

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

**БИЗНЕС-ПРОЕКТ. ЭЛЕКТРОННЫЙ РЫНОК
ФЕРМЕРСКИХ ПРОДУКТОВ. СЕРВЕРНАЯ**

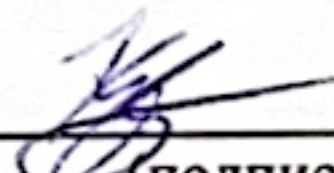
ЧАСТЬ

Студент

 03.06.2025
(подпись, дата)

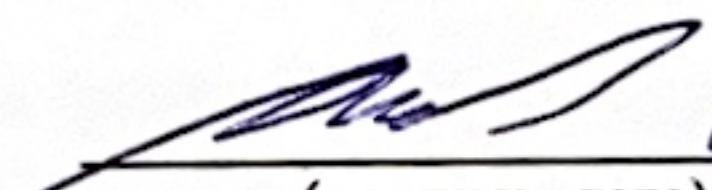
Суцёв Максим Валерьевич
(ФИО полностью)

Руководитель

 03.06.2025
(подпись, дата)

Кузнецов А.В.
(фамилия, инициалы)

Нормоконтролёр

 03.06.2025
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой


(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

6 июня 2025 г. № 13)

Секретарь ГЭК


(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2025

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

« 16 » 05 2025 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Сущёв Максим Валерьевич гр. 21ВВП1 факультета вычислительной техники
2. Тема работы Бизнес-проект. Электронный рынок фермерских продуктов. Серверная часть
Тема утверждена приказом ПГУ № 601/о от 06.05.2025 г.
3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Кузнецов А.В.

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

Разработать серверную часть web-приложения. Составить бизнес-модель и финансовый план Язык программирования — Python, фреймворк — Flask, система управления базами данных — MySQL, среда разработки PyCharm. Серверная часть должна обеспечивать надёжное взаимодействие с клиентским интерфейсом через REST API и реализовывать следующие функциональные возможности:

1. Регистрацию и аутентификацию с использованием токенов;
2. Управление каталогом товаров, отзывами и профилями пользователей;
3. Поиск товаров исходя из местоположения покупателя;
4. Хранение и обработку данных в реляционной СУБД (MySQL);
5. Интеграцию с клиентской частью через REST API.

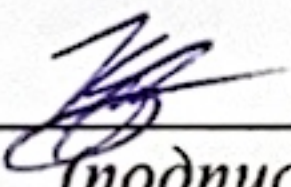
5. Перечень подлежащих разработке вопросов

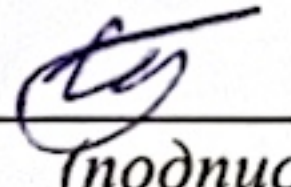
Требуется провести анализ предметной области: изучить существующие аналоги цифровых торговых платформ, проанализировать архитектурные решения и подходы к построению backend архитектуры. На основе анализа выбрать язык программирования, серверного фреймворка и базы данных. Описать архитектуру взаимодействия компонентов системы, разработать модели базы данных и реализовать ключевые REST-эндпоинты. Выгрузить программу на сервер, настроить его для взаимодействия через REST API и для дальнейшей разработки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	5%	06.05.2025 – 09.05.2025	
2	Анализ требований	10%	10.05.2025 – 13.05.2025	
3	Выбор технологий, применяемых при проектировании и разработке	10%	13.05.2025 – 15.05.2025	
4	Разработка серверной части	35%	16.05.2025 – 21.05.2025	
5	Выгрузка программы на сервер и его настройка	25%	22.05.2025 – 24.05.2025	
6	Оформление пояснительной записки	15%	25.05.2025 – 30.05.2025	

Дата выдачи задания "06" 05 2025 г.

Руководитель бакалаврской работы _____  06.05.2025 Кузнецов А.В.
(подпись, дата)

Задание к исполнению принял студент _____  06.05.2025 Сущёв М.В.
(подпись, дата)

Работу к защите допустить
Декан факультета _____  06.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 113 листов, 22 рисунка, 1 таблица, 3 источника, 3 приложения

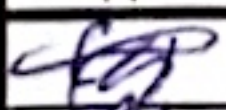
WEB-ПРИЛОЖЕНИЕ, МАРКЕТПЛЕЙС, ТЕСТИРОВАНИЕ, MYSQL, PYTHON, FLASK, FLASK-SQLALCHEMY, REST API, NGINX

Цель работы – разработать серверную часть web-приложения, которое обеспечивает прямое взаимодействие между фермерами и покупателями, упрощая доступ к свежей фермерской продукции.

Разрабатываемая система предоставляет покупателям возможность находить фермеров по регионам, просматривать ассортимент товаров с подробными описаниями и отзывами, оформлять заказы и выбирать способ доставки. Продавцы, в свою очередь, получают инструменты для управления своим ассортиментом, обработки заказов и продвижения продукции.

В итоге была разработана серверная часть web-приложения на Python с использованием MySQL, Flask и Flask-SQLAlchemy. Обмен данными между клиентской частью и серверной осуществляется через REST API. Вся информация хранится в базе данных MySQL, а развертывание и обработка запросов производится через web-сервер NGINX.

ПГУ1.090301.21.001 ПЗ

Изм.	Лист	№ докум.	Подпись	Дата	Бизнес-проект. Электронный рынок фермерских продуктов. Серверная часть Пояснительная записка	Лит.	Лист	Листов
Разраб.		Суцёв М.В.		03.06.25				
Провер.		Кузнецов А.В.		03.06.25			2	113
Н. Контр		Бычков А.С.		03.06.25		ФВТ, каф. ВТ гр. 21ВВП1		
Утверд.		Митрохин М.А.		03.06.25				

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	7
1 Бизнес-проект. Электронный рынок фермерских продуктов.....	8
1.1 Анализ предметной области	8
1.2 Анализ аналогов	10
1.2.1 Оффлайн рынок	10
1.2.2 АгроМП.....	11
1.2.3 Авито	12
1.2.4 Яндекс Маркет.....	13
1.2.5 Вывод.....	14
1.3 Постановка задачи.....	15
1.4 Требования к аппаратуре и программному обеспечению	16
1.4.1 Клиентская часть	16
1.4.2 Серверная часть	16
2 Архитектура web-приложения.....	18
3 Выбор инструментов разработки	22
3.1 Выбор языка программирования	22
3.2 Выбор базы данных.....	23
3.3 Выбор среды разработки	24
3.4 Выбор среды для взаимодействия с сервером	25
4 Бизнес-модель.....	26
4.1 Описание продукта	26
4.2 Потоки доходов	27
4.3 Каналы распространения.....	28
4.4 Взаимоотношения с клиентами	28
4.5 Ключевые ресурсы	28
4.6 Партнеры.....	28

4.7 Оценка затрат и доходов	29
4.8 Расчет срока окупаемости	30
5 Реализация приложения	32
5.1 Описание функционала	32
5.2 Описание структуры БД	35
5.2.1 Buyers.....	36
5.2.2 Sellers	37
5.2.3 SellerAddress	37
5.2.4 Categories.....	37
5.2.5 Products	37
5.2.6 Reviews	38
5.3 Описание бизнес-логики	38
5.4 Описание документации.....	41
5.5 Тестирование приложения	42
5.5.1 Регистрация.....	42
5.5.2 Авторизация.....	43
5.5.3 Получение ближайших продуктов	44
5.5.4 Получение ближайших продавцов	45
5.6 Описание деплоя приложения	45
6 Безопасность жизнедеятельности.....	47
6.1 Правила и техника безопасности при работе с ПК.....	47
6.2 Правила и техника безопасности при работе со смартфоном	48
Заключение	50
Список использованных источников	52
Приложение А	53
Приложение Б.....	58
Приложение В.....	104

Перечень сокращений и обозначений

В данной работе применяются следующие сокращения и обозначения

БД – база данных

ОС – операционная система

ПК – персональный компьютер

ПО – программное обеспечение

СУБД – система управления базами данных

API – (Application Programming Interface) – интерфейс программирования приложений, набор правил для взаимодействия программных компонентов

CPU – (Central Processing Unit) – центральный процессор, основной вычислительный компонент компьютера

CRUD – (Create, Read, Update, Delete) – основные операции управления данными в базе данных

HTTP – (HyperText Transfer Protocol) – протокол передачи гипертекста, используемый для обмена данными между web-браузером и сервером

IDE – (Integrated Development Environment) – интегрированная среда разработки

JWT – (JSON Web Token) – компактный и безопасный способ передачи информации между сторонами как объекта JSON, часто используется для авторизации

REST – (Representational State Transfer) – архитектурный стиль взаимодействия между компонентами в распределённой среде, часто используется для создания API

SEO – (Search Engine Optimization) – это комплекс мер по улучшению сайта для его ранжирования в поисковых системах

SFTP – (SSH File Transfer Protocol) – безопасный протокол передачи файлов, работающий поверх SSH

SSH – (Secure Shell) – протокол удалённого доступа к серверу через защищённое соединение

UI – (User Interface) – пользовательский интерфейс

UML – (Unified Modeling Language) – унифицированный язык моделирования

UX – (User Experience) – это совокупность всех аспектов взаимодействия пользователя с продуктом, системой или сервисом

VPS – (Virtual Private Server) – виртуальный частный сервер

					<i>ПГУ1.090301.21.001 ПЗ</i>	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В условиях активного развития цифровых технологий и электронной коммерции растёт потребность в специализированных платформах, упрощающих взаимодействие между производителями и конечными потребителями. Особенно это актуально для агропромышленного сектора, где фермеры часто сталкиваются с трудностями в организации прямых продаж. Современные web-приложения позволяют автоматизировать процессы торговли, расширить рынок сбыта и сделать продукцию доступной широкой аудитории.

Данная выпускная квалификационная работа посвящена разработке серверной части web-приложения «Электронный рынок фермерских продуктов». Целью проекта является создание надёжной backend-системы, обеспечивающей функциональность платформы для продажи фермерской продукции через интернет. Основные задачи системы включают: регистрацию и аутентификацию пользователей, управление товарами, формирование отзывов, а также реализацию API для взаимодействия с клиентской частью.

С ростом популярности локальных и экологически чистых продуктов появляется необходимость в удобных, масштабируемых решениях, которые бы объединяли продавцов и покупателей на одной цифровой платформе. Разработка такой системы способствует не только повышению цифровизации сельского хозяйства, но и развитию малого и среднего бизнеса в аграрной сфере.

Цель данной дипломной работы — создание функциональной, масштабируемой и безопасной серверной части web-приложения, реализованной с использованием технологий Python, Flask, MySQL и REST API, а также выгрузка программы на сервер, его настройка для взаимодействия с ним и дальнейшей разработки.

1 Бизнес-проект. Электронный рынок фермерских продуктов

1.1 Анализ предметной области

«Электронный рынок фермерской продукции» - электронная платформа, которая обеспечивает прямое взаимодействие между потребителями и фермерами, упрощает поиск свежих фермерских продуктов и поддерживает местных производителей.

Развитие электронной торговли фермерскими продуктами в России набирает обороты, и идея создания платформы для прямого взаимодействия между производителями и потребителями отлично вписывается в этот тренд. Согласно данным, рынок онлайн-продаж продуктов питания в России стремительно растет и уже превысил 1 трлн рублей к концу 2024 года. При этом специализированные фермерские маркетплейсы становятся все более востребованными, а объем продаж фермерской продукции может достигнуть 80 млрд рублей в 2025 году.

Вместе с развитием онлайн продажи продуктов растет и спрос на фермерскую продукцию. Например, офлайн-продажи на рынках в июле 2023 года выросли на 28% по сравнению с аналогичным периодом 2022 года, а онлайн-заказы фермерских продуктов увеличились на 33%. Кроме того, объем онлайн-продаж фермерской продукции достиг 50 млрд рублей к 2024 году [1]. В ближайшие годы спрос будет смещаться в сторону свежего мяса, птицы, яиц и молока, поскольку потребители все больше ценят натуральные и качественные продукты.

Электронный рынок является стартапом, основная цель которого – решение проблемы доступности качественных продуктов. Это позволит покупателям приобретать свежие фермерские товары напрямую у производителей, минуя посредников. К тому же мы сможем поддерживать местных фермеров, так как

					ПГУ1.090301.21.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

платформа обеспечит им новые каналы сбыта и даст возможность расширения бизнеса. Ну и конечно нельзя забывать про прозрачность и доверие, которые так ценятся в современном мире. Пользователям будут доступны рейтинги фермеров, основанные на отзывах, которые они смогут оставить.

С помощью нашей платформы покупатели смогут фильтровать запросы по регионам, категориям и производителям, тем самым облегчая поиск. Так же они смогут воспользоваться картой, на которой смогут найти ближайших производителей и быстрее купить нужный товар. Фермеры в свою очередь будут получать заказы и предлагать возможные варианты доставки (самовывоз, курьер, транспортные компании).

Таким образом, электронный рынок объединит в себе функции маркетплейса, рекламной площадки и инструмента продвижения для фермеров и сельских хозяйств. Оно поможет покупателям облегчить поиск свежих фермерских продуктов, а фермерам поможет продвигать и развивать свой дело. На рисунке 1 представлена диаграмма прецедентов, описывающая возможности пользователей.

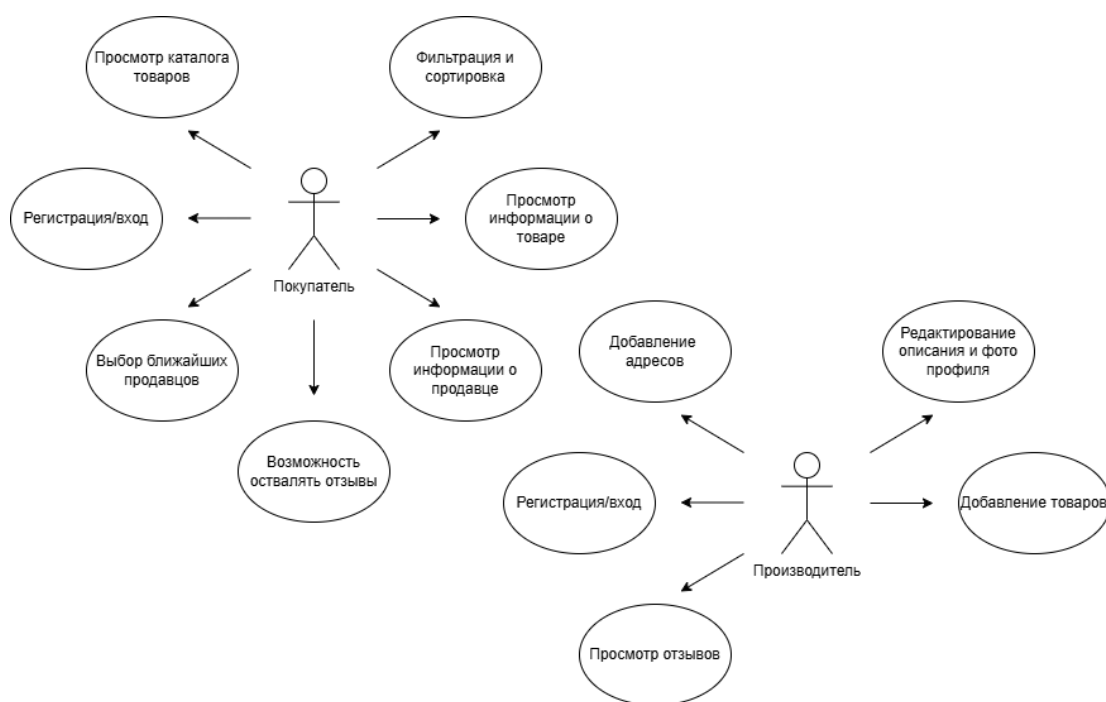


Рисунок 1 – UML-диаграмма прецедентов

1.2 Анализ аналогов

В рамках исследования были проанализированы оффлайн рынок, а также наиболее известные web-платформы и приложения, предназначенные для электронной коммерции, в частности — для продажи сельскохозяйственной продукции. Основное внимание уделялось функциональным возможностям этих решений, удобству использования, отзывам пользователей и реализованным бизнес-механизмам. Такой подход позволил выявить ключевые достоинства и недостатки каждого из рассматриваемых продуктов, что стало основой для проектирования собственной системы, более ориентированной на потребности фермеров и покупателей.

Большинство существующих платформ сосредоточены на продаже универсальных товаров, часто без учёта специфики фермерской продукции и особенностей локального аграрного рынка.

1.2.1 Оффлайн рынок

Оффлайн рынок — обычный продовольственный рынок продуктов. На рисунке 2 представлен оффлайн рынок.



Рисунок 2 – Оффлайн рынок

Достоинства:

- возможность «здесь и сейчас» убедиться в качестве товара;
- не нужно ждать доставки;
- возможность поторговаться.

Недостатки:

- можно купить только в определенном месте и в определенный период времени;
- нет возможности узнать ассортимент заранее;
- нет информации от других покупателей о качестве товара.

1.2.2 АгроМП

АгроМП – онлайн площадка для продажи и покупки агропродукции. Сервис для фермеров, владельцев магазинов любого размера, центров питания и производителей продуктов питания. На рисунке 3 представлена главная страница портала.

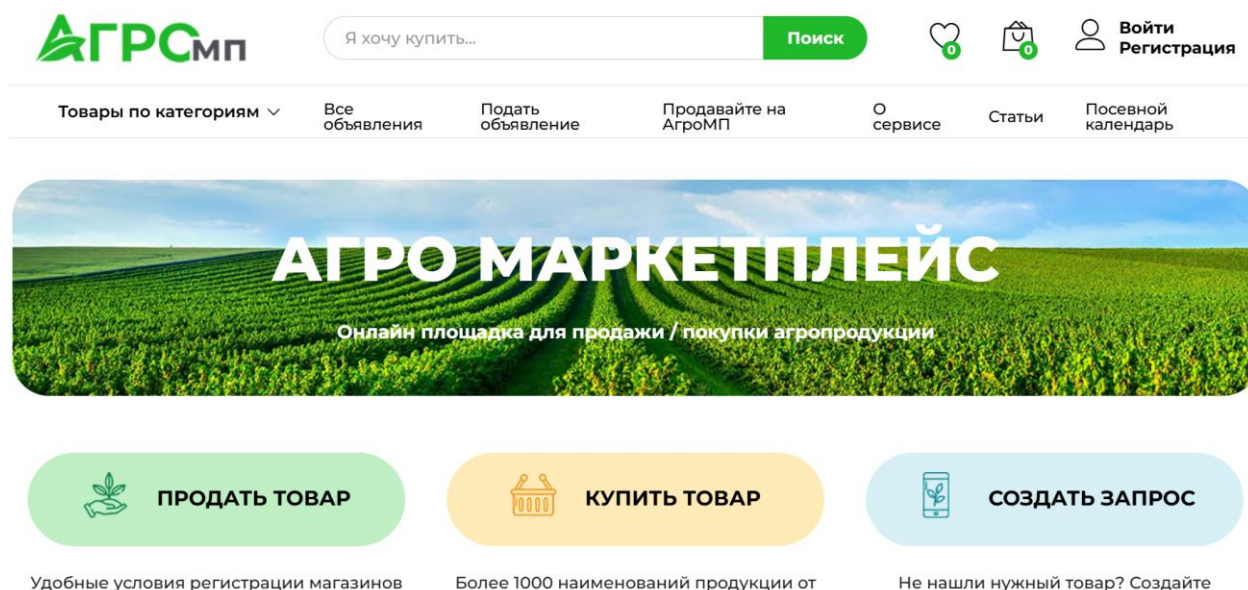


Рисунок 3 – Главная страница АгроМП

Достоинства:

- помимо продуктов продается с/х оборудование и техника, а также семена;

					ПГУ1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		11

- возможно создание запроса на товары, которых нет;
- есть дополнительные статьи по теме с/х и посевной календарь.

Недостатки:

- нет доставки товара;
- нет карты фермеров;
- платные тарифы для фермеров;
- отсутствует система отзывов и рейтинга;
- больше ориентировано на оптовые заказы;
- нет возможности приобрести скоропортящиеся товары.

1.2.3 Авито

Авито — российский интернет-сервис для размещения объявлений о товарах, недвижимости, вакансиях и резюме на рынке труда, а также услугах, занимающий первое место в мире среди сайтов объявлений. Объявления могут размещать как частные лица, так и компании. На рисунке 4 представлена главная страница портала.

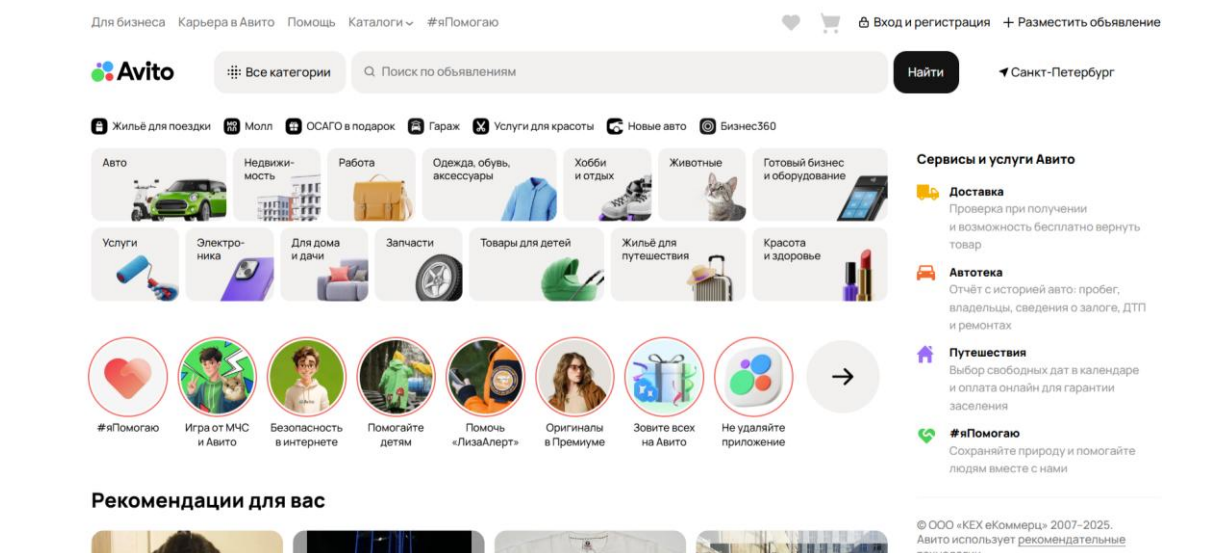


Рисунок 4 – Главная страница Авито

Достоинства:

- возможность поиска товаров в своём городе;

					ПГУ1.090301.21.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		12

- наличие чата для связи;
- рейтинг продавцов;
- наличие доставки.

Недостатки:

- нет карты фермеров;
- сайт предлагает слишком разнообразный ассортимент товаров;
- нет системы подписок.

1.2.4 Яндекс Маркет

Яндекс Маркет – сервис компании Яндекс в сфере электронной коммерции. Состоит из маркетплейса Яндекс Маркет с сетью постаматов, складов и пунктов выдачи заказов (ПВЗ), и сервиса «Маркет для бизнеса». На рисунке 5 представлена главная страница портала.

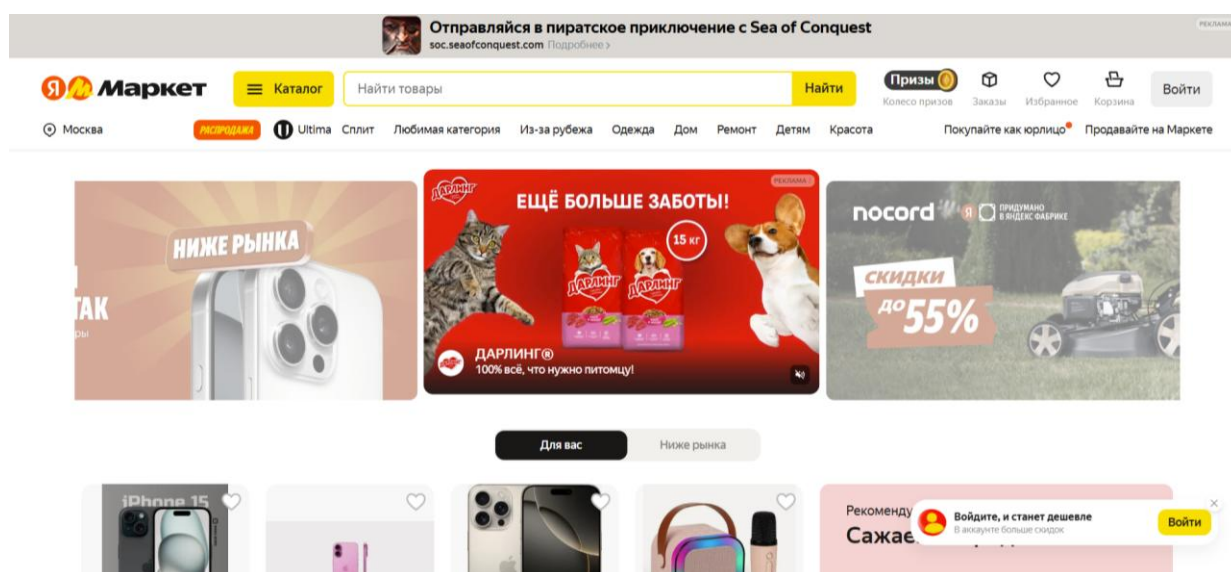


Рисунок 5 – Главная страница Яндекс Маркет

Достоинства:

- рейтинг продавцов;
- наличие доставки.

Недостатки:

- нет карты фермеров;

- нет системы подписок;
- нет возможности приобрести многие товары;
- нет возможности приобрести скоропортящиеся товары.

1.2.5 Вывод

Анализ существующих решений в области торговли аграрной продукцией показал, что ни одно из рассмотренных решений не ориентировано строго на узкую нишу — электронную торговлю исключительно фермерскими товарами и не обладает полным набором необходимых функций. В таблице 1 приведены основные критерии анализа аналогов.

Таблица 1 – Достоинства и недостатки аналогов

	Оффлайн рынок	АгроМП	Авито	Яндекс Маркет
Возможность покупки без доставки	+	-	+	-
Карта фермеров	-	-	-	-
Система отзывов и рейтингов	-	-	+	+
Возможность заказа скоропортящегося товара	+	-	+	-
Ориентация на локальный рынок	+	-	+	-
Ориентация на фермерские товары	+	+	-	-

Большинство решений являются универсальными, охватывая широкий ассортимент товаров и услуг, что снижает удобство поиска именно фермерской продукции и ослабляет доверие со стороны покупателей. Отсутствие специализированных функций (карты фермеров, отзывов о продукции) препятствует формированию устойчивых каналов взаимодействия между покупателями и производителями.

Таким образом, актуальной задачей является создание специализированной платформы, сфокусированной исключительно на нише фермерских продуктов, обеспечивающей удобное взаимодействие, высокую прозрачность и доверие со стороны пользователей.

1.3 Постановка задачи

Для решения существующих проблем, связанных с ограниченным доступом потребителей к фермерской продукции и отсутствием централизованной цифровой платформы, необходимо разработать серверную часть web-приложения «Электронный рынок фермерских продуктов». Проект должен обеспечивать удобное, безопасное и масштабируемое взаимодействие между продавцами (фермерами) и покупателями.

Разрабатываемая система должна включать функциональность для регистрации и аутентификации пользователей с разграничением ролей (покупатель, продавец), ведения каталога товаров, удобного поиска товаров, а также публикации и просмотра отзывов. Для взаимодействия с клиентской частью необходимо реализовать REST API, позволяющий выполнять все основные операции.

Данные о продуктах, покупателях, продавцах, и отзывах должны храниться в реляционной базе данных (MySQL). API-интерфейсы должны поддерживать основные методы (GET, POST, PUT, DELETE) и обеспечивать корректную валидацию и защиту данных, в том числе с использованием токенов доступа.

Особое внимание в проекте уделяется удобству взаимодействия и масштабируемости. Система должна быть спроектирована таким образом, чтобы в дальнейшем её можно было интегрировать с web-интерфейсом или мобильным приложением.

Таким образом, реализация web-приложения «Электронный рынок фермерских продуктов» позволит централизовать процессы электронной торговли фермерской продукцией, повысить доступность локальных товаров, а также улучшить пользовательский опыт за счёт быстрого доступа к информации, в том числе и к отзывам других пользователей.

1.4 Требования к аппаратуре и программному обеспечению

1.4.1 Клиентская часть

Для полноценной работы клиентской части необходимы следующие минимальные требования к устройству:

- любое устройство, поддерживающее HTTP(S)-протокол — ПК, ноутбук, планшет, смартфон;
- современный браузер с поддержкой JavaScript, CSS3 и HTML5 (например, Google Chrome, Mozilla Firefox, Microsoft Edge);
- процессор — двухъядерный CPU с частотой от 2.0 ГГц (рекомендуется — 4 ядра и выше);
- объём оперативной памяти от 2 ГБ (рекомендуется 4 ГБ и выше);
- стабильное подключение к интернету.

1.4.2 Серверная часть

Для размещения серверной части приложения, реализованной на Python с использованием Flask и СУБД MySQL, предъявляются следующие минимальные требования к серверу:

- операционная система — Linux (Ubuntu 20.04+ или аналог) или Windows Server;

- процессор — двухъядерный CPU с частотой от 2.0 ГГц (рекомендуется — 4 ядра и выше);
- объём оперативной памяти от 4 ГБ (рекомендуется 8 ГБ и выше);
- объём постоянной памяти от 10 ГБ;
- сеть — постоянное подключение к интернету с пропускной способностью от 10 Мбит/с.

					ПГУ1.090301.21.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

2 Архитектура web-приложения

Для разработки web-приложения «Электронный рынок фермерской продукции» была выбрана клиент-серверная архитектура с чётким разделением на фронтенд и бэкенд, взаимодействующих между собой через REST API. Такой подход обеспечивает масштабируемость, модульность, удобство сопровождения и даёт возможность разрабатывать клиентскую и серверную части независимо друг от друга.

На серверной стороне используется архитектурный паттерн Model-View-Controller (MVC), который структурирует код и способствует упрощённому управлению бизнес-логикой, пользовательским интерфейсом и взаимодействием с данными. На рисунке 6 представлена диаграмма компонентов web-приложения, оформленная в стиле архитектуры MVC.

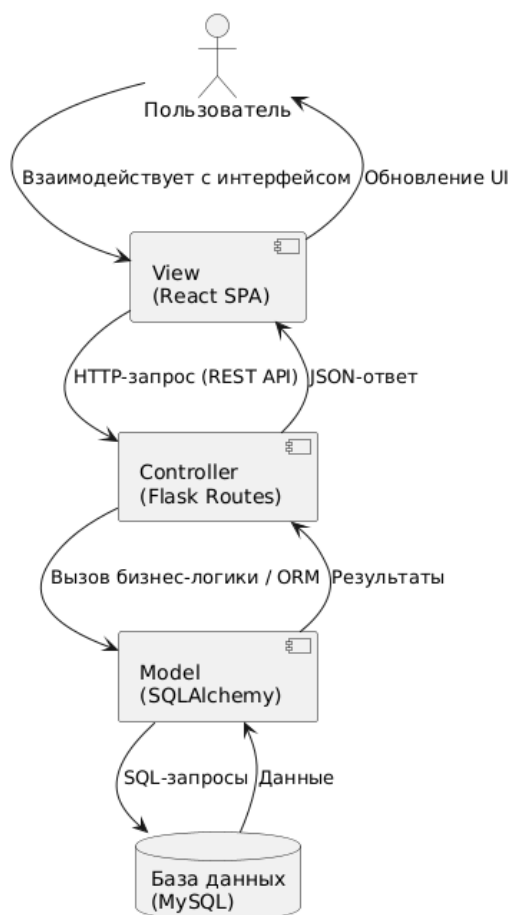


Рисунок 6 – UML-диаграмма компонентов

Клиентская часть реализована с использованием библиотеки React. Она отвечает за отображение данных, взаимодействие с пользователем и отправку запросов к серверу. Интерфейс приложения построен на компонентной архитектуре. Основные разделы:

- каталог товаров;
- карточки продавцов;
- карточки товаров;
- личный кабинет пользователя;

React-компоненты отображают данные, полученные через REST API, и динамически обновляются при изменении состояния, что обеспечивает корректное отображение интерфейса для пользователя.

Серверная часть приложения реализована с использованием Python и микрофреймворка Flask. Flask отвечает за обработку запросов, выполнение бизнес-логики и взаимодействие с базой данных. REST API формирует и возвращает ответы в формате JSON.

Контроллеры Flask принимают запросы от клиента, обрабатывают их, выполняют соответствующую логику, взаимодействуют с моделями и возвращают данные обратно. Они организуют маршруты API и обрабатывают доступ к данным в соответствии с правами пользователя.

Модели реализованы с использованием SQLAlchemy, выступающей в роли ORM. Это позволяет описывать структуру базы данных через Python-классы, избавляя от необходимости вручную писать SQL-запросы.

Особенностью архитектуры система отзывов — покупатель может оставить отзыв о продавце. Это повышает уровень доверия между участниками рынка.

Роль представления на клиентской стороне выполняют React-компоненты. Получая данные от API, они формируют интерфейс, отображая, например, карточки товаров, профили продавцов, отзывы и другую информацию. Таким

образом, визуальная часть полностью изолирована от бизнес-логики и может развиваться независимо.

Аутентификация реализована на основе токенов доступа. После успешной авторизации пользователь получает токен, который прикрепляется к последующим запросам. Это позволяет контролировать доступ к определённым разделам API, реализовать проверку прав (например, запрет на редактирование чужих данных).

На серверной стороне реализована проверка прав и логика ограничения доступа, обеспечивающая безопасность операций. Например:

- почта и номер телефона пользователя должны быть уникальными;
- продавец может редактировать только свои товары;
- отзыв пользователь может оставить только один раз.

Всё взаимодействие между клиентом и сервером осуществляется через REST API, в котором реализованы CRUD-операции для всех ключевых сущностей. Формат обмена — JSON. API легко расширяем: добавление новых маршрутов и логики не требует пересмотра клиентской части, если структура данных остаётся прежней. На рисунке 7 представлена общая диаграмма последовательности.

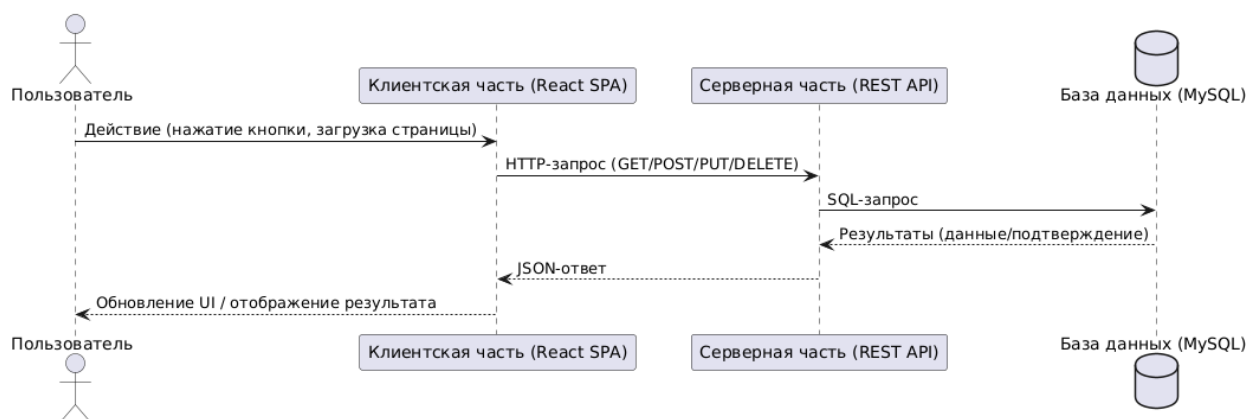


Рисунок 7 – UML- диаграмма последовательности

В качестве СУБД используется MySQL. Работа с базой осуществляется через SQLAlchemy, что упрощает миграции, валидацию и взаимодействие с

таблицами. Важной особенностью является наличие внешних ключей, обеспечивающих согласованность информации. Модели связаны друг с другом отношениями "один ко многим" и "многие ко многим", где это необходимо. На рисунке 8 представлена общая диаграмма развёртывания.

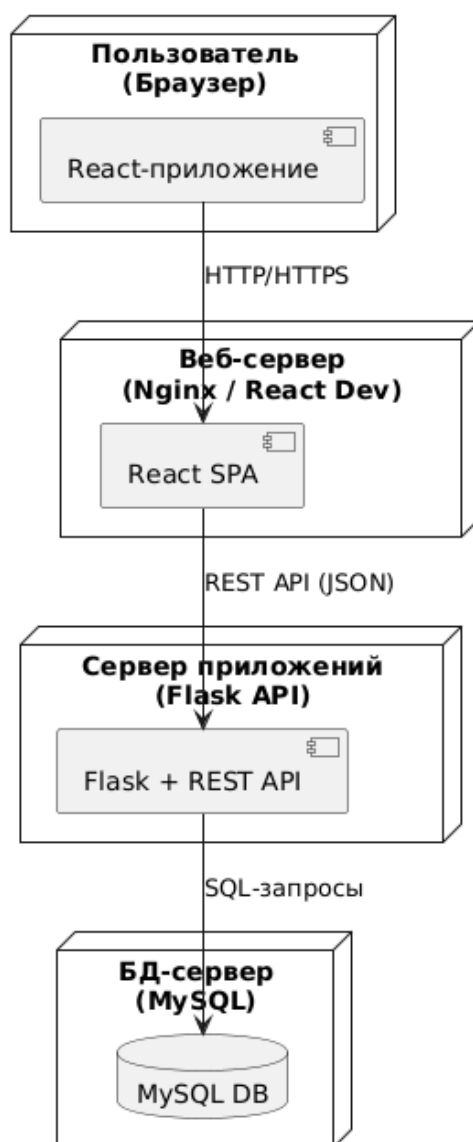


Рисунок 8 – UML- диаграмма развёртывания

3 Выбор инструментов разработки

3.1 Выбор языка программирования

Для реализации серверной части web-приложения было принято решение использовать язык Python совместно с фреймворком Flask. Данный выбор обусловлен рядом преимуществ, которые делают связку Python + Flask оптимальной для разработки лёгких и гибких серверных решений с поддержкой REST API [2].

Преимущества языка Python:

- простота синтаксиса и читаемость кода. Python отличается интуитивно понятным синтаксисом, что упрощает разработку;

- большая экосистема библиотек. Python предоставляет огромное количество готовых решений для работы с базами данных, сериализацией данных, валидацией, аутентификацией и др. — это значительно ускоряет процесс разработки;

- кроссплатформенность и гибкость;

- популярность и большое сообщество.

Преимущества языка Flask:

- гибкость и минимализм. Flask предоставляет только базовый каркас для разработки, позволяя разработчику самостоятельно выбирать необходимые инструменты и расширения;

- большое количество библиотек, взаимодействующих с этим фреймворком;

- простота и высокая скорость разработки. Благодаря лёгкой структуре и отсутствию избыточности, создание REST API и серверной логики происходит быстрее по сравнению с более тяжёлыми фреймворками;

- масштабируемость. Flask позволяет по мере необходимости подключать модули, такие как Flask-SQLAlchemy (работа с базой данных), Flask-JWT (аутентификация), Flask-Migrate (миграции схемы БД) и другие;
- совместимость с современными frontend-фреймворками;
- наличие хорошей документации и большого сообщества.

Таким образом, использование Python и Flask обеспечивает оптимальный баланс между производительностью, гибкостью, расширяемостью и удобством разработки, что делает их эффективным выбором для построения backend-а для web-приложения.

3.2 Выбор базы данных

Для хранения информации о покупателях, фермах, заказах, продуктах и отзывах используется MySQL в связке с SQLAlchemy в качестве ORM.

Причины выбора MySQL и SQLAlchemy:

- эффективность. MySQL отлично справляется с большими объемами данных, а SQLAlchemy оптимизирует взаимодействие с базой через эффективные запросы и кеширование;
- гибкость. SQLAlchemy позволяет работать как с высокоуровневыми ORM-моделями, так и с "сырыми" SQL-запросами при необходимости. Поддерживает миграции (через Alembic), что упрощает изменение структуры БД;
- интеграция. Flask-SQLAlchemy предоставляет готовую интеграцию с Flask, а множество расширений (например, Flask-Migrate) ускоряют разработку;
- целостность данных. SQLAlchemy обеспечивает целостность данных через механизмы внешних ключей, транзакций и сессий;
- поддержка. SQLAlchemy поддерживает не только MySQL, но и PostgreSQL, SQLite и другие СУБД, что упрощает переход между ними при масштабировании.

Таким образом, связка MySQL и SQLAlchemy обеспечивает надёжную, гибкую и масштабируемую основу для хранения и управления данными в проекте. MySQL эффективно справляется с большими объёмами информации, а SQLAlchemy предоставляет удобный и мощный инструмент для работы с базой данных на уровне кода, сочетая преимущества ORM и прямого SQL-доступа [3]. Такая архитектура упрощает поддержку проекта, ускоряет разработку и обеспечивает высокую степень контроля над целостностью и согласованностью данных.

3.3 Выбор среды разработки

Для реализации серверной части приложения на Python + Flask была выбрана среда разработки PyCharm Community Edition. Несмотря на отсутствие некоторых профессиональных функций, эта бесплатная версия IDE предоставляет все необходимые инструменты для эффективной работы.

Преимущества PyCharm Community Edition для разработки на Flask:

- PyCharm предлагает интеллектуальное автодополнение кода;
- удобная навигация между файлами проекта;
- специальная подсветка синтаксиса для Flask-роутов;
- встроенный отладчик позволяет находить и исправлять ошибки непосредственно в процессе разработки;
- работа с системой контроля версий Git прямо из IDE;
- подсветка изменений относительно текущей ветки Git.

Для оптимальной работы в PyCharm Community Edition были выполнены следующие настройки:

- настройка горячих клавиш;
- настройка цветовой схемы интерфейса;
- настроена интеграция с системой контроля версий Git.

Таким образом, PyCharm Community Edition является эффективным инструментом для разработки приложения с использованием Python и Flask. Несмотря на некоторые ограничения бесплатной версии, он предоставляет все необходимые функции для комфортной работы.

3.4 Выбор среды для взаимодействия с сервером

Для управления сервером и взаимодействия с его файловой системой, а также для запуска, настройки и отладки серверного приложения была выбрана утилита Termius. Это кроссплатформенное SSH-приложение, предоставляющее удобный графический интерфейс для подключения к удалённым серверам по защищённому протоколу [4].

Преимущества Termius для взаимодействия с сервером:

- поддержка SSH, SFTP и Mosh-соединений;
- возможность сохранять подключение к серверу с логином и паролем или с использованием SSH-ключей;
- доступ к истории команд в виде списка;
- возможность сохранения команд быстрого ввода в консоль;
- синхронизация настроек и подключений между устройствами через облачный аккаунт Termius;
- визуальное оформление и кастомизация интерфейса для повышения удобства.

Таким образом, Termius является эффективным и надёжным инструментом для взаимодействия с серверной инфраструктурой. Он значительно упрощает управление удалёнными серверами, особенно в условиях работы над проектами, требующими регулярного доступа к backend-части через терминал.

4 Бизнес-модель

4.1 Описание продукта

Электронный рынок фермерской продукции – это web-платформа, которая объединяет фермеров и покупателей, обеспечивает удобный поиск.

Целевая аудитория приложения – покупателя, предпочитающие свежие фермерские продукты. К ним можно отнести жителей города, которые придерживаются здорового питания, рестораны и кафе, ищущие локальных поставщиков, люди, желающие покупать экологически чистые продукты напрямую у производителей. Так же целевой аудиторий выступают фермеры и мелкие производства, которые хотят расширить сбыт:

- частные фермерские хозяйства;
- владельцы садов и огородов;
- производители органической продукции.

К основным функциям платформы можно отнести поиск фермеров и товаров при помощи интерактивной карты с геолокацией продавцов и фильтру по категории, прозрачность сделок, которая будет осуществляться с помощью подробных карточек товара с описанием, персонализированный опыт в виде личного кабинета продавца, где он может управлять ассортиментом.

В данной платформе реализованы пользователи двух ролей – покупатели и продавцы. Для первых преимущество будет заключаться в доступе к свежим фермерским продуктам без посредников, в возможности проверить происхождение товаров через отзывы и рейтинги, в экономии времени за счет удобного поиска ближайшего производителя на карте. Для продавцов же очевидными преимуществами будут расширение клиентской базы без затрат на маркетинг и управлением продажами через удобный личный кабинет.

В процессе реализации был создан адаптивный интерфейс, созданный с помощью языка JavaScript и библиотеки React, а также серверная часть, который

был написан на языке Python (Flask). Общение между клиентской и серверной частью осуществлялось при помощи REST API. Хранение практически всей информации происходило в базе данных (MySQL).

Планом развития нашего проекта является создание чата, где покупатель и производитель смогли бы договориться о деталях заказа, внедрение системы подписок на фермерские продукты (регулярные заказы) будет очень актуально для кафе и ресторанов, интеграция с платежными системами, разработка мобильного приложения для iOS/Android.

Платформа устраняет ключевые проблемы рынка: недоверие покупателей, сложность поиска локальных продуктов и отсутствие удобных инструментов для фермеров.

Таким образом, «Электронный рынок фермерской продукции» — это не просто маркетплейс, а экосистема для честной и удобной торговли фермерскими товарами.

4.2 Потоки доходов

Платформа позволит получать доход за счет следующих функций:

– контекстная реклама. За счет показа сначала товаров от тех производителей, которые оплатили рекламу, эти фермеры смогут получать больше заказов, а значит и прибыли. А платформа сможет получать доход от партнеров за рекламу;

– система подписок. Благодаря платным подпискам покупатели смогут оформлять регулярные заказы свежих и натуральных продуктов, особый интерес это вызовет у ресторанов и кафе. Так же подписку смогут оформлять и обычные покупатели, например, для доступа к редким товарам производителя;

– доставка. Взимание платы за доставку товаров от производителя к покупателю.

4.3 Каналы распространения

Для развития приложения требуется не только его поддержка и доработка. Необходимо также его продвигать и привлекать новых клиентов.

Продвижение платформы будет проводиться с помощью онлайн-маркетинга и таких инструментов как SEO, контекстная реклама, социальные сети.

Привлечь новых клиентов также помогут партнерские программы. Проведение совместных акций с министерством сельского хозяйства поможет не только увеличить уровень доверия к министерству у фермеров, но также и привлечь новых производителей на платформу.

4.4 Взаимоотношения с клиентами

Привлечение новых клиентов важный аспект, однако немаловажным также является выстраивание и поддержка взаимоотношений с действующими клиентами. Для этого необходимо ведение сообщества пользователей, система онлайн поддержки, а также постоянное обновление контента.

Ведение сообщества пользователей позволит создать среду для обмена опытом фермеров и обычных людей. Также в сообществе можно информировать пользователей об акциях, предложениях партнеров и уведомлять о нововведениях на платформе.

4.5 Ключевые ресурсы

Для разработки и поддержания работоспособности приложения нужны такие ресурсы как разработчики, тестировщики, контент, специалист по маркетингу.

На разработчиках лежит задача разработки архитектуры приложения, написания кода. Так как платформа должна работать качественно из-за большого масштаба, то разработчики не смогут уследить за всеми неточностями, поэтому с этими задачами будет справляться специалист по тестированию, который сможет

своевременно протестировать платформу и дать точный отзыв разработчикам. Чтобы на нашу платформу приходило больше продавцов и покупателей, нужно развивать интерес в двух направлениях соответственно: продвигать и рассказывать о нашем проекте в различных фермерских объединениях и клубах для поиска фермеров и создание актуальной, интересной рекламы для привлечения покупателей. Для выполнения этой задачи нужен специалист по маркетингу и рекламе.

4.6 Партнеры

Для успешного развития платформы ключевыми партнерами могут стать:

- сельскохозяйственные кооперативы и объединения помогут привлекать на платформу новых производителей, могут учувствовать в сертификации продукции и обеспечат контроль качества товаров;
- эко-магазины и кафе здорового питания смогут размещать рекламу и специальные предложения. Также можно создать партнерские программы лояльности и организацию совместных промоакций и дегустаций;
- логистические компании обеспечат доставку товаров покупателям. Могут предложить специальные тарифы для пользователей платформы;
- платежные системы и банки обеспечат безопасные транзакции;
- государственные и общественные организации.

Каждый из партнеров получит выгоду от сотрудничества: фермеры – новые возможности сбыта, покупатели – гарантии качества и удобство покупок, бизнес-партнеры – доступ к целевой аудитории, государство – развитие локальных производителей.

4.7 Оценка затрат и доходов

Разработка и запуск:

- дизайн UX/UI и прототипирование: 100 000 руб.;

					ПГУ1.090301.21.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

- разработка фронтенда: 130 000 руб.;
- разработка бекенда: 150 000 руб.;
- настройка и развертывание серверной инфраструктуры: 70 000 руб.;
- тестирование и отладка: 60 000 руб.

Итого первоначальные затраты: 510 000 руб.

Ежемесячные расходы:

- хостинг и обслуживание серверов: 30 000 руб./мес.;
- поддержка и обновление платформы: 60 000 руб./мес.;
- модерация контента и служба поддержки: 35 000 руб./мес.;
- маркетинг и продвижение: 75 000 руб./мес.

Итого ежемесячные затраты: 200 000 руб.

Прогнозируемые доходы (при 5 000 активных пользователях).

Комиссия с доставки.

Средний стоимость доставки ~ 200 руб.

Количество заказов в месяц: 5 000.

Комиссия платформы (30 руб. за каждую доставку): 150 000 руб./мес.

Подписка на фермерскую продукцию.

10% покупателей (750 пользователей) * 200 руб. = 150 000 руб./мес.

Контекстная реклама и партнерские программы.

Контекстная реклама: 10 000 руб./мес.

Партнерские программы с эко-магазинами: 10 000 руб./мес.

Итого ежемесячный доход: 320 000 руб.

4.8 Расчет срока окупаемости

Для расчета срока окупаемости необходимо учесть первоначальные и ежемесячные затраты и сравнить их с ежемесячными доходами.

Чистый месячный доход: 320 000р. (доход) - 200 000 (расходы) = 120 000р.

Срок окупаемости: 510 000 (первоначальные затраты) / 120 000 (чистый ежемесячный доход) = 5 месяцев.

Платформа окупится при следующих условиях:

– активная пользовательская база должна состоять из 5,000 активных покупателей;

– при конверсии премиум подписки = 10% от общей базы;

– средний доход от рекламы должен быть 10 000 р. на пользователя в месяц.

Таким образом, при указанных условиях, приложение окупится через 7 месяцев.

					ПГУ1.090301.21.001 ПЗ	Лист
						31
Изм.	Лист	№ докум.	Подпись	Дата		

5 Реализация приложения

Для успешно и эффективной реализации серверной части web-приложение необходимо выстроить правильную последовательность этапов для грамотного использования рабочих ресурсов и создания качественного и стабильного приложения.

Этапы разработки серверной части web-приложения:

- описание функционала;
- разработка структуры БД;
- разработка бизнес-логики;
- написание документации;
- тестирование приложения;
- деплой приложения.

5.1 Описание функционала

В web-приложении предусмотрена авторизация пользователя, как продавца или покупателя, пользователь регистрируется или входит в уже имеющийся аккаунт. Доступ к функционалу зависит от того, кем является пользователь покупателем или продавцом. Так же есть функции, к которым есть доступ без авторизации:

- просмотр каталога фермерской продукции;
- фильтрация товаров по категориям;
- поиск ближайших продавцов на карте;
- просмотр личной страницы продавца;
- просмотр карточки товара.

Если пользователь авторизировался как покупатель он получает возможность оставить отзыв о продавце, но не больше одного, во избежание накрутки отзывов, так же покупатель может удалить отзыв.

Пользователь авторизованный, как продавец может:

- изменить или добавить описание в своей карточке продавца, для привлечения пользователя (например, оставить свои контакты, или указать возможность доставки продуктов);
- добавить аватар для своей карточки продавца;
- создать/удалить карточку товара;
- указать адреса точек продажи с их времен работы.

При первом входе в приложение клиентская часть отправляет несколько запросов серверу на получение данных:

- получить список категорий;
- получить список товаров от ближайших продавцов (клиентская часть отправляет вместе с запросом геопозицию пользователя и радиус поиска);
- получить адреса ближайших продавцов (для отображения их на карте) (клиентская часть отправляет вместе с запросом геопозицию пользователя и радиус поиска).

Серверная часть принимает запросы и их входные данные, обрабатывает их и отправляет им ответ. На рисунке 9 представлен пример главной странице не авторизованного пользователя, получившей ответы на запросы.

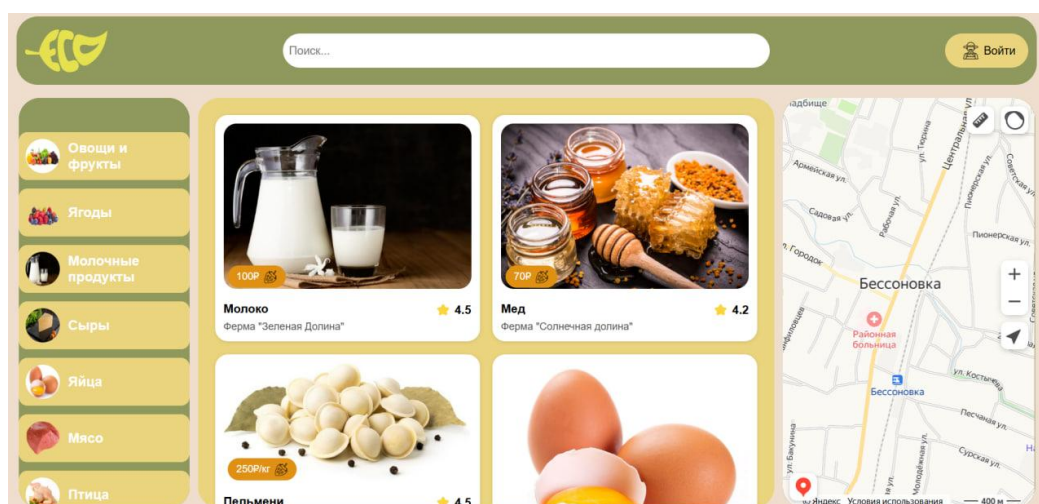


Рисунок 9 – Страница для не авторизованных пользователей

Находясь на главной странице, пользователь может зарегистрироваться, ему необходимо указать:

- имя;
- фамилию;
- название производства (только продавцу);
- номер телефона;
- почту;
- пароль.

Номер телефона и почта должны быть уникальными, нельзя использовать одну и ту же почту для создания одного аккаунта, это проверяется в серверной части и, если номер и почта уже ранее зарегистрированы, то со стороны сервера поступит ответ, сообщающий об этом. При создании пользователем второго аккаунта, отличающегося по типу (продавец/покупатель) от существующего, можно использовать один и тот же номер или почту. На рисунке 10 представлен пример окна регистрации пользователя.

Рисунок 10 – Окно регистрации пользователя

Для входа пользователя необходимо ввести почту и пароль. На рисунке 11 представлен пример окна входа пользователя.

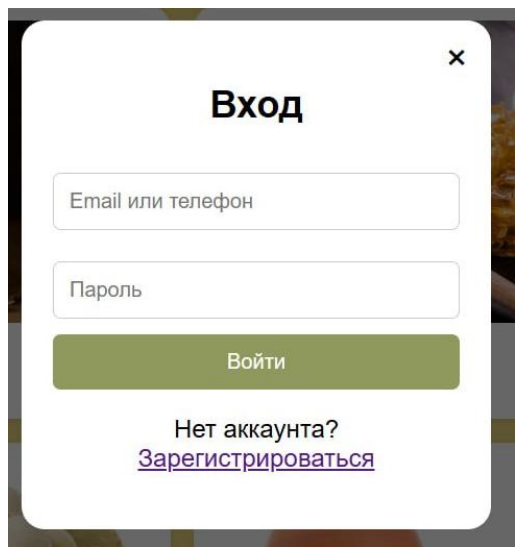


Рисунок 11 – Окно входа пользователя

После авторизации пользователя как покупателя у него появляется возможность оставить отзыв о продавце на его карточке. После авторизации пользователя как продавца откроется его личный кабинет, в котором он уже сможет изменить внешний вид своей карточки, добавить адреса и товар.

5.2 Описание структуры БД

Для реализации вышеописанного функционала необходимо реализовать структуру хранения данных. Реализованы они будут при помощи MySQL, а взаимодействие с ними будет происходить при ORM SQLAlchemy. Необходимо использовать несколько сущностей, чтобы правильно структурировать данные. Вот эти сущности:

- Buyers (покупатели);
- Sellers (продавцы);
- SellerAddress (адреса продавцов);
- Categories (категории);

– Products (товары);

– Reviews (отзывы).

На рисунке 12 представлена диаграмма сущностей и их отношений.

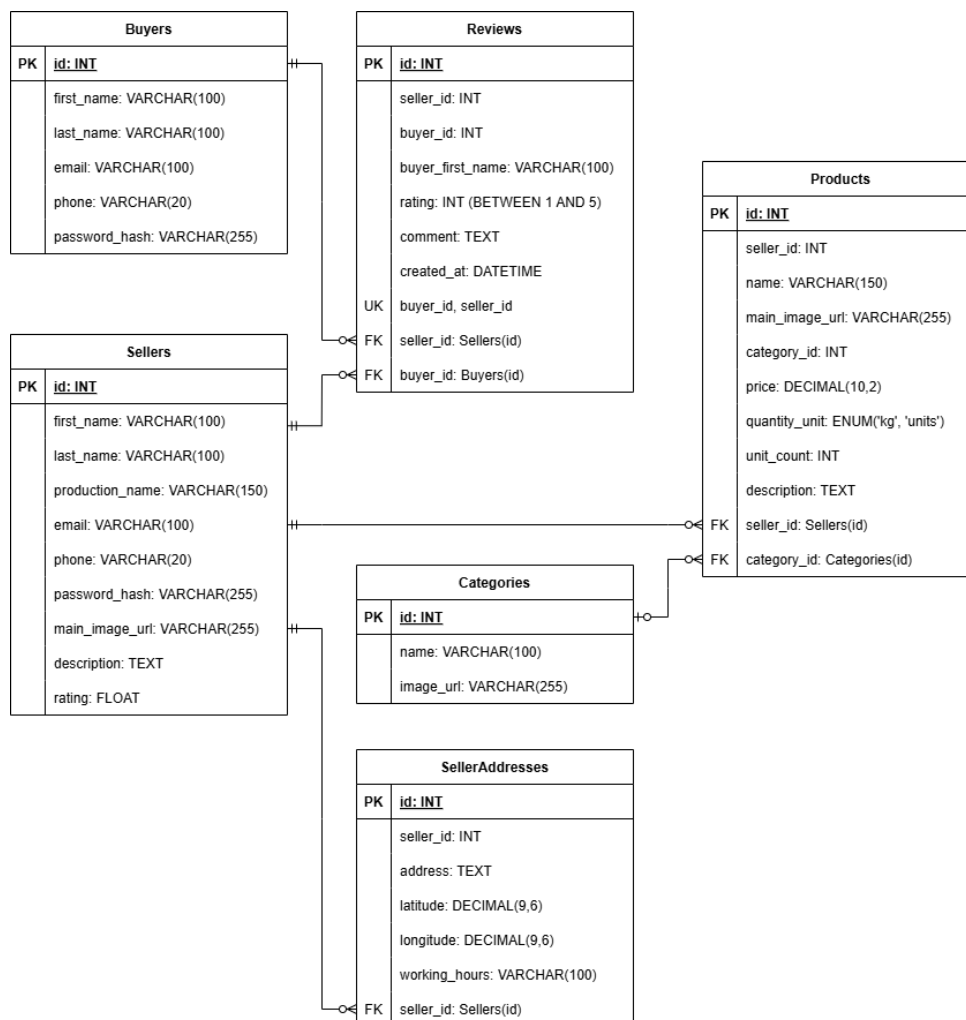


Рисунок 12 – Диаграмма сущностей

5.2.1 Buyers

Хранит информацию о зарегистрированных покупателях:

- id — уникальный идентификатор покупателя;
- first_name — имя;
- last_name — фамилия;
- email — почта;
- phone — номер телефона;

– password_hash — зашифрованный пароль для аутентификации.

5.2.2 Sellers

Содержит данные о продавцах:

- id — уникальный идентификатор продавца;
- first_name — имя;
- last_name — фамилия;
- production_name — название фермерского хозяйства;
- email — почта;
- phone — номер телефона;
- password_hash — зашифрованный пароль для аутентификации;
- main_image_url — изображение профиля или логотип;
- description — описание продавца;
- rating — средняя оценка на основе отзывов покупателей.

5.2.3 SellerAddress

Хранит информацию об адресах продавцов:

- id — уникальный идентификатор адреса;
- seller_id — внешний ключ, связанный с таблицей Sellers;
- address — текстовое описание адреса (город, улица, дома);
- latitude — координата широты;
- longitude — координата долготы;
- working_hours — время работы.

5.2.4 Categories

Хранит информацию о категориях товара:

- id — уникальный идентификатор категории;
- name — название категории;
- image_url — ссылка на иконку или изображения категории.

5.2.5 Products

Хранит информацию о товаре, выставленном продавцами:

- id — уникальный идентификатор продукта;
- seller_id — внешний ключ на продавца (Sellers);
- name — название продукта;
- main_image_url — ссылка на изображение продукта;
- category_id — внешний ключ на категорию (Categories);
- price — цена;
- quantity_unit — единица измерения: килограммы или штуки;
- unit_count — количество в одной продаже (например, 1 кг или 6 штук);
- description — описание товара.

5.2.6 Reviews

Хранит информацию об отзывах:

- id — уникальный идентификатор отзыва;
- seller_id — внешний ключ на продавца (Sellers);
- buyer_id — внешний ключ на покупателя (Buyers);
- buyer_first_name — имя покупателя (сохраняется отдельно для отображения);
- rating — оценка от 1 до 5;
- comment — текстовый отзыв;
- created_at — дата и время создания отзыва;
- UNIQUE KEY (UK) — обеспечивает, что один покупатель может оставить только один отзыв для одного продавца.

5.3 Описание бизнес-логики

Пользователь взаимодействует с клиентской частью, а она в свою очередь с серверной. Взаимодействия между ними происходит при помощи REST API. Пользователь совершает какое-либо действие в интерфейсе, требующее работы с данными (получение, отправка, проверка, изменения, удаление и т.д.), клиентская

часть отправляет HTTP-запрос, серверная часть получает их, выполняет обработку и возвращает ответ в формате JSON-ответа.

Каждая функция приложения реализована как отдельный эндпоинт API, поддерживающий стандартные HTTP-методы:

- GET — получение данных (список товаров, заказов, пользователей);
- POST — создание новых сущностей (регистрация, добавление товара, оформление заказа);
- PUT/PATCH — обновление данных (редактирование товара, изменение статуса заказа);
- DELETE — удаление сущностей (удаление товара, отзыва и т.д.).

Все данные хранятся в реляционной базе данных MySQL, а доступ к данным осуществляется через ORM-библиотеку SQLAlchemy. Для реализации работы этой библиотеки с БД были созданы следующие классы:

- Buyer;
- Seller;
- SellerAddress;
- Category;
- Product;
- Review.

Использование такого подхода упрощает написание кода и позволяет значительно ускорить обработку. Каждый класс содержит поля, описывающие то, как работать со столбцами в таблице. Благодаря этому работая с классом и его полями мы работаем с данными в БД.

В рамках разработки REST API, в соответствии с заявленным функционалом, были реализованы ключевые эндпоинты для управления учетной записью и другими данным. На рисунке 13 представлена диаграмма REST API эндпоинтов, иллюстрирующая возможные варианты работы с серверной частью и данными.

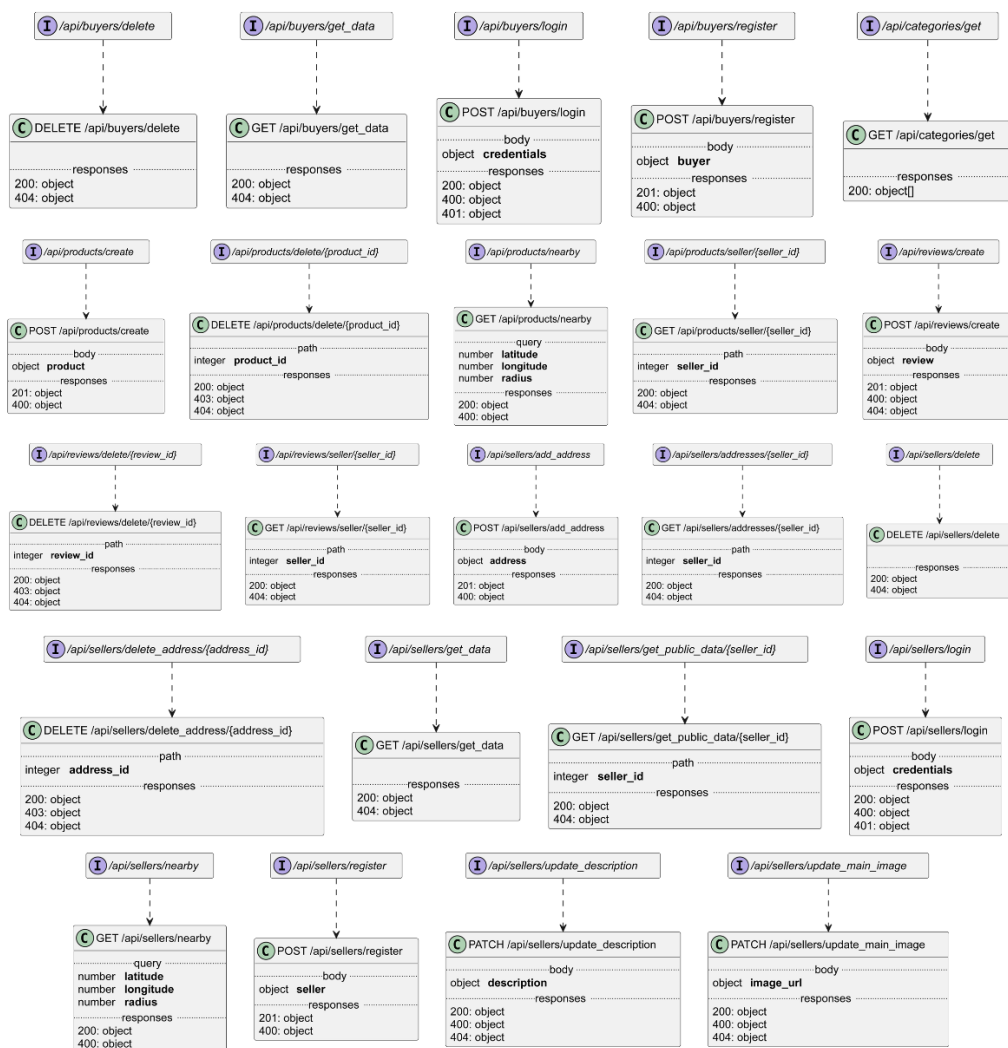


Рисунок 13 – Диаграмма REST API эндпоинтов

Для обеспечения безопасного взаимодействия с данными реализованы:

- аутентификация с использованием JWT;
- защищённые маршруты требуют наличие JWT в заголовке;
- хэширование паролей;
- валидация данных при входе или регистрации.

Так же для обеспечения безопасности используется файл .env для хранения переменных окружения, такие как JWT-ключ и данные для авторизации пользователя базы данных. Важно отметить, что пользователь, используемый для взаимодействия с базой данных через приложение ограничен в правах и может только добавлять, удалять и обновлять данные.

5.4 Описание документации

При разработке серверной части важно учесть, что разработчик клиентской части должен знать о функционале, который ему предоставляет серверная часть. В качестве инструмента для описания был выбран Flasgger это — расширение для Flask, которое упрощает разработку API, используя Swagger UI и спецификацию OpenAPI. Swagger — это инструмент для документирования и проверки API. На рисунке 14 представлен Swagger UI.

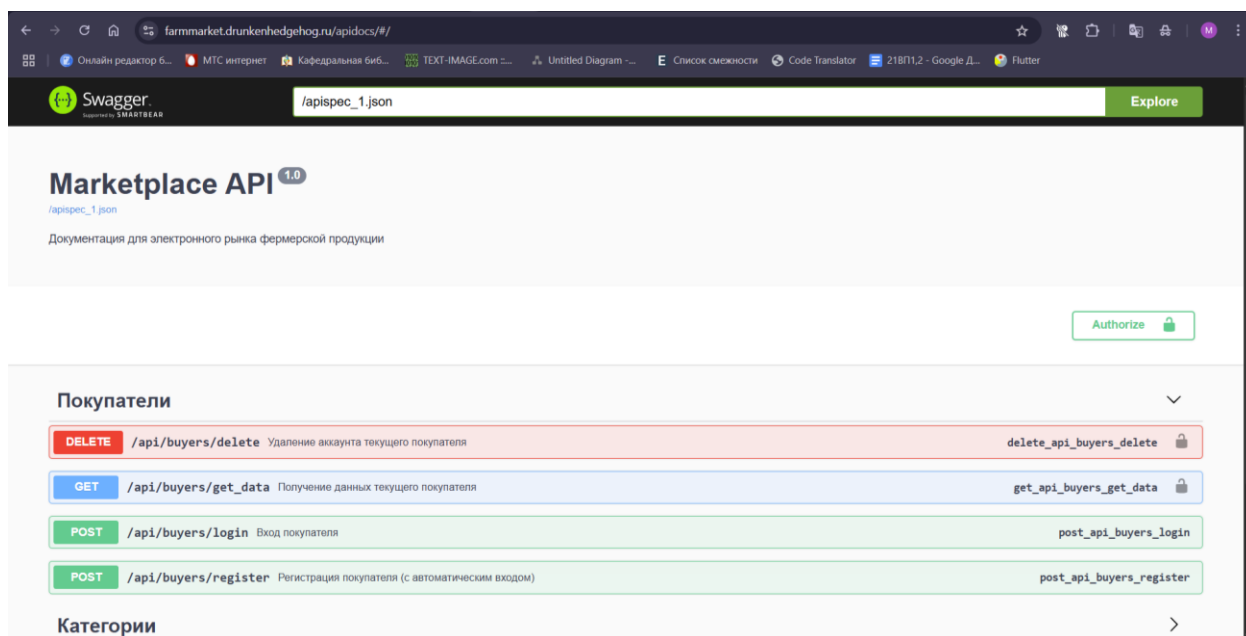


Рисунок 14 – Swagger UI

Причины выбора Swagger и Flasgger:

- интерактивный интерфейс Swagger UI;
- автоматическая генерация документации;
- упрощение командной работы и интеграции;
- минимизация ошибок и ускорение разработки;
- поддержка валидации входных данных.

5.5 Тестирование приложения

Одной из ключевых задач при создании серверной части web-приложения является его тестирование. Этот процесс помогает обнаружить ошибки в коде, сбои в работе системы и ошибки в логике приложения. Тестирование проводилось в Swagger-е, так он имеет удобный и наглядный интерфейс, в котором видны ответы на запросы.

С целью выявления неисправностей были проверены следующие функции приложения:

- регистрация;
- авторизация;
- получение ближайших продуктов;
- получение ближайших продавцов;

5.5.1 Регистрация

На рисунке 15 представлен запрос на авторизацию и положительный ответ на него. В поле Curl находится содержимое запроса и его структуру, поле Response body содержит ответ в формате JSON. Код ответа 201 значит, что запрос выполнен успешно и привёл к созданию ресурса. В файле JSON находятся токен авторизации, сообщение и статус “успешно”.

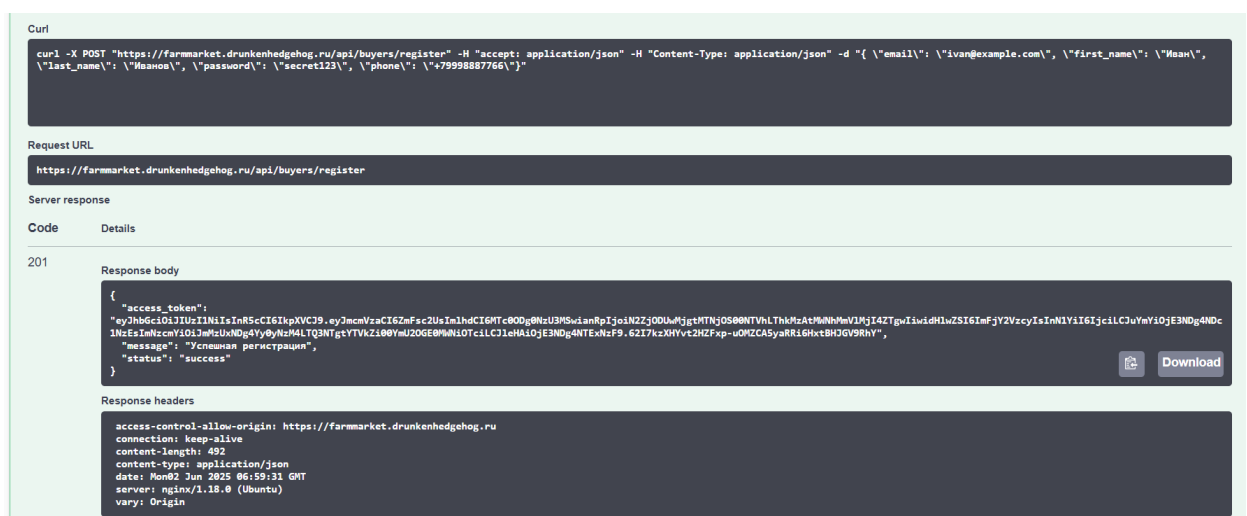


Рисунок 15 – Запрос на регистрацию

На рисунке 16 представлен запрос на авторизацию и отрицательный ответ на него. Код ответа 400 значит, что пользователь оправляет запрос с неверными данными. В файле JSON находятся причина ошибки, которую можно показать пользователю и статус ответа “отрицательный”.

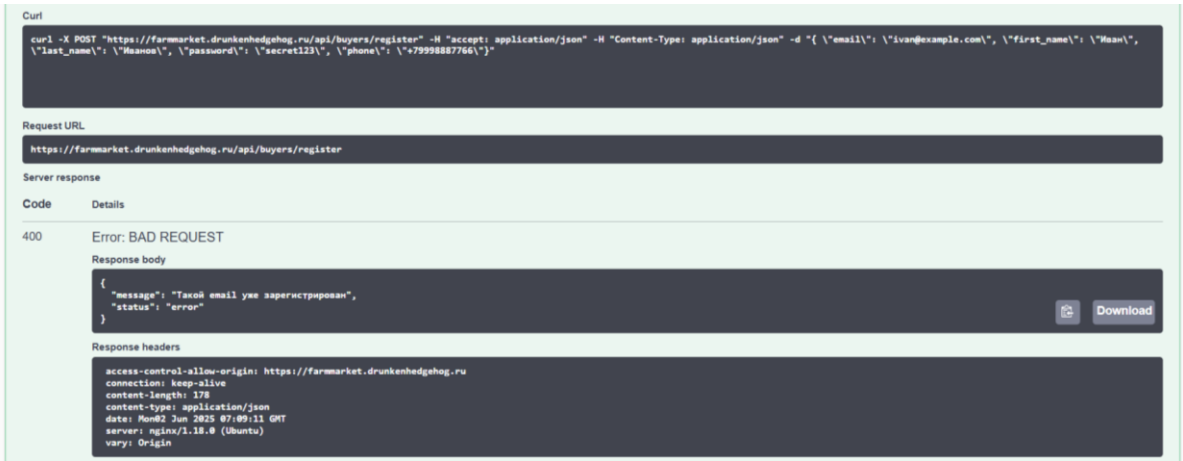


Рисунок 16 – Запрос на регистрацию

5.5.2 Авторизация

Если пользователь введёт верные данные для авторизации он получит ответ с кодом 200, в JSON файле будет токен, сообщение и статус. Пример такого запроса на авторизацию представлен на рисунке 17.

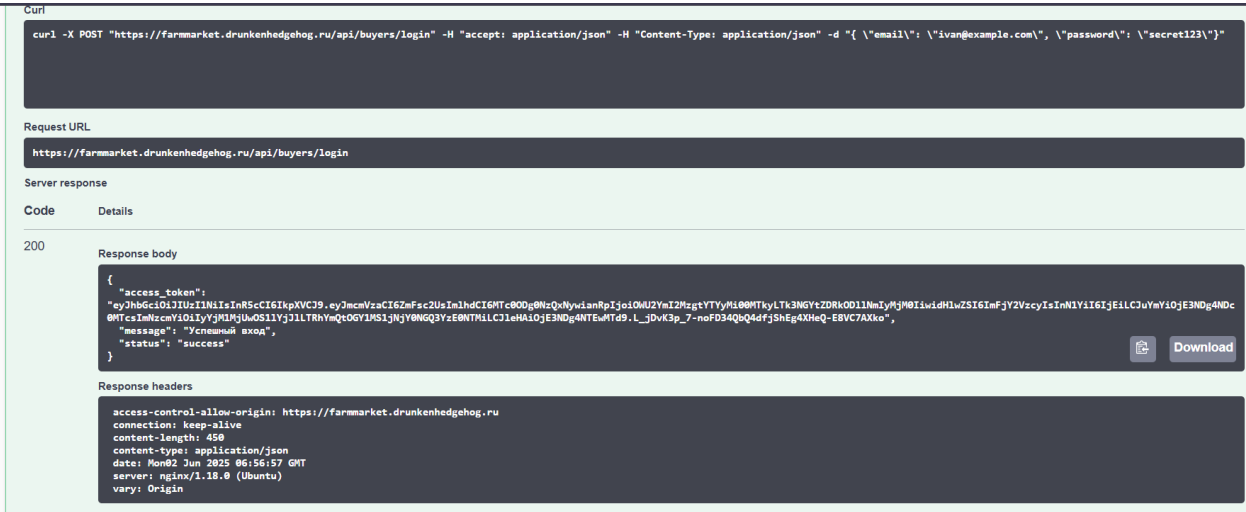


Рисунок 17 – Запрос на авторизацию

Если пользователь введёт неверные данные, например пароль, он получит ответ с кодом 401, которые значит, что пользователя вводит неверный данные для авторизации. Пример такого запроса на авторизацию представлен на рисунке 18.

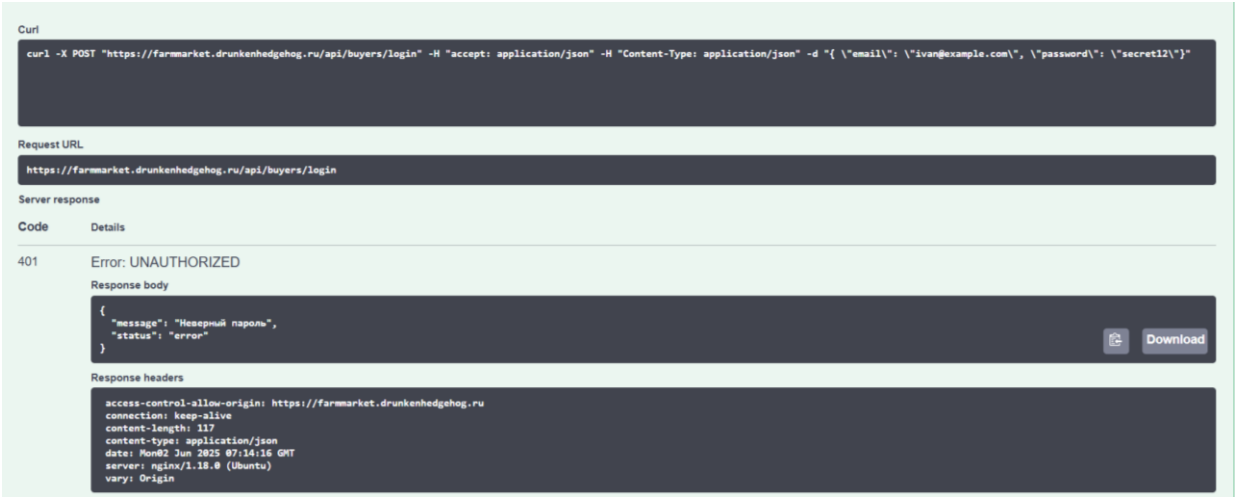


Рисунок 18 – Запрос на авторизацию

5.5.3 Получение ближайших продуктов

Для получения ближайших продуктов пользователь должен отправить широту, долготу и радиус поиска. На рисунке 19 представлен запрос поиска ближайших продуктов. Ответ получен в виде массива JSON.

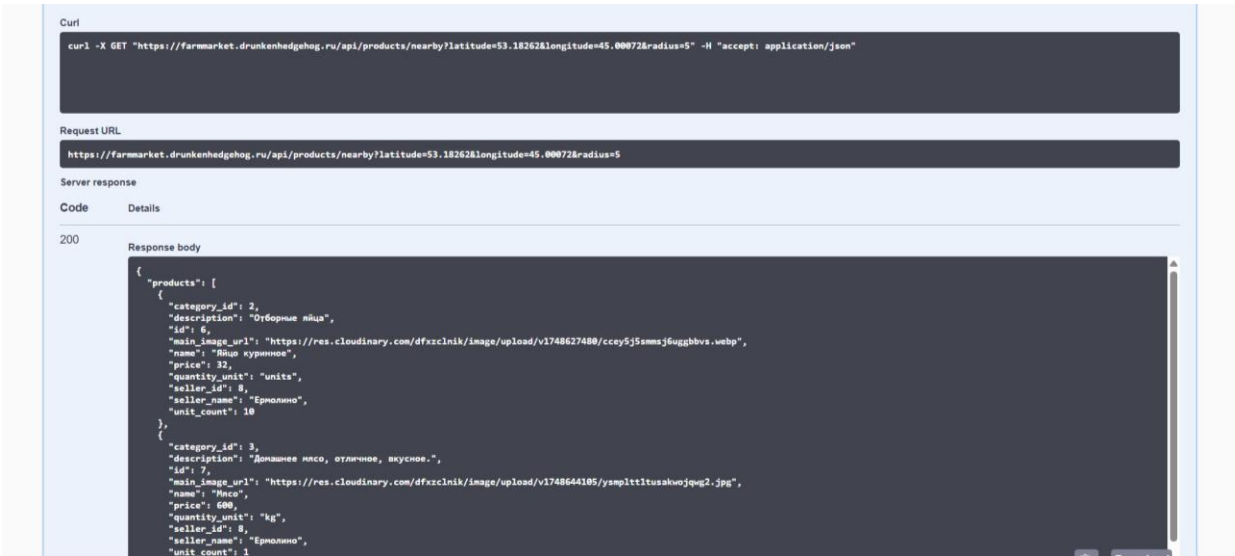


Рисунок 19 – Запрос поиска ближайших продуктов

5.5.4 Получение ближайших продавцов

Для получения ближайших продавцов пользователь должен отправить широту, долготу и радиус поиска. На рисунке 20 представлен запрос поиска ближайших продавцов. Ответ получен в виде массива JSON.

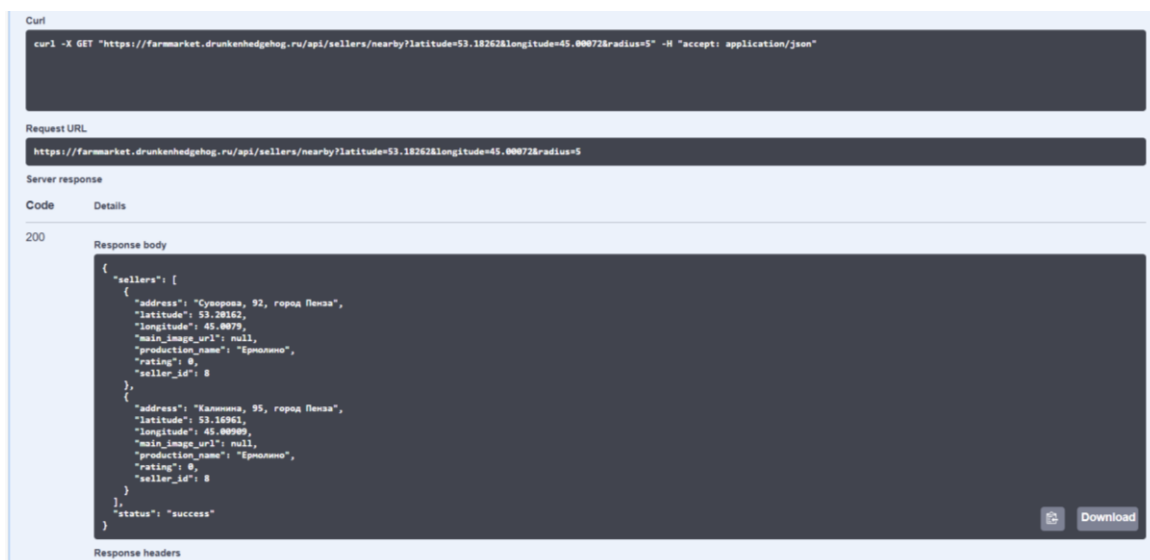


Рисунок 20 – Запрос поиска ближайших продавцов

5.6 Описание деплоя приложения

Для деплоя был арендован сервер на ОС Ubuntu. После создаём конфиг для подключения через SSH в Termius. На рисунке 21 представлен Termius.

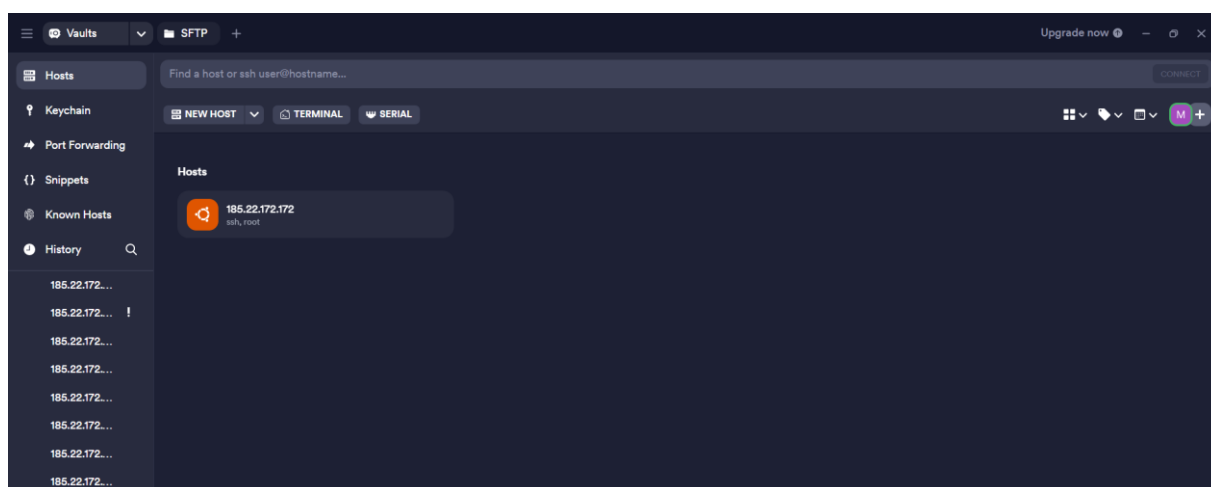


Рисунок 21 – Termius

На сервере должны быть установлены и настроены базовые инструменты:

- Python 3;
- MySQL;
- NGINX;
- Git;

Если чего-то из этого нет, то устанавливаем при помощи команды `sudo apt install <название пакета>` необходимые программы. Копируем файл с запросами для создания базы данных в виде файла `.sql` на сервер. Затем при помощи команды `git clone <ссылка на репозиторий>` клонируем проект. После этого создаём виртуальное окружение (`python3 -m venv venv`), активируем виртуальное окружение (`source venv/bin/activate`) и устанавливаем зависимости (`pip install -r requirements.txt`), настраиваем файл `.env` с переменными окружения (БД, секретный ключ). Устанавливаем `gunicorn`. Затем создаём файл конфигурации Nginx, перезапускаем Nginx. Запускаем `gunicorn` (`gunicorn --bind 0.0.0.0:5000 app:app`), предварительно открыв 5000 порт. На рисунке 22 представлен запуск серверной части.

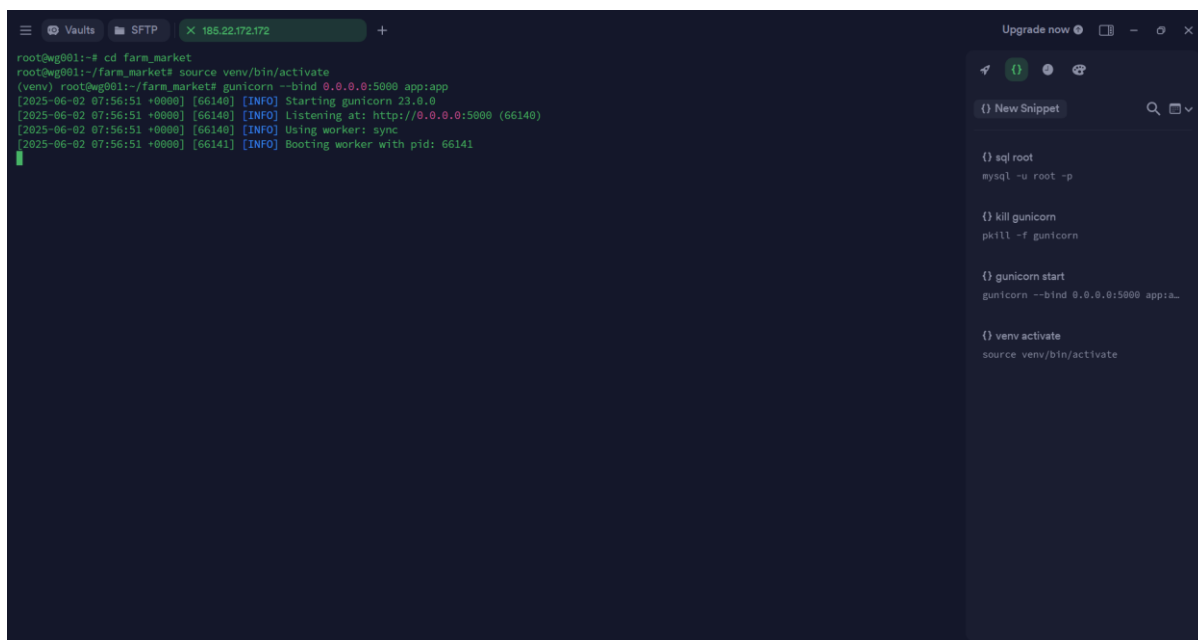


Рисунок 22 – Запуск серверной части

6 Безопасность жизнедеятельности

Использование персонального компьютера и смартфона совсем не так безопасно, как можно подумать. Это убедительно подтверждается информацией, содержащейся в стандарте ГОСТ 12.0.003-2015, описывающем основные вредные и опасные факторы такого трудового процесса. Для обеспечения безопасной работы пользователей и разработчиков были разработаны специальные правила и нормативы. Несоблюдение данных правил может привести к негативным последствиям для физического состояния пользователя и его имущества.

6.1 Правила и техника безопасности при работе с ПК

Согласно требованиям стандарта ГОСТ 12.0.003-2015, при работе с персональным компьютером могут воздействовать следующие вредные факторы:

- высокая температура, исходящая от отдельных компонентов компьютерного оборудования и создающая повышенный температурный фон в помещении;
- высокая степень однообразия выполняемой работы;
- значительная зрительная нагрузка на работника;
- вероятность поражения статическим электричеством при случайном прикосновении к отдельным элементам техники;
- недостаточный уровень освещенности рабочего пространства;
- повышенные уровни электромагнитных и электрических полей, возникающих при работе оборудования;
- чрезмерная яркость и контрастность экрана, способные негативно влиять на зрение при длительном воздействии.

Для снижения вредного влияния необходимо соблюдать определённые меры и рекомендации.

Следует правильно организовать эргономику рабочего места:

					ПГУ1.090301.21.001 ПЗ	Лист
						47
Изм.	Лист	№ докум.	Подпись	Дата		

– правильная посадка. Пользователь должен сидеть с прямой спиной, руки должны находиться на уровне клавиатуры. Стул должен регулироваться по высоте и иметь спинку, обеспечивающую возможность полностью выпрямить ноги и устойчиво поставить ступни на пол. Расстояние до экрана должно быть примерно на вытянутую руку, при этом экран желательно наклонить на 30 градусов для уменьшения нагрузки на спину, шею и руки;

– освещение. Рабочая место должно быть достаточно освещено, но без излишней яркости, чтобы избежать зрительного переутомления. Также важно исключить появление бликов на экране монитора.

В целях безопасности запрещается использовать компьютерную технику при наличии видимых повреждений. Следует регулярно проверять техническое состояние оборудования и не оставлять его включённым без присмотра на длительное время. Рабочие места рекомендуется размещать вдоль стен, чтобы задняя сторона монитора была обращена к стене — это способствует снижению уровня вредного электромагнитного излучения [5].

6.2 Правила и техника безопасности при работе со смартфоном

Так как разработанное приложение будет функционировать на телефоне, то следует также изучить возможное вредное влияние и все правила безопасности по обращению с ним.

Поскольку разработанное web-приложение может использоваться на мобильных устройствах, важно учитывать возможное вредное воздействие смартфонов и соблюдать основные правила безопасной эксплуатации

Не рекомендуется самостоятельно вскрывать, разбирать или изменять конструкцию устройства. Также следует избегать физических воздействий на смартфон — его нельзя ронять, сгибать, бросать, прокалывать, а также подвергать воздействию огня или влаги.

Использование мобильного телефона или зарядного устройства при наличии неисправностей запрещено. Чтобы избежать перегрева и короткого замыкания, следует применять только оригинальные зарядные устройства и кабели. Не оставляйте смартфон подключённым к зарядке под подушкой или в плохо проветриваемых местах — это может привести к перегреву устройства.

Мобильные телефоны постоянно излучают радиоволны, поэтому в целях безопасности не следует держать их близко к жизненно важным органам. Во время разговора рекомендуется периодически менять положение устройства относительно головы. Следует также учитывать, что на данный момент отсутствуют достоверные и всесторонние исследования, подтверждающие вред радиоизлучения [6].

Соблюдение этих рекомендаций способствует не только продлению срока службы мобильного устройства, но и сохранению здоровья пользователя, обеспечивая безопасное и комфортное использование смартфона в повседневной жизни.

Заключение

В ходе выполнения выпускной квалификационной работы была разработана web-приложение «Электронный рынок фермерской продукции». Во время разработки были задействованы следующие технологии:

- HTML – используется для создания структуры web-страниц (разметки интерфейса);

- CSS – применяется для оформления внешнего вида страниц: цветов, шрифтов, отступов и адаптивности;

- JavaScript (JS) – обеспечивает интерактивность пользовательского интерфейса, обработку событий и динамическое обновление контента на клиентской стороне;

- React – JavaScript-библиотека, использованная для построения компонентного и отзывчивого интерфейса клиента (frontend);

- Python – основной язык программирования серверной части (backend), на котором реализована бизнес-логика приложения;

- Flask – web-фреймворк для Python, применяемый для создания REST API и маршрутизации запросов между клиентом и сервером;

- MySQL – система управления базами данных, предназначенная для хранения и управления всей информацией приложения;

- NGINX – web-сервер, использующийся для обработки входящих HTTP-запросов.

Разработанное web-приложение позволяет покупателям находить местных фермеров через интерактивную карту, просматривать каталог свежих фермерских продуктов с подробными описаниями, возможность оставлять отзывы и оценки продавцам. А фермерам, в свою очередь, предлагается возможность создавать карточки своих товаров, управлять ассортиментом и ценами, получаться обратную связь от покупателей.

Одним из достоинств разработанной платформы является наличие удобного и интуитивно понятного для пользователя интерфейса с адаптивным дизайном. А система рейтингов и отзывов и гибкие фильтры поиска обеспечивает прозрачность и простоту использования платформы.

В дальнейшем можно усовершенствовать разработанное приложение, внедрить системы подписок на регулярные поставки, разработать мобильное приложение для iOS/Android, интеграция с платежными системами, создание чата, а добавление системы рекомендаций на основе предпочтений и разработка программы лояльности позволят расширить количество потенциальных пользователей.

					ПГУ1.090301.21.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Tastesofrussia.ru: [Электронный ресурс]. URL: <https://www.tastesofrussia.ru/novosti/objom-onlajn-prodazh-fermerskih-produktov-vyrastet-do-50-mlrd-k-2024/> (дата обращения: 26.05.2025)
2. Habr.com: [Электронный ресурс]. URL: <https://habr.com/ru/companies/skillbox/articles/764384/> (дата обращения: 26.05.2025)
3. Skillbox.ru: [Электронный ресурс]. URL: <https://skillbox.ru/media/code/sqlalchemy-cto-eto-takoe-i-zachem-nuzhno/> (дата обращения: 26.05.2025)
4. DesdeLinux.net: [Электронный ресурс]. URL: <https://blog.desdelinux.net/ru/termius---кросс-платформенный-клиент-SSH-для-сквозного-шифрования/> (дата обращения: 26.05.2025)
5. Oxrana-bez.ru: [Электронный ресурс]. URL: <https://oxrana-bez.ru/stati/tehnika-bezopasnosti-pri-rabote-s-personalnym-kompyuterom/> (дата обращения: 28.05.2025)
6. Stupinoadm.ru: [Электронный ресурс]. URL: <https://stupinoadm.ru/novosti/detail/mery-bezopasnosti-pri-ispolzovanii-mobilnykh-ustroystv/> (дата обращения: 28.05.2025)

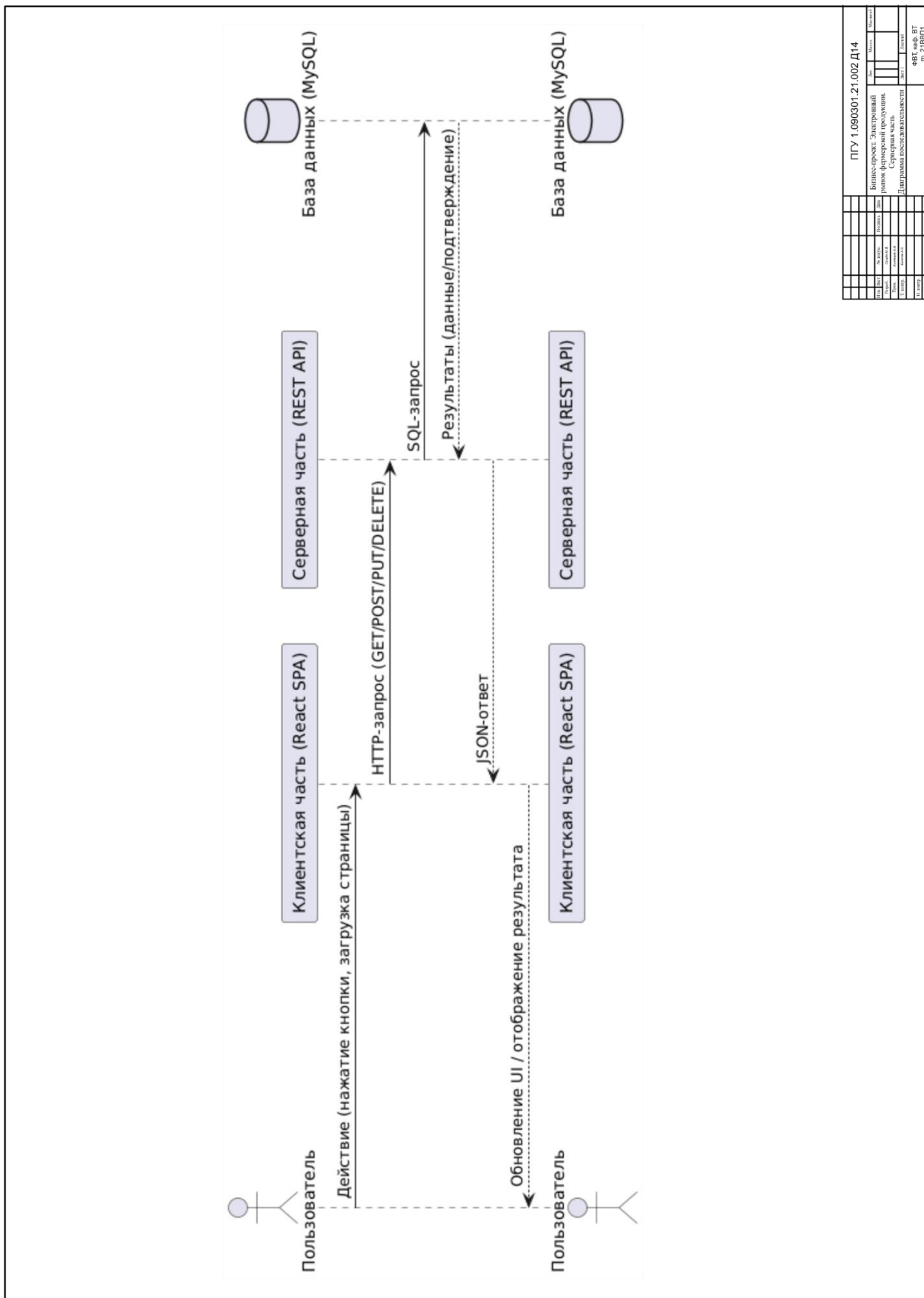
Приложение А

(Обязательное)

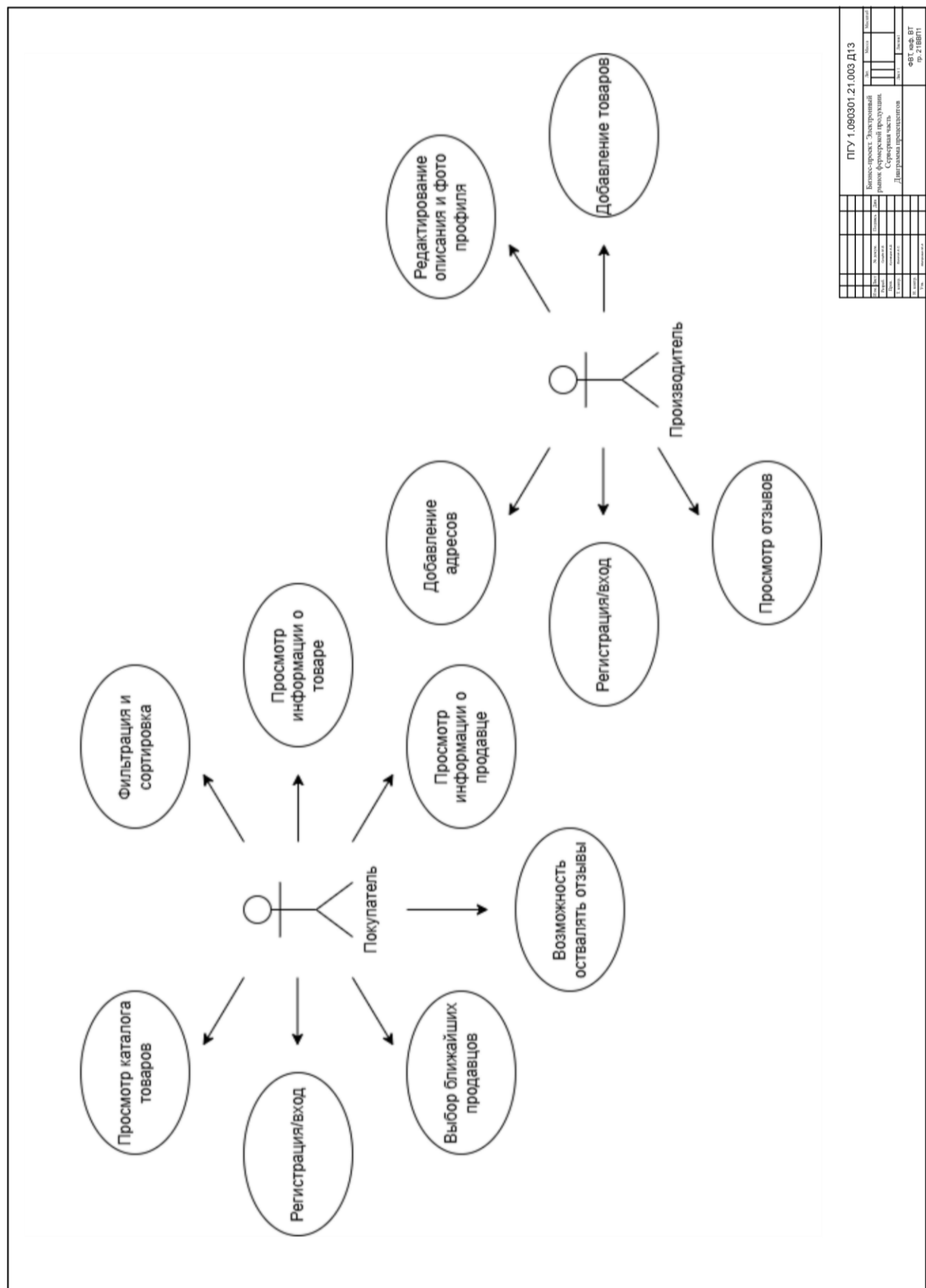
Бизнес-проект. Электронный рынок фермерских продуктов. Серверная часть
UML-диаграммы

					ПГУ1.090301.21.001 ПЗ	Лист
						53
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1-Диаграмма последовательности



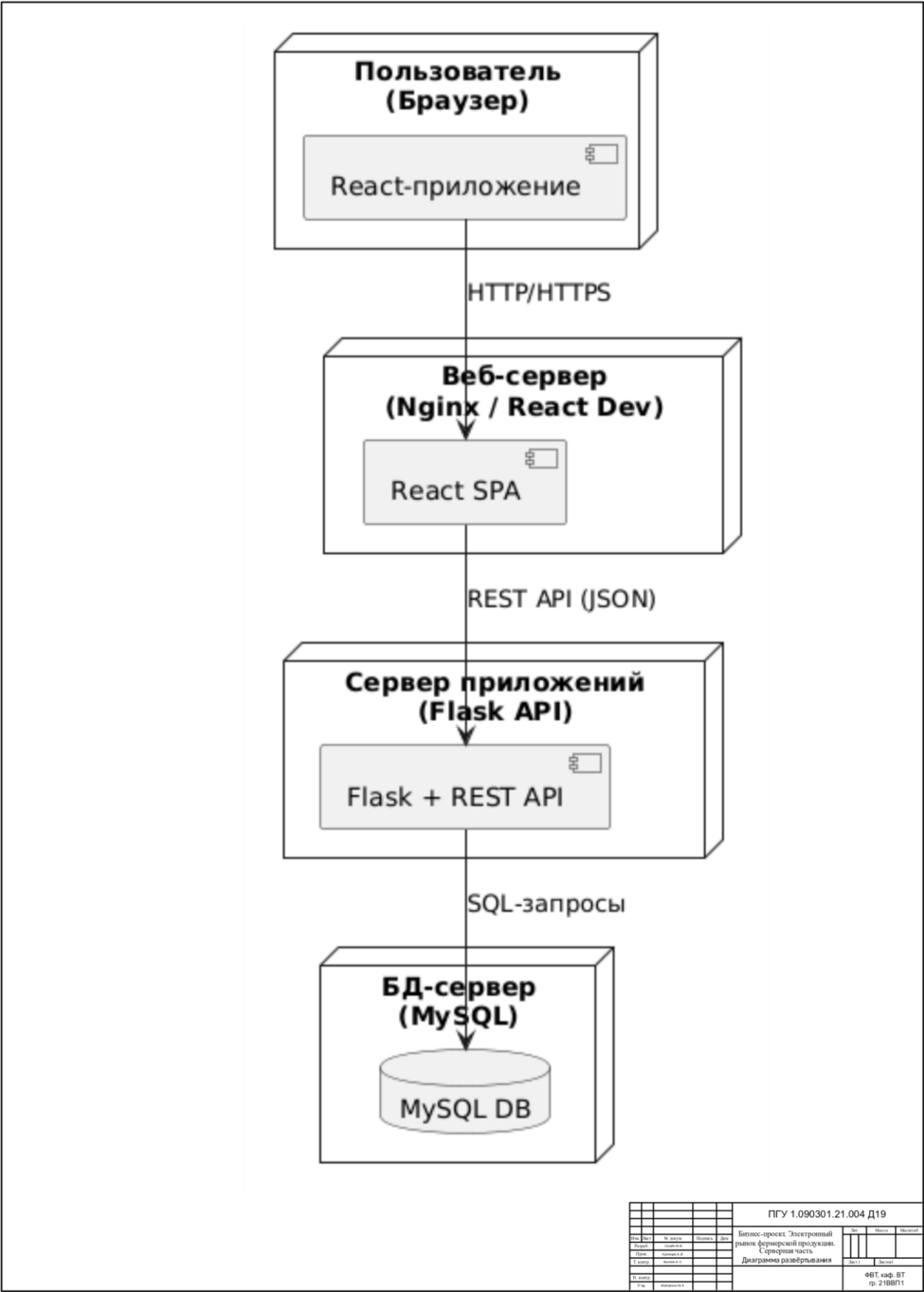
Приложение А.2-Диаграмма прецедентов



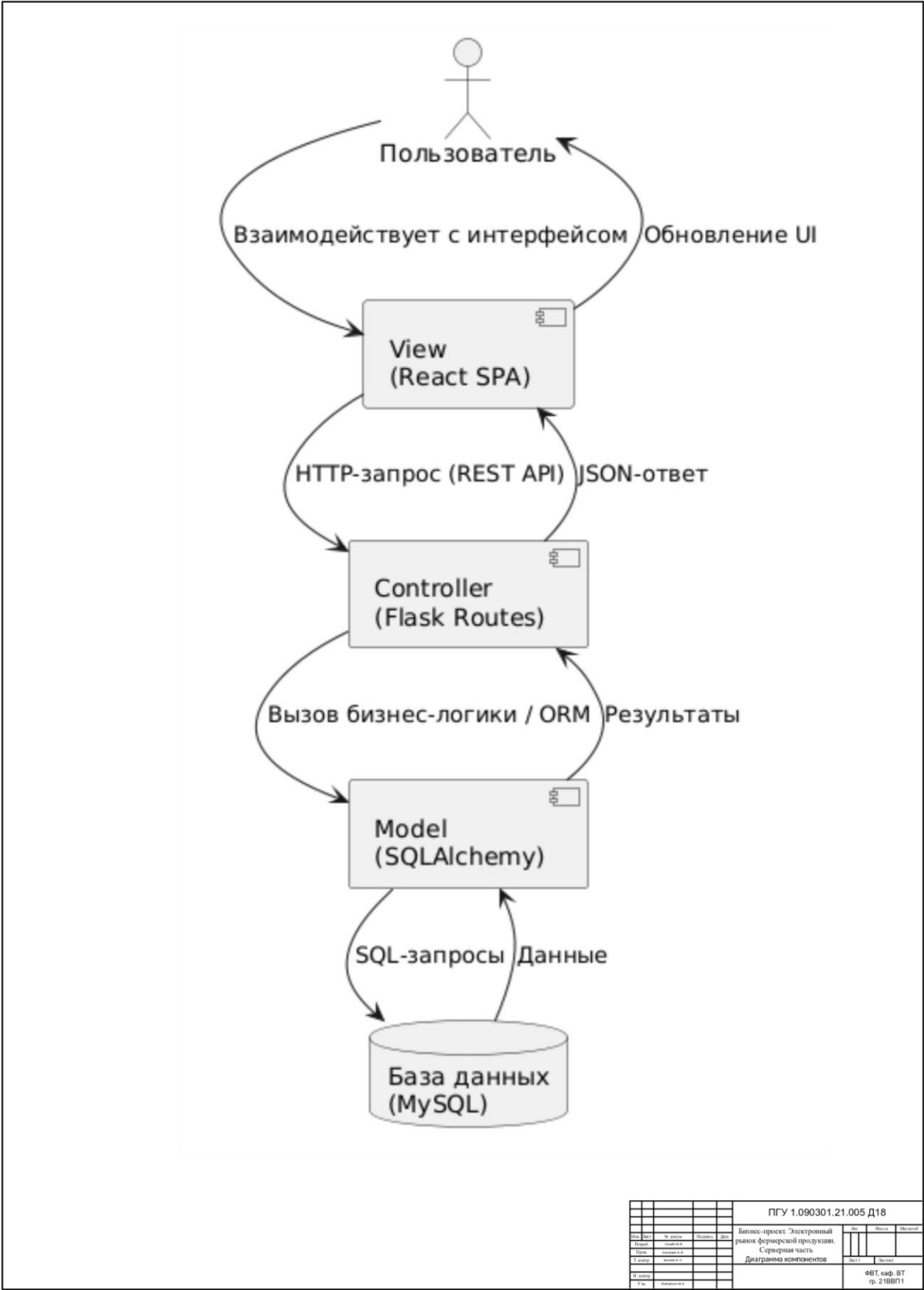
ПГУ 1.090301.21.003 Д13									
Изм.	Лист	№ докум.	Подпись	Дата	Изм.	Лист	№ докум.	Подпись	Дата
Бизнес-проект. Электронный рынок фармацевтической продукции. Сервисная часть. Диаграмма прецедентов					487 с.вб. БТ п. 218БТ11				

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.3-Диаграмма развёртывания



Приложение А.4-Диаграмма компонентов



Приложение Б

(Обязательное)

Бизнес-проект. Электронный рынок фермерских продуктов. Серверная часть
Листинг программы

					ПГУ1.090301.21.001 ПЗ	Лист
						58
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «app.py»

```
import os

from datetime import timedelta
from dotenv import load_dotenv
from flask import Flask
from flask_cors import CORS
from flasgger import Swagger
from flask_jwt_extended import JWTManager

from db import db # импорт объекта db из db.py

from routes.sellers import seller_routes
from routes.buyers import buyer_routes
from routes.categories import category_routes
from routes.reviews import review_routes
from routes.products import product_routes

app = Flask(__name__)

CORS(app)

swagger_template = {
    "swagger": "2.0",
    "info": {
        "title": "Marketplace API",
        "description": "Документация для электронного рынка
фермерской продукции",
        "version": "1.0"
    },
    "securityDefinitions": {
        "Bearer": {
            "type": "apiKey",
            "name": "Authorization",
            "in": "header",
            "description": "Введите JWT токен в формате:
**Bearer <ваш_токен>*"
        }
    }
}

swagger = Swagger(app, template=swagger_template)

load_dotenv()
user = os.getenv("DB_USER")
password = os.getenv("DB_PASSWORD")
host = os.getenv("DB_HOST")
dbname = os.getenv("DB_NAME")

jwt_key = os.getenv("JWT_KEY")
```

```

app.config['JWT_ACCESS_TOKEN_EXPIRES'] = timedelta(hours=1)

app.config['SQLALCHEMY_DATABASE_URI'] =
f'mysql+pymysql://{user}:{password}@{host}/{dbname}'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.config['JWT_SECRET_KEY'] = jwt_key

jwt = JWTManager(app)

# Инициализация объекта db с приложением Flask
db.init_app(app)

# Регистрируем Blueprint
app.register_blueprint(seller_routes, url_prefix='/api/sellers')
app.register_blueprint(buyer_routes, url_prefix='/api/buyers')
app.register_blueprint(category_routes,
url_prefix='/api/categories')
app.register_blueprint(review_routes, url_prefix='/api/reviews')
app.register_blueprint(product_routes,
url_prefix='/api/products')

# Создание таблиц (один раз, например, при запуске)
with app.app_context():
    db.create_all()

if __name__ == '__main__':
    app.run(debug=True, port=5000, host="127.0.0.1")

```

Файл «db.py»

```

from flask_sqlalchemy import SQLAlchemy

# Инициализация объекта SQLAlchemy
db = SQLAlchemy()

```

Файл «models.py»

```

from sqlalchemy import CheckConstraint

from db import db
from datetime import datetime

class Buyer(db.Model):
    __tablename__ = 'buyers'

    id = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)

```

```

        email = db.Column(db.String(100), unique=True,
nullable=False)
        phone = db.Column(db.String(20), unique=True,
nullable=False)
        password_hash = db.Column(db.String(255), nullable=False)

        reviews = db.relationship('Review', backref='buyer',
cascade="all, delete", lazy=True)

class Seller(db.Model):
    __tablename__ = 'sellers'

    id = db.Column(db.Integer, primary_key=True)
    first_name = db.Column(db.String(100), nullable=False)
    last_name = db.Column(db.String(100), nullable=False)
    production_name = db.Column(db.String(150), nullable=False)
    email = db.Column(db.String(100), unique=True,
nullable=False)
    phone = db.Column(db.String(20), unique=True,
nullable=False)
    password_hash = db.Column(db.String(255), nullable=False)
    main_image_url = db.Column(db.String(255))
    description = db.Column(db.Text)
    rating = db.Column(db.Float, default=0)

    addresses = db.relationship('SellerAddress',
backref='seller', cascade="all, delete", lazy=True)
    products = db.relationship('Product', backref='seller',
cascade="all, delete", lazy=True)
    reviews = db.relationship('Review', backref='seller',
cascade="all, delete", lazy=True)

class SellerAddress(db.Model):
    __tablename__ = 'selleraddresses'

    id = db.Column(db.Integer, primary_key=True)
    seller_id = db.Column(db.Integer,
db.ForeignKey('sellers.id', ondelete='CASCADE'), nullable=False)
    address = db.Column(db.Text, nullable=False)
    latitude = db.Column(db.Numeric(9, 6), nullable=False) #
Широта
    longitude = db.Column(db.Numeric(9, 6), nullable=False) #
Долгота
    working_hours = db.Column(db.String(100))

class Category(db.Model):
    __tablename__ = 'categories'

```

```

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    image_url = db.Column(db.String(255), nullable=False)

    products = db.relationship('Product', backref='category',
                               lazy=True)

class Product(db.Model):
    __tablename__ = 'products'

    id = db.Column(db.Integer, primary_key=True)
    seller_id = db.Column(db.Integer,
db.ForeignKey('sellers.id', ondelete='CASCADE'), nullable=False)
    name = db.Column(db.String(150), nullable=False)
    main_image_url = db.Column(db.String(255))
    category_id = db.Column(db.Integer,
db.ForeignKey('categories.id'))
    price = db.Column(db.Numeric(10, 2), nullable=False)
    quantity_unit = db.Column(db.Enum('kg', 'units', 'liter'),
default='units')
    unit_count = db.Column(db.Integer, default=1)
    description = db.Column(db.Text)

class Review(db.Model):
    __tablename__ = 'reviews'

    id = db.Column(db.Integer, primary_key=True)
    seller_id = db.Column(db.Integer,
db.ForeignKey('sellers.id', ondelete='CASCADE'), nullable=False)
    buyer_id = db.Column(db.Integer, db.ForeignKey('buyers.id',
ondelete='CASCADE'), nullable=False)
    buyer_first_name = db.Column(db.String(100), nullable=False)
    rating = db.Column(db.Integer)
    comment = db.Column(db.Text)
    created_at = db.Column(db.DateTime, default=datetime.utcnow)

    __table_args__ = (
        db.UniqueConstraint('buyer_id', 'seller_id',
name='unique_pair'),
        CheckConstraint('rating BETWEEN 1 AND 5',
name='check_rating_range'),
    )

```

Файл «buyers.py»

```

flask import Blueprint, request, jsonify
from models import db, Buyer
from werkzeug.security import generate_password_hash,
check_password_hash

```

```

from flask_jwt_extended import create_access_token,
jwt_required, get_jwt_identity, get_jwt

buyer_routes = Blueprint('buyers', __name__)

@buyer_routes.route('/register', methods=['POST'])
def register_buyer():
    """
    Регистрация покупателя (с автоматическим входом)
    ---
    tags:
      - Покупатели
    parameters:
      - in: body
        name: buyer
        description: Данные для регистрации покупателя
        required: true
        schema:
          type: object
          required:
            - first_name
            - last_name
            - email
            - phone
            - password
          properties:
            first_name:
              type: string
              example: Иван
            last_name:
              type: string
              example: Иванов
            email:
              type: string
              format: email
              example: ivan@example.com
            phone:
              type: string
              example: "+79998887766"
            password:
              type: string
              example: secret123
    responses:
      201:
        description: Успешная регистрация и вход
        schema:
          type: object
          properties:
            status:
              type: string

```

```

        example: success
    message:
        type: string
        example: Успешная регистрация
    access_token:
        type: string
        example: eyJ0eXAiOiJKV1QiLCJh...
400:
    description: Ошибка валидации данных
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Такой email уже зарегистрирован
"""
data = request.get_json()
required_fields = ['first_name', 'last_name', 'email',
'phone', 'password']

    if not all(field in data and data[field].strip() for field
in required_fields):
        return jsonify({'status': 'error', 'message': 'Не все
данные получены'}), 400

    first_name = data['first_name']
    last_name = data['last_name']
    email = data['email']
    phone = data['phone']
    password_hash = generate_password_hash(data['password'])

    if Buyer.query.filter_by(email=email).first():
        return jsonify({'status': 'error', 'message': 'Такой
email уже зарегистрирован'}), 400

    if Buyer.query.filter_by(phone=phone).first():
        return jsonify({'status': 'error', 'message': 'Такой
номер телефона уже зарегистрирован'}), 400

    new_buyer = Buyer(
        first_name=first_name,
        last_name=last_name,
        email=email,
        phone=phone,
        password_hash=password_hash
    )
    db.session.add(new_buyer)
    db.session.commit()

```

```

# Автоматическая авторизация
access_token = create_access_token(
    identity=str(new_buyer.id),
    additional_claims={"user_type": "buyer"}
)

return jsonify({
    'status': 'success',
    'message': 'Успешная регистрация',
    'access_token': access_token
}), 201

@buyer_routes.route('/login', methods=['POST'])
def login_buyer():
    """
    Вход покупателя
    ---
    tags:
      - Покупатели
    parameters:
      - in: body
        name: credentials
        description: Учетные данные покупателя
        required: true
        schema:
          type: object
          required:
            - email
            - password
          properties:
            email:
              type: string
              format: email
              example: ivan@example.com
            password:
              type: string
              example: secret123
    responses:
      200:
        description: Успешный вход
        schema:
          type: object
          properties:
            status:
              type: string
            message:
              type: string
            buyer_id:
              type: integer

```

					ПГУ1.090301.21.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

400:
  description: Неверные данные
  schema:
    type: object
    properties:
      status:
        type: string
        example: error
      message:
        type: string
        example: Не указан email или пароль
401:
  description: Неверный пароль
  schema:
    type: object
    properties:
      status:
        type: string
        example: error
      message:
        type: string
        example: Неверный пароль
"""
data = request.get_json()
email = data.get('email')
password = data.get('password')

if not email or not password:
    return jsonify({'status': 'error', 'message': 'Не указан
email или пароль'}), 400

buyer = Buyer.query.filter_by(email=email).first()
if not buyer:
    return jsonify({'status': 'error', 'message': 'Неверный
email'}), 401

if not check_password_hash(buyer.password_hash, password):
    return jsonify({'status': 'error', 'message': 'Неверный
пароль'}), 401

access_token = create_access_token(
    identity=str(buyer.id),
    additional_claims={"user_type": "buyer"}
)

return jsonify({
    'status': 'success',
    'message': 'Успешный вход',
    'access_token': access_token
}), 200

```

```

@buyer_routes.route('/get_data', methods=['GET'])
@jwt_required()
def get_buyers_data():
    """
    Получение данных текущего покупателя
    ---
    tags:
      - Покупатели
    security:
      - Bearer: []
    responses:
      200:
        description: Данные покупателя успешно получены
        schema:
          type: object
          properties:
            status:
              type: string
              example: success
            buyer:
              type: object
              properties:
                id:
                  type: integer
                  example: 3
                first_name:
                  type: string
                  example: Иван
                last_name:
                  type: string
                  example: Иванов
                email:
                  type: string
                  example: ivan@example.com
                phone:
                  type: string
                  example: "+79998887766"
      404:
        description: Покупатель не найден
        schema:
          type: object
          properties:
            status:
              type: string
              example: error
            message:
              type: string
              example: Покупатель не найден
    """
    jwt_data = get_jwt()

```

```

        if jwt_data.get("user_type") != "buyer":
            return jsonify({'status': 'error', 'message': 'Доступ
только для покупателей'}), 403

        buyer_id = int(get_jwt_identity()) # Получаем ID покупателя
из токена
        buyer = Buyer.query.get(buyer_id)
        if not buyer:
            return jsonify({'status': 'error', 'message':
'Покупатель не найден'}), 404

        return jsonify({
            'status': 'success',
            'buyer': {
                'id': buyer.id,
                'first_name': buyer.first_name,
                'last_name': buyer.last_name,
                'email': buyer.email,
                'phone': buyer.phone
            }
        }), 200

```

```

@buyer_routes.route('/delete', methods=['DELETE'])
@jwt_required()
def delete_buyer():
    """
    Удаление аккаунта текущего покупателя
    ---
    tags:
        - Покупатели
    security:
        - Bearer: []
    responses:
        200:
            description: Аккаунт покупателя удалён
            schema:
                type: object
                properties:
                    status:
                        type: string
                    message:
                        type: string
        404:
            description: Покупатель не найден
            schema:
                type: object
                properties:
                    status:
                        type: string
                    example: error

```

```

        message:
            type: string
            example: Продавец не найден
    """
    jwt_data = get_jwt()
    if jwt_data.get("user_type") != "buyer":
        return jsonify({'status': 'error', 'message': 'Доступ
только для покупателей'}), 403

    buyer_id = int(get_jwt_identity())
    buyer = Buyer.query.get(buyer_id)
    if not buyer:
        return jsonify({'status': 'error', 'message':
'Покупатель не найден'}), 404

    db.session.delete(buyer)
    db.session.commit()
    return jsonify({'status': 'success', 'message': 'Аккаунт
покупателя удалён'}), 200

```

Файл «products.py»

```

from flask import Blueprint, request, jsonify
from flask_jwt_extended import jwt_required, get_jwt_identity,
get_jwt
from models import db, Product, Category, Seller
from sqlalchemy.sql import text

product_routes = Blueprint('products', __name__)

@product_routes.route('/create', methods=['POST'])
@jwt_required()
def create_product():
    """
    Добавление нового продукта продавцом
    ---
    tags:
      - Продукты
    security:
      - Bearer: []
    parameters:
      - in: body
        name: product
        description: Данные продукта
        required: true
        schema:
          type: object
          required:
            - name
            - price
    """

```

```

properties:
  name:
    type: string
    example: Яблоки
  price:
    type: number
    format: float
    example: 120.50
  category_id:
    type: integer
    example: 2
  quantity_unit:
    type: string
    enum: [kg, units, liter]
    example: kg
  unit_count:
    type: integer
    example: 5
  main_image_url:
    type: string
    example: https://example.com/image.jpg
  description:
    type: string
    example: Сочные и свежие яблоки с фермы
responses:
  201:
    description: Продукт успешно создан
    schema:
      type: object
      properties:
        status:
          type: string
          example: success
        message:
          type: string
          example: Продукт добавлен
        product_id:
          type: integer
          example: 42
  400:
    description: Ошибка валидации данных
    schema:
      type: object
      properties:
        status:
          type: string
          example: error
        message:
          type: string
          example: Название и цена обязательны

```

"""

					ПГУ1.090301.21.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

data = request.get_json()
jwt_data = get_jwt()
if jwt_data.get("user_type") != "seller":
    return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

seller_id = int(get_jwt_identity())

# Проверка обязательных полей
name = data.get('name')
price = data.get('price')
category_id = data.get('category_id')
quantity_unit = data.get('quantity_unit', 'units')
unit_count = data.get('unit_count', 1)
main_image_url = data.get('main_image_url', '')
description = data.get('description', '')

if not name or not price:
    return jsonify({'status': 'error', 'message': 'Название
и цена обязательны'}), 400

if quantity_unit not in ['kg', 'units', 'liter']:
    return jsonify({'status': 'error', 'message':
'Недопустимая единица измерения'}), 400

# Проверка на существование категории (опционально)
if category_id and not Category.query.get(category_id):
    return jsonify({'status': 'error', 'message': 'Категория
не найдена'}), 400

product = Product(
    seller_id=seller_id,
    name=name,
    price=price,
    category_id=category_id,
    quantity_unit=quantity_unit,
    unit_count=unit_count,
    main_image_url=main_image_url,
    description=description
)

db.session.add(product)
db.session.commit()

return jsonify({'status': 'success', 'message': 'Продукт
добавлен', 'product_id': product.id}), 201

@product_routes.route('/delete/<int:product_id>',
methods=['DELETE'])
@jwt_required()

```

```

def delete_product(product_id):
    """
    Удаление продукта продавцом
    ---
    tags:
      - Продукты
    security:
      - Bearer: []
    parameters:
      - in: path
        name: product_id
        required: true
        type: integer
        description: ID продукта для удаления
    responses:
      200:
        description: Продукт успешно удалён
        schema:
          type: object
          properties:
            status:
              type: string
              example: success
            message:
              type: string
              example: Продукт удалён
      403:
        description: Продавец не имеет права удалить чужой
        продукт
        schema:
          type: object
          properties:
            status:
              type: string
              example: error
            message:
              type: string
              example: Вы не можете удалить чужой продукт
      404:
        description: Продукт не найден
        schema:
          type: object
          properties:
            status:
              type: string
              example: error
            message:
              type: string
              example: Продукт не найден
    """
    jwt_data = get_jwt()

```

```

        if jwt_data.get("user_type") != "seller":
            return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

        seller_id = int(get_jwt_identity())

        product = Product.query.get(product_id)
        if not product:
            return jsonify({'status': 'error', 'message': 'Продукт
не найден'}), 404

        if product.seller_id != seller_id:
            return jsonify({'status': 'error', 'message': 'Вы не
можете удалить чужой продукт'}), 403

        db.session.delete(product)
        db.session.commit()

        return jsonify({'status': 'success', 'message': 'Продукт
удалён'}), 200

@product_routes.route('/nearby', methods=['GET'])
def get_nearby_products():
    """
    Получение списка продуктов от ближайших продавцов
    ---
    tags:
      - Продукты
    parameters:
      - in: query
        name: latitude
        required: true
        type: number
        description: Широта пользователя
        example: 55.7558
      - in: query
        name: longitude
        required: true
        type: number
        description: Долгота пользователя
        example: 37.6173
      - in: query
        name: radius
        required: true
        type: number
        description: Радиус в километрах
        example: 10
    responses:
      200:
        description: Список продуктов поблизости

```

```

schema:
  type: object
  properties:
    status:
      type: string
      example: success
    products:
      type: array
      items:
        type: object
        properties:
          id:
            type: integer
            example: 7
          name:
            type: string
            example: Картофель
          price:
            type: number
            format: float
            example: 45.0
          quantity_unit:
            type: string
            example: kg
          unit_count:
            type: integer
            example: 1
          description:
            type: string
            example: Натуральный картофель, выращенный

```

без химикатов

```

      main_image_url:
        type: string
        example: https://example.com/img.jpg
      category_id:
        type: integer
        example: 3
      seller_id:
        type: integer
        example: 4
      seller_name:
        type: string
        example: Ферма "Зелёный Сад"

```

```

400:
  description: Ошибка в параметрах
  schema:
    type: object
    properties:
      status:
        type: string
        example: error

```

```

        message:
            type: string
            example: Неверные координаты или радиус
    """
    try:
        user_lat = float(request.args.get('latitude'))
        user_lon = float(request.args.get('longitude'))
        radius_km = float(request.args.get('radius'))
    except (TypeError, ValueError):
        return jsonify({'status': 'error', 'message': 'Неверные
координаты или радиус'}), 400

    # Расчёт расстояния по формуле haversine (в километрах)
    haversine_sql = text(f"""
        SELECT sa.seller_id
        FROM selleraddresses sa
        WHERE 6371 * 2 * ASIN(SQRT(
            POWER(SIN(RADIANS(sa.latitude - :lat) / 2), 2) +
            COS(RADIANS(:lat)) * COS(RADIANS(sa.latitude)) *
            POWER(SIN(RADIANS(sa.longitude - :lon) / 2), 2)
        )) <= :radius
    """)

    seller_ids = db.session.execute(haversine_sql, {
        'lat': user_lat,
        'lon': user_lon,
        'radius': radius_km
    }).fetchall()

    seller_ids = [row[0] for row in seller_ids]

    if not seller_ids:
        return jsonify({'status': 'success', 'products': []}),
200

    products =
    Product.query.filter(Product.seller_id.in_(seller_ids)).all()

    results = []
    for product in products:
        results.append({
            'id': product.id,
            'name': product.name,
            'price': float(product.price),
            'quantity_unit': product.quantity_unit,
            'main_image_url': product.main_image_url,
            'seller_id': product.seller_id,
            'category_id': product.category_id,
            'unit_count': product.unit_count,
            'description': product.description,

```

```

        'seller_name': product.seller.production_name if
product.seller else None
    })

    return jsonify({'status': 'success', 'products': results}),
200

@product_routes.route('/seller/<int:seller_id>',
methods=['GET'])
def get_products_by_seller(seller_id):
    """
    Получение всех товаров по ID продавца
    ---
    tags:
      - Продукты
    parameters:
      - in: path
        name: seller_id
        required: true
        type: integer
        description: ID продавца
    responses:
      200:
        description: Список продуктов продавца
        schema:
          type: object
          properties:
            status:
              type: string
              example: success
            products:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: integer
                    example: 1
                  name:
                    type: string
                    example: Морковь
                  price:
                    type: number
                    format: float
                    example: 30.5
                  quantity_unit:
                    type: string
                    example: kg
                  unit_count:
                    type: integer

```

					ПГУ1.090301.21.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

```

        example: 1
        description:
            type: string
            example: Сочная морковь, выращенная без
пестицидов
        main_image_url:
            type: string
            example: https://example.com/image.jpg
        category_id:
            type: integer
            example: 2
        seller_id:
            type: integer
            example: 4
        seller_name:
            type: string
            example: Ферма "Солнечный край"
404:
    description: Продавец не найден
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Продавец не найден
"""
seller = Seller.query.get(seller_id)
if not seller:
    return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

products =
Product.query.filter_by(seller_id=seller_id).all()

results = []
for product in products:
    results.append({
        'id': product.id,
        'name': product.name,
        'price': float(product.price),
        'quantity_unit': product.quantity_unit,
        'unit_count': product.unit_count,
        'description': product.description,
        'main_image_url': product.main_image_url,
        'category_id': product.category_id,
        'seller_id': product.seller_id,
        'seller_name': seller.production_name
    })

```

```
    return jsonify({'status': 'success', 'products': results}),
200
```

Файл «reviews.py»

```
from flask import Blueprint, request, jsonify
from models import db, Review, Buyer, Seller
from flask_jwt_extended import jwt_required, get_jwt_identity,
get_jwt
from datetime import datetime
```

```
review_routes = Blueprint('reviews', __name__)
```

```
@review_routes.route('/create', methods=['POST'])
@jwt_required()
def create_review():
    """
    Создание отзыва покупателем
    ---
    tags:
      - ОТЗЫВЫ
    security:
      - Bearer: []
    parameters:
      - in: body
        name: review
        description: Данные отзыва
        required: true
        schema:
          type: object
          required:
            - seller_id
            - rating
          properties:
            seller_id:
              type: integer
              example: 1
            rating:
              type: integer
              example: 4
            comment:
              type: string
              example: Хорошее качество, быстрая доставка
    responses:
      201:
        description: Отзыв успешно создан
        schema:
          type: object
          properties:
```

					ПГУ1.090301.21.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

```

        status:
            type: string
            example: success
        message:
            type: string
            example: Отзыв создан
        review_id:
            type: integer
            example: 42
400:
    description: Ошибка валидации данных или повторный отзыв
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Рейтинг обязателен
404:
    description: Продавец не найден
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Продавец не найден
"""
jwt_data = get_jwt()
if jwt_data.get("user_type") != "buyer":
    return jsonify({'status': 'error', 'message': 'Доступ
только для покупателей'}), 403

data = request.get_json()
buyer_id = int(get_jwt_identity())

seller_id = data.get('seller_id')
rating = data.get('rating')
comment = data.get('comment', '').strip()

if not seller_id:
    return jsonify({'status': 'error', 'message': 'ID
продавца обязателен'}), 400

if not rating:
    return jsonify({'status': 'error', 'message': 'Рейтинг
обязателен'}), 400

```

```

        if not (1 <= int(rating) <= 5):
            return jsonify({'status': 'error', 'message': 'Оценка
должна быть от 1 до 5'}), 400

        seller = Seller.query.get(seller_id)
        if not seller:
            return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

        existing_review =
Review.query.filter_by(seller_id=seller_id,
buyer_id=buyer_id).first()
        if existing_review:
            return jsonify({'status': 'error', 'message': 'Вы уже
оставили отзыв этому продавцу'}), 404

        buyer = Buyer.query.get(buyer_id)
        if not buyer:
            return jsonify({'status': 'error', 'message':
'Покупатель не найден'}), 404

        review = Review(
            seller_id=seller_id,
            buyer_id=buyer_id,
            buyer_first_name=buyer.first_name,
            rating=rating,
            comment=comment,
            created_at=datetime.utcnow()
        )

        db.session.add(review)
        db.session.commit()

        recalculate_seller_rating(seller_id)

        return jsonify({'status': 'success', 'message': 'Отзыв
создан', 'review_id': review.id}), 201

@review_routes.route('/delete/<int:review_id>',
methods=['DELETE'])
@jwt_required()
def delete_review(review_id):
    """
    Удаление отзыва покупателем
    ---
    tags:
      - ОТЗЫВЫ
    security:
      - Bearer: []

```

```

parameters:
  - in: path
    name: review_id
    required: true
    type: integer
    description: ID отзыва для удаления
responses:
  200:
    description: Отзыв успешно удалён
    schema:
      type: object
      properties:
        status:
          type: string
          example: success
        message:
          type: string
          example: Отзыв удалён
  403:
    description: Отказ в доступе – отзыв принадлежит другому
покупателю
    schema:
      type: object
      properties:
        status:
          type: string
          example: error
        message:
          type: string
          example: Вы не можете удалить чужой отзыв
  404:
    description: Отзыв не найден
    schema:
      type: object
      properties:
        status:
          type: string
          example: error
        message:
          type: string
          example: Отзыв не найден
"""
jwt_data = get_jwt()
if jwt_data.get("user_type") != "buyer":
    return jsonify({'status': 'error', 'message': 'Доступ
только для покупателей'}), 403

buyer_id = int(get_jwt_identity())
review = Review.query.get(review_id)

if not review:

```

```

        return jsonify({'status': 'error', 'message': 'Отзыв не
найден'}), 404

    if review.buyer_id != buyer_id:
        return jsonify({'status': 'error', 'message': 'Вы не
можете удалить чужой отзыв'}), 403

    seller_id = review.seller_id

    db.session.delete(review)
    db.session.commit()

    recalculate_seller_rating(seller_id)

    return jsonify({'status': 'success', 'message': 'Отзыв
удалён'}), 200

@review_routes.route('/seller/<int:seller_id>', methods=['GET'])
def get_reviews_by_seller(seller_id):
    """
    Получение всех отзывов о продавце по его ID
    ---
    tags:
      - Отзывы
    parameters:
      - in: path
        name: seller_id
        required: true
        type: integer
        description: ID продавца
    responses:
      200:
        description: Список отзывов о продавце
        schema:
          type: object
          properties:
            status:
              type: string
            reviews:
              type: array
              items:
                type: object
                properties:
                  id:
                    type: integer
                    example: 10
                  buyer_id:
                    type: integer
                    example: 5
                  buyer_first_name:

```

```

        type: string
        example: Алексей
    rating:
        type: integer
        example: 4
    comment:
        type: string
        example: Отличные овощи, быстро доставили!
    created_at:
        type: string
        format: date-time
        example: "2025-05-24T14:12:33"

404:
    description: Продавец не найден
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Продавец не найден
"""
seller = Seller.query.get(seller_id)
if not seller:
    return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

reviews =
Review.query.filter_by(seller_id=seller_id).order_by(Review.crea
ted_at.desc()).all()

reviews_list = []
for r in reviews:
    buyer = Buyer.query.get(r.buyer_id)
    reviews_list.append({
        'id': r.id,
        'buyer_id': r.buyer_id,
        'buyer_first_name': buyer.first_name if buyer else
None,
        'rating': r.rating,
        'comment': r.comment,
        'created_at': r.created_at.isoformat()
    })

    return jsonify({'status': 'success', 'reviews':
reviews_list}), 200

def recalculate_seller_rating(seller_id):

```

```

        """Пересчитывает средний рейтинг продавца и сохраняет
его."""
        from sqlalchemy import func

        avg_rating = db.session.query(func.avg(Review.rating))\
            .filter(Review.seller_id == seller_id).scalar()

        seller = Seller.query.get(seller_id)
        seller.rating = round(avg_rating or 0, 2)

        db.session.commit()

```

Файл «sellers.py»

```

from flask import request, jsonify, Blueprint
from models import db, Seller, SellerAddress
from werkzeug.security import generate_password_hash,
check_password_hash
from flask_jwt_extended import create_access_token,
jwt_required, get_jwt_identity, get_jwt
from sqlalchemy.sql import text

seller_routes = Blueprint('sellers', __name__)

@seller_routes.route('/register', methods=['POST'])
def register_seller():
    """
    Регистрация продавца (с автоматическим входом)
    ---
    tags:
      - Продавцы
    parameters:
      - in: body
        name: seller
        description: Данные для регистрации продавца
        required: true
        schema:
          type: object
          required:
            - first_name
            - last_name
            - production_name
            - email
            - phone
            - password
          properties:
            first_name:
              type: string
              example: Иван
            last_name:

```

```

        type: string
        example: Иванов
    production_name:
        type: string
        example: Ферма "Родные поля"
    email:
        type: string
        format: email
        example: farm@example.com
    phone:
        type: string
        example: "+79995554433"
    password:
        type: string
        example: strongpassword
responses:
    201:
        description: Успешная регистрация и вход
        schema:
            type: object
            properties:
                status:
                    type: string
                    example: success
                message:
                    type: string
                    example: Успешная регистрация
                access_token:
                    type: string
                    example: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1...
    400:
        description: Ошибка валидации или дубликаты
        schema:
            type: object
            properties:
                status:
                    type: string
                    example: error
                message:
                    type: string
                    example: Такой email уже зарегистрирован
"""
data = request.get_json()
required_fields = ['first_name', 'last_name',
'production_name', 'email', 'phone', 'password']

if not all(field in data and data[field].strip() for field
in required_fields):
    return jsonify({'status': 'error', 'message': 'Не все
данные получены'}), 400

```

```

        if
Seller.query.filter_by(production_name=data['production_name']).
first():
            return jsonify({'status': 'error', 'message': 'Такое имя
уже зарегистрировано'}), 400

        if Seller.query.filter_by(email=data['email']).first():
            return jsonify({'status': 'error', 'message': 'Такой
email уже зарегистрирован'}), 400

        if Seller.query.filter_by(phone=data['phone']).first():
            return jsonify({'status': 'error', 'message': 'Такой
номер телефона уже зарегистрирован'}), 400

new_seller = Seller(
    first_name=data['first_name'].strip(),
    last_name=data['last_name'].strip(),
    production_name=data['production_name'].strip(),
    email=data['email'].strip(),
    phone=data['phone'].strip(),
    password_hash=generate_password_hash(data['password']),
    description='',
    rating=0.0
)
db.session.add(new_seller)
db.session.commit()

# Автоматическая авторизация
access_token = create_access_token(
    identity=str(new_seller.id),
    additional_claims={"user_type": "seller"}
)

return jsonify({
    'status': 'success',
    'message': 'Успешная регистрация',
    'access_token': access_token
}), 201

@seller_routes.route('/login', methods=['POST'])
def login_seller():
    """
        Вход продавца
        ---
        tags:
            - Продавцы
        parameters:
            - in: body
              name: credentials
              description: Учетные данные продавца
    """

```

```

    required: true
    schema:
      type: object
      required:
        - email
        - password
      properties:
        email:
          type: string
          format: email
          example: farm@example.com
        password:
          type: string
          example: strongpassword
  responses:
    200:
      description: Успешный вход
      schema:
        type: object
        properties:
          status:
            type: string
            example: success
          message:
            type: string
            example: Успешный вход
          access_token:
            type: string
            example: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1...
    400:
      description: Неверные данные
      schema:
        type: object
        properties:
          status:
            type: string
            example: error
          message:
            type: string
            example: Не указан email или пароль
    401:
      description: Неверный пароль
      schema:
        type: object
        properties:
          status:
            type: string
            example: error
          message:
            type: string
            example: Неверный пароль

```

```

        """
data = request.get_json()
email = data.get('email')
password = data.get('password')

if not email or not password:
    return jsonify({'status': 'error', 'message': 'Не указан
email или пароль'}), 400

seller = Seller.query.filter_by(email=email).first()
if not seller:
    return jsonify({'status': 'error', 'message': 'Неверный
email'}), 401

if not check_password_hash(seller.password_hash, password):
    return jsonify({'status': 'error', 'message': 'Неверный
пароль'}), 401

access_token = create_access_token(
    identity=str(seller.id),
    additional_claims={"user_type": "seller"}
)

return jsonify({
    'status': 'success',
    'message': 'Успешный вход',
    'access_token': access_token
}), 200

@seller_routes.route('/get_data', methods=['GET'])
@jwt_required()
def get_seller_data():
    """
    Получение данных текущего продавца
    ---
    tags:
      - Продавцы
    security:
      - Bearer: []
    responses:
      200:
        description: Данные продавца
        schema:
          type: object
          properties:
            status:
              type: string
              example: success
            seller:
              type: object

```

```

properties:
  id:
    type: integer
    example: 3
  first_name:
    type: string
    example: Иван
  last_name:
    type: string
    example: Иванов
  production_name:
    type: string
    example: Ферма "Родные поля"
  email:
    type: string
    example: farm@example.com
  phone:
    type: string
    example: "+79995554433"
  main_image_url:
    type: string
    example: https://example.com/img.jpg
  description:
    type: string
    example: Продаём только свежее
  rating:
    type: number
    format: float
    example: 4.67
404:
  description: Продавец не найден
  schema:
    type: object
    properties:
      status:
        type: string
        example: error
      message:
        type: string
        example: Продавец не найден
"""
jwt_data = get_jwt()
if jwt_data.get("user_type") != "seller":
    return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

seller_id = int(get_jwt_identity())
seller = Seller.query.get(seller_id)

if not seller:

```

```
        return jsonify({'status': 'error', 'message': 'Продавец  
не найден'}), 404
```

```
    return jsonify({  
        'status': 'success',  
        'seller': {  
            'id': seller.id,  
            'first_name': seller.first_name,  
            'last_name': seller.last_name,  
            'production_name': seller.production_name,  
            'email': seller.email,  
            'phone': seller.phone,  
            'main_image_url': seller.main_image_url,  
            'description': seller.description,  
            'rating': seller.rating  
        }  
    }), 200
```

```
@seller_routes.route('/get_public_data/<int:seller_id>',  
methods=['GET'])
```

```
def get_public_seller_data(seller_id):
```

```
    """
```

```
    Получение публичных данных продавца по ID
```

```
    ---
```

```
    tags:
```

```
        - Продавцы
```

```
    parameters:
```

```
        - name: seller_id
```

```
            in: path
```

```
            required: true
```

```
            type: integer
```

```
            description: ID продавца
```

```
    responses:
```

```
        200:
```

```
            description: Публичные данные продавца
```

```
            schema:
```

```
                type: object
```

```
                properties:
```

```
                    status:
```

```
                        type: string
```

```
                        example: success
```

```
                    seller:
```

```
                        type: object
```

```
                        properties:
```

```
                            id:
```

```
                                type: integer
```

```
                                example: 3
```

```
                            production_name:
```

```
                                type: string
```

```
                                example: Ферма "Родные поля"
```

					ПГУ1.090301.21.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

        main_image_url:
            type: string
            example: https://example.com/img.jpg
        description:
            type: string
            example: Продаём только свежее
        rating:
            type: number
            format: float
            example: 4.67
    404:
        description: Продавец не найден
        schema:
            type: object
            properties:
                status:
                    type: string
                    example: error
                message:
                    type: string
                    example: Продавец не найден
    """
    seller = Seller.query.get(seller_id)

    if not seller:
        return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

    return jsonify({
        'status': 'success',
        'seller': {
            'id': seller.id,
            'production_name': seller.production_name,
            'main_image_url': seller.main_image_url,
            'description': seller.description,
            'rating': seller.rating
        }
    }), 200

@seller_routes.route('/update_description', methods=['PATCH'])
@jwt_required()
def update_seller_description():
    """
    Обновление описания продавца
    ---
    tags:
        - Продавцы
    security:
        - Bearer: []
    parameters:

```

```

- in: body
  name: description
  description: Новое описание продавца
  required: true
  schema:
    type: object
    required:
      - description
    properties:
      description:
        type: string
        example: Продаю свежие фермерские продукты каждый
день
responses:
  200:
    description: Описание обновлено
    schema:
      type: object
      properties:
        status:
          type: string
          example: success
        message:
          type: string
          example: Описание обновлено
  400:
    description: Неверные данные
    schema:
      type: object
      properties:
        status:
          type: string
          example: error
        message:
          type: string
          example: Неверные данные
  404:
    description: Продавец не найден
    schema:
      type: object
      properties:
        status:
          type: string
          example: error
        message:
          type: string
          example: Продавец не найден
"""
data = request.get_json()
description = data.get('description', '').strip()
if not description:

```

```

        return jsonify({'status': 'error', 'message': 'Описание
не должно быть пустым'}), 400

    jwt_data = get_jwt()
    if jwt_data.get("user_type") != "seller":
        return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

    seller_id = int(get_jwt_identity())
    seller = Seller.query.get(seller_id)
    if not seller:
        return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

    seller.description = description
    db.session.commit()

    return jsonify({'status': 'success', 'message': 'Описание
обновлено'}), 200

@seller_routes.route('/update_main_image', methods=['PATCH'])
@jwt_required()
def update_seller_main_image():
    """
    Обновление главного изображения продавца
    ---
    tags:
      - Продавцы
    security:
      - Bearer: []
    parameters:
      - in: body
        name: image_url
        description: URL главного изображения продавца
        required: true
        schema:
          type: object
          required:
            - image_url
          properties:
            image_url:
              type: string
              format: uri
              example: https://example.com/images/seller123.jpg
    responses:
      200:
        description: Описание обновлено
        schema:
          type: object
          properties:

```

```

        status:
            type: string
            example: success
        message:
            type: string
            example: Описание обновлено
400:
    description: Неверные данные
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Некорректный URL
404:
    description: Продавец не найден
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Продавец не найден
"""

data = request.get_json()
image_url = data.get('image_url', '').strip()
if not image_url or not image_url.startswith(('http://',
'https://')):
    return jsonify({'status': 'error', 'message':
'Некорректный URL'}), 400

jwt_data = get_jwt()
if jwt_data.get("user_type") != "seller":
    return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

seller_id = int(get_jwt_identity())
seller = Seller.query.get(seller_id)
if not seller:
    return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

seller.main_image_url = image_url
db.session.commit()

```

```

        return jsonify({'status': 'success', 'message': 'Главное
изображение обновлено'}), 200

@seller_routes.route('/delete', methods=['DELETE'])
@jwt_required()
def delete_seller():
    """
    Удаление аккаунта продавца
    ---
    tags:
      - Продавцы
    security:
      - Bearer: []
    responses:
      200:
        description: Аккаунт продавца удалён
        schema:
          type: object
          properties:
            status:
              type: string
              example: success
            message:
              type: string
              example: Аккаунт продавца удалён
      404:
        description: Продавец не найден
        schema:
          type: object
          properties:
            status:
              type: string
              example: error
            message:
              type: string
              example: Продавец не найден
    """
    jwt_data = get_jwt()
    if jwt_data.get("user_type") != "seller":
        return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

    seller_id = int(get_jwt_identity())
    seller = Seller.query.get(seller_id)
    if not seller:
        return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

    db.session.delete(seller)
    db.session.commit()

```

```
    return jsonify({'status': 'success', 'message': 'Аккаунт  
продавца удалён'}), 200
```

```
@seller_routes.route('/add_address', methods=['POST'])
```

```
@jwt_required()
```

```
def add_seller_address():
```

```
    """
```

```
    Добавление адреса продавца
```

```
    ---
```

```
    tags:
```

```
        - Адреса
```

```
    security:
```

```
        - Bearer: []
```

```
    parameters:
```

```
        - in: body
```

```
            name: address
```

```
            required: true
```

```
            schema:
```

```
                type: object
```

```
                required:
```

```
                    - address
```

```
                    - latitude
```

```
                    - longitude
```

```
                properties:
```

```
                    address:
```

```
                        type: string
```

```
                        example: ул. Ленина, 10, село Зеленое
```

```
                    latitude:
```

```
                        type: number
```

```
                        format: float
```

```
                        example: 55.755826
```

```
                    longitude:
```

```
                        type: number
```

```
                        format: float
```

```
                        example: 37.6173
```

```
                    working_hours:
```

```
                        type: string
```

```
                        example: Пн-Пт 9:00-18:00
```

```
    responses:
```

```
        201:
```

```
            description: Адрес добавлен
```

```
            schema:
```

```
                type: object
```

```
                properties:
```

```
                    status:
```

```
                        type: string
```

```
                        example: success
```

```
                    message:
```

```
                        type: string
```

					ПГУ1.090301.21.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		

```

        example: Адрес добавлен
400:
    description: Все обязательные поля должны быть заполнены
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Все обязательные поля должны быть
заполнены
        """
    data = request.get_json()
    jwt_data = get_jwt()
    if jwt_data.get("user_type") != "seller":
        return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

    seller_id = int(get_jwt_identity())

    address = data.get('address')
    latitude = data.get('latitude')
    longitude = data.get('longitude')
    working_hours = data.get('working_hours', '')

    if not all([address, latitude, longitude]):
        return jsonify({'status': 'error', 'message': 'Все
обязательные поля должны быть заполнены'}), 400

    new_address = SellerAddress(
        seller_id=seller_id,
        address=address,
        latitude=latitude,
        longitude=longitude,
        working_hours=working_hours
    )

    db.session.add(new_address)
    db.session.commit()

    return jsonify({'status': 'success', 'message': 'Адрес
добавлен'}), 201

@seller_routes.route('/delete_address/<int:address_id>',
methods=['DELETE'])
@jwt_required()
def delete_seller_address(address_id):
    """

```

```

Удаление адреса продавца
---
tags:
  - Адреса
security:
  - Bearer: []
parameters:
  - name: address_id
    in: path
    required: true
    type: integer
    description: ID адреса продавца
responses:
  200:
    description: Адрес успешно удалён
    schema:
      type: object
      properties:
        status:
          type: string
          example: success
        message:
          type: string
          example: Адрес удалён
  403:
    description: Нет прав на удаление этого адреса
    schema:
      type: object
      properties:
        status:
          type: string
          example: error
        message:
          type: string
          example: У вас нет прав на удаление этого адреса
  404:
    description: Адрес не найден
    schema:
      type: object
      properties:
        status:
          type: string
          example: error
        message:
          type: string
          example: Адрес не найден
"""
jwt_data = get_jwt()
if jwt_data.get("user_type") != "seller":
    return jsonify({'status': 'error', 'message': 'Доступ
только для продавцов'}), 403

```

```

seller_id = int(get_jwt_identity())
address = SellerAddress.query.get(address_id)

if not address:
    return jsonify({'status': 'error', 'message': 'Адрес не
найден'}), 404

if address.seller_id != seller_id:
    return jsonify({'status': 'error', 'message': 'У вас нет
прав на удаление этого адреса'}), 403

db.session.delete(address)
db.session.commit()

return jsonify({'status': 'success', 'message': 'Адрес
удалён'}), 200

@seller_routes.route('/addresses/<int:seller_id>',
methods=['GET'])
def get_seller_addresses(seller_id):
    """
    Получение всех адресов продавца по его ID
    ---
    tags:
    - Адреса
    parameters:
    - in: path
      name: seller_id
      required: true
      schema:
        type: integer
      description: ID продавца
    responses:
    200:
      description: Список адресов
      schema:
        type: object
        properties:
          status:
            type: string
            example: success
          addresses:
            type: array
            items:
              type: object
              properties:
                id:
                  type: integer
                  example: 12

```

```

        address:
            type: string
            example: "г. Москва, ул. Ленина, д. 10"
        latitude:
            type: number
            format: float
            example: 55.7558
        longitude:
            type: number
            format: float
            example: 37.6173
        working_hours:
            type: string
            example: "Пн-Пт 09:00-18:00, Сб 10:00-14:00"
404:
    description: Продавец не найден или у него нет адресов
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Продавец не найден
"""
seller = Seller.query.get(seller_id)
if not seller:
    return jsonify({'status': 'error', 'message': 'Продавец
не найден'}), 404

addresses =
SellerAddress.query.filter_by(seller_id=seller_id).all()

address_list = [{
    'id': addr.id,
    'address': addr.address,
    'latitude': float(addr.latitude),
    'longitude': float(addr.longitude),
    'working_hours': addr.working_hours
} for addr in addresses]

return jsonify({'status': 'success', 'addresses':
address_list}), 200

@seller_routes.route('/nearby', methods=['GET'])
def get_nearby_sellers_by_addresses():
    """
    Получение ближайших продавцов (каждый адрес – отдельный
элемент)

```

```

---
tags:
  - Продавцы
parameters:
  - in: query
    name: latitude
    required: true
    type: number
    description: Широта пользователя
    example: 55.7558
  - in: query
    name: longitude
    required: true
    type: number
    description: Долгота пользователя
    example: 37.6173
  - in: query
    name: radius
    required: true
    type: number
    description: Радиус в километрах
    example: 10
responses:
  200:
    description: Список ближайших продавцов с адресами
    schema:
      type: object
      properties:
        status:
          type: string
          example: success
        sellers:
          type: array
          items:
            type: object
            properties:
              seller_id:
                type: integer
                example: 5
              production_name:
                type: string
                example: Ферма "Зелёный Луг"
              main_image_url:
                type: string
                example: https://example.com/image.jpg
              rating:
                type: number
                format: float
                example: 4.7
              address:
                type: string

```

```

        example: Москва, ул. Фермерская, д. 12
latitude:
    type: number
    format: float
    example: 55.7558
longitude:
    type: number
    format: float
    example: 37.6173
400:
    description: Ошибка в параметрах
    schema:
        type: object
        properties:
            status:
                type: string
                example: error
            message:
                type: string
                example: Неверные координаты или радиус
    """
    try:
        user_lat = float(request.args.get('latitude'))
        user_lon = float(request.args.get('longitude'))
        radius_km = float(request.args.get('radius'))
    except (TypeError, ValueError):
        return jsonify({'status': 'error', 'message': 'Неверные
координаты или радиус'}), 400

    haversine_sql = text(f"""
        SELECT sa.id
        FROM selleraddresses sa
        WHERE 6371 * 2 * ASIN(SQRT(
            POWER(SIN(RADIANS(sa.latitude - :lat) / 2), 2) +
            COS(RADIANS(:lat)) * COS(RADIANS(sa.latitude)) *
            POWER(SIN(RADIANS(sa.longitude - :lon) / 2), 2)
        )) <= :radius
    """)
    address_ids_rows = db.session.execute(haversine_sql, {
        'lat': user_lat,
        'lon': user_lon,
        'radius': radius_km
    }).fetchall()

    address_ids = [row[0] for row in address_ids_rows]
    if not address_ids:
        return jsonify({'status': 'success', 'sellers': []}),
200

```

```

        addresses =
SellerAddress.query.filter(SellerAddress.id.in_(address_ids)).all()

        results = []
        for addr in addresses:
            seller = addr.seller
            if not seller:
                continue
            results.append({
                'seller_id': seller.id,
                'production_name': seller.production_name,
                'main_image_url': seller.main_image_url,
                'rating': seller.rating,
                'address': addr.address,
                'latitude': float(addr.latitude),
                'longitude': float(addr.longitude),
            })

        return jsonify({'status': 'success', 'sellers': results}),
200

```

					ПГУ1.090301.21.001 ПЗ	Лист
						103
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(Обязательное)

Бизнес-проект. Электронный рынок фермерских продуктов. Серверная часть
Презентация

Бизнес-проект. Электронный рынок фермерской продукции

*Авторы работы: студенты гр.21ВВП1 Митрошин А.Д. и Суцёв М.В.
Научные руководители: к.т.н. доцент кафедры ВТ Карамышева Н.С. и доцент
кафедры ВТ Кузнецов А.В.*

Цель и актуальность проекта

Задача этой дипломной работы — создать удобную и практичную веб-платформу, которая станет мощным инструментом для развития электронной коммерции в сфере сельского хозяйства. Она поможет цифровизировать сельское хозяйство и улучшить взаимодействие между фермерами и покупателями.

2



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист
104

Краткая постановка задачи

Обеспечить удобные инструменты для управления ассортиментом и обратной связью, что улучшит взаимодействие и доверие между продавцами и потребителями.

Создать электронную платформу, которая позволит фермерам продавать продукцию напрямую минуя посредников, что повысит доходы и доступность товаров для покупателей

Разработать функционал, для быстрого и удобного поиска фермерских продуктов

3

Обзор аналогов

Торговые рынки обеспечивают прямой контакт продавцов и покупателей, а также возможность сразу оценить качество и свежесть продуктов. Основные ограничения — сезонность и ограниченный ассортимент.

01

АгроМП — специализированная платформа для агропродукции и оборудования с функциями создания заявок и информацией по сельскому хозяйству. Минусы — платные, отсутствие самовывоза, отсутствие системы рейтинга и отзывов и ориентированность на опт.

03

02

Авито — крупнейшая платформа объявлений в России, удобна для локального поиска товаров с поддержкой чата и рейтингов. Недостаток — отсутствие специализированной карты фермеров.

04

Яндекс Маркет предлагает удобную доставку и рейтинги продавцов, но не поддерживает специализированный поиск фермерских товаров и ограничен в ассортименте свежей продукции.

4

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

105



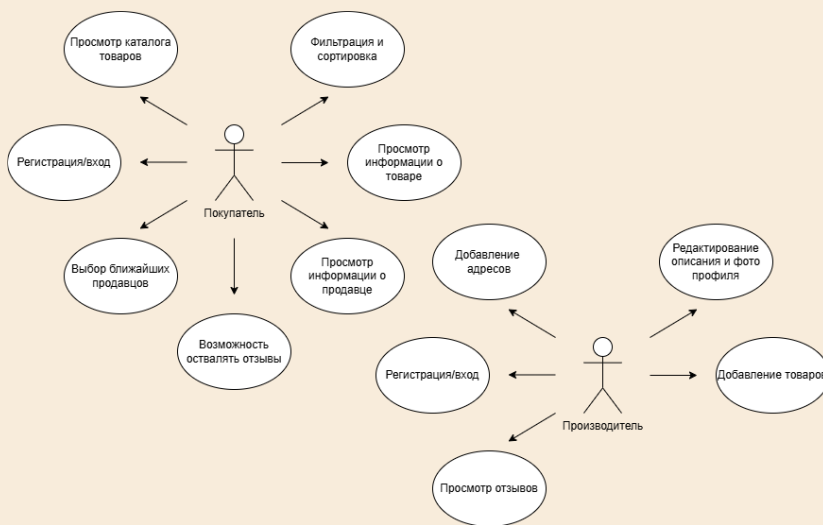
Описание функционала платформы

Платформа предлагает фермерам удобный интерфейс, который позволяет им размещать и управлять своим ассортиментом, а также отслеживать рейтинг и получать обратную связь. Платформа предоставляет удобный интерфейс для фермеров, позволяющий размещать и управлять своим ассортиментом. Отслеживать рейтинг и следить за обратной связью.

Клиенты могут ознакомиться с ассортиментом товаров, применить фильтр по категориям и найти ближайших продавцов на интерактивной карте. Кроме того, у покупателей есть возможность оставлять отзывы и оценивать продавцов. А покупатели могут просматривать каталог товаров, воспользоваться фильтром по категориям, а так же искать на интерактивной карте ближайших продавцов. Покупатели так же могут оставлять отзывы и ставить рейтинги продавцам

5

Диаграмма прецедентов



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

106



Стек технологий

Клиентская часть

React позволяет создавать адаптивные интерфейсы с использованием компонентной архитектуры, что повышает эффективность и масштабируемость клиентской части приложения. Такой подход упрощает поддержку и развитие приложения.

React-компоненты отображают данные, полученные через REST API, и автоматически обновляются при изменении состояния, обеспечивая плавное взаимодействие с пользователем.

Серверная часть

Серверная часть приложения была разработана с применением языка программирования Python и микрофреймворка Flask. Flask выполняет функции маршрутизации HTTP-запросов, обработки бизнес-логики и взаимодействия с базой данных. API формирует и отправляет ответы в формате JSON, что обеспечивает лёгкую интеграцию с интерфейсом пользователя.

7

Диаграмма разворачивания



8

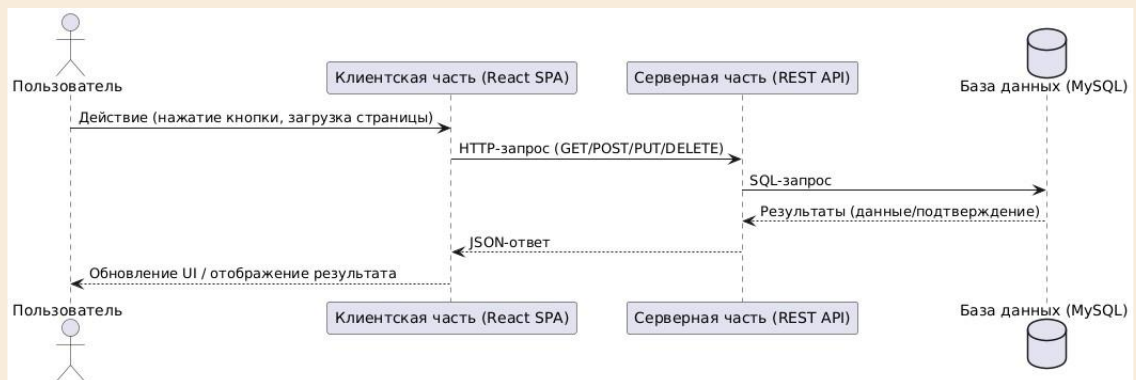
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

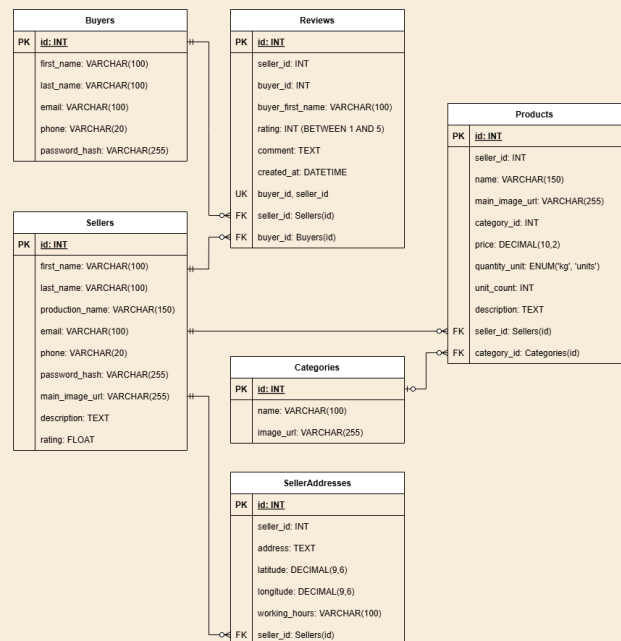
107

Диаграмма последовательности



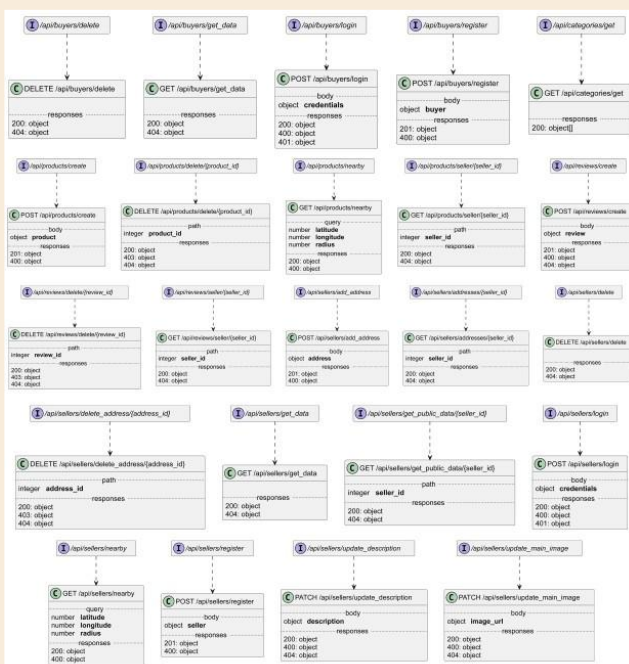
9

ER-диаграмма



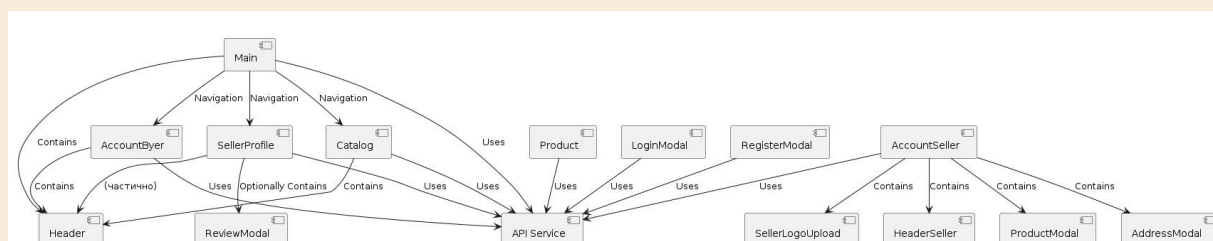
10

Диаграмма REST API ЭНДПОИНТОВ



11

Диаграмма компонентов React JS



12

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

109

Потоки доходов платформы

Эта диаграмма показывает, что основные доходы генерируются из рекламы, доставки и подписок, поддерживая финансовую устойчивость сервиса.

Разнообразие источников дохода снижает риски и способствует стабильности проекта, обеспечивая интерес как для фермеров, так и для инвесторов.



13

Каналы распространения платформы

01 Для роста платформы необходима активная рекламная кампания, направленная на привлечение новых пользователей, помимо технической поддержки и совершенствования приложения.

02 Партнёрские программы, например совместные акции с Министерством сельского хозяйства, укрепляют доверие фермеров и стимулируют участие новых производителей.

03 Основные инструменты продвижения включают SEO, контекстную рекламу и активное использование социальных сетей для максимального охвата целевой аудитории.

14

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

110

Расходы на разработку и запуск

Структура затрат по основным этапам разработки и внедрения электронной платформы.

Наибольшие затраты приходятся на backend разработку и frontend, что обусловлено необходимостью создания качественной и надежной системы.

Этап	Сумма (руб.)
Дизайн UX/UI и прототипирование	100 000
Frontend разработка	130 000
Backend разработка	150 000
Настройка серверов	70 000
Тестирование и отладка	60 000

15

Ежемесячные расходы на обслуживание

1

Хостинг и поддержка серверов требуют значительных вложений для обеспечения стабильной работы платформы и обработки пользовательских запросов без задержек.

2

Регулярное обновление и развитие функционала платформы обеспечивают соответствие современным требованиям рынка и повышают удобство использования для фермеров и покупателей.

3

Модерация контента направлена на поддержание качества и достоверности информации, а также оперативное разрешение возможных конфликтов между участниками торгов.

4

Маркетинговые мероприятия, в том числе реклама и продвижение в цифровых каналах, способствуют привлечению новых пользователей и увеличению объема продаж на площадке.

16

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

111

Расчет срока окупаемости проекта

Рост чистой прибыли отражает успешное развитие платформы и стабилизацию доходов на фоне снижения затрат.

Срок окупаемости составляет около пяти месяцев, после чего проект начинает приносить прибыль, обеспечивая финансовую устойчивость.



При расчете на 5000 активных пользователей.

17

Итоги разработки платформы

Создана функциональная веб-платформа со зрелым интерфейсом, обеспечивающим интуитивное взаимодействие пользователей с каталогом.

Платформа оптимизирует коммуникацию между фермерами и конечными потребителями, сокращая количество посредников и повышая прозрачность сделок.

Поддержка локальных производителей усиливается через механизм маркетинга и аналитики, что способствует развитию сельского хозяйства на региональном уровне.

18

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

112



Поддержка локальных производителей

Площадка предоставляет отечественным фермерам доступ к широкой базе клиентов, расширяя возможности реализации продукции и укрепляя финансовую устойчивость хозяйств.

Наша платформа поможет производителям увеличивать базу своих клиентов и развивать бизнес в долгосрочной перспективе.

19

Заключение и перспективы развития

Проект успешно воплощён в жизнь, открывая новые перспективы для аграриев и потребителей.

В перспективе можно создать мобильное приложение для iOS и Android, интегрировать платформу с платёжными системами, разработать чат и внедрить систему рекомендаций, основанную на предпочтениях пользователей. Также разработка программы лояльности позволит привлечь новых клиентов.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.090301.21.001 ПЗ

Лист

113