



*Задание*

Данная работа посвящена разработке информационной системы для автоматизации логистических процессов службы доставки. Система реализована как клиент-серверное приложение с клиентской частью на React и серверной частью на Ruby on Rails. Основные функциональные модули включают управление заказами, складской учет с использованием QR-кодов и маршрутизацию доставок. Особое внимание уделено безопасности данных, надежности хранения информации и удобству пользовательского интерфейса. В ходе работы проведены анализ требований, проектирование архитектуры, реализация и тестирование системы. Результатом является готовое к внедрению решение, соответствующее современным стандартам автоматизации логистических операций.

This work presents the development of an information system for automating delivery service logistics processes. The system is implemented as a client-server application with a React-based client and Ruby on Rails server. Core functional modules encompass order management, warehouse operations with QR-code verification, and delivery routing. Particular attention was given to data security, storage reliability, and user interface convenience. The project involved requirements analysis, architectural design, implementation, and system testing. The outcome is a production-ready solution that meets modern standards for logistics operations automation.

## Содержание

Введение .....	5
1 Анализ технического задания .....	7
1.1 Описание предметной области.....	7
1.2 План интервью для определения бизнес-требований.....	7
1.3 Формирование требований .....	8
1.4 Поиск и сравнение аналогов.....	10
1.5 Функциональные возможности.....	15
1.6 Обоснование выбора средств реализации.....	16
2 Проектирование архитектуры системы.....	19
2.1 Диаграмма вариантов использований .....	19
2.2 Диаграмма последовательностей.....	20
2.3 Логическая модель данных.....	21
2.4 Физическая модель данных .....	21
3 Реализация системы .....	23
3.1 Выбор и обоснование алгоритмов .....	23
3.2 Руководство программиста.....	27
3.3 Руководство пользователя .....	44
4 Тестирование системы .....	64
Заключение.....	69
Список литературы.....	70
Приложение А.....	71
Приложение Б .....	76
Приложение В .....	78

					<b>МИВУ.09.03.04-4.000 ПЗ</b>		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Чернышев А.Е.			Автоматизация процессов доставки грузов через разработку информационной системы управления заказами. Пояснительная записка	Лит.	Лист
Провер.		Кульков Я.Ю.					4
Реценз.							78
Н. Контр.		Холкина Н.Е.				<b>МИ ВлГУ ПИНз-120</b>	
Утверд.		Жизняков А.Л.					

## Введение

Бурное развитие информационных технологий в последние десятилетия привело к необходимости поиска новых решений для автоматизации деятельности различных организаций, предприятий и служб. Рабочие процессы таких организаций часто связаны с обработкой и хранением больших объемов информации. Особенно это актуально для служб, занимающихся учетом данных о клиентах, товарах, доставках и других аспектах бизнеса.

Еще совсем недавно информация подобного рода хранилась в картотеках с использованием бумажных носителей. Этот процесс требовал значительных физических ресурсов, данных заносились вручную на карточки, что не только занимало много времени, но и создавало большие неудобства, повышая вероятность ошибок. Вся информация должна была быть найдена вручную, а сама картотека со временем изнашивалась, что снижало ее надежность.

Сегодня, в эпоху компьютерных технологий и автоматизации, физические картотеки и бумажные носители постепенно уступают место более современным и удобным решениям. Вместо сложных и неудобных процессов теперь используются мощные компьютерные системы, обеспечивающие быструю и надежную обработку данных. Однако, несмотря на значительный прогресс, по-прежнему существуют проблемы, такие как необходимость быстрого поиска нужной информации, обеспечение надежности хранения данных и соблюдение конфиденциальности.

Для решения этих задач используются специализированные программные продукты, которые часто объединены в крупные информационно-справочные системы. Эти системы предназначены для управления различными массивами данных, включая базы данных, и предоставляют удобный интерфейс для пользователей. Современные информационные системы включают в себя функции для добавления, редактирования, удаления данных, а также для их поиска и простого просмотра. Важным аспектом таких систем является обеспечение надежности хранения информации и предотвращение ее утрат. Реализация этих решений зависит от возможностей используемых технологий, поставленных задач и квалификации разработчика.

Разработка комплексного программного решения для автоматизации логистических процессов службы доставки грузов, включающего:

1) Кроссплатформенную систему:

а) Реализация на React

б) Поддержка четырех ролевых моделей  
(клиент/доставщик/менеджер/администратор)

с) Интеграция с QR-код технологиями

2) Серверную часть:

а) RESTful API для централизованного управления данными

б) Безопасные HTTP-коммуникации

с) Масштабируемая архитектура

3) Ключевые функциональные модули:

а) Система управления доставками (маршрутизация, трекинг)

б) Складской учет с технологией QR-верификации

4) Особое внимание уделяется:

а) Обеспечению надежности хранения данных

б) Реализации удобного пользовательского интерфейса

с) Соблюдению требований информационной безопасности

д) Оптимизации бизнес-процессов доставки

Работа представляет собой законченное решение, сочетающее современные подходы к разработке мобильных приложений с эффективной backend-архитектурой, соответствующее актуальным требованиям цифровизации логистических операций.

# 1 Анализ технического задания

## 1.1 Описание предметной области

Предметная область проекта связана с разработкой веб-приложения для автоматизации логистических процессов компании, занимающейся доставкой грузов. Основные задачи системы включают:

- управление заказами (создание, отслеживание, отмена);
- приём и выдача грузов на складах (ПВЗ);
- контроль за доставками и маршрутами;
- управление пользователями (регистрация, авторизация, роли);
- формирование цены и обработка оплат.

Система должна обеспечивать функциональность для различных классов пользователей:

- клиенты: оформление заказов, отслеживание статусов, просмотр истории;
- менеджеры склада: приём и выдача грузов, управление оплатами;
- доставщики: приём грузов в доставку, указание параметров транспорта;
- администраторы: управление ролями, настройка системы, аналитика.

## 1.2 План интервью для определения бизнес-требований

Для уточнения требований к системе необходимо провести интервью с заинтересованными лицами. Основные вопросы:

1. Какие основные задачи должна решать система?
2. Кто будет основными пользователями (клиенты, менеджеры, доставщики, администраторы)?
3. Какие функции являются приоритетными для каждого класса пользователей?
4. Какие данные необходимо учитывать при управлении заказами, складами и доставками?

					МИВУ 09.03.04 - 4.000ПЗ	Лист
						7
Изм	Лист	№ докум	Подпись	Дата		



5. Какие ограничения по времени и ресурсам существуют для выполнения заказов?
6. Какие требования к безопасности и авторизации пользователей?
7. Какие интеграции с другими системами необходимы (CRM, ERP, платежные системы)?
8. Какие отчеты и аналитика должны быть доступны в системе?
9. Какие требования к производительности и доступности системы?
10. Какие бизнес-цели должны быть достигнуты с помощью системы?

### 1.3 Формирование требований

При проектировании системы автоматизации логистических процессов ключевое значение имеет четкое определение функциональных и нефункциональных характеристик, которые обеспечат эффективную работу всех участников цепочки доставки. Исходя из анализа бизнес-процессов компании и потребностей различных категорий пользователей, система должна реализовывать комплексный подход к управлению заказами, складами и доставками, обеспечивая при этом высокий уровень безопасности, удобства взаимодействия и масштабируемости. Основные требования к системе можно структурировать следующим образом:

#### 1.3.1 Функциональные требования

##### 1) Управление пользователями и доступом:

- а) многоуровневая система регистрации с дифференциацией по типам пользователей (клиенты, курьеры, менеджеры складов, администраторы);
- б) механизм назначения и динамического изменения ролей с детализированными правами доступа;
- с) возможность делегирования полномочий между сотрудниками;

##### 2) Работа с заказами:

					МИВУ 09.03.04 - 4.000ПЗ	Лист
Изм	Лист	№ докум	Подпись	Дата		8

- d) полноценный жизненный цикл обработки заказов от создания до финального исполнения;
  - e) интеллектуальная система отмены заказов с учетом временных ограничений и бизнес-правил;
  - f) архив выполненных заказов;
- 3) Складская логистика:
- g) учет операций приема-передачи грузов;
  - h) интеграция с системами маркировки (QR-коды);
- 4) Организация доставки:
- i) учет транспортных характеристик;
  - j) мониторинг выполнения доставок в реальном времени.

### 1.3.2 Нефункциональные требования

- 5) Безопасность и надежность:
- a) ролевая модель доступа с детализированными правами;
- 6) Пользовательский опыт:
- b) адаптивный интерфейс для разных устройств;
  - c) персонализированные рабочие пространства;
- 7) Интеграционные возможности:
- d) API для подключения внешних сервисов;
  - e) поддержка стандартных протоколов обмена данными;
- 8) Масштабируемость:
- f) возможность наращивания функционала;
  - g) поддержка роста количества пользователей;
  - h) горизонтальное масштабирование системы.

## 1.4 Поиск и сравнение аналогов

Для проведения всестороннего анализа рыночных решений и выявления оптимальных подходов к реализации системы рекомендуется выполнить детальное исследование существующих платформ, специализирующихся на логистике и управлении доставками. В рамках сравнительного анализа целесообразно рассмотреть следующие популярные отраслевые решения:

1) 4Logist.com - Комплексная платформа для управления логистикой, предлагающая:

- a) интегрированные инструменты управления складскими операциями;
- b) систему маршрутизации и планирования доставок;
- c) мобильные приложения для курьеров и клиентов;
- d) аналитические инструменты для оптимизации логистических процессов;

2) Битрикс24 (логистический модуль) - Корпоративное решение, включающее:

- a) CRM-систему с интеграцией логистических функций;
- b) инструменты управления транспортными средствами;
- c) систему учета грузоперевозок;
- d) возможности автоматизации документооборота;

3) TMS (Transportation Management System) системы:

- a) Oracle Transportation Management;

4) Специализированные SaaS-решения:

- a) LogistiX.

При проведении анализа следует учитывать:

- соответствие функциональных возможностей требованиям бизнеса;
- гибкость и масштабируемость архитектуры;
- уровень интеграции с существующей ИТ-инфраструктурой;
- стоимость владения и масштабирования;

- опыт внедрения в аналогичных бизнес-моделях;
- качество технической поддержки и экосистемы разработчиков.

Такой комплексный подход к анализу позволит не только выбрать оптимальные технологические решения, но и избежать типичных ошибок при проектировании системы.

### 1.4.1 4Logist.com

Специализированная логистическая платформа для среднего и крупного бизнеса. Предлагает комплексное решение для управления цепочками поставок с акцентом на маршрутизацию и контроль доставок. Отличается глубокой отраслевой специализацией и развитыми аналитическими инструментами. Пример рабочего окна платформы представлен на рисунке 1.

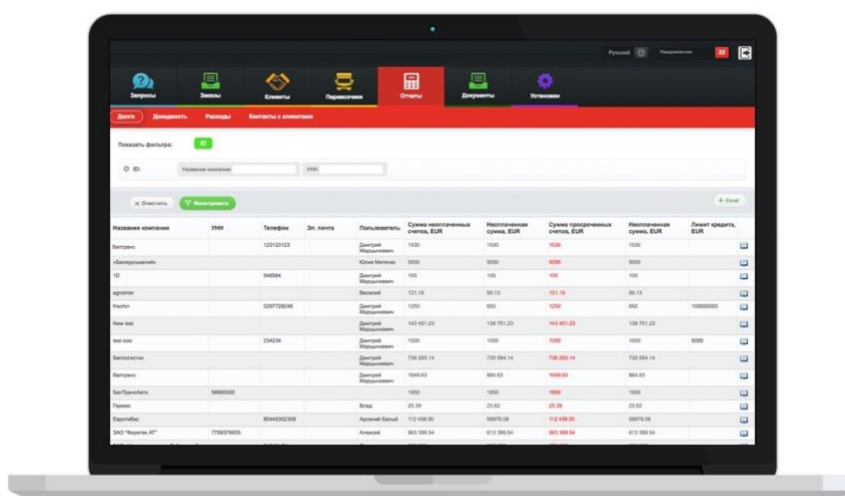


Рисунок 1 – Рабочее окно платформы 4Logist.

#### 1.4.2 Битрикс24 (Логистический модуль)

CRM-система с расширенными возможностями для управления заказами и базовыми логистическими функциями. Оптимальна для компаний, где логистика

тесно интегрирована с продажами и клиентским сервисом. Ограничена в специализированных функциях для сложной логистики. Рабочее окно модуля представлено на рисунке 2.

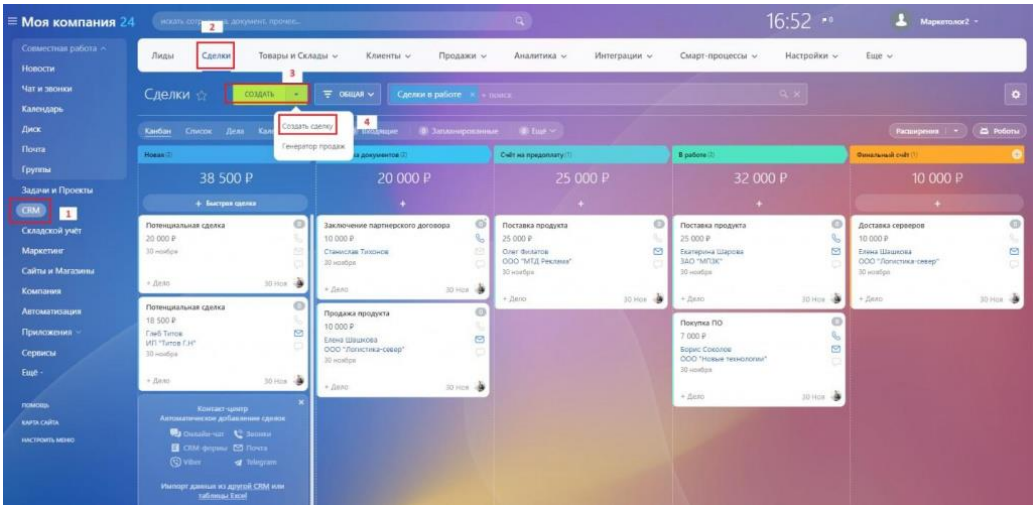


Рисунок 2 – Процесс доставки в Битрикс24.

### 1.4.3 Oracle Transportation Management (OTM)

Корпоративная система управления транспортом класса Enterprise. Обеспечивает глобальное управление цепочками поставок с мощными алгоритмами оптимизации. Требуется значительных ресурсов для внедрения и поддержки. Организация рабочего процесса в OTM представлена на рисунке 3.

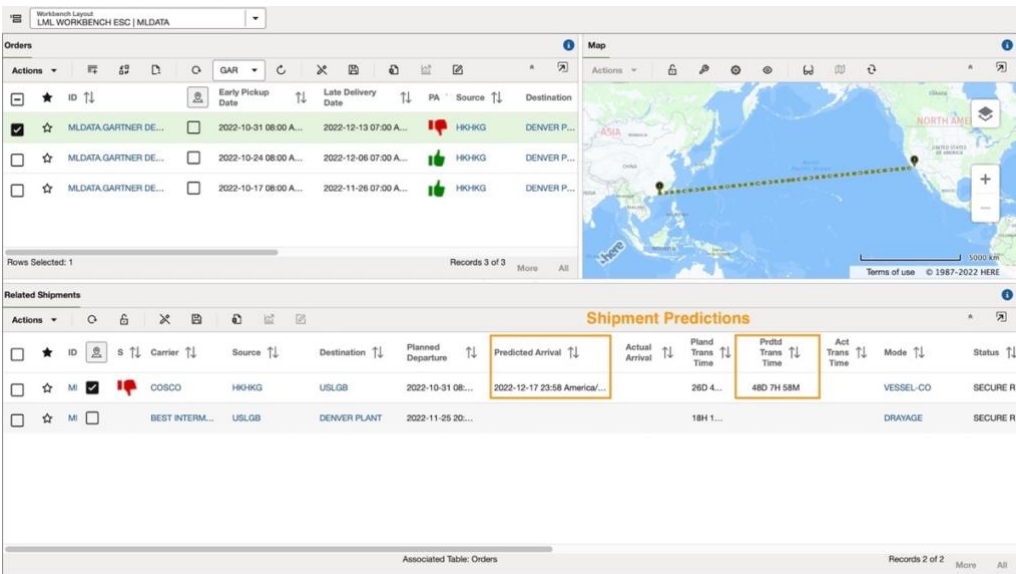


Рисунок 3 – Рабочее окно приложения Oracle Transportation Management.

## 1.4.4 LogistiX

Облачное SaaS-решение для оперативного управления доставкой. Отличается быстрым развертыванием и гибкой подпиской. Подходит для малого и среднего бизнеса с типовыми логистическими процессами. Рабочее окно приложения представлено на рисунке 4.

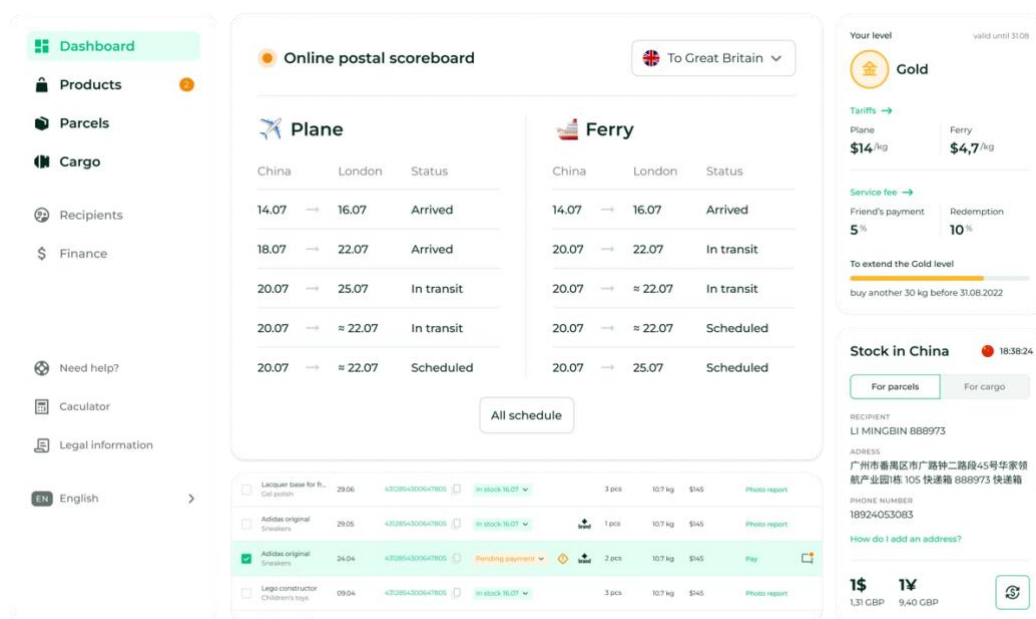


Рисунок 4 – Рабочее окно приложения LogistiX.

## 1.4.5 Сравнение аналогов

Было проведено сравнение характеристик рассмотренных аналогов, которое отображено в таблице 1.

Таблица 1 – Сравнение аналогов разрабатываемой система

Параметр сравнения	4Laogist.com	Битрикс24 (Логистика)	Oracle TMS	LogistiX
Тип решения	Специализированная логистическая платформа	CRM с логистическим модулем	Корпоративная TMS-система	Облачное SaaS-решение

Продолжение таблицы 1

Параметр сравнения	4Laogist.com	Битрикс24 (Логистика)	Oracle TMS	LogistiX
Основные функции	Управление заказами, маршрутизация, складской учет	Управление заказами, CRM, документооборот	Планирование перевозок, оптимизация маршрутов, фрахт	Управление доставками в реальном времени
Интеграции	API для 1C, ERP, маркетплейсов	Встроенная интеграция с продуктами 1C	Интеграция с SAP, Oracle ERP	REST API, Webhooks
Мобильные приложения	Да (отдельные для клиентов и курьеров)	Только общее мобильное приложение	Нет (только веб-интерфейс)	Да (кроссплатформенное)
Ценовая модель	Подписка (от \$50/пользователь)	Помесячная оплата (от \$20/польз.)	Корпоративные лицензии	Pay-as-you-go
Плюсы	Глубокая специализация на логистике	Удобство интеграции с CRM	Мощные алгоритмы оптимизации	Гибкость и быстрое развертывание
Минусы	Высокий порог входа	Ограниченная логистическая функциональность	Сложность внедрения	Ограниченная кастомизация
Поддержка	24/7 с SLA	Бизнес-часы	Премиум-поддержка	Чат/email поддержка
Масштабируемость	До 1000+ пользователей	До 500 пользователей	Неограниченная	До 300 пользователей

Продолжение таблицы 1

Параметр сравнения	4Laogist.com	Битрикс24 (Логистика)	Oracle TMS	LogistiX
Аналитика	Встроенные BI-инструменты	Базовые отчеты	Расширенная аналитика и прогнозирование	Пользовательские дашборды

## 1.5 Функциональные возможности

### 1.5.1 Управление пользователями и доступом

Регистрация и профили:

1) Многоуровневая регистрация с верификацией:

- а) клиенты (email);
- б) сотрудники (корпоративная авторизация);

2) Расширенные профили с:

- а) предпочтениями доставки (клиенты);
- б) квалификациями и допусками (сотрудники);

3) Авторизация и безопасность:

4) Ролевая модель:

5) Иерархия ролей (4 уровней доступа):

- а) клиент;
- б) курьер;
- с) менеджер склада;
- д) администратор.

### 1.5.2 Управление заказами

Создание заказа:

1) Мультиплатформенное оформление:



- а) веб-интерфейс;
- б) API для интеграций.

### 1.5.3 Складские операции

Прием грузов:

1) Автоматизированная сверка:

- а) по штрих-кодам/QR;

2) Контроль качества:

- а) весовой контроль.

### 1.5.4 Идентификация грузов

Для идентификации грузов используются QR-коды, которые выпускает система. QR-коды содержат информацию по грузу и заказу.

### 1.5.6 Жизненный цикл заказа

Каждый заказ в системе проходит четко структурированный путь, обеспечивающий прозрачность и контроль на всех этапах. Разработан оптимизированный workflow, который автоматизирует ключевые процессы, минимизирует ручные операции и предоставляет полную видимость статусов для всех участников процесса. Отображение жизненного цикла представлено на изображении (Приложение А, Рисунок 4).

## 1.6 Обоснование выбора средств реализации

Для разработки системы выбраны следующие технологии:

- Backend: Ruby on Rails. Выбор обусловлен высокой производительностью, удобством разработки и поддержкой MVC-архитектуры;

- Frontend: React. Позволяет создавать динамичные и интерактивные пользовательские интерфейсы;
- База данных: PostgreSQL. Надежная СУБД с поддержкой сложных запросов и транзакций;
- Деплой: Docker + AWS/Heroku. Обеспечивает масштабируемость и удобство развертывания.

Использование данного стека технологий позволит создать надежное, производительное и легко поддерживаемое решение.

### 1.6.1 Серверная часть. Ruby on Rails

Выбор пал на Ruby on Rails как оптимальный фреймворк для backend-разработки благодаря следующим ключевым преимуществам:

#### 1) Архитектурные преимущества:

- a) полноценная поддержка MVC-паттерна;
- b) Convention over Configuration принцип;
- c) встроенные механизмы RESTful API;
- d) Active Record ORM для работы с БД;

#### 2) Производительность и надежность:

- a) JIT-компиляция в Ruby 3.x;
- b) поддержка многопоточности;
- c) встроенные механизмы кэширования;
- d) безопасность по умолчанию (защита от OWASP Top 10);

#### 3) Экосистема для логистики:

- a) Готовые решения для:
  - i) маршрутизации (Routing Engine);
  - ii) работы с временными интервалами;
- b) Широкий выбор gem'ов для интеграций.

### 1.6.1 Клиентская часть. React + Mui/Joу

Комбинация React с библиотеками MUI и Joy UI обеспечивает:

Эффективность разработки:

1) Готовые компоненты для:

- a) интерактивных карт (Mapbox);
- b) таблиц с виртуализацией (TanStack Table);
- c) сложных форм (Formik + Yup);

2) ThemeProvider для быстрой смены тем.

Особенности реализации:

```
<ThemeProvider theme={logisticsTheme}>
  <JoyCssVarsProvider>
    <MapDashboard>
      <RouteOptimizer />
      <RealTimeTracking />
    </MapDashboard>
  </JoyCssVarsProvider>
</ThemeProvider>
```

Преимущества Joy UI:

1) Специализированные компоненты для:

- a) визуализации маршрутов;
- b) таблиц с большими объемами данных.

### 1.5.3 База данных PostgreSQL

База данных: PostgreSQL

Выбор PostgreSQL обусловлен требованиями к логистической системе:

Ключевые возможности:

- поддержка геопространственных данных (PostGIS);
- оконные функции для аналитики;
- полнотекстовый поиск;
- JSONB для гибких схем данных.

## 2 Проектирование архитектуры системы

Чтобы снизить количество ошибок при проектировании структуры базы данных и бизнес-логики приложения, важно наглядно представить и систематизировать информацию. Для этого были разработаны:

- 1) Диаграмма вариантов использования (отображает взаимодействие пользователей с системой);
- 2) Диаграмма состояний (визуализирует состояния системы и точки перехода);
- 3) Логическая модель данных (описывает сущности и их взаимосвязи без привязки к СУБД);
- 4) Физическая модель данных (определяет конкретную реализацию базы данных с учетом технических ограничений).

Это позволяет четко спланировать архитектуру системы и избежать недочетов на этапе разработки.

### 2.1 Диаграмма вариантов использований

Эта диаграмма была разработана для наглядного представления функциональных требований к системе. С её помощью удалось:

- определить ключевые сценарии взаимодействия пользователей с системой;
- выделить основные функции, которые система должна предоставлять;
- проанализировать, как эти функции удовлетворяют потребности пользователей;
- позволяет определить роли пользователей.

Таким образом, диаграмма вариантов использования стала важным инструментом для проектирования логики приложения и уточнения его функционала.

Диаграмма вариантов использования представлена изображением (Рисунок 6).

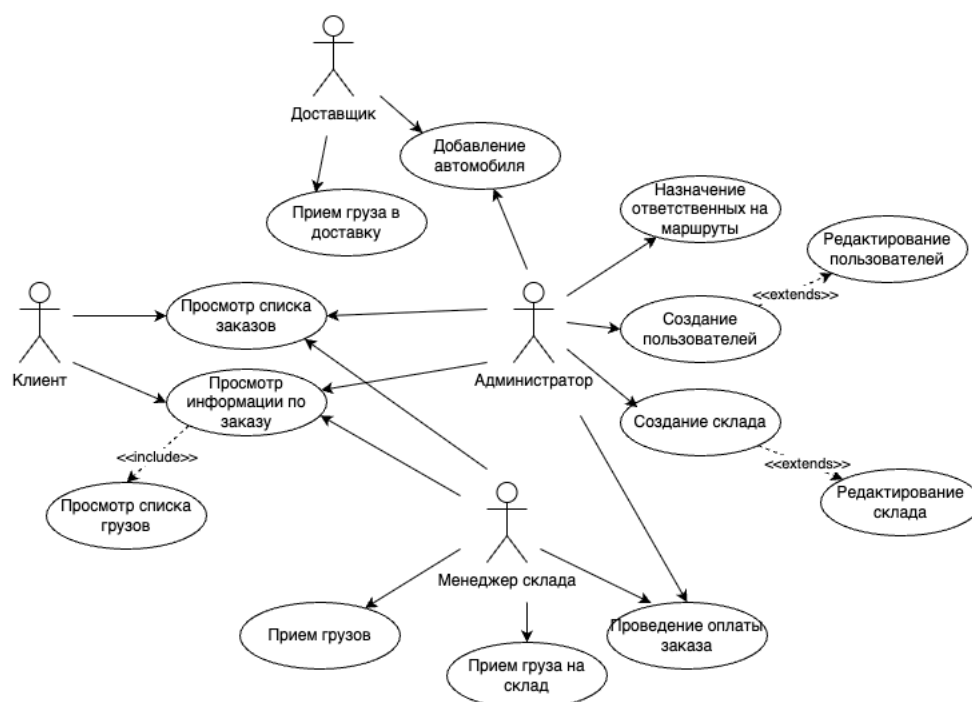


Рисунок 6 – Диаграмма вариантов использования.

## 2.2 Диаграмма последовательностей

Диаграмма последовательности действий была использована для наглядного представления взаимодействия между модулями системы и её участниками в хронологическом порядке. Такой подход необходим для получения целостного представления о поведении системы во времени. Диаграмма позволила подробно отразить порядок вызовов, передаваемые параметры и характер взаимодействия компонентов, что способствовало раннему выявлению и устранению потенциальных проблем в архитектуре.

Этот тип диаграммы помогает обнаружить скрытые зависимости и возможные узкие места, которые могут негативно повлиять на производительность или привести к сбоям в работе системы. Проведённый анализ на основе диаграммы последовательности повысил качество проектных решений и позволил значительно сократить время, затрачиваемое на выявление и исправление ошибок в процессе тестирования и внедрения.

Указанные диаграммы приведены в (Приложение А. Рисунок 1–4).

### 2.3 Логическая модель данных

Логическая модель данных представляет собой абстрактное описание структуры информации в системе, включающее сущности, их атрибуты и взаимосвязи между ними, без учета технических аспектов хранения. Она играет ключевую роль в проектировании информационных систем, позволяя формализовать и организовать данные, необходимые для работы системы. Благодаря ей обеспечивается целостность, согласованность и корректность данных, что существенно упрощает последующую разработку физической модели, уже учитывающей конкретные технологические и аппаратные условия хранения.

Кроме того, логическая модель данных способствует упрощению и ускорению процесса проектирования, повышая качество и надежность итоговой системы. Она позволяет выявить и устранить потенциальные ошибки на ранних этапах разработки, улучшить согласованность данных и создать эффективную архитектуру для обработки и хранения больших объемов информации.

Логическая модель представлена на изображении (Приложение Б. Рисунок 1).

### 2.4 Физическая модель данных

Физическая модель данных была разработана для конкретизации структуры данных с учетом особенностей реализации в СУБД PostgreSQL. При построении данной модели было определено, как именно данные будут физически храниться, индексироваться и управляться в базе данных. Это включало описание таблиц, столбцов, типов данных, индексов, первичных и внешних ключей, а также взаимосвязей между таблицами.

Разработка физической модели данных позволила повысить производительность системы, обеспечить эффективное хранение информации и быстрый доступ к ней. Модель также послужила основой для создания и настройки

объектов базы данных в рамках проектируемой системы, что значительно упростило процесс интеграции с существующей инфраструктурой.

Процесс создания модели включал анализ и выбор оптимальных структур хранения с учетом специфики PostgreSQL, а также использование профильной литературы. Определение ключей и связей между таблицами обеспечило соблюдение целостности и согласованности данных в системе.

Физическая модель данных представлена на изображении (Приложение Б. Рисунок 2).

### 3 Реализация системы

#### 3.1 Выбор и обоснование алгоритмов

Разрабатываемое приложение в соответствии с техническим заданием представляет из себя клиент-серверное приложение. Для серверной части приложения использован Ruby и фреймворк Ruby on Rails, в качестве базы данных PostgreSQL, Redis в качестве кэш-хранилища. Для клиентской части приложения использовался фреймворк React.

##### 3.1.1 Ruby on Rails (RoR)

Ruby — это интерпретируемый объектно-ориентированный язык программирования, известный своей лаконичностью и читаемостью. Его основные преимущества:

- простота и выразительность — код легко читается и пишется;
- объектно-ориентированность — всё является объектом, включая примитивы;
- большое сообщество и множество библиотек (гемов);
- поддержка метапрограммирования — позволяет писать гибкий и расширяемый код.

Ruby on Rails — это популярный веб-фреймворк, написанный на Ruby, следящий принципу "Convention over Configuration" (соглашения важнее конфигураций) и "Don't Repeat Yourself" (не повторяйся). Для изучения особенностей работы с Ruby on Rails использовалась литература [1-3].

Основные возможности RoR:

- MVC-архитектура (Model–View–Controller) — разделение логики приложения;
- автоматическая генерация кода (генераторы scaffolding);
- интеграция с базой данных через Active Record;
- маршрутизация, обработка запросов, RESTful API;



- система миграций для управления схемой БД.

В Rails используется ORM Active Record:

- модели соответствуют таблицам БД;
- позволяет выполнять SQL-запросы с помощью Ruby-методов (`User.find(1)`, `Post.where(published: true)`);
- миграции позволяют версионировать структуру базы данных.

Rails хорошо работает с PostgreSQL, включая поддержку JSONB, индексов, транзакций и расширений.

Для выполнения долгих или периодических задач используются фоновые воркеры. Распространённые библиотеки:

- Sidekiq — асинхронная очередь заданий, использующая Redis;
- позволяет выполнять задачи в фоне (например, отправку писем, обработку данных);
- интегрируется с Rails и Active Job.

### 3.1.2 PostgreSQL

PostgreSQL — это объектно-реляционная система управления базами данных с открытым исходным кодом, которая отличается высокой надёжностью, масштабируемостью и соответствием стандартам SQL. Для изучения особенностей работы с PostgreSQL использовалась литература [4].

Преимущества PostgreSQL:

- надёжность и стабильность — используется во многих критически важных системах;
- расширяемость — поддержка пользовательских типов данных, функций, индексов и расширений (например, PostGIS для геоданных);
- поддержка ACID — транзакции обеспечивают согласованность и целостность данных;

- мощный язык запросов SQL — включая оконные функции, CTE (with-запросы), подзапросы;
- поддержка JSON и JSONB — позволяет использовать PostgreSQL как гибридную реляционно-документную СУБД;
- индексация — поддержка различных типов индексов: B-tree, GIN, GiST, BRIN и др;
- репликация и масштабируемость — встроенная поддержка потоковой репликации, логической репликации, шардирования через сторонние решения.

Использование в проекте:

В разработанном приложении PostgreSQL используется как основная реляционная база данных для хранения информации о пользователях, записях и связанной бизнес-логике. Благодаря хорошей интеграции с Ruby on Rails через ORM Active Record, осуществляется удобное взаимодействие с базой данных на уровне моделей.

Также применяются миграции, которые позволяют версионировать схему базы данных, а транзакции и индексы обеспечивают надёжность хранения и высокую производительность при больших объёмах данных.

### 3.1.3 Redis

Redis (REmote DIctionary Server) — это высокопроизводительное хранилище структур данных в памяти с открытым исходным кодом. Оно поддерживает различные структуры данных, такие как строки, хэши, списки, множества, отсортированные множества и другие.

Преимущества Redis:

- скорость — хранение данных в оперативной памяти обеспечивает время доступа в пределах миллисекунд и ниже;
- поддержка различных структур данных — позволяет реализовать кэширование, очереди, сессии и счётчики;

- простота использования — удобный синтаксис команд, возможность работы напрямую с ключами и значениями;
- поддержка Pub/Sub — встроенная реализация публикации/подписки для реализации real-time механизмов;
- персистентность (по желанию) — поддержка снапшотов (RDB) и логирования изменений (AOF);
- масштабируемость — возможность горизонтального масштабирования с помощью Redis Cluster;
- интеграция с фреймворками — легко подключается к Ruby on Rails и Sidekiq для работы с фоновыми задачами.

Использование в проекте:

В данном приложении Redis используется в следующих целях:

- Очереди задач — в связке с Sidekiq Redis используется как хранилище фоновых задач (например, отправка писем, обработка уведомлений).

Использование Redis позволяет повысить производительность и масштабируемость приложения, особенно при высоких нагрузках и большом количестве параллельных запросов.

### 3.1.4 React

React — это популярная JavaScript-библиотека для создания пользовательских интерфейсов. Она используется при разработке одностраничных приложений (SPA) и позволяет эффективно обновлять и отображать данные без перезагрузки страницы. React широко применяется в коммерческой разработке благодаря своей гибкости, масштабируемости и большому сообществу. Для изучения особенностей работы с React использована литература [5-7].

Почему выбран React:

– компонентная архитектура позволяет разбивать интерфейс на независимые модули, что облегчает разработку, тестирование и повторное использование компонентов;

– интерактивность и производительность React использует виртуальный DOM, что делает обновления интерфейса быстрыми и плавными — критично для динамичных приложений, таких как логистические системы;

– поддержка современных UI-библиотек Совместимость с MUI и Joy UI позволяет быстро создавать адаптивный, функциональный интерфейс с готовыми компонентами (таблицы, карты, формы);

– широкая экосистема и поддержка React активно развивается, имеет множество сторонних библиотек для работы с формами, состоянием, API и др., что ускоряет разработку и повышает надёжность решения.

Таким образом, React обеспечивает оптимальный баланс между удобством разработки, высокой производительностью и качеством пользовательского опыта, что делает его логичным выбором для реализации клиентской части логистической системы.

### 3.2 Руководство программиста

Данный раздел предназначен для разработчиков, которые будут сопровождать, развивать или интегрировать представленную информационную систему. Он содержит описание архитектуры клиентской и серверной частей, рекомендации по развёртыванию, настройке и запуску проекта, а также пояснения к основным модулям и технологиям, использованным при разработке.

Руководство поможет быстрее освоить структуру проекта, понять принципы взаимодействия компонентов и обеспечить бесперебойную поддержку системы в дальнейшем. Особое внимание уделено стандартам кодирования, организации каталогов, использованию сторонних библиотек и механизмам расширения функционала.

### 3.2.1 Структура Ruby on Rails приложения

Ruby on Rails приложение (далее RoR-приложение) имеет устоявшуюся базовую структуру компоновки приложения.

Основные компоненты системы:

1) Ядро приложения (app/):

a) controllers/ - Содержит контроллеры, обрабатывающие HTTP-запросы (например, DeliveriesController);

b) models/ - Бизнес-логика и ORM-модели (например, Delivery, Warehouse);

c) views/ - Шаблоны представления (ERB/HAML/SLIM);

d) services/ - Сервисные классы для сложной бизнес-логики;

e) policies/ - Правила авторизации (Pundit);

f) jobs/ - Фоновые задачи (Active Job);

2) Конфигурация (config/):

g) routes.rb - Определение маршрутов REST API;

h) database.yml - Настройки подключения к PostgreSQL;

i) puma.rb - Конфигурация веб-сервера;

j) environments/ - Окружения (development/production/test);

k) initializers/ - Инициализационные скрипты;

3) Работа с данными (db/):

l) migrate/ - Миграции для изменения структуры БД;

m) schema.rb - Текущее состояние схемы БД;

n) seeds.rb - Начальное заполнение данных;

4) Тестирование (spec/):

o) Модельные тесты (models/);

p) Интеграционные тесты (requests/);

q) Фабрики тестовых данных (factories/);

5) Документация API (swagger/):

- r) API документация (swagger/);
- s) Схемы данных (schemas/);
- t) Описание endpoints (v1/).

Корневые файлы:

1) Управление зависимостями:

- a) Gemfile - Список Ruby-зависимостей;
- b) Gemfile.lock - Фиксированные версии гемов;

2) Конфигурация сервера:

- a) config.ru - Rack-конфигурация;
- b) Dockerfile - Сборка контейнера;
- c) docker-compose.yml - Оркестрация сервисов (PostgreSQL, Redis);

3) Операционные файлы:

- a) Rakefile - Пользовательские задачи;
- b) Makefile - Автоматизация команд;
- c) redis.conf - Конфигурация Redis для кэширования.

Характеристики структуры:

- модульность: Четкое разделение компонентов;
- масштабируемость: Готовность к росту функционала;
- поддержка API: Оптимизирована для RESTful сервисов;
- DevOps-готовность: Docker-конфигурации из коробки [9];

### 3.2.2 Использование AASM для Ruby on Rails

AASM (Acts As State Machine) — это гем (библиотека) для Ruby on Rails, который позволяет удобно реализовывать конечные автоматы состояний (state machines) в моделях ActiveRecord. Он помогает описывать бизнес-логику, связанную с переходами между различными состояниями объекта.

Преимущества AASM:

- 1) простая DSL-синтаксис для описания состояний и переходов;

- 2) явная структура состояний — легко отследить текущий статус объекта;
- 3) поддержка колбэков до/после переходов;
- 4) поддержка сохранения состояний в базе данных.

AASM может использоваться для управления статусами заказов, задач, заявок, платежей и других сущностей, состояние которых меняется во времени. Это упрощает контроль переходов и делает бизнес-логику более читаемой и предсказуемой.

Нижу представлен пример использования AASM для состояний заказа:

```
class Order < ApplicationRecord
  include AASM

  belongs_to :sender, class_name: 'User'
  belongs_to :receiver, class_name: 'User', optional: true
  belongs_to :start_warehouse, class_name: 'Warehouse'
  belongs_to :end_warehouse, class_name: 'Warehouse'
  has_many :cargos, dependent: :destroy

  after_create :schedule_status_check

  aasm column: 'status' do
    state :created, initial: true
    state :wait_payment
    state :paid
    state :in_delivery
    state :awaiting_pickup
    state :completed
    state :canceled
    state :return_in_process
    state :returned

    event :cargo_accepted do
      transitions from: :created, to: :wait_payment
    end

    event :pay do
      transitions from: :wait_payment, to: :paid
    end

    event :accept_for_delivery do
```

```

        transitions from: :paid, to: :in_delivery
      end

      event :stock_received do
        transitions from: :in_delivery, to: :awaiting_pickup
      end

      event :complete do
        transitions from: :awaiting_pickup, to: :completed
      end

      event :cancel, guards: [:not_paid?] do
        transitions from: %i[created], to: :canceled
      end
    end
  end
end

```

Использование AASM в модели добавляет ей методы проверки возможности перехода по состояниям (`may_complete?`, `may_pay?`), а так же добавляет методы соответствующие именам событий для изменения статуса состояния.

```

unless @order.may_pay?
  render json: { errors: 'Payment not available' }, status:
:unprocessable_entity
  raise ActiveRecord::Rollback
end

```

### 3.2.3 Авторизация и аутентификация в приложении.

В системе реализована безопасная аутентификация и ролевая авторизация на основе JWT (JSON Web Token). Это позволяет:

- аутентифицировать пользователей через механизм токенов, обеспечивая защищенный доступ к API;
- контролировать права доступа с помощью ролей (например, клиент, менеджер, администратор), хранящихся в модели User.



Ключевые компоненты:

- JWT-токены – используются для верификации пользователей после входа.
- Ролевая модель – определяет доступные действия для каждого типа пользователя.
- Шифрование паролей – с помощью bcrypt для безопасного хранения в БД.

Аутентификация происходит при каждом запросе к защищенным endpoints, а авторизация проверяется через политики (Pundit). Примеры кода и настройки приведены ниже.

### 3.2.3.1 Управление доступами с помощью pundit в Ruby on Rails

Гем pundit предоставляет простой и гибкий механизм для управления правами доступа (авторизации) на основе политик (policies). Позволяет разделять логику проверки прав доступа для разных ролей пользователей.

Использование в проекте:

Создаются политики (например, OrderPolicy) для контроля доступа к действиям (создание, редактирование, удаление) в зависимости от роли пользователя (клиент, менеджер, администратор, курьер).

Пример реализации класса политик представлен ниже:

```
class OrderPolicy < ApplicationPolicy
  class Scope < Scope
    def resolve
      if user.high_rule?
        scope.all
      else
        scope.where('sender_id = ? OR receiver_id = ?', user.id, user.id)
      end
    end
  end

  def index?
    show?
  end
end
```

```

end

def show?
  return @record.sender == user || @record.receiver == user if user.low_rule?

  true
end

def create?
  return record.sender_id == user.id if user.low_rule?

  true
end

def hand_over?
  cargo_accepted?
end

def destroy?
  user.admin?
end
end

```

Данный гем так же предоставляет реализацию для управления списками посредством переопределения метода `resolve` для внутреннего класса `Score`. Используя данную реализацию можно легко определять условные выборки из базы данных.

Использование политики для определения доступа к конкретному экземпляру ресурса осуществляется через вызов метода `authorize`, в методе контроллера, и передачи в качестве параметра экземпляра.

```

def payment
  authorize @order
  # todo some
end

```

Если переданный экземпляр не удовлетворяет параметрам то pundit выбрасывает исключение `Pundit::NotAuthorizedError`, которое имеет глобальный обработчик в `ApplicationController`.

```
rescue_from Pundit::NotAuthorizedError, with: :user_not_authorized

def user_not_authorized(exception)
  render json: { errors: 'You are not authorized to perform this action',
                policy: exception.policy.class.to_s, # Название политики
                action: exception.query }, status: :forbidden # Действие,
которое запрещено
end
```

### 3.2.4 Расширение запросов с помощью гемов kaminari и ransack для RoR

В современных веб-приложениях работа с большими объемами данных требует эффективных инструментов для пагинации, поиска и фильтрации. В данном проекте эти задачи решаются с помощью двух мощных гемов:

- 1) `kaminari` – обеспечивает удобное разбиение данных на страницы с поддержкой метаданных (общее количество записей, текущая страница и т. д.).
- 2) `ransack` – предоставляет гибкий механизм для построения сложных запросов, включая фильтрацию, сортировку и поиск по связанным моделям.

Их совместное использование позволяет:

- 1) Оптимизировать нагрузку на базу данных, загружая только нужные записи.
- 2) Реализовать удобные интерфейсы для пользователей (таблицы с пагинацией, фильтры в CRM).
- 3) Легко интегрировать функционал с API, передавая параметры запроса (page, per, q).

#### 3.2.4.1 Пагинирование запросов с помощью Kaminari в Ruby on Rails

Гем `kaminari` предоставляет удобный и гибкий механизм для пагинации (разбивки на страницы) данных в Rails-приложениях. Он поддерживает работу с

					МИВУ 09.03.04 - 4.000ПЗ	Лист
Изм	Лист	№ докум	Подпись	Дата		34

ActiveRecord и позволяет настраивать отображение элементов на странице, общее количество страниц и стилизацию пагинации.

Использование в проекте:

1) Применяется для разбивки больших списков (например, заказов, пользователей или товаров) на страницы с возможностью переключения между ними;

2) Интегрируется с фронтендом (React) через API, возвращая метаданные (общее количество страниц, текущая страница и т. д.).

### 3.2.4.1 Поиск и фильтрация с помощью Ransack в Ruby on Rails

Гем ransack предназначен для поиска и фильтрации данных в Rails-приложениях. Он позволяет строить сложные запросы к базе данных через простой синтаксис, включая сортировку, фильтрацию по условиям и полнотекстовый поиск.

Использование в проекте:

1) Реализует поиск по моделям (например, фильтрация заказов по дате, статусу или клиенту).

2) Поддерживает сортировку (например, по имени пользователя или дате создания).

3) Интегрируется с API, принимая параметры запроса (q) и возвращая отфильтрованные данные.

Для осуществления поиска к запросу добавляются query-параметры которые представляют из себя сочетание:

q[имя\_параметра\_правило\_поиска]=значение.

Ransack имеет богатый набор правил поиска:

1) Поиск по точному и частичному совпадению (\_eq, \_cont, \_match);

2) Поиск по большему и меньшему значению (\_lt, \_gt, \_lteq, \_gteq);

3) Поиск по присутствию значений (\_null, \_not\_null);

4) И т.д.

Для работы Ransack в модели нужно указать какие поля будут доступны для поиска.

```
def self.ransackable_attributes(_auth_object = nil)
  %w[created_at end_warehouse_id id price receiver_id sender_id
start_warehouse_id status
  updated_at] + _ransackers.keys
end
```

Так же он поддерживает поиск по цепочки связей. Запрос для поиска по цепочки формируется следующим образом:

q[имя\_связи\_имя\_параметра\_параметр\_поиска]=значение.

Пример: /api/users?q[roles\_name\_eq]=courier.

```
def self.ransackable_associations(_auth_object = nil)
  authorizable_ransackable_associations + ['roles']
end
```

В случае если для решения задач не хватает стандартных Ransack-правил, гем предоставляет интерфейс для написания собственных правил. Собственные правила уже предоставляют полное управление запросами. Пример реализации собственного правила представлен ниже. Данное правило выполняет поиск по нескольким таблицам используя join для объединения и поиска складов по которым, по каким-либо причинам, не были созданы автоматически маршруты, либо по маршрутам не назначены ответственные.

```
scope :with_unassigned_or_no_routes, lambda {
  joins('LEFT JOIN routes AS from_routes ON from_routes.start_warehouse_id =
warehouses.id')
  .joins('LEFT JOIN routes AS to_routes ON to_routes.end_warehouse_id =
warehouses.id')
  .where(
    'from_routes.id IS NULL OR from_routes.user_id IS NULL OR
cardinality(from_routes.delivery_days) = 0 OR ' \
```

```

        'to_routes.id IS NULL OR to_routes.user_id IS NULL OR
cardinality(to_routes.delivery_days) = 0'
    )
    .distinct
}

def self.ransackable_scopes(_auth_object = nil)
  %i[with_null_routes with_assigned_routes with_unassigned_or_no_routes]
End

```

### 3.2.5 Тестирование и документирование API RoR

В разработке надежного API критически важны два аспекта:

- 1) Автоматизированное тестирование – для проверки корректности работы endpoints и бизнес-логики;
- 2) Документирование – для удобства интеграции фронтенда и сторонних сервисов.

В проекте эти задачи решаются с помощью следующих инструментов:

- 1) rspec-rails - главный фреймворк для тестирования Rails-приложений. Позволяет писать юнит-тесты, интеграционные тесты и тесты запросов к API;
- 2) Faker - это библиотека для генерации реалистичных фейковых данных (имена, email, адреса, даты и т. д.). Используется при тестировании, заполнении базы демо-данными и создании fixtures;
- 3) Rswag - инструмент для автоматической генерации Swagger-документации на основе RSpec-тестов. Создает интерактивную документацию в формате OpenAPI [8].

### 3.2.6 Структура приложения React

Проект организован по методологии Feature-Sliced Design (FSD), которая разделяет код по бизнес-логике и обеспечивает масштабируемость.

Основные слои (layers):

					МИВУ 09.03.04 - 4.000ПЗ	Лист
Изм	Лист	№ докум	Подпись	Дата		37

1) app/

a) Назначение: Инициализация приложения, глобальные настройки (роутинг, store, стили).

b) Что включает:

i) Конфигурация Redux, Router;

ii) Глобальные провайдеры (ThemeProvider, ErrorBoundary);

2) Назначение: Страницы приложения (например, HomePage, ProfilePage).

a) Особенности:

i) Собирают фичи и виджеты в полноценные экраны;

ii) Не содержат бизнес-логику (только композиция компонентов);

3) features/

a) Назначение: Бизнес-фичи (например, auth, cart, notifications);

b) Что внутри:

i) UI-логика (хуки, стор-слайсы);

ii) Компоненты, специфичные для фичи;

4) entities/

a) Назначение: Бизнес-сущности (например, User, Product, Order).

b) Особенности:

i) Переиспользуемые модели и API-взаимодействие;

ii) Независимы от конкретных страниц/фич;

5) shared/

a) Назначение: Общие низкоуровневые компоненты и утилиты;

b) Что включает:

i) UI-кит (кнопки, инпуты);

ii) Хелперы (formatDate, api);

iii) Константы, типы;

6) widgets/

a) Назначение: Комплексные блоки (например, Header, Sidebar);

b) Отличие от shared:

- i) Содержат собственную логику (например, состояние бокового меню);
- ii) Могут использовать фичи/сущности.



Таблица 2 – Список корневых файлов React-приложения

Файл	Назначение
vite.config.ts	Конфигурация сборки (Vite).
docker-compose.yml	Настройка контейнеризации (Nginx, сервер).
nginx.conf	Конфиг Nginx для раздачи статики и проксирования API.
eslint.config.js	Правила линтинга.
tsconfig.*.json	Настройки TypeScript для клиента/сервера.

Принципы FSD в проекте:

1) Слоистая архитектура:

а) app → pages → features → entities → shared;

2) Изоляция: Фичи/сущности не зависят от страниц;

3) Переиспользование: shared и entities используются во всех слоях.

### 3.2.7 Описание API сервера

API системы разделено на несколько ключевых групп маршрутов, обеспечивающих управление пользователями, заказами, складами, перевозками и другими сущностями. Все маршруты требуют аутентификации через JWT-токен (кроме публичных).

1) Аутентификация

а) POST /api/login

Вход в систему. Возвращает JWT-токен и данные пользователя.

*Параметры:* email, password.

2) Пользователи (users)

- a) GET /api/users Список пользователей с фильтрацией по ролям (Ransack);
- b) GET /api/users/{id} Информация о конкретном пользователе;
- c) PUT /api/users/{id} Обновление данных пользователя (ФИО, документы);
- d) POST /api/users/{id}/add\_roles Добавление ролей пользователю;
- e) POST /api/users/{id}/remove\_roles Удаление ролей;

### 3) Заказы (orders)

- a) GET /api/orders Список заказов с фильтрацией по статусу (Ransack);
- b) POST /api/orders Создание заказа;  
*Параметры:* sender\_id, receiver\_id, start\_warehouse\_id, end\_warehouse\_id.
- c) GET /api/orders/{id} Детали заказа (со статусами, складами, отправителем/получателем);
- d) POST /api/orders/{id}/cargo\_accepted Подтверждение груза и расчет стоимости;
- e) POST /api/orders/{id}/payment Оплата заказа;

### 4) Грузы (cargos)

- a) GET /api/orders/{order\_id}/cargos Список грузов в заказе (с пагинацией);
- b) POST /api/orders/{order\_id}/cargos Добавление груза.  
*Параметры:* size, dimensions, description;
- c) POST /orders/{order\_id}/cargos/{id}/hand\_over  
Изменение статуса груза на "выдан";

### 5) Перевозки (shippings)

- a) GET /api/shippings Список перевозок с фильтрацией по статусу;
- b) GET /api/shippings/{id} Детали перевозки (маршрут, водитель, статус);
- c) POST /api/shippings/{id}/start\_load Начало погрузки;

d) POST /api/shippings/{id}/start\_delivery Начало доставки;

#### 6) Склады (warehouses)

a) GET /api/warehouses Список складов с фильтрацией по городу;

b) POST /api/warehouses Создание склада.

*Параметры:* name, address, city\_id;

c) POST /api/warehouses/{warehouse\_id}/upload\_cargo/{id} Прием груза на склад;

#### 7) Автомобили пользователей (user cars)

a) GET /api/users/{user\_id}/cars Список автомобилей пользователя;

b) POST /api/users/{user\_id}/cars Добавление автомобиля.

*Параметры:* capacity, load\_capacity, name;

#### 8) Города (cities)

a) GET /api/cities Список городов (публичный, без аутентификации).

Общие особенности:

– фильтрация: Для списков (orders, users, shippings) используется Ransack (пример: q[status\_eq]=completed);

– безопасность: Все маршруты (кроме /api/login и /api/cities) требуют заголовка Authorization: Bearer <token>;

– ашибки: Стандартные HTTP-коды (403 — доступ запрещен, 422 — неверные данные).

Полная спецификация доступна в Swagger UI (/api-docs).

### 3.2.8 Описание алгоритма оплаты и распределения грузов по доставкам

Проведение оплаты по заказ осуществляется менеджером вручную, так как по техническому заданию подключение платежного сервиса в систему на данном этапе заказчику не требуется. Оплата выполняется запросом к серверу `POST /api/orders/{id}/payment`. При выполнении данного запроса проверяются доступы пользователя на выполнение данной операции и возможность проведения по оплаты по заказу. После чего запускается метод класса `CargoDistributor#distribute`.

Данный метод выполняет валидацию заказа по наличию в нем грузов для доставки, а также маршрутов доставки на их наличии и назначение ответственных курьеров. Если валидация успешна то запускается цикл распределения каждого груза по доставкам. В ходе цикла по маршруту находится ближайшая доставка в которую может быть размещен груз, если такая доставка не найдена, то создается новая доставка на дату, соответствующую дням перевозки по данному маршруту. После чего груз прикрепляется к доставке.

Графическое исполнение алгоритма представлено на изображении (Приложение В, Рисунок 1).

### 3.3 Руководство пользователя

В данном подпункте описано руководство пользователя для приложения «Система управления доставками».

#### 3.3.1 Авторизация в личном кабинете

Для доступа к личному кабинету, не зависимо от роли пользователя, необходимо пройти авторизацию в системе. Для того чтобы открыть окно ввода данных необходимо нажать на кнопку «Войти» в верхней навигационной панели (Рисунок 7).

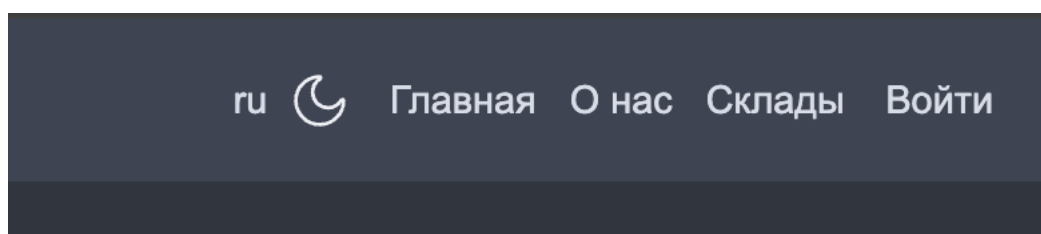


Рисунок 7 – Меню навигации с кнопкой входа в личный кабинет.

Пользователю будет предоставлено модальное окно для ввода его Email и пароля (Рисунок 8).

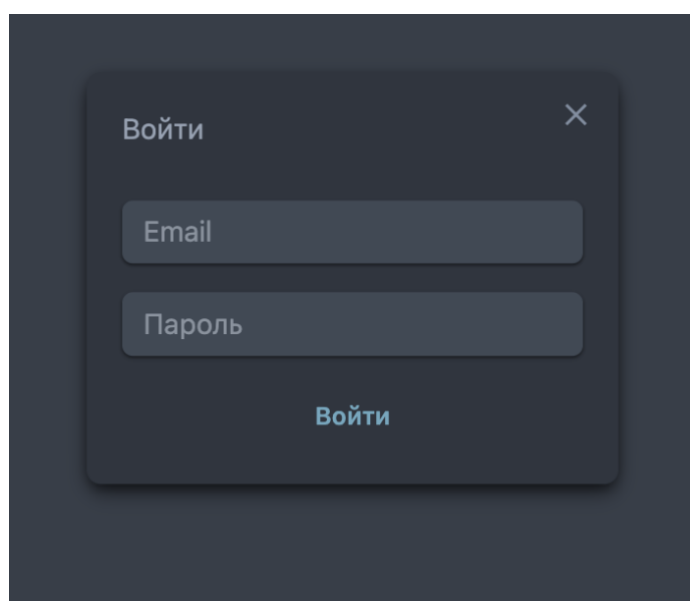


Рисунок 8 – Модальное окно ввода персональных данных для авторизации.

Если пользователем указаны некорректные данные для входа, ему будет выведено сообщение с ошибкой (Рисунок 9).

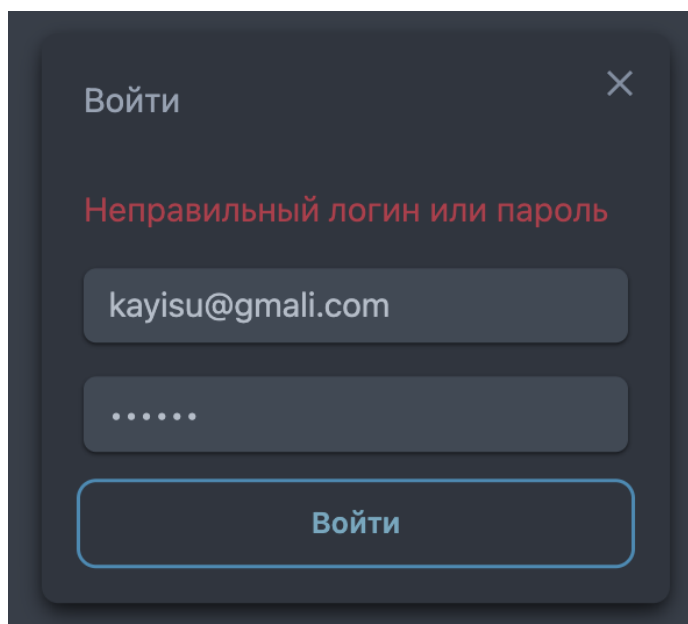


Рисунок 9 – Вывод ошибки при попытке входа.

Если данные введены правильно, то пользователь будет перенаправлен на домашнюю страницу личного кабинета в соответствии с ролью:

- 1) Для клиента, менеджера, администратора – список заказов;
- 2) Для курьера – список активных доставок.

### 3.3.2 Просмотр списка доступных складов

Для всех пользователей (включая гостей) доступна возможность просмотра складов, относительно которых может осуществляться доставка. Для просмотра складов необходимо нажать на кнопку «Склады» в верхней панели приложения, после чего пользователь будет перенаправлен на страницу списка складов (Рисунок 10)

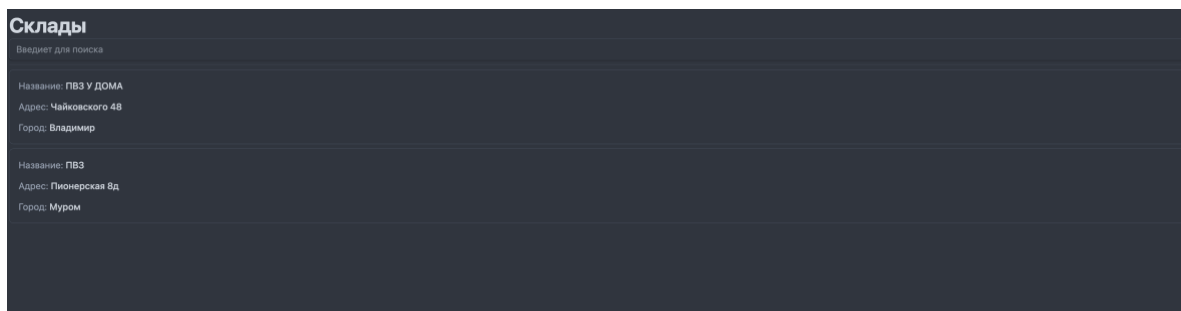


Рисунок 10 – Список складов.

Для удобства пользователей присутствует поиск складов по параметрам: название, адрес, город. Поиск осуществляется с помощью ввода поискового запроса в строку поиска и последующим выводом информации в списке (Рисунок 11).

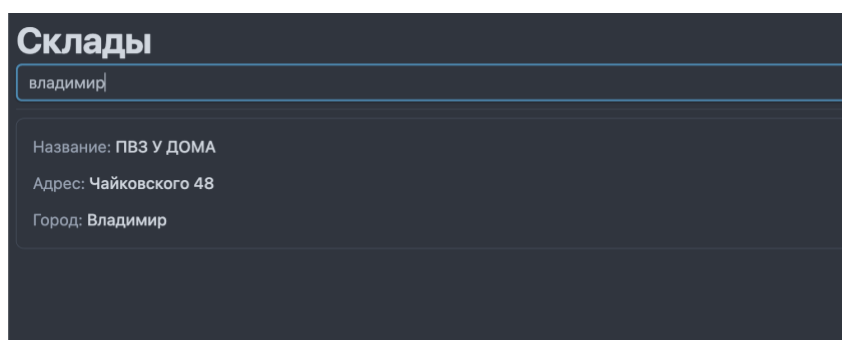
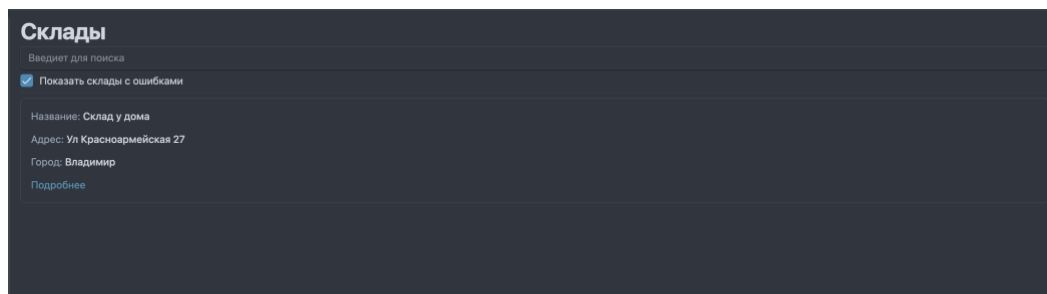


Рисунок 11 – Поиск по городу.

Для администратора на данной странице доступен дополнительный фильтр «Показать склады с ошибками» (Рисунок 12). Данный фильтр отображает все склады у которых:

- 1) Нет назначенных ответственных по маршрутам;
- 2) Есть проблемы с назначением маршрутов.



Д

Рисунок 12 – Список складов с ошибками.

Также администратору доступен функционал перейти на детальную информацию по складу, для этого необходимо нажать кнопку «Подробнее» рядом с интересующим складом.

### 3.3.3 Навигация по блокам в приложении

Для навигации по основным блокам приложения предусмотрено меню, расположенное в боковой панели приложения (Рисунок 13). В зависимости от роли пользователя количество пунктов меню различается.

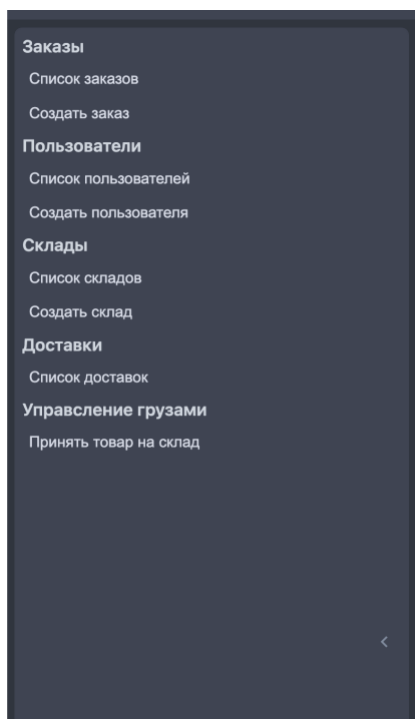


Рисунок 13 – Боковое меню.

### 3.3.4 Список заказов

Для просмотра списка заказов необходимо открыть страницу «Список заказов», для клиентов, менеджеров и администраторов так же можно перейти по ссылке «Главная» (Рисунок 14).



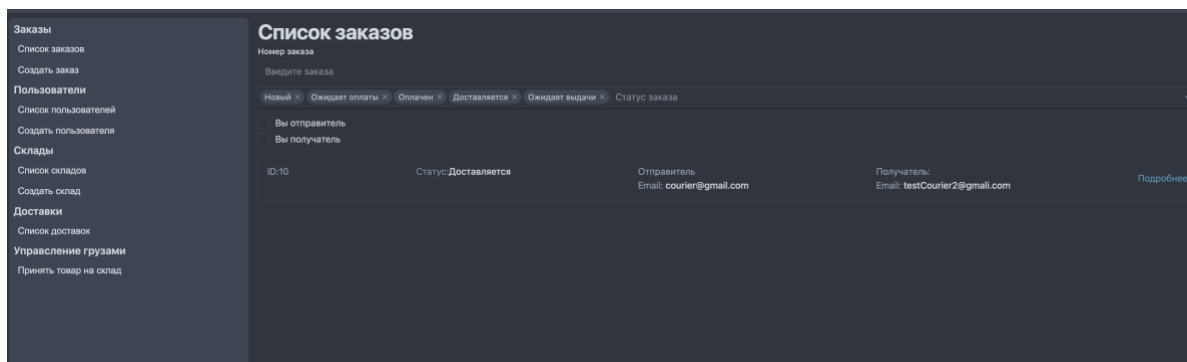


Рисунок 14 – Список заказов пользователя.

Для более быстрого поиска по заказам предоставляются фильтры. В фильтрах можно осуществлять поиск по номеру заказа, статусу заказа, отправителю или получателю (Рисунок 15).

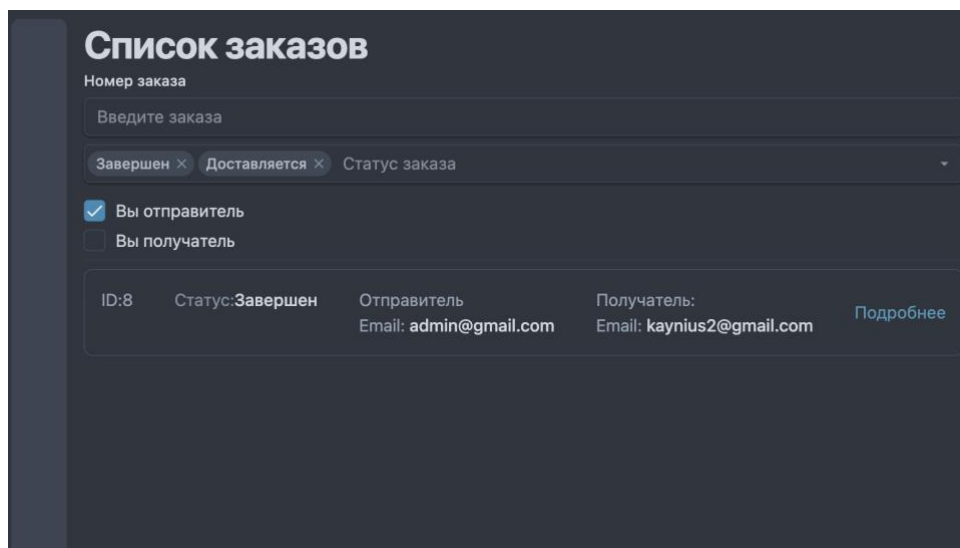


Рисунок 15 – Применение фильтров для заказов.

Для получения более подробной информации по заказу необходимо нажать на кнопку «Подробнее», после чего пользователь будет перенаправлен на страницу информации по заказу.

### 3.3.5 Создание заказа

Для создания заказа менеджеру или администратору необходимо выбрать соответствующий пункт в боковом меню. Для создания заказа необходимо

заполнить форму заказа, указав: начальный и конечный склады и отправителя и получателя заказ (Рисунок 16).

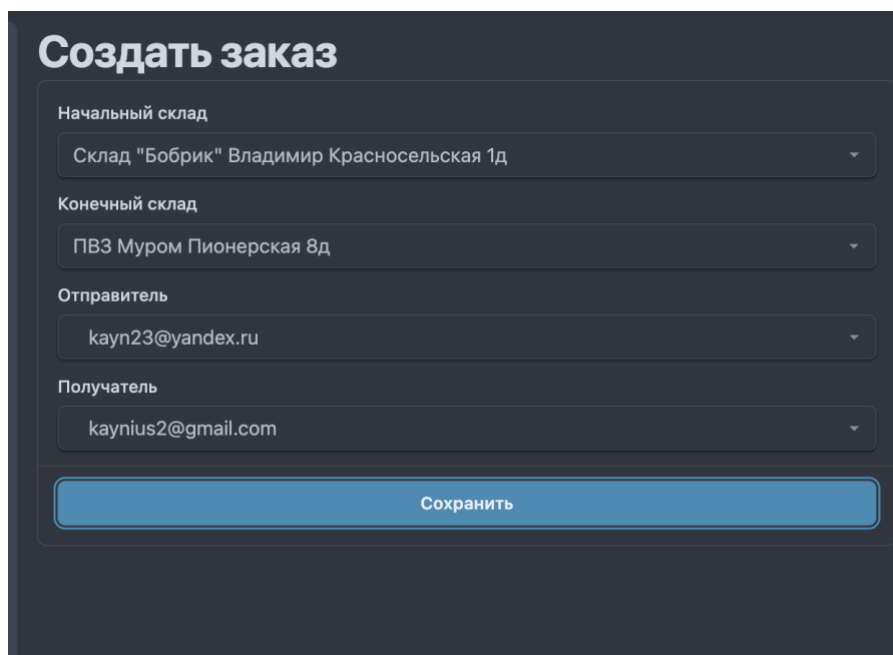


Рисунок 16 - Интерфейс создания заказа.

После создания заказа пользователь будет перенаправлен на страницу информации по заказу.

### 3.3.6 Информация по заказу

Страница информации по заказу включает основные блоки: информация об отправителе и получателе, информация по маршруту, информация по грузам, информация по статусу и цене заказа (Рисунок 17).

Для клиента и курьера информация представляет из себя просто справочную информацию по заказу.

←

Информация по заказу

ID: 10

Статус: Доставляется

Цена: 3800.0

Пользователи

Отправитель

Email: courier@gmail.com  
ФИО: Пушка Петр Александрович  
Номер документа: 1709283648  
[Подробнее](#)

Получатель

Email: testCourier2@gmail.com  
ФИО: Чернышев Анатолий  
[Подробнее](#)

Маршрут

Начальный склад

Название: ПВЗ у ДОМА  
Адрес: Чайковского 48  
Город: Владимир  
[Подробнее](#)

Конечный склад

Название: ПВЗ  
Адрес: Пионерская 8д  
Город: Муром  
[Подробнее](#)

Грузы

Id: 16

Габариты (м3): 3

Вес (кг): 300

Статус: Принят

Рисунок 17 – Детальная информация по заказу.

Для менеджера и администратора имеется расширенный функционал по заказу. В зависимости от статуса заказа предоставляется различный интерфейс для выполнения действий с заказом: добавление грузов в заказ, проведение оплаты (Рисунок 18).

Рисунок 18 – Функционал при статусе заказа «Новый».

В новом заказе менеджер может добавить грузы в заказ. При нажатии кнопки «Добавить груз» будет открыто окно добавления груза в заказ, в котором нужно указать параметры груза (Рисунок 19), после чего нажать кнопку «Добавить».

Рисунок 19 – Добавление груза в заказ.

После добавления груза в заказ, у каждого груза будет уникальный QRCode который можно посмотреть, нажав соответствующую пиктограмму рядом с

информацией о грузе. Данный код печатается и прикрепляется к грузу, и является его идентификатором на протяжении всей доставки.

Так же становится доступна кнопка «Подтвердить грузы» (Рисунок 20), нажатие на которую активирует возможность указать цену заказа (Рисунок 21) и перевести заказ в статус «Ожидание оплаты».

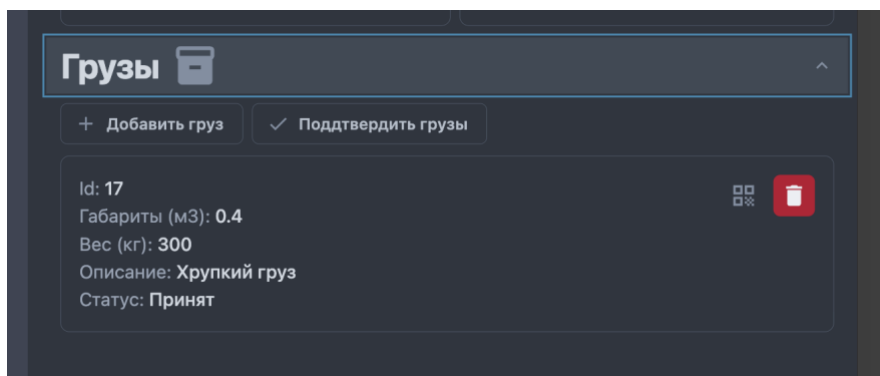


Рисунок 20 – Список грузов и подтверждение грузов.

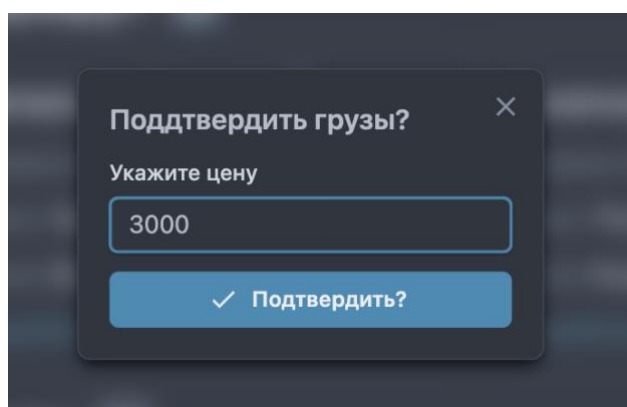


Рисунок 21 – Подтверждение принятия грузов, указание цены за доставку.

Когда заказ находится в статусе «Ожидается оплата» менеджеру доступна кнопка проведения оплаты по заказу (Рисунок 22). После проведения оплаты грузы из заказа автоматически распределяются по доставкам.

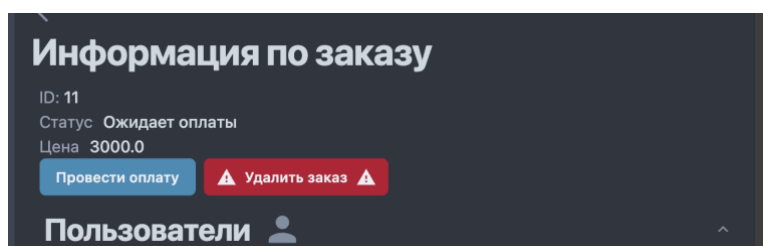


Рисунок 22 – Кнопка проведения оплаты.

Возможность удаления заказа доступна менеджеру или администратору только когда заказ находится в статусе «Новый», «Ожидает оплату». При попытке удалить заказ пользователь должен подтвердить удаление заказа (Рисунок 23), так как после удаления заказа, восстановить его будет невозможно. Данную операцию следует выполнять с максимальной осторожностью.

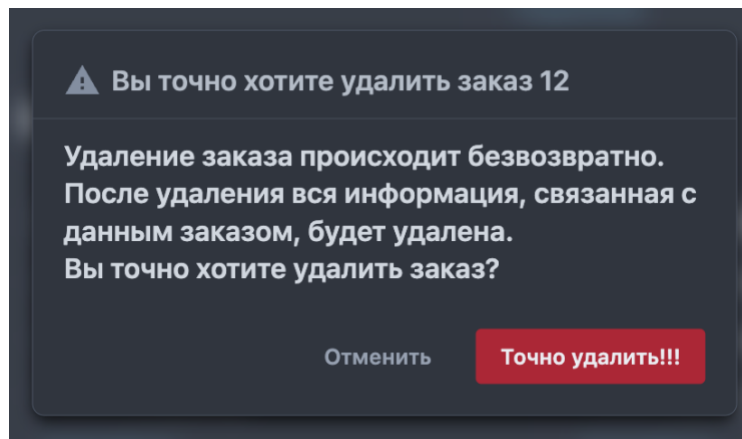


Рисунок 23 – Диалоговое окно удаления заказа.

После того как все грузы из заказа доставлены в конечный пункт выдачи заказ изменяет статус на «Ожидает выдачи», на странице информации о заказе менеджеру и администратору становится доступна кнопка «Выдать груз» (Рисунок 24)

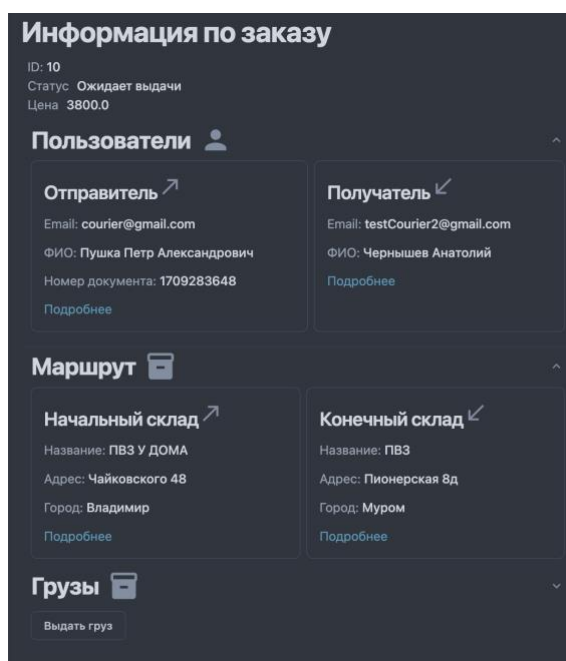


Рисунок 24 – Статус заказа «Ожидает выдачи»

При нажатие на данную кнопку менеджеру или администратору будет открыто окно сканирования qrcode груза и окно подтверждения выдачи (Рисунок 25).

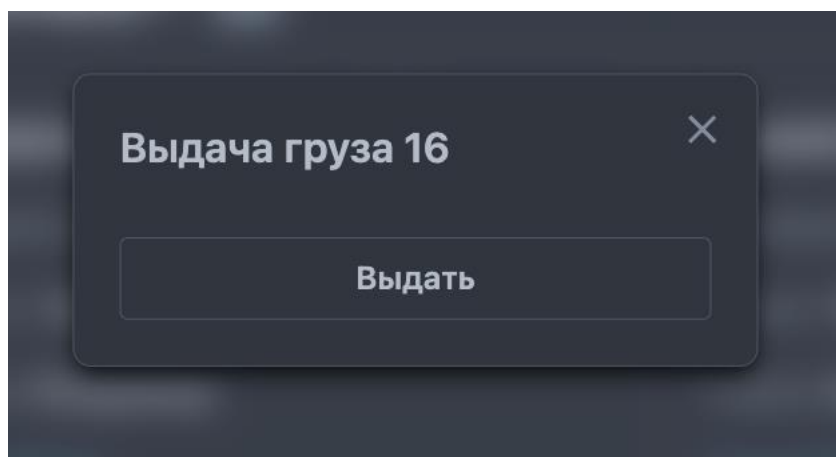


Рисунок 25 – Окно подтверждения выдачи груза.

После того как все грузы из заказа будут выданы заказ автоматически перейдет в статус «Завершен» (Рисунок 26).

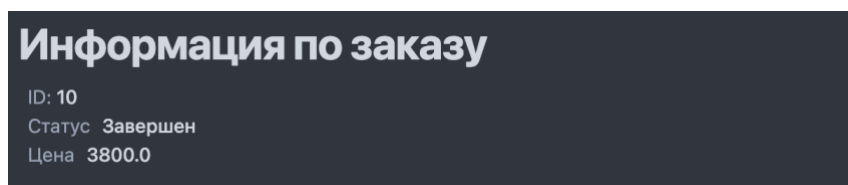


Рисунок 26 – Статус заказа «Завершен».

### 3.3.7 Управление доставками

Список доставок и взаимодействие с ними доступны курьеру и администратору. Страница с списком доставок является домашней страницей для пользователя с ролью курьер. Для того чтобы попасть на данную страницу с других страниц нужно выбрать пункт «Список доставок» в боковом меню.

Список доставок отображает все активные доставки пользователя (Рисунок 27).

### Список доставок

ID 5

Статус Новая

Дата доставки 26.05.2025

Откуда ПВЗ У ДОМА, Чайковского 48, Владимир

Куда РЦ, Заводская 2д, Владимир

[Подробнее](#)

ID 7

Статус Новая

Дата доставки 06.06.2025

Откуда Склад "Бобрик", Красносельская 1д, Владимир

Куда РЦ, Заводская 2д, Владимир

[Подробнее](#)

Рисунок 27 - Список доставок.

Нажав кнопку «Подробнее», пользователь попадает на страницу работы с доставкой (Рисунок 28).

### Доставка 5

ID 5

Статус Новая

Дата доставки 26.05.2025

Откуда ПВЗ У ДОМА, Чайковского 48, Владимир

Куда РЦ, Заводская 2д, Владимир

Ответственный доставщик

Email: courier@gmail.com

ФИО: Пушка Петр Александрович

[Подробнее](#)

Начать загрузку

Рисунок 28 - Информация по доставке.

Когда доставка находится в статусе «Новая» пользователь может начать загрузку товара, после чего у него появится возможность принимать грузы в доставку с помощью сканирования QR-кодов грузов (Рисунок 29).



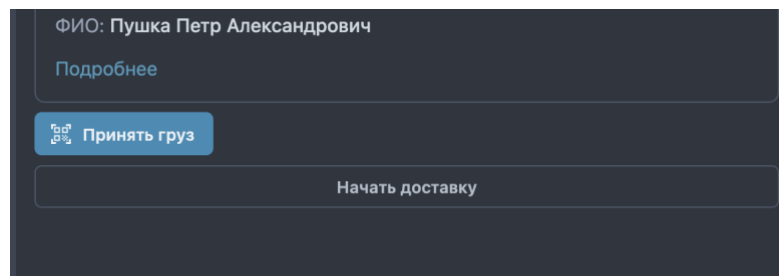


Рисунок 29 - Кнопка «Принять груз»

Приняв груз в доставку выводится оповещение о том что груз загружен (Рисунок 30).

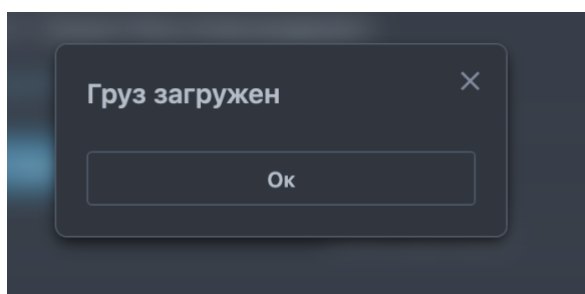


Рисунок 30 - Окно подтверждения загрузки груза.

Завершив загрузку грузов необходимо перевести доставку в статус «Доставляется» нажатием кнопки «Начать доставку».

### 3.3.8 Прием грузов на склад

Прием грузов на склад является важным технологическим процессом. Данная операция в ходе доставки выполняется минимум два раза: при поступлении груза в распределительный центр, при поступлении груза в конечный склад. Выполнение данного процесса доступно только менеджерам и администраторам приложения.

Для получения доступа к данному функционалу необходимо перейти на страницу «Принять товар на склад» в боковом меню.

Чтобы приступить к приему грузов на склад необходимо выбрать склад (Рисунок 31), на котором осуществляется приемка товара. Выбор склада имеет возможности поиска, для которого необходимо начать вводить название, адрес или город. Подтверждение выбора осуществляется нажатием кнопки «Ок».

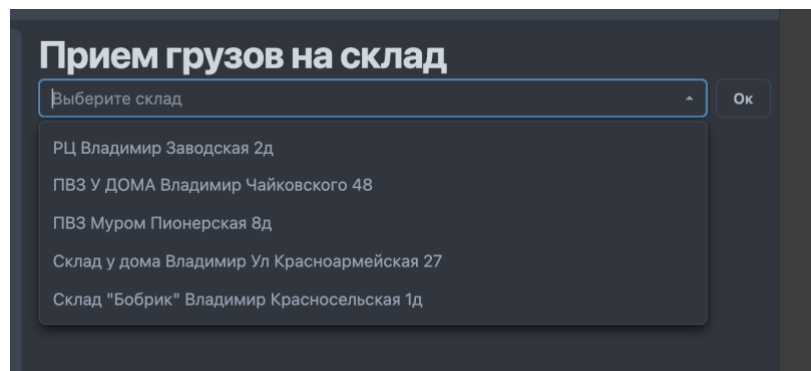


Рисунок 31 - Выбор склада приема грузов.

После чего пользователю станут доступна кнопка «Принять груз» (Рисунок 32) для открытия сканера QR-кодов. Сканером необходимо произвести сканирование грузов, после чего подтвердить прием груза на склад (Рисунок 33).

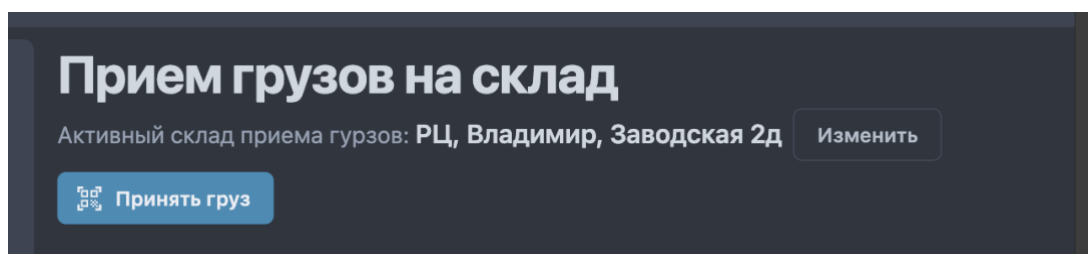


Рисунок 32 - Кнопка приема груза на склад.

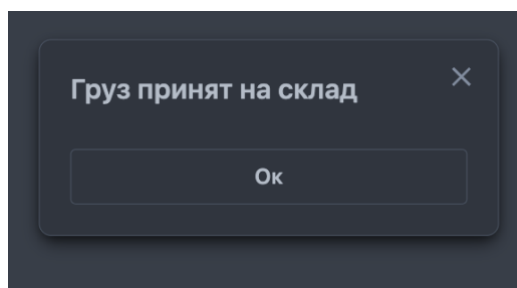


Рисунок 33 - Окно подтверждения приема груза.

Как только будут приняты все грузы из доставки, статус доставки автоматически изменится на «Закончена».

### 3.3.9 Управление пользователями

Добавление и просмотр списка пользователей доступны менеджерам и администраторам системы.

Для того чтобы посмотреть список пользователей необходимо выбрать в боковом меню пункт «Список пользователей», после чего будет представлена страница с списком пользователей (Рисунок 34).

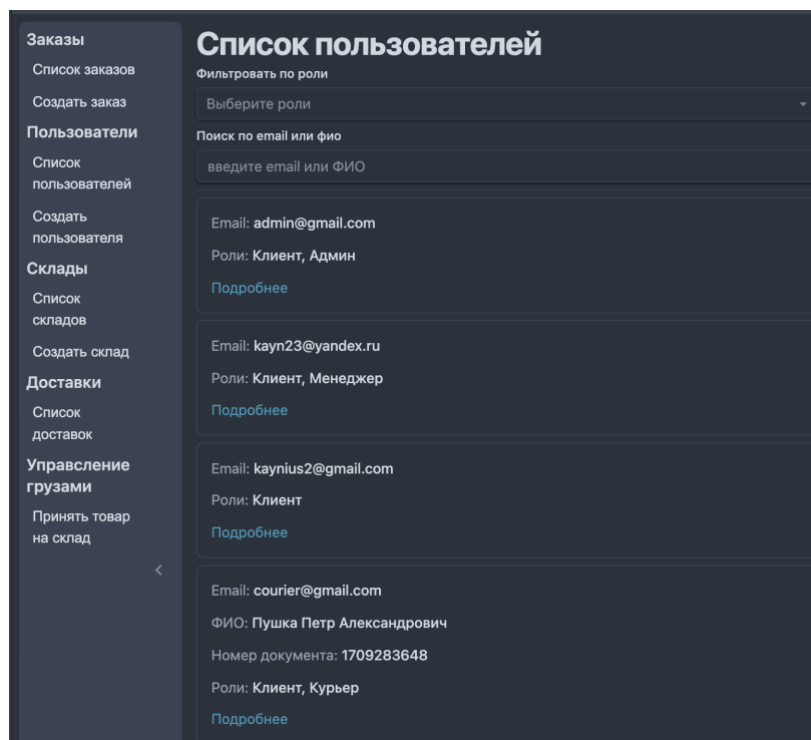


Рисунок 34 – Страница «Список пользователей».

Для удобства поиска определенного пользователя на данной странице предоставлены фильтры и поиск. Поиск доступен по email или ФИО. Так же доступна сортировка по роли пользователя. Пример поиска представлен на изображении (Рисунок 35).

**Список пользователей**

Фильтровать по роли

Курьер × Выберите роли ×

Поиск по email или фио

черн

Email: testCourier2@gmail.com

ФИО: Чернышев Анатолий

Роли: Клиент, Курьер

[Подробнее](#)

Рисунок 35 – Поиск по пользователям.

При нажатии на кнопку «Подробнее» менеджер попадает на страницу информации по конкретному пользователю (Рисунок 36). На данной странице предоставлен функционал для внесения изменений в данные пользователя (Рисунок 37), изменения ролей пользователя (Рисунок 38) и добавление информации по автомобилю для курьеров (Рисунок 39).

←

**Информация о пользователе id: 4**

Email: courier@gmail.com

ФИО: Пушка Петр Александрович

Номер документа: 1709283648

Роли: Клиент, Курьер

[Добавить роль](#) [Изменить данные](#)

[Добавить автомобиль](#)

Название Газель

Объем 50 м3

Грузоподъемность 2000 кг

Активная машина

[Изменить](#)

Название транспортёр

Объем 7 м3

Грузоподъемность 700 кг

[Изменить](#)

Рисунок 36 – Информация по пользователю.

Изменить информация о пользователе

Введите email\*

courier@gmail.com

Введите имя

Петр

Введите отчество

Александрович

Введите фамилию

Пушка

Введите паспортные данные

1709283648

Сохранить

Рисунок 37 – Изменение данных пользователя.

Добавить пользователю роль

Клиент × Курьер × × ▾

✓ Сохранить

Рисунок 38 – Изменение ролей пользователя.

Добавить автомобиль

Название

Введите название

Объем (m3)

Введите объем

Грузоподъемность (кг)

Введите грузоподъемность

☐ Активный

Сохранить

Рисунок 39 – Добавление информации по автомобилю.

Для того чтобы создать нового пользователя в системе необходимо перейти на страницу «Создать пользователя» из бокового меню, после чего заполнить форму нового пользователя (Рисунок 40). После создания пользователя, пароль выдается ему менеджером.

Рисунок 40 – Форма создания нового пользователя.

### 3.3.10 Управление складами

Функционал создания и изменения информации по складам доступна только администратору системы.

Для того чтобы создать новый склад необходимо перейти в соответствующий пункт меню в боковой панели «Создать склад», после чего заполнить форму нового склада (Рисунок 41).

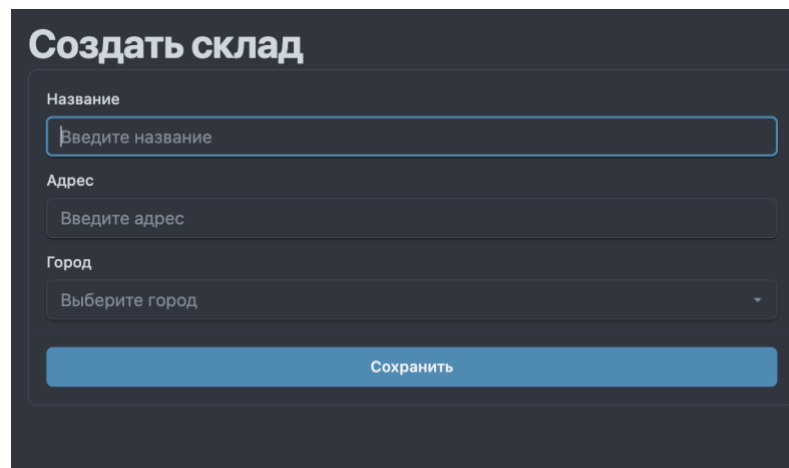


Рисунок 41 – Форма создания нового склада.

После чего администратор будет перенаправлен на страницу информации по складу (Рисунок 42).

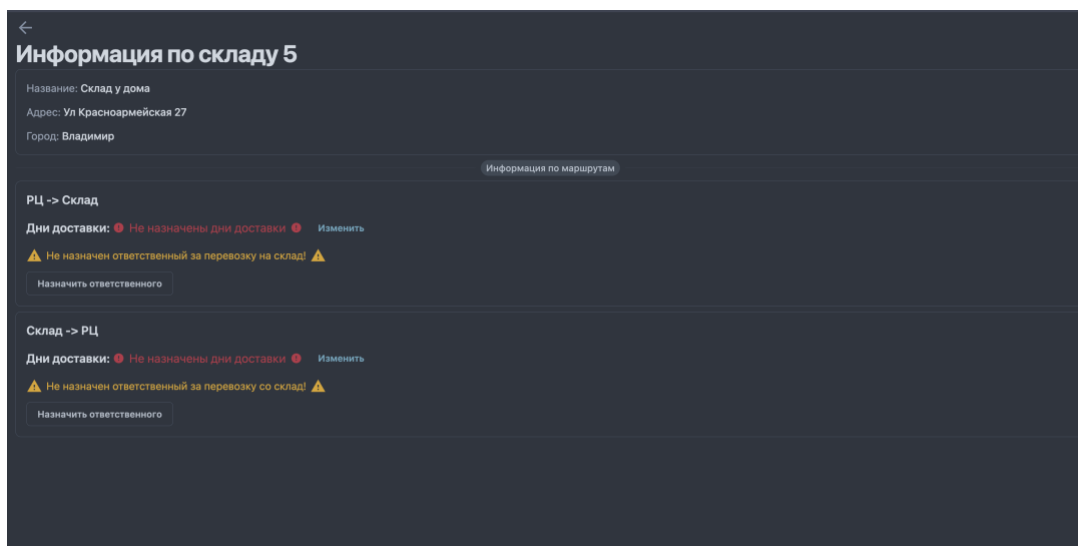


Рисунок 42 - Информация по складу с не назначенными ответственными по маршрутам.

Для того чтобы склад стал доступен для оформления заказов для него необходимо назначить ответственного (Рисунок 43) и назначить дни перевозки (Рисунок 44) для маршрутов до распределительного центра и из распределительного центра.

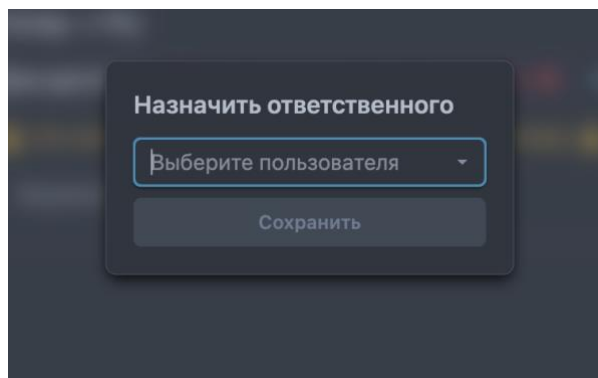


Рисунок 43 - Назначение ответственного по маршруту.

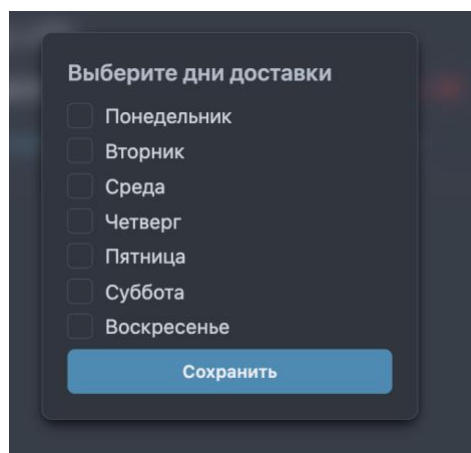


Рисунок 44 - Назначение дней доставки по маршруту.



#### 4 Тестирование системы

В процессе разработки программного продукта особое внимание уделялось контролю качества разрабатываемых компонентов. На ранних и последующих этапах разработки для серверной части системы проводилось модульное тестирование с использованием инструмента gswag, что позволило не только проверить корректность бизнес-логики и API-интерфейсов, но и одновременно сформировать актуальную документацию в формате Swagger/OpenAPI. Такой подход обеспечил быстрое выявление ошибок и недочётов, а также способствовал прозрачности при интеграции с клиентской частью приложения.

Следующим этапом верификации системы стало ручное тестирование, направленное на комплексную проверку функциональной корректности всех компонентов. Данный вид тестирования позволил удостовериться в работоспособности пользовательского интерфейса, корректности взаимодействия между модулями, а также в соответствии реализованной логики требованиям технического задания. Ручное тестирование охватило все ролевые модели пользователей и охватывало как типовые, так и пограничные сценарии использования системы.

Благодаря совокупному применению модульного и ручного тестирования удалось повысить надёжность и стабильность работы системы, а также обеспечить её готовность к эксплуатации в условиях реального бизнеса.

Таблица 3 – Чек-лист тестирования приложения

№	Проверка	Результат
Авторизации и проверка уровня доступа		
1	Авторизация в систему с валидными данными	Успешная авторизация
2	Авторизация в системе с невалидными данными	Сообщение об ошибке "Неправильные логин или пароль"
3	Авторизация с данными клиента	Переход на список заказов

Продолжение таблицы 3

№	Проверка	Результат
4	Авторизация с данными курьера	Переход на список доставок
5	Попытка перехода на страницу без необходимых ролей	Перенаправление на главную страницу пользователя
6	Попытка получения данных без права доступа	Отображение ошибки доступа
Работа с списком заказов		
7	Фильтрация заказов по статусу	Отображаются все заказы с выбранным статусом
8	Поиск заказа по идентификатору	Отображаются заказы подходящие под введеный идентификатор
9	Фильтрация заказов по получателю	Отображаются все заказы в которых пользователь является получателем
10	Фильтрация заказов по отправителю	Отображаются все заказы в которых пользователь является отправителем
11	Переход на детальную информацию по заказу	Открывается страница с детальной информацией по заказу
Работа с заказом		
12	Добавление груза в заказ	Открывается сканер QR-кодов, груз добавляется в заказ.
13	Просмотр QR-кода груза	Открывается модальное окно с изображением QR-кода
14	Подтверждение приема грузов в заказ	Открывается окно для указания цены доставки. Изменяется статус заказа на "Ожидается оплата"
15	Подтверждение оплаты	Изменяется статус заказа на "Доставляется", грузы добавляются в доставки
16	Выдача грузов	Открывается сканер QR-кодов, изменяется статус заказа на "Выполнен"
Создание заказа		
17	Заполнение формы заказа	Создается заказ, перенаправляется на детальную информацию по заказу

Продолжение таблицы 3

№	Проверка	Результат
Работа с списком пользователей		
18	Фильтрация по роли пользователя	Отображаются все пользователи у которых присутствует выбранная роль
19	Фильтрация по email	Отображаются все пользователи, email которых подходит под введенное значение
20	Фильтрация по ФИО	Отображаются все пользователи, ФИО которых подходит под введенное значение
21	Переход на детальную информацию о пользователе	Открывается страница с детальной информации о пользователе
Работа с пользователем		
22	Добавить роль пользователю	Добавляется роль у пользователя
23	Удалить роль пользователю	Удаляется роль у пользователя
24	Изменить данные по пользователю	Изменяются данные в базе у пользователя
25	Добавить новый автомобиль пользователю	Добавляется новый автомобиль, остальные автомобили становятся неактивными
26	Изменить данные по автомобилю	Изменяются данные в базе по автомобилю
27	Изменить статус автомобиля на активный	Запись по автомобилю становится активной, остальные автомобили изменяют статус на не активный
Создание пользователя		
28	Заполнение формы пользователя	Создается запись о пользователе, открывается страница детальной информации по пользователю
Создание склада		
29	Заполнение формы склада	Создается запись о складе, открывается страница детальной информации по складу.

Продолжение таблицы 3

№	Проверка	Результат
Работа с списком складов		
30	Поиск по имеи склада	Отображаются все склады, название которых подходит под введеное значение
31	Поиск по адресу склада	Отображаются все склады, адрес которых подходит под введеное значение
32	Поиск по городу	Отображаются все склады, город которых подходит под введенное значение
33	Фильтрация по складам с ошибками	Отображаются все склады у которых отсутствуют назначенные ответственные, не назначены дни перевозки, или у ответственного нет активного автомобиля.
34	Переход на детальную информацию о складе	Открывается страница с детальной информацией по складу
Работа с складом		
35	Просмотр информации по складу	Отображается описание склада и информация по маршрутам, связанным с данным складом
36	Отсутствие назначенного ответственного за маршрут	Отображается ошибка "Не назначен ответственный за перевозку"
37	Отсутствует назначение дней перевозки	Отображается ошибка "На назначены дни доставки"
38	Нажать изменить дни доставки	Отображается окно изменения дней доставки
39	Изменить дни доставки	Изменяются данные в базе по дням доставок по маршруту
40	Нажать назначить ответственного	Открывается окно выбора ответственного по маршруту
41	Назначить ответственного по маршруту	Изменяются данные в базе по ответственному по маршруту
42	Нажать изменить ответственного	Открывается окно выбора ответственного по маршруту

Продолжение таблицы 3

№	Проверка	Результат
43	Изменить ответственного по маршруту	Изменяются данные в базе по ответственному по маршруту

## Заключение

В ходе выполнения бакалаврской работы была разработана информационная система для автоматизации логистических процессов службы доставки. Основное внимание было уделено созданию удобного и функционального инструмента для работы с заказами, складами, пользователями и доставками, с возможностью масштабирования.

Разработка включала в себя проектирование архитектуры системы, построение логической и физической моделей данных, реализацию клиентской части на основе библиотеки React и серверной части на Ruby on Rails с использованием PostgreSQL, Redis и Docker. Особое внимание было уделено вопросам безопасности, надёжности хранения данных и удобству пользовательского интерфейса.

В процессе реализации были внедрены механизмы ролевой авторизации, взаимодействие с QR-кодами, автоматическое распределение грузов по доставкам, а также интеграция аналитических инструментов. Для серверной части были написаны модульные тесты с использованием фреймворка rswag, что позволило обеспечить не только тестирование API, но и его автоматическую документацию. Также было проведено ручное тестирование пользовательских сценариев, что позволило убедиться в корректной работе системы на всех этапах.

Результатом проделанной работы стал полнофункциональный программный продукт, соответствующий требованиям технического задания и обеспечивающий автоматизацию ключевых операций логистической службы. Разработанная система может быть использована как основа для дальнейшего внедрения и развития цифровых решений в сфере управления доставками и логистикой.

Вся разработанная кодовая база представлена на GitHub [10].

## Список литературы

- 1) Hartl, P. Ruby on Rails Tutorial: Learn Web Development with Rails. – 7th ed. – Boston: Addison-Wesley, 2022. – 784 p.
- 2) Ruby, S., Thomas, D., Hansson, D. Agile Web Development with Rails 7. – Raleigh: Pragmatic Bookshelf, 2022. – 558 p.
- 3) Official Ruby on Rails Documentation [Электронный ресурс]. – URL: <https://guides.rubyonrails.org> (дата обращения: 10.05.2025).
- 4) Obe, R., Hsu, H. PostgreSQL: Up and Running. – 4th ed. – Sebastopol: O'Reilly Media, 2022. – 342 p.
- 5) Banks, A., Porcello, E. Learning React: Modern Patterns for Developing React Apps. – 3rd ed. – Sebastopol: O'Reilly Media, 2023. – 412 p.
- 6) Feature-Sliced Design Documentation [Электронный ресурс]. – URL: <https://feature-sliced.design> (дата обращения: 10.05.2025).
- 7) React Official Documentation [Электронный ресурс]. – URL: <https://react.dev> (дата обращения: 10.05.2025).
- 8) Chelimsky, D., Astels, D., Helmkamp, B. The RSpec Book: Behaviour-Driven Development with RSpec, Cucumber, and Friends. – Raleigh: Pragmatic Bookshelf, 2010. – 450 p.
- 9) Kane, S., Matthias, K. Docker: Up & Running. – 3rd ed. – Sebastopol: O'Reilly Media, 2023. – 298 p.
- 10) [https://github.com/kayn23/diplom\\_docs](https://github.com/kayn23/diplom_docs)

## Приложение А

### Бизнес процессы

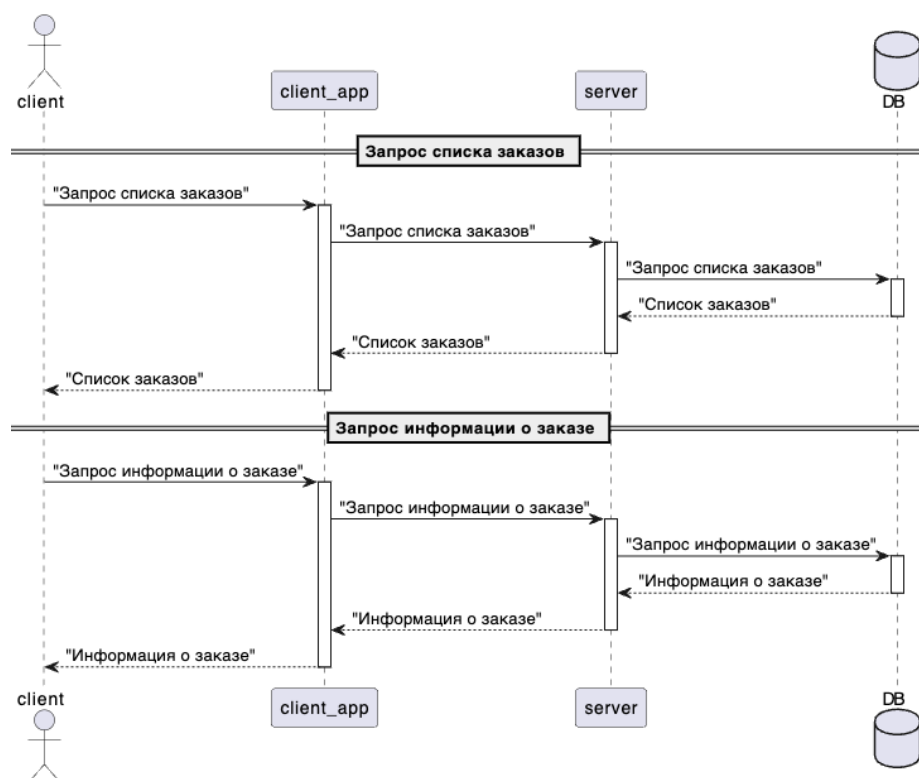


Рисунок 1 – Диаграмма последовательностей пользователя.

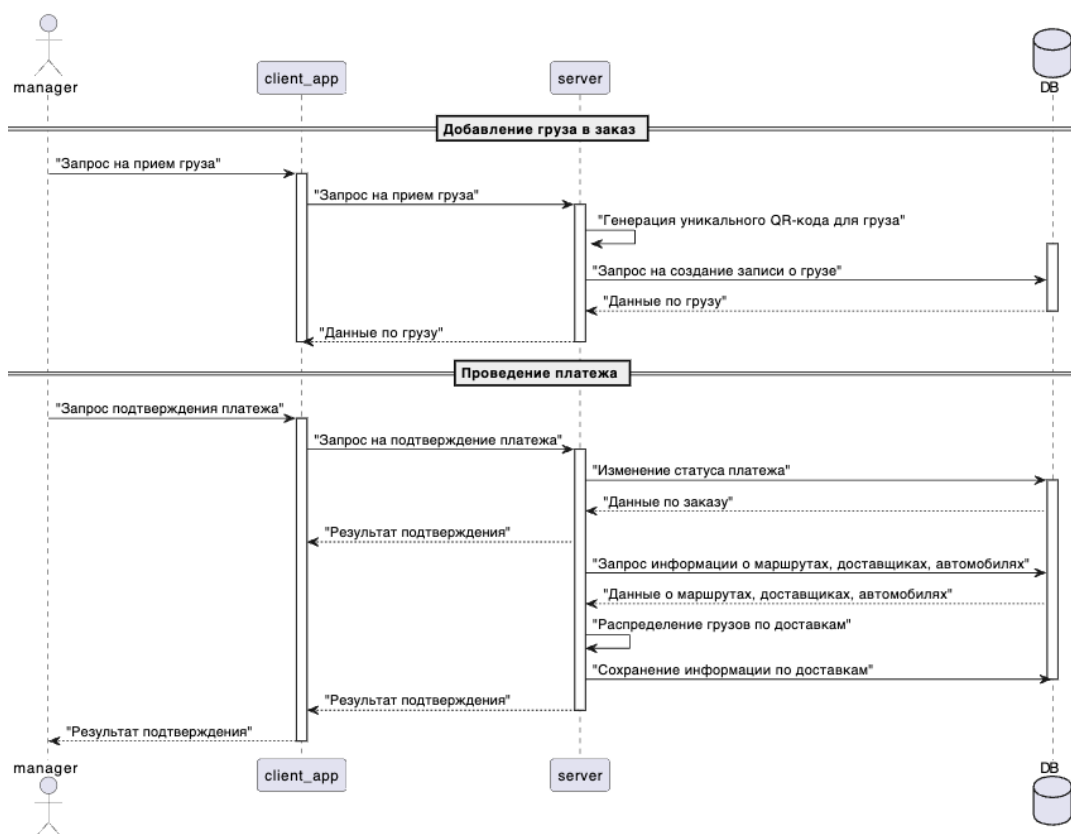


Рисунок 2 – Диаграмма последовательностей для менеджера склада, часть 1.

Изм.	Лист	№ докум	Подпись	Дата



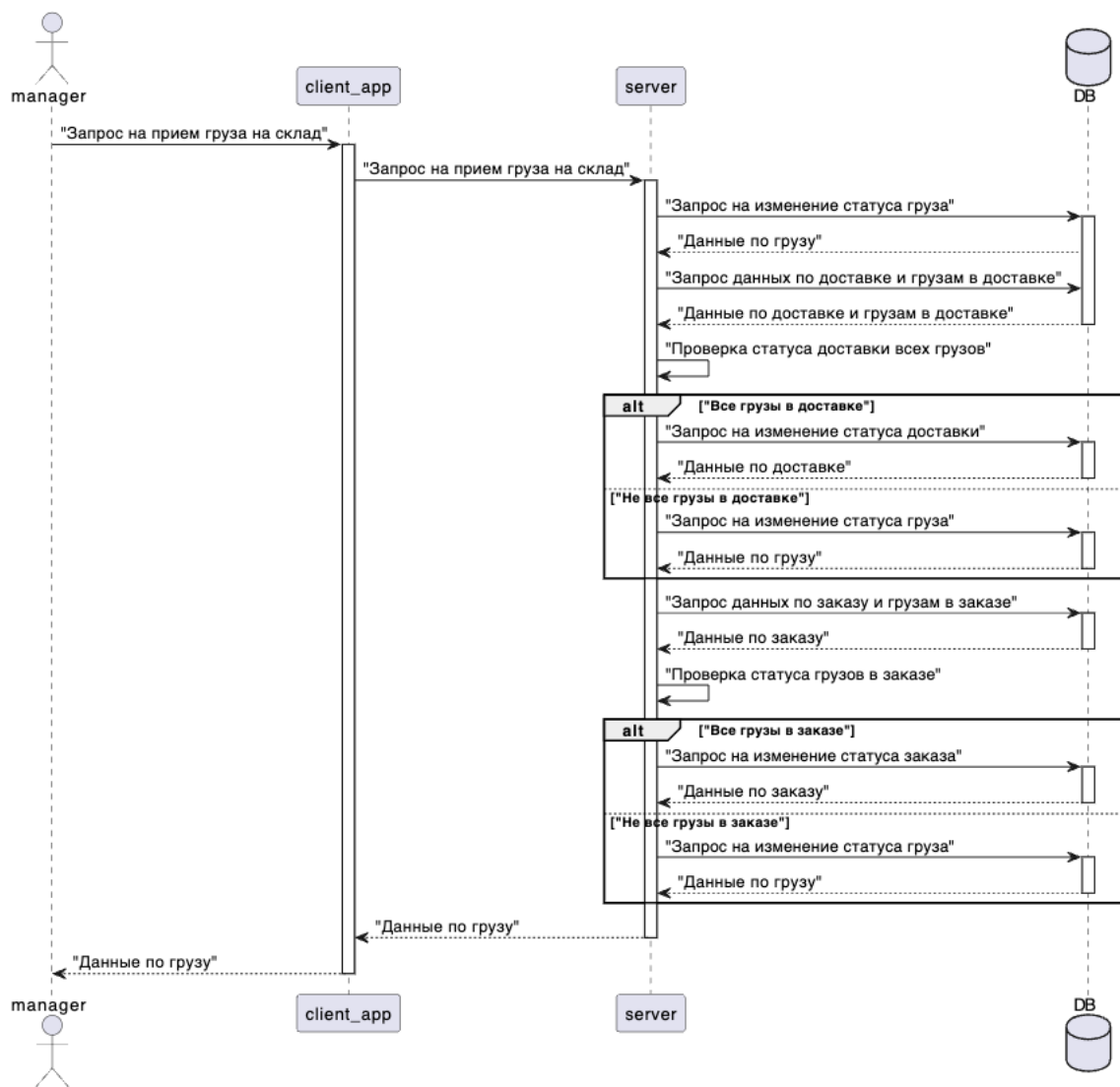


Рисунок 3 – Диаграмма последовательностей для менеджера склада, часть 2.

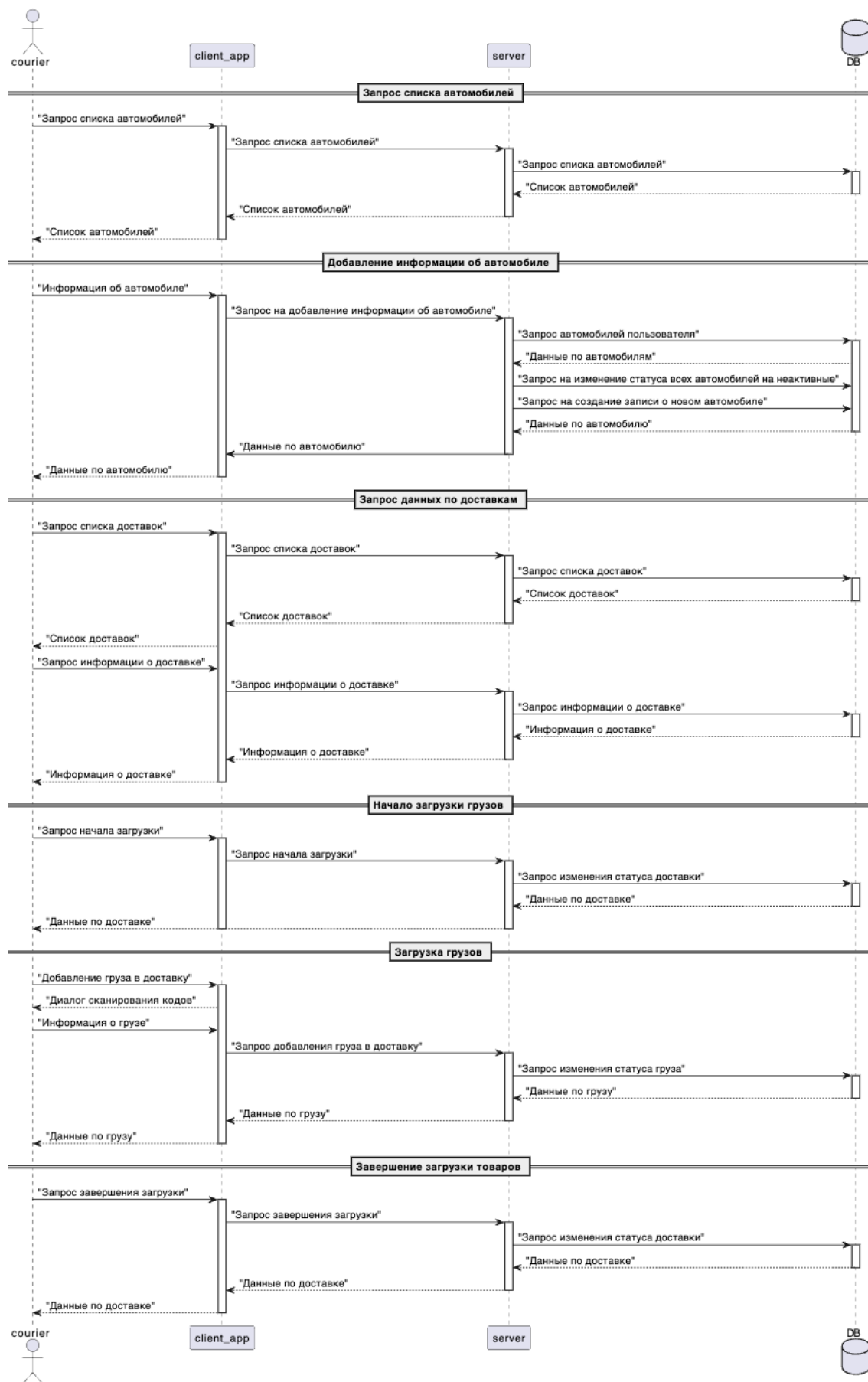


Рисунок 3 – Диаграмма последовательностей для курьера.

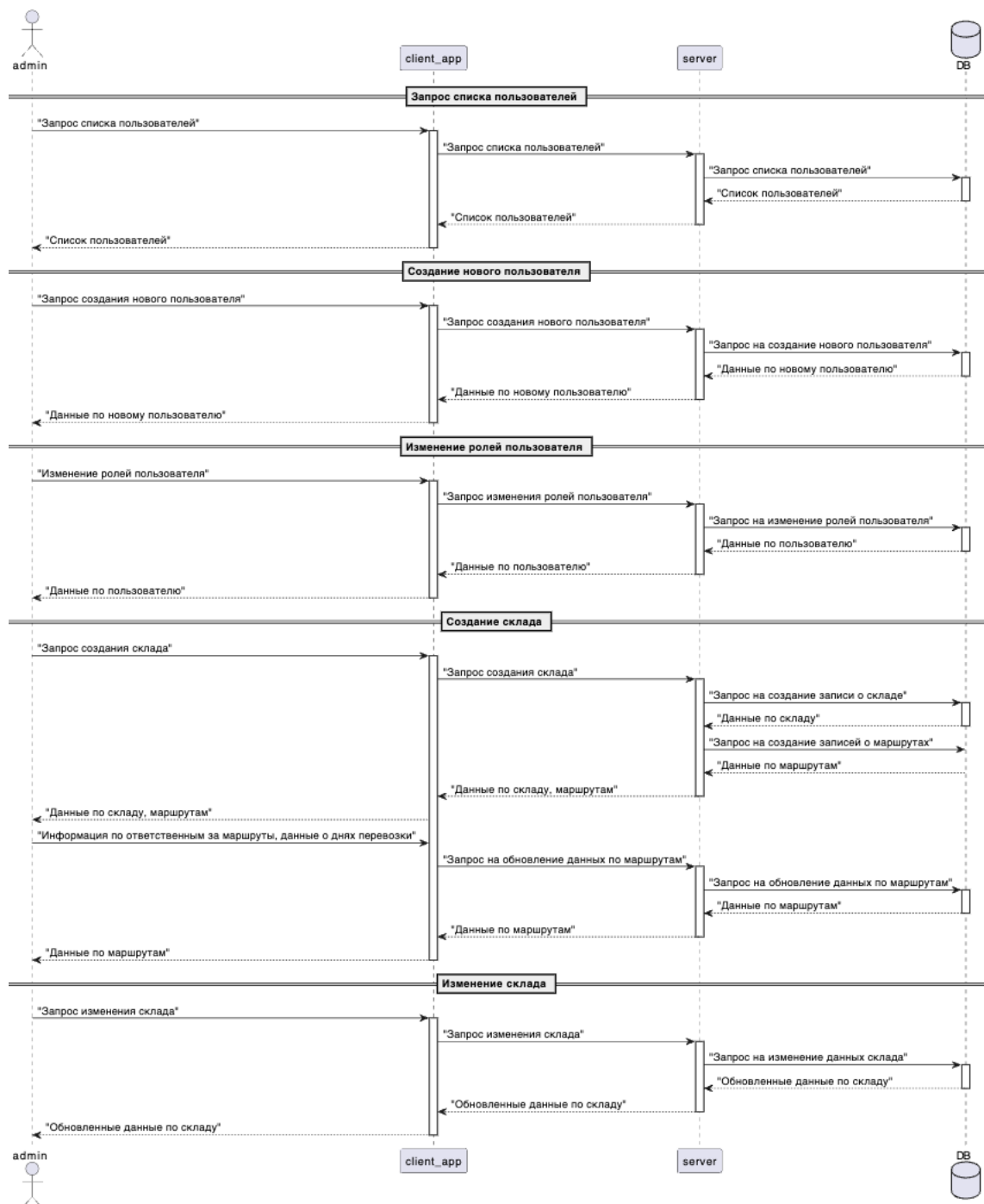


Рисунок 4 – Диаграмма последовательностей для администратора.

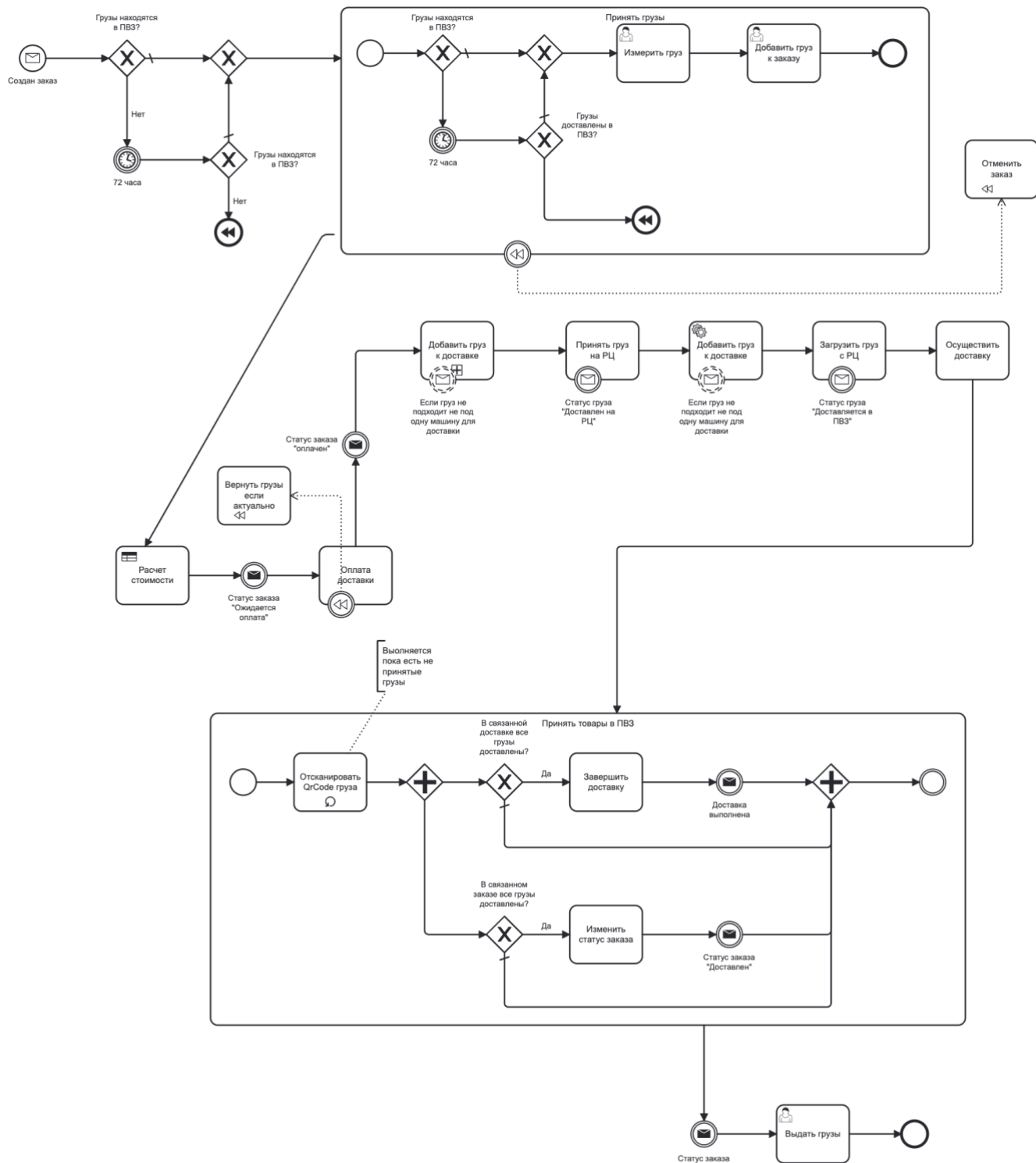


Рисунок 4 – Схема бизнес-процесса.

## Приложение Б

### Моделирование данных

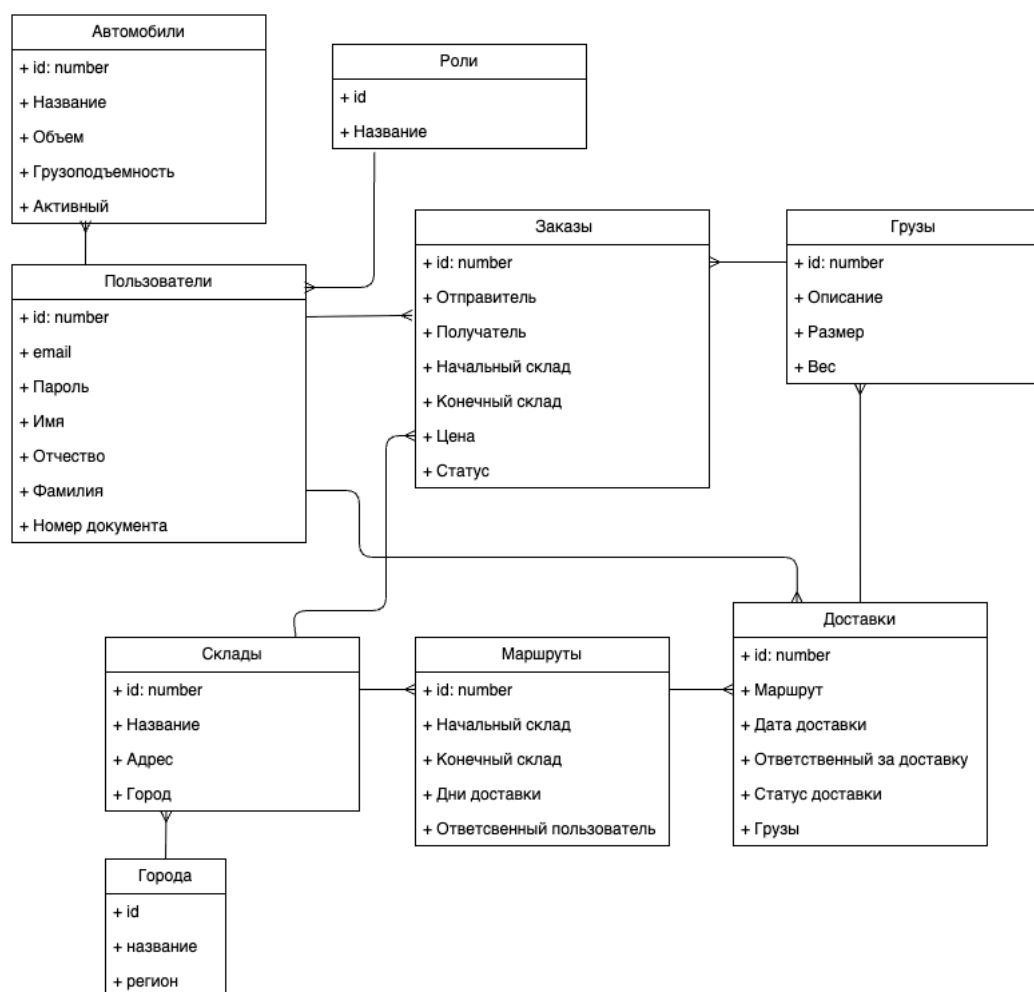


Рисунок 1 – Логическая модель данных

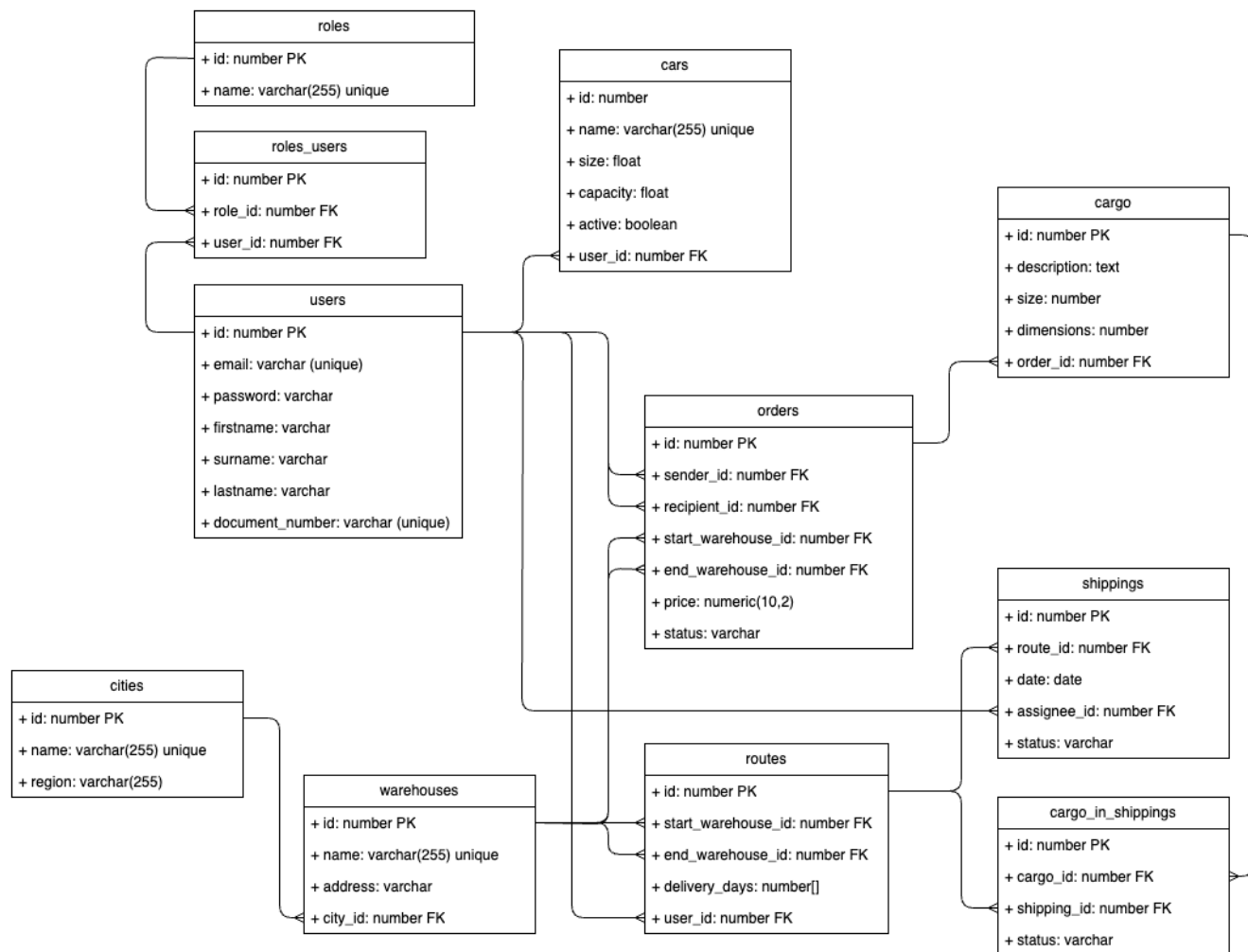


Рисунок 2 – Физическая модель данных.

# Приложение В

## Алгоритмы

Прием оплаты

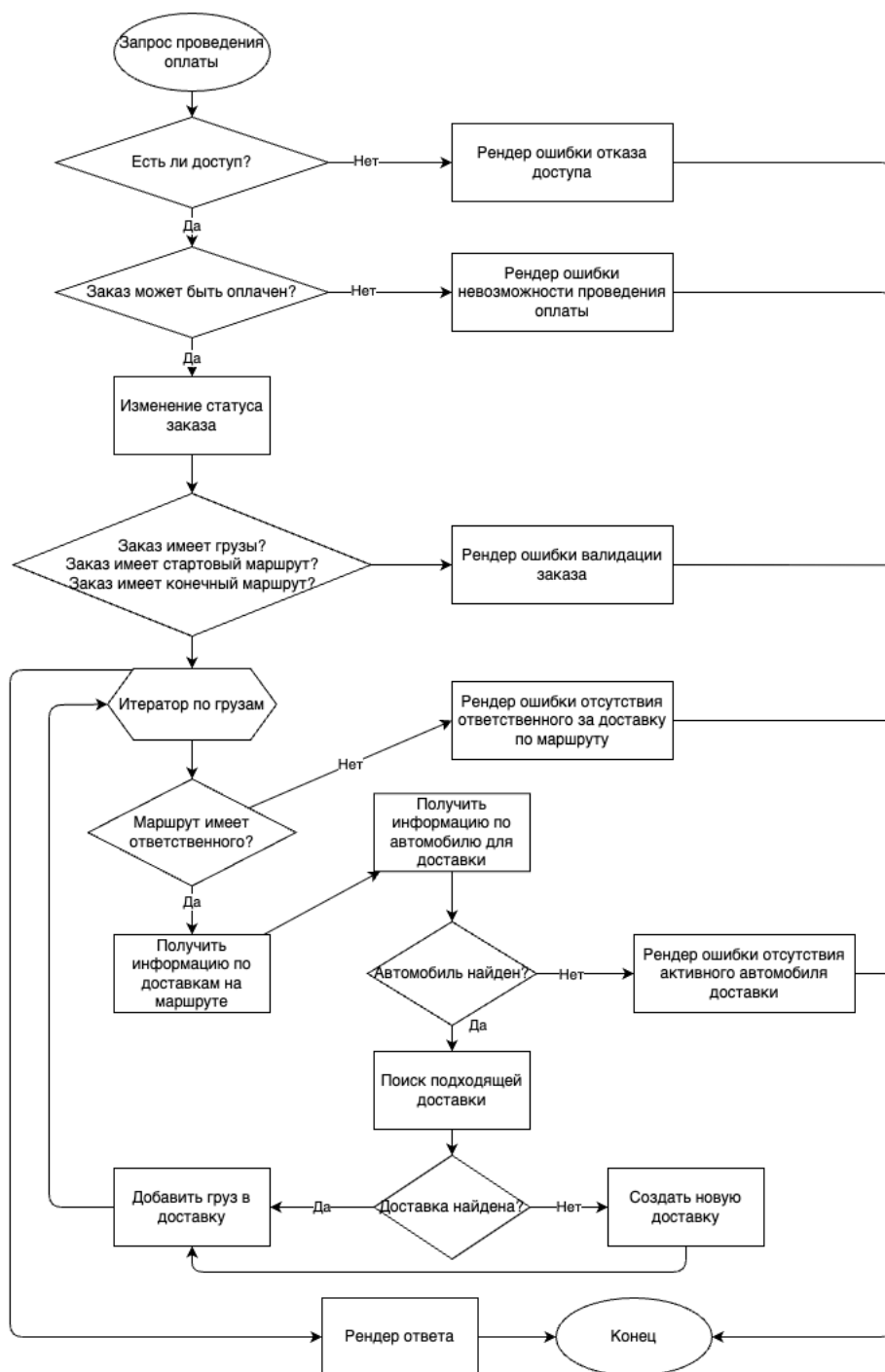


Рисунок 1 – Алгоритм приема оплаты.