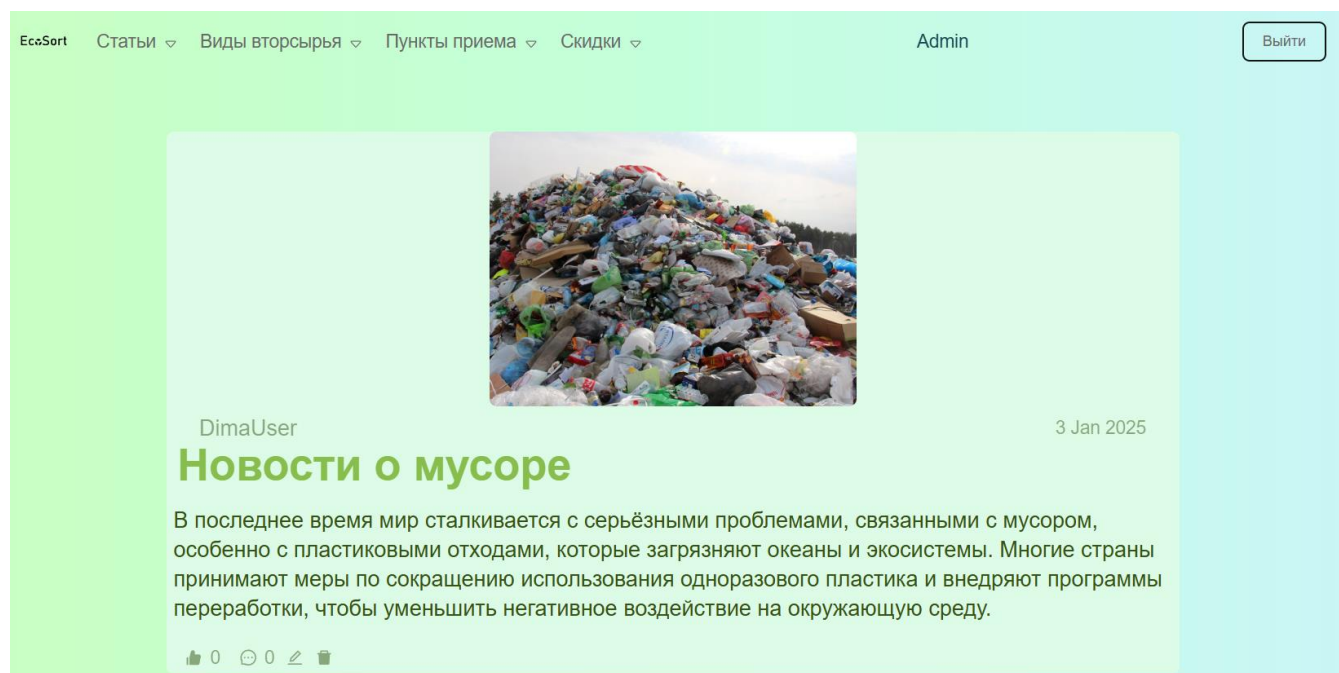


Рисунок 5.29 – Редактирование статьи пользователя

На странице статьи, если пользователь имеет роль администратора, рядом с каждой статьей будет отображаться кнопка для редактирования. Администратор может нажать на эту кнопку, чтобы перейти к форме редактирования статьи.

5.28 Удаление любых статей

Администраторы имеют возможность удалять любые статьи на платформе, независимо от того, кто является автором статьи (рисунок 5.30).

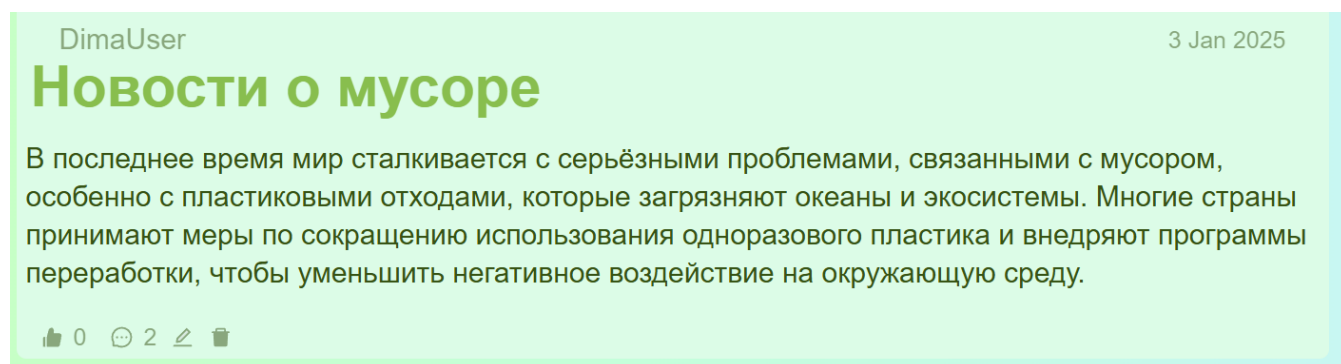


Рисунок 5.30 – Удаление статьи пользователя

На странице статьи, если пользователь имеет роль администратора, рядом с каждой статьей будет отображаться кнопка для удаления. Администратор может нажать на эту кнопку, чтобы удалить статью.

5.29 Удаление комментариев пользователей

Администраторы имеют возможность удалять любые комментарии на платформе, независимо от того, кто является автором комментария (рисунок 5.31).

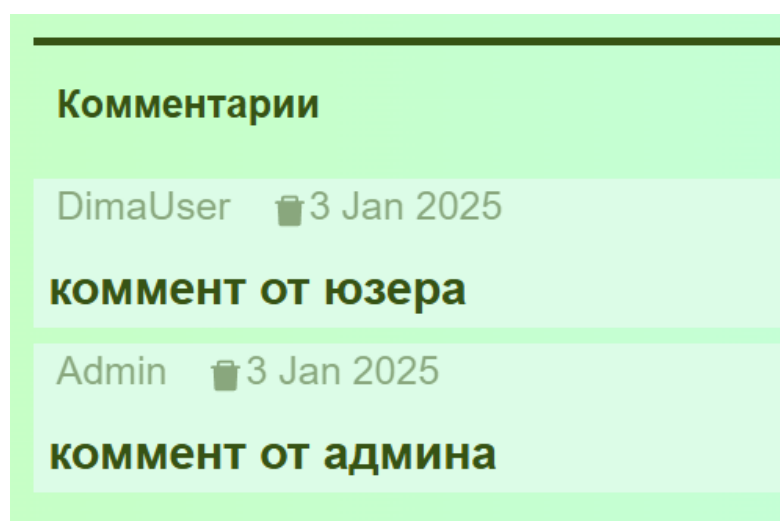


Рисунок 5.31 – Удаление всех комментариев администратором

На странице статьи, где отображаются комментарии, если пользователь имеет роль администратора, рядом с каждым комментарием будет отображаться кнопка

для удаления. Администратор может нажать на эту кнопку, чтобы удалить комментарий.

5.30 Выводы по разделу

1. Разработано руководство, описывающее действия пользователей системы: гостя, клиента и администратора, с учетом уникальных функций каждой роли, описанных в диаграмме вариантов использования.

2. У всех пользователей есть возможность просматривать статьи. Гости могут зарегистрироваться и авторизоваться. Пользователи могут добавлять и удалять статьи, комментировать статьи, а также просматривать пункты приема вторсырья, скидки и обменивать баллы на скидки в сервисах. Администратор имеет все возможности пользователя, а также может создавать, редактировать и удалять статьи, комментарии, пункты приема вторсырья, скидки, ключи для доступа к пунктам приема, а также изменять время работы пунктов приема, добавлять и изменять проверки веса сданного вторсырья, и удалять пользователей из системы.

Заключение

В ходе выполнения проекта было реализовано веб-приложение, которое соответствует всем заявленным целям и требованиям.

1. В системе предусмотрено три ключевые роли – пользователь, администратор и гость.

2. Реализовано 29 функций, включая регистрацию, авторизацию, просмотр статей, просмотр пунктов приема вторсырья, обмен баллов на скидки, создание, изменение и удаление статей, оценку статей, а также управление системой со стороны администратора.

3. В базе данных для хранения было создано 12 таблиц: пользователи, статьи, комментарии, лайки, пункты приема, виды вторсырья, секретные ключи пункты приема, скидки, промокоды, таблица для статистики и таблица для связи «многие-ко-многим» между таблицами с пунктами приема и видами вторсырья.

4. Архитектура приложения построена на принципах клиент-серверного взаимодействия. Серверная часть разработана с использованием платформа Node.js, а клиентская – на React.js.

5. В проекте написано 9500 строк кода, включая серверную и клиентскую части.

6. Разработаны и проведены 29 успешных функциональных тестов, а покрытие тестами составило 100%

В соответствии с полученным результатом работы можно сделать вывод, что цель достигнута, а требования технического задания полностью выполнены.

Список используемых источников

1. RFC 2616 HTTP/1.1 [Электронный ресурс]. – Режим доступа: <https://www.rfc-editor.org/rfc/rfc2616> – Дата доступа: 01.10.2024.
2. Что значит раздельный сбор мусора? [Электронный ресурс] – Режим доступа: центр ЖКХ – Дата обращения: 01.11.2024. Node.js [Электронный ресурс]. – Режим доступа: <https://nodejs.org/en/> – Дата доступа: 03.12.2024.
3. Сайт по раздельному сбору вторсырья Зеленая карта [Электронный ресурс]. – Режим доступа: <http://greenmap.by/> – Дата доступа: 05.10.2024.
4. Сайт по раздельному сбору вторсырья Rsbor [Электронный ресурс]. – Режим доступа: <https://rsbor.ru/> – Дата доступа: 05.10.2024.
5. Сайт по раздельному сбору вторсырья Target99 [Электронный ресурс]. – Режим доступа: <https://target99.by/> – Дата доступа: 09.10.2024.
6. MySQL [Электронный ресурс]. – Режим доступа: <https://dev.mysql.com/doc/> – Дата доступа: 21.10.2024.
7. Sequelize Documentation. [Электронный ресурс] – Режим доступа: <https://sequelize.org/docs/v6/>. – Дата обращения: 02.11.2024.
8. Object-relational mapping. [Электронный ресурс] – Режим доступа: https://en.wikipedia.org/wiki/Object-relational_mapping. – Дата обращения: 26.11.2024.
9. Nginx [Электронный ресурс]. – Режим доступа: <https://nginx.org/ru/> – Дата доступа: 23.10.2024.
10. Cloudinary [Электронный ресурс]. – Режим доступа: <https://cloudinary.com/> – Дата доступа: 26.10.2024.
11. SMTP Yandex [Электронный ресурс]. – Режим доступа: <https://yandex.ru/support/yandex-360/customers/mail/ru/mail-clients/others> – Дата доступа: 03.11.2024.
12. RFC 9293 TCP [Электронный ресурс]. – Режим доступа: <https://www.rfc-editor.org/info/rfc9293> – Дата доступа: 30.10.2024.
13. SMTP [Электронный ресурс]. – Режим доступа: <https://www.ietf.org/rfc/rfc5321.txt> – Дата доступа: 03.11.2024.
14. RFC 2818 HTTP Over TLS [Электронный ресурс]. – Режим доступа: <https://www.rfc-editor.org/rfc/rfc2818> – Дата доступа: 26.10.2024.
15. Docker Compose [Электронный ресурс]. – Режим доступа: <https://docs.docker.com/compose/> – Дата доступа: 31.10.2024.
16. Express.js [Электронный ресурс]. – Режим доступа: <https://expressjs.com/ru/> – Дата доступа: 01.11.2024.
17. bcrypt [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/bcrypt> – Дата доступа: 03.11.2024.
18. joi [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/joi> – Дата доступа: 03.11.2024.
19. jsonwebtoken [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/jsonwebtoken> – Дата доступа: 03.11.2024.
20. mysql2 [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/mysql2> – Дата доступа: 03.11.2024.

21. nodemailer [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/nodemailer> – Дата доступа: 03.11.2024.
22. nodemon [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/nodemon> – Дата доступа: 03.11.2024.
23. swagger-jsdoc [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/swagger-jsdoc> – Дата доступа: 03.11.2024.
24. swagger-ui-express [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/swagger-ui-express> – Дата доступа: 03.11.2024.
25. body-parser [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/body-parser> – Дата доступа: 03.11.2024.
26. chalk [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/chalk> – Дата доступа: 03.11.2024.
27. cookie-parser [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/cookie-parser> – Дата доступа: 03.11.2024.
28. cors [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/cors> – Дата доступа: 03.11.2024.
29. express-validator [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/express-validator> – Дата доступа: 03.11.2024.
30. fs [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/fs> – Дата доступа: 03.11.2024.
31. multer [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/multer> – Дата доступа: 03.11.2024.
32. uuid [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/uuid> – Дата доступа: 03.11.2024.
33. axios [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/axios> – Дата доступа: 03.11.2024.
34. react-redux [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/react-redux> – Дата доступа: 03.11.2024.
35. react-toastify [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/react-toastify> – Дата доступа: 03.11.2024.
36. moment [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/moment> – Дата доступа: 03.11.2024.
37. @mui/material [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@mui/material> – Дата доступа: 03.11.2024.
38. @tensorflow/tfjs [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@tensorflow/tfjs> – Дата доступа: 03.11.2024.
39. @reduxjs/toolkit [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@reduxjs/toolkit> – Дата доступа: 03.11.2024.
40. @testing-library/react [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@testing-library/react> – Дата доступа: 03.11.2024.
41. tailwindcss [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/tailwindcss> – Дата доступа: 03.11.2024.
42. @emotion/react [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@emotion/react> – Дата доступа: 03.11.2024.

43. @emotion/styled [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/@emotion/styled> – Дата доступа: 03.11.2024.

44. React [Электронный ресурс]. – Режим доступа: <https://react.dev/> – Дата доступа: 23.11.2024.

45. react-router-dom [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/react-router-dom> – Дата доступа: 03.11.2024.

46. react-scripts [Электронный ресурс]. – Режим доступа: <https://www.npmjs.com/package/react-scripts> – Дата доступа: 03.11.2024.

ПРИЛОЖЕНИЕ А

Скрипт создания базы данных

```

show databases;
create database ecosort;
drop database ecosort;

select * from ecosort.users;           -- пользователи
select * from ecosort.articles;       -- статьи
select * from ecosort.ratings;        -- комментарии
select * from ecosort.likes;         -- лайки
select * from ecosort.s_keys;        -- ключи от пунктов приема
select * from ecosort.points;        -- пункты приема
select * from ecosort.receptions;    -- переписано
select * from ecosort.points_marks;  -- переписано
select * from ecosort.marks;         -- виды вторсырья
select * from ecosort.check_weight;  -- подтверждение сдачи
select * from ecosort.points_marks;  -- используется для связи
«многие-ко-многим» таблиц points и marks
select * from ecosort.discounts;     -- скидки
select * from ecosort.promo_codes;   -- промокоды

UPDATE ecosort.users SET points = 1500 WHERE id = 2;

-- Пользователь
CREATE TABLE IF NOT EXISTS ecosort.users (
    id int auto_increment,
    username          varchar(20)          not null,
    email             varchar(100)         not null,
    password_hash     varchar(200)        not null,
    points            int default 0        null,
    role              varchar(5) default 'user' not null,
    is_activated      tinyint(1) default 0 null,
    activation_link   varchar(150)         null,
    constraint users_ck CHECK (role IN ('admin', 'user')),
    constraint email_un unique (email),
    constraint password_hash_un unique (password_hash),
    constraint users_pk primary key (id));

-- Статьи --
CREATE TABLE IF NOT EXISTS ecosort.articles (
    id int auto_increment,
    title            varchar(100) not null,
    text            text          not null,
    date_of_pub     date          not null,
    image_url       varchar(150) not null,
    author          int           not null,
    likes           int default 0 null,
    constraint articles_pk primary key (id),
    constraint title_un unique (title),
    constraint articles_fk_users foreign key (author) references
ecosort.users(id) on delete cascade);

-- Комментарии --

```

```

CREATE TABLE IF NOT EXISTS ecosort.ratings (
    id            int auto_increment,
    article_id    int            not null,
    commentator   int            not null,
    comment       varchar(500) not null,
    date_of_add   DATETIME DEFAULT CURRENT_TIMESTAMP,
    constraint ratings_pk primary key (id),
    constraint ratings_fk_users foreign key (commentator) references
ecosort.users(id) on delete cascade,
    constraint ratings_fk_articles foreign key (article_id)
references ecosort.articles(id) on delete cascade);
-- Лайки --
CREATE TABLE IF NOT EXISTS ecosort.likes(
    id            INT AUTO_INCREMENT,
    user_id       INT NOT NULL,
    article_id    INT NOT NULL,
    date_of_add   DATETIME DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT likes_pk PRIMARY KEY (id),
    CONSTRAINT likes_fk_users FOREIGN KEY (user_id) REFERENCES
ecosort.users(id) ON DELETE CASCADE,
    CONSTRAINT likes_fk_articles FOREIGN KEY (article_id) REFERENCES
ecosort.articles(id) ON DELETE CASCADE
);
-- Секретный ключ для ПП--
create table IF NOT EXISTS ecosort.s_keys(
    id int auto_increment,
    secret_key varchar(100) not null,
    is_used    int default 0 not null,
    constraint s_keys_ck CHECK (is_used IN (1,0)),
    constraint s_keys_pk primary key (id));
-- Пункт сдачи --
CREATE TABLE IF NOT EXISTS ecosort.points(
    id            int auto_increment,
    address       varchar(100) not null,
    time_of_work  varchar(100) not null,
    key_id        int            not null,
    admin_id      int            not null,
    link_to_map   text           not null,
    point_name    varchar(100) not null,
    constraint address_un unique (address),
    constraint key_id_un unique (key_id),
    constraint point_name_un unique (point_name),
    constraint points_pk primary key (id),
    constraint points_fk_users foreign key (admin_id) references
ecosort.users(id) on delete cascade,
    constraint points_fk_s_keys foreign key (key_id) references
ecosort.s_keys(id) on delete cascade);
-- Отходы --
CREATE TABLE IF NOT EXISTS ecosort.marks(
    id int auto_increment,
    rubbish varchar(50) not null,
    points_per_kg int not null,
    new_from_kg float not null,

```

```

        image_link varchar(255) null,
        constraint marks_pk primary key (id));
-- Точки сбора с отходами --
CREATE TABLE IF NOT EXISTS ecosort.points_marks(
    id          int auto_increment,
    points_id int not null,
    marks_id   int not null,
    constraint points_marks_pk primary key (id),
    constraint points_marks_fk_points foreign key (points_id)
references ecosort.points(id) on delete cascade,
    constraint points_marks_fk_marks foreign key (marks_id)
references ecosort.marks(id) on delete cascade);
-- Проверка веса --
CREATE TABLE IF NOT EXISTS ecosort.check_weight(
    id int auto_increment,
    rubbish_id int not null,
    weight int not null,
    key_of_weight varchar(100) not null,
    is_used INT DEFAULT 0 NOT NULL,
    original_key VARCHAR(100) NOT NULL,
    CONSTRAINT check_weight_is_used_ck CHECK (is_used IN (0, 1)),
    constraint key_of_weight_un unique (key_of_weight),
    constraint check_weight_pk primary key (id),
    constraint check_weight_fk_marks foreign key (rubbish_id)
references ecosort.marks(id) on delete cascade);
-- Сдача --
CREATE TABLE IF NOT EXISTS ecosort.receptions(
    id          int auto_increment,
    user_id     int not null,
    weight      float not null,
    accrued     int null,
    new_kg      int null,
    type_waste  int not null,
    station_key int not null,
    weight_key  int not null,
    constraint receptions_pk primary key (id),
    constraint receptions_fk_users foreign key (user_id) references
ecosort.users(id) on delete cascade,
    constraint receptions_fk_s_keys foreign key (station_key)
references ecosort.s_keys(id) on delete cascade,
    constraint receptions_fk_marks foreign key (type_waste)
references ecosort.marks(id) on delete cascade,
    constraint receptions_fk_check_weight foreign key (weight_key)
references ecosort.check_weight(id) on delete cascade);
-- Скидки --
CREATE TABLE IF NOT EXISTS ecosort.discounts(
    id int auto_increment,
    discount varchar(50) not null,
    promo_code varchar(50) not null,
    count_for_dnt int not null,
    constraint discount_un unique (discount),
    constraint promo_code_un unique (promo_code),
    constraint discounts_pk primary key (id));

```

```
-- Промокоды --
CREATE TABLE IF NOT EXISTS ecosort.promo_codes(
    id int auto_increment,
    promo_code varchar(50) not null,
    user_id int not null,
    discount_id int not null,
    date_of_add DATE not null DEFAULT '2023-05-11',
    constraint discounts_pk primary key (id),
    constraint user_discount_ids_un unique (user_id, discount_id),
    constraint promo_codes_fk_users foreign key (user_id) references
ecosort.users (id) on delete cascade,
    constraint promo_codes_fk_discounts foreign key (discount_id)
references ecosort.discounts (id) on delete cascade
);
```


ПРИЛОЖЕНИЕ Б

Функция регистрации

```

RegisterUser: async (req, res) => {
  try {
    const i_password = req.body.password;
    const salt = '$2b$10$qNuSSupDD53DkQfO8wqpf.';
    const o_password = await bcrypt.hash(i_password, salt);

    const v_check_user = await db.models.Users.findOne({
      where: { username: req.body.username }
    })

    const v_check_email = await db.models.Users.findOne({
      where: { email: req.body.email }
    })

    if (v_check_user == null) {
      if (v_check_email == null) {

        const v_activation_link = uuid.v4();

        const candidate = await db.models.Users.create({
          username: req.body.username,
          email: req.body.email,
          password_hash: o_password,
          role: 'user',
          is_activated: false,
          activation_link: v_activation_link
        })

        const send_mail = req.body.email;
        const send_link =
'http://localhost:8082/activate/' + v_activation_link;

        const transporter = nodemailer.createTransport({
          host: "smtp.yandex.ru",
          port: 465,
          secure: true,
          auth: {
            user: 'dimatruba2004@yandex.ru',
            pass: 'akllfknkhqfbhtl'
          }
        });

        const mailOptions = {
          from: 'dimatruba2004@yandex.ru',
          to: send_mail,
          subject: 'Подтверждение регистрации на сайте
EcoSort',
          html: `

```

```

        <div style="font-family: Arial, sans-
        serif; line-height: 1.6; color: #333;">
            <h2 style="color:
            #4CAF50;">Здравствуйте, ${req.body.username}!</h2>
            <p>
                Благодарим за регистрацию на
                платформе <strong>EcoSort</strong> – месте, где мы вместе заботимся
                о нашем будущем через переработку отходов.
            </p>
            <p>
                Для завершения регистрации и
                активации вашего аккаунта, пожалуйста, перейдите по ссылке ниже:
            </p>
            <a
                href="${send_link}"
                style="display: inline-block;
margin: 10px 0; padding: 10px 20px; color: #fff; background-color:
            #4CAF50; text-decoration: none; border-radius: 5px;"
            >
                Активировать аккаунт
            </a>
            <p>
                Или вы можете скопировать ссылку
                в браузер:
            </p>
            <p style="font-size: 14px; color:
            #555;">${send_link}</p>
            <hr style="border: none; border-top:
            1px solid #ddd;">
            <p style="font-size: 12px; color:
            #777;">
                Если вы не регистрировались на
                сайте EcoSort, проигнорируйте это сообщение.
            </p>
            <p>
                С уважением,<br>Команда EcoSort
            </p>
        </div>
    }

    let info = await
transporter.sendMail(mailOptions)

    // const accessToken = jwt.sign({ id:
candidate.null, username: candidate.username, role: candidate.role
}, accessKey, { expiresIn: 30 * 60 })
    // const refreshToken = jwt.sign({ id:
candidate.null, username: candidate.username, role: candidate.role
}, refreshKey, { expiresIn: 24 * 60 * 60 })
    //
    //

```

```

        // res.cookie('accessToken', accessToken, {
        //     httpOnly: true,
        //     sameSite: 'strict'
        // })
        // res.cookie('refreshToken', refreshToken, {
        //     httpOnly: true,
        //     sameSite: 'strict'
        // })
        res.json({
            message: 'Мы выслали вам письмо на указанную
почту для ее подтверждения',
            // accessToken,
            // user: {
            //     id: candidate.null,
            //     username: candidate.username,
            //     role: candidate.role
            // }
        });
    }
    else {
        res.json({
            message: 'Почта занята, введите другую'
        });
    }
}
else {
    res.json({
        message: 'Имя пользователя занято, введите
другое'
    });
}
}
catch (err) {
    console.log(err);
    res.json({
        message: 'Не удалось зарегистрироваться'
    });
}
},

```

Функция авторизации

```

LoginUser: async (req, res) => {
    try {
        const i_user = await db.models.Users.findOne({
            where: {
                [Op.and]: [{ email: req.body.email }, {
is_activated: 1 }],
            }
        });
        logger.info(`User tried to login with email:
${req.body.email}`);
    }
}

```

```

        if (i_user == null) {
            logger.warning('Login failed: Invalid email or
account not activated');
            res.json({ message: 'Неверная почта или пароль' });
        } else {
            const isValidPass = await
bcrypt.compare(req.body.password, i_user.password_hash);
            if (!isValidPass) {
                logger.warning('Login failed: Incorrect
password');
                res.json({ message: 'Неверная почта или пароль'
});
            } else {
                const accessToken = jwt.sign({ id: i_user.id,
username: i_user.username, role: i_user.role }, accessKey, {
expiresIn: 30 * 60 });
                const refreshToken = jwt.sign({ id: i_user.id,
username: i_user.username, role: i_user.role }, refreshKey, {
expiresIn: 24 * 60 * 60 });

                // ВЫВОД ТОКЕНОВ В КОНСОЛЬ ДЛЯ ОТЛАДКИ
                console.log('Access Token:', accessToken);
                console.log('Refresh Token:', refreshToken);

                res.cookie('accessToken', accessToken, {
                    httpOnly: true,
                    sameSite: 'strict'
                });
                res.cookie('refreshToken', refreshToken, {
                    httpOnly: true,
                    sameSite: 'strict'
                });
                res.json({
                    message: 'Авторизация прошла успешно',
                    accessToken,
                    user: {
                        id: i_user.id,
                        username: i_user.username,
                        role: i_user.role,
                        points: i_user.points
                    }
                });
            }
        }
    } catch (err) {
        console.log(err);
        res.json({
            message: 'Не удалось авторизоваться'
        });
    }
},

```

Функция добавления статьи

```

addArticles: async (req, res) => {
  try {
    const v_check_title = await db.models.Articles.findOne({
      where: { title: req.body.title },
    })

    const v_b_image_url = req.body.image_url;
    if (v_check_title == null) {
      logger.info(`Attempting to add article with title:
${req.body.title}`);

      if (v_b_image_url != undefined) {
        const article = await
db.models.Articles.create({
          author: req.userId,
          title: req.body.title,
          text: req.body.text,
          image_url: req.body.image_url,
          date_of_pub: Date.now(),
        })
        logger.success(`Article with title
"${req.body.title}" added with image.`);
        res.json({
          article,
          message: 'Статья добавлена с картинкой'
        });
      }
      else {
        const article = await
db.models.Articles.create({
          author: req.userId,
          title: req.body.title,
          text: req.body.text,
          image_url: '',
          date_of_pub: Date.now(),
        })
        logger.success(`Article with title
"${req.body.title}" added without image.`);
        res.json({
          article,
          message: 'Статья добавлена без картинки'
        });
      }
    }
    else {
      logger.warning(`Article with title
"${req.body.title}" already exists.`);
      res.json({
        message: 'Статья с таким названием уже
существует'
      });
    }
  }
}

```

```

    }
  } catch (err) {
    logger.error(`Error adding article: ${err.message}`);
    console.log(err);
    res.json({
      message: 'Не удалось добавить статью'
    });
  }
},

```

Функция изменения статьи

```

updateArticles: async (req, res) => {
  try {
    const v_check_u_title = await
db.models.Articles.findOne({
      where: { title: req.body.title },
    })
    const v_u_image_url = req.body.image_url
    if (v_u_image_url == undefined) {
      const article = await db.models.Articles.update({
        title: req.body.title,
        text: req.body.text,
        date_of_pub: Date.now(),
        image_url: req.body.image_url,
      }, {
        where: { id: req.params.id }
      })
      logger.success(`Article with ID ${req.params.id}
updated successfully.`);
      res.json({
        article,
        message: 'Статья изменена'
      });
    }
    else {
      const article = await db.models.Articles.update({
        title: req.body.title,
        text: req.body.text,
        date_of_pub: Date.now(),
        image_url: req.body.image_url
      }, {
        where: { id: req.params.id }
      })
      logger.success(`Article with ID ${req.params.id}
updated with image.`);
      res.json({
        article,
        message: 'Статья изменена'
      });
    }
  } catch (err) {

```

```

        logger.error(`Error updating article with ID
${req.params.id}: ${err.message}`);
        res.json({
            message: 'Не удалось изменить статью'
        });
    }
},

```

Функция отметки статьи

```

like: async (req, res) => {
    try {
        const i_user = req.userId
        const i_article = req.params.id
        const v_likes = await db.models.Likes.findOne({
            where: {
                [Op.and]: [{ user_id: i_user }, { article_id:
i_article }],
            }
        })
        if (!v_likes) {
            logger.info(`User ${i_user} liked article
${i_article}`);
            const likes = await db.models.Likes.create({
                user_id: i_user,
                article_id: i_article,
            })
            const count_of_likes = await
db.models.Likes.findAndCountAll({
                attributes: ['article_id'],
                where: { article_id: i_article }
            })
            const update_like = await
db.models.Articles.update({
                likes: count_of_likes.count
            }, {
                where: { id: req.params.id }
            })
            db.models.Articles.findAll({
                attributes: ["id", "title", "text",
"date_of_pub", "image_url", "likes"],
                include: [{
                    model: db.models.Users,
                    required: true,
                    attributes: ["id", "username"]
                }]
            }).then(expense => {
                logger.success(`Article ${i_article} likes
updated. Total likes: ${count_of_likes.count}`);
                res.json({
                    message: "Вы поставили лайк",
                    expense
                });
            });
        }
    }
}

```

```

        })
      }
      else {
        logger.info(`User ${i_user} removed like from
article ${i_article}`);
        const likes = await db.models.Likes.destroy({
          where: {
            [Op.and]: [{ user_id: i_user }, {
article_id: i_article }],
          }
        })
        const count_of_likes = await
db.models.Likes.findAndCountAll({
          attributes: ['article_id'],
          where: { article_id: i_article }
        })
        const update_like = await
db.models.Articles.update({
          likes: count_of_likes.count
        }, {
          where: { id: req.params.id }
        })
        db.models.Articles.findAll({
          attributes: ["id", "title", "text",
"date_of_pub", "image_url", "likes"],
          include: [{
            model: db.models.Users,
            required: true,
            attributes: ["id", "username"]
          }]
        }).then(expense => {
          logger.success(`Article ${i_article} likes
updated after unliking. Total likes: ${count_of_likes.count}`);
          res.json({
            message: "Вы убрали лайк",
            expense
          });
        })
      }
    } catch (error) {
      logger.error(`Error processing like action for article
${req.params.id}: ${error.message}`);
      res.json({
        message: 'Не удалось найти статью',
      });
    }
  },
},

```

Функция отметки статьи

```

Receptions: async (req, res) => {
  try {
    // Считываем ключ станции

```



```

const i_station_key = req.body.station_key;
const salt = '$2b$10$qNuSSupDD53DkQfO8wqpf.';
const o_station_key = await bcrypt.hash(i_station_key,
salt);

// Находим запись по ключу станции
const v_find_key = await db.models.Keys.findOne({
  where: { secret_key: o_station_key }
});

if (!v_find_key || v_find_key.is_used !== 1) {
  return res.status(400).json({
    message: 'Ключ станции недействителен'
  });
}

// Находим пункт приема, связан ли с ключом станции
const v_find_key_point = await
db.models.Points.findOne({
  where: { key_id: v_find_key.id }
});

if (!v_find_key_point) {
  return res.status(400).json({
    message: 'Станция с таким ключом не найдена'
  });
}

// Находим виды отходов, которые принимает этот пункт
сдачи
const pointWasteTypes = await
db.models.Points_marks.findAll({
  where: { points_id: v_find_key_point.id },
  attributes: ['marks_id'],
});

const pointWasteIds = pointWasteTypes.map(item =>
item.marks_id);

// Считываем ключ для проверки веса
const i_key_of_weight = req.body.key_of_weight;
const o_key_of_weight = await
bcrypt.hash(i_key_of_weight, salt);

// Находим запись по ключу веса
const v_check_key_w = await
db.models.Check_weights.findOne({
  attributes: ["id", "rubbish_id", "weight",
"is_used"],
  where: { key_of_weight: o_key_of_weight }
});

if (!v_check_key_w) {

```

```

        return res.status(400).json({
            message: 'Ключ веса недействителен'
        });
    }

    // Проверка, был ли уже использован ключ веса
    if (v_check_key_w.is_used === 1) {
        return res.status(400).json({
            message: 'Ключ веса уже использован'
        });
    }

    // Проверяем, соответствует ли вид отхода пункту сдачи
    if (!pointWasteIds.includes(v_check_key_w.rubbish_id)) {
        return res.status(400).json({
            message: 'Этот вид отходов не принимается на
данный пункт сдачи'
        });
    }

    // Получаем данные о вторсырье
    const v_rubbish = await db.models.Marks.findOne({
        attributes: ["id", "rubbish", "points_per_kg",
"new_from_kg"],
        where: { id: v_check_key_w.rubbish_id }
    });

    if (!v_rubbish) {
        return res.status(400).json({
            message: 'Вид вторсырья не найден'
        });
    }

    // Рассчитываем начисление баллов
    const v_weight = v_check_key_w.weight;
    const o_new_points = v_weight * v_rubbish.points_per_kg;
    const o_new_kg = v_weight * v_rubbish.new_from_kg;

    // Находим текущие баллы пользователя
    const i_user_points = await db.models.Users.findOne({
        attributes: ["points"],
        where: { id: req.userId }
    });

    if (!i_user_points) {
        return res.status(400).json({
            message: 'Пользователь не найден'
        });
    }

    // Обновляем баллы пользователя
    const o_new_points_user = o_new_points +
i_user_points.points;

```

```

        await db.models.Users.update({
            points: o_new_points_user
        }, {
            where: { id: req.userId }
        });

        // ОТМЕТИМ КЛЮЧ ВЕСА КАК ИСПОЛЬЗОВАННЫЙ
        await db.models.Check_weights.update({
            is_used: 1
        }, {
            where: { id: v_check_key_w.id }
        });

        // Создаем запись в receptions
        await db.models.Receptions.create({
            user_id: req.userId,
            accrued: o_new_points,
            new_kg: o_new_kg,
            weight: v_weight,
            type_waste: v_rubbish.id,
            station_key: v_find_key.id,
            weight_key: v_check_key_w.id
        });

        // Отправляем УСПЕШНЫЙ ОТВЕТ
        res.json({
            o_new_kg,
            o_new_points,
            o_new_points_user,
            message: 'Ваши баллы успешно начислены',
        });
    } catch (error) {
        console.log(error);
        res.status(500).json({
            message: 'Не удалось начислить баллы. Попробуйте ещё
раз.'
        });
    }
}

```

Функция обмена баллов

```

usedMyDiscounts: async (req, res) => {
    try {
        const v_user = await db.models.Users.findOne({
            attributes: ["points"],
            where: {id: req.userId}
        })

        // console.log(v_user.points);

        const i_discounts = await db.models.Discounts.findOne({

```

```

        attributes: ["id", "discount", "count_for_dnt",
"promo_code"],
        where: {id: req.params.id}
    })

    // console.log(i_discounts.count_for_dnt);

    const o_new_points = v_user.points -
i_discounts.count_for_dnt

    // console.log(o_new_points);

    await db.models.Users.update({
        points: o_new_points,
    }, {where: {id: req.userId}})

    const v_check_promo_codes =
await db.models.Promo_codes.findOne({
        where: {
            [Op.and]: [{ user_id: req.userId }, {
discount_id: i_discounts.id }],
        }
    })

    console.log(v_check_promo_codes)
    if (v_check_promo_codes!=null) {
        await db.models.Promo_codes.update({
            promo_code: i_discounts.promo_code,
            date_of_add: Date.now(),
            {where: {
                [Op.and]: [{ user_id: req.userId }, {
discount_id: i_discounts.id }],
            }
        })
    }
    else{
        await db.models.Promo_codes.create({
            promo_code: i_discounts.promo_code,
            user_id: req.userId,
            discount_id: i_discounts.id,
            date_of_add: Date.now(),
        })
    }

    function generateString(length) {
        let result = '';
        const characters =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';

        for (let i = 0; i < length; i++) {
            const randomIndex = Math.floor(Math.random() *
characters.length);
            result += characters.charAt(randomIndex);

```

```

        }
        return result;
    }

    const generatedString = generateString(8);
    // console.log(generatedString);

    await db.models.Discounts.update({
        promo_code: generatedString,
        {where: {id: req.params.id}
    })

    res.json({
        o_new_points,
        message: 'Скидка использована'
    });
}
catch (error) {
    console.log(error);
}
},

```

Функция изменения ключа

```

editPointsKey: async (req, res) => {
    try {
        const v_point_key_id = await db.models.Points.findOne({
            attributes: ["key_id"],
            where: { id: req.params.id }
        })

        const v_last_id_k = await db.models.Keys.findOne({
            order: [['id', 'DESC']],
        });
        const o_new_id_k = v_last_id_k.id + 1;

        // console.log(v_point_key_id.key_id);

        const v_key = req.body.secret_key;
        const i_sk_salt = '$2b$10$qNuSSupDD53DkQfO8wqpf.';
        const o_secret_key = await bcrypt.hash(v_key,
i_sk_salt);

        const v_old_key = await db.models.Keys.findOne({
            attributes: ["secret_key"],
            where: { secret_key: o_secret_key }
        });

        // console.log(v_old_key.id);

        if (v_old_key == null) {
            const new_record_keys = await
db.models.Keys.create({
                id: o_new_id_k,

```

```

        secret_key: o_secret_key,
        is_used: 1,
    })

    // console.log(new_record_keys.toJSON())
    // console.log(new_record_keys.null);

    await db.models.Points.update({
        key_id: new_record_keys.null

    }, {
        where: { id: req.params.id }
    })

    res.json({
        message: 'Секретный ключ обновлен'
    });
}
else {
    res.json({
        message: 'Такой код уже используется исправьте
его'

    })
}
}
catch (err) {
    console.log(err);
    res.json({
        message: 'Не удалось обновить пункт сдачи отходов'
    });
}
},

```

Функция добавления пункта приема

```

addPoints: async (req, res) => {
    try {
        function splitString(stringToSplit, separator) {
            return stringToSplit.split(separator);
        }

        const comma = ',';
        const i_rubbish = req.body.rubbish;
        const arrayOfStrings = splitString(i_rubbish, comma);
        const o_length = arrayOfStrings.length;
        const v_check_address = await db.models.Points.findOne({
            where: { address: req.body.address }
        })

        const v_check_point_name = await
db.models.Points.findOne({
            where: { point_name: req.body.point_name }
        })
    }
}

```

```

        if (v_check_address == null) {
            if (v_check_point_name == null) {
                const v_find_used = await
db.models.Keys.findOne({
                    where: { is_used: 0 }
                })

                if (v_find_used != null) {
                    const c_point = await
db.models.Points.create({
                        address: req.body.address,
                        time_of_work: req.body.time_of_work,
                        key_id: v_find_used.id,
                        admin_id: req.userId,
                        link_to_map: req.body.link_to_map,
                        point_name: req.body.point_name,
                    })

                    await db.models.Keys.update({
                        is_used: 1,
                    }, {
                        where: { id: v_find_used.id }
                    })

                    const v_point_id = c_point.id
                    console.log(v_point_id)

                    for (let j = 0; j < o_length; j++) {
                        const i_rubbish = await
db.models.Marks.findOne({
                            where: { rubbish: arrayOfStrings[j]
                        })
                    }
                    if (i_rubbish == null) {
                        res.json({
                            message: `Такого
${arrayOfStrings[j]} вида вторсырья нет сначала добавить его во
вторсырье`
                        });
                        return;
                    }
                    else {
                        const o_rubbish_id = i_rubbish.id
                        console.log(o_rubbish_id)

                        await
db.models.Points_marks.create({
                            points_id: v_point_id,
                            marks_id: o_rubbish_id,
                        })
                    }
                }
            }
        }
    }
}

```

```

        }
        res.json({
            message: 'Пунк сдачи отходов добавлен'
        });
    } else {
        res.json({
            message: 'Пунк сдачи отходов не может
быть добавлен, нет свободных ключей'
        });
    }
}
else {
    res.json({
        message: 'Имя пункта приема уже
используется, введите новое'
    });
}
}
else {
    res.json({
        message: 'Адрес уже используется, введите новый
адрес'
    });
}
}
catch (err) {
    console.log(err);
    res.json({
        message: 'Не удалось добавить пункт сдачи отходов'
    });
}
},

```

Функция добавления проверки веса

```

addWeight: async (req, res) => {
    try {
        const i_key_of_weight = req.body.key_of_weight;
        const salt_for_key = '$2b$10$qNuSSupDD53DkQfO8wqpf.';
        const o_key_of_weight = await
bcrypt.hash(i_key_of_weight, salt_for_key);

        // Находим вид вторсырья по запросу
        const o_rubbish = await db.models.Marks.findOne({
            attributes: ["id"],
            where: { rubbish: req.body.rubbish_w }
        });

        if (o_rubbish == null) {
            // Если вид вторсырья не найден, возвращаем
сообщение
            res.json({

```



```

        message: `Такого вида вторсырья
        (${req.body.rubbish_w}) нет, сначала добавьте его в список`
    });
    return;
}

// Проверка, не существует ли уже запись с таким ключом
const v_check_key_w = await
db.models.Check_weights.findOne({
    where: { key_of_weight: o_key_of_weight }
});

if (v_check_key_w != null) {
    // Если ключ уже существует, возвращаем сообщение
    res.json({
        message: 'Этот ключ уже добавлен, введите новый'
    });
    return;
}

// Записываем новый ключ с указанным видом, весом и
оригинальным ключом
await db.models.Check_weights.create({
    rubbish_id: o_rubbish.id,
    weight: req.body.weight,
    key_of_weight: o_key_of_weight,
    original_key: i_key_of_weight, // Сохраняем
оригинальный ключ
});

// Возвращаем успешное сообщение
res.json({
    message: 'Ключ успешно добавлен'
});

} catch (error) {
    console.log(error);
    res.json({
        message: 'Не удалось добавить ключ для проверки
веса'
    });
}
},

```

Функция изменения проверки веса

```

editWeight: async (req, res) => {
    try {
        const { id, rubbish_w, weight, key_of_weight,
original_key } = req.body;
        const salt_for_key = '$2b$10$qNuSSupDD53DkQfO8wqpf.';
        const o_key_of_weight = await bcrypt.hash(key_of_weight,
salt_for_key);
    }
}

```

```

        // Находим ID вида вторсырья
        const o_rubbish = await db.models.Marks.findOne({
            attributes: ["id"],
            where: { rubbish: rubbish_w }
        });

        if (o_rubbish == null) {
            res.json({
                message: `Такого ${rubbish_w} вида вторсырья
нет, сначала добавьте его во вторсырье`
            });
            return;
        }

        const existingRecord = await
db.models.Check_weights.findOne({
            where: { id }
        });

        if (existingRecord) {
            // Обновление записи, включая original_key
            await db.models.Check_weights.update({
                rubbish_id: o_rubbish.id,
                weight,
                key_of_weight: o_key_of_weight,
                original_key: key_of_weight
            }, {
                where: { id }
            });

            res.json({
                message: 'Запись успешно обновлена'
            });
        } else {
            res.json({
                message: 'Запись не найдена'
            });
        }
    } catch (error) {
        console.log(error);
        res.json({
            message: 'Не удалось обновить запись'
        });
    }
}

```