

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Клиентское приложение для логистического
сервиса**

Студент

(подпись, дата)

Медведев Максим Викторович
(ФИО полностью)

Руководитель

(подпись, дата)

Балашова И.Ю.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись)

Козлов А.Ю.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись)

Попова Н.А.
(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
А.Ю. Козлов
«20» декабря 2024 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Медведев Максим Викторович гр. 21ВП2 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Клиентское приложение для логистического сервиса

Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____

2. Руководитель работы к.т.н., доцент, доцент кафедры «МО и ПЭВМ» Балашова И.Ю.

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

1) *Назначение: клиентская часть логистического сервиса*

2) *Среда выполнения: Windows 10*

Языковые средства разработки: TypeScript

Инструментальные средства: Visual Studio Code, React, React Flow, openapi-fetch, shadcn/ui, Visual Paradigm 17.0

Технологии разработки: функциональное программирование, компонентный подход

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) выбор товара из каталога;

в) добавление, изменение, удаление товарами;

г) оформление заказа выбранных товаров;

д) мониторинг логистической сети;

е) построение кратчайшего маршрута;

ж) отображение отчетов о покупках и продажах.

4. Перечень подлежащих разработке вопросов:

1) <i>Анализ предметной области и постановка задачи;</i>
2) <i>Проектирование;</i>
3) <i>Реализация;</i>
4) <i>Контроль качества программного обеспечения;</i>
5) <i>Оформление пояснительной записки.</i>

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 100 страницы, 25 рисунков, 10 таблиц, 20 использованных источников и 3 приложения.

ЛОГИСТИКА, ЛОГИСТИЧЕСКИЙ СЕРВИС, КЛИЕНТСКОЕ ПРИЛОЖЕНИЕ, МОНИТОРИНГ ДАННЫХ, ЛОГИСТИЧЕСКИЕ ПРОЦЕССЫ, TYPESCRIPT, UML, REACT, REACT FLOW, ГРАФОВАЯ МОДЕЛЬ, ВИЗУАЛИЗАЦИЯ ДАННЫХ.

Объект исследования – клиентское web-приложение для логистического сервиса.

Цель выпускной квалификационной работы – спроектировать и разработать программное обеспечение для управления логистическими процессами в рамках клиентского логистического сервиса по полному логистическому циклу.

Технологии разработки – язык программирования TypeScript и библиотека для создания интерфейсов React, библиотека для работы с графовой моделью – React Flow, библиотека для работы с запросами к серверу – openapi-fetch, средство сборки web-приложения – Vite, инструмент для работы с визуалом интерфейса – shadcn/ui, среда разработки – Visual Studio Code.

Среда исполнения – Windows 10, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы разработано клиентское приложение логистического сервиса с удобным интерфейсом для управления логистическими процессами.

					ПГУ 09.03.04 – 08ВР212.16 ПЗ						
Изм.	Лист	№ докум.	Подпись	Дата	Клиентское приложение для логистического сервиса Пояснительная записка	Лит.		Лист	Листов		
Разраб.		Медведев М.В.						4	100		
Пров.		Балашова И.Ю.									
Реценз.											
Н. Контр.		Такташкин Д.В									
Утв.		Козлов А.Ю.									
					Группа 21ВР2						

Содержание

Введение.....	7
1 Анализ предметной области и постановка задачи на разработку клиентского приложения.....	8
1.1 Анализ предметной области приложения	8
1.2 Обзор и анализ программных средств для логистических сервисов	9
1.3 Постановка задачи на разработку приложения	11
1.3.1 Цели разработки и назначения приложения	11
1.3.2 Классы пользователей логистического сервиса	13
1.4 Анализ требований	14
1.4.1 Анализ функциональных требований.....	14
1.4.2 Анализ нефункциональных требований.....	17
1.5 Анализ средств для визуализации данных	19
1.6 Планирование разработки и оценка бюджета.....	20
2 Проектирование клиентской части логистического сервиса	22
2.1 Проектирование сценариев взаимодействия пользователей с системой	22
2.2 Разработка концептуальной модели	25
2.3 Проектирование пользовательского интерфейса логистического сервиса.....	29
3 Реализация клиентского приложения для логистического сервиса	43
3.1 Разработка архитектуры и структуры клиентского приложения	43
3.2 Разработка пользовательского интерфейса.....	45
3.2.1 Разработка компонентной базы приложения и стилизация пользовательского интерфейса.....	45
3.2.2 Реализация модуля визуализации логистической сети.....	47
3.2.3 Реализация отображения кратчайшего маршрута.....	48
3.3 Разработка средства взаимодействия с серверным приложением.....	50
3.4 Разработка системы управления состоянием.....	51

4	Контроль качества программного обеспечения.....	54
4.1	Обоснование выбора модели тестирования и метрик кода.....	54
4.2	Тестирование клиентского приложения для логистического сервиса....	55
4.2.1	Функциональное тестирование программного средства.....	55
4.2.2	Модульное тестирование метода отправки метаданных.....	58
4.2.3	Метрики Холстеда кода класса тиражирования.....	60
	Заключение	64
	Список использованных источников	66
	Приложение А (справочное) Глоссарий	68
	Приложение Б (обязательное) Листинг клиентского приложения	71
	Приложение В (рекомендуемое) Графическая часть	98

Введение

Рынок электронной коммерции активно развивается, что способствует увеличению объема грузоперевозок и глобализации торговых связей. С ростом спроса на быструю и надежную доставку возрастает потребность в специализированных логистических сервисах [1].

Современные логистические операции – это очень сложная сеть взаимодействий. Это не только мощная серверная часть, которая должна обрабатывать большой объем данных и находить оптимальные решения, но и удобный, интуитивно понятный пользовательский интерфейс, способный наглядно представлять сложную информацию и обеспечивать быструю реакцию на действия пользователя.

Создание клиентского приложения для логистического сервиса – крайне актуальная тема, потому что с быстрорастущим рынком электронной коммерции увеличивается сложность в контроле над логистическими процессами. Очень важно иметь удобный, интуитивно понятный и надежный интерфейс для эффективного управления такими системами. Неудачное клиентское приложение может парализовать даже самый продвинутый сервис [2].

Целью выпускной квалификационной работы является проектирование и разработка клиентского приложения, которое предоставит удобный интерфейс для управления логистическими процессами. Данное приложение позволит смоделировать полный логистический цикл – от заказа магазином до поставки от завода на склад, а также обеспечит удобный контроль над сложными процессами через систему мониторинга и визуализации логистической сети.

Исходя из цели можно поставить следующие задачи: проанализировать предметную область и аналоги, спроектировать программные средства, разработать модули системы, а также проконтролировать их качество.

Успешное решение поставленных задач позволит создать удобное и надежное клиентское приложение.

1 Анализ предметной области и постановка задачи на разработку клиентского приложения

1.1 Анализ предметной области приложения

В основе предметной области выпускной квалификационной работы лежит логистика и управление цепями поставок, рассматриваемыми в контексте разработки клиентского web-приложения для моделирования и управления основными логистическими процессами. Для понимания задач, стоящих перед анализом и проектированием приложения, необходимо определить основные термины и понятия данной предметной области, понимание которых пригодится для дальнейшего анализа и проектирования.

Логистика – процесс управления перемещением и хранением товаров и материалов от их источников до точки конечного потребителя, а также связанный с этим информационный поток [3].

Логистический сервис – комплекс услуг, оказываемых участниками логистической системы для удовлетворения потребностей потребителей в процессе поставки и распределения товаров.

Логистическая сеть – совокупность элементов логистической системы, таких как поставщики, производители, склады, распределительные центры и потребители, а также связи между ними, обеспечивающие выполнение логистических функций и процессов [4].

Логистические процессы – совокупность последовательно выполняемых действий и операций, направленных на реализацию логистических функций в цепях поставок. К логистическим процессам относятся закупка, производство, распределение, транспортировка, складирование, обработка заказов, управление запасами и другие операции.

Визуализация данных – графическое представление данных, позволяющее человеку воспринимать большие объемы информации, выявлять закономерности, тенденции и аномалии, которые могут быть неочевидны при работе с исходными данными в текстовом или табличном виде. Визуализация

данных является мощным инструментом анализа и поддержки принятия решений.

Графовая модель – математическая структура, представляющая совокупность объектов (вершин) и связей (ребер) между ними. Используется для моделирования сложных систем, где важна структура взаимодействий и отношений [5].

Таким образом, анализ предметной области показывает, что разрабатываемое клиентское приложение должно стать эффективным инструментом, обеспечивающим пользователям с разными ролями удобное взаимодействие с основными сущностями логистических процессов, а также предоставлять наглядное представление о структуре и состоянии логистической сети посредством визуализации данных с помощью графовой модели.

1.2 Обзор и анализ программных средств для логистических сервисов

Анализ существующих программных решений, представляющих собой клиентские приложения для логистических сервисов или их отдельные модули, позволит оценить текущий уровень развития в данной предметной области, выявить наиболее востребованные функции и подходы к проектированию пользовательских интерфейсов. Изучение аналогичных разработок позволяет выявить лучшие практики и определить области для совершенствования функционала будущего приложения.

Для сравнения аналогичных разработок, необходимо определить критерии, по которым можно сравнивать клиентские решения:

- актуальность дизайна и удобство использования;
- поддерживаемые типы пользователей и сценарии взаимодействия;
- осуществление заказов доступных в сервисе товаров;
- добавление, удаление и изменение товаров в сервисе;
- возможности анализа закупок и продаж товаров;
- средства мониторинга и визуализации данных;

- техническая реализация и доступность.

Оценка функционала аналогов проводилась по следующей шкале:

- 0 баллов – функционал полностью отсутствует;
- 1 балл – функционал реализован частично;
- 2 балла – функционал полностью реализован.

Под эти критерии были выбраны следующие клиентские приложения:

- СДЭК;
- DHL Express;
- Яндекс Маршрутизатор;
- СберЛогистика.

Сравнение перечисленных выше клиентских приложений для логистических систем представлено в таблице 1.

Таблица 1 – Сравнительный анализ

Критерий	СДЭК	DHL Express	Яндекс Маршрутизатор	СберЛогистика
Актуальность дизайна и удобство использования	2	2	1	1
Типы пользователей и сценарии взаимодействия	2	2	2	1
Осуществление заказов доступных в сервисе товаров	2	2	2	2
Добавление, удаление и изменение товаров в сервисе	1	1	1	1
Возможности анализа закупок и продаж товаров	1	1	1	1
Средства мониторинга и визуализации данных	1	1	2	1

Продолжение таблицы 1

Критерий	СДЭК	DHL Express	Яндекс Маршрутизатор	СберЛогистика
Техническая реализация и доступность	2	2	2	2
Итого	11	11	11	9

Проведенный количественный анализ показывает, что существующие на рынке решения обладают рядом сильных сторон. Многие сервисы предлагают развитые интерфейсы для управления отправлениями и обеспечивают удобное взаимодействие для разных классов пользователей. Некоторые платформы также демонстрируют эффективные подходы к визуализации данных, в частности, при отображении маршрутов и мониторинге транспортных потоков.

Изучение сильных сторон похожих сервисов позволит сформировать понимание текущих стандартов и ожиданий пользователей от логистических приложений. Также стоит выделить потенциальные области для дальнейшего развития, например, область интерактивной визуализации сложных логистических структур и процессов. Существующие решения, хотя и предлагают инструменты мониторинга, не всегда предоставляют администратору визуальное представление для управления всей логистической системы.

Собранные данные в ходе анализа похожих систем станут основой при проектировании и разработки клиентского приложения для логистического сервиса.

1.3 Постановка задачи на разработку приложения

1.3.1 Цели разработки и назначения приложения

Разрабатываемое клиентское приложение должно предоставлять возможность эффективного взаимодействия с логистической системой. Основной целью разработки является обеспечение эффективного контроля над логистическими процессами со стороны различных классов пользователей.

Необходимо разработать удобный и надежный пользовательский интерфейс, который обеспечит эффективное взаимодействие всех участников системы и позволит им выполнять специфичные для их роли операции. В рамках разработки клиентского приложения были поставлены следующие задачи:

- проектирование и реализация отдельных пользовательских представлений для каждого класса пользователей, учитывающих специфику их взаимодействия с системой;
- разработка функционала для взаимодействия с основными сущностями логистического сервиса (товарами, заказами, поставками, пользователями, складом) через API с серверной частью;
- внедрение механизмов аутентификации и авторизации, а также разграничение доступа к частям приложения по ролевой модели;
- разработка админ-панели, предоставляющей инструменты для мониторинга и управления логистическими процессами и сущностями системы через средства визуализации данных;
- разработка модуля для формирования и отображения аналитических отчетов.

Особое внимание в рамках данной работы уделяется разработке административной панели, поскольку именно через нее будет осуществляться централизованное управление логистическими процессами и мониторинг состояния системы. Такой системе нужны продвинутое средства визуализации данных, например, графовые модели. Этот инструмент позволит в удобной форме отобразить некоторые логистические процессы – доставку товаров через несколько логистических пунктов, или то, как эти пункты связаны между собой, обеспечивая наглядный контроль [6].

В результате успешного решения задач должно быть создано клиентское приложение, которое станет эффективным инструментом для управления и контроля над постоянно усложняющимися логистическими процессами.

1.3.2 Классы пользователей логистического сервиса

Разрабатываемое клиентское приложение для логистического сервиса будет полезно для трех классов пользователей: администратор, производитель, потребитель. Описание каждого класса представлено ниже:

- потребитель – это представитель магазина, подключенного к системе, который имеет возможность заказывать товары со склада;
- поставщик – пользователь, представляющий производителя продукции, который может поставлять товары на склад;
- администратор – центральная управляющая роль, контролирующая всю систему через возможности мониторинга и визуализации логистических процессов.

Подробное описание бизнес-процессов каждого класса пользователей представлено в таблице 2.

Таблица 2 – Классы пользователей приложения

№	Наименование класса	Затрагиваемая часть сервиса	Бизнес-процессы	Используемые объекты
1	Администратор	Приложение администратора	Управление пользователя	Пользователи
			Управление каталогом товаров	Товары
			Модерация заявок на добавление товаров	Клиенты, Товары
			Мониторинг доставки заказов	Заказы
			Управление логистической инфраструктурой	Клиенты
2	Потребитель	Приложение потребителя	Авторизация и регистрация	Пользователи
			Просмотр и выбор товаров	Товары
			Оформление заказа	Заказы, Фин. операции
			Отслеживание заказов	Заказы
			Управление профилем	Клиенты
			Просмотр отчетности	Заказы, Фин. операции
3	Поставщик	Приложение поставщика	Авторизация и регистрация	Пользователи
			Создание карточки товара	Товары
			Управление созданными товарами	Товары
			Отправка товаров на склад	Товары
			Мониторинг продаж	Заказы, Фин. операции

Клиентское приложение для логистического сервиса необходимо спроектировать так, чтобы в нем были учтены все описанные классы пользователей с реализацией их бизнес-процессов.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

На основе ранее проведенного анализа предметной области и выделенных классов пользователей с их ключевыми бизнес-процессами, был сформирован детализированный перечень функциональных требований к разрабатываемому клиентскому приложению. Определение этих требований является фундаментальным этапом, поскольку именно они ложатся в основу всего последующего проектирования и реализации системы, гарантируя, что конечный продукт будет соответствовать ожиданиям и решать поставленные задачи.

Для представления функциональных требований к разрабатываемому клиентскому приложению используется диаграмма вариантов использования. Этот инструмент визуализации позволяет описать систему с точки зрения внешнего наблюдателя, акцентируя внимание на взаимодействии различных классов пользователей (акторов) с системой для достижения конкретных целей. Диаграмма вариантов использования демонстрирует, какие основные действия каждый класс пользователей сможет выполнять в приложении, и какие ключевые сценарии использования при этом будут реализованы. При этом фокус делается именно на том, что система должна делать, а не на деталях ее внутренней реализации, архитектурных решениях или конкретных технологиях, которые будут рассмотрены на последующих этапах работы [7].

Разработанная диаграмма вариантов использования, в которой отражено взаимодействие пользователей с логистическим сервисом, приведена на рисунке 1.

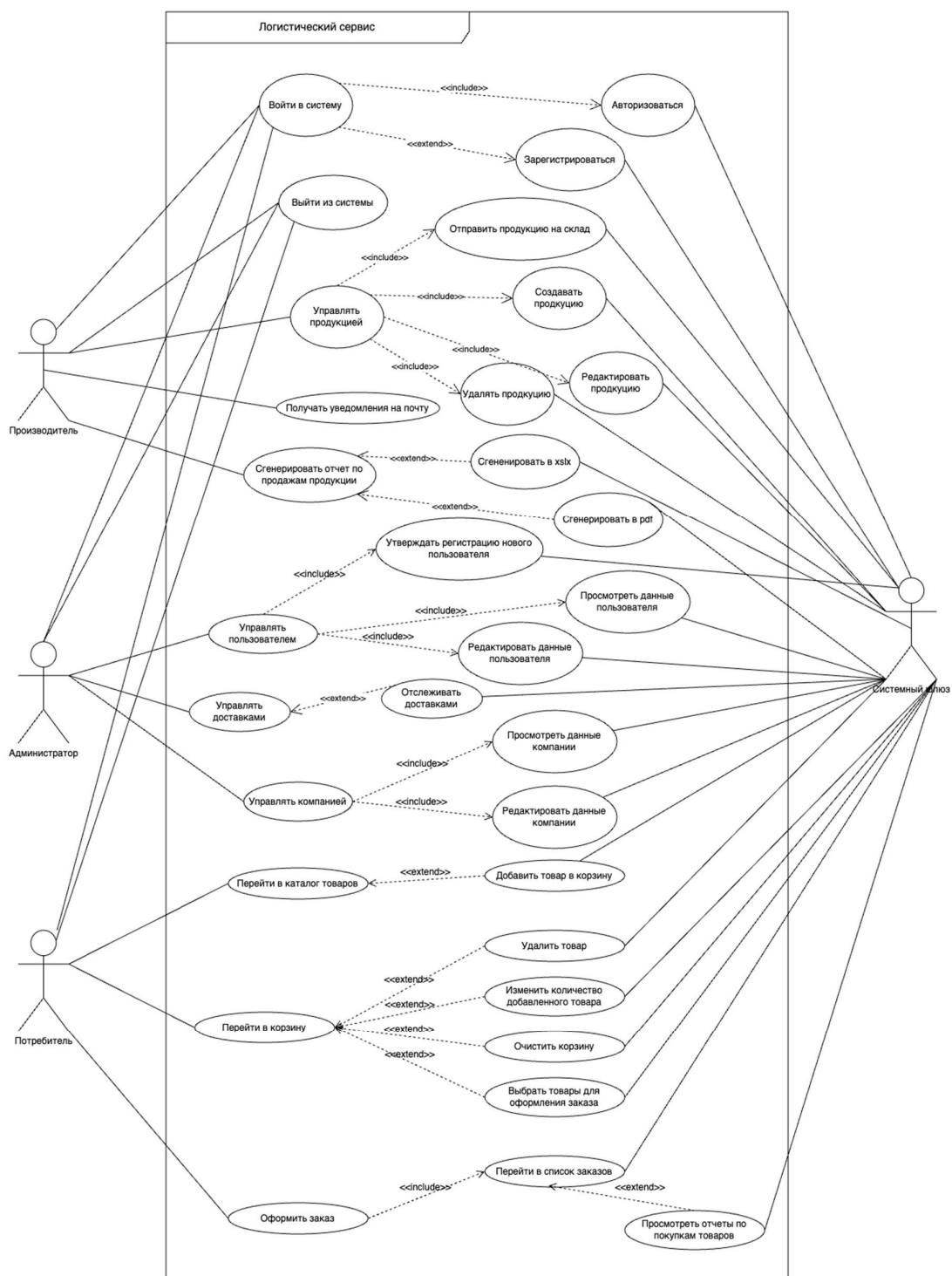


Рисунок 1 – Диаграмма вариантов использования логистического сервиса

Спецификации наиболее важных вариантов использования приведены ниже.

Сценарий варианта использования «Создавать продукцию» представлен на рисунке 2.

Прецедент: Создавать продукцию
ID: 1
Краткое описание: Пользователь/поставщик создает новый товар
Главные актеры: Пользователь/поставщик
Второстепенные актеры: Нет
Предусловия: 1. Авторизован в системе 2. Находится на странице управления своими товарами
Основной поток: 1. Просматривает список своих товаров 2. Выбирает действие – добавить новый товар 3. Заполняет необходимые поля 4. Сохраняет изменения и отправляет новый товар на модерацию Альтернативные потоки: 3а. Введены некорректные данные, получает сообщение об ошибке валидации 3б. Отправка на модерацию не удалась, получает уведомление 4а. Закрывает окно, не сохранив изменения
Постусловия: 1. Создан новый продукт, информация о товарах обновлена,

Рисунок 2 – Сценарий варианта использования «Создавать продукцию»

Сценарий варианта использования «Утверждать регистрацию нового пользователя» изображен на рисунке 3.

Прецедент: Утверждать регистрацию нового пользователя
ID: 2
Краткое описание: Пользователь/администратор рассматривает заявки на регистрацию новых пользователей и принимает решение об их одобрении или отклонении
Главные актеры: Пользователь/администратор
Второстепенные актеры: Нет
Предусловия: 1. Авторизован в системе 2. Находится на странице управления пользователями
Основной поток: 1. Выбирает конкретную заявку на регистрацию 2. Анализирует предоставленные данные 3. Нажимает на кнопку «Одобрить» Альтернативные потоки: 2а. Заявки на регистрацию отсутствуют 3а. Нажимает на кнопку «Отклонить» 3б. Изменяет статус заявки на «Отклонено»
Постусловия: 1. Новый пользователь получает доступ к системе 2. Список заявок, ожидающих модерацию, обновлен

Рисунок 3 – Сценарий варианта использования «Утверждать регистрацию нового пользователя»

Сценарий варианта использования «Отслеживать доставки» представлен на рисунке 4.

Прецедент: Отслеживать доставки
ID: 3
Краткое описание: Пользователь/администратор просматривает актуальную информацию о логистических процессах (поставках и заказах), чтобы контролировать их статус и ход выполнения
Главные актеры: Пользователь/администратор
Второстепенные актеры: Нет
Предусловия: 1. Пользователь/администратор авторизован в приложении 2. В системе существуют логистические процессы (поставки или заказы) в различных статусах
Основной поток: 1. Переходит в раздел админ-панели, предназначенный для мониторинга логистики 2. Видит список логистических процессов 3. Выбирает пункт «Заказы магазинам» 4. Включает просмотр графовой визуализации 5. Просматривает отображение кратчайшего маршрута доставок через несколько логистических пунктов Альтернативные потоки: 2а. Список пуст или недоступен, пользователь/администратор получает сообщение, что нет доступных логистических процессов 3а. Выбирает пункт «Поставки заводов» 3б. Просматривает список поставок 4а. Не включает просмотр графовой визуализации 4б. Кратчайший маршрут отображается как список логистических пунктов
Постусловия: 1. Пользователь/администратор получил доступ к актуальной информации о состоянии и деталях логистических процессов

Рисунок 4 – Сценарий варианта использования «Отслеживать доставки»

В результате проведенного анализа функциональных требований были определены ключевые варианты использования разрабатываемого приложения. Для основных вариантов использования были разработаны спецификации, которые служат для наглядного представления последовательности действия, выполняемых пользователем и системой.

1.4.2 Анализ нефункциональных требований

Прежде чем перейти к проектированию программного средства, необходимо также провести анализ нефункциональных требований. Нефункциональные требования определяют характеристики качества программного обеспечения и ограничения, накладываемые на его разработку и

функционирование. Они описывают, как система должна себя вести, насколько она должна быть производительной, надежной, безопасной, удобной в использовании, поддерживаемой.

Клиентское приложение может быть реализовано для различных платформ, таких как web, мобильные устройства или настольные компьютеры. Для логистического сервиса оптимальным выбором является формат web-приложения, поскольку оно обеспечивает максимальную доступность для пользователей с любыми устройствами через браузер. Кроме того, web-приложения отличаются легкостью развертывания и обновления, что положительно сказывается на сопровождаемости системы.

Приложение должно обеспечивать высокую скорость загрузки и интерактивность пользовательского интерфейса при работе с логистическими данными. Для этих задач подойдет библиотека React, которая является одним из ведущих инструментов для создания производительных и доступных web-приложений. Для оптимизации процесса сборки модулей клиентского приложения и ускорения доставки готового кода пользователю будет использоваться сборщик Vite.

Отдельным требованием, связанным с удобством использования и наглядностью, является необходимость визуализации графовых моделей логистической системы. Эта функция критически важна для административной панели. Существует множество библиотек, которые подходят для решения этой задачи: Vis.js, Cytoscape.js, D3.js, React Flow. Анализ и выбор оптимального инструмента будет рассмотрен ниже.

Взаимодействие клиентского приложения с серверной частью должно осуществляться посредством REST API. А для обеспечения безопасной работы необходимо применять механизмы аутентификации и авторизации пользователей с использованием JWT-токенов. Это позволит гибко управлять доступом к функционалу согласно конкретному пользовательскому классу.

Также предполагается поддержка стабильной работы клиентского приложения в актуальных версиях основных web-браузерах, таких как Google Chrome, Mozilla Firefox, Apple Safari.

1.5 Анализ средств для визуализации данных

После анализа требований можно прийти к выводу, что основной функционал клиентского приложения по взаимодействию с логистическими процессами должен находиться в панели администратора сервиса. Это мониторинг и управление логистической сетью. Так, для наглядного представления структуры сети (склад, заводы, магазины и связи между ними), а также перемещения товаров по маршрутам поставок, требуется эффективная визуализация графовой модели. Она поможет администратору быстро оценить текущую ситуацию, выявить проблемные участки и принять своевременные решения [8].

Существуют различные подходы к визуализации графовых моделей:

- статическая, отображающая неизменные данные для фиксированных схем;
- динамическая, обеспечивающая обновление данных в реальном времени;
- интерактивная, позволяющая пользователю взаимодействовать с графом (масштабировать, перемещать узлы, получать подробную информацию для анализа).

Но наиболее универсальным является гибридный подход, сочетающий возможности динамического обновления и интерактивного взаимодействия, что соответствует потребностям логистических систем, где важен мониторинг процессов и работа с данными [6].

Для реализации визуализации графовых моделей в клиентском web-приложении на React был проведен анализ популярных инструментальных средств: Vis.js, Cytoscape.js, D3.js и React Flow. На основе проведенного анализа с учетом специфики разрабатываемого сервиса, для реализации которого

требуется гибридная визуализация графовой модели при сравнительно небольшом количестве узлов, наиболее подходящим инструментом стала библиотека React Flow. Она обеспечивает лучший баланс между простой интеграцией в web-приложение и функциональностью для создания интерактивных и динамичных графов. Также эта библиотека достаточно производительная для решения задачи визуализации логистической сети [8].

1.6 Планирование разработки и оценка бюджета

Планирование процесса разработки и оценка необходимых ресурсов – это важные составляющие для успешной реализации любого программного продукта. Тщательное планирование позволяет определить последовательность выполнения работ, необходимые трудозатраты и помогает сформировать ориентировочный бюджет проекта.

Для управления данным проектом была использована программа ProjectLibre. Она предоставляет функционал для определения этапов разработки, назначения ресурсов, установки зависимостей между задачами и отслеживания общего хода выполнения проекта.

Были определены основные этапы разработки клиентского приложения и их трудоемкости, а затем внесены в ProjectLibre. На основе этой информации была сформирована диаграмма Ганта. Она отображает последовательность и временные рамки выполнения ключевых этапов работы, представляет собой горизонтальный столбчатый график, который нужен для иллюстрации плановых сроков и длительности выполнения задач проекта [9].

В ходе планирования было выделено 23 основных задач, охватывающих все стадии от анализа предметной области и требований до реализации, тестирования и подготовки итоговой документации. Работы по проекту были начаты 3 апреля 2025 года и, согласно сформированной диаграмме Ганта, их полное завершение запланировано на 11 июля 2025 года.

Детализированный список поставленных задач с указанием их длительности, дат начала и окончания, а также назначенных ресурсов

представлен на рисунках В.1 и В.2 в Приложении В. Все задачи в рамках разработки непосредственно клиентского приложения выполнялись одним ресурсом – «Разработчик».

Общие плановые трудозатраты на разработку клиентской части логистического сервиса, включая анализ, проектирование, реализацию и контроль качества, согласно данным календарного плана, составили 588 часов.

Для расчета условной стоимости разработки был принята среднерыночная часовая ставка web-разработчика начального уровня, которая на момент расчета составляет 475 рублей в час. Таким образом, условная полная стоимость проекта по разработке клиентского приложения для логистического сервиса оценивается в 248900 рублей.

2 Проектирование клиентской части логистического сервиса

2.1 Проектирование сценариев взаимодействия пользователей с системой

На этапе проектирования клиентского приложения особенно важно не только определить функциональные возможности системы, но и детально проработать то, как классы пользователей будут с ними взаимодействовать. Клиентское приложение, будучи основным интерфейсом между пользователем и сложной логистической системой, требует особенно тщательного подхода к проектированию пользовательских сценариев. От того, насколько интуитивно понятны и логичны будут потоки действий, напрямую зависит удобство использования приложения, эффективность работы пользователей и, как следствие, общая ценность всей системы. Тщательно проработанные сценарии служат основой для создания интуитивно понятной навигации и логичного расположения элементов управления на страницах приложения.

В связи с этим, необходимо уделить особое внимание моделированию ключевых сценариев взаимодействия всех определенных ранее классов пользователей (потребитель, поставщик, администратор) с разрабатываемым логистическим сервисом. Основная цель моделирования – это не только формализация последовательности шагов в рамках типовых сценариев, но и выявление возможных точек принятия решений, альтернативных и параллельных потоков выполнения задач, а также определение областей ответственности для каждого участника системы. Такой подход позволяет предвидеть и разрешить потенциальные проблемы взаимодействия еще на этапе проектирования [10].

На рисунке 5 представлена схема последовательности действий и разграничения зоны ответственности для каждого класса пользователей.

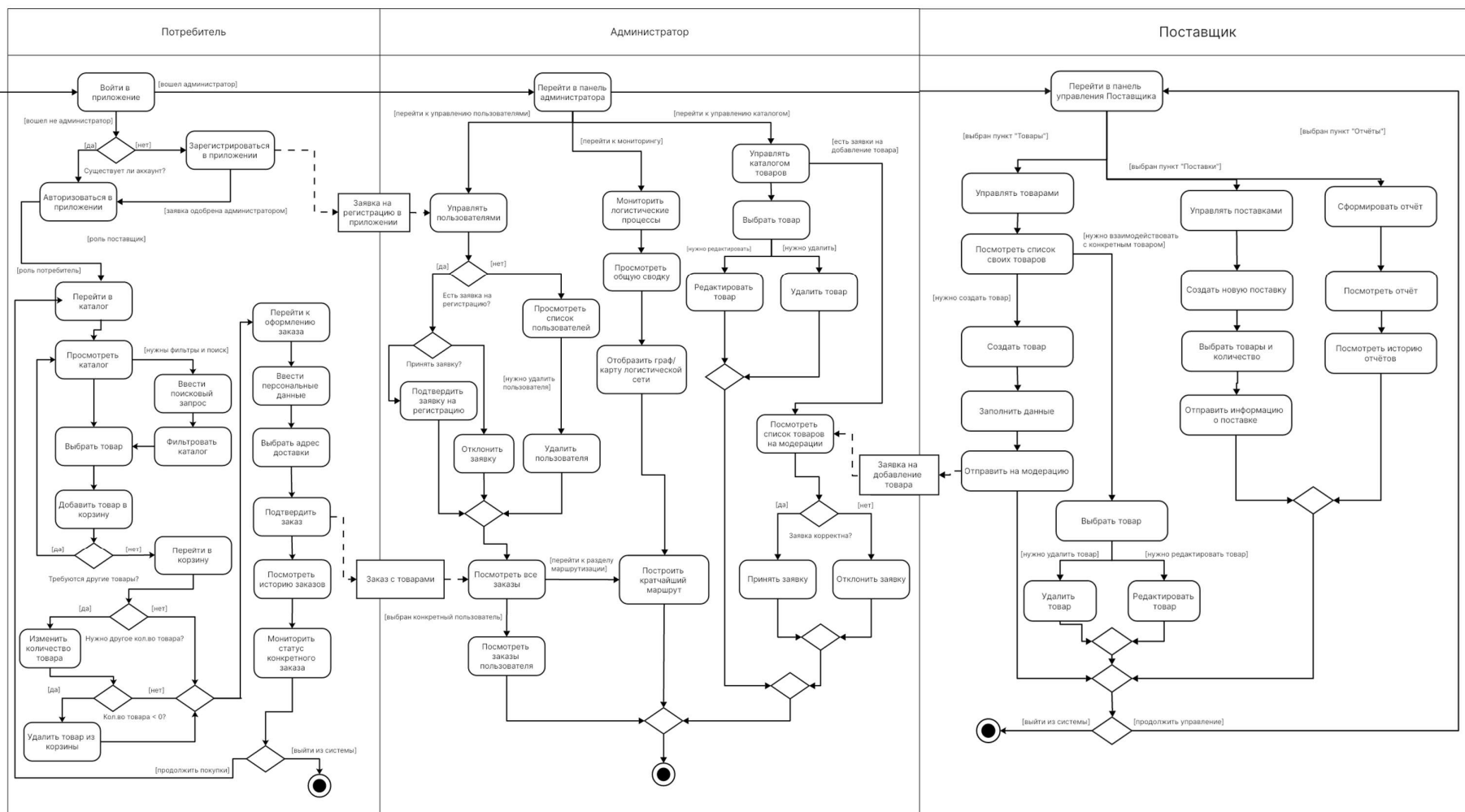


Рисунок 5 – Сценарии выполнения бизнес-процессов в системе

Представленная схема иллюстрирует ключевые сценарии взаимодействия пользователей с системой. Для всех классов пользователей начальным этапом является вход в приложение, с возможностью регистрации для новых участников. Авторизованный потребитель осуществляет работу с каталогом товаров, включая просмотр, фильтрацию и поиск, формирует заказ через корзину с возможностью изменения позиций и переходит к оформлению, указывая необходимые данные. Также ему доступна история и отслеживание заказов. Поставщик управляет своими товарными позициями, создавая, редактируя и отправляя их на модерацию, а также формирует поставки на склад и просматривает отчеты по продажам. Администратор помимо стандартных функций управления учетными записями, модерации заявок на добавление товаров и полного контроля над каталогом, выполняет ключевые задачи по мониторингу и управлению логистическими процессами. Особое значение здесь уделяется именно визуализации данных: администратор получает доступ к инструментам для отображения карты логистической сети в виде интерактивной графовой модели. Это позволяет наглядно представить структуру связей между логистическими сущностями: складом, заводами и магазинами. Все сценарии предусматривают точки принятия решений, определяющие дальнейший ход процесса, и возможность выхода из системы. Важно отметить, что данные сценарии также учитывают альтернативные потоки и обработку исключительных ситуаций, что позволяет спроектировать более отказоустойчивое и предсказуемое поведение приложения.

Разработанные и визуализированные сценарии взаимодействия пользователей служат основой для последующих этапов проектирования. Они определяют логическую структуру пользовательского интерфейса, последовательность экранов и элементов управления, а также влияют на проектирования модели данных приложения клиентского приложения для логистического сервиса.

2.2 Разработка концептуальной модели

После анализа предметной области и проектирования сценариев взаимодействия классов пользователей с системой, необходимо спроектировать структуры данных, которые будут поддерживать весь функционал разрабатываемого клиентского приложения. Основой для этого служит разработка концептуальной модели базы данных. Важно сделать её правильной, поскольку ошибки в структуре данных могут привести к сложностям в реализации бизнес-логики, избыточности хранения или проблемам с производительностью на более поздних стадиях разработки.

Концептуальная модель данных – это высокоуровневое описание основных информационных сущностей, их атрибутов и взаимосвязей в рамках предметной области. Главная задача – определить, какая информация должна храниться в системе и как различные части этой информации будут связаны между собой, что в дальнейшем понадобится для представления данных пользователю. Для разрабатываемого логистического сервиса это очень важный этап в проектировании, поскольку такой сервис оперирует большим количеством взаимосвязанных данных (пользователи, товары, заказы, поставки, логистические узлы). Тщательно проработанная модель данных позволяет не только корректно хранить информацию, но и оптимизировать запросы к ней, что напрямую влияет на скорость работы всего приложения. Стоит отметить, что на основе модели в дальнейшем будет строиться структура пользовательского интерфейса клиентского приложения, поэтому нужно знать, какие данные будут храниться в базе данных [11].

Процесс концептуального моделирования включает в себя выделение ключевых сущностей предметной области, определение их атрибутов и установление связей различных видов (один-к-одному, один-ко-многим, многие-ко-многим) между ними для определения логики взаимодействия объектов. Для наглядного представления структуры такой модели используется диаграмма «сущность-связь», которая представлена на рисунке 6.



Рисунок 6 – Диаграмма «сущность-связь» концептуальной модели базы данных

Описание сущностей концептуальной модели данных представлено в таблице 3.

Таблица 3 – Сущности базы данных

№	Наименование сущности	Описание сущности	Категория	Количество экземпляров
1	Пользователи	Учетная запись пользователя системы	Стержневая	1000
2	Роль	Определяет права доступа пользователя	Справочная	3
3	Клиент	Профиль магазина-потребителя или завода-поставщика	Стержневая	1000
4	Продукт	Общая информация о товаре	Стержневая	500
5	Модель продукта	Конкретная вариация товара (размер, цвет)	Стержневая	5000
6	Заказ	Запрос потребителя на покупку товаров	Стержневая	10000
7	Содержимое заказа	Позиции товаров в конкретном заказе	Стержневая	50000
8	Корзина	Временное хранилище выбранных товаров, готовых к заказу	Стержневая	500
9	Содержимое корзины	Позиции товаров в конкретной корзине	Стержневая	2000
10	Заявка	Запрос поставщика на добавление или изменение товара	Стержневая	1000
11	Финансовая операция	Запись о движении денежных средств	Стержневая	20000
12	Деталь операции	Конкретизация деталей финансовой операции	Стержневая	20000

Спецификация связей между сущностями концептуальной модели базы данных представлена в таблице 4.

Таблица 4 – Связи между сущностями базы данных

№	Сущность 1	Сущность 2	Описание связи	Мощность связи
1	Пользователь	Роль	Пользователь имеет одну роль	1:1
2	Пользователь	Клиент	Пользователь может быть клиентом (иметь профиль)	1:1
3	Пользователь	Заказ	Пользователь (потребитель) делает заказы	1:M
4	Пользователь	Корзина	Пользователь (потребитель) имеет корзину	1:1
5	Клиент	Продукт	Клиент (поставщик) предлагает продукты	1:M
6	Продукт	Модель продукта	Продукт имеет несколько моделей	1:M
7	Модель продукта	Содержимое корзины	Модель продукта добавляется в корзину	1:M
8	Корзина	Содержимое корзины	Корзина содержит товарные позиции	1:M
9	Модель продукта	Содержимое заказа	Модель продукта входит в заказ	1:M
10	Заказ	Содержимое заказа	Заказ включает товарные позиции	1:M
11	Заказ	Финансовая операция	С заказом связана финансовая операция	1:1
12	Финансовая операция	Деталь операции	Операция имеет детализацию	1:M

В процессе разработки концептуальной модели данных были определены ключевые сущности логистического сервиса, их атрибуты и взаимосвязи. Для визуализации этой структуры в среде Visual Paradigm была построена диаграмма «сущность-связь». Данный подход позволил четко структурировать информацию и заложить основу для последующего проектирования эффективного пользовательского интерфейса, который будет оперировать этими данными.

2.3 Проектирование пользовательского интерфейса логистического сервиса

Проектирование пользовательского интерфейса является важным этапом разработки любого современного web-приложения, и его значимость многократно возрастает для таких многофункциональных систем, как разрабатываемый логистический сервис. Так как система оперирует значительными объемами данных, управляет сложными бизнес-процессами и предполагает взаимодействие различных классов пользователей с уникальными задачами, качество интерфейса становится критически важным. Интуитивно понятный, логически выстроенный и отзывчивый интерфейс не просто повышает удобство, но и выступает ключевым инструментом для эффективного управления логистическими процессами, обеспечивая точность операций и минимизацию ошибок [12].

При проектировании пользовательского интерфейса особое внимание необходимо уделить потребностям и сценариям работы каждого из трех классов пользователей: потребителя (представителя магазина), поставщика (представителя завода) и администратора сервиса. Для каждой роли нужно создать не просто набор отдельных экранов, а целостную рабочую среду, предоставляющую весь необходимый функционал для выполнения задач.

Разрабатываемый логистический сервис – закрытое приложение, поэтому доступ можно получить только авторизованным пользователям. Это нужно для обеспечения безопасности и контроля над данными и операциями. Поэтому начальной точкой взаимодействия с системой должны стать страницы авторизации и регистрации.

Страница авторизации, представленная на рисунке 7, предназначена для уже зарегистрированных пользователей. Пользователю предлагается ввести свои учетные данные. Для удобства предусмотрена кнопка «Войти» для подтверждения ввода и ссылка «Регистрация» для тех, кто еще не имеет аккаунта в системе.

Вход в систему
Введите ваш email и пароль для входа в систему

Email

admin@gmail.com

Пароль

.....

Войти

Регистрация

Рисунок 7 – Страница авторизации

Страница регистрации спроектирована для создания новых учетных записей. Процесс регистрации разбит на два логических шага для упрощения ввода информации. Сначала пользователю нужно указать данные для входа, а затем личную информацию, в том числе и выбрать желаемую роль: поставщик или потребитель. Также можно вернуться назад к странице авторизации. Часть страницы с шагом «Данные для входа» представлена на рисунке 8.

Регистрация
Зарегистрируйтесь в логистической системе

Данные для входа Информация о вас

Email

admin@gmail.com

Пароль

.....

Подтвердите пароль

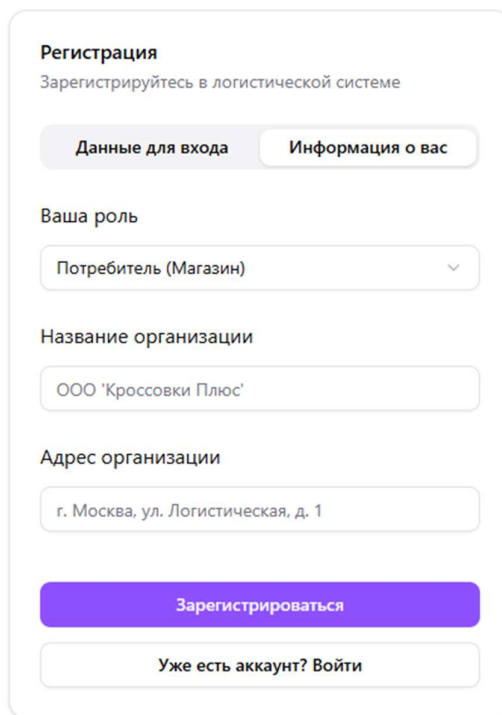
xxxxxxx

Зарегистрироваться

Уже есть аккаунт? Войти

Рисунок 8 – Страница регистрации с шагом «Данные для входа»

Шаг страницы регистрации для ввода личной информации представлен на рисунке 9.



The screenshot shows a registration form titled "Регистрация" (Registration) with the subtitle "Зарегистрируйтесь в логистической системе" (Register in the logistics system). There are two tabs: "Данные для входа" (Login data) and "Информация о вас" (Information about you), with the latter being active. The form includes the following fields:

- Ваша роль** (Your role): A dropdown menu with the selected option "Потребитель (Магазин)" (Consumer (Store)).
- Название организации** (Organization name): A text input field containing "ООО 'Кроссовки Плюс'" (LLC 'Sneakers Plus').
- Адрес организации** (Organization address): A text input field containing "г. Москва, ул. Логистическая, д. 1" (Moscow, Logistics St., 1).

At the bottom of the form, there is a prominent purple button labeled "Зарегистрироваться" (Register) and a white button labeled "Уже есть аккаунт? Войти" (Already have an account? Log in).

Рисунок 9 – Страница регистрации с шагом «Информация о вас»

После успешной регистрации или авторизации пользователь перенаправляется в рабочую область, соответствующую его роли в системе.

Для пользователя с классом «Потребитель» центральным элементом взаимодействия является страница каталога товаров. Здесь реализован функционал просмотра доступных на складе товаров с возможностью их фильтрации по различным параметрам (тип, цена, бренд) и сортировки. Для быстрого нахождения конкретных позиций предусмотрена поисковая строка. У каждого товара есть своя «карточка» с указанием ключевых характеристик и возможностью добавления в корзину. При переходе к детальному просмотру товара пользователю предоставляется необходимая информация о выбранной позиции, включая ее подробные характеристики с отображением актуального количества на складе и указанием необходимого для заказа объема. После выбора всех параметров товар может быть добавлен в корзину. Страница каталога товаров представлена на рисунках 10.

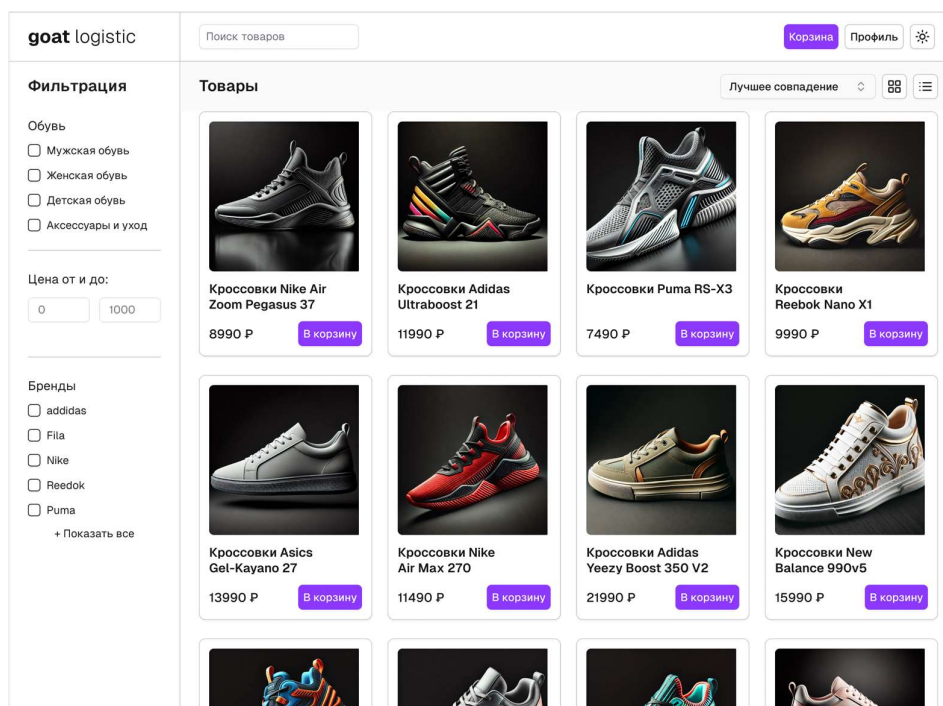


Рисунок 10 – Страница каталога товаров

Страница детального просмотра конкретного товара представлена на рисунке 11.

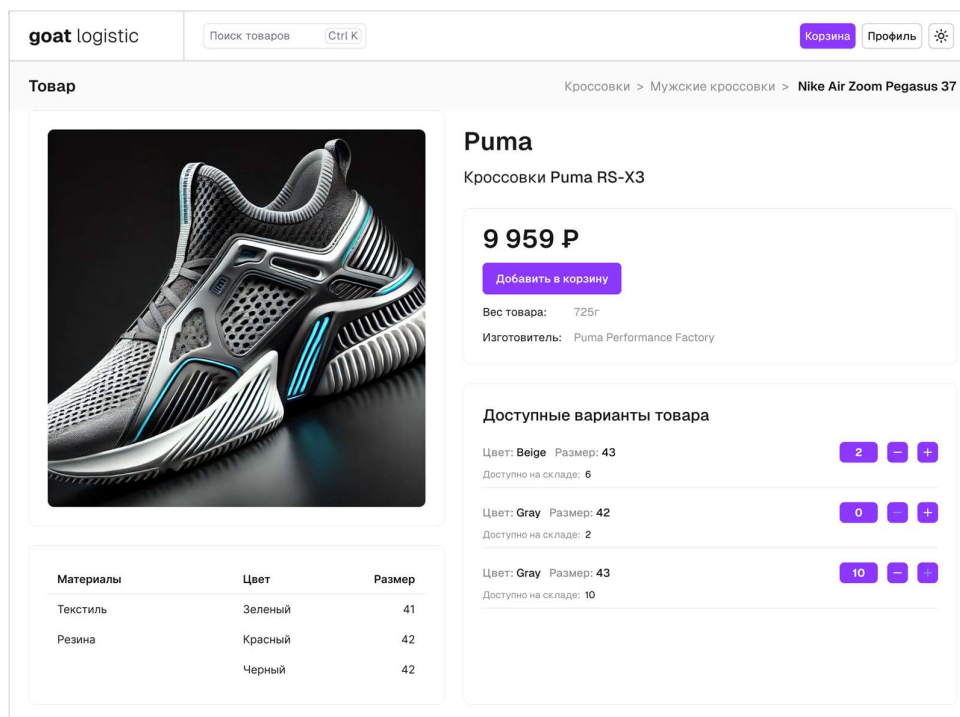


Рисунок 11 – Страница конкретного товара

Для управления wybranнми товарами и отслеживания истории покупок спроектирована комбинированная страница «Корзина и заказы». Она представлена на рисунке 12. В левой части страницы пользователю доступен функционал корзины, где отображаются добавленные товары с указанием их вариаций и стоимости. Здесь реализована возможность изменения количества каждой позиции и удаления товаров из корзины, а также ее полной очистки. Правая часть страницы отведена под историю заказов, где представлен список ранее совершенных покупок с указанием их статусов, дат и сумм. Предусмотрен поиск и сортировка по заказам, а также отображение подробного состава выбранного заказа. С этой страницы пользователь может инициировать оформление нового заказа.

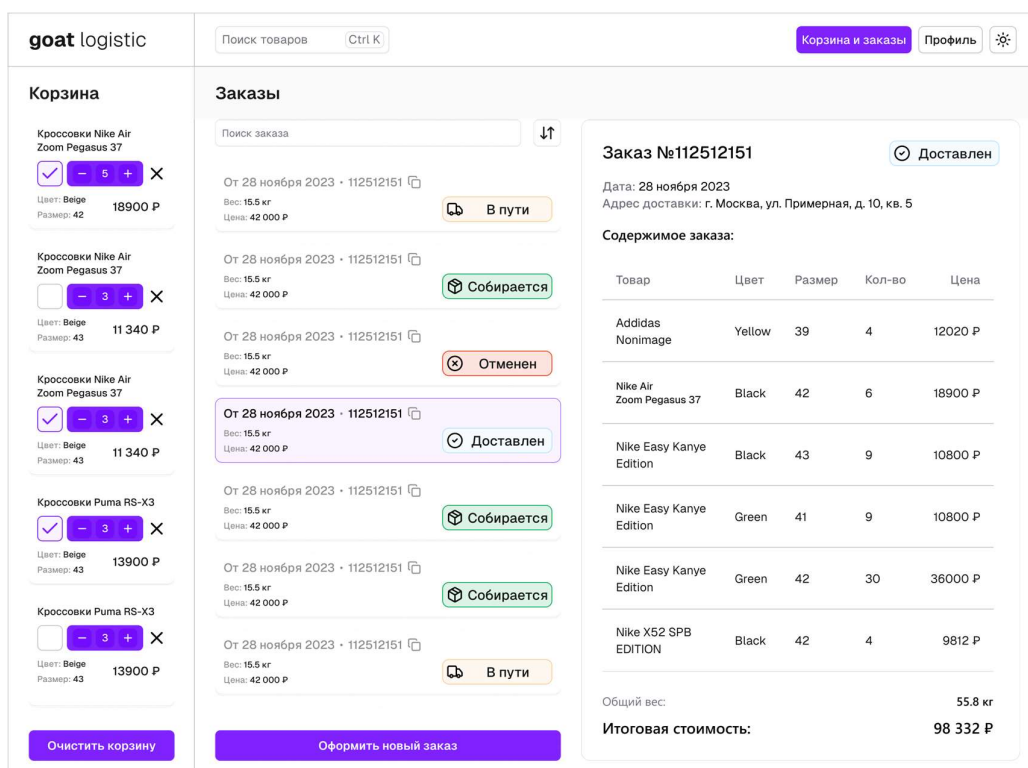


Рисунок 12 – Страница с корзиной и списком заказов

Процесс оформления заказа реализован в виде модального окна, представленного на рисунке 13. Оно появляется после соответствующего действия пользователя. В данном окне отображается полный перечень товаров, готовых к заказу, с возможностью финальной корректировки их количества или

удаления отдельных позиций. Пользователю также демонстрируются итоговые данные по заказу: информация о заказчике, адрес доставки и общая стоимость.

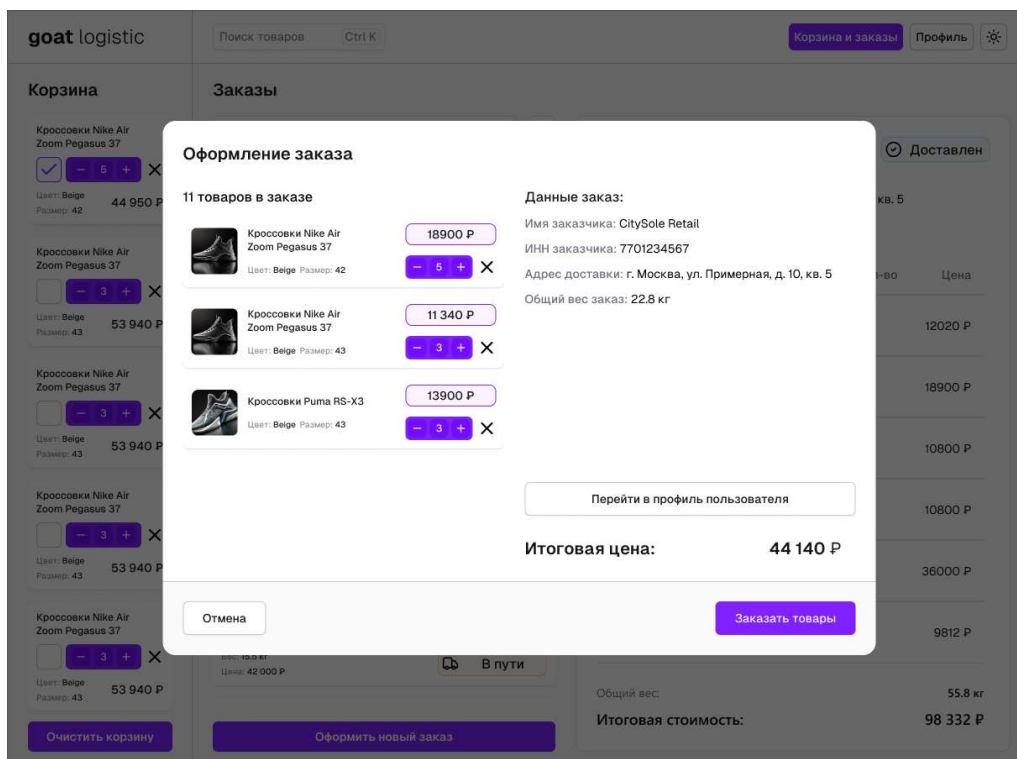


Рисунок 13 – Модальное окно с оформлением заказа

Такой подход к оформлению заказа через модальное окно позволяет пользователю оставаться в контексте предыдущей страницы, не теряя текущего состояния.

Также есть страница с профилем пользователя, где для потребителя предусмотрено управление персональными данными и анализ закупочной деятельности. Страница разделена на две функциональные зоны. В первой зоне пользователю доступна информация о его компании и данные для входа в систему, с возможностью их изменения. Во второй зоне находится раздел «Аналитика закупок», предназначенный для визуального представления статистики по совершенным заказам. Сюда добавлены интерактивные графики, отображающие разные показатели: динамику объема закупок, структуру закупок по категориям и распределение закупок по брендам. Страница с профилем представлена на рисунке 14.

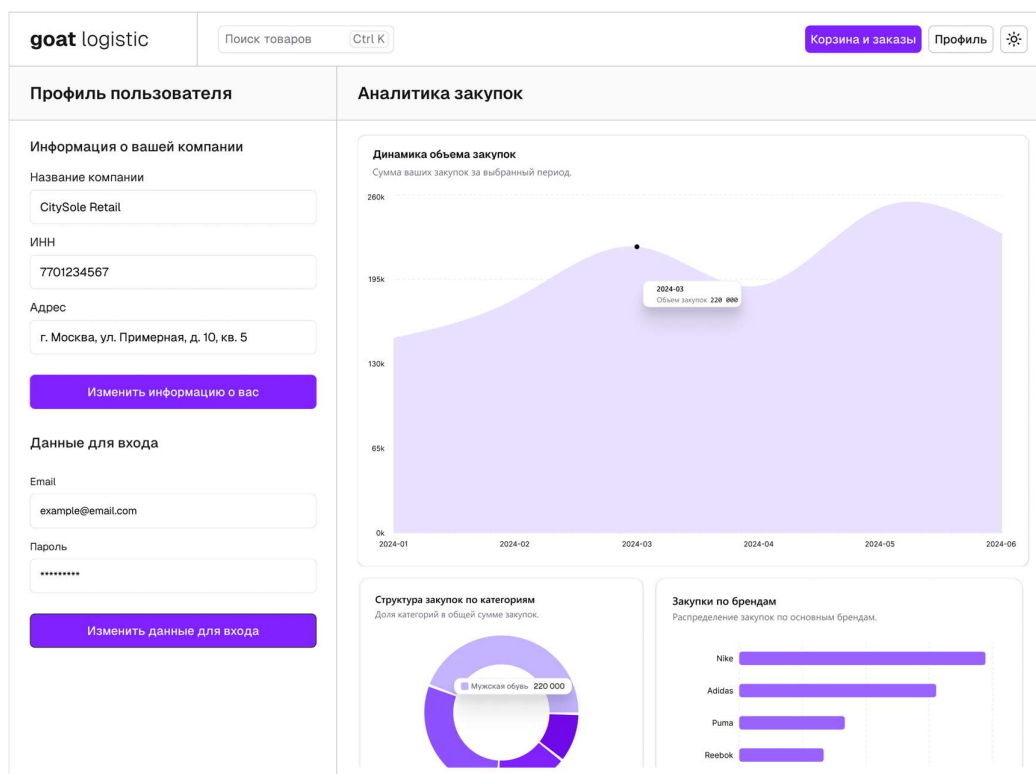


Рисунок 14 – Страница с профилем пользователя роли «Потребитель»

Следующим классом пользователей, для которого был спроектирован интерфейс, является «Поставщик» или представитель завода, который изготавливает и поставляет товары. Основной задачей данной роли является управление каталогом предлагаемой продукции и формирование поставок на центральный склад системы.

Ключевая страница для этой роли – раздел «Мои товары». На этой странице представлен полный список созданных товаров, с отображением их статуса, цены и основных материалов. Для каждого товара детализированы доступные варианты (цвет, размер, количество на складе системы). Интерфейс позволяет пользователю создавать, редактировать, и удалять каждую товарную позицию. Процесс добавления или редактирования реализован через модальное окно. В нем пользователь заполняет всю необходимую информацию об основном товаре. Также особое внимание уделено управлению вариантами товара: есть возможность добавления нескольких модификаций товара с указанием их параметров. После заполнения всех данных товар отправляется на

модерацию администратору системы. Страница с товарами поставщика представлена на рисунке 15.

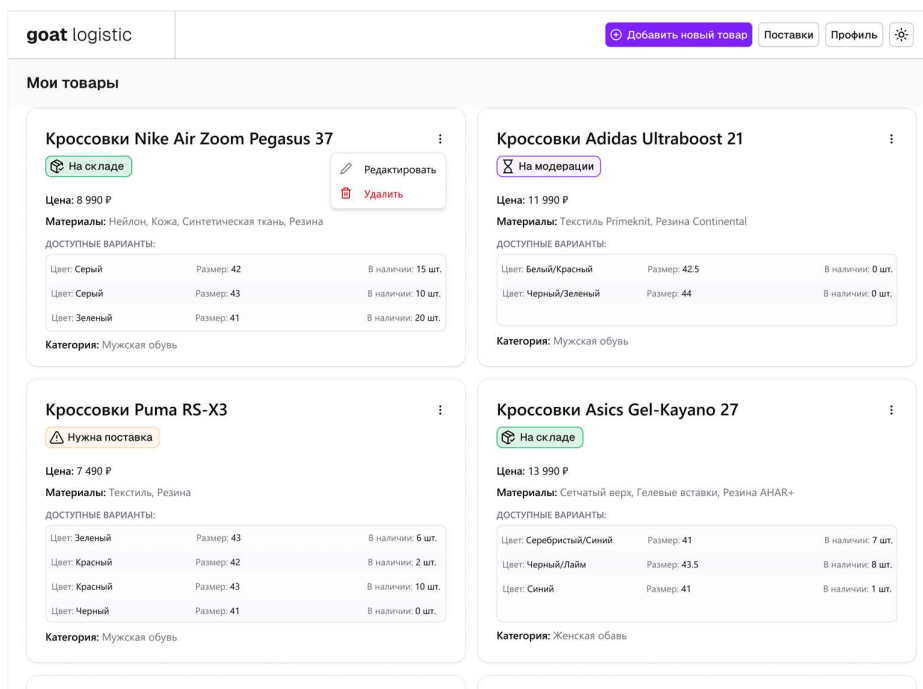


Рисунок 15 – Страница «Мои товары»

Модальное окно для добавления нового товара представлено на рисунке 16.

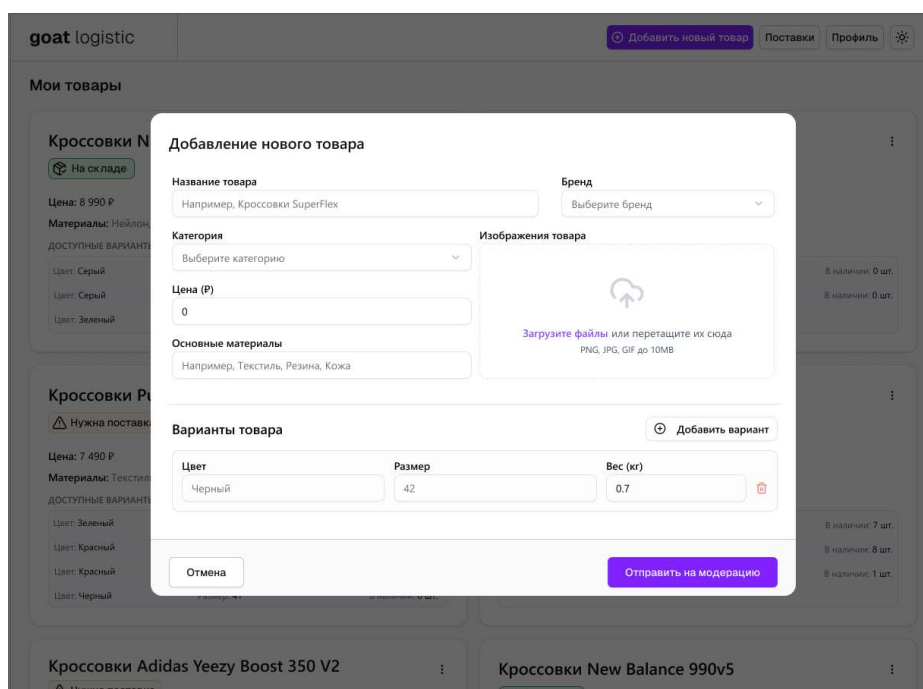


Рисунок 16 – Модальное окно «Добавление нового товара»

Для управления отправкой товаров на склад и оперативного реагирования на потребности системы спроектирована страница «Управление поставками», представленная на рисунке 17. Интерфейс также разделен на две части. В левой части поставщику представлен список всех его активных вариантов товаров с указанием текущего количества каждого варианта на складе системы. Для каждой позиции предусмотрен функционал по формировании поставки: пользователь может указать необходимое количество товара и инициировать их отправку на склад. Правая часть отображает срочные запросы на поставку. Здесь выводятся варианты товаров, количество которых на складе достигло критически низкого уровня или нуля. Такие варианты товаров можно сразу же пополнить, сформировав поставку.

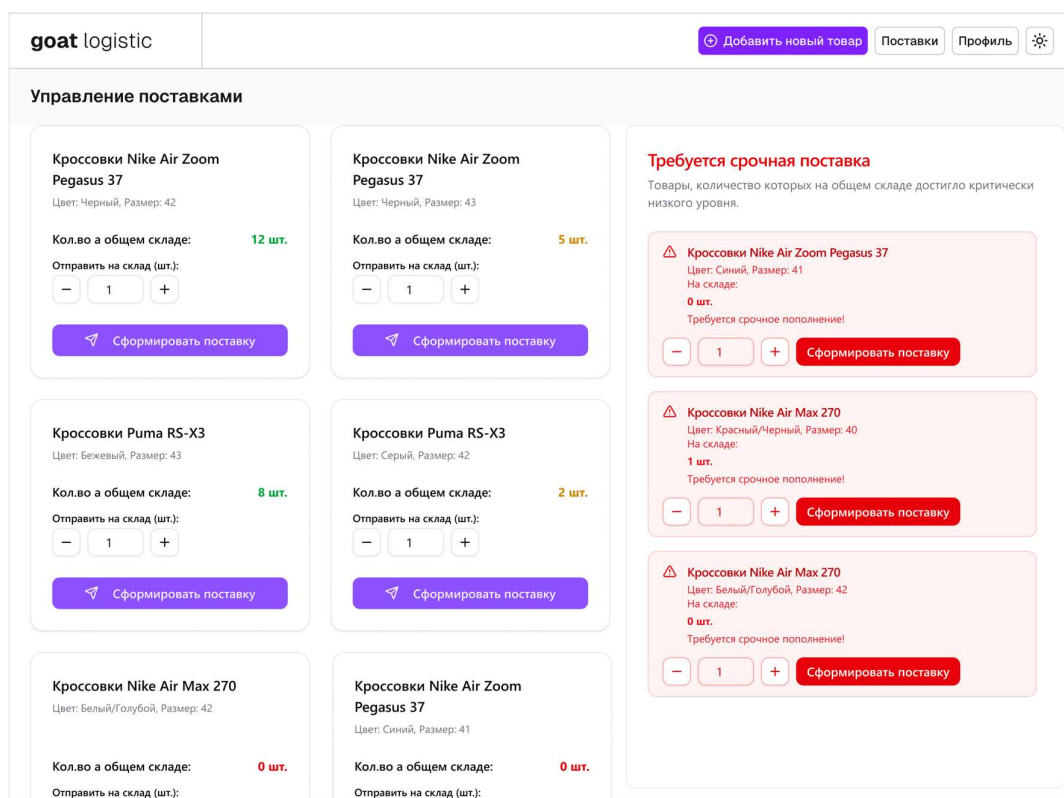


Рисунок 17 – Страница для управления поставками

Аналогично рабочей зоне потребителя, для поставщика предусмотрен профиль пользователя, и в нем также скомбинированы панели управления личными данными и доступ к аналитике. Управление личными данными ничем не отличается от профиля потребителя, а вот раздел «Аналитика продаж»

предназначен для отслеживания эффективности реализации товаров через сервис. Ключевой элемент – интерактивный график, отображающий динамику объемов продаж за выбранный период. Дополнительно представлен детализированный отчет по продажам товаров в табличном виде за каждую дату, когда были совершены закупки товаров конкретного поставщика. Профиль поставщика представлен на рисунке 18.

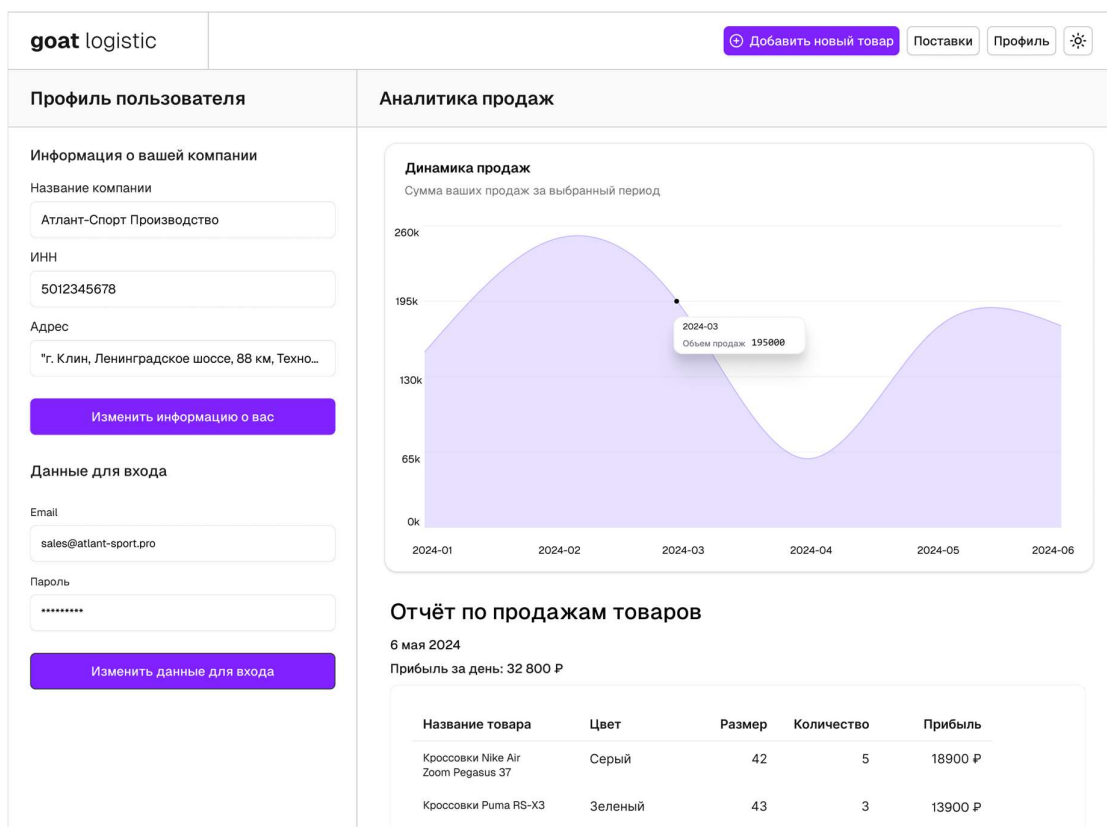


Рисунок 18 – Страница с профилем пользователя роли «Поставщик»

Интерфейс для пользователя с классом «Администратор» спроектирован как центральный пункт управления всеми аспектами логистического сервиса. Главная страница административной панели предоставляет основную информацию по ключевым системным метрикам. Есть секция «Быстрые действия», которая содержит ссылки для быстрого перехода на основные страницы управления и мониторинга. Главная страница представлена на рисунке 19.

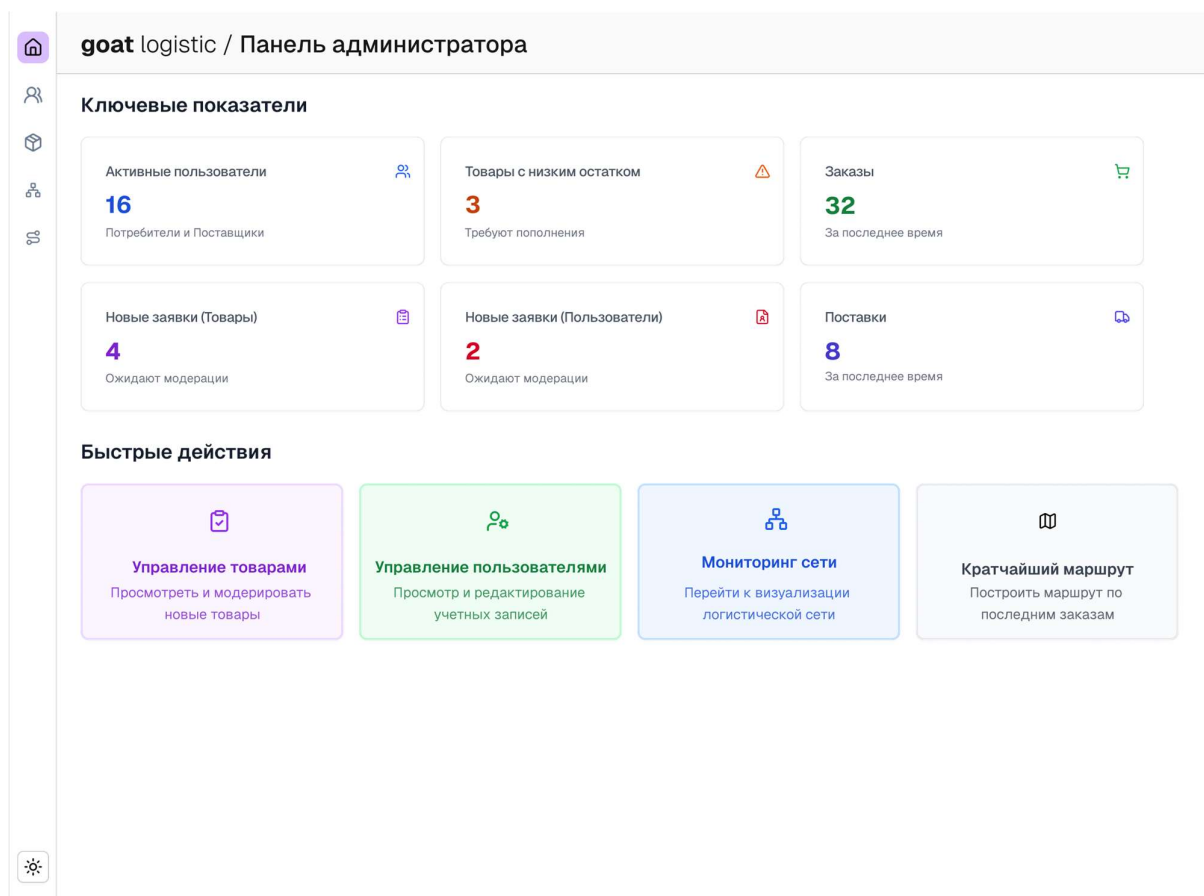


Рисунок 19 – Главная страница административной панели

Для администрирования ключевых сущностей системы спроектированы страницы: «Управление пользователями» и «Управление товарами». Первая из них, предназначенная для работы с учетными записями, предоставляет администратору список всех пользователей системы в виде информативных карточек с их данными и ролью. Администратор может удалить учетную запись пользователя. Правая часть страницы отведена под модерацию заявок на регистрацию новых пользователей. Вторая страница, «Управление товарами», спроектирована по аналогичному принципу. Здесь администратору доступен полный список всех товаров с их атрибутами и текущими статусами. Каждая товарная карточка также имеет функцию удаления, обеспечивая необходимый контроль над каталогом. Кроме того, предусмотрена важная функция модерации заявок на добавление новых товарных позиций от поставщиков. Страница «Управление пользователями» представлена на рисунке 20.

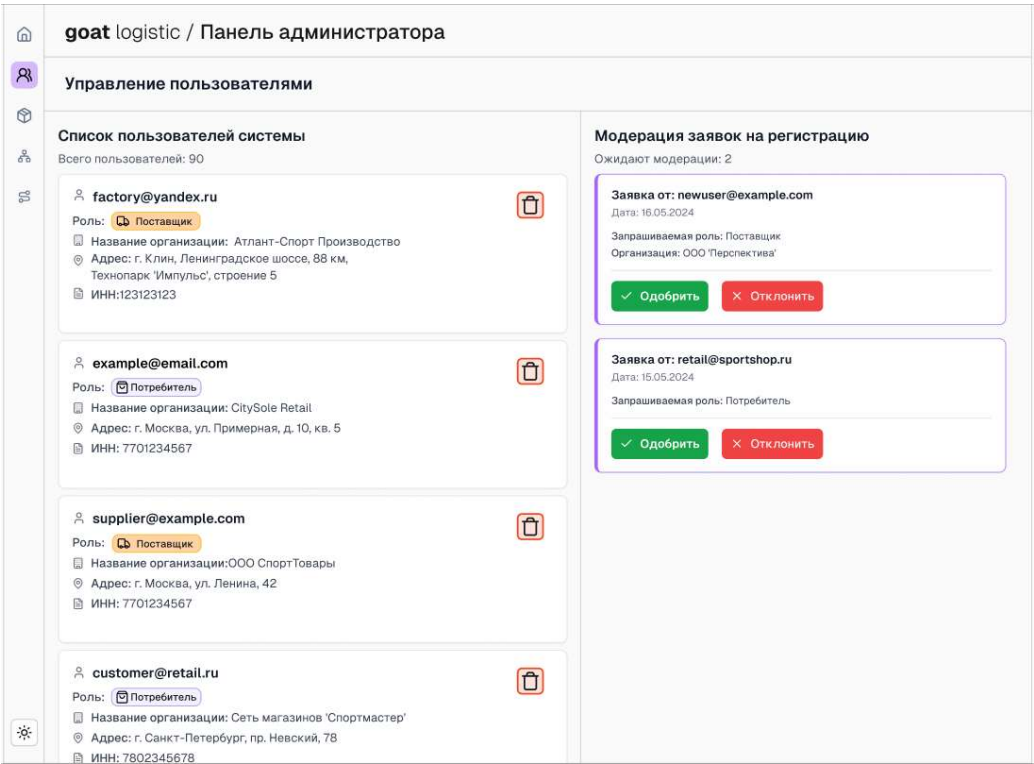


Рисунок 20 – Страница «Управление пользователями»

Страница с управлением товаров представлена на рисунке 21.

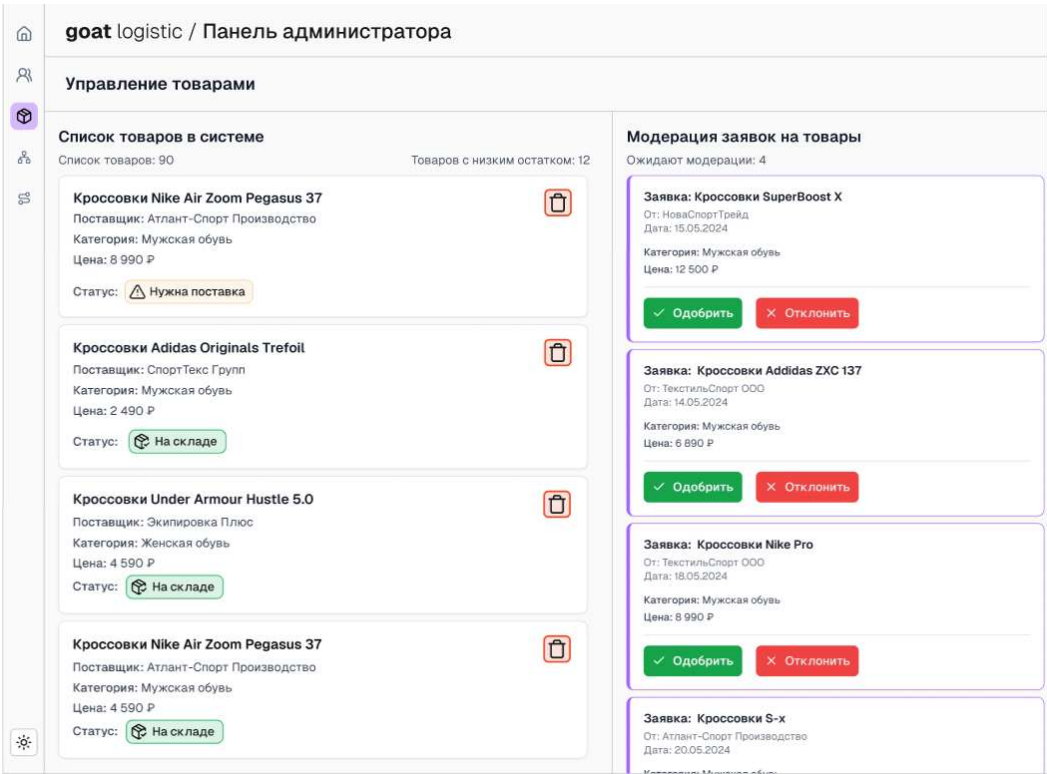


Рисунок 21 – Страница «Управление товарами»

Основные инструменты мониторинга в панели администратора реализованы через визуализацию данных на графовых моделях, что решает две ключевые задачи: мониторинга логистической сети и построения кратчайшего маршрута через заданные пункты. На графе отображается структура всей логистической сети из набора узлов и связей между ними [5].

Страница «Мониторинг логистической сети» предназначена для визуального представления структуры всей логистической системы в виде интерактивной графовой модели. В центральной части экрана отображается граф, где узлами являются «Центральный Склад», «Заводы» и «Магазины», а ребрами – связи между ними. В левой части страницы располагается информационная панель, отображающая детали по выбранному на графе элементу, легенду с описанием типов узлов и связей, а также инструменты для управления масштабом отображаемых элементов графа. Страница для мониторинга логистической сети представлена на рисунке 22.

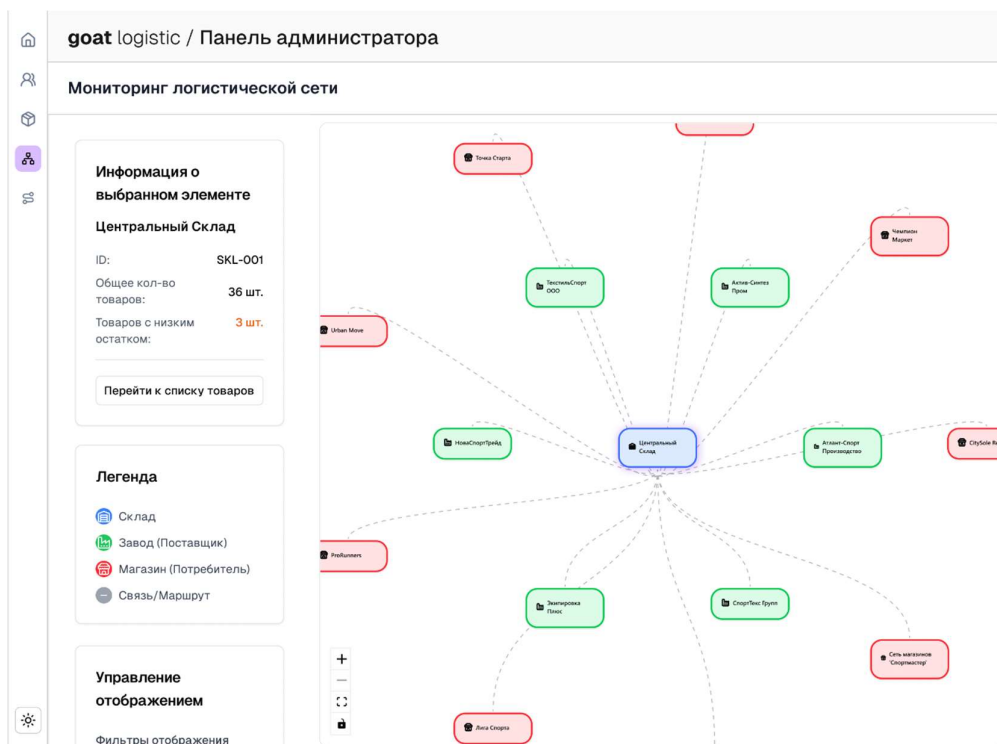


Рисунок 22 – Страница с мониторингом логистической сети

Для анализа и отображения оптимальных путей доставки товаров спроектирована страница «Поиск кратчайшего маршрута», представленная на

рисунке 23. На этой странице администратору доступна визуализация построения маршрута на стилизованной карте, где отмечены ключевые точки (склад, заводы, магазины) и последовательность их посещения. В правой части экрана представлена детализация маршрута: данные по количеству заказов и поставок, а также таблица с упорядоченным списком всех точек маршрута, их типов и адресов.

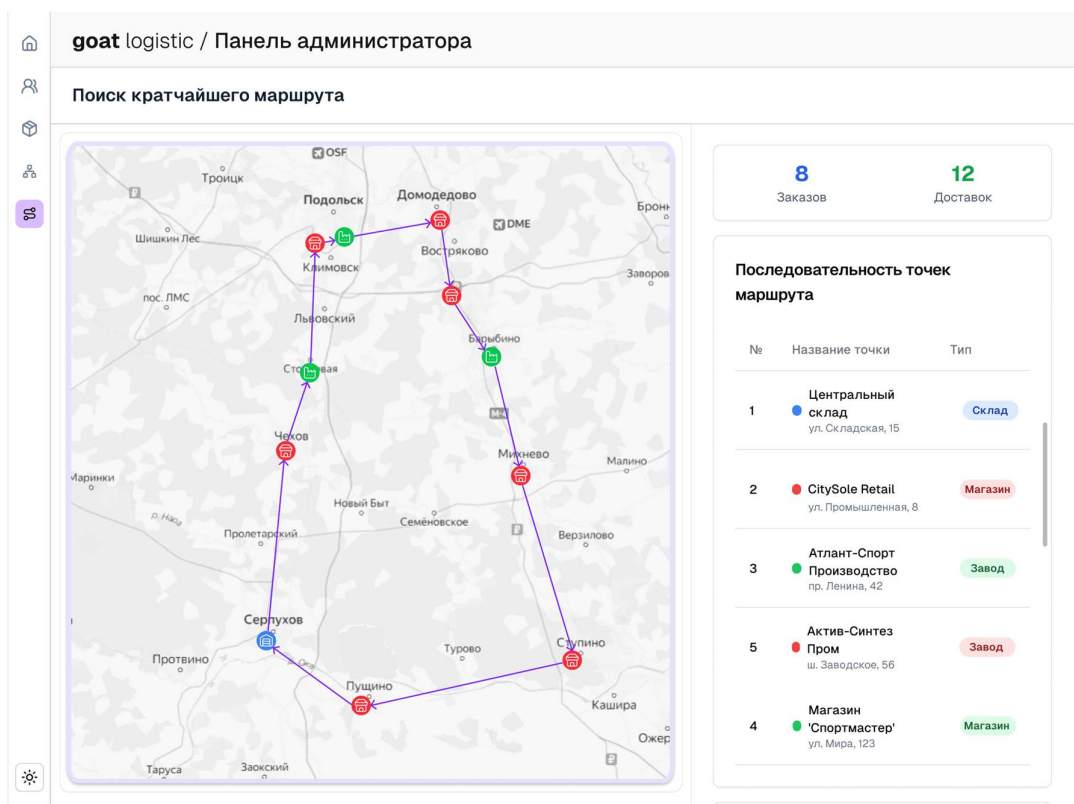


Рисунок 23 – Страница с поиском кратчайшего маршрута

Таким образом, в рамках данного раздела был спроектирован пользовательский интерфейс клиентского приложения для логистического сервиса по всем ключевым функциям для каждого класса пользователей. Также особое внимание было уделено созданию интуитивно понятных инструментов отображения данных и визуализации структуры логистической сети и логистических маршрутов. На основе спроектированного интерфейса будет строиться последующая реализация клиентского приложения.

3 Реализация клиентского приложения для логистического сервиса

3.1 Разработка архитектуры и структуры клиентского приложения

Клиентская часть логистического сервиса реализована как одностраничное SPA приложение на базе библиотеки React. Такой подход обусловлен способностью React эффективно управлять динамическим отображением данных и создавать сложные интерактивные интерфейсы, что критически важно для многофункциональности сервиса. SPA-подход позволяет обновлять только необходимые части страницы без полной перезагрузки, обеспечивая плавность и отзывчивость взаимодействия для пользователя [13].

В основе архитектуры приложения лежит компонентный подход React и модульный принцип организации кода. Проект разделен на три основных слоя: `app` (инициализация и глобальные сущности приложения), `features` (функциональные модули) и `shared` (переиспользуемый атомарный код). Каждый модуль в директории `features` (например, аутентификация, управление каталогом товара, административная панель) представляет собой независимый блок, инкапсулирующий собственную логику, `ui`-компоненты, модели данных, и, при необходимости, страницы. Ключевым аспектом является четкое разделение функциональности на независимые модули и общие, переиспользуемые компоненты и утилиты. Для каждого функционального модуля определен свой интерфейс взаимодействия с другими модулями. При этом контролируется направление зависимостей: например, общие элементы из `shared` могут использоваться в `features` и `app`, но не наоборот. Функциональные модули разработаны так, чтобы минимально зависеть друг от друга. Слабая связанность между модулями упрощает их независимую разработку, тестирование и последующее сопровождение, она является одним из принципов написания чистого кода. А главное, что подход с использованием данной архитектуры значительно снижает данную связность кода внутри проекта, повышает его переиспользуемость и модульность [14].

Файловая структура проекта выглядит следующим образом:

- `app` – корневой слой приложения с точкой входа, глобальными компонентами, конфигурацией и маршрутизацией;
- `app/main.tsx` – точка входа в приложение;
- `features` – набор функциональных модулей, где каждый модуль имеет свою внутреннюю структуру;
 - `features/ui` – компоненты пользовательского интерфейса, специфичные для конкретного модуля;
 - `features/model` – логика работы с данными, хранилище состояния модуля;
 - `features/pages` – страницы приложения, относящиеся к модулю;
 - `features/index.ts` – публичный интерфейс модуля, экспортирующий только то, что должно быть доступно для внешнего использования;
- `shared` – общий переиспользуемый код, доступный в любом месте приложения;
 - `shared/kit/ui` – общие атомарные UI-компоненты;
 - `shared/api` – общие функции взаимодействия с серверной частью приложения;
 - `shared/lib` – вспомогательные функции, константы.

Для сборки проекта и быстрой разработки клиентского приложения используется сборщик приложений Vite. Он значительно ускоряет процесс работы с кодом благодаря мгновенному обновлению изменений в браузере, что особенно удобно при модульной структуре проекта. Vite также эффективно подготавливает приложение к финальной загрузке приложения на сервер для доставки его пользователям.

Диаграмма развертывания, иллюстрирующая логическую структуру и взаимодействие основных архитектурных слоев клиентского приложения для логистического сервиса, представлена на рисунке 24.

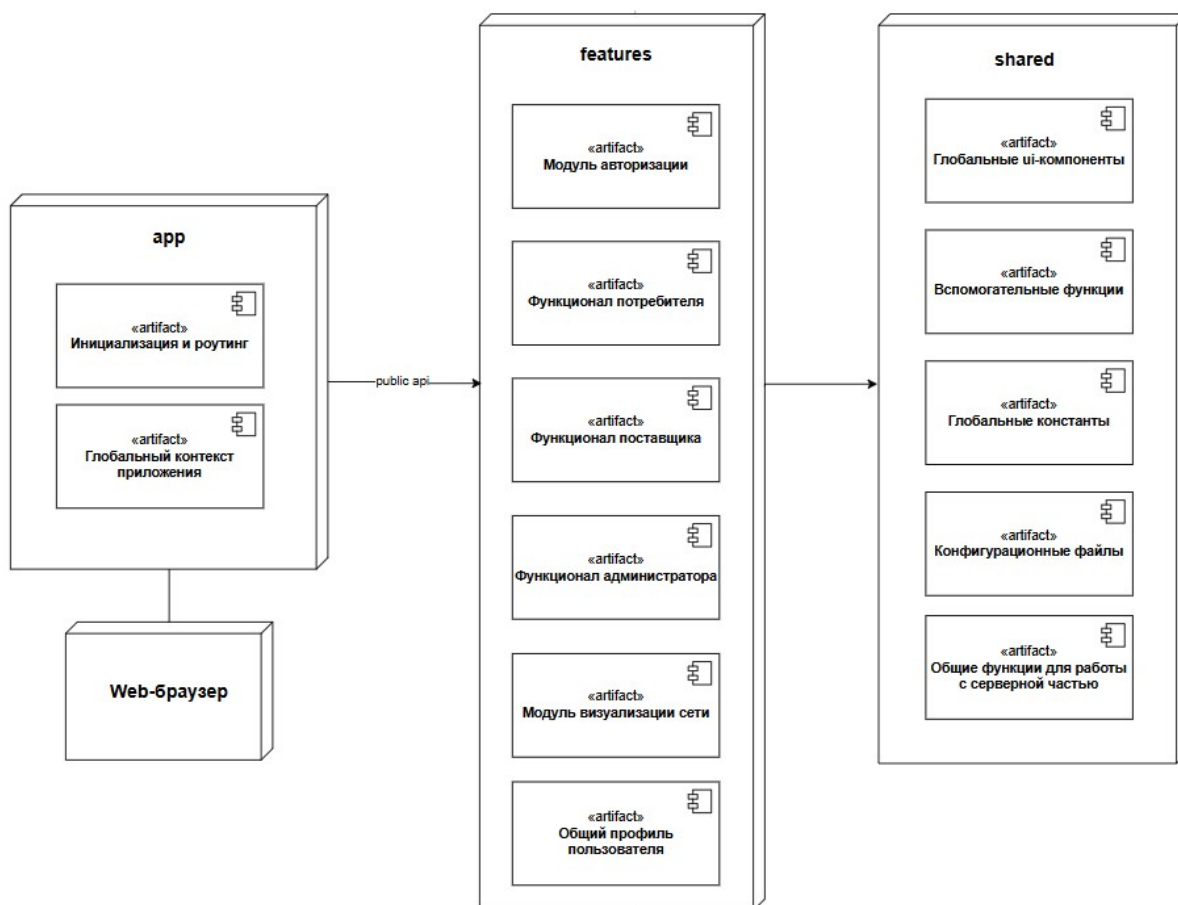


Рисунок 24 – Архитектура клиентского приложения для логистического сервиса

Таким образом, разработанная архитектура и структура проекта обеспечивают прочную и гибкую основу для дальнейшей реализации функционала клиентского приложения.

3.2 Разработка пользовательского интерфейса

3.2.1 Разработка компонентной базы приложения и стилизация пользовательского интерфейса

Внешний вид спроектированного ранее клиентского приложения реализовывался на основе компонентов React. Для построения интерфейса использовалась библиотека shadcn/ui. Она представляет собой коллекцию переиспользуемых UI-элементов, которые интегрируются в кодовую базу

проекта. Благодаря этому их можно легко стилизовать и кастомизировать. Стилизация данных компонентов и общая верстка страниц осуществляется с помощью Tailwind – утилитарного CSS-фреймворка, который позволяет определять стили через набор готовых классов в разметке компонентов [15].

Интерфейсы рабочих областей для всех классов пользователей были сформированы комбинаций элементов из shadcn/ui и самописных React-компонентов. Например, для отображения списков сущностей – товаров или пользователей, часто применялась сеточная верстка из компонентов-карточек (ProductCard, UserCard). Эти карточки могли содержать более мелкие компоненты из shadcn/ui, например, кнопки или значки. Формы ввода данных также строились на базе стандартных компонентов Input, Select, Checkbox и Button для обеспечения визуального единообразия.

Пример того, как стандартные элементы из shadcn/ui (Card) сочетаются с самописными React-компонентами (ProductCardContent) для отображения списка товаров представлен в листинге 1.

Листинг 1 – Комбинирование shadcn/ui элементов и самописных компонентов

```
<div className="lg:col-span-2 space-yx-4">
  <div className="space-yx-2">
    <h2 className="text-6xl font-semibold">Список товаров в
системе</h2>
    <p className="text-sm text-gry-600">Всего товаров:
{products.length}</p>
  </div>
  <Card>
    <CardContent className="pz-0">
      <ScrollArea className="h-458">
        <div className="pz-4 space-yx-4">
          {products.map((product) => (
            < ProductCardContent
              key={product.id}
              product={product}
              onDelete={handleDeleteProduct}
            />
          ))}
        </div>
      </ScrollArea>
    </CardContent>
  </Card>
</div>
```

Подход к построению интерфейса, сочетающий использование готовой библиотеки UI-элементов и разработку собственных компонентов, позволил достичь необходимой гибкости и скорости при реализации спроектированного дизайна.

3.2.2 Реализация модуля визуализации логистической сети

Для наглядного представления структуры логистического сервиса и взаимодействия его ключевых элементов, был разработан модуль визуализации на основе библиотеки React Flow, выбранной во время анализа средств визуализации данных.

Построение графа начинается с получения данных о пользователях и информации о центральной части системы с сервера. Эти данные преобразуются в два массива: массив узлов и массив ребер. Каждый узел – это либо склад, либо завод, либо магазин, характеризуется своим идентификатором, типом, определяемым ролью пользователя, и начальной позицией на холсте. Также у узла присутствуют некоторые метаданные, например, его название. Ребра формируются для визуализации логистических связей узлов с центральной частью системы. При добавлении или удалении пользователей в системе граф автоматически перестраивается, отражая актуальную структуру сети [5].

Особое внимание при реализации было уделено внешнему виду узлов для улучшения визуального восприятия и навигации по графу. Для этого были разработаны React-компоненты для каждого типа узла (склад, завод, магазин). Один из таких компонентов представлен на листинге 2, где тип узла определяется через переданное свойство data. Такой подход с различными цветовыми темами и иконками позволяет администратору мгновенно идентифицировать элементы сети и облегчает общий анализ структуры.

Листинг 2 – Пример написания узла для графовой модели

```
import React from 'react';
import { Handle, Position } from '@xyflow/react-flow';

function CustomGraphNode({ data }) {
```

```

    let nodeStyle = { background: '#E0E0E0', border: '1px solid
#B0B0B0' };
    if (data.type === 'factory') {
      nodeStyle = { background: '#D1FAE5', border: '1px solid
#10B981' };
    } else if (data.type === 'shop') {
      nodeStyle = { background: '#FEE2E2', border: '1px solid
#EF4444' };
    } else if (data.type === 'warehouse') {
      nodeStyle = { background: '#DBEAFE', border: '1px solid
#3B82F6' };
    }

    return (
      <div style={{ padding: '10px', borderRadius: '5px', minWidth:
'120px', textAlign: 'center', ...nodeStyle }}>
        <Handle type="target" position={Position.Top} id="t" />
        <div>{data.label}</div>
        <Handle type="source" position={Position.Bottom} id="s" />
      </div>
    );
  }
}

export default CustomGraphNode;

```

Важной особенностью реализации также является интерактивность графа. Пользователю доступны функции масштабирования холста для удобного исследования сети. Реализована обработка события по клику на узел: при выборе элемента пользователем, его данные используются для отображения специфичной информации в отдельной панели интерфейса. Управление состоянием узлов и ребер, включая их позицию и выбранные элементы, осуществляются с помощью встроенных механизмов библиотеки React, обеспечивая синхронизацию данных и корректное отображение графа. Так и получается модуль для информативной и интерактивной визуализации структуры логистической сети через графовые модели [16].

3.2.3 Реализация отображения кратчайшего маршрута

Для визуализации кратчайших логистических маршрутов был разработан соответствующий модуль. Он отображает последовательность посещения точек на основе их координатных данных, получаемых с сервера в заранее определенном порядке.

Основой для визуализации служит статическое изображение карты, используемое в качестве фона для рабочей области, где будет отображаться маршрут. Данные о точках маршрута, их координаты (X, Y) и порядок следования поступают с сервера. Они также преобразуются в массивы узлов и ребер для интеграции с React Flow. Координаты позволяют точно позиционировать узлы поверх фонового изображения, а для визуального различия типов точек (склад, заводы, магазины) применяются свои React-компоненты, специфичные для данной страницы, но работающие по такому же принципу, как при создании логистической сети.

Ребра графа React Flow формируются для соединения точек маршрута в порядке, указанном сервером, так как именно на нем заранее определяется оптимальность маршрута. Таким образом, получаются четко отрисованные линии, которые образуют путь следования. В листинге 3 представлен пример функции, создающей объект ребра для соединения двух точек маршрута.

Листинг 3 – Пример функции создания объекта ребра для маршрута

```
interface RoutePointNode {
  id: string;          // ID узла (точки)
}

const createRouteSegmentEdge = (
  sourceNode: RoutePointNode,
  targetNode: RoutePointNode,
  segmentOrder: number // Порядок сегмента в маршруте
): Edge => {
  return {
    id: `route-edge-${sourceNode.id}-${targetNode.id}`,
    source: sourceNode.id,
    target: targetNode.id,
    type: 'smoothstep', // Тип линии ребра
    animated: true,     // Анимация для визуализации потока
    label: `Шаг ${segmentOrder}`, // Метка, отображающая этап
    маршрута
    style: { stroke: '#6D28D9', strokeWidth: 2 },
  };
};
```

Также при проектировании была предусмотрена дополнительная информация для модуля помимо карты. Так, в правой части отображается таблица с последовательностью точек маршрута. Информация для таблицы

формируется на основе тех же данных, что и для карты с графом. Базовая интерактивность обеспечивается стандартными возможностями React Flow [16].

3.3 Разработка средства взаимодействия с серверным приложением

Взаимодействие клиентского приложения с REST API серверной части построено на основе строго типизированного подхода с использованием спецификации OpenAPI. Это обеспечивает согласованность данных между клиентом и сервером.

Для выполнения HTTP-запросов к API применяется библиотека `openapi-fetch`. Она автоматически создает необходимые шаблоны для запросов на основе описания данных на сервере используя схему под названием OpenAPI версии три. Это позволяет формировать запросы и обрабатывать ответы с правильной типизированной структурой. В библиотеке есть основной инструмент – `fetchClient`, который достаточно настроить один раз, указав адрес сервера. Это значительно упрощает обращение к разным серверным маршрутам [17].

Пример настройки `fetchClient` в клиентском приложении представлен на листинге 4.

Листинг 4 – Настройка `fetchClient`

```
export const fetchClient = createFetchClient<ApiPaths>({
  baseUrl: CONFIG.API_BASE_URL,
});
export const rqClient = createClient(fetchClient);
```

Для работы с данными, которые приходят с сервера (при их получении, временном хранении и обновлении), используется библиотека `tanstack/react-query`. Чтобы эта библиотека эффективно взаимодействовала с `openapi-fetch`, применяется подход, когда создаются готовые хуки – специальные инструменты для отправки запросов к конкретному маршруту на сервере.

Для получения данных с сервера используется хук `useQuery`, работающий совместно с `fetchClient`. Он не только загружает нужную информацию, но и предоставляет компонентам сведения о процессе загрузки и возможных ошибках. Эти ошибки очень легко можно отобразить в интерфейсе. Для

операций, которые изменяют данные на сервер применяется хук `useMutation`. Комбинация этих хуков позволяет автоматически обновлять данные на клиенте, чтобы пользователь всегда видел актуальную информацию [17].

Пример операции для создания товара, при котором автоматически обновится список товаров, представлен на листинге 5.

Листинг 5 – Оправка запроса на сервер и обновление интерфейса

```
const addProductMutation = rqClient.useMutation(
  "post",
  "/products",
  {
    onSuccess: () => {
      queryClient.invalidateQueries(["products"]);
    },
  },
);

addProductMutation.mutate(newProductData);
```

Такой подход с использованием различных библиотек для формирования шаблонов запроса и их вызова использовался для всех маршрутов с сервера.

3.4 Разработка системы управления состоянием

При разработке интерактивных одностраничных приложений с помощью React нужно правильно управлять состоянием данных. Оно определяет как текущее отображение данных в пользовательском интерфейсе, так и поведение приложения в ответ на действия пользователя или события системы. Корректная организация состояния напрямую влияет на предсказуемость поведения приложения и удобство его дальнейшей поддержки и расширения [18].

В рамках реализации клиентской части логистического сервиса для управления состоянием использовались встроенные механизмы из самой библиотеки React. Основные инструменты – это хук `useState` для управления локальным состоянием отдельных компонентов и `React Context API` с хуком `useContext` для работы с данными, которые используются в разных частях приложения.

Управление локальным состоянием используется для хранения и обновления данных, которые специфичны только для конкретного компонента и не требует доступа из разных частей приложения. Это могут быть, например, элементы управления значений полей ввода в формах, отслеживание состояний интерактивных компонентов, модальных окон или временных данных [18].

Если нужно разделяемое состояние, например в сложных интерактивных модулях визуализации данных, применяется React Context API. С помощью него можно передавать массивы ребер и узлов или информацию о выбранном элементе через дерево компонентов на несколько уровней вниз. Так, например был создан Context для визуализации логистической сети. В нем инкапсулируется логика хранения и обновления данных графа, а также предоставляются функции для взаимодействия с графом. Это позволяет различным компонентам – холсту графа и отдельным информационные панелям, оперировать единым и синхронизированным набором данных о состоянии сети. Компоненты, которые отвечают за отображения самого графа React Flow, и информационные панели используют хук useContext для доступа к синхронизированным данным.

В листинге 6 представлен пример создания и использования контекста для управления состоянием узлов графа в модуле визуализации.

Листинг 6 – Пример использования контекста в модуле визуализации

```
import React, { createContext, useState, useContext, ReactNode }
from 'react';
import { Node } from '@xyflow/reactflow';

interface GraphState {
  nodes: Node[];
  setNodes: React.Dispatch<React.SetStateAction<Node[]>>;
  selectedNodeId: string | null;
  setSelectedNodeId: (id: string | null) => void;
}

const GraphContext = createContext<GraphState |
undefined>(undefined);

export const GraphProvider = ({ children, initialNodes = [] }: {
  children: ReactNode, initialNodes?: Node[] }) => {
  const [nodes, setNodes] = useState<Node[]>(initialNodes);
```

```

    const [selectedNodeId, setSelectedNodeId] = useState<string |
null>(null);

    return (
      <GraphContext.Provider value={{ nodes, setNodes,
selectedNodeId, setSelectedNodeId }}>
        {children}
      </GraphContext.Provider>
    );
  };

export const useGraph = (): GraphState => {
  const context = useContext(GraphContext);
  if (context === undefined) {
    throw new Error('useGraph must be used within a
GraphProvider');
  }
  return context;
};

```

В примере создается `GraphContext`, который управляет состоянием узлов и идентификатором выбранного узла, а состояние графа становится доступным и управляемым для всех частей модуля визуализации [16].

4 Контроль качества программного обеспечения

4.1 Обоснование выбора модели тестирования и метрик кода

Обеспечение качества программного средства – это неотъемлемая часть процесса разработки, особенно для клиентского приложения логистического сервиса, которое служит основной точкой взаимодействия пользователя с системой. Качественное клиентское приложение гарантирует корректное выполнение операций, удобство использования и общую надежность сервиса. Для достижения этих целей в рамках данной работы применялся подход, сочетающий различные виды тестирования и анализ метрик исходного кода.

Выбор конкретных методов тестирования был обусловлен необходимостью проверки разных аспектов работы клиентского приложения. В рамках контроля качества были использованы следующие модели тестирования:

- функциональное тестирование для проверки соответствия реализованных функций приложения заявленным требованиям, определенных на этапе анализа;
- модульное тестирование для проверки работоспособности отдельных, изолированных фрагментов кода – компонентов, утилитарных функций [19].

Также были применены метрики Холстеда для количественной оценки характеристик разработанного кода и его сложности. Данные метрики позволяют получить объективные показатели длины программы, ее словаря, вычислительные сложности и сложности программы. Они основаны на подсчете уникального и общего количества операторов и операндов в программе.

Комплексное применение описанных методов тестирования и анализа метрик кода направлено на достижение высокого уровня качества разработанного клиентского приложения для логистического сервиса, его надежности и соответствия ожиданиям пользователей.

4.2 Тестирование клиентского приложения для логистического сервиса

4.2.1 Функциональное тестирование программного средства

После реализации программных средств, проведено функциональное тестирование.

Функциональное тестирование является важной частью процесса обеспечения качества программного обеспечения. Оно направлено на проверку того, соответствует ли программное обеспечение заявленным требованиям и спецификациям.

Для проведения функционального тестирования был выбрана форма для регистрации организации. Форма состоит из двух полей для ввода ИНН и названия организации. В поля ввода предусмотрена валидация данных по некоторым условиям.

Для данной формы выявлены следующие требования:

- программа должна сообщать об некорректной длине ИНН;
- программа должна сообщать о том, что ИНН состоит не из цифр;
- программа должна сообщать о том, что ИНН состоит не из цифр;
- программа должна сообщать об некорректной длине названия организации;
- программа должна сообщать об некорректных символах при вводе названия организации.

Были определены причины и следствия для работы данного класса. Они представлены в таблице 5.

Таблица 5 – Причины и следствия

Причины	
П1	Корректная длина ИНН организации (10 символов)
П2	ИНН состоит только из цифр
П3	ИНН уникален (не занят)

По данной диаграмме составлена таблица решений для всех возможных вариантов поведения, в зависимости от правильности выполнения каждого метода.

Таблица 6 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12
П1	0	0	0	1	1	1	1	1	1	1	1	1
П2	0	0	0	0	0	0	1	1	1	1	1	1
П3	0	0	0	0	0	0	0	0	0	1	1	1
П4	0	1	1	0	1	1	0	1	1	0	1	1
П5	0	0	1	0	0	1	0	0	1	0	0	1
С1	1	1	1	0	0	0	0	0	0	0	0	0
С2	0	0	0	1	1	1	0	0	0	0	0	0
С3	0	0	0	0	0	0	1	1	1	0	0	0
С4	1	0	0	1	0	0	1	0	0	1	0	0
С5	0	1	0	0	1	0	0	1	0	0	1	0
С6	0	0	0	0	0	0	0	0	0	0	0	1

На основе таблицы была выявлена необходимость в тестировании следующих вариантов поведения, выявления ошибок и проверки реакции системы на возникающие ситуации. План тестирования представлен в таблице 7.

Таблица 7 – Тест-кейсы

№	Входные данные	Ожидаемый результат
1	ИНН: 123456789, Название: ОО	Некорректная длина ИНН, Некорректная длина названия
2	ИНН: 12345678901, Название: ООО Тест#	Некорректная длина ИНН, Недопустимые символы в названии

Продолжение таблицы 7

№	Входные данные	Выходные данные
3	ИНН: 12345, Название: ООО «Ромашка»	Некорректная длина ИНН
4	ИНН: 12A3456789, Название: PU	ИНН может содержать только цифры, Некорректная длина названия
5	ИНН: 12C3456789, Название: ООО \$Компания	ИНН может содержать только цифры, Недопустимые символы в названии
6	ИНН: 123B567890, Название: Reebok Classic	ИНН может содержать только цифры
7	ИНН: 1122334455, Название: К	Этот ИНН уже занят, Некорректная длина названия
8	ИНН: 5544332211, Название: Магазин%	Этот ИНН уже занят, Недопустимые символы в названии
9	ИНН: 6677889900, Название: ООО Весна	Этот ИНН уже занят
10	ИНН: 7744001122, Название: VF	Некорректная длина названия
11	ИНН: 9988776655, Название: ООО 'Старт+	Недопустимые символы в названии
12	ИНН: 1122334455, Название: ИП Иванов	Организация успешно создана

Программа успешно прошла все функциональные тесты. Поведение программы соответствует требованиям.

4.2.2 Модульное тестирование метода отправки метаданных

Для обеспечения корректности работы разработанного приложения и проверки ожидаемого поведения программы проведено модульное тестирование.

Тестирование утилиты для работы с CSS стилями и классами, разработанного на языке TypeScript, проводится с помощью написания Unit-тестов на Vitest. Исходный код теста представлен в листинге 7.

Листинг 7 – Unit-тест для утилиты cn

```
describe('Утилита cn', () => {
  test('должна объединять простые строковые классы', () => {
    const result = cn('class1', 'class2', 'class3');
    expect(result).toBe('class1 class2 class3');
  });

  test('должна удалять дублирующиеся классы', () => {
```

```

    const result = cn('p-4', 'p-4', 'text-red-500');
    expect(result).toBe('p-4 text-red-500');
  });

  test('должна корректно обрабатывать условные классы', () => {
    const isActive = true;
    const isDisabled = false;

    const result = cn('button', isActive && 'active', isDisabled
    && 'disabled');

    expect(result).toBe('button active');
  });

  test('должна правильно объединять tailwind классы с
  конфликтами', () => {
    const result = cn('p-4 text-red-500', 'p-6 text-blue-700');
    expect(result).toBe('p-6 text-blue-700');
  });

  test('должна корректно обрабатывать объекты с классами', () => {
    const result = cn(
      {
        'bg-blue-500': true,
        'text-white': true,
        'rounded-lg': false,
        'p-4': true,
      },
      'flex items-center',
    );

    expect(result).toBe('bg-blue-500 text-white p-4 flex items-
    center');
  });
});

```

Результаты проведения тестов представлены в таблице 8.

Таблица 8 – Результаты модульного тестирования

	Входные данные	Ожидаемый результат	Фактический результат (предполагается)	Статус теста
1	'class1', 'class2', 'class3'	Утилита cn должна объединять простые строковые классы: 'class1 class2 class3'	Утилита cn объединила все простые строковые классы: 'class1 class2 class3'	Пройден
2	'p-4', 'p-4', 'text-red-500' (дублирующиеся классы)	Утилита cn должна удалять дублирующиеся классы: 'p-4 text-red-500'	Утилита cn должна удалила все дублирующиеся классы: 'p-4 text-red-500'	Пройден

Продолжение таблицы 8

	Входные данные	Ожидаемый результат	Фактический результат (предполагается)	Статус теста
3	'button', isActive && 'active', isDisabled && 'disabled' (isActive=true, isDisabled=false)	Утилита cn должна корректно обрабатывать условные классы: 'button active'	Утилита cn должна корректно обработала условные классы: 'button active'	Пройден
4	'p-4 text-red-500', 'p-6 text-blue-700' (конфликтующие Tailwind-классы)	Утилита cn должна правильно объединять tailwind классы с конфликтами: 'p-6 text-blue-700'	Утилита cn правильно объединила tailwind классы с конфликтами: 'p-6 text-blue-700'	Пройден
5	Объект: {bg-blue-500: true, 'text-white': true, 'rounded-lg': false, 'p-4': true}, 'flex items-center'	Утилита cn должна корректно обрабатывать объекты с классами: 'bg-blue-500 text-white p-4 flex items-center'	Утилита cn корректно обработал объекты с классами: 'bg-blue-500 text-white p-4 flex items-center'	Пройден

После выполнения юнит-тестов был получен положительный результат.

4.2.3 Метрики Холстеда кода класса тиражирования

Метрики Холстеда – это метрики, которые рассчитываются на основе анализа исходного кода программы и элементов синтаксиса [20].

В качестве примера расчета метрик рассматривается функция `createFetchClient`, отвечающий за запросы клиента к серверу. Код данного класса представлен в листинге 8.

Листинг 8 – Http клиент для запросов к серверу

```
export const BASE_URL = 'http://localhost:3000';

class ApiError extends Error {
  constructor(public response: Response) {
    super(`ApiError: ${response.status}`);
  }
}

export const createFetchClient = async <T>(url: string, init?:
RequestInit & { json?: unknown }) => {
  let headers = init?.headers ?? {};

  if (init?.json) {
    headers = {
```

```

        'Content-Type': 'application/json',
        ...headers,
    };

    init.body = JSON.stringify(init.json);
}
const response = await fetch(`${BASE_URL}${url}`, {
    ...init,
    ...headers,
});

if (!response.ok) {
    throw new ApiError(response);
}

const data = (await response.json()) as Promise<T>;
return data;
};

```

Для оценки сложности необходимо подсчитать количество уникальных операторов и операндов и их общее число. Операторы данной функции представлены в таблице 9.

Таблица 9 – Операторы функции createFetchClient

Оператор	Число вхождений
export	2
=	6
class	1
extends	1
constructor	1
public	1
super	1
\${} (интерполяция строки)	1
async	1
const	4
let	1
? (опциональный параметр)	2
?? (оператор нулевого слияния)	1
if	1
: (типизация)	3
{}	3

Продолжение таблицы 9

Оператор	Число вхождений
... (оператор расширения)	3
await	2
! (отрицание)	1
throw	1
as (приведение типа)	1
return	1
.	7
JSON.stringify	1
new	1
response.json	1
fetch	1
n1= 26	N1=44

Операнды функции createFetchClient приведены в таблице 10.

Таблица 10 – Операнды функции createFetchClient

Операнд	Число вхождений
BASE_URL	3
'http://localhost:3000'	1
ApiError	2
Error	1
response	4
Response	1
status	1
fetchApiInstance	1
T	1
url	2
init	5
RequestInit	1
json	3
headers	4
'Content-Type'	1
'application/json'	1

Продолжение таблицы 10

Операнд	Число вхождений
body	1
JSON.stringify	1
ok	1
data	1
Promise	1
n2= 22	N2=33

Длина программы рассчитывается по формуле:

$$N = N_1 + N_2 = 26 + 22 = 48.$$

Словарь программы рассчитывается по формуле:

$$n = n_1 + n_2 = 44 + 33 = 77.$$

Вычислительная сложность рассчитывается по формуле:

$$HV = N * \log_2 n = 77 * 5.585 = 430.045.$$

Сложность программы рассчитывается по формуле:

$$HD = (n_1/2) * (N_2/n_2) = 12.5 * 1.478 = 18.478.$$

Комплексный показатель кода составляет 18.478.

Полученные показатели находятся в пределах нормы, соответствуя средним значениям поддерживаемости, и удовлетворяют требованиям по метрикам программного кода.

Заключение

В рамках выпускной квалификационной работы осуществлено проектирование и разработка клиентского приложения для логистического сервиса.

Данное приложение реализовано на языке TypeScript с использованием библиотеки React и других технологий для создания интерактивных и производительных пользовательских интерфейсов. Взаимодействие с серверной частью обеспечивается с помощью типизированного подхода на основе спецификации OpenAPI.

В данной работе проведен анализ предметной области, сравнительный анализ аналогичных логистических клиентских систем, а также выявлены и поставлены функциональные и нефункциональные требования к разработанному приложению. Выявлены цели и поставлены задачи для логистического сервиса и определены классы пользователей в системе.

В рамках организационной работы выполнено планирование этапов разработки и проведена оценка бюджета проекта. Общие трудозатраты на создание клиентского приложения составили 588 часов, а условная стоимость проекта оценена в 248900 рублей.

В ходе проектирования разработана архитектура клиентского приложения, определены сценарии взаимодействия пользователей с системой, спроектирована концептуальная модель. Также для всех страниц приложения под рабочую область каждого класса пользователей спроектирован интерфейс для последующей реализации.

Разработка клиентского части логистического сервиса включала в себя создание всех компонентов интерфейса по заданной архитектуре, реализацию взаимодействия с серверным API, а также систему управления состоянием всего приложения. Также разработаны модули визуализации на графовые моделях с использованием библиотеки React Flow.

Для обеспечения качества разработанного клиентского приложения проведено функциональное и модульное тестирование ключевых компонентов и функций. Также выполнен расчет метрик Холстеда для оценки сложности и характеристик исходного кода. Результаты тестирования подтвердили корректность работы приложения и его соответствие требованиям, а анализ метрик показал приемлемый уровень качества кода.

Таким образом, в результате выполнения выпускной квалификационной работы создано клиентское приложение для логистического сервиса. Данное приложение предоставляет удобный и наглядный интерфейс для всех классов пользователей, позволяет моделировать полный цикл логистических процессов и обеспечивает эффективный контроль над процессами благодаря развитым средствам мониторинга и визуализации данных.

Список использованных источников

1. Как e-commerce трансформирует логистику // Кампус [Электронный ресурс]. URL: <https://campus.stt.ru/articles/kak-e-commerce-transformiruet-logistiku> (дата обращения: 12.05.2025).
2. IT в логистике и складах: как технологии меняют рынок // РБК Компании [Электронный ресурс]. URL: <https://companies.rbc.ru/news/RCS4VLs4Ui/it-v-logistike-i-skladah-kak-tehnologii-menyayut-ryinok/> (дата обращения: 12.05.2025)
3. Гаджинский А. М. Логистика: Учебник. – 15-е изд., перераб. и доп. – М.: Издательско-торговая корпорация «Дашков и К°», 2007. – 472 с.
4. Уваров С. А. Логистика: общие основы теории : учеб. пособие для студентов вузов / С. А. Уваров. – СПб. : Изд-во СПбГУЭФ, 2009. – 223 с.
5. Емеличев В. А. Лекции по теории графов / В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич. – Москва : Наука, 1990. – 383 с.
6. Визуализация больших графов для самых маленьких // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/post/466303/> (дата обращения: 17.05.2025)
7. UML – диаграмма вариантов использования (use case diagram) // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/47940/> (дата обращения: 17.05.2025).
8. Медведев М.В., Антонов И.И., Балашова И.Ю. Визуализация сущностей логистического сервиса: отображение результата решения задачи маршрутизации в интерфейсе. // Вестник Пензенского государственного университета. 2024. С. 94 – 97. (Дата обращения: 18.05.2025)
9. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. –102 с
10. Пользовательские сценарии (User Scenarios) и их роль в проектировании интерфейсов // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/articles/542678/> (дата обращения: 22.05.2025).

11. Концептуальная модель [Электронный ресурс]. <https://fb.ru/article/144365/chto-takoe-kontseptualnaya-model> (дата обращения: 22.05.2025)
12. Веб-дизайн: книга Джесса Гарретта. Элементы опыта взаимодействия». – Пер. с англ. – СПб.: Символ-Плюс, 2008. – 192 с.
13. Прожариваем React // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/892406/> (дата обращения: 28.05.2025)
14. Архитектура front-end приложений – react, react native, angular. Обзор // Хабр. [Электронный ресурс]. URL: <https://habr.com/ru/articles/726354/> (дата обращения: 30.05.2025)
15. Анатомия shadcn/ui // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/timeweb/articles/781346/> (дата обращения: 01.06.2025)
16. Quickstart // React Flow [Электронный ресурс]. URL: <https://reactflow.dev/learn> (дата обращения: 02.06.2025)
17. Getting Started // OpenAPI Fetch [Электронный ресурс]. URL: <https://openapi-ts.dev/openapi-fetch/> (дата обращения: 03.06.2025)
18. Что такое состояние // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/ruvds/articles/706086/> (дата обращения: 04.06.2025)
19. Фундаментальная теория тестирования // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/549054/> (дата обращения: 05.06.2025)
20. Программный код и его метрики // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 05.06.2025)

Приложение А
(справочное)
Глоссарий

Клиентское приложение – часть программного комплекса, которая выполняется на стороне пользователя и отвечает за предоставление пользовательского интерфейса, взаимодействие с пользователем и обмен данными с серверной частью.

API (Application Programming Interface) – набор правил, протоколов и инструментов, который позволяет различным программным компонентам или системам взаимодействовать друг с другом.

HTTP – протокол передачи гипертекста. Набор правил, по которым данные в интернете передаются между разными источниками, обычно между компьютерами и серверами.

REST API (Representational State Transfer API) – архитектурный стиль для построения web-сервисов, основанный на протоколе HTTP. Характеризуется использованием стандартных HTTP-методов (GET, POST, PUT, DELETE) для взаимодействия с ресурсами, идентифицируемыми по URL.

React – JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов, особенно для одностраничных приложений (SPA). Основана на компонентном подходе, что позволяет создавать переиспользуемые и изолированные части интерфейса.

Одностраничное приложение (Single Page Application, SPA) – тип web-приложения, которое загружает одну HTML-страницу и динамически обновляет ее содержимое по мере взаимодействия пользователя с приложением, без необходимости полной перезагрузки страницы с сервера.

Компонент – независимый, переиспользуемый блок кода, отвечающий за определенную часть пользовательского интерфейса и ее логику. Компоненты могут быть вложены друг в друга для построения сложных интерфейсов.

Состояние – данные, которые определяют поведение и отображение компонента или всего приложения в React. Изменение состояния приводит к перерисовке соответствующей части интерфейса.

Хук – специальная функция, которая позволяет подключаться к возможностям React (таким как состояние и жизненный цикл) из функциональных компонентов.

React Flow – библиотека для React, предназначенная для создания интерактивных графовых и узловых диаграмм.

Пользовательский интерфейс (UI) – совокупность средств, при помощи которых пользователь взаимодействует с программой или устройством. Включает в себя визуальные элементы (кнопки, меню, формы) и логику их работы.

OpenAPI – спецификация для описания и документирования web-сервисов API.

Приложение Б
(обязательное)
Листинг клиентского приложения

Листинг Б.1 – Файл ShopPage.tsx

```
import { Filters } from "../(shop-components)/Filters/Filters";
import { ProductsCatalog } from "../(shop-components)/Catalog/ProductsCatalog";
import { CatalogSettings } from "../(shop-components)/Catalog/CatalogSettings";
import { CatalogContainer } from "../(shop-components)/Catalog/CatalogContainer";

export default async function Shop() {
  return (
    <>
      <aside className="w-[var(--filters-width)] py-4 px-6 border-r border-b border-t">
        <Filters />
      </aside>
      <CatalogContainer
        headerTitle="Товары"
        headerDetails={<CatalogSettings />}
      >
        <ProductsCatalog />
      </CatalogContainer>
    </>
  );
}
```

Листинг Б.2 – Файл ShopProduct.tsx

```
"use client";
import { useGetProductQuery } from "@/hooks/query/useGetProductQuery";
import { ProductLeftSide } from "../ProductLeftSide";
import { ProductRightSide } from "../ProductRightSide";

interface CatalogItemProps {
  productId: string;
}

export const Product = ({ productId }: CatalogItemProps) => {
  const { data: product, isLoading } = useGetProductQuery(+productId);

  return (
    <div className="px-6 flex justify-between gap-6 ">
      <ProductLeftSide product={product?.data}
        isLoading={isLoading} />
      <ProductRightSide product={product?.data}
        isLoading={isLoading} />
    </div>
  );
};
```

Листинг Б.3 – Файл ShopProductItemCard.tsx

```

import { Button } from "@ui/button";
import { Minus, Plus, ShoppingCart } from "lucide-react";
import { useState } from "react";
import { CartDB, ProductDB, ProductItemDB } from
"../../../../../@types/types";
import { useCreateCartItem } from
"../../../../../database/(services)/requests/createCartItem";

interface ProductItemCardProps {
  item: ProductDB;
  productItem: ProductItemDB;
  cart: CartDB;
  onClose: () => void;
}

export const ProductItemCard = ({
  productItem,
  cart,
  item,
  onClose,
}: ProductItemCardProps) => {
  const { mutate } = useCreateCartItem(cart.userId);

  const handleAddItemToCard = () => {
    mutate({
      id: 0,
      productId: productItem.id,
      productName: item.name,
      price: item.price,
      cartId: cart.id,
      color: productItem.color,
      size: productItem.size,
      selectCount: 1,
      isSelected: false,
    });
    onClose();
  };

  return (
    <div className="flex gap-4 items-center justify-between bg-
background border border-border hover:border-border-hover shadow
rounded-lg p-6">
      <div className="flex gap-2">
        <div className="font-medium">Цвета:</div>
        <div className="text-[#8F8F8F] w-[80px] truncate">
          {productItem.color}
        </div>
      </div>
      <div className="flex gap-2">
        <div className="font-medium">Размер:</div>
        <span className="text-[#8F8F8F]">{productItem.size}</span>
      </div>
    </div>
  );
}

```

```

        <div className="font-medium">Бес:</div>
        <span className="text-
[#8F8F8F]">{productItem.weight}</span>
        </div>

        <Button onClick={handleAddItemToCard} className="w-[114px]">
          <ShoppingCart className="w-5 h-5 mr-2" />В корзину
        </Button>
      </div>
    );
  };
};

```

Листинг Б.4 – Файл ProductRightSide.tsx

```

import { ProductDB } from "../../../../../types/types";
import { ProductItems } from "../ProductItems";
import { Skeleton } from "@ui/skeleton";

interface ProductRightSideProps {
  product?: ProductDB;
  isLoading?: boolean;
  onClose: () => void;
}

export const ProductRightSide = ({
  product,
  isLoading,
  onClose,
}: ProductRightSideProps) => {
  if (isLoading) {
    return (
      <div className="flex flex-col space-y-6 w-[60%]">
        {Array.from({ length: 6 }, (_, index) => (
          <Skeleton key={index} className="h-[98px] rounded-xl" />
        ))}
      </div>
    );
  }

  return (
    <div className="flex-1 flex flex-col gap-6 ">
      <ProductItems
        productItems={product?.items}
        product={product}
        onClose={onClose}
      />
    </div>
  );
};

```

Листинг Б.5 – Файл ProductRightSide.tsx

```

"use client";
import { useClickOutside } from "@siberiacancode/reactuse";
import { useState } from "react";

import { cn } from "@utils/helpers/cn";
import { Input } from "@ui/input";

interface SeachInputProps {
  className?: string;
}

export const SeachInput = ({ className }: SeachInputProps) => {
  const [focused, setFocused] = useState(false);

  const clickOutsideRef = useClickOutside<HTMLDivElement>(() => {
    setFocused(false);
  });

  return (
    <div ref={clickOutsideRef} className={cn("")}>
      <label htmlFor="search-input">
        <Icon20Search />
      </label>
      <Input
        className={
          "h-8 rounded-md border bg-secondary py-[1px] pr-[6px]
pl-[8px]"
        }
        id="search-input"
        onBlur={() => setFocused(false)}
        onFocus={() => setFocused(true)}
        placeholder="Поиск товаров"
      />
    </div>
  );
};

```

Листинг Б.6 – Файл CartItem.tsx

```

import { Button } from "@ui/button";
import { cn } from "@utils/helpers/cn";
import { Check, Minus, Plus, X } from "lucide-react";
import { useState } from "react";

interface CartItemProps {
  className?: string;
  item: any;
}

export const CartItem = ({ className, item }: CartItemProps) => {
  const [isChecked, setIsChecked] = useState(false);

  return (

```

```

<div
  className={cn(
    "bg-white p-4 rounded-lg hover:border-border-hover border
shadow flex flex-col gap-2",
    className
  )}
>
  <p className="font-medium">{item.name}</p>
  <div className="flex justify-between items-center">
    <p className="font-semibold text-lg">{item.price}</p>
    <div className="flex gap-2">
      <div className="flex gap-1 text-sm">
        <span className="font-medium">Цвета:</span>
        <span className="text-[#8F8F8F]">{item.color}</span>
      </div>
      <div className="flex gap-1 text-sm">
        <span className="font-medium">Размер:</span>
        <span className="text-[#8F8F8F]">{item.size}</span>
      </div>
    </div>
  </div>
  <div className="flex justify-between items-center">
    <Button
      variant={item.isSelected ? "default" : "outline"}
      size="icon"
      className="w-8 h-8"
      onClick={() => setIsChecked(!isChecked)}
    >
      <Check size={24} className={cn(!item.isSelected &&
"hidden")} />
    </Button>
    <div className="flex justify-end items-center gap-2">
      <div className="flex gap-[5px] bg-black rounded-lg p-1">
        <Button
          size="icon"
          className="w-8 h-8"
          //   onClick={() => setCount(count - 1)}
        >
          <Minus />
        </Button>

        <div className="bg-primary text-white w-8 h-8 rounded-
lg p-2 flex items-center justify-center cursor-pointer">
          {item.StockCount}
        </div>
        <Button
          size="icon"
          className="w-8 h-8"
          //   onClick={() => setCount(count + 1)}
          //   disabled={count === productItem.StockCount}
        >
          <Plus />
        </Button>
      </div>
    </div>
  </div>

```

```

        </div>
        <Button
          variant={"ghost"}
          size="icon"
          className="w-8 h-8"
          //   onClick={() => setCount(count - 1)}
        >
          <X size={24} />
        </Button>
      </div>
    </div>
  </div>
);
};

```

Листинг Б.7 – Файл ProductsCatalog.tsx

```

"use client";
import { useRouter } from "next/navigation";
import { SkeletonCard } from "@components/SkeletonCard";
import { ProductCard } from "../Product/ProductCard";
import { useGetProductsQuery } from
"@/hooks/query/useGetProductsQuery";

export const ProductsCatalog = () => {
  const router = useRouter();

  const { data: products, isLoading } = useGetProductsQuery();

  const onRedirectToProduct = (product: Product) => {
    router.push(`/shop/${product.id}`);
  };

  return (
    <div className="grid grid-cols-4 gap-6 p-6 pt-0">
      {isLoading &&
        Array(12)
          .fill(0)
          .map((_, index) => {
            return <SkeletonCard key={index} />;
          })
      }
      {products &&
        products?.data.map((product, index) => (
          <ProductCard
            key={index}
            product={product}
            onClick={() => onRedirectToProduct(product)}
          />
        ))
      }
    </div>
  );
};

```

Листинг Б.8 – Файл CheckboxFiltersGroup.tsx

```

"use client";
import { Input } from "@ui/input";
import { Title } from "@ui/title";
import { useState } from "react";
import { FilterCheckbox } from "../FilterCheckbox";
import { Skeleton } from "@ui/skeleton";

interface CheckboxFiltersGroupProps {
  title: string;
  items: any[];
  defaultItems?: any[];
  limit?: number;
  loading?: boolean;
  searchInputPlaceholder?: string;
  onClickCheckbox?: (id: string) => void;
  defaultValue?: string[];
  selected?: Set<string>;
  className?: string;
  name?: string;
}

export const CheckboxFiltersGroup = ({
  title,
  items,
  defaultItems,
  limit = 5,
  searchInputPlaceholder = "Поиск",
  className,
  loading,
  onClickCheckbox,
  selected,
  name,
}: CheckboxFiltersGroupProps) => {
  const [showAll, setShowAll] = useState(false);
  const [term, setTeam] = useState("");

  const list = showAll
    ? items.filter(({ text }) =>
      text.toLowerCase().includes(term))
    : (defaultItems || items)?.slice(0, limit);

  const onChangeSearchInput = (e:
    React.ChangeEvent<HTMLInputElement>) => {
    setTeam(e.target.value.toLowerCase());
  };

  if (loading) {
    return (
      <div className={className}>
        <Title className="font-medium mb-3" size="md" text={title}
      />

```

```

        {...new Array(limit)].map((_, index) => (
            <Skeleton key={index} className="mb-4 h-6 rounded-[8px]"
        />
        )))
    </div>
    );
}

return (
    <div className="flex flex-col gap-3">
        <Title className="font-medium" size="md" text={title} />
        {showAll && (
            <div>
                <Input
                    onChange={onChangeSearchInput}
                    placeholder={searchInputPlaceholder}
                />
            </div>
        )}
        <div className="scrollbar flex max-h-96 flex-col gap-4
overflow-auto pr-2">
            {list.map((item, index) => (
                <FilterCheckbox
                    key={index}
                    text={item.text}
                    value={item.value}
                    checked={selected?.has(item.value)}
                    onCheckedChange={() => onClickCheckbox?.(item.value)}
                    name={name}
                />
            )})}
        </div>

        {items.length > limit && (
            <div className="">
                <button
                    onClick={() => setShowAll(!showAll)}
                    className="text-primary text-xs"
                >
                    {showAll ? "Скрыть" : "Показать все"}
                </button>
            </div>
        )}
    </div>
);
};

```

Листинг Б.9 – Файл CreateProduct.tsx

```

"use client";
import {
  Dialog,
  DialogClose,
  DialogContent,
  DialogDescription,
  DialogFooter,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@ui/dialog";
import { cn } from "@utils/helpers/cn";
import { Button } from "@ui/button";
import { CreateProductForm } from "../CreateProductForm";
import { useState } from "react";

export const CreateProduct = () => {
  const [isOpen, setIsOpen] = useState(false);

  const handleClose = () => {
    setIsOpen(false);
  };

  return (
    <div className={cn("")}>
      <Dialog open={isOpen} onOpenChange={setIsOpen}>
        <DialogTrigger className="text-sm rounded-md border
border-border h-8 px-1.5 font-medium shadow-sm hover:bg-secondary-
hover ">
          Создать товар
        </DialogTrigger>
        <DialogContent className="bg-white p-0 min-w-[1300px]">
          <DialogHeader className="px-6 pt-6 ">
            <DialogTitle>Создание товара</DialogTitle>
            <DialogDescription className="text-base">
              Введите нужные параметры товара в форму создания
            </DialogDescription>
          </DialogHeader>
          <CreateProductForm onClose={handleClose} />
          <DialogFooter className="bg-background p-6 rounded-b-lg
border-t border-border max-w-[1298px]">
            <DialogClose asChild>
              <Button type="button" variant="secondary">
                Отмена
              </Button>
            </DialogClose>
          </DialogFooter>
        </DialogContent>
      </Dialog>
    </div>
  );
};

```

```
};
```

Листинг Б.10 – Файл AddImagesFormdb.tsx

```
import { useUserStore } from "@store/userStore";
import { Button } from "@ui/button";
import { FormControl, FormField, FormItem, FormMessage } from
"@ui/form";
import { Input } from "@ui/input";
import { Title } from "@ui/title";
import { productSchema } from "@utils/constants/productSchema";
import { Plus } from "lucide-react";
import { useFieldArray, UseFormReturn } from "react-hook-form";
import { z } from "zod";

interface AddImagesFormProps {
  form: UseFormReturn<z.infer<typeof productSchema>, any,
undefined>;
}

export const AddImagesForm = ({ form }: AddImagesFormProps) => {
  // const { selectedFactoryUser } = useUserStore((state) =>
state);

  const { fields, append, remove } = useFieldArray({
    control: form.control,
    name: "images",
  });

  return (
    <div className="flex flex-col gap-2 min-w-[265px]">
      <Title
        className="text-sm font-medium"
        tag="h5"
        text="Изображения товара"
      />

      <div className="flex flex-col gap-2 justify-start">
        <Button
          type="button"
          size="small"
          onClick={() => append({ id: 0, url: "", productId: 0 })}
        >
          Добавить изображение товара
        </Button>
        <p className="text-red-500 text-[13px] font-medium">
          {form.formState.errors.images?.message}
        </p>
      </div>

      <div className="flex flex-col gap-2 h-[300px] overflow-auto
pr-3 scrollbar">
        {fields.map((field, index) => (
```

```

<FormField
  key={field.id}
  control={form.control}
  name={`images.${index}.url`}
  render={({ field }) => (
    <div className="flex items-center gap-2">
      <FormItem>
        <FormControl>
          <Input
            type="text"
            placeholder="Введите URL изображения"
            {...field}
          />
        </FormControl>
        <FormMessage />
      </FormItem>
      <Button
        type="button"
        className="h-6 w-6"
        size="icon"
        onClick={() => remove(index)}
      >
        ✕
      </Button>
    </div>
  )}
/>
  )})
</div>
</div>
);
};

```

Листинг Б.11 – Файл DBAddMaterialForm.tsx

```

import { Button } from "@ui/button";
import { FormControl, FormField, FormItem, FormMessage } from
"@ui/form";
import {
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@ui/select";
import { Title } from "@ui/title";
import { productSchema } from "@utils/constants/productSchema";
import { Select } from "@radix-ui/react-select";
import { useFieldArray, UseFormReturn } from "react-hook-form";
import { z } from "zod";
// import { ProductMaterialDB } from
".../.../(services)/types/types";

```

```

interface AddMatetialsFormProps {
  form: UseFormReturn<z.infer<typeof productSchema>, any,
undefined>;
}

const productMaterials = [
  { id: 1, name: "Сетчатый текстиль (Mesh)" },
  { id: 2, name: "Нейлон" },
  { id: 3, name: "Кожа" },
  { id: 4, name: "Синтетическая ткань (PU или PVC)" },
  { id: 5, name: "Гума (резина)" },
  { id: 6, name: "ЭВА (EVA)" },
  { id: 7, name: "Пенополиуретан (PU)" },
  { id: 8, name: "Термопластичный полиуретан (TPU)" },
  { id: 9, name: "Карбоновое волокно" },
  { id: 10, name: "Ткань с водоотталкивающей пропиткой" },
];

export const AddMatetialsForm = ({ form }: AddMatetialsFormProps)
=> {
  const { fields, append, remove } = useFieldArray({
    control: form.control,
    name: "materials",
  });

  return (
    <div className="flex flex-col gap-2 min-w-[240px]">
      <Title className="text-sm font-medium" tag="h5"
text="Материалы" />
      <div className="flex flex-col gap-2 justify-start ">
        <Button
          type="button"
          size="small"
          onClick={() => {
            append({
              id: productMaterials[0].id,
              name: productMaterials[0].name,
            });
          }}
        >
          Добавить материал
        </Button>
        <p className="text-red-500 text-[13px] font-medium">
          {form.formState.errors.materials?.message}
        </p>
      </div>
      <div className="flex flex-col gap-2 h-[300px] overflow-auto
pr-3 scrollbar px-3 py-1">
        {fields.map((field, index) => (
          <div key={field.id} className="flex items-center gap-2
">
            <FormField
              control={form.control}

```

```

name={`materials.${index}.id`}
render={({ field }) => (
  <FormItem>
    <Select
      // При выборе материала обновляем Id и
Material
      onChange={(value) => {
        const selectedMaterial =
productMaterials.find(
          (material) => String(material.id) ===
value
        );
        if (selectedMaterial) {
          form.setValue(
            `materials.${index}.id`,
            selectedMaterial.id
          );
          form.setValue(
            `materials.${index}.name`,
            selectedMaterial.name
          );
        }
      }}
      value={String(field.value)} // Устанавливаем
значение Id
    >
      <FormControl>
        <SelectTrigger className="w-[250px] text-
left text-ellipsis">
          <SelectValue placeholder="Выберите
материал" />
        </SelectTrigger>
      </FormControl>
      <SelectContent>
        {productMaterials.map((material) => (
          <SelectItem
            key={material.id}
            value={String(material.id)}
          >
            {material.name}
          </SelectItem>
        ))}
      </SelectContent>
    </Select>
    <FormMessage />
  </FormItem>
)}
/>
<Button
  type="button"
  className="h-6 w-6"
  size="icon"
  onClick={() => remove(index)}

```

```

        >
        X
      </Button>
    </div>
  )))
</div>
</div>
);
};

```

Листинг Б.12 – Файл DBOrderCard.tsx

```

import { CardWrapper } from "@components/CardWrapper";
import { InfoRow } from "@components/InfoRow";
import { StatusBadge } from "@components/StatusBadge";
import { Button } from "@ui/button";
import {
  CircleCheck,
  CircleX,
  Copy,
  Package,
  PackageCheck,
  Trash,
  Truck,
} from "lucide-react";
import {
  OrderDB,
  OrderStatusDB,
  OrderTypeDB,
} from "../../../../../(services)/types/types";
import { format } from "date-fns";
import { useUpdateOrderStatus } from
"../../../../../(services)/requests/order/updateOrderStatus";

interface OrderCardProps {
  item: OrderDB;
  isAdmin?: boolean;
}

type StatusStyle = {
  label: string;
  styles: string;
  Icon: React.FC<React.SVGProps<SVGSVGElement>>;
};

const STATUS_STYLES: Record<OrderStatusDB, StatusStyle> = {
  [OrderStatusDB.Pending]: {
    label: "Собирается",
    styles: "bg-[#F5FBFF] border-[#BDE3FF]",
    Icon: Package,
  },
  [OrderStatusDB.Delivering]: {

```

```

        label: "В пути",
        styles: "bg-[#FFF8F0] border-[#FCD19C]",
        Icon: Truck,
      },
      [OrderStatusDB.Cancelled]: {
        label: "Отменен",
        styles: "bg-[#FDE3DE] border-[#F24822]",
        Icon: CircleX,
      },
      [OrderStatusDB.Delivered]: {
        label: "Доставлен",
        styles: "bg-[#DCF3E6] border-[#14AE5C]",
        Icon: CircleCheck,
      },
      // Если нужно учесть статус Deleted
      [OrderStatusDB.Deleted]: {
        label: "Удален",
        styles: "bg-gray-200 border-gray-400",
        Icon: CircleX,
      },
    ],
  };

const TYPE_STYLE: Record<OrderTypeDB, StatusStyle> = {
  [OrderTypeDB.Order]: {
    label: "Заказ",
    styles: "bg-[#f5f2ff] border-[#aa99ec]",
    Icon: PackageCheck,
  },
  [OrderTypeDB.Supply]: {
    label: "Поставка",
    styles: "bg-[#ffd3a4] border-[#ffbala]",
    Icon: Truck,
  },
};

export const OrderCard = ({ item, isAdmin }: OrderCardProps) => {
  const { id, createDate, totalWeight, totalPrice, status, type }
  = item;
  const formattedDate = format(new Date(createDate), "dd MMMM
  yyyy");
  const statusData = STATUS_STYLES[status];
  const typeData = TYPE_STYLE[type];

  const { mutate } = useUpdateOrderStatus();

  const handleDeleteOrder = () => {
    mutate({ orderId: id, status: OrderStatusDB.Deleted });
  };

  return (
    <CardWrapper>
      <div className="flex flex-col gap-3 flex-1">

```

```

<div className="text-base text-[#8F8F8F] flex gap-2 items-
center">
  <p>От {formattedDate}</p>•<p>{id}</p>
  <Button
    variant="ghost"
    size="icon"
    onClick={() => navigator.clipboard.writeText(id)}
  >
    <Copy size={18} />
  </Button>
</div>
</div>

{isAdmin && (
  <div className="flex gap-2 flex-col">
    <InfoRow label="Username пользователя:"
value={item.username} />
  </div>
)}

<div className="flex justify-between items-center">
  <div className="flex gap-2 flex-col text-sm">
    <InfoRow label="Вес:" value={` ${totalWeight} кг`} />
    {type === OrderTypeDB.Order && (
      <InfoRow label="Цена:" value={` ${totalPrice} $`} />
    )}
  </div>

  <div className="flex gap-2">
    {isAdmin && (
      <StatusBadge
        label={typeData.label}
        Icon={typeData.Icon}
        className={typeData.styles}
      />
    )}
    <StatusBadge
      label={statusData.label}
      Icon={statusData.Icon}
      className={statusData.styles}
    />
    {isAdmin && (
      <Button
        variant="ghost"
        className="bg-[#FDE3DE] border-[#F24822] border-2"
        size="icon"
        onClick={handleDeleteOrder}
      >
        <Trash />
      </Button>
    )}
  </div>
</div>

```

```

    </CardWrapper>
  );
};

```

Листинг Б.13 – Файл DBUsersPanel.tsx

```

"use client";
import { UserCard } from "../DBUserCard";
import { UnifiedPanel } from "@components/UnifiedPannel";
import {
  useGetGroupedUsersQuery,
  useGetUsersQuery,
} from "../../../../../(services)/requests/getUsers";
import { RegisterDialog } from "../DBCreateUserDialog";
import { Skeleton } from "@ui/skeleton";
import { useMemo, useState } from "react";
import { UserStatusDB } from "../../../../../(services)/types/types";
import { GroupedUsersSwitch } from "../DBGropedUsersSwitch";
import { Title } from "@ui/title";
import React from "react";

export const UsersPanel = () => {
  const [isGrouped, setIsGrouped] = useState(false);

  const { data: users, isLoading: isLoadingUsers } =
    useGetUsersQuery();
  const { data: groupedUsers, isLoading: isLoadingGrouped } =
    useGetGroupedUsersQuery();

  const handleToggleGrouped = () => {
    setIsGrouped((state) => !state);
  };

  const activeUsers = useMemo(() => {
    return users?.data.filter((user) => user.status !==
UserStatusDB.Deleted);
  }, [users]);

  const sortedGroupedUsers = useMemo(() => {
    if (groupedUsers?.data) {
      return groupedUsers.data.sort((a, b) => (a.role ===
"factory" ? -1 : 1));
    }
    return [];
  }, [groupedUsers]);

  if (isLoadingUsers || isLoadingGrouped) {
    return (
      <UnifiedPanel
        title="Пользователи сервиса"
        items={Array(6).fill({})}

```

```

        renderItem={() => <Skeleton className="w-full h-[176px]"
/>}
      />
    );
  }

  return (
    <section className="py-4 px-6 border-r border-b border w-[50%]
max-h-full">
      <header className="flex items-center justify-between mb-2">
        <Title
          text="Пользователи сервиса"
          size="lg"
          className="font-semibold"
          tag="h2"
        />
        <div className="flex flex-col items-center gap-2">
          <GroupedUsersSwitch
            switcher={handleToggleGrouped}
            state={isGrouped}
          />
          <RegisterDialog />
        </div>
      </header>

      <div className="flex flex-col gap-3 px-2 pb-2 overflow-auto
max-h-[calc(100%-64px)] scrollbar">
        {isGrouped
          ? sortedGroupedUsers.map((group, index) => (
              <div key={index} className="flex flex-col gap-3">
                <h3 className="font-semibold mb-2">
                  {group.role === "factory" ? "Заводы" :
"Магазины"} (
                    {group.count})
                </h3>
                {group.users.map((user) => (
                  <UserCard key={user.id} item={user} />
                ))}
              </div>
            ))
          : activeUsers?.map((item, index) => (
              <React.Fragment key={index}>
                <UserCard item={item} />
              </React.Fragment>
            ))}
      </div>
    </section>
  );
};

```

Листинг Б.14 – Файл clearCart.ts

```

import { dbapi } from "@utils/api/dbinstance";
import { queryClient } from "@utils/api/query-client";
import { useMutation, UseMutationOptions } from "@tanstack/react-query";

const deleteCart = async (userId: number) => {
  await dbapi.delete(`cart/clear/${userId}`);
};

export const useDeleteCart = (
  userId: number,
  options?: UseMutationOptions<void, Error, number>
) => {
  return useMutation<void, Error, number>({
    mutationFn: deleteCart,
    onSuccess: () => {
      queryClient.invalidateQueries({ queryKey: ["cart", userId]
    });
  },
  ...options,
});
};

```

Листинг Б.15 – Файл createCart.ts

```

import { dbapi } from "@utils/api/dbinstance";
import { CartItemDB } from "../types/types";
import { useMutation, UseMutationOptions } from "@tanstack/react-query";
import { queryClient } from "@utils/api/query-client";

interface CartRequest {
  id: number;
  cartId: number;
  productId: number;
  productName: string;
  price: number;
  color: string;
  size: number;
  selectCount: number;
  isSelected: boolean;
}

const createCartItem = async (payload: CartRequest) => {
  const response = await dbapi.post<CartRequest>("cart/item",
payload);
  return response.data;
};

export const useCreateCartItem = (
  userId: number,

```

```

    options?: UseMutationOptions<CartRequest, Error, CartItemDB>
  ) => {
    return useMutation<CartRequest, Error, CartRequest>({
      onSuccess: () => {
        queryClient.invalidateQueries({ queryKey: ["cart", userId]
      });
    },
    mutationFn: createCartItem,
    ...options,
  });
};

```

Листинг Б.16 – Файл instance.ts

```

import createFetchClient from "openapi-fetch";
import createClient from "openapi-react-query"; // generated by
openapi-typescript
import { CONFIG } from "@shared/model/config";
import { ApiPaths, ApiSchemas } from "../schema";
import { useSession } from "../model/session";

export const fetchClient = createFetchClient<ApiPaths>({
  baseUrl: CONFIG.API_BASE_URL,
});
export const rqClient = createClient(fetchClient);

export const publicFetchClient = createFetchClient<ApiPaths>({
  baseUrl: CONFIG.API_BASE_URL,
});
export const publicRqClient = createClient(publicFetchClient);

fetchClient.use({
  async onRequest({ request }) {
    const token = await useSession.getState().refreshToken();

    if (token) {
      request.headers.set("Authorization", `Bearer ${token}`);
    } else {
      return new Response(
        JSON.stringify({
          code: "NOT_AUTHORIZED",
          message: "You are not authorized to access this
resource",
        }) as ApiSchemas["Error"],
        {
          status: 401,
          headers: {
            "Content-Type": "application/json",
          },
        },
      );
    }
  }
});

```

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

Листинг Б.17 – Файл react.ts

```
import { useState } from "react";
import { useEffect } from "react";

export function useDebouncedValue<T>(value: T, delay: number) {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const timer = setTimeout(() => setDebouncedValue(value),
    delay);

    return () => clearTimeout(timer);
  }, [value, delay]);

  return debouncedValue;
}
```

Листинг Б.18 – Файл routes.ts

```
import "react-router-dom";

export const ROUTES = {
  HOME: "/",
  LOGIN: "/login",
  REGISTER: "/register",
  BOARDS: "/boards",
  BOARD: "/boards/:boardId",
  FAVORITE_BOARDS: "/boards/favorite",
  RECENT_BOARDS: "/boards/recent",
} as const;

export type PathParams = {
  [ROUTES.BOARD]: {
    boardId: string;
  };
};

declare module "react-router-dom" {
  interface Register {
    params: PathParams;
  }
}
```

Листинг Б.19 – Файл session.ts

```

import { useState } from "react";
import { jwtDecode } from "jwt-decode";
import { createGStore } from "create-gstore";
import { publicFetchClient } from "../api/instance";

type Session = {
  userId: string;
  email: string;
  exp: number;
  iat: number;
};

const TOKEN_KEY = "token";

let refreshTokenPromise: Promise<string | null> | null = null;

export const useSession = createGStore(() => {
  const [token, setToken] = useState(() =>
    localStorage.getItem(TOKEN_KEY));

  const login = (token: string) => {
    localStorage.setItem(TOKEN_KEY, token);
    setToken(token);
  };

  const logout = () => {
    localStorage.removeItem(TOKEN_KEY);
    setToken(null);
  };

  const session = token ? jwtDecode<Session>(token) : null;

  const refreshToken = async () => {
    if (!token) {
      return null;
    }

    const session = jwtDecode<Session>(token);

    if (session.exp < Date.now() / 1000) {
      if (!refreshTokenPromise) {
        refreshTokenPromise = publicFetchClient
          .POST("/auth/refresh")
          .then((r) => r.data?.accessToken ?? null)
          .then((newToken) => {
            if (newToken) {
              login(newToken);
              return newToken;
            } else {
              logout();
              return null;
            }
          });
      }
    }
  };

```

```

        }
      })
      .finally(() => {
        refreshTokenPromise = null;
      });
    }

    const newToken = await refreshTokenPromise;

    if (newToken) {
      return newToken;
    } else {
      return null;
    }
  }

  return token;
};

return { refreshToken, login, logout, session };
});

```

Листинг Б.20 – Файл use-login.ts

```

import { publicRqClient } from "@shared/api/instance";
import { ApiSchemas } from "@shared/api/schema";
import { ROUTES } from "@shared/model/routes";
import { useSession } from "@shared/model/session";
import { useNavigate } from "react-router-dom";

export function useLogin() {
  const navigate = useNavigate();

  const session = useSession();
  const loginMutation = publicRqClient.useMutation("post",
"/auth/login", {
  onSuccess(data) {
    session.login(data.accessToken);
    navigate(ROUTES.HOME);
  },
});

  const login = (data: ApiSchemas["LoginRequest"]) => {
    loginMutation.mutate({ body: data });
  };

  const errorMessage = loginMutation.isError
    ? loginMutation.error.message
    : undefined;

  return {
    login,
    isPending: loginMutation.isPending,
    errorMessage,
  };
}

```

```
};
}
```

Листинг Б.21 – Файл use-register.ts

```
import { publicRqClient } from "@shared/api/instance";
import { ApiSchemas } from "@shared/api/schema";
import { ROUTES } from "@shared/model/routes";
import { useSession } from "@shared/model/session";
import { useNavigate } from "react-router-dom";

export function useRegister() {
  const navigate = useNavigate();

  const session = useSession();
  const registerMutation = publicRqClient.useMutation(
    "post",
    "/auth/register",
    {
      onSuccess(data) {
        session.login(data.accessToken);
        navigate(ROUTES.HOME);
      },
    },
  );

  const register = (data: ApiSchemas["RegisterRequest"]) => {
    registerMutation.mutate({ body: data });
  };

  const errorMessage = registerMutation.isError
    ? registerMutation.error.message
    : undefined;

  return {
    register,
    isPending: registerMutation.isPending,
    errorMessage,
  };
}
```

Листинг Б.22 – Файл login.page.tsx

```
import { ROUTES } from "@shared/model/routes";
import { Link } from "react-router-dom";
import { AuthLayout } from "../ui/auth-layout";
import { LoginForm } from "../ui/login-form";

function LoginPage() {
  return (
    <AuthLayout
      title="Вход в систему"
      description="Введите ваш email и пароль для входа в систему"
```

```

    form={<LoginForm />}
    footerText={
      <>
        Нет аккаунта? <Link
to={ROUTES.REGISTER}>Зарегистрироваться</Link>
      </>
    }
  />
);
}

```

```
export const Component = LoginPage;
```

Листинг Б.23 – Файл template-card.tsx

```

import { Button } from "@shared/ui/kit/button";
import { PlusIcon } from "lucide-react";
import { cn } from "@shared/lib/css";

interface Template {
  id: string;
  name: string;
  description: string;
  thumbnail: string;
}

interface TemplateCardProps {
  template: Template;
  onSelect: (template: Template) => void;
  className?: string;
}

export function TemplateCard({
  template,
  onSelect,
  className,
}: TemplateCardProps) {
  return (
    <div
      className={cn(
        "group relative rounded-lg border p-4 hover:border-primary
transition-colors cursor-pointer",
        className,
      )}
      onClick={() => onSelect(template)}
    >
      <div className="aspect-video rounded-md bg-gray-100 mb-4
overflow-hidden">
        <img
          src={template.thumbnail}
          alt={template.name}
          className="w-full h-full object-cover"
        />
      </div>
    </div>
  );
}

```

```

    </div>
    <h3 className="font-medium mb-1">{template.name}</h3>
    <p className="text-sm text-gray-
500">{template.description}</p>
    <Button
      size="sm"
      className="absolute top-4 right-4 opacity-0 group-
hover:opacity-100 transition-opacity"
    >
      <PlusIcon className="h-4 w-4 mr-2" />
      ИСПОЛЬЗОВАТЬ
    </Button>
  </div>
);
}

```

Приложение В
(рекомендуемое)
Графическая часть

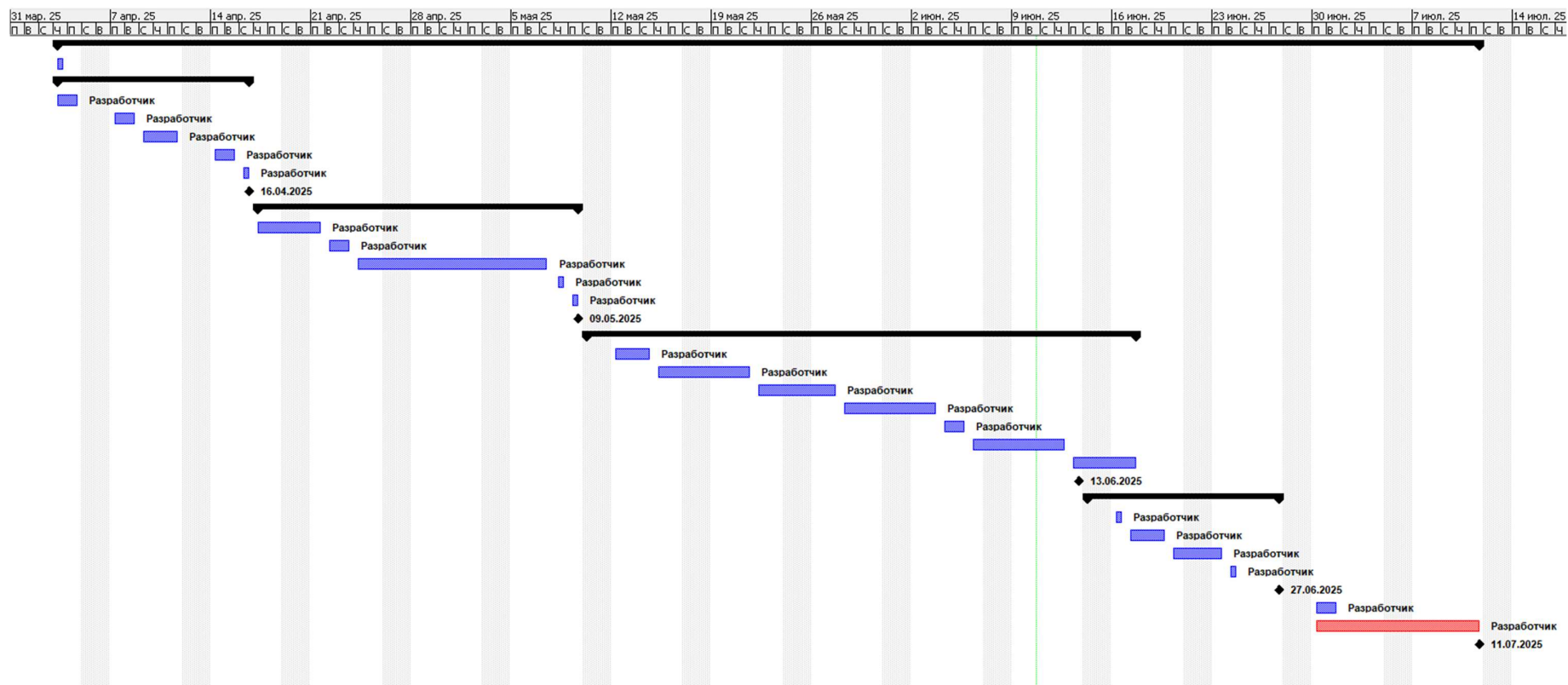


Рисунок В.1 – Диаграмма Ганта

	⑩	Название	Продолжительность	Начало	Окончание	Работа	Стоимость	Название ресурса
1		Разработка клиентского приложения	72 дней	03.04.2025, 08:00	11.07.2025, 17:00	588 часов	248900,00 Р	
2		Начало проекта	1 день	03.04.2025, 08:00	03.04.2025, 17:00	8 часов	0,00 Р	
3		Анализ предметной области и требований к клиентскому приложению	10 дней	03.04.2025, 08:00	16.04.2025, 17:00	80 часов	38600,00 Р	
4		Изучение предметной области логистики	2 дней	03.04.2025, 08:00	04.04.2025, 17:00	16 часов	8000,00 Р	Разработчик
5		Обзор и анализ аналогов клиентских приложений	2 дней	05.04.2025, 08:00	08.04.2025, 17:00	16 часов	7800,00 Р	Разработчик
6		Формулирование функциональных требований к клиенту	3 дней	09.04.2025, 08:00	11.04.2025, 17:00	24 часов	11400,00 Р	Разработчик
7		Формулирование нефункциональных требований к клиенту	2 дней	12.04.2025, 08:00	15.04.2025, 17:00	16 часов	7600,00 Р	Разработчик
8		Постановка задачи на разработку	1 день	16.04.2025, 08:00	16.04.2025, 17:00	8 часов	3800,00 Р	Разработчик
9		Конец анализа предметной области	0 дней	16.04.2025, 17:00	16.04.2025, 17:00	0 часов	0,00 Р	
10		Проектирование клиентского приложения	17 дней	17.04.2025, 08:00	09.05.2025, 17:00	132 часов	62700,00 Р	
11		Проектирование сценариев взаимодействия пользователей	3 дней	17.04.2025, 08:00	21.04.2025, 17:00	24 часов	11400,00 Р	Разработчик
12		Проектирование концептуальной модели данных	2 дней	22.04.2025, 08:00	23.04.2025, 17:00	16 часов	7600,00 Р	Разработчик
13		Проектирование пользовательского интерфейса	9,5 дней	24.04.2025, 08:00	07.05.2025, 13:00	76 часов	36100,00 Р	Разработчик
14		Выбор технологий разработки	1 день	08.05.2025, 08:00	08.05.2025, 17:00	8 часов	3800,00 Р	Разработчик
15		Составление требований к аппаратному обеспечению	1 день	09.05.2025, 08:00	09.05.2025, 17:00	8 часов	3800,00 Р	Разработчик
16		Конец проектирования приложения	0 дней	09.05.2025, 17:00	09.05.2025, 17:00	0 часов	0,00 Р	
17		Разработка клиентского приложения	27 дней	10.05.2025, 08:00	17.06.2025, 17:00	216 часов	72200,00 Р	
18		Настройка среды и архитектуры проекта	3 дней	10.05.2025, 08:00	14.05.2025, 17:00	24 часов	11400,00 Р	Разработчик
19		Разработка рабочей области для класса "Потребитель"	5 дней	15.05.2025, 08:00	21.05.2025, 17:00	40 часов	19000,00 Р	Разработчик
20		Разработка рабочей области для класса "Поставщик"	4 дней	22.05.2025, 08:00	27.05.2025, 17:00	32 часов	15200,00 Р	Разработчик
21		Разработка рабочей области для класса "Администратор"	5 дней	28.05.2025, 08:00	03.06.2025, 17:00	40 часов	19000,00 Р	Разработчик
22		Реализация модуля визуализации логистической сети	2 дней	04.06.2025, 08:00	05.06.2025, 17:00	16 часов	7600,00 Р	Разработчик
23		Реализация модуля отображения маршрутов	5 дней	06.06.2025, 08:00	12.06.2025, 17:00	40 часов	0,00 Р	
24		Реализация управления состоянием и навигации	3 дней	13.06.2025, 08:00	17.06.2025, 17:00	24 часов	0,00 Р	
25		Конец разработки приложения	0 дней	13.06.2025, 17:00	13.06.2025, 17:00	0 часов	0,00 Р	
26		Контроль качества приложения	21 дней	24.05.2025, 08:00	23.06.2025, 17:00	56 часов	27800,00 Р	
27		Разработка плана тестирования	1 день	14.06.2025, 08:00	16.06.2025, 17:00	8 часов	4000,00 Р	Разработчик
28		Проведение функционального тестирования	3 дней	17.06.2025, 08:00	19.06.2025, 17:00	24 часов	12000,00 Р	Разработчик
29		Проведение модульного тестирования	2 дней	20.06.2025, 08:00	23.06.2025, 17:00	16 часов	8000,00 Р	Разработчик
30		Анализ метрик кода	1 день	24.05.2025, 08:00	26.05.2025, 17:00	8 часов	3800,00 Р	Разработчик
31		Конец контроля качества приложения	0 дней	27.05.2025, 17:00	27.05.2025, 17:00	0 часов	0,00 Р	
32		Составление глоссария	2 дней	28.05.2025, 08:00	29.05.2025, 17:00	16 часов	7600,00 Р	Разработчик
33		Составление отчета	10 дней	30.06.2025, 08:00	11.07.2025, 17:00	80 часов	40000,00 Р	Разработчик
34		Конец проекта	0 дней	11.07.2025, 17:00	11.07.2025, 17:00	0 часов	0,00 Р	

Рисунок В.2 – Список задач проекта

	⑩	Название	E-mail адрес	Стандартная ставка	Начислять	Основной календарь
1		Разработчик	schubert.walk@yandex.ru	475,00 Р/час	Пропорционально	Пятидневка

Рисунок В.3 – Список ресурсов проекта