ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту

Тема «Разработка веб-приложения для службы доставки».

Руководитель: _____ (С.Г. Ковыляева)

Консультант по экономической части: _____ (Т.Г. Ксенофонтова)

Рецензент: _____ (М.А. Гречкина)

Обучающийся: _____ (М.А. Дубков)

Зав. Отделением: _____ (О.В. Бондаренко)

Диплом защищен с оценкой _____

_____ протокол №_____
дата

Нижний Новгород
2025

Содержание

Введение.....3
1 Анализ предметной области.....5
2 Разработка технического задания.....7
2.1 Область применения.....7
2.2 Назначение и цели создания системы.....7
2.3 Требования к информационной системе.....7
2.4 Условия эксплуатации.....8
2.5 Требования к информационной и программной совместимости.....8
3 Проектирование информационной системы.....10
3.1 Диаграмма вариантов использования.....10
3.2 Информационно-логическая модель базы данных.....16
3.3 Диаграмма последовательности.....23
3.4 Диаграмма состояний.....25
3.5 Диаграмма компонентов.....26
3.6 Диаграмма размещения.....27
4 Разработка информационной системы.....28
4.1 Описание структуры.....28
4.2 Разработка серверной части.....30
4.3 Разработка клиентской части.....56
5 Тестирование информационной системы.....122
5.1 Автоматизированное тестирование с использованием Selenium IDE.....122
5.2 Тестовые сценарии.....122
5.3 Проверка адаптивного дизайна.....124
6 Руководство пользователя.....128
6.1 Введение.....128
6.2 Общее руководство для всех ролей.....128
6.3 Руководство для клиента.....133
6.4 Руководство управляющего.....136
6.5 Руководство администратора.....137
7 Экономический анализ.....139
7.1 Расчёт полезного времени работы.....139
7.2 Обоснование выбора специалистов для разработки.....140
7.3 Расчёт заработной платы и отчислений во внебюджетные фонды.....142
7.4 Расчёт заработной платы для рабочих повременщиков.....143
7.5 Расчёт материальных затрат разработки.....144
7.6 Расчёт суммы платежей за электроэнергию.....145
7.7 Расчёт сумм амортизационных отчислений.....145
7.8 Расчет сметы затрат.....146
7.9 Расчёт дохода от использования системы.....147
7.10 Расчет срока окупаемости.....148
7.11 Выводы.....148
Заключение.....149
Список использованных источников.....150

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Дубков М.А.		
Проверил		Ковыляева С.Г.		
Рецензент		Гречкина М.А.		
Н. контр.				
Утв.				

ТКИС.508120.001ПЗ

Разработка веб-приложения для службы доставки

Лит.	Лист	Листов
Т	2	150

НРТК 4ИСИП-21-1

Копировал

Формат А4

Введение

Современная веб-разработка — это область, в которой сочетаются техническая сложность и стремление к максимальной простоте и удобству для конечного пользователя. Сегодня на рынке представлено множество прикладных средств программирования, существенно упрощающих процесс создания веб-интерфейсов, автоматизирующих рутинные задачи и ускоряющих выход цифрового продукта на рынок. Особенно высокие требования предъявляются к системам, работающим в сфере электронной коммерции и логистики, где важны точность, скорость и надежность.

Одним из ключевых направлений цифровизации бизнеса стало создание специализированных веб-приложений, обеспечивающих автоматизацию логистических процессов, в частности — служб доставки. Разработка таких систем позволяет компаниям эффективно управлять заказами, отслеживать передвижение посылок, оптимизировать маршруты и взаимодействовать с клиентами в режиме реального времени. Современные пользователи ожидают высокой скорости доставки и прозрачности на всех этапах, что требует от разработчиков внедрения умных решений и отказа от устаревших подходов.

В данном дипломном проекте реализовано веб-приложение для службы доставки, предоставляющее пользователю удобный интерфейс для оформления и отслеживания заказов, а сотрудникам — инструменты для учета посылок, планирования маршрутов и аналитики. Основное внимание уделено созданию интуитивно понятного пользовательского интерфейса, надежной архитектуры системы и эффективного взаимодействия с базой данных.

Разработанное веб-приложение позволяет достичь следующих целей:

- автоматизация процессов оформления и отслеживания заказов в службе доставки;
- повышение удобства для пользователей за счёт простой и понятной системы взаимодействия;
- сокращение времени на логистические операции;
- централизованное хранение информации о клиентах, заказах и маршрутах;
- предоставление административного доступа для мониторинга и анализа работы системы.

Для достижения поставленных целей были решены следующие задачи:

- проведение анализа предметной области и выявление ключевых бизнес-процессов;
- проектирование структуры веб-приложения и его интерфейса;
- разработка архитектуры системы, включая клиентскую и серверную части;

- реализация базы данных с учётом масштабируемости и безопасности;
- тестирование функциональности и оценка эффективности приложения в условиях, приближенных к реальным.

Таким образом, актуальность разработки обусловлена растущей потребностью в универсальных, адаптивных и удобных решениях в сфере логистики. Целью данной работы является создание современного веб-приложения для службы доставки, позволяющего упростить и ускорить процесс взаимодействия между клиентами и пунктами доставки. Такой подход особенно актуален в условиях конкуренции и стремительного роста количества онлайн-заказов.

1 Анализ предметной области

Служба доставки — это логистическая система, задачей которой является оперативная и точная передача посылок от отправителя к получателю. С развитием информационных технологий и стремительным ростом электронной коммерции роль цифровых инструментов в данной сфере существенно возросла. Одним из ключевых решений становится создание веб-приложений, автоматизирующих процессы оформления, отслеживания и обработки заказов доставки.

Веб-приложение — это программное обеспечение, работающее в браузере и обеспечивающее доступ к функционалу через интернет. Оно позволяет пользователям (как клиентам, так и сотрудникам компании) взаимодействовать с системой доставки в реальном времени: оформлять посылки, отслеживать их местоположение, получать уведомления.

В контексте службы доставки веб-приложение включает в себя несколько функциональных блоков:

- клиентский интерфейс (оформление и отслеживание посылок);
- интерфейс курьера (управление принятиями и отправками посылок);
- административная панель (вывод анализа работы системы).

Для корректной работы такой системы важно организовать чёткую структуру хранения данных, обеспечить безопасность пользовательской информации и реализацию удобного, интуитивно понятного интерфейса.

Создание веб-приложения для службы доставки включает в себя ряд этапов:

- сбор и анализ требований к функциональности и интерфейсу;
- разработка макетов и прототипов, отражающих логику взаимодействия;
- вёрстка интерфейса с учётом адаптивности под различные устройства;
- программирование клиентской и серверной частей;
- интеграция базы данных, обеспечивающей хранение заказов, маршрутов и пользовательских данных;
- тестирование системы на корректность работы всех компонентов.

Особое внимание при разработке следует уделить пользовательскому опыту (UX), так как конечные пользователи — это люди с разным уровнем технической подготовки. Интерфейс должен быть понятным и функциональным как для клиентов, так и для сотрудников службы доставки.

Основные задачи веб-приложения службы доставки:

- автоматизация приёма и обработки посылок;
- предоставление пользователям информации о статусе доставки;
- формирование маршрутов доставки;
- предоставление отчётности и статистики для администраторов.

Современные службы доставки нередко используют собственные цифровые платформы, однако для малого и среднего бизнеса часто недоступны решения крупного масштаба. В таких случаях требуется создание адаптированного веб-приложения, которое отвечает конкретным потребностям компании, позволяет сократить операционные издержки и улучшает сервис.

Таким образом, анализ предметной области показывает, что создание веб-приложения для службы доставки является актуальной задачей, способной существенно повысить эффективность логистических процессов и улучшить качество обслуживания клиентов.

2 Разработка технического задания

2.1 Область применения

Разрабатываемое веб-приложение предназначено для использования в логистических компаниях и службах доставки. Система предназначена для автоматизации процессов приёма, обработки и отслеживания заказов, а также построения логистических маршрутов и взаимодействия между клиентами и пунктами сортировки.

2.2 Назначение и цели создания системы

2.2.1 Назначение системы

Разрабатываемое веб-приложение будет включать две основные части:

- Клиентскую часть — веб-интерфейс, через который пользователи смогут взаимодействовать с системой;
- Серверную часть — осуществляет взаимодействие с базой данных.

2.2.2 Цели создания системы

Цель разработки веб-приложения заключается в создании универсальной и доступной платформы, позволяющей автоматизировать основные бизнес-процессы службы доставки.

2.3 Требования к информационной системе

2.3.1 Исходные данные

- Сведения о пользователях (паспортные данные, электронная почта);
- Сведения о пунктах сортировки (адрес, управляющий);
- Сведения о посылках (код отслеживания, пункт отправки, пункт назначения, вес, объем).

2.3.2 Функциональные требования

- ведение базы данных;
- удаленный доступ;
- разграничение прав доступа;
- адаптивный web-дизайн;
- регистрация/авторизация клиентов;
- наличие личного кабинета;
- наличие калькулятора стоимости доставки;
- возможность оформления доставки клиентом;
- возможность отслеживания этапа доставки клиентом;
- наличие карты, отображающей текущее местоположение посылки;
- наличие дашборда с графиками эффективности пунктов сортировки в

личном кабинете администратора;
- возможность изменить статус посылки управляющим пункта сортировки;
- функционал автоматического расчета наиболее эффективного маршрута доставки посылки;
- наличие оповещения о том, что посылка прибыла в пункт назначения на электронную почту клиента.

2.4 Условия эксплуатации

2.4.1 Требования к техническому и программному обеспечению

Для корректной работы информационной системы необходимо выполнение следующих требований:

- операционная система: MS Windows NT 8/10/11/Server 2016, Linux;
- процессор Intel Pentium Celeron 2400 Мгц и выше;
- оперативная память 4096 Мбайт и выше (рекомендуется 8-16 Гб и
- выше);
- видеокарта: Intel Iris Graphics XE;
- жёсткий диск: от 40 Гб;
- среда выполнения JavaScript: Node.js v22.14.0;
- фреймворк React;

2.5 Требования к информационной и программной совместимости

2.5.1 Требования к языкам программирования

Базовый язык программирования: JavaScript.

Необходимые библиотеки и расширения языка JavaScript:

- antd==5.25.1;
- axios== 1.9.0;
- dayjs==1.11.13;
- dotenv==16.5.0;
- jwt-decode==4.0.0;
- leaflet==1.9.4;
- leaflet-routing-machine==3.2.12;
- mobx==6.13.7;
- mobx-react-lite==4.1.0;
- react==19.1.0;
- react-input-mask==2.0.4;
- react-leaflet==5.0.0;
- react-router-dom=7.6.0;
- recharts==2.15.3;
- bcrypt==6.0.0;

- cors==2.8.5;
- express==5.1.0;
- jsonwebtoken==9.0.2;
- mysql==2.18.1;
- mysql2==3.14.1;
- nodemailer==7.0.3;
- sequelize==6.37.7.

2.5.2 Требования к программным средствам, используемым программой

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows, среды выполнения Node.js v22.14.0.

3 Проектирование информационной системы

Для проектирования данной информационной системы был выбран язык объектно-ориентированного подхода UML.

Унифицированный язык моделирования UML (Unified Modeling Language) – это язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. Данный язык одновременно является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения.

3.1 Диаграмма вариантов использования

Диаграммы прецедентов играют основную роль в моделировании поведения системы, подсистемы или класса. Каждая такая диаграмма показывает множество прецедентов, актёров и отношения между ними.

Диаграммы прецедентов применяются для моделирования вида системы с точки зрения прецедентов (или вариантов использования). Чаще всего это предполагает моделирование контекста системы, подсистемы или класса либо моделирование требований, предъявляемых к поведению указанных элементов.

Диаграммы прецедентов имеют большое значение для визуализации, специфицирования и документирования поведения элемента. Они облегчают понимание систем, подсистем или классов, представляя взгляд извне на то, как данные элементы могут быть использованы в соответствующем контексте.

Кроме того, такие диаграммы важны для тестирования исполняемых систем в процессе прямого проектирования и для понимания их внутреннего устройства при обратном проектировании.

На рисунке 3.1 представлена диаграмма прецедентов системы.

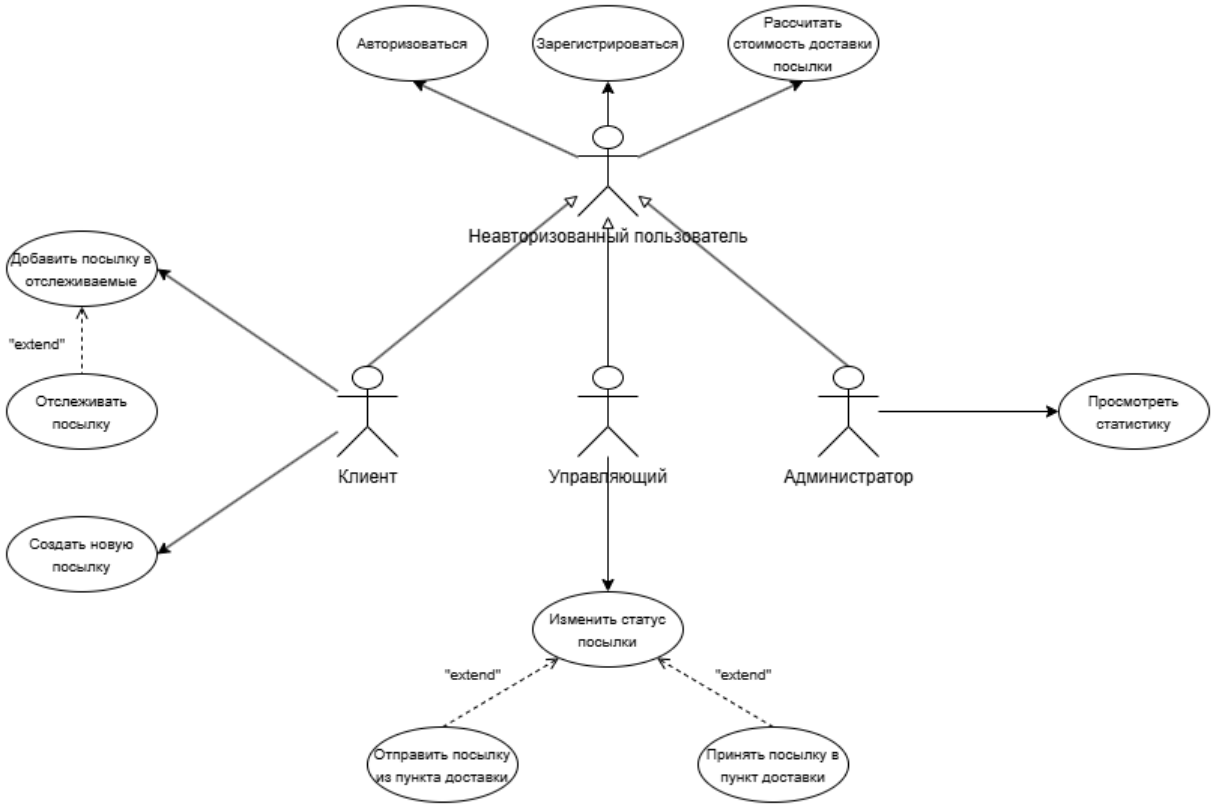

Рисунок 3.1 Диаграмма прецедентов

На диаграмме представлены все доступные роли и варианты использования. В системе доступны четыре роли: неавторизованный пользователь, клиент, управляющий и администратор, от каждой роли указаны связи к вариантам использования.

В таблицах 3.1 – 3.9 представлены описательные спецификации прецедентов, представленных на диаграмме.

Таблица 3.1

Описательная спецификация прецедента «Рассчитать стоимость доставки посылки»

Прецедент	Рассчитать стоимость доставки посылки
Краткое описание	Пользователь рассчитывает стоимость доставки посылки на странице калькулятора
Субъекты	Неавторизованный пользователь, клиент, управляющий, администратор
Основной поток	Ввод данных в поля формы расчёта стоимости доставки посылки
Предусловие	Открытие страницы калькулятора доставки
Постусловие	Вывод информации о стоимости доставки посылки

Таблица 3.2

Описательная спецификация прецедента «Авторизоваться»

Прецедент	Авторизоваться
Краткое описание	Пользователь авторизуется на сайте
Субъекты	Неавторизованный пользователь
Основной поток	Ввод данных в поля формы авторизации
Предусловие	Открытие страницы авторизации
Постусловие	Вывод информации о стоимости доставки посылки

Таблица 3.3

Описательная спецификация прецедента «Зарегистрироваться»

Прецедент	Зарегистрироваться
Краткое описание	Пользователь регистрируется в веб-приложении
Субъекты	Неавторизованный пользователь
Основной поток	Ввод данных в поля формы регистрации
Предусловие	Открытие страницы регистрации
Постусловие	Добавление пользователя в базу данных

Таблица 3.4

Описательная спецификация прецедента «Добавить посылку в отслеживаемые»

Прецедент	Добавить посылку в отслеживаемые
Краткое описание	Клиент добавляет посылку в отслеживаемые
Субъекты	Клиент
Основной поток	Ввод трек-кода посылки в поле
Предусловие	Открытие личного кабинета клиента
Постусловие	Добавление посылки в отслеживаемые пользователем в базе данных

Таблица 3.5

Описательная спецификация прецедента «Отслеживать посылку»

Прецедент	Отслеживать посылку
Краткое описание	Клиент отслеживает прогресс доставки посылки
Субъекты	Клиент
Основной поток	Выбор посылки из списка отслеживаемых
Предусловие	Открытие личного кабинета клиента
Постусловие	Вывод информации о прогрессе доставки посылки

Таблица 3.6

Описательная спецификация прецедента «Создать новую посылку»

Прецедент	Создать новую посылку
Краткое описание	Клиент создает новую посылку перед отправкой
Субъекты	Клиент
Основной поток	Ввод данных в поля формы создания новой посылки
Предусловие	Открытие страницы создания новой посылки
Постусловие	Добавление информации о посылке в базу данных

Таблица 3.7

Описательная спецификация прецедента «Отправить посылку из пункта доставки»

Прецедент	Отправить посылку из пункта доставки
Краткое описание	Управляющий подтверждает прибытие посылки в пункт доставки
Субъекты	Управляющий
Основной поток	Выбор посылки из списка ожидаемых в пункте доставки
Предусловие	Открытие личного кабинета управляющего
Постусловие	Обновление прогресса доставки посылки в базе данных

Таблица 3.8

Описательная спецификация прецедента «Принять посылку в пункт доставки»

Прецедент	Принять посылку в пункт доставки
Краткое описание	Управляющий подтверждает отбытие посылки из пункта доставки
Субъекты	Управляющий
Основной поток	Выбор посылки из списка находящихся в пункте доставки
Предусловие	Открытие личного кабинета управляющего
Постусловие	Обновление прогресса доставки посылки в базе данных

Таблица 3.9

Описательная спецификация прецедента «Просмотреть статистику»

Прецедент	Просмотреть статистику
Краткое описание	Администратор просматривает графики статистики работы системы
Субъекты	Администратор
Основной поток	Вывод графиков статистики работы системы
Предусловие	Открытие личного кабинета администратора
Постусловие	Вывод графиков статистики работы системы

3.2 Информационно-логическая модель базы данных

Информационно-логическая модель отображает данные предметной области в виде совокупности информационных объектов и связей между ними. Информационно-логическая модель базы данных представлена на рисунке 3.2.

Для управления базой данных выбрана система управления базой данных (СУБД) MySQL, так как данная СУБД выделяется простотой использования и хорошей производительностью.

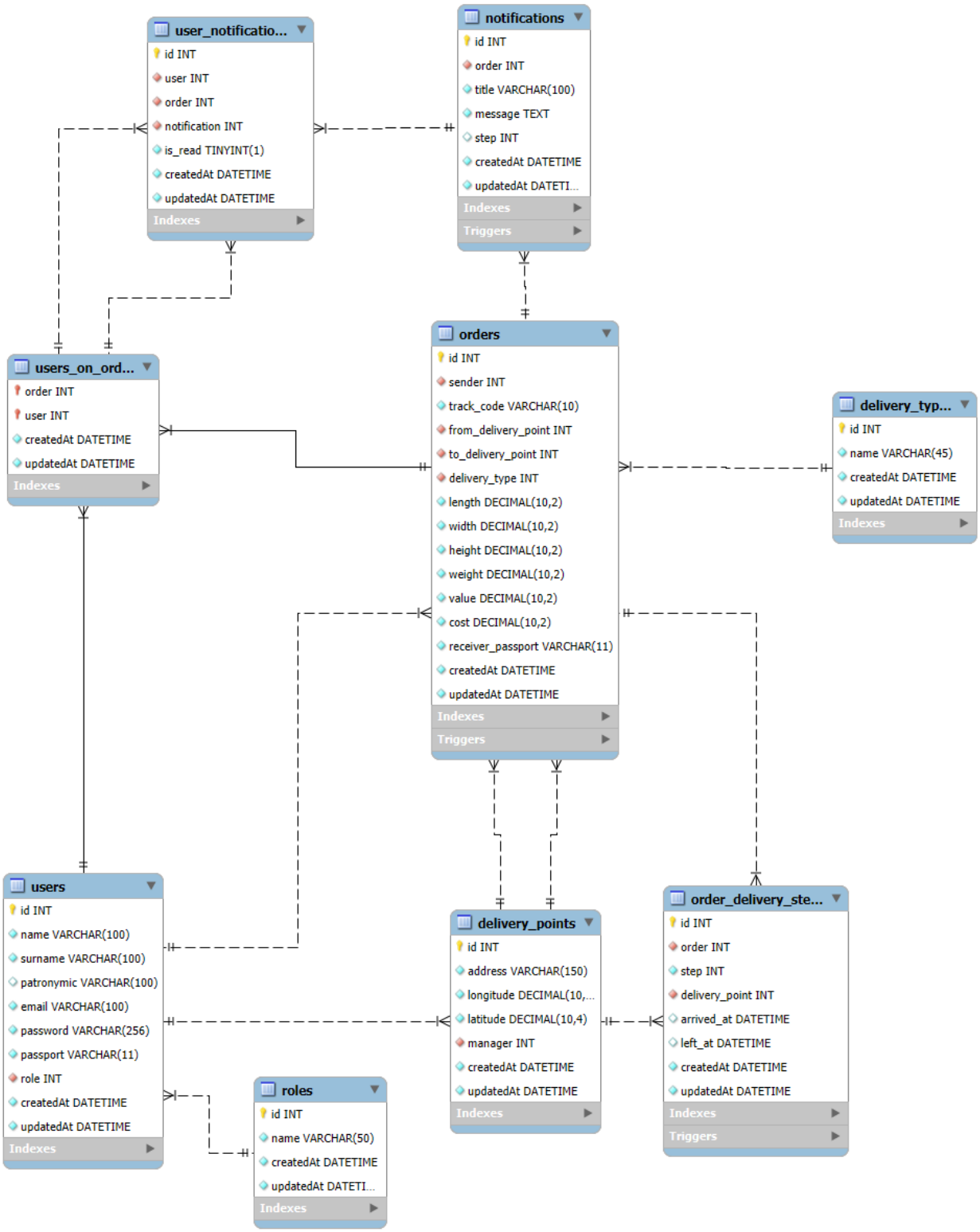

Рисунок 3.2 Информационно-логическая модель базы данных

В таблицах 3.10 – 3.18 представлены спецификации на таблицы базы данных.

Таблица 3.10

Описательная спецификация таблицы «Пользователи» (users)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор пользователя	PK	id	INT
Имя	-	name	VARCHAR(100)
Фамилия	-	surname	VARCHAR(100)
Отчество	-	patronymic	VARCHAR(100)
Электронная почта	-	email	VARCHAR(100)
Пароль	-	password	VARCHAR(256)
Серия и номер паспорта	-	passport	VARCHAR(11)
Роль	FK	role	INT
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.11

Описательная спецификация таблицы «Роли» (roles)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор роли	PK	id	INT
Наименование	-	name	VARCHAR(50)
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.12

Описательная спецификация таблицы «Пункты доставки» (delivery_points)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор пункта доставки	PK	id	INT
Адрес	-	name	VARCHAR(150)
Долгота	-	longitude	DECIMAL(10, 4)
Ширина	-	latitude	DECIMAL(10, 4)
Управляющий	FK	manager	INT
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.13

Описательная спецификация таблицы «Шаги доставки» (order_delivery_steps)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор шага доставки	PK	id	INT
Посылка	FK	order	INT
Порядковый номер шага	-		INT
Пункт доставки	FK	delivery_point	INT
Дата и время прибытия в пункт доставки	-	arrived_at	DATETIME
Дата и время отбытия из пункта доставки	-	left_at	DATETIME
Дата и время	-	createdAt	DATETIME

создания записи			
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.14

Описательная спецификация таблицы «Посылки, отслеживаемые пользователем» (users_on_order)

Наименование	Ключи	Обозначение в БД	Тип данных
Посылка	PK, FK	order	INT
Пользователь	PK, FK	user	INT
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.15

Описательная спецификация таблицы «Посылки» (orders)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор посылки	PK	id	INT
Отправитель	FK	sender	INT
Трек-код	-	track_code	VARCHAR(10)
Пункт отправки	FK	from_delivery_point	INT
Пункт назначения	FK	to_delivery_point	INT
Тип доставки	FK	delivery_type	INT
Длина	-	length	DECIMAL(10, 2)
Ширина	FK	width	DECIMAL(10, 2)
Высота	-	height	DECIMAL(10, 2)

Вес	-	weight	DECIMAL(10, 2)
Оценочная стоимость посылки	-	value	DECIMAL(10, 2)
Стоимость доставки	-	cost	DECIMAL(10, 2)
Серия и номер паспорта получателя	-	receiver_passport	VARCHAR(11)
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.16

Описательная спецификация таблицы «Виды доставки» (delivery_types)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор типа доставки	PK	id	INT
Наименование	-	name	VARCHAR(45)
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.17

Описательная спецификация таблицы «Уведомления пользователей» (user_notifications)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор уведомления пользователя	PK	id	INT
Пользователь	FK	user	INT
Посылка	FK	order	INT
Уведомление	FK	notification	INT
Статус прочтитанности	-	is_read	TINYINT(1)
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

Таблица 3.18

Описательная спецификация таблицы «Уведомления» (notifications)

Наименование	Ключи	Обозначение в БД	Тип данных
Идентификатор уведомления	PK	id	INT
Посылка	FK	order	INT
Текст заголовка	-	title	VARCHAR(100)
Текст содержания	-	message	TEXT
Порядковый номер шага	-	step	INT
Дата и время создания записи	-	createdAt	DATETIME
Дата и время последнего обновления записи	-	updatedAt	DATETIME

3.3 Диаграмма последовательности

На диаграммах последовательности показывают связи, включающие множество объектов и отношений между ними, в том числе сообщения, которыми объекты обмениваются. При этом диаграмма последовательностей акцентирует внимание на временной упорядоченности сообщений, а диаграмма кооперации – на структурной организации посылающих и принимающих сообщения объектов.

Диаграммы последовательности используются для моделирования динамических аспектов системы. Сюда входит моделирование конкретных и прототипических экземпляров классов, интерфейсов, компонентов и узлов, а также сообщений, которыми они обмениваются, – и всё это в контексте сценария, иллюстрирующего данное поведение.

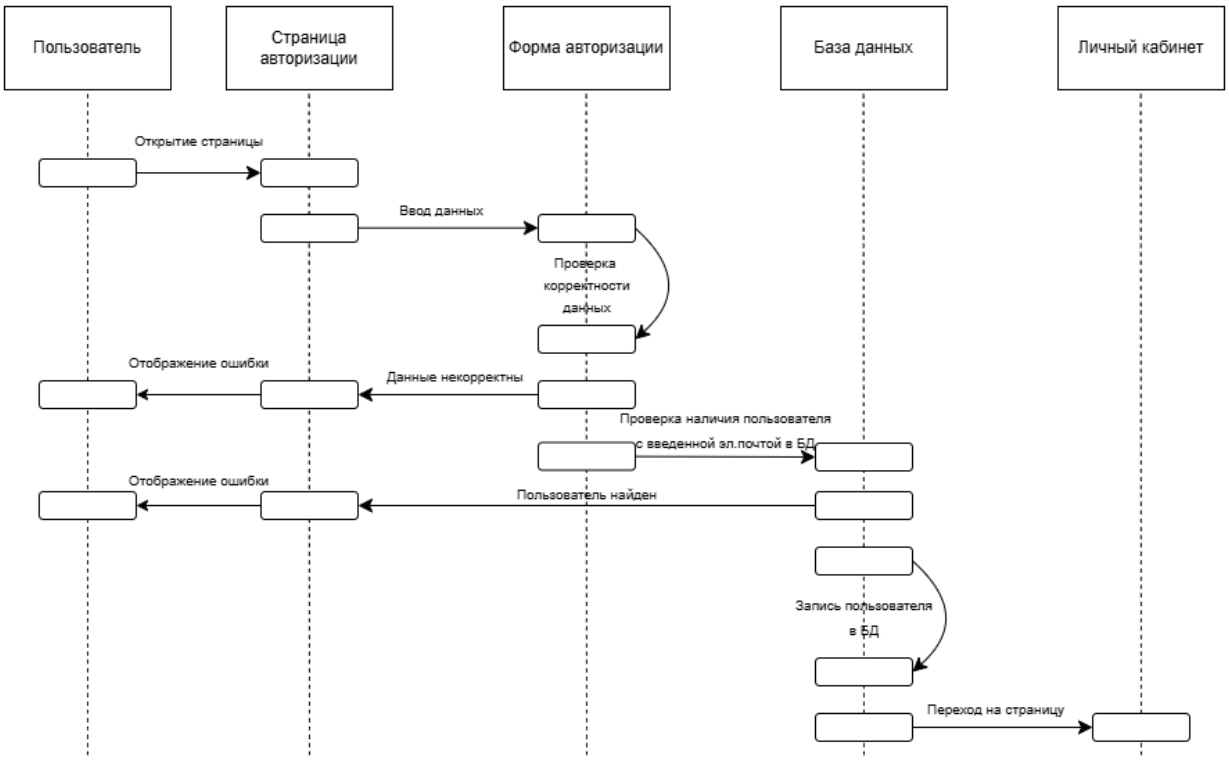

Рисунок 3.3.1 Диаграмма последовательности «Авторизация»

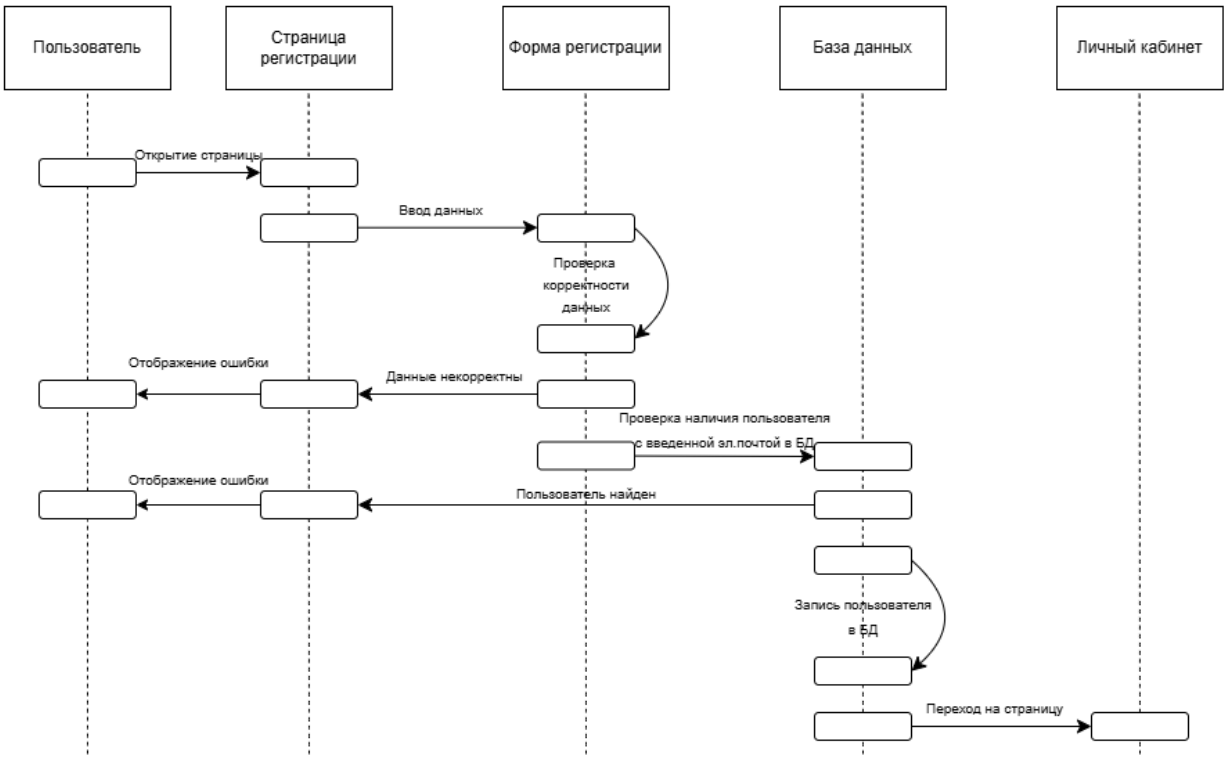

Рисунок 3.3.2 Диаграмма последовательности «Регистрация»

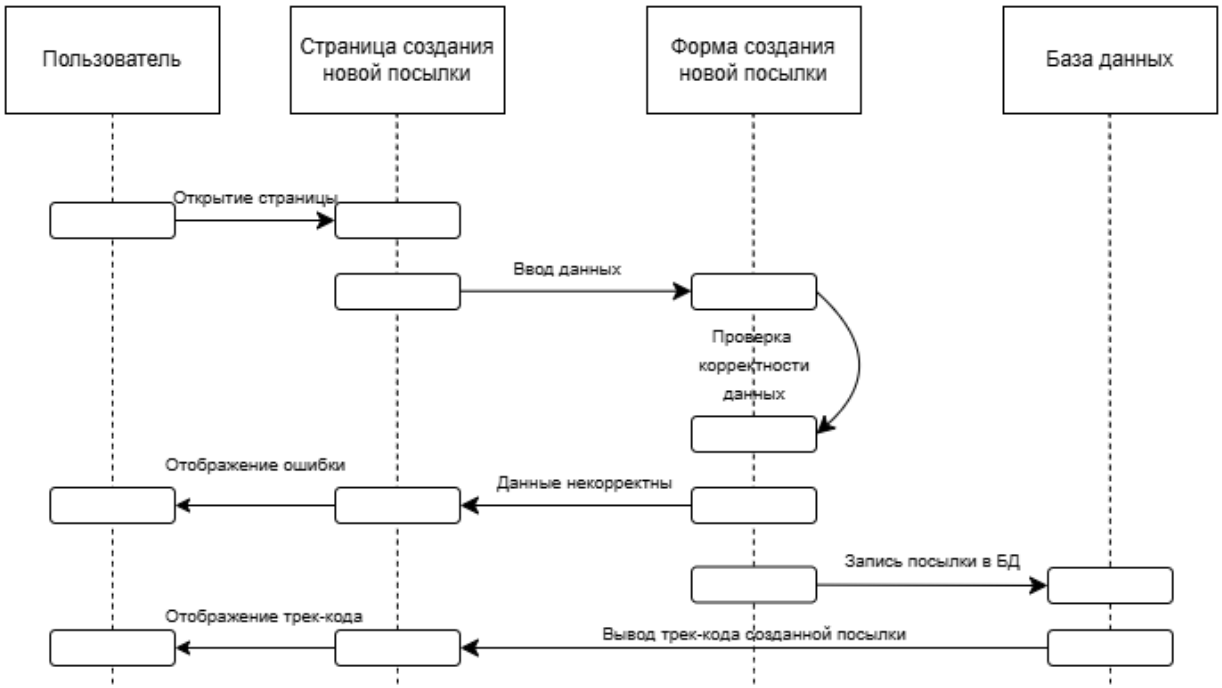

Рисунок 3.3.3 Диаграмма последовательности «Создание новой посылки»

3.4 Диаграмма состояний

Диаграмма состояний показывает автомат. Её частной разновидностью является диаграмма деятельности, в которой все или большая часть состояний – это состояния деятельности, а все или большая часть переходов инициируются в результате завершения деятельности в исходном состоянии. Таким образом, при моделировании жизненного цикла объекта полезны как диаграммы деятельности, так и диаграммы состояний. Но если диаграмма деятельности показывает поток управления от деятельности к деятельности, то на диаграмме состояний представлен поток управления от состояния к состоянию.

Диаграммы состояний используются для моделирования динамических аспектов системы. По большей части под этим подразумевается моделирование поведения реактивных объектов. Реактивным называется объект, поведение которого лучше всего характеризуется его реакцией на события, произошедшие вне его собственного контекста. У реактивного объекта есть чётко выраженный жизненный цикл, когда текущее поведение обусловлено прошлым. Диаграммы состояний можно присоединять к классам, прецедентам или системе в целом для визуализации, специфицирования, конструирования и документирования динамики отдельного объекта.

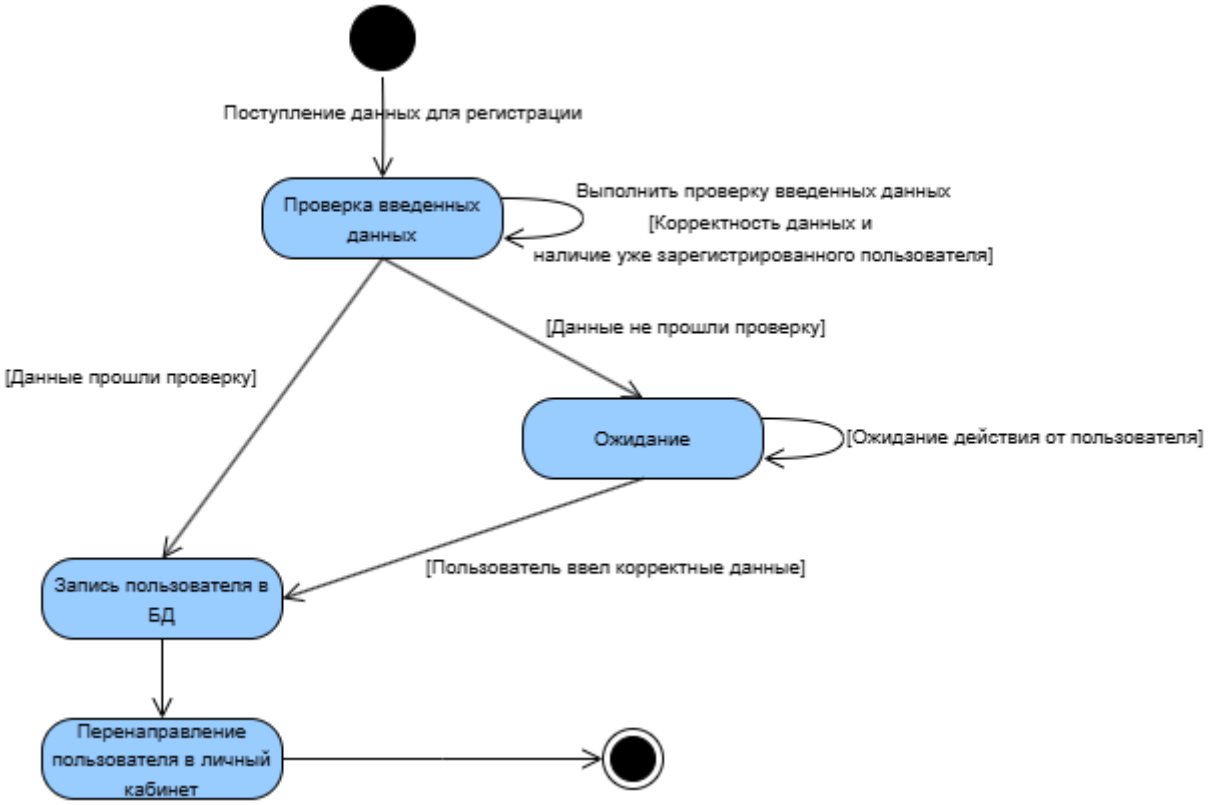

Рисунок 3.4 Диаграмма состояний посылки.

3.5 Диаграмма компонентов

Диаграмма компонентов (англ. Component diagram) — элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами.

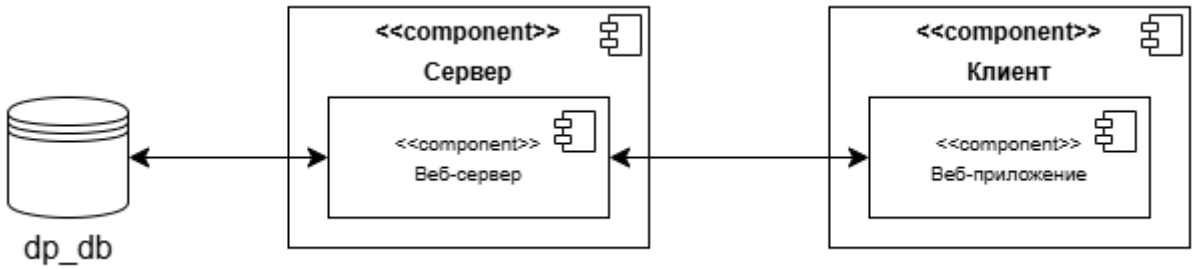

Рисунок 3.5 Диаграмма компонентов

На диаграмме компонентов, которая представлена на рисунке 3.5, изображены следующие аппаратно-программные компоненты:

- сервер базы данных, включающий в себя базу данных;
- сервер, включающий в себя веб-сервер;

- клиент, включающий в себя веб-приложение и браузер.

3.6 Диаграмма размещения

На диаграмме размещения, представленной на рисунке 3.6, находятся узлы выполнения программных компонентов реального времени, а также процессов и объектов. Диаграмма размещения показывает, как компоненты системы физически размещены на вычислительных ресурсах во время выполнения. На сервере установлено следующее программное обеспечение: операционная система – Windows или семейства Unix, база данных, среда выполнения JavaScript Node.js. Веб-клиент представлен одним из известных браузеров. На устройстве клиента также может быть любая операционная система.

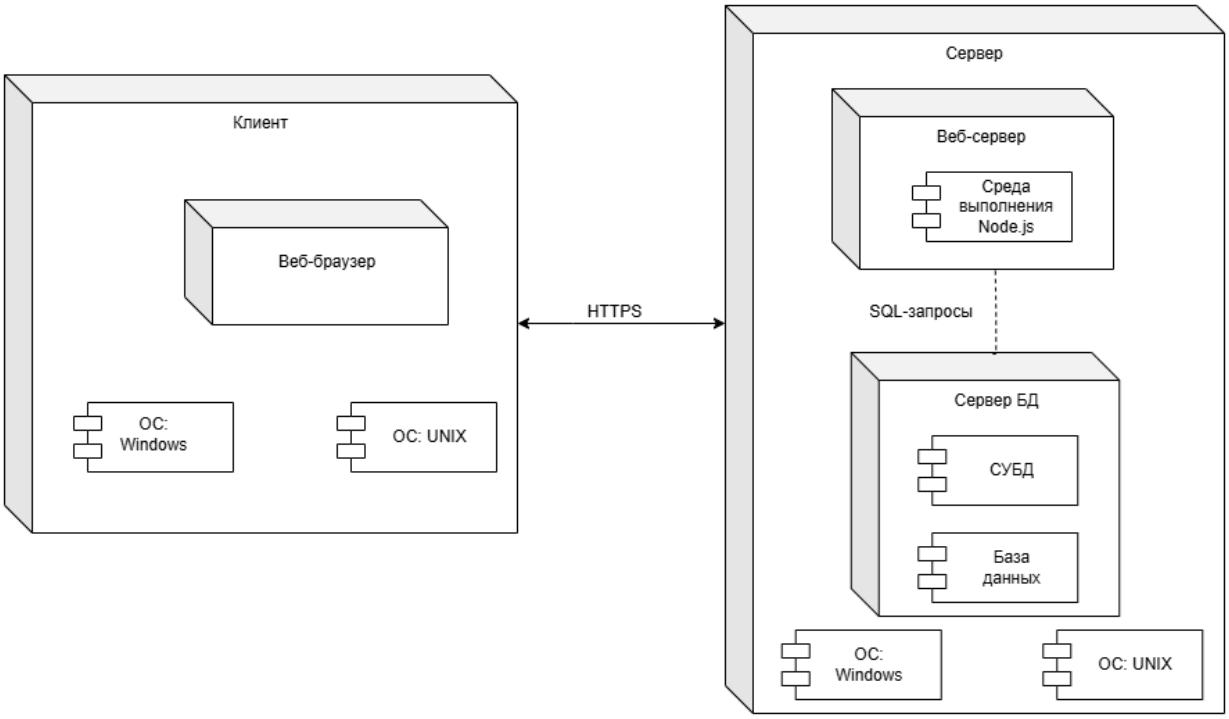

Рисунок 3.6 Диаграмма размещения

4 Разработка информационной системы

4.1 Описание структуры

Таблица 4.1

Файловая структура проекта

Diplom/	
├── client/	Папка клиентской части
│ ├── src/	
│ │ ├── components/	Папка React-компонентов
│ │ │ ├── AppRouter.jsx	Компонент роутера
│ │ │ ├── Sidebar.jsx	Компонент боковой панели
│ │ │ ├── ProfileMap.jsx	Компонент карты
│ │ ├── http/	Файлы с функциями для запросов к серверу
│ │ │ ├── usersAPI.js	Запросы, связанные с пользователями
│ │ │ ├── ordersAPI.js	Запросы, связанные с посылками
│ │ │ ├── deliveryPointsAPI.js	Запросы, связанные с посылками
│ │ │ ├── deliveryTypesAPI.js	Запросы, связанные с посылками
│ │ │ ├── notificationsAPI.js	Запросы, связанные с посылками
│ │ │ ├── orderDeliveryStepsAPI.js	Запросы, связанные с посылками
│ │ │ ├── index.js	Объединение API для удобного импорта
│ │ ├── pages/	Папка компонентов-страниц
│ │ │ ├── Auth.jsx	Страница авторизации
│ │ │ ├── Reg.jsx	Страница регистрации
│ │ │ ├── Profile.jsx	Страница личного кабинета клиента
│ │ │ ├── Calculator.jsx	Страница калькулятора стоимости доставки
│ │ │ ├── CreateOrder.jsx	Страница создания нового заказа
│ │ │ ├── Manager.jsx	Страница личного кабинета управляющего
│ │ │ ├── Admin.jsx	Страница личного кабинета администратора
│ │ ├── store/	Папка с файлами-состояний
│ │ │ ├── UserStore.js	Состояние пользователя
│ │ ├── utils/	Утилиты и константы для всего приложения

│ │ │ ├── consts.js	Константы
│ │ │ ├── routes.js	Описание маршрутов
│ │ ├── App.jsx	Корневой React-компонент приложения
│ ├── .env	Конфигурационные переменные окружения
│ ├── index.html	Корневой HTML-шаблон для приложения
│ ├── package.json	Зависимости и скрипты приложения
│ ├── vite-config.js	Конфигурация сборщика Vite
├── server/	Папка серверной части
│ ├── controllers/	Папка контроллеров
│ │ ├── deliveryPointsController.js	Контроллер пунктов доставки
│ │ ├── deliveryTypesController.js	Контроллер типов доставки
│ │ ├── orderDeliveryStepsController.js	Контроллер шагов доставки
│ │ ├── ordersController.js	Контроллер посылок
│ │ ├── userNotificationsController.js	Контроллер уведомлений
│ │ ├── usersController.js	Контроллер пользователей
│ │ ├── usersOnOrderController.js	Контроллер посылок пользователя
│ ├── error/	
│ │ ├── ApiError.js	Класс обработки ошибок
│ ├── middleware/	Папка с промежуточными функциями
│ │ ├── authMiddleware.js	Проверка авторизованности
│ │ ├── ErrorHandlingMiddleware.js	Централизованная обработка всех ошибок
│ │ ├── roleMiddleware.js	Проверка прав доступа по ролям
│ ├── models/	Папка с моделями Sequelize
│ │ ├── data.js	Предустановленные данные
│ │ ├── models.js	Основные модели
│ ├── routes/	Папка с маршрутами
│ │ ├── deliveryPointsRouter.js	Маршруты пунктов доставки
│ │ ├── deliveryTypesRouter.js	Маршруты типов доставки
│ │ ├── index.js	Объединение всех маршрутов
│ │ ├── orderDeliveryStepsRouter.js	Маршруты шагов доставки
│ │ ├── ordersRouter.js	Маршруты посылок
│ │ ├── userNotificationsRouter.js	Маршруты уведомлений
│ │ ├── usersOnOrderRouter.js	Маршруты посылок пользователя
│ │ ├── usersRouter.js	Маршруты пользователей
│ ├── utils/	Вспомогательные утилиты

│ │ └── routeBuilder.js	Логика построения маршрута для доставки
│ ├── .env	Переменные окружения
│ ├── db.js	Подключение к базе данных
│ ├── index.js	Точка входа
│ ├── package.json	Скрипты и зависимости

4.2 Разработка серверной части

Серверная часть разработана на Node.js с использованием JavaScript и реализует бизнес-логику веб-приложения. Для работы с базой данных используется Sequelize ORM и MySQL, что позволяет удобно управлять данными и связями между сущностями.

Основные функции сервера:

- Обработка API-запросов от клиентской части (создание заказов, отслеживание, работа с пунктами доставки, предоставление статистики по работе сервиса).
- Управление пользователями по ролям: клиент, управляющий, администратор.
- Генерация маршрута доставки при создании посылки;
- Отправка сообщения на почту при прибытии посылки в пункт назначения.

Структура сервера включает модули маршрутов, контроллеров, моделей и middleware для авторизации и обработки ошибок. Все запросы защищены через JWT и проверку ролей пользователей.

4.2.1 Разработка главной точки входа серверной части приложения.

Файл index.js инициализирует сервер Express, подключает маршруты, middleware и базу данных, а также выполняет первичную настройку проекта, включая создание триггеров и загрузку начальных данных в БД.

Основные функции и логика:

1. Импорт зависимостей и конфигурации:
   - dotenv, express, cors, bcrypt, sequelize — стандартные модули для запуска сервера, работы с БД и обработки запросов.
   - Импортируются маршруты (routes/index) и обработчик ошибок (ErrorHandlingMiddleware).
2. Инициализация Express-сервера:
   - Устанавливаются middleware (cors, express.json()).
   - Все API-запросы направляются через /api к маршрутизатору.
3. Запуск сервера (start()):
   - Подключение к БД и синхронизация моделей через Sequelize.
   - Загрузка начальных данных в таблицы (роли, пользователи, типы и

пункты доставки).

- Создание SQL-триггеров:
  - orders_AFTER_INSERT — при создании заказа автоматически связывает его с отправителем.
  - order_delivery_steps_AFTER_UPDATE — генерирует уведомления при изменении статуса шага доставки.
  - notifications_AFTER_INSERT — автоматически рассылает уведомление всем пользователям, отслеживающим заказ.

4. Запуск приложения на заданном порту.

Код:

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const router = require('./routes/index');
const errorHandler = require('./middleware/ErrorHandlingMiddleware');
const sequelize = require('./db');
const {
    DeliveryTypes, DeliveryPoints, Users, Roles, Orders,
    UsersOnOrder, OrderDeliverySteps
} = require("./models/models");
const {
    deliveryTypesData, deliveryPointsData, rolesData,
    usersData, ordersData, usersOnOrderData, orderDeliveryStepsData
} = require("./models/data");
const bcrypt = require("bcrypt");

const PORT = process.env.SERVER_PORT || 8080;

const app = express();
app.use(cors());
app.use(express.json());
app.use('/api', router);

app.use(errorHandler);

const start = async () => {
    try {
        await sequelize.authenticate();
        // Пересоздаёт все таблицы каждый раз (всё стирается)
        await sequelize.sync({ force: true });

        // Начальные данные
        await Roles.bulkCreate(rolesData);
        await Users.bulkCreate(usersData);
        await DeliveryTypes.bulkCreate(deliveryTypesData);
        await DeliveryPoints.bulkCreate(deliveryPointsData);

        // Триггер: после создания заказа — связать с отправителем
        await sequelize.query(
            `DROP TRIGGER IF EXISTS \`${process.env.DB_NAME}\`.\`orders_AFTER_INSERT\`;`
        );
        await sequelize.query(
            `CREATE TRIGGER orders_AFTER_INSERT
                AFTER INSERT ON orders
```

```
            FOR EACH ROW
            BEGIN
              INSERT INTO dp_db.users_on_order (\`order\`, \`user\`, createdAt, updatedAt)
              VALUES (NEW.id, NEW.sender, NOW(), NOW());
            END;`
        );

        // Триггер: при обновлении шага доставки — создать уведомление
        await sequelize.query(
          `DROP TRIGGER IF EXISTS \`${process.env.DB_NAME}\`.\`order_delivery_steps_AFTER_UPDATE\`;`
        );
        await sequelize.query(
          `CREATE TRIGGER \`order_delivery_steps_AFTER_UPDATE\` AFTER UPDATE ON
\`order_delivery_steps\`
            FOR EACH ROW
            BEGIN
              -- Если поменялось время прибытия
              IF NOT (OLD.arrived_at <=> NEW.arrived_at) THEN

                -- Первый шаг
                IF NEW.step = 1 THEN
                  INSERT INTO \`${process.env.DB_NAME}\`.\`notifications\`
                  (\`order\`, \`title\`, \`message\`, \`step\`, \`createdAt\`, \`updatedAt\`)
                  VALUES (NEW.order, 'Заказ в точке отправления',
                  'Заказ принят в пункте отправления, ожидайте дальнейших обновлений!',
                  NEW.step, now(), now());

                -- Последний шаг
                ELSEIF NEW.step = (
                  SELECT MAX(step) FROM \`order_delivery_steps\` WHERE \`order\` = NEW.order
                ) THEN
                  INSERT INTO \`${process.env.DB_NAME}\`.\`notifications\`
                  VALUES (NEW.order, 'Заказ доставлен!',
                  'Заказ ожидает выдачи в пункте назначения, возьмите с собой паспорт!',
                  NEW.step, now(), now());

                -- Промежуточный шаг
                ELSE
                  INSERT INTO \`${process.env.DB_NAME}\`.\`notifications\`
                  VALUES (NEW.order, 'Изменение статуса заказа!',
                  'Заказ принят в новом пункте доставки, ожидайте дальнейших обновлений!',
                  NEW.step, now(), now());
                END IF;

              -- Если поменялось время отбытия
              ELSEIF NOT (OLD.left_at <=> NEW.left_at) THEN

                -- Не последний шаг
                IF NEW.step <> (
                  SELECT MAX(step) FROM \`order_delivery_steps\` WHERE \`order\` = NEW.order
                ) THEN
                  INSERT INTO \`${process.env.DB_NAME}\`.\`notifications\`
                  VALUES (NEW.order, 'Изменение статуса заказа!',
                  'Заказ покинул пункт доставки, ожидайте дальнейших обновлений!',
                  NEW.step, now(), now());

                -- Последний шаг — заказ забрали
                ELSE
                  INSERT INTO \`${process.env.DB_NAME}\`.\`notifications\`
```

```
                VALUES (NEW.order, 'Заказ забрали!',
                'Заказ забрали, доставка считается завершенной!',
                NEW.step, now(), now());
              END IF;
            END IF;
          END`
        );

        // Триггер: при создании уведомления — разослать пользователям
        await sequelize.query(
          `DROP TRIGGER IF EXISTS \`${process.env.DB_NAME}\`.\`notifications_AFTER_INSERT\`;`
        );
        await sequelize.query(
          `CREATE TRIGGER \`notifications_AFTER_INSERT\` AFTER INSERT ON \`notifications\`
          FOR EACH ROW
          BEGIN
            INSERT INTO \`${process.env.DB_NAME}\`.\`user_notifications\`
            (\`user\`, \`order\`, \`notification\`, \`createdAt\`, \`updatedAt\`)
            SELECT \`users_on_order\`.\`user\`, \`users_on_order\`.\`order\`, NEW.id, now(), now()
            FROM \`users_on_order\`
            WHERE \`users_on_order\`.\`order\` = NEW.\`order\`;
          END`
        );

        app.listen(PORT, () => console.log(`Сервер запущен на порту: ${PORT}`));
    } catch (e) {
        console.log(e);
    }
}

start();
```

4.2.2 Разработка файла конфигурации подключения к БД

Файл создаёт и экспортирует экземпляр Sequelize — ORM-библиотеки для работы с базой данных. Он используется для подключения серверной части приложения к СУБД MySQL.

Код:

```
const {Sequelize} = require('sequelize');

module.exports = new Sequelize(
    process.env.DB_NAME,
    process.env.DB_USER,
    process.env.DB_PASSWORD,
    {
        dialect: 'mysql',
        host: process.env.DB_HOST,
        port: process.env.DB_PORT,
        timezone: '+03:00'
    }
)
```

4.2.3 Разработка файла описания моделей БД

Файл служит для определения всех таблиц базы данных как моделей Sequelize и описания их связей друг с другом. Он работает в связке с db.js,

используя созданное там подключение к базе данных. Все модели экспортируются для использования в других частях приложения (например, в контроллерах или сервисах).

Код:

```
const sequelize = require('../db');
const { DataTypes } = require('sequelize');

// === МОДЕЛИ ===

const Roles = sequelize.define('roles', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    name: { type: DataTypes.STRING(50), allowNull: false },
}, { tableName: 'roles' });

const Users = sequelize.define('users', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    name: { type: DataTypes.STRING(100), allowNull: false },
    surname: { type: DataTypes.STRING(100), allowNull: false },
    patronymic: { type: DataTypes.STRING(100), allowNull: true },
    email: { type: DataTypes.STRING(100), allowNull: false },
    password: { type: DataTypes.STRING(256), allowNull: false },
    passport: { type: DataTypes.STRING(11), allowNull: false },
    role: { type: DataTypes.INTEGER, allowNull: false, defaultValue: 3 }
}, { tableName: 'users' });

const DeliveryPoints = sequelize.define('delivery_points', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    address: { type: DataTypes.STRING(150), allowNull: false },
    longitude: { type: DataTypes.DECIMAL(10, 4), allowNull: false },
    latitude: { type: DataTypes.DECIMAL(10, 4), allowNull: false },
    manager: { type: DataTypes.INTEGER, allowNull: false }
}, { tableName: 'delivery_points' });

const DeliveryTypes = sequelize.define('delivery_types', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    name: { type: DataTypes.STRING(45), allowNull: false }
}, { tableName: 'delivery_types' });

const Orders = sequelize.define('orders', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    sender: { type: DataTypes.INTEGER, allowNull: false },
    track_code: { type: DataTypes.STRING(10), allowNull: false, unique: true },
    from_delivery_point: { type: DataTypes.INTEGER, allowNull: false},
    to_delivery_point: { type: DataTypes.INTEGER, allowNull: false },
    delivery_type: { type: DataTypes.INTEGER, allowNull: false },
    length: { type: DataTypes.DECIMAL(10, 2), allowNull: false },
    width: { type: DataTypes.DECIMAL(10, 2), allowNull: false },
    height: { type: DataTypes.DECIMAL(10, 2), allowNull: false },
    weight: { type: DataTypes.DECIMAL(10, 2), allowNull: false },
    value: { type: DataTypes.DECIMAL(10, 2), allowNull: false },
    cost: { type: DataTypes.DECIMAL(10, 2), allowNull: false },
    receiver_passport: { type: DataTypes.STRING(11), allowNull: false },
}, { tableName: 'orders', timestamps: true });

const Notifications = sequelize.define('notifications', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
```

```
    order: { type: DataTypes.INTEGER, allowNull: false },
    title: { type: DataTypes.STRING(100), allowNull: false },
    message: { type: DataTypes.TEXT, allowNull: false },
    step: { type: DataTypes.INTEGER, allowNull: true },
}, { tableName: 'notifications' });

const OrderDeliverySteps = sequelize.define('order_delivery_steps', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    order: { type: DataTypes.INTEGER, allowNull: false },
    step: { type: DataTypes.INTEGER, allowNull: false },
    delivery_point: { type: DataTypes.INTEGER, allowNull: false },
    arrived_at: { type: DataTypes.DATE, allowNull: true },
    left_at: { type: DataTypes.DATE, allowNull: true }
}, { tableName: 'order_delivery_steps' });

const UsersOnOrder = sequelize.define('users_on_order', {
    order: { type: DataTypes.INTEGER, primaryKey: true },
    user: { type: DataTypes.INTEGER, primaryKey: true },
}, { tableName: 'users_on_order' });

const UserNotifications = sequelize.define('user_notifications', {
    id: { type: DataTypes.INTEGER, autoIncrement: true, primaryKey: true },
    user: { type: DataTypes.INTEGER, references: { model: 'users_on_order', key: 'user' } },
    order: { type: DataTypes.INTEGER, references: { model: 'users_on_order', key: 'order' } },
    notification: { type: DataTypes.INTEGER, primaryKey: true },
    is_read: { type: DataTypes.BOOLEAN, allowNull: false, defaultValue: false }
}, { tableName: 'user_notifications' });


// === СВЯЗИ ===

Roles.hasMany(Users, {foreignKey: 'role'});
Users.belongsTo(Roles, {foreignKey: 'role', targetKey: 'id', as: 'roleData'});

Users.hasOne(DeliveryPoints, {foreignKey: 'manager'});
DeliveryPoints.belongsTo(Users, {foreignKey: 'manager', targetKey: 'id', as: 'managerData'});

Users.hasMany(Orders, { foreignKey: 'sender' });
Orders.belongsTo(Users, { foreignKey: 'sender', targetKey: 'id', as: 'senderData' });

Users.hasMany(UsersOnOrder, { foreignKey: 'user' });
UsersOnOrder.belongsTo(Users, { foreignKey: 'user', targetKey: 'id', as: 'userData' });

DeliveryTypes.hasMany(Orders, { foreignKey: 'delivery_type' });
Orders.belongsTo(DeliveryTypes, { foreignKey: 'delivery_type', targetKey: 'id', as: 'deliveryType' });

DeliveryPoints.hasMany(Orders, { foreignKey: 'from_delivery_point', as: 'fromOrders' });
Orders.belongsTo(DeliveryPoints, { foreignKey: 'from_delivery_point', targetKey: 'id', as: 'fromPoint' });

DeliveryPoints.hasMany(Orders, { foreignKey: 'to_delivery_point', as: 'toOrders' });
Orders.belongsTo(DeliveryPoints, { foreignKey: 'to_delivery_point', targetKey: 'id', as: 'toPoint' });

Orders.hasMany(UsersOnOrder, { foreignKey: 'order' });
UsersOnOrder.belongsTo(Orders, { foreignKey: 'order', targetKey: 'id', as: 'orderData' });

Orders.hasMany(OrderDeliverySteps, { foreignKey: 'order' });
OrderDeliverySteps.belongsTo(Orders, { foreignKey: 'order', targetKey: 'id', as: 'orderData' });

DeliveryPoints.hasMany(OrderDeliverySteps, { foreignKey: 'delivery_point' });
```

```
OrderDeliverySteps.belongsTo(DeliveryPoints, { foreignKey: 'delivery_point', targetKey: 'id', as: 'point' });

Orders.hasMany(Notifications, { foreignKey: 'order' });
Notifications.belongsTo(Orders, { foreignKey: 'order', targetKey: 'id', as: 'orderData' });

Notifications.hasMany(UserNotifications, { foreignKey: 'notification' });
UserNotifications.belongsTo(Notifications, { foreignKey: 'notification', targetKey: 'id', as: 'notificationData' });

module.exports = {
    Roles,
    Users,
    DeliveryPoints,
    DeliveryTypes,
    Orders,
    Notifications,
    OrderDeliverySteps,
    UsersOnOrder,
    UserNotifications
}
```

4.2.4 Разработка файлов промежуточных обработчиков

Промежуточные обработчики (Middleware) - это функции, которые выполняются между получением запроса и отправкой ответа в серверных приложениях.

AuthMiddleware — промежуточный обработчик, который защищает маршруты, требующие авторизации. Проверяет наличие и валидность JWT-токена в заголовке Authorization.

Код:

```
const jwt = require("jsonwebtoken");
module.exports = function(req, res, next) {

    if (req.method === "OPTIONS") {
        next();
    }

    try {
        const token = req.headers.authorization.split(' ')[1];
        if (!token) {
            return res.status(401).json({message: "Не авторизован"});
        }
        const decodedToken = jwt.verify(token, process.env.JWT_SECRET_KEY);
        req.user = decodedToken;
        next();
    } catch (e) {
        res.status(401).json({message: "Не авторизован"});
    }

}
```

roleMiddleware — промежуточный обработчик, который проверяет роль пользователя, основываясь на JWT-токене, и ограничивает доступ к

определённым маршрутам в зависимости от роли.

Код:

```
const jwt = require("jsonwebtoken");
module.exports = function(role) {
   return function(req, res, next) {
      if (req.method === "OPTIONS") {
         next();
      }

      try {
         const token = req.headers.authorization.split(' ')[1];
         if (!token) {
            return res.status(401).json({message: "Не авторизован"});
         }
         const decodedToken = jwt.verify(token, process.env.JWT_SECRET_KEY);
         if (decodedToken.role > role) {
            return res.status(403).json({message: "Нет доступа"});
         }
         req.user = decodedToken;
         next();
      } catch (e) {
         res.status(401).json({message: "Не авторизован"});
      }
   }
}
```

ErrorHandlingMiddleware – промежуточный обработчик ошибок, которые возникают в процессе обработки запросов.

Код:

```
const ApiError = require('../error/ApiError');

module.exports = function (error, req, res, next) {
   if (error instanceof ApiError) {
      return res.status(error.status).json({message: error.message});
   }
   return res.status(500).json({message: error.message});
}
```

4.2.5 Разработка класса обработчика ошибок

Класс ApiError расширяет стандартный класс Error и предназначен для создания ошибок с HTTP-статусом и сообщением, которые удобно использовать в приложении для обработки ошибок и формирования ответов клиенту.

Код:

```
class ApiError extends Error {
   constructor(status, message) {
      super();
      this.status = status;
      this.message = message;
```

```
    }

    static badRequest(message) {
        return new ApiError(404, message);
    }

    static internal(message) {
        return new ApiError(500, message);
    }

    static forbidden(message) {
        return new ApiError(403, message);
    }
}

module.exports = ApiError;
```

4.2.6 Разработка файлов маршрутизации

Роутер (Router) — это компонент, который отвечает за обработку HTTP-запросов, направленных на определённые URL-пути (маршруты) веб-приложения. Он связывает адрес (URL) и HTTP-метод (GET, POST, PUT, DELETE и т.д.) с определённой функцией или обработчиком, который выполнит нужные действия.

index.js – файл организующий маршруты приложения, собирая в одном месте подключение всех основных роутеров и экспортируя их как единый маршрутизатор для использования в основном серверном файле.

Код:

```
const Router = require("express");
const router = Router();

const usersRouter = require("./usersRouter");
const ordersRouter = require("./ordersRouter");
const deliveryPointsRouter = require("./deliveryPointsRouter");
const deliveryTypesRouter = require("./deliveryTypesRouter");
const orderDeliveryStepsRouter = require("./orderDeliveryStepsRouter");
const userNotificationsRouter = require("./userNotificationsRouter");
const usersOnOrderRouter = require("./usersOnOrderRouter");
const rolesRouter = require("./rolesRouter");

router.use('/users', usersRouter)
router.use('/orders', ordersRouter)
router.use('/deliveryPoints', deliveryPointsRouter)
router.use('/deliveryTypes', deliveryTypesRouter)
router.use('/notifications', notificationsRouter)
router.use('/orderDeliverySteps', orderDeliveryStepsRouter)
router.use('/userNotifications', userNotificationsRouter)
router.use('/usersOnOrder', usersOnOrderRouter)
router.use('/roles', rolesRouter)

module.exports = router;
```

deliveryPointsRouter.js – роутер для работы с пунктами доставки.

Код:

```
const Router = require("express");
const router = Router();
const deliveryPointsController = require("../controllers/deliveryPointsController");
router.get("/getAll", deliveryPointsController.getAll);
router.get("/getAllOrdersDeliveryPoints", deliveryPointsController.getAllOrdersDeliveryPoints);
router.get("/getOrdersRoutes", deliveryPointsController.getOrdersRoutes);
router.get("/getDeliveryPointsStats", deliveryPointsController.getDeliveryPointsStats)


module.exports = router;
```

deliveryTypesRouter.js – роутер для работы с типами доставки.

Код:

```
const Router = require("express");
const router = Router();
const deliveryTypesController = require("../controllers/deliveryTypesController");

router.get("/getAll", deliveryTypesController.getAll);

module.exports = router;
```

orderDeliverySteps.js – роутер для работы с шагами доставки.

Код:

```
const Router = require("express");
const router = Router();
const orderDeliveryStepsController = require("../controllers/orderDeliveryStepsController");
const authMiddleware = require("../middleware/authMiddleware");

router.get("/getOrderNotifications", authMiddleware, orderDeliveryStepsController.getOrderNotifications);

module.exports = router;
```

ordersRouter.js – роутер для работы с посылками.

Код:

```
const Router = require("express");
const router = Router();
const ordersController = require("../controllers/ordersController");

router.post("/createOrder", ordersController.createOrder);
router.get("/getExpectedOrders", ordersController.getExpectedOrders);
router.get("/getCurrentOrders", ordersController.getCurrentOrders);
router.put("/setOrderArrived", ordersController.setOrderArrived);
router.put("/setOrderLeft", ordersController.setOrderLeft);
router.get("/getDeliveryTypesStats", ordersController.getDeliveryTypesStats);
router.get("/getOrdersStats", ordersController.getOrdersStats);
```

```
module.exports = router;
```

rolesRouter.js – роутер для работы с ролями.

Код:

```
const Router = require("express");
const router = Router();
const rolesController = require("../controllers/rolesController");

router.get("/getAll", rolesController.getAll);

module.exports = router;
```

userNotificationsRouter.js – роутер для работы с уведомлениями.

Код:

```
const Router = require("express");
const router = Router();
const userNotificationsController = require("../controllers/userNotificationsController");

router.get("/getUsersNewNotifications", userNotificationsController.getUsersNewNotifications)
router.put("/readNotification", userNotificationsController.readNotification)

module.exports = router;
```

usersOnOrderRouter.js – роутер для работы с информацией о заказах, отслеживаемых пользователем.

Код:

```
const Router = require("express");
const router = Router();
const usersOnOrderController = require("../controllers/usersOnOrderController");
const authMiddleware = require("../middleware/authMiddleware");

router.get("/getUsersOrdersNotFinished", authMiddleware, usersOnOrderController.getUsersOrdersNotFinished);
router.get("/getUsersOrdersFinished", authMiddleware, usersOnOrderController.getUsersOrdersFinished);
router.post("/addUsersOrder", usersOnOrderController.addUsersOrder)

module.exports = router;
```

usersRouter.js – роутер для работы с пользователями.

Код:

```
const Router = require("express");
const router = Router();
const usersController = require("../controllers/usersController");
const authMiddleware = require("../middleware/authMiddleware");

router.post("/register", usersController.register);
router.post("/login", usersController.login);
```

```
router.get("/check", authMiddleware, usersController.check)
router.get("/getUsersRolesStats", usersController.getUsersRolesStats);

module.exports = router;
```

4.2.7 Разработка контроллеров

Котроллеры — файлы, содержащие функции для обработки бизнес-логики при получении HTTP-запросов. Контроллеры отделяют логику приложения от маршрутов и от базы данных.

deliveryPointsController – контроллер, управляющий логикой, связанной с пунктами доставки.

Методы контроллера:

- getAll(req, res) — получение всех пунктов доставки;
- getAllOrdersDeliveryPoints(req, res, next) - получение всех пунктов доставки, по которым проходит конкретный заказ;
- getOrdersRoutes(req, res, next) - получение маршрута заказа: уже пройденных и будущих пунктов;
- getDeliveryPointsStats(req, res, next) – получение статистики по пунктам доставки.

Код:

```
const { DeliveryPoints, OrderDeliverySteps } = require("../models/models");
const ApiError = require("../error/ApiError");
const { Op, Sequelize } = require("sequelize");

class DeliveryPointsController {

    // Получить все пункты доставки
    async getAll(req, res) {
        const deliveryPoints = await DeliveryPoints.findAll();
        return res.json(deliveryPoints);
    }

    // Получить все пункты маршрута конкретного заказа
    async getAllOrdersDeliveryPoints(req, res, next) {
        const orderId = req.query.orderId;
        if (!orderId) {
            return next(ApiError.badRequest("Не указан ID заказа"));
        }

        const deliveryPoints = await OrderDeliverySteps.findAll({
            where: { order: orderId },
            include: [{ model: DeliveryPoints, as: "point" }]
        });

        const pointsArray = deliveryPoints.map(dp => dp.point);
        return res.json(pointsArray);
    }

    // Получить завершённые и предстоящие шаги маршрута по заказу
```

```
async getOrdersRoutes(req, res, next) {
    const orderId = req.query.orderId;
    if (!orderId) {
        return next(ApiError.badRequest("Не указан ID заказа"));
    }

    // Завершённые шаги
    const finishedRoute = await OrderDeliverySteps.findAll({
        where: {
            order: orderId,
            arrived_at: { [Op.ne]: null }
        },
        include: [{ model: DeliveryPoints, as: "point" }],
        order: [['step', 'ASC']]
    });

    const finishedRoutePoints = finishedRoute.map(dp => dp.point);

    // Все шаги маршрута
    const allSteps = await OrderDeliverySteps.findAll({
        where: { order: orderId },
        include: [{ model: DeliveryPoints, as: 'point' }],
        order: [['step', 'ASC']]
    });

    // Будущие шаги
    const futureRoute = allSteps.filter((step, index, array) => {
        const nextStep = array[index + 1];
        return (
            step.arrived_at === null ||
            (step.arrived_at !== null && step.left_at === null) ||
            (
                step.arrived_at !== null &&
                step.left_at !== null &&
                nextStep && nextStep.arrived_at === null
            )
        );
    });

    const futureRoutePoints = futureRoute.map(dp => dp.point);

    return res.json({ finishedRoutePoints, futureRoutePoints });
}

// Получить среднее время обработки в пунктах
async getDeliveryPointsStats(req, res, next) {
    try {
        const stats = await OrderDeliverySteps.findAll({
            attributes: [
                [Sequelize.col('point.address'), 'address'],
                [
                    Sequelize.fn(
                        'AVG',
                        Sequelize.literal(`TIMESTAMPDIFF(MINUTE, arrived_at, left_at)`)
                    ),
                    'avgTime'
                ]
            ],
            include: [{ model: DeliveryPoints, as: 'point', attributes: [] }],
```

```
            where: {
              arrived_at: { [Op.ne]: null },
              left_at: { [Op.ne]: null }
            },
            group: ['point.address'],
            order: [[Sequelize.col('point.address'), 'ASC']],
            raw: true
          });

          const result = stats.map(({ address, avgTime }) => ({
            address,
            avgTime: parseFloat(avgTime)
          }));

          return res.json(result);
        } catch (error) {
          next(error);
        }
    }
}

module.exports = new DeliveryPointsController();
```

deliveryTypesController – контроллер, управляющий логикой, связанной с типами доставки.

Методы контроллера:

- getAll(req, res) — получение всех типов доставки.

Код:

```
const {DeliveryTypes} = require('../models/models');

class DeliveryTypesController {

    async getAll(req, res) {
        const deliveryTypes = await DeliveryTypes.findAll();
        return res.json(deliveryTypes);
    }

}

module.exports = new DeliveryTypesController();
```

orderDeliveryStepsController – контроллер, управляющий логикой, связанной с шагами маршрута доставки и уведомлениями пользователей.

Методы контроллера:

- getOrderNotifications(req, res, next) — получение уведомлений по конкретному заказу с указанием адресов пунктов доставки;

Код:

```
const { Notifications, DeliveryPoints, OrderDeliverySteps, Orders } = require("../models/models");
const ApiError = require("../error/ApiError");

class OrderDeliveryStepsController {
```

```
    // Получить уведомления по заказу с адресами
    async getOrderNotifications(req, res, next) {
        const orderId = req.query.orderId;
        if (!orderId) {
            return next(ApiError.badRequest("Не указан ID посылки!"));
        }

        const steps = await Notifications.findAll({
            where: { order: orderId },
            include: [
                {
                    model: Orders,
                    as: 'orderData',
                    include: [
                        {
                            model: OrderDeliverySteps,
                            as: 'order_delivery_steps',
                            include: [
                                { model: DeliveryPoints, as: 'point' }
                            ],
                            order: [['step', 'ASC']]
                        }
                    ]
                }
            ]
        });

        const stepsMessages = [];

        steps.forEach((step) => {
            const deliveryStep = step.orderData.order_delivery_steps.find(
                s => s.step === step.step
            );

            stepsMessages.push({
                id: step.id,
                message: step.message.split(',')[0],
                address: deliveryStep?.point?.address || 'Адрес не найден',
                datetime: step.createdAt
            });
        });

        return res.status(200).json(stepsMessages);
    }
}

module.exports = new OrderDeliveryStepsController();
```

ordersController – контроллер, управляющий логикой, связанной с посылками.

Методы контроллера:

- createOrder(req, res, next) — создание нового заказа и маршрута доставки;
- getExpectedOrders(req, res, next) - получение ожидаемых заказов для пункта, управляемого пользователем;

- getCurrentOrders(req, res, next) - получение текущих заказов, которые уже прибыли, но ещё не покинули пункт;
- setOrderArrived(req, res, next) - отметить заказ как прибывший в пункт;
- setOrderLeft(req, res, next) - отметить, что заказ покинул текущий пункт;
- getDeliveryTypesStats(req, res, next) - получение статистики по количеству заказов для каждого типа доставки.

Код:

```
const { Orders, DeliveryPoints, OrderDeliverySteps, Users, UsersOnOrder } = require("../models/models");
const { buildRoute } = require("../utils/routeBuilder");
const ApiError = require("../error/ApiError");
const sequelize = require("../db");
const { DataTypes, Op } = require("sequelize");
const nodemailer = require('nodemailer');

class OrdersController {

  // Создание нового заказа
  async createOrder(req, res, next) {
    const { sender, from, to, type, length, width, height, weight, value, passport, cost } = req.body;

    // Генерация случайного трек-кода (10 символов)
    const chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
    let result = '';
    for (let i = 0; i < 10; i++) {
      result += chars.charAt(Math.floor(Math.random() * chars.length));
    }
    const track_code = result;

    // Создание записи заказа
    const order = await Orders.create({
      sender,
      track_code,
      from_delivery_point: from,
      to_delivery_point: to,
      delivery_type: type,
      length,
      width,
      height,
      weight,
      value,
      receiver_passport: passport,
      cost
    });

    // Построение маршрута
    const from_hub = await DeliveryPoints.findOne({ where: { id: from } });
    const to_hub = await DeliveryPoints.findOne({ where: { id: to } });
    const hubs = await DeliveryPoints.findAll();
    const route = buildRoute(from_hub, to_hub, hubs, 45);

    // Сохранение шагов маршрута
    const points = [];
    let iterator = 1;
    route.forEach(route => {
      points.push({
```

```
            order: order.id,
            step: iterator,
            delivery_point: route.id
        });
        iterator += 1;
    });
    OrderDeliverySteps.bulkCreate(points);

    return res.status(200).json({ track_code });
}

// Получение ожидаемых заказов (которые должны прибыть в точку пользователя)
async getExpectedOrders(req, res, next) {
    const userId = req.query.userId;
    if (!userId) return next(ApiError.badRequest("Не указан ID пользователя!"));

    const point = await DeliveryPoints.findOne({ where: { manager: userId } });
    if (!point) return res.status(200).json([]);

    const steps = await OrderDeliverySteps.findAll({
        where: { delivery_point: point.id },
        include: [{ model: Orders, as: 'orderData', include: [{ model: Users, as: 'senderData' }] }]
    });

    const expectedOrders = [];

    for (const step of steps) {
        const { step: stepNum, order, orderData, arrived_at } = step;

        if (stepNum === 1 && arrived_at == null) {
            expectedOrders.push(orderData);
        } else if (arrived_at == null) {
            const prevStep = await OrderDeliverySteps.findOne({
                where: { order, step: stepNum - 1 }
            });

            if (prevStep?.left_at) {
                expectedOrders.push(orderData);
            }
        }
    }

    return res.json(expectedOrders);
}

// Получение заказов, которые уже прибыли в точку, но ещё не отправлены
async getCurrentOrders(req, res, next) {
    const userId = req.query.userId;
    if (!userId) return next(ApiError.badRequest("Не указан ID пользователя!"));

    const point = await DeliveryPoints.findOne({ where: { manager: userId } });
    if (!point) return res.status(200).json([]);

    const steps = await OrderDeliverySteps.findAll({
        where: {
            delivery_point: point.id,
            arrived_at: { [Op.not]: null },
            left_at: null,
        },
```

```
            include: [{ model: Orders, as: 'orderData', include: [{ model: Users, as: 'senderData' }] }]
        });

        const currentOrders = steps.map(step => step.orderData);
        return res.json(currentOrders);
    }

    // Отметить, что заказ прибыл в текущий пункт
    async setOrderArrived(req, res, next) {
        const userId = req.query.userId;
        const orderId = req.query.orderId;
        if (!userId || !orderId) return next(ApiError.badRequest("Не указаны данные!"));

        const point = await DeliveryPoints.findOne({ where: { manager: userId } });
        if (!point) return next(ApiError.badRequest("Пункт выдачи не найден для данного пользователя!"));

        const [updatedCount] = await OrderDeliverySteps.update(
            { arrived_at: new Date() },
            {
                where: {
                    order: orderId,
                    delivery_point: point.id,
                    arrived_at: null
                }
            }
        );

        if (updatedCount === 0) {
            return res.status(404).json({ message: 'Шаг уже был обновлён или не найден.' });
        }

        // Проверка: является ли этот шаг последним
        const steps = await OrderDeliverySteps.findAll({
            where: { order: orderId },
            order: [['step', 'DESC']]
        });

        if (!steps.length) return res.status(404).json({ message: 'Шаги доставки не найдены.' });

        const maxStep = steps[0].step;
        const justUpdatedStep = await OrderDeliverySteps.findOne({
            where: { order: orderId, delivery_point: point.id }
        });

        if (justUpdatedStep.step === maxStep) {
            const order = await Orders.findByPk(orderId);

            // Получаем всех отслеживающих пользователей
            const trackingUsers = await UsersOnOrder.findAll({
                where: { order: orderId },
                include: [{ model: Users, as: 'userData' }]
            });

            const emails = trackingUsers.map(u => u.userData?.email).filter(email => !!email);

            // Отправка писем
            if (emails.length > 0) {
                const transporter = nodemailer.createTransport({
                    host: 'smtp.mail.ru',
```

```
            port: 465,
            secure: true,
            auth: {
              user: 'dubkov.05@mail.ru',
              pass: 'R9nb8nxphsvfXduBxbXL' // 🔒 вынести в .env
            }
          });

          for (const email of emails) {
            await transporter.sendMail({
              from: '"Служба доставки" <dubkov.05@mail.ru>',
              to: email,
              subject: 'Заказ доставлен!',
              text: `Здравствуйте! Заказ с трек-номером ${order.track_code} прибыл в пункт назначения.`
            });
          }
        }
      }

      return res.status(200).json({ updated: updatedCount });
    }

    // Отметить, что заказ покинул текущий пункт
    async setOrderLeft(req, res, next) {
      const userId = req.query.userId;
      const orderId = req.query.orderId;
      if (!userId || !orderId) return next(ApiError.badRequest("Не указаны данные!"));

      const point = await DeliveryPoints.findOne({ where: { manager: userId } });
      if (!point) return next(ApiError.badRequest("Пункт выдачи не найден для данного пользователя!"));

      const update = await OrderDeliverySteps.update(
        { left_at: new Date() },
        {
          where: {
            order: orderId,
            delivery_point: point.id,
            arrived_at: { [Op.not]: null },
            left_at: null
          }
        }
      );

      return res.status(200).json({ updated: update[0] });
    }

    // Статистика по типам доставки: сколько заказов у каждого
    async getDeliveryTypesStats(req, res, next) {
      const typesWithOrdersCount = await sequelize.query(`
        SELECT
          delivery_types.name,
          COUNT(orders.id) AS orders_count
        FROM
          delivery_types
        LEFT JOIN
          orders ON orders.delivery_type = delivery_types.id
        GROUP BY
          delivery_types.name;
      `, {
```

```
            type: sequelize.QueryTypes.SELECT
        });

        return res.json(typesWithOrdersCount);
    }

    // Статистика по заказам за последние 10 дней
    async getOrdersStats(req, res, next) {
        const ordersCount = await sequelize.query(`
            WITH RECURSIVE dates AS (
                SELECT CURDATE() - INTERVAL 9 DAY AS order_date
                UNION ALL
                SELECT order_date + INTERVAL 1 DAY
                FROM dates
                WHERE order_date + INTERVAL 1 DAY <= CURDATE()
            )
            SELECT
                dates.order_date,
                COUNT(orders.id) AS orders_count
            FROM dates
            LEFT JOIN orders ON DATE(orders.createdAt) = dates.order_date
            GROUP BY dates.order_date
            ORDER BY dates.order_date;
        `, {
            type: sequelize.QueryTypes.SELECT
        });

        return res.json(ordersCount);
    }
}

module.exports = new OrdersController();
```

userNotificationsController – контроллер, управляющий логикой, связанной с уведомлениями.

Методы контроллера:

- getUsersNewNotifications(req, res, next) — получение всех непрочитанных уведомлений пользователя;
- readNotification(req, res, next) - отметить уведомление как прочитанное;

Код:

```
const ApiError = require("../error/ApiError");
const {UserNotifications, Orders, Notifications} = require("../models/models");

class UserNotificationsController {

    async getUsersNewNotifications(req, res, next) {
        const userId = req.query.userId;

        if (!userId) {
            return next(ApiError.badRequest("Не указан ID пользователя!"));
        }

        const notifications = await UserNotifications.findAll({
            where: {
```

```
            user: userId,
            is_read: false
          },
          include: [
            {
              model: Notifications,
              as: "notificationData",
            }
          ]
        });
        return res.json(notifications);
    }

    async readNotification(req, res, next) {
        const notificationId = req.query.notificationId;

        if (!notificationId) {
            return next(ApiError.badRequest("Не указан ID уведомления!"));
        }

        const [updatedRowsCount] = await UserNotifications.update(
            { is_read: true },
            { where: { id: notificationId } }
        );

        if (updatedRowsCount === 0) {
            return next(ApiError.badRequest("Уведомление не найдено или уже прочитано"));
        }

        return res.json({ message: "Уведомление помечено как прочитанное" });
    }

}

module.exports = new UserNotificationsController();
```

usersController – контроллер, управляющий логикой, связанной с пользователями.

Методы контроллера:

- register(req, res) - регистрация пользователя;
- login(req, res, next) — авторизация пользователя;
- check(req, res, next) — проверка авторизации;
- getUsersRolesStats(req, res, next) - получение статистики по количеству пользователей в разрезе ролей.

Код:

```
const bcrypt = require('bcrypt'); // Для хеширования паролей
const jwt = require('jsonwebtoken'); // Для генерации JWT-токенов
const { Users } = require("../models/models"); // Модель пользователей
const ApiError = require("../error/ApiError"); // Обработка ошибок
const sequelize = require("../db"); // Подключение к БД (для raw-запросов)

// Функция генерации JWT
const createJWT = (id, name, surname, patronymic, email, passport, role) => {
    return jwt.sign(
```

```
        { id, name, surname, patronymic, email, passport, role }, // payload токена
        process.env.JWT_SECRET_KEY, // секретный ключ из .env
        { expiresIn: '1d' } // срок действия токена: 1 день
    );
}

class UsersController {

    // Регистрация нового пользователя
    async register(req, res) {
        const { email, password, name, surname, patronymic, passport } = req.body;

        // Хешируем пароль
        const hashedPassword = await bcrypt.hash(password, 12);

        // Создаём пользователя
        const user = await Users.create({
            name,
            surname,
            patronymic,
            email,
            password: hashedPassword,
            passport
        });

        // Генерация токена
        const token = createJWT(user.id, user.name, user.surname, user.patronymic, user.email, user.passport,
user.role);
        return res.status(200).json({ token });
    }

    // Авторизация пользователя
    async login(req, res, next) {
        const { email, password } = req.body;

        // Проверяем, существует ли пользователь
        const user = await Users.findOne({ where: { email } });
        if (!user) {
            return next(ApiError.badRequest("Пользователь не найден!"));
        }

        // Сравниваем введённый пароль с хешом из базы
        const comparePassword = await bcrypt.compare(password, user.password);
        if (!comparePassword) {
            return next(ApiError.badRequest("Неверный пароль!"));
        }

        // Генерируем токен
        const token = createJWT(user.id, user.name, user.surname, user.patronymic, user.email, user.passport,
user.role);
        res.status(200).json({ token });
    }

    // Проверка авторизации (вызывается middleware, когда пользователь уже аутентифицирован)
    async check(req, res, next) {
        const token = createJWT(
            req.user.id,
            req.user.name,
            req.user.surname,
```

```
            req.user.patronymic,
            req.user.email,
            req.user.passport,
            req.user.role
        );

        return res.status(200).json({ token });
    }

    // Получение статистики по ролям пользователей (сколько пользователей с каждой ролью)
    async getUsersRolesStats(req, res, next) {
        const usersCount = await sequelize.query(`
            SELECT
                roles.name,
                COUNT(users.id) AS user_count
            FROM
                roles
            LEFT JOIN
                users ON users.role = roles.id
            GROUP BY
                roles.name;
        `, {
            type: sequelize.QueryTypes.SELECT
        });

        return res.json(usersCount);
    }

}

module.exports = new UsersController();
```

usersOnOrderController – контроллер, управляющий логикой, связанной с посылками пользователей.

Методы контроллера:

- getUsersOrdersNotFinished(req, res, next) — получить список заказов пользователя, которые ещё не завершены (последний шаг доставки не завершён);
- getUsersOrdersFinished(req, res, next) — получить список завершённых заказов пользователя (последний шаг доставки выполнен);
- addUsersOrder(req, res, next) — добавить заказ к отслеживаемым пользователем по трек-коду;

Код:

```
const {UsersOnOrder, Orders, OrderDeliverySteps} = require("../models/models");
const ApiError = require("../error/ApiError");
const sequelize = require("../db");
const {Op} = require("sequelize");

class UsersOnOrderController {

    async getUsersOrdersNotFinished(req, res, next) {
        const userId = req.query.userId;
```

```
        if (!userId) {
            return next(ApiError.badRequest("Не указан ID пользователя!"));
        }
        const orders = await UsersOnOrder.findAll({
            where: { user: userId },
            include: [
                {
                    model: Orders,
                    as: "orderData",
                    required: true,
                    include: [
                        {
                            model: OrderDeliverySteps,
                            as: "order_delivery_steps",
                            required: true,
                            where: {
                                step: sequelize.literal(`(
                                SELECT MAX(ods2.step)
                                FROM order_delivery_steps AS ods2
                                WHERE ods2.order = orderData.id
                            )`),
                                left_at: null
                            }
                        }
                    ]
                }
            ],
            order: [[{ model: Orders, as: 'orderData' }, 'id', 'DESC']]
        });

        const ordersData = orders.map(order => order.orderData);

        return res.json(ordersData);
    }

    async getUsersOrdersFinished(req, res, next) {
        const userId = req.query.userId;
        if (!userId) {
            return next(ApiError.badRequest("Не указан ID пользователя!"));
        }
        const orders = await sequelize.query(`
    SELECT o.*
    FROM users_on_order uoo
    JOIN orders o ON o.id = uoo.order
    WHERE uoo.user = :userId
      AND EXISTS (
        SELECT 1
        FROM order_delivery_steps ods
        WHERE ods.order = o.id
          AND ods.left_at IS NOT NULL
          AND ods.step = (
            SELECT MAX(ods2.step)
            FROM order_delivery_steps ods2
            WHERE ods2.order = o.id
        )
      )
    ORDER BY o.id DESC
    `, {
            replacements: { userId },
```

```
            type: sequelize.QueryTypes.SELECT
        });

        return res.json(orders);
    }

    async addUsersOrder(req, res, next) {
        const userId = req.query.userId;
        const trackCode = req.query.trackCode;
        if (!userId || !trackCode) {
            return next(ApiError.badRequest("Не указаны данные!"));
        }

        const order = await Orders.findOne({
            where: {track_code: trackCode},
        })

        if (!order) {
            return next(ApiError.badRequest("Посылка не найдена!"));
        }

        const userOnOrder = await UsersOnOrder.findOne(
            {where: {user: userId, order: order.id}}
        )

        if (userOnOrder) {
            return next(ApiError.badRequest("Вы уже отслеживаете эту посылку!"));
        }

        const created = await UsersOnOrder.create({
            user: userId,
            order: order.id,
        })

        return res.json(created);
    }
}

module.exports = new UsersOnOrderController();
```

4.2.8 Разработка модуля постройки маршрута

Модуль routeBuilder.js реализует алгоритм построения маршрута доставки между двумя точками (A и B) через промежуточные узлы, основываясь на географических координатах, направлении и допустимом отклонении от прямого пути.

Код:

```
function toRadians(degrees) {
    return degrees * (Math.PI / 180);
}

function toDegrees(radians) {
    return radians * (180 / Math.PI);
}

// Гаверсинус: расстояние между двумя координатами
```

```
function haversineDistance(p1, p2) {
    const R = 6371; // радиус Земли в км
    const dLat = toRadians(p2.latitude - p1.latitude);
    const dLon = toRadians(p2.longitude - p1.longitude);
    const lat1 = toRadians(p1.latitude);
    const lat2 = toRadians(p2.latitude);

    const a = Math.sin(dLat / 2) ** 2 +
        Math.cos(lat1) * Math.cos(lat2) *
        Math.sin(dLon / 2) ** 2;
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

    return R * c;
}

// Азимут: направление от p1 к p2
function calculateBearing(p1, p2) {
    const lat1 = toRadians(p1.latitude);
    const lat2 = toRadians(p2.latitude);
    const dLon = toRadians(p2.longitude - p1.longitude);

    const y = Math.sin(dLon) * Math.cos(lat2);
    const x = Math.cos(lat1) * Math.sin(lat2) -
        Math.sin(lat1) * Math.cos(lat2) * Math.cos(dLon);
    const bearing = Math.atan2(y, x);

    return (toDegrees(bearing) + 360) % 360;
}

// Разность углов (минимальная, с учётом круга)
function angleDifference(a1, a2) {
    const diff = Math.abs(a1 - a2) % 360;
    return diff > 180 ? 360 - diff : diff;
}

// Основной алгоритм построения маршрута
function buildRoute(A, B, hubs, maxAngle = 15) {
    const route = [A];
    let current = A;
    let remainingHubs = [...hubs];

    while (true) {
        const bearingToB = calculateBearing(current, B);
        let candidates = remainingHubs.filter(hub => {
            const distToHub = haversineDistance(current, hub);
            const distHubToB = haversineDistance(hub, B);
            const bearingToHub = calculateBearing(current, hub);
            const angle = angleDifference(bearingToB, bearingToHub);

            return distToHub < distHubToB && angle <= maxAngle;
        });

        if (candidates.length === 0) break;

        // Выбрать ближайший подходящий хаб
        candidates.sort((a, b) =>
            haversineDistance(current, a) - haversineDistance(current, b)
        );
```

```
        const nextHub = candidates[0];
        route.push(nextHub);
        remainingHubs = remainingHubs.filter(h => h !== nextHub);
        current = nextHub;
    }

    route.push(B);
    return route;
}

module.exports = {
    buildRoute,
}
```

4.3 Разработка клиентской части

Клиентская часть веб-приложения разработана на JavaScript с использованием фреймворка React и реализует пользовательский интерфейс для взаимодействия с сервером и управления заказами доставки. Для управления состоянием используется mobx, а для маршрутизации — React Router.

4.3.1 Разработка корневого React-компонента

App.jsx – главный компонент приложения, который отвечает за инициализацию пользовательского состояния и роутинг.

- Используется mobx-react-lite и контекст Context для управления состоянием пользователя.
- При монтировании компонента вызывается check() (из usersAPI.js) — запрос на сервер для проверки и получения данных о текущем пользователе.
- Пока идёт загрузка данных пользователя, отображается спиннер загрузки (Spin из antd), центрированный с помощью Flex.
- После успешной проверки, в состояние user записываются данные пользователя, и выставляется флаг isAuthenticated.
- Обернут в BrowserRouter для маршрутизации.
- Вёрстка строится из двух компонентов: боковой панели Sidebar и основного блока роутинга AppRouter, который отвечает за отображение страниц.

Код:

```
import Sidebar from "./components/Sidebar.jsx";
import AppRouter from "./components/AppRouter.jsx";
import {BrowserRouter} from "react-router-dom";
import {observer} from "mobx-react-lite";
import {useContext, useEffect, useState} from "react";
import {Context} from "./main.jsx";
import {check} from "./http/usersAPI.js";
import {Flex, Spin} from "antd";
```

```
const App = observer(() => {

    const { user } = useContext(Context);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        check().then(data => {
            user.setUser(data);
            user.setIsAuthenticated(true);
        }).finally(() => setLoading(false));
    })

    if (loading) {
        return (
            <Flex className="h-[100vh] w-full" align="center" justify='center' gap="middle">
                <Spin size="large" />
            </Flex>
        )
    }

    return (
    <>
        <BrowserRouter>
            <Sidebar>
                <AppRouter/>
            </Sidebar>
        </BrowserRouter>
    </>
  )
});

export default App
```

4.3.2 Разработка компонента боковой панели

Компонент боковой навигационной панели (Sidebar) отвечает за отображение меню в зависимости от роли пользователя, а также содержит элементы управления, такие как переключатель сворачивания меню и блок уведомлений.

Ключевая функциональность:

- Адаптивное меню:
  - В зависимости от роли пользователя (admin, manager, client, гость) отображаются разные пункты навигации.
  - Используются маршруты из utils/consts.js.
- Уведомления:
  - Для клиента отображается иконка уведомлений с счётчиком непрочитанных.
  - При открытии поповера все уведомления автоматически помечаются

как прочитанные через readNotification.

- Новые уведомления проверяются каждые 30 секунд с помощью getUsersNewNotifications.

- Интеграция с MobX:
  - Контекст пользователя получен через Context и useContext.
  - Наблюдение за изменениями состояния — через observer.
- Используемые библиотеки:
  - Ant Design для визуальных компонентов (Layout, Menu, Popover, Badge, Avatar).
  - react-router-dom для навигации (NavLink, useLocation).

Код:

```
import React, {useContext, useEffect, useState} from 'react';
import {
    MenuFoldOutlined,
    MenuUnfoldOutlined,
    UserOutlined,
    NotificationOutlined,
    CalculatorOutlined,
    DashboardOutlined, CodeSandboxOutlined,
} from '@ant-design/icons';
import {Button, Layout, Menu, Avatar, theme, Space, Badge, Popover, Flex, Typography} from 'antd';
import {
    ADMIN_ROUTE,
    AUTH_ROUTE,
    CALCULATOR_ROUTE,
    CREATE_ORDER_ROUTE,
    MANAGER_ROUTE,
    PROFILE_ROUTE
} from "../utils/consts.js";
import {Context} from "../main.jsx";
import {observer} from "mobx-react-lite";
import {NavLink, useLocation} from "react-router-dom";
import {getUsersNewNotifications, readNotification} from "../http/notificationsAPI.js";

const {Header, Sider, Content} = Layout;
const {Text} = Typography;
```

```
const Sidebar = observer(({children}) => {
    const [collapsed, setCollapsed] = useState(false);
    const [notificationCount, setNotificationCount] = useState(0);
    const [notifications, setNotifications] = useState([]);
    const [popoverOpen, setPopoverOpen] = useState(false);

    const {
        token: {colorBgContainer},
    } = theme.useToken();

    const {user} = useContext(Context);
    const location = useLocation();

    const [sidebarItems, setSidebarItems] = useState([]);

    const getSelectedKey = () => {
        if (location.pathname.startsWith('/authorization')) return AUTH_ROUTE;
        if (location.pathname.startsWith('/registration')) return AUTH_ROUTE;
        if (location.pathname.startsWith('/profile')) return PROFILE_ROUTE;
        if (location.pathname.startsWith('/admin')) return ADMIN_ROUTE;
        if (location.pathname.startsWith('/calculator')) return CALCULATOR_ROUTE;
        if (location.pathname.startsWith('/createOrder')) return CREATE_ORDER_ROUTE;
        if (location.pathname.startsWith('/manager')) return MANAGER_ROUTE;
        return '';
    };

    useEffect(() => {
        if (!user.isAuthenticated) {
            setSidebarItems([
                {
                    key: AUTH_ROUTE,
                    icon: <UserOutlined/>,
                    label: <NavLink to={AUTH_ROUTE}>Авторизация</NavLink>,
                },
                {
                    key: CALCULATOR_ROUTE,
                    icon: <CalculatorOutlined/>,
                    label: <NavLink to={CALCULATOR_ROUTE}>Калькулятор</NavLink>,
```

```
      },
    ]);
  } else if (user.isAuthenticated && user.user.role === 1) {
    setSidebarItems([
      {
        key: ADMIN_ROUTE,
        icon: <DashboardOutlined/>,
        label: <NavLink to={ADMIN_ROUTE}>Админ-панель</NavLink>,
      },
      {
        key: CALCULATOR_ROUTE,
        icon: <CalculatorOutlined/>,
        label: <NavLink to={CALCULATOR_ROUTE}>Калькулятор</NavLink>,
      },
    ]);
  } else if (user.isAuthenticated && user.user.role === 2) {
    setSidebarItems([
      {
        key: MANAGER_ROUTE,
        icon: <DashboardOutlined/>,
        label: <NavLink to={MANAGER_ROUTE}>Управляющий</NavLink>,
      },
      {
        key: CALCULATOR_ROUTE,
        icon: <CalculatorOutlined/>,
        label: <NavLink to={CALCULATOR_ROUTE}>Калькулятор</NavLink>,
      },
    ]);
  } else if (user.isAuthenticated && user.user.role === 3) {
    setSidebarItems([
      {
        key: PROFILE_ROUTE,
        icon: <DashboardOutlined/>,
        label: <NavLink to={PROFILE_ROUTE}>Профиль</NavLink>,
      },
      {
        key: CALCULATOR_ROUTE,
        icon: <CalculatorOutlined/>,
        label: <NavLink to={CALCULATOR_ROUTE}>Калькулятор</NavLink>,
```

```
            },
            {
              key: CREATE_ORDER_ROUTE,
              icon: <CodeSandboxOutlined/>,
              label: <NavLink to={CREATE_ORDER_ROUTE}>Новая посылка</NavLink>,
            }
          ]);
        }
    }, [user.isAuthenticated]);

    useEffect(() => {
        let intervalId;

        const fetchNotifications = async () => {
            try {
                if (user.isAuthenticated && user.user?.id ) {
                    const data = await getUsersNewNotifications(user.user.id);
                    setNotifications(data);
                    setNotificationCount(data.length);
                }
            } catch (error) {
                console.error('Ошибка при получении уведомлений:', error);
            }
        };

        if (user.isAuthenticated) {
            fetchNotifications(); // первая загрузка
            intervalId = setInterval(fetchNotifications, 30000);
        }

        return () => {
            if (intervalId) clearInterval(intervalId);
        };
    }, [user.isAuthenticated, user.user?.id]);

    const handlePopoverOpen = async (newOpen) => {
        if (newOpen && notifications.length > 0) {
            try {
                await Promise.all(
```

```
            notifications.map((n) => readNotification(n.id))
          );
          // Обнуляем счётчик и обновляем список
          setNotificationCount(0);
        } catch (e) {
          console.error('Ошибка при пометке уведомлений как прочитанных:', e);
        }
      }
    };

    return (
      <Layout style={{minHeight: '100vh'}}>
        <Sider trigger={null} collapsible collapsed={collapsed}>
          <div className="demo-logo-vertical"/>
          <Menu
            theme="dark"
            mode="inline"
            selectedKeys={[getSelectedKey()]}
            items={sidebarItems}
          />
        </Sider>
        <Layout>
                    <Header className="flex justify-between items-center" style={{padding: 0, background: colorBgContainer}}>
          <Button
            type="text"
            icon={collapsed ? <MenuUnfoldOutlined/> : <MenuFoldOutlined/>}
            onClick={() => setCollapsed(!collapsed)}
            style={{fontSize: '16px', width: 64, height: 64}}/>
          {user.isAuthenticated && user.user.role === 3 &&
            <Space style={{marginRight: 12}}>
              <Popover
                title="Уведомления"
                trigger="click"
                onOpenChange={handlePopoverOpen}
                content={
                  <Flex vertical gap="small" style={{ maxWidth: 300 }}>
                    {notifications.length === 0 ? (
                      <Text type="secondary">Нет новых уведомлений</Text>
```

```
                    ) : (
                      notifications.map((n) => (
                        <div
                          key={n.id}
                          style={{
                            padding: '12px',
                            borderRadius: '8px',
                            backgroundColor: '#f6f6f6',
                            border: '1px solid #e0e0e0',
                            boxShadow: '0 1px 3px rgba(0, 0, 0, 0.05)'
                          }}
                        >
                          <Flex vertical>
                            <Text strong ellipsis style={{ fontSize: '14px', marginBottom: '4px' }}>
                              🔔 {n.notificationData.title}
                            </Text>
                                  <Text type="secondary" style={{ fontSize: '12px' }} ellipsis={{ tooltip:
n.notificationData.message }}>
                              {n.notificationData.message}
                            </Text>
                          </Flex>
                        </div>
                      ))
                    )}
                  </Flex>
                }
              >
                <Badge count={notificationCount}>
                  <Avatar icon={<NotificationOutlined/>}/>
                </Badge>
              </Popover>
            </Space>
          }
        </Header>
        <Content className="p-4">
          {children}
        </Content>
      </Layout>
    </Layout>
```

```
    );
});

export default Sidebar;
```

4.3.3 Разработка компонента маршрутизации

Компонент маршрутизации приложения AppRouter.jsx определяет, какие страницы доступны пользователю в зависимости от его роли и состояния авторизации.

Ключевая функциональность:

- Ролевая маршрутизация:
  - Использует данные из MobX-хранилища (user), полученного из контекста приложения (Context).
  - Определяет доступные маршруты по ролям:
    - role === 1 — Администратор (adminRoutes)
    - role === 2 — Менеджер (managerRoutes)
    - role === 3 — Клиент (authRoutes)
    - Гость — publicRoutes
- Обработка несуществующих маршрутов (*):
  - Перенаправляет:
    - Авторизованных пользователей — на соответствующую главную страницу (например, PROFILE_ROUTE для клиента).
    - Неавторизованных — на страницу авторизации (AUTH_ROUTE).
- Используемые библиотеки:
  - react-router-dom — маршрутизация через <Routes> и <Route>, перенаправления через <Navigate>.
  - mobx-react-lite — реактивное наблюдение за состоянием авторизации (observer).

Код:

```
import React, {useContext} from 'react';
import {Navigate, Route, Routes} from "react-router-dom";
import {adminRoutes, authRoutes, managerRoutes, publicRoutes} from "../utils/routes.js";
import {ADMIN_ROUTE, AUTH_ROUTE, MANAGER_ROUTE, PROFILE_ROUTE} from "../utils/consts.js";
import {Context} from "../main.jsx";
import {observer} from "mobx-react-lite";


const AppRouter = observer(() => {
```

```
const {user} = useContext(Context);

return (
    <Routes>
        {
            user.isAuthenticated && user.user.role === 3 && authRoutes.map(({path, Component}) => (
                <Route path={path} element={<Component/>}/>
            ))
        }
        {
            user.isAuthenticated && user.user.role === 1 && adminRoutes.map(({path, Component}) => (
                <Route path={path} element={<Component/>}/>
            ))
        }
        {
            user.isAuthenticated && user.user.role === 2 && managerRoutes.map(({path, Component}) => (
                <Route path={path} element={<Component/>}/>
            ))
        }
        {
            !user.isAuthenticated && publicRoutes.map(({path, Component}) => (
                <Route path={path} element={<Component/>}/>
            ))
        }
        {
                    user.isAuthenticated && user.user.role === 3 && <Route path="*" element={<Navigate to={PROFILE_ROUTE}/>} />
        }
        {
                    user.isAuthenticated && user.user.role === 1 && <Route path="*" element={<Navigate to={ADMIN_ROUTE}/>} />
        }
        {
                    user.isAuthenticated && user.user.role === 2 && <Route path="*" element={<Navigate to={MANAGER_ROUTE}/>} />
        }
        <Route path="*" element={<Navigate to={AUTH_ROUTE}/>} />
    </Routes>
);
```

```
});

export default AppRouter;
```

4.3.4 Разработка компонента карты

Компонент карты ProfileMap.jsx отображает интерактивную карту маршрута посылки: показывает точки доставки (начальные, промежуточные и конечные) и маршрут движения (завершённый и будущий).

Функциональность:

- Загрузка данных:
  - Запрашивает список всех точек маршрута по заказу (getAllOrdersDeliveryPoints).
  - Получает информацию о текущем маршруте: какие точки уже пройдены и какие ещё предстоят (getOrdersRoutes).
- Отображение на карте:
  - Использует react-leaflet:
    - MapContainer, TileLayer — карта OpenStreetMap.
    - Marker и Popup — отображение каждой точки маршрута с адресом.
    - Polyline — построение линий маршрута.
      - Завершённый маршрут — зелёный.
      - Будущий маршрут — красный.
- Построение маршрута (геокодирование):
  - Маршруты строятся через OSRM API (https://router.project-osrm.org/route/v1/driving).
  - Полученные координаты маршрута преобразуются для отображения ([longitude, latitude] → [latitude, longitude]).

Код:

```
import React, { useEffect, useState } from 'react';
import { MapContainer, TileLayer, Marker, Polyline, Popup } from 'react-leaflet';
import { getAllOrdersDeliveryPoints, getOrdersRoutes } from "../http/deliveryPointsAPI.js";

const ProfileMap = ({ orderId }) => {

    const [orderDeliveryPoints, setOrderDeliveryPoints] = useState([]);
    const [routes, setRoutes] = useState(null);

    const [routeCoordsFinished, setRouteCoordsFinished] = useState([]);
```

```
  const [routeCoordsFuture, setRouteCoordsFuture] = useState([]);

  useEffect(() => {
    if (!orderId) return;

    const fetchDeliveryPoints = async () => {
      const data = await getAllOrdersDeliveryPoints(orderId);
      setOrderDeliveryPoints(data);
    };

    const fetchRoutes = async () => {
      const data = await getOrdersRoutes(orderId);
      setRoutes(data);
    };

    setRouteCoordsFinished([]);
    setRouteCoordsFuture([]);

    fetchDeliveryPoints();
    fetchRoutes();
  }, [orderId]);

  useEffect(() => {
    if (!routes || !routes.finishedRoutePoints || !routes.futureRoutePoints) return;

    const fetchRouteCoords = async () => {
      try {
        // Finished route
        if (routes.finishedRoutePoints.length > 1) {
          const coordinatesFinished = routes.finishedRoutePoints.map(c => `${c.longitude},${c.latitude}`).join(';');
          const url1 = `https://router.project-osrm.org/route/v1/driving/${coordinatesFinished}?overview=full&geometries=geojson`;
          const res1 = await fetch(url1);
          const data1 = await res1.json();
          if (data1.routes?.length) {
            const coords = data1.routes[0].geometry.coordinates.map(([lon, lat]) => [lat, lon]);
            setRouteCoordsFinished(coords);
          }
        }
```

```
        // Future route
        if (routes.futureRoutePoints.length > 1) {
          const coordinatesFuture = routes.futureRoutePoints.map(c => `${c.longitude},${c.latitude}`).join(';');
                                const url2 = `https://router.project-osrm.org/route/v1/driving/${coordinatesFuture}?
overview=full&geometries=geojson`;
          const res2 = await fetch(url2);
          const data2 = await res2.json();
          if (data2.routes?.length) {
            const coords = data2.routes[0].geometry.coordinates.map(([lon, lat]) => [lat, lon]);
            setRouteCoordsFuture(coords);
          }
        }
      } catch (err) {
        console.error('Ошибка при построении маршрута:', err);
      }
    };

    fetchRouteCoords();
  }, [routes]);

  return (
    <MapContainer center={[60, 60]} zoom={3} style={{ height: '100%', minHeight: '450px', width: '100%' }}>
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
        attribution="&copy; OpenStreetMap contributors"
      />

      {orderDeliveryPoints.map((point, i) => (
        <Marker key={i} position={[point.latitude, point.longitude]}>
          <Popup>{point.address}</Popup>
        </Marker>
      ))}

      {routeCoordsFinished.length > 0 && (
        <Polyline positions={routeCoordsFinished} pathOptions={{ color: 'green', weight: 5 }} />
      )}
      {routeCoordsFuture.length > 0 && (
        <Polyline positions={routeCoordsFuture} pathOptions={{ color: 'red', weight: 5 }} />
```

```
        )}
      </MapContainer>
    );
};

export default ProfileMap;
```

4.3.5 Разработка страницы авторизации

Функциональность:

- Форма входа:
  Используется библиотека antd (Form, Input, Button, message, Card).
  - Поля формы:
    - Электронная почта (с иконкой MailOutlined)
    - Пароль (с иконкой LockOutlined)
  - Валидация: обязательное заполнение полей.
- Обработка отправки (**onFinish**):
  - Отправляет данные на сервер с помощью функции login(email, password).
  - В случае успеха:
    - Устанавливает пользователя в MobX-хранилище через user.setUser(...).
    - Устанавливает флаг аутентификации (user.setIsAuthenticated(true)).
    - Перенаправляет пользователя на /profile.
  - В случае ошибки:
    - Выводит сообщение через messageApi.error(...).
- Навигация:
  - Ссылка на регистрацию, если аккаунта нет: NavLink to="/registration".

Код:

```
import React, {useContext} from 'react';
import {NavLink, useNavigate} from 'react-router-dom';
import {Button, Card, Form, Input, message} from "antd";
import {LockOutlined, MailOutlined} from "@ant-design/icons";
import { Typography } from 'antd';
import {observer} from "mobx-react-lite";
import {Context} from "../main.jsx";
import {login} from "../http/usersAPI.js";
const { Link, Text } = Typography;
```

```
const Auth = observer(() => {

    const [messageApi, contextHolder] = message.useMessage();
    const error = (message) => {
        messageApi.open({
            type: 'Ошибка',
            content: message,
        });
    };

    const {user} = useContext(Context);

    const navigate = useNavigate();

    const onFinish = async (values) => {
        try {
            const data = await login(values.email, values.password);
            user.setUser(data);
            user.setIsAuthenticated(true);
            navigate("/profile");
        } catch (e) {
            messageApi.error(e?.response?.data?.message || "Произошла ошибка при входе");
        }
    }

    return (
        <div className="flex items-center justify-center h-full">
            {contextHolder}
            <Card title="Авторизация" className="w-full max-w-[600px]">
                <Form
                    name="login"
                    initialValues={{ remember: true }}
                    onFinish={onFinish}
                    layout="vertical"
                >
                    <Form.Item
                        name="email"
                        label="Электронная почта"
```

```
      normalize={value => value.trim().replace(/\s+/g, "")}
      rules={[{ required: true, message: 'Пожалуйста, введите электронную почту!' }]}
    >
      <Input
        type="email"
        prefix={<MailOutlined />}
        placeholder="Электронная почта"
        size="large"
      />
    </Form.Item>

    <Form.Item
      name="password"
      label="Пароль"
      normalize={value => value.trim().replace(/\s+/g, "")}
      rules={[
        { required: true, message: 'Пожалуйста, введите пароль!' }
      ]}
    >
      <Input.Password
        prefix={<LockOutlined />}
        placeholder="Пароль"
        size="large"
      />
    </Form.Item>

    <Form.Item>
      <Button
        type="primary"
        htmlType="submit"
        className="w-full bg-blue-600 hover:bg-blue-700"
        size="large"
      >
        Войти
      </Button>
    </Form.Item>

    <Text>Еще нет аккаунта? <NavLink to="/registration">Зарегистрироваться!</NavLink></Text>
  </Form>
```

```
            </Card>
        </div>
    );
});

export default Auth;
```

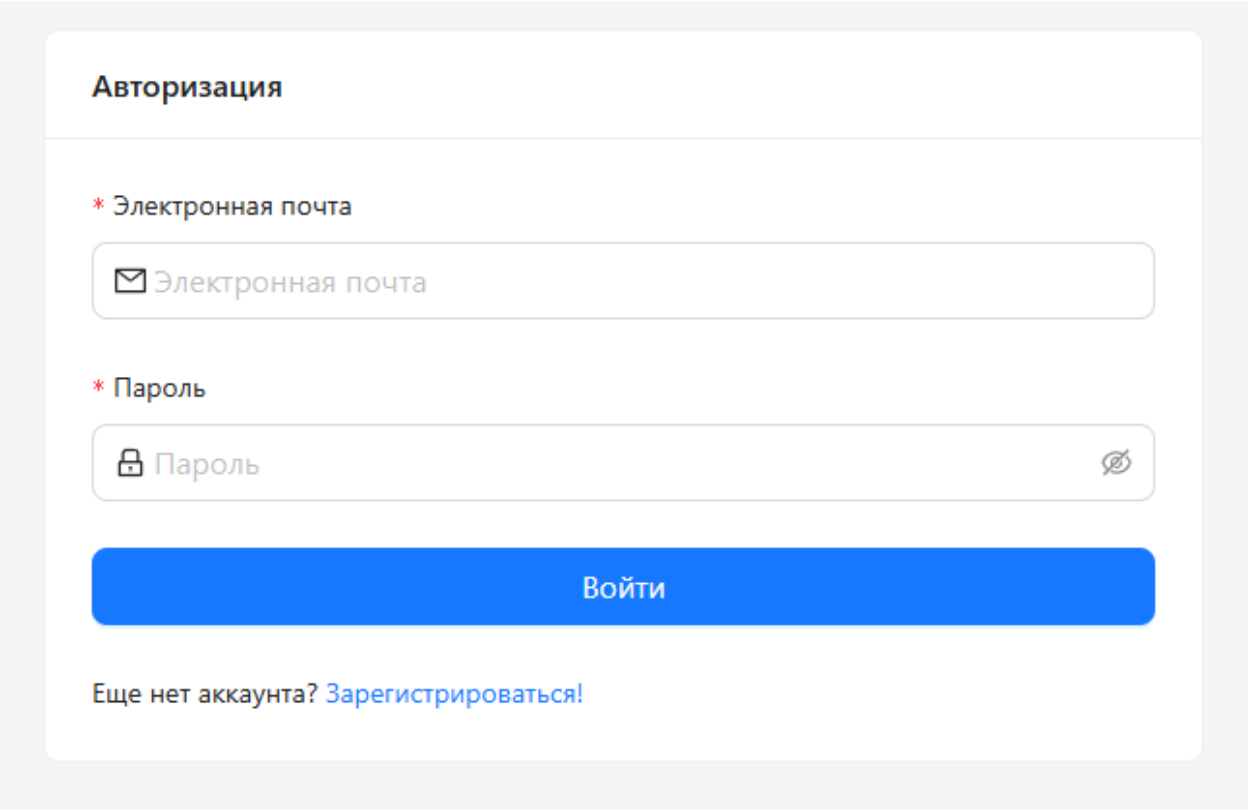

Рисунок 4.3.5 Страница авторизации

4.3.6 Разработка страницы регистрации

Функциональность:

- Форма регистрации включает поля:
  - Электронная почта (email)
  - Фамилия (surname)
  - Имя (name)
  - Отчество (необязательное, patronymic)
  - Серия и номер паспорта (passportSerie, passportNumber)
  - Пароль (password) и подтверждение пароля (passwordRepeat)
- Валидация:
  - Все обязательные поля проверяются на заполненность.
  - Поле пароля проверяется на минимальную длину (от 3 символов).

- Повтор пароля сравнивается с основным паролем.
- Удаление лишних пробелов через normalize.

- Обработка отправки (onFinish):
  - Собранные данные передаются в функцию registartion(...).
  - После успешной регистрации:
    - Устанавливаются данные пользователя в глобальное MobX-хранилище.
    - Выставляется флаг isAuthenticated = true.
- Интерфейс:
  - Используется Ant Design (Form, Input, Card, Row, Col, Button, иконки).
  - Адаптивная сетка: Row и Col для паспортных данных.
  - Ввод серии и номера паспорта реализован через Input.OTP для ограничения длины.
- Навигация:
  - Ссылка на страницу авторизации для уже зарегистрированных пользователей.

Код:

```
import React, {useContext} from 'react';
import {Button, Card, Col, Form, Input, Row} from "antd";
import {UserOutlined, LockOutlined, MailOutlined} from "@ant-design/icons";
import { Typography } from 'antd';
import {registartion} from "../http/usersAPI.js";
import {Context} from "../main.jsx";
import {observer} from "mobx-react-lite";
import {NavLink} from "react-router-dom";
const { Link, Text } = Typography;

const Reg = observer(() => {

    const {user} = useContext(Context);

    const onFinish = async (values) => {
        try {
                const data = await registartion(values.email, values.password, values.name, values.surname, values.patronymic, `${values.passportSerie} ${values.passportNumber}`);
            user.setUser(data);
            user.setIsAuthenticated(true);
        } catch (e) {
```

```
        alert(e.response.data.message);
    }
}

return (
    <div className="flex items-center justify-center h-full">
        <Card title="Регистрация" className="w-full max-w-[600px]">
            <Form
                name="login"
                initialValues={{ remember: true }}
                onFinish={onFinish}
                layout="vertical"
            >
                <Form.Item
                    name="email"
                    label="Электронная почта"
                    normalize={value => value.trim().replace(/\s+/g, "")}
                    rules={[{ required: true, message: 'Пожалуйста, введите электронную почту!' }]}
                >
                    <Input
                        type="email"
                        prefix={<MailOutlined />}
                        placeholder="Электронная почта"
                        size="large"
                    />
                </Form.Item>

                <Form.Item
                    name="surname"
                    label="Фамилия"
                    normalize={value => value.trim().replace(/\s+/g, "")}
                    rules={[{ required: true, message: 'Пожалуйста, введите фамилию!' }]}
                >
                    <Input
                        prefix={<UserOutlined/>}
                        placeholder="Фамилия"
                        size="large"
                    />
                </Form.Item>
```

```
<Form.Item
   name="name"
   label="Имя"
   normalize={value => value.trim().replace(/\s+/g, "")}
   rules={[{ required: true, message: 'Пожалуйста, введите имя!' }]}
>
   <Input
      prefix={<UserOutlined/>}
      placeholder="Имя"
      size="large"
   />
</Form.Item>

<Form.Item
   name="patronymic"
   label="Отчество"
   normalize={value => value.trim().replace(/\s+/g, "")}
   rules={[{ required: false}]}
>
   <Input
      prefix={<UserOutlined/>}
      placeholder="Отчество"
      size="large"
   />
</Form.Item>

<Row gutter={24}>
   <Col span={12} xs={24} sm={12}>
      <Form.Item
         name="passportSerie"
         label="Серия паспорта"
         normalize={value => value.trim().replace(/\s+/g, "")}
         rules={[{ required: true, message: 'Пожалуйста, введите серию паспорта!'}]}
      >
         <Input.OTP
            length={4}
            size="large"
         />
```

```
            </Form.Item>
        </Col>
        <Col span={12} xs={24} sm={12}>
            <Form.Item
                name="passportNumber"
                label="Номер паспорта"
                normalize={value => value.trim().replace(/\s+/g, "")}
                rules={[{ required: true, message: 'Пожалуйста, введите номер паспорта!' }]}
            >
                <Input.OTP
                    length={6}
                    size="large"
                />
            </Form.Item>
        </Col>
    </Row>

    <Form.Item
        name="password"
        label="Пароль"
        normalize={value => value.trim().replace(/\s+/g, "")}
        rules={[
            { required: true, message: 'Пожалуйста, введите пароль!' },
            {type: "string", min: 3, message: "Пароль слишком короткий!"}
        ]}
    >
        <Input.Password
            prefix={<LockOutlined />}
            placeholder="Пароль"
            size="large"
        />
    </Form.Item>

    <Form.Item
        name="passwordRepeat"
        label="Повторите пароль"
        dependencies={['password']}
        normalize={value => value.trim().replace(/\s+/g, "")}
        rules={[{ required: true, message: 'Пароли не совпадают!' },
```

```
                ({ getFieldValue }) => ({
                  validator(_, value) {
                    if (!value || getFieldValue('password') === value) {
                      return Promise.resolve();
                    }
                    return Promise.reject(new Error('Пароли не совпадают!'));
                  },
                }),
              ]}
            >
              <Input.Password
                prefix={<LockOutlined />}
                placeholder="Повторите пароль"
                size="large"
              />
            </Form.Item>

            <Form.Item>
              <Button
                type="primary"
                htmlType="submit"
                className="w-full bg-blue-600 hover:bg-blue-700"
                size="large"
              >
                Зарегистрироваться
              </Button>
            </Form.Item>

            <Text>Уже есть аккаунт? <NavLink to="/authorization">Войти!</NavLink></Text>
          </Form>
        </Card>
      </div>
    );
});

export default Reg;
```

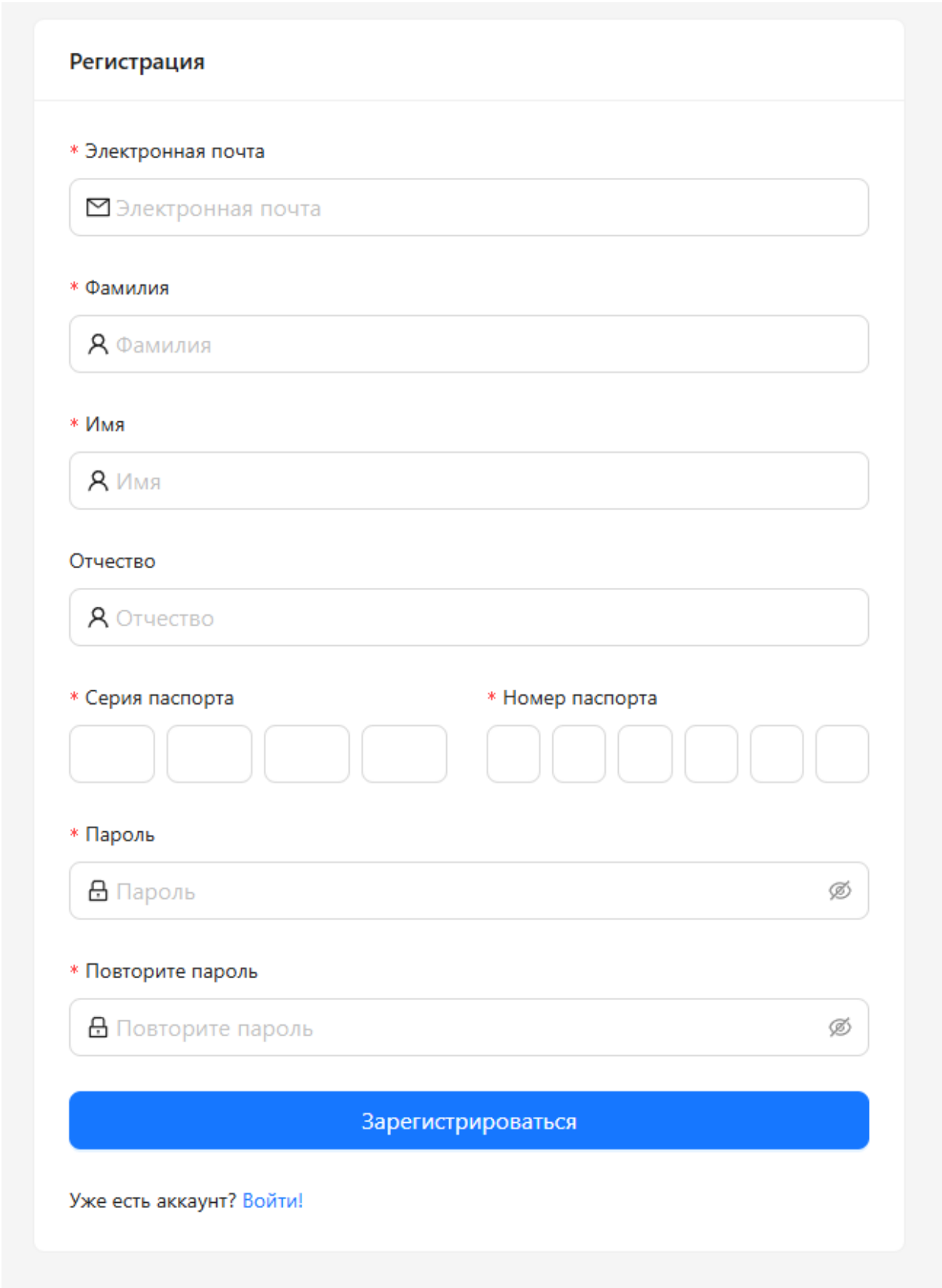

Рисунок 4.3.6 Страница регистрации

4.3.7 Разработка страницы калькулятора стоимости доставки

Функциональность:

- Форма включает поля:
  - Пункт отправки (Select);
  - Пункт назначения (Select);
  - Размер посылки (выбор из предустановленных вариантов или ввод вручную через Popover с Tabs, Radio.Group, InputNumber);
  - Тип доставки (Radio.Group с кнопками);
  - Ценность посылки (InputNumber).
- Данные пунктов и типов доставки загружаются из API (getAllPoints, getAllTypes);
- Валидация полей осуществляется через правила Form.Item (обязательность заполнения);
- Расчёт стоимости происходит при отправке формы (onFinish), с учётом расстояния, веса, объёма и типа доставки;
- Управление формой реализовано через Form.useForm(), обновление полей через form.setFieldValue;
- Отображение результата и ошибок реализовано через компоненты Typography.Title и Alert.

Код:

```
import React, {useEffect, useState} from 'react';
import {
    Button,
    Col,
    Form,
    InputNumber,
    Row,
    Select,
    Typography,
    Radio,
    Card,
    Input,
    Popover,
    Tabs,
    Flex,
    Avatar, Alert
} from 'antd';
import {getAll as getAllPoints} from "../http/deliveryPointsAPI.js";
```

```
import {getAll as getAllTypes}  from "../http/deliveryTypesAPI.js";
const { Title, Text } = Typography;

const sizes = [
    {
        value: 'convert',
        label: (
            <Flex align={'center'} gap={20}>
                <Avatar size={'large'} shape={"square"} src="./public/sizes/convert.svg"/>
                <Flex vertical gap={4} justify={"space-between"}>
                    <Typography.Text>
                        Конверт
                    </Typography.Text>
                    <Typography.Paragraph>
                        34x27x2 см, до 0.5 кг
                    </Typography.Paragraph>
                </Flex>
            </Flex>
        ),
        length: '34',
        width: '27',
        height: '2',
        weight: '0.5',
    },
    {
        value: 'box_xs',
        label: (
            <Flex align={'center'} gap={20}>
                <Avatar size={'large'} shape={"square"} src="./public/sizes/box_xs.svg"/>
                <Flex vertical gap={4} justify={"space-between"}>
                    <Typography.Text>
                        Короб XS
                    </Typography.Text>
                    <Typography.Paragraph>
                        17x12x9 см, до 0.5 кг
                    </Typography.Paragraph>
                </Flex>
            </Flex>
        ),
```

```
        length: '17',
        width: '12',
        height: '9',
        weight: '0.5'
    },
    {
        value: 'box_s',
        label: (
            <Flex align={'center'} gap={20}>
                <Avatar size={'large'} shape={"square"} src="./public/sizes/box_s.svg"/>
                <Flex vertical gap={4} justify={"space-between"}>
                    <Typography.Text>
                        Короб S
                    </Typography.Text>
                    <Typography.Paragraph>
                        23x19x10 см, до 2 кг
                    </Typography.Paragraph>
                </Flex>
            </Flex>
        ),
        length: '23',
        width: '19',
        height: '10',
        weight: '2'
    },
]

const Calculator = () => {

    const [form] = Form.useForm();

    const [open, setOpen] = useState(false);
    const handleOpenChange = (newOpen) => {
        setOpen(newOpen);
    };

    const [rValue, setRValue] = useState(1);
    const onRChange = e => {
        const size = sizes.find(size => size.value == e.target.value);
```

```
        setSizeValue(`Длина: ${size.length} см; Ширина: ${size.width} см; Высота: ${size.height} см; Вес: ${size.weight} кг`);
        setRValue(e.target.value);
        form.setFieldValue("size", size);
    };

    const [sizeValue, setSizeValue] = useState("");
    const onSizeFinish = values => {
        if (values.length === null || values.length === undefined) {
            values.length = 1;
        }
        if (values.width === null || values.width === undefined) {
            values.length = 1;
        }
        if (values.height === null || values.height === undefined) {
            values.length = 1;
        }
        if (values.weight === null || values.weight === undefined) {
            values.length = 1;
        }
        setSizeValue(`Длина: ${values.length} см; Ширина: ${values.width} см; Высота: ${values.height} см; Вес: ${values.weight} кг`);
        form.setFieldValue("size", values);
    };

    function getDistanceFromLatLonInKm(lat1, lon1, lat2, lon2) {
        const toRad = (value) => value * Math.PI / 180;

        const R = 6371; // радиус Земли в км
        const dLat = toRad(lat2 - lat1);
        const dLon = toRad(lon2 - lon1);

        const a =
            Math.sin(dLat / 2) * Math.sin(dLat / 2) +
            Math.cos(toRad(lat1)) * Math.cos(toRad(lat2)) *
            Math.sin(dLon / 2) * Math.sin(dLon / 2);

        const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

        const distance = R * c;
```

```
    return distance;
}

const [errorText, setErrorText] = useState('');
const [deliveryCost, setDeliveryCost] = useState(null);
const onCalculate = values => {
    if (!values.size || !values.deliveryType || !values.from || !values.to) {
        setDeliveryCost(null);
        setErrorText('Заполните все поля!')
        return;
    }

    if (values.from === values.to) {
        setDeliveryCost(null);
        setErrorText('Пункт отправки и пункт доставки должны быть разными!')
        return;
    }

    const length = parseFloat(values.size.length) || 1;
    const width = parseFloat(values.size.width) || 1;
    const height = parseFloat(values.size.height) || 1;
    const weight = parseFloat(values.size.weight) || 1;
    const value = parseFloat(values.value) || 0;
    const deliveryType = values.deliveryType;

    // Получаем точки отправки и назначения
    const fromPoint = deliveryPoints.find(point => point.id === values.from);
    const toPoint = deliveryPoints.find(point => point.id === values.to);

    if (!fromPoint || !toPoint) {
        setDeliveryCost(null);
        return;
    }

    // Расчет расстояния в километрах
    const distanceKm = getDistanceFromLatLonInKm(
        fromPoint.latitude,
        fromPoint.longitude,
```

```
        toPoint.latitude,
        toPoint.longitude
    );

    // Объём в кубических сантиметрах
    const volume = length * width * height;

    // Весовой коэффициент (5000 см³ = 1 кг)
    const volumetricWeight = volume / 5000;

    // Используем максимальный из фактического веса и объёмного
    const billableWeight = Math.max(weight, volumetricWeight);

    // Тарифы доставки (примерные)
    const deliveryTypeRates = {
        1: 100,
        2: 150,
        3: 200
    };

    const baseRate = deliveryTypeRates[deliveryType] || 100;

    // Расчет стоимости с учетом расстояния (например, стоимость увеличивается с километражом)
    // Например, базовая ставка + расстояние * 3 руб * вес * коэффициент
    let cost = baseRate + distanceKm * 3 * billableWeight;

    // Доплата за ценность посылки
    if (value > 0) {
        cost += value * 0.01;
    }

    setDeliveryCost(cost.toFixed(2));
};

const [deliveryTypes, setDeliveryTypes] = useState([]);
useEffect(() => {
    const fetchData = async () => {
        const data = await getAllTypes();
        setDeliveryTypes(data);
```

```
    }
    fetchData()
}, [])

const [deliveryPoints, setDeliveryPoints] = useState([])
useEffect(() => {
    const fetchData = async () => {
        const data = await getAllPoints();
        setDeliveryPoints(data);
    }
    fetchData()
}, []);

return (
    <div className="flex items-center justify-center h-full">
        <Card title="Калькулятор доставки" className="w-full max-w-[1600px]">
            <Form layout="vertical" className="mt-4" onFinish={onCalculate} form={form}>
                <Row gutter={[16, 16]} wrap>
                    <Col xs={24} md={12} lg={8}>
                        <Form.Item label="Точка отправки" name="from"
                                   rules={[
                                       {required: true, message: 'Выберите точку отправки' }
                                   ]}>
                            <Select showSearch placeholder="Выберите точку отправки" options={
                                deliveryPoints.map((point) => (
                                    {
                                        value: point.id,
                                        label: point.address
                                    }
                                ))}
                                    optionFilterProp="label"
                                    filterSort={(optionA, optionB) => {
                                        var _a, _b;
                                        return (
                                            (_a = optionA === null || optionA === void 0 ? void 0 : optionA.label) !== null &&
                                            _a !== void 0
                                                ? _a
                                                : ''
                                        )
```

```
                .toLowerCase()
                .localeCompare(
                    ((_b = optionB === null || optionB === void 0 ? void 0 : optionB.label) !== null
&&
                    _b !== void 0
                      ? _b
                      : ''
                  ).toLowerCase(),
                );
            }}
          />
        </Form.Item>
      </Col>
      <Col xs={24} md={12} lg={8}>
        <Form.Item label="Точка назначения" name="to" rules={[
          {required: true, message: "Выберите точку назначения"}
        ]}>
          <Select showSearch placeholder="Выберите точку назначения" options={
            deliveryPoints.map((point) => (
              {
                value: point.id,
                label: point.address
              }
            ))}
            optionFilterProp="label"
            filterSort={(optionA, optionB) => {
              var _a, _b;
              return (
                (_a = optionA === null || optionA === void 0 ? void 0 : optionA.label) !== null &&
                _a !== void 0
                  ? _a
                  : ''
              )
                .toLowerCase()
                .localeCompare(
                    ((_b = optionB === null || optionB === void 0 ? void 0 : optionB.label) !== null
&&
                    _b !== void 0
                      ? _b
                      : ''
```

```
                    ).toLowerCase(),
                  );
              }}
          />
        </Form.Item>
      </Col>
      <Col xs={24} md={12} lg={8}>
        <Form.Item label="Размер посылки" name="size"
            rules={[
              {required: true, message: 'Введите размер посылки' }
            ]}>
          <Popover open={open}
            onOpenChange={handleOpenChange}
            placement={'bottom'}
            trigger={'click'}
            content={
              <div onClick={(e) => e.stopPropagation()}>
                <Tabs items={
                  [
                    {
                      key: '1',
                      label: "Выбрать",
                      children: (
                        <Radio.Group
                          style={
                            {
                              display: 'flex',
                              flexDirection: 'column',
                              gap: 8
                            }
                          }
                          value={rValue}
                          onChange={onRChange}
                          options={sizes}/>
                      )
                    },
                    {
                      key: '2',
                      label: "Вручную",
```

```
                                children: (
                                  <Form layout={"vertical"} onFinish={onSizeFinish}>
                                    <Row gutter={[16, 16]} className="max-w-[400px]" wrap>
                                      <Col span={12} >
                                              <Form.Item label="Длина" name="length" required={true}
initialValue={1}>
                                          <InputNumber min={1} max={199} suffix={"см"}>

                                          </InputNumber>
                                        </Form.Item>
                                      </Col>
                                      <Col span={12} >
                                              <Form.Item label="Ширина" name="width" required={true}
initialValue={1}>
                                          <InputNumber min={1} max={199} suffix={"см"}>

                                          </InputNumber>
                                        </Form.Item>
                                      </Col>
                                      <Col span={12} >
                                              <Form.Item label="Высота" name="height" required={true}
initialValue={1}>
                                          <InputNumber min={1} max={199} suffix={"см"}>

                                          </InputNumber>
                                        </Form.Item>
                                      </Col>
                                      <Col span={12} >
                                              <Form.Item label="Вес" name="weight" required={true}
initialValue={1}>
                                          <InputNumber min={1} max={199} suffix={"кг"}>

                                          </InputNumber>
                                        </Form.Item>
                                      </Col>
                                      <Col span={24} >
                                        <Form.Item className="mt-6">
                                              <Button type="primary" size="large" className="w-full"
htmlType={"submit"}>
                                            Подтвердить
                                          </Button>
```

```
                                        </Form.Item>
                                    </Col>
                                </Row>
                            </Form>
                        )
                    },
                ]
            }>
            </Tabs>
        </div>
    }>
        <Input placeholder="Выберите размер посылки" className="w-full max-w-[400px]"
value={sizeValue} readOnly>
        </Input>
    </Popover>
</Form.Item>
</Col>
</Row>

<Title level={5} className="mt-4">Тип доставки</Title>
<Form.Item name="deliveryType" className="mt-2" rules={[
    {required: true, message: 'Выбирите тип доставки' }
]}>
    <Radio.Group>
        {
            deliveryTypes.map((deliveryType) => (
                                                    <Radio.Button   key={deliveryType.id}
value={deliveryType.id}>{deliveryType.name}</Radio.Button>
            ))
        }
    </Radio.Group>
</Form.Item>

<Title level={5} className="mt-4">Ценность посылки, руб.</Title>
<Form.Item name="value" className="mt-2" rules={[
    {required: true, message: 'Введите ценность посылки' }
]}>
        <InputNumber min={0} max={100000} style={{width: '100%'}} placeholder="Введите сумму"
className="w-full max-w-[400px]" />
</Form.Item>
```

```
            <Form.Item className="mt-6">
              <Button type="primary" size="large" className="w-[200px]" htmlType={"submit"}>
                Рассчитать
              </Button>
            </Form.Item>
          </Form>
          {/* Показываем стоимость, если она есть */}
          {deliveryCost !== null && (
            <div className="mt-6">
              <Title level={4}>Стоимость доставки: <Text type="success">{deliveryCost} руб.</Text></Title>
            </div>
          )}
          {errorText.length > 0 && (
            <Alert
              message="Ошибка"
              description={errorText}
              type="error"
              closable
            />
          )}
        </Card>
      </div>
    );
};

export default Calculator;
```

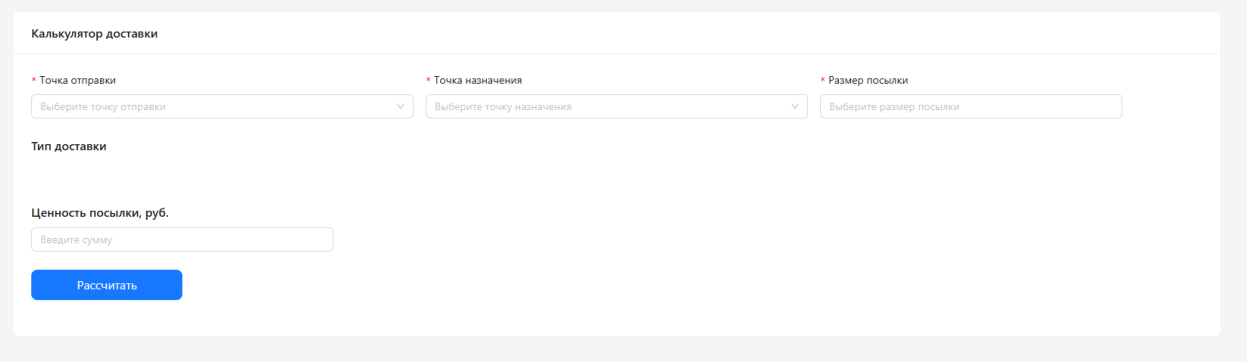

Рисунок 4.3.7 Страница калькулятора стоимости доставки

4.3.8 Разработка страницы личного кабинета клиента

Функциональность:

- Отображение данных пользователя (ФИО, email) с кнопкой выхода.
- Загрузка и отображение заказов пользователя, разделённых на активные и завершённые, с помощью Tabs и List.
- Добавление заказа по трек-коду через Popover с Input и кнопкой.
- При выборе заказа отображается подробное отслеживание в виде Timeline с этапами доставки и карта (ProfileMap).
- Асинхронные запросы к API: получение заказов (getUsersOrdersNotFinished, getUsersOrdersFinished), получение шагов доставки (getOrderDeliveryStepsMessages), добавление заказа (addUserToOrder).
- Визуальные компоненты — из библиотеки Ant Design: Card, Button, Descriptions, Tabs, Timeline, Popover, Input, Typography, Space, Row, Col.
- Управление сообщениями об ошибках через message.useMessage.

Код:

```
import React, {useContext, useEffect, useMemo, useState} from 'react';
import {
    Button,
    Card,
    Col,
    Descriptions, Input,
    List, message, Popover, Radio,
    Row,
    Space, Tabs,
    Timeline,
    Typography
```

```
} from 'antd';
import ProfileMap from '../components/ProfileMap.jsx';
import {observer} from "mobx-react-lite";
import {Context} from "../main.jsx";
import {addUserToOrder, getUsersOrdersNotFinished, getUsersOrdersFinished} from "../http/ordersAPI.js";
import dayjs from "dayjs";
import {getOrderDeliveryStepsMessages} from "../http/orderDeliveryStepsAPI.js";

const Profile = observer(() => {

    const [messageApi, contextHolder] = message.useMessage();
    const error = (message) => {
        messageApi.open({
            type: 'Ошибка',
            content: message,
        });
    };

    const { user } = useContext(Context);

    const userData = user.user;

    const [userOrdersFinished, setUserOrdersFinished] = useState([]);
    const [userOrdersNotFinished, setUserOrdersNotFinished] = useState([]);
    const fetchOrders = async () => {
        const dataNotFinished = await getUsersOrdersNotFinished(user.user.id);
        const dataFinished = await getUsersOrdersFinished(user.user.id);
        setUserOrdersNotFinished(dataNotFinished);
        setUserOrdersFinished(dataFinished);
    }
    useEffect(() => {
        fetchOrders();
    }, []);

    const [selectedOrderId, setSelectedOrderId] = useState(null);
    const [deliverySteps, setDeliverySteps] = useState([]);
    useEffect(() => {
        if (!selectedOrderId) return;
```

```
    const fetchData = async () => {
      const data = await getOrderDeliveryStepsMessages(selectedOrderId);
      setDeliverySteps(data);
    };

    fetchData();
    }, [selectedOrderId]);

  const logOut = () => {
    localStorage.removeItem('token');
    user.setUser({});
    user.setIsAuthenticated(false);
  }

  const [trackCode, setTrackCode] = useState('');
  const addTrackCode = async () => {
    try {
      await addUserToOrder(userData.id, trackCode);
      await fetchOrders();
    } catch (e) {
      messageApi.error(e?.response?.data?.message || "Произошла ошибка при добавлении посылки");
    }
  }

  return (
    <div className="p-4">
      {contextHolder}
      <Row gutter={[16, 16]}>
        {/* Профиль */}
        <Col xs={24} md={24} lg={6}>
          <Card title="Мой профиль" className="h-full">
            <Descriptions layout="vertical" column={1}>
                        <Descriptions.Item label="ФИО">{`${userData.surname} ${userData.name || ''} ${userData.patronymic || ''}`}</Descriptions.Item>
              <Descriptions.Item label="Эл. почта">{userData.email}</Descriptions.Item>
              <Descriptions.Item>
                <Button type="primary" danger block onClick={logOut}>
                  Выйти
                </Button>
```

```
                </Descriptions.Item>
              </Descriptions>
            </Card>
          </Col>

          {/* Заказы */}
          <Col xs={24} md={24} lg={18} xl={18}>
            <Card
              title={
                <div className="flex flex-col sm:flex-row sm:justify-between sm:items-center gap-2">
                  <Typography.Text strong>Отслеживаемые заказы</Typography.Text>
                  <Popover placement="bottom" title="Добавление заказа" content={(
                    <div>
                      <Space>
                                                <Input placeholder="Трек-код" value={trackCode} onChange={(e) =>
setTrackCode(e.target.value)} />
                        <Button onClick={() => {addTrackCode()}}>
                          Добавить
                        </Button>
                      </Space>
                    </div>
                  )}>
                    <Button type="primary">Добавить заказ</Button>
                  </Popover>
                </div>
              }
              className="h-[45vh] overflow-auto"
            >
              <Tabs items={[
                {
                  key: '1',
                  label: "Активные",
                  children: (
                    <List
                      dataSource={userOrdersNotFinished}
                      renderItem={(item, index) => (
                        <List.Item key={index}>
                          <div className="flex justify-between w-full items-center">
                            <Space direction="vertical" align="start">
```

```
                                        <Typography.Text className="text-base">Заказ
№{item.id}</Typography.Text>
                                        <Typography.Text className="text-base">Трек:
{item.track_code}</Typography.Text>
                                        <Typography.Text type="secondary">От:
{dayjs(item.createdAt).format('DD.MM.YYYY HH:mm')}</Typography.Text>
                                    </Space>
                                        <Button onClick={() => setSelectedOrderId(item.id)}
size="small">Подробнее</Button>
                                </div>
                            </List.Item>
                        )}
                    />
                )
            },
            {
                key: '2',
                label: "Завершенные",
                children: (
                    <List
                        dataSource={userOrdersFinished}
                        renderItem={(item, index) => (
                            <List.Item key={index}>
                                <div className="flex justify-between w-full items-center">
                                    <Space direction="vertical" align="start">
                                        <Typography.Text className="text-base">Заказ
№{item.id}</Typography.Text>
                                        <Typography.Text className="text-base">Трек:
{item.track_code}</Typography.Text>
                                        <Typography.Text type="secondary">От:
{dayjs(item.createdAt).format('DD.MM.YYYY HH:mm')}</Typography.Text>
                                    </Space>
                                        <Button onClick={() => setSelectedOrderId(item.id)}
size="small">Подробнее</Button>
                                </div>
                            </List.Item>
                        )}
                    />
                )
            },
        ]}>
```

```
            </Tabs>
          </Card>
        </Col>

        {/* Отслеживание */}
        {selectedOrderId !== null &&
        <Col xs={24} md={24} lg={24} xl={24}>
          <Card title="Отслеживание" className="h-[44vh] overflow-auto" >
            <Row gutter={[16, 16]} className="h-full">
              <Col xs={24} md={12}>
                <Timeline
                  items={deliverySteps.map(step => ({
                    children: (
                      <div>
                        <Typography.Text strong>{step.message}</Typography.Text>
                        <br />
                        <Typography.Text type="secondary">{step.address}</Typography.Text>
                        <br />
                          <Typography.Text type="secondary">{dayjs(step.datetime).format('DD.MM.YYYY
HH:mm')}</Typography.Text>
                      </div>
                    )
                  }))}
                />
              </Col>
              <Col xs={24} md={12}>
                <div className="h-full w-full">
                  <ProfileMap orderId={selectedOrderId} />
                </div>
              </Col>
            </Row>
          </Card>
        </Col>
        }
      </Row>
    </div>
  );
});
```

export default Profile;

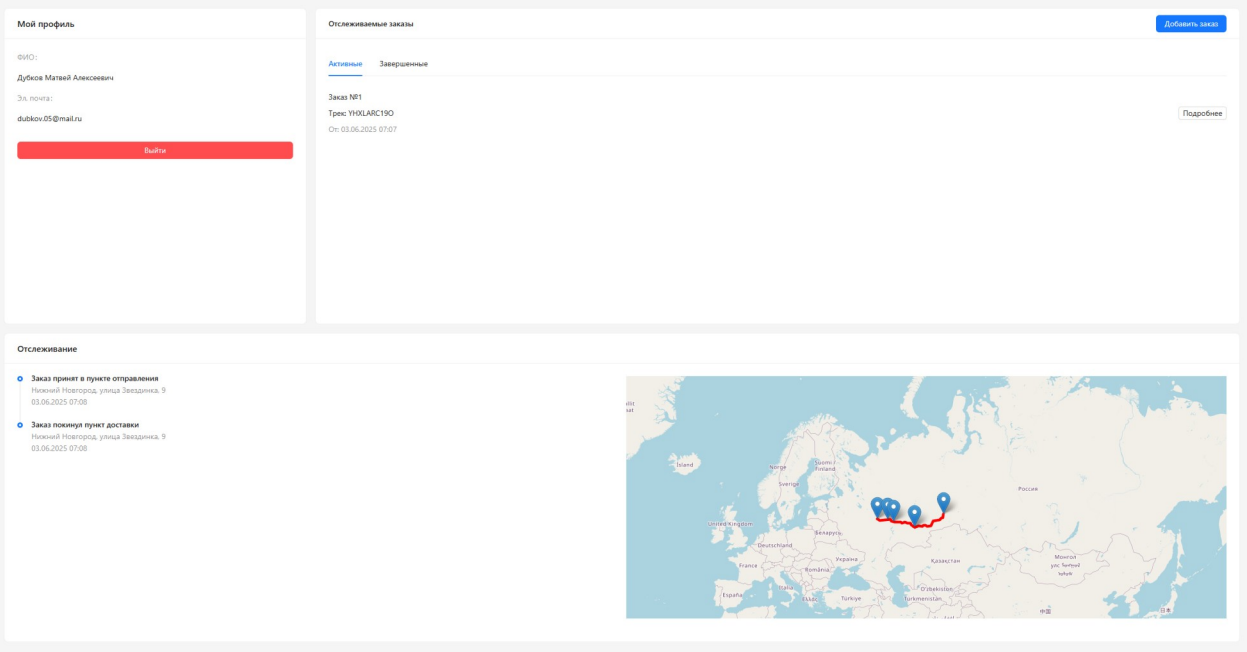

Рисунок 4.3.8 Страница личного кабинета клиента

4.3.9 Разработка страницы личного кабинета управляющего

Функциональность:

- Отображение профиля пользователя с возможностью выхода.
- Получение и отображение двух списков заказов: ожидаемых (для приёма) и текущих (готовых к отправке) в таблицах (Table).
- Форматирование данных отправителя и габаритов посылок перед отображением.
- Поиск по всем полям таблиц с фильтрацией через ввод текста в поле поиска (Input.Search).
- В таблицах отображаются колонки: трек-код, данные отправителя, паспортные данные получателя, габариты, цена и действие.
- Кнопки в действиях:
  - «Принять посылку» — вызывает API setOrderArrived, обновляет данные.
  - «Отправить посылку» — вызывает API setOrderLeft, обновляет данные.
- Используются компоненты Ant Design: Row, Col, Card, Descriptions, Button, Input.Search, Table, Space.
- Асинхронные запросы к API для получения данных и изменения статусов заказов.

Код:

```
import React, {useContext, useEffect, useState} from 'react';
import {Button, Card, Col, Descriptions, Row, Space, Table, Input} from "antd";
import {Context} from "../main.jsx";
import {observer} from "mobx-react-lite";
import {getCurrentOrders, getExpectedOrders, setOrderArrived, setOrderLeft} from "../http/ordersAPI.js";

const Manager = observer(() => {
    const {user} = useContext(Context);
    const userData = user.user;

    const logOut = () => {
        localStorage.removeItem('token');
        user.setUser({});
        user.setIsAuthenticated(false);
    };

    const [expectedOrdersData, setExpectedOrdersData] = useState([]);
    const [currentOrdersData, setCurrentOrdersData] = useState([]);

    const fetchData = async () => {
        const data1 = await getExpectedOrders(userData.id);
        data1.map(data => {
                    data.senderData = `ФИО: ${data.senderData.surname} ${data.senderData.name || ""} ${data.senderData.patronymic || ""};\n email: ${data.senderData.email};\n Паспортные данные: ${data.senderData.passport}`;
            data.sizes = `Д: ${data.length}; Ш: ${data.width}; В: ${data.height}; Вес: ${data.weight}`;
        });
        setExpectedOrdersData(data1);

        const data2 = await getCurrentOrders(userData.id);
        data2.map(data => {
                    data.senderData = `ФИО: ${data.senderData.surname} ${data.senderData.name || ""} ${data.senderData.patronymic || ""};\n email: ${data.senderData.email};\n Паспортные данные: ${data.senderData.passport}`;
            data.sizes = `Д: ${data.length}; Ш: ${data.width}; В: ${data.height}; Вес: ${data.weight}`;
        });
        setCurrentOrdersData(data2);
    };
```

```
useEffect(() => {
  fetchData();
}, []);

const [searchText, setSearchText] = useState("");

const handleSearch = (value) => {
  setSearchText(value.toLowerCase());
};

// Фильтрация
const filteredExpected = expectedOrdersData.filter(order =>
  JSON.stringify(order).toLowerCase().includes(searchText)
);

const filteredCurrent = currentOrdersData.filter(order =>
  JSON.stringify(order).toLowerCase().includes(searchText)
);

const columnsExpected = [
  {
    title: 'Трек',
    dataIndex: 'track_code',
    key: 'track_code',
    render: text => <div>{text}</div>,
  },
  {
    title: 'Данные отправителя',
    dataIndex: 'senderData',
    key: 'senderData',
    render: text => <div>{text}</div>,
  },
  {
    title: 'Паспортные данные получателя',
    dataIndex: 'receiver_passport',
    key: 'receiver_passport',
    render: text => <div>{text}</div>,
  },
  {
```

```
        title: 'Габариты',
        dataIndex: 'sizes',
        key: 'sizes',
        render: text => <div>{text}</div>,
    },
    {
        title: 'Цена',
        dataIndex: 'cost',
        key: 'cost',
        render: text => <div>{text} руб.</div>,
    },
    {
        title: 'Действие',
        key: 'action',
        render: (_, record) => (
            <Space size="middle">
                <Button onClick={async () => {
                    await setOrderArrived(userData.id, record.id);
                    await fetchData();
                }}>
                    Принять посылку
                </Button>
            </Space>
        ),
    },
];

const columnsCurrent = [
    {
        title: 'Трек',
        dataIndex: 'track_code',
        key: 'track_code',
        render: text => <div>{text}</div>,
    },
    {
        title: 'Данные отправителя',
        dataIndex: 'senderData',
        key: 'senderData',
        render: text => <div>{text}</div>,
```

```
    },
    {
      title: 'Паспортные данные получателя',
      dataIndex: 'receiver_passport',
      key: 'receiver_passport',
      render: text => <div>{text}</div>,
    },
    {
      title: 'Габариты',
      dataIndex: 'sizes',
      key: 'sizes',
      render: text => <div>{text}</div>,
    },
    {
      title: 'Цена',
      dataIndex: 'cost',
      key: 'cost',
      render: text => <div>{text} руб.</div>,
    },
    {
      title: 'Действие',
      key: 'action',
      render: (_, record) => (
        <Space size="middle">
          <Button onClick={async () => {
            await setOrderLeft(userData.id, record.id);
            await fetchData();
          }}>
            Отправить посылку
          </Button>
        </Space>
      ),
    },
  ];

  return (
    <Row gutter={[16, 16]}>
      <Col span={24}>
        <Card>
```

```
                <Input.Search
                  placeholder="Поиск по всем полям..."
                  allowClear
                  enterButton="Поиск"
                  size="middle"
                  onSearch={handleSearch}
                  onChange={e => handleSearch(e.target.value)}
                />
              </Card>
            </Col>
            <Col xs={24} md={24} lg={6}>
              <Card title="Мой профиль" className="h-full">
                <Descriptions layout="vertical" column={1}>
                                <Descriptions.Item label="ФИО">{`${userData.surname} ${userData.name || ''} ${userData.patronymic || ''}`}</Descriptions.Item>
                  <Descriptions.Item label="Эл. почта">{userData.email}</Descriptions.Item>
                  <Descriptions.Item>
                    <Button type="primary" danger block onClick={logOut}>
                      Выйти
                    </Button>
                  </Descriptions.Item>
                </Descriptions>
              </Card>
            </Col>
            <Col xs={24} md={24} lg={18}>
              <Card title="Ожидаемые посылки" className="h-full">
                <Table columns={columnsExpected} dataSource={filteredExpected} rowKey="id"/>
              </Card>
            </Col>
            <Col xs={24} md={24} lg={24}>
              <Card title="Посылки, ожидающие отправки" className="h-full">
                <Table columns={columnsCurrent} dataSource={filteredCurrent} rowKey="id"/>
              </Card>
            </Col>
          </Row>
      );
});

export default Manager;
```

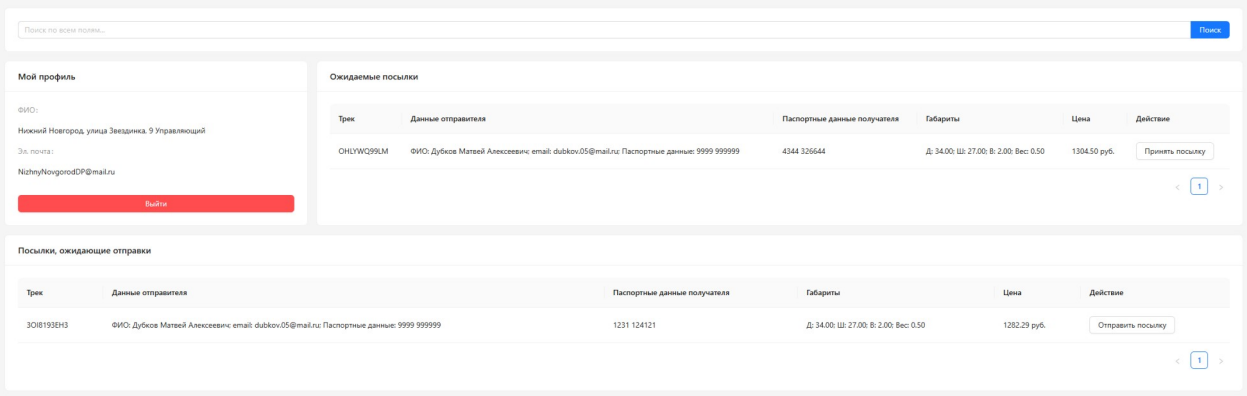

Рисунок 4.3.9 Страница личного кабинета управляющего

4.3.10 Разработка страницы личного кабинета администратора

Функциональность:

- Отображение профиля пользователя с возможностью выхода из системы.
- Получение и отображение статистических данных из различных источников:
  - Эффективность работы пунктов сортировки (среднее время обработки) — отображается в виде вертикальной гистограммы (BarChart).
  - Распределение посылок по типам доставки — отображается круговой диаграммой (PieChart).
  - Распределение пользователей по ролям — отображается круговой диаграммой (PieChart).
  - Динамика количества заказов за последние дни — отображается линейным графиком (LineChart).
- Использование цветовой схемы для наглядности (цвета выбираютс
- зависимости от значений или ролей).
- Применение компонентов Ant Design для структурирования интерфейса: Row, Col, Card, Descriptions, Button, Space.
- Использование библиотеки recharts для визуализации данных в различных типах графиков и диаграмм.
- Асинхронные запросы к API обеспечивают получение актуальных статистических данных.
- Логика цветового оформления столбцов в гистограмме зависит от значения среднего времени обработки.
- Кнопка выхода очищает токен в localStorage и обновляет состояние аутентификации в контексте.

Код:

```
import React, {useContext, useEffect, useState} from 'react';
import { Button, Card, Col, Descriptions, Row, Space } from "antd";
import { observer } from "mobx-react-lite";
import { Context } from "../main.jsx";
import {
    BarChart, Bar, XAxis, YAxis, Tooltip, ResponsiveContainer, Cell, PieChart, Pie, Legend, LineChart, Line
} from 'recharts';
import {getOrderDeliveryStepsMessages} from "../http/orderDeliveryStepsAPI.js";
import {getDeliveryPointsStats} from "../http/deliveryPointsAPI.js";
import {getDeliveryTypesStats, getOrdersStats} from "../http/ordersAPI.js";
import {getUsersRolesStats} from "../http/usersAPI.js";

const Admin = observer(() => {
    const { user } = useContext(Context);
    const userData = user.user;

    const [deliveryPointsStats, setDeliveryPointsStats] = useState([]);
    const [deliveryTypesStats, setDeliveryTypesStats] = useState([]);
    const [usersRolesStats, setUsersRolesStats] = useState([]);
    const [ordersStats, setOrdersStats] = useState([]);

    useEffect(() => {
        const fetchData = async () => {
            const dataDeliveryPointsStats = await getDeliveryPointsStats();
            setDeliveryPointsStats(dataDeliveryPointsStats);
            const dataDeliveryTypesStats = await getDeliveryTypesStats();
            setDeliveryTypesStats(dataDeliveryTypesStats);
            const usersRolesStats = await getUsersRolesStats();
            setUsersRolesStats(usersRolesStats);
            const ordersStats = await getOrdersStats();
            setOrdersStats(ordersStats);
        };

        fetchData();
        }, [])
```

```
console.log(deliveryTypesStats);


const logOut = () => {
    localStorage.removeItem('token');
    user.setUser({});
    user.setIsAuthenticated(false);
};

// Цвета для графиков
const deliveryTypeColors = ['#1890ff', '#52c41a', '#faad14'];
const userRoleColors = ['#ff4d4f', '#13c2c2', '#722ed1'];


const getBarColor = (time) => {
    if (time > 60) return '#ff4d4f';
    if (time > 40) return '#faad14';
    return '#52c41a';
};

return (
    <Space className="w-full" direction="vertical">
      <Row gutter={[16, 16]}>
        {/* Профиль */}
        <Col xs={24} md={24} lg={6}>
          <Card title="Мой профиль" className="h-full">
            <Descriptions layout="vertical" column={1}>
              <Descriptions.Item label="ФИО">
                {`${userData.surname} ${userData.name || ''} ${userData.patronymic || ''}`}
              </Descriptions.Item>
              <Descriptions.Item label="Эл. почта">{userData.email}</Descriptions.Item>
              <Descriptions.Item>
                <Button type="primary" danger block onClick={logOut}>
                  Выйти
                </Button>
              </Descriptions.Item>
            </Descriptions>
          </Card>
        </Col>
```

```
            {/* График 1: Эффективность пунктов */}
            <Col xs={24} lg={18}>
              <Card title="Эффективность пунктов сортировки (в минутах)">
                <ResponsiveContainer width="100%" height={1080}>
                  <BarChart layout="vertical" data={deliveryPointsStats}>
                        <XAxis type="number" label={{ value: 'Минуты', position: 'insideBottomRight', offset:
-5 }} />
                    <YAxis type="category" dataKey="address" width={200} />
                    <Tooltip />
                    <Bar dataKey="avgTime">
                      {deliveryPointsStats.map((entry, index) => (
                        <Cell key={`cell-${index}`} fill={getBarColor(entry.avgTime)} />
                      ))}
                    </Bar>
                  </BarChart>
                </ResponsiveContainer>
              </Card>
            </Col>
          </Row>

          <Row gutter={[16, 16]}>
            {/* График 2: Заказы по типу */}
            <Col xs={24} md={12}>
              <Card title="Распределение посылок по типу доставки">
                <ResponsiveContainer width="100%" height={300}>
                  <PieChart>
                        <Pie data={deliveryTypesStats} dataKey="orders_count" nameKey="name" cx="50%"
cy="50%" outerRadius={100} label>
                      {deliveryTypesStats.map((entry, index) => (
                                <Cell key={`cell-orders-${index}`} fill={deliveryTypeColors[index %
deliveryTypeColors.length]} />
                      ))}
                    </Pie>
                    <Tooltip />
                    <Legend />
                  </PieChart>
                </ResponsiveContainer>
              </Card>
            </Col>
```

```
            {/* График 3: Пользователи по ролям */}
            <Col xs={24} md={12}>
              <Card title="Распределение пользователей по ролям">
                <ResponsiveContainer width="100%" height={300}>
                  <PieChart>
                    <Pie data={usersRolesStats} dataKey="user_count" nameKey="name" cx="50%" cy="50%" outerRadius={100} label>
                      {usersRolesStats.map((entry, index) => (
                        <Cell key={`cell-users-${index}`} fill={userRoleColors[index % userRoleColors.length]} />
                      ))}
                    </Pie>
                    <Tooltip />
                    <Legend />
                  </PieChart>
                </ResponsiveContainer>
              </Card>
            </Col>
          </Row>

          <Row gutter={[16, 16]}>
            {/* График 4: Заказы по дням */}
            <Col xs={24}>
              <Card title="Динамика заказов за последние дни">
                <ResponsiveContainer width="100%" height={300}>
                  <LineChart data={ordersStats}>
                    <XAxis dataKey="order_date" />
                    <YAxis />
                    <Tooltip />
                    <Line type="monotone" dataKey="orders_count" stroke="#722ed1" strokeWidth={3} />
                  </LineChart>
                </ResponsiveContainer>
              </Card>
            </Col>
          </Row>
        </Space>
    );
});
```

export default Admin;

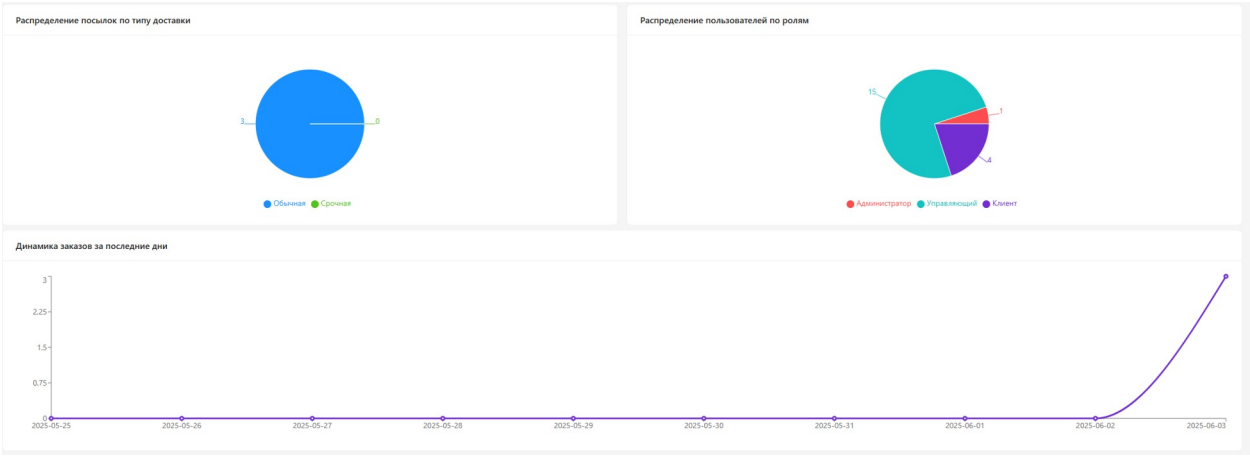

Рисунок 4.3.10 Страница личного кабинета администратора

4.3.11 Разработка страницы создания новой посылки

Функциональность:

- Форма включает поля:
  - Пункт отправки (Select);
  - Пункт назначения (Select);
  - Размер посылки (выбор из предустановленных вариантов или ввод вручную через Popover с Tabs, Radio.Group, InputNumber);
  - Тип доставки (Radio.Group с кнопками);
  - Ценность посылки (InputNumber).
- Данные пунктов и типов доставки загружаются из API (getAllPoints, getAllTypes);
- Валидация полей осуществляется через правила Form.Item (обязательность заполнения);
- Расчёт стоимости происходит при отправке формы (onFinish), с учётом расстояния, веса, объёма и типа доставки;
- Управление формой реализовано через Form.useForm(), обновление полей через form.setFieldValue;
- Отображение результата и ошибок реализовано через компоненты Typography.Title и Alert.

Код:

```
import React, {useContext, useEffect, useState} from 'react';
import {
    Alert,
    Avatar,
```

```
    Button,
    Card,
    Col,
    Flex,
    Form,
    Input,
    InputNumber, Popconfirm,
    Popover,
    Radio,
    Row,
    Select,
    Tabs, Typography
} from "antd";
import {getAll as getAllTypes} from "../http/deliveryTypesAPI.js";
import {getAll as getAllPoints} from "../http/deliveryPointsAPI.js";
import {createOrder as create} from "../http/ordersAPI.js";
import {Context} from "../main.jsx";
import {observer} from "mobx-react-lite";
const  {Title} = Typography;

const sizes = [
    {
        value: 'convert',
        label: (
            <Flex align={'center'} gap={20}>
                <Avatar size={'large'} shape={"square"} src="./public/sizes/convert.svg"/>
                <Flex vertical gap={4} justify={"space-between"}>
                    <Typography.Text>
                        Конверт
                    </Typography.Text>
                    <Typography.Paragraph>
                        34x27x2 см, до 0.5 кг
                    </Typography.Paragraph>
                </Flex>
            </Flex>
        ),
        length: '34',
        width: '27',
        height: '2',
```

```
    weight: '0.5',
  },
  {
    value: 'box_xs',
    label: (
      <Flex align={'center'} gap={20}>
        <Avatar size={'large'} shape={"square"} src="./public/sizes/box_xs.svg"/>
        <Flex vertical gap={4} justify={"space-between"}>
          <Typography.Text>
            Короб XS
          </Typography.Text>
          <Typography.Paragraph>
            17x12x9 см, до 0.5 кг
          </Typography.Paragraph>
        </Flex>
      </Flex>
    ),
    length: '17',
    width: '12',
    height: '9',
    weight: '0.5'
  },
  {
    value: 'box_s',
    label: (
      <Flex align={'center'} gap={20}>
        <Avatar size={'large'} shape={"square"} src="./public/sizes/box_s.svg"/>
        <Flex vertical gap={4} justify={"space-between"}>
          <Typography.Text>
            Короб S
          </Typography.Text>
          <Typography.Paragraph>
            23x19x10 см, до 2 кг
          </Typography.Paragraph>
        </Flex>
      </Flex>
    ),
    length: '23',
    width: '19',
```

```
        height: '10',
        weight: '2'
    },
]

const CreateOrder = observer(() => {

    const {user} = useContext(Context);

    const [form] = Form.useForm();

    const [open, setOpen] = useState(false);
    const handleOpenChange = (newOpen) => {
        setOpen(newOpen);
    };

    const [rValue, setRValue] = useState(1);
    const onRChange = e => {
        const size = sizes.find(size => size.value == e.target.value);
            setSizeValue(`Длина: ${size.length} см; Ширина: ${size.width} см; Высота: ${size.height} см; Вес: ${size.weight} кг`);
        setRValue(e.target.value);
        form.setFieldValue("size", size);
    };

    const [sizeValue, setSizeValue] = useState("");
    const onSizeFinish = values => {
        if (values.length === null || values.length === undefined) {
            values.length = 1;
        }
        if (values.width === null || values.width === undefined) {
            values.length = 1;
        }
        if (values.height === null || values.height === undefined) {
            values.length = 1;
        }
        if (values.weight === null || values.weight === undefined) {
            values.length = 1;
        }
```

```
        setSizeValue(`Длина: ${values.length} см; Ширина: ${values.width} см; Высота: ${values.height} см; Вес: ${values.weight} кг`);
        form.setFieldValue("size", values);
    };

    function getDistanceFromLatLonInKm(lat1, lon1, lat2, lon2) {
        const toRad = (value) => value * Math.PI / 180;

        const R = 6371; // радиус Земли в км
        const dLat = toRad(lat2 - lat1);
        const dLon = toRad(lon2 - lon1);

        const a =
            Math.sin(dLat / 2) * Math.sin(dLat / 2) +
            Math.cos(toRad(lat1)) * Math.cos(toRad(lat2)) *
            Math.sin(dLon / 2) * Math.sin(dLon / 2);

        const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

        const distance = R * c;

        return distance;
    }

    const [error, setError] = useState({});
    const [deliveryCost, setDeliveryCost] = useState(null);
    const onCalculate = values => {
        if (!values.size || !values.deliveryType || !values.from || !values.to) {
            setDeliveryCost(null);
            setError({type: "error", message: 'Заполните все поля!'})
            return;
        }

        if (values.from === values.to) {
            setDeliveryCost(null);
            setError({type: "error", message: 'Пункт отправки и пункт доставки должны быть разными!'})
            return;
        }
```

```
const length = parseFloat(values.size.length) || 1;
const width = parseFloat(values.size.width) || 1;
const height = parseFloat(values.size.height) || 1;
const weight = parseFloat(values.size.weight) || 1;
const value = parseFloat(values.value) || 0;
const deliveryType = values.deliveryType;

// Получаем точки отправки и назначения
const fromPoint = deliveryPoints.find(point => point.id === values.from);
const toPoint = deliveryPoints.find(point => point.id === values.to);

if (!fromPoint || !toPoint) {
    setDeliveryCost(null);
    return;
}

// Расчет расстояния в километрах
const distanceKm = getDistanceFromLatLonInKm(
    fromPoint.latitude,
    fromPoint.longitude,
    toPoint.latitude,
    toPoint.longitude
);

// Объём в кубических сантиметрах
const volume = length * width * height;

// Весовой коэффициент (5000 см³ = 1 кг)
const volumetricWeight = volume / 5000;

// Используем максимальный из фактического веса и объёмного
const billableWeight = Math.max(weight, volumetricWeight);

// Тарифы доставки (примерные)
const deliveryTypeRates = {
    1: 100,
    2: 150,
    3: 200
};
```

```
    const baseRate = deliveryTypeRates[deliveryType] || 100;

    // Расчет стоимости с учетом расстояния (например, стоимость увеличивается с километражом)
    // Например, базовая ставка + расстояние * 3 руб * вес * коэффициент
    let cost = baseRate + distanceKm * 3 * billableWeight;

    // Доплата за ценность посылки
    if (value > 0) {
        cost += value * 0.01;
    }

    setDeliveryCost(cost.toFixed(2));
};

const [deliveryTypes, setDeliveryTypes] = useState([]);
useEffect(() => {
    const fetchData = async () => {
        const data = await getAllTypes();
        setDeliveryTypes(data);
    }
    fetchData()
}, [])

const [deliveryPoints, setDeliveryPoints] = useState([])
useEffect(() => {
    const fetchData = async () => {
        const data = await getAllPoints();
        setDeliveryPoints(data);
    }
    fetchData()
}, []);

const createOrder = async (values) => {
    if (values.from === values.to) {
        setDeliveryCost(null);
        setError({type: "error", message: 'Пункт отправки и пункт доставки должны быть разными!'})
        return;
    }
```

```
        const length = parseFloat(values.size.length) || 1;
        const width = parseFloat(values.size.width) || 1;
        const height = parseFloat(values.size.height) || 1;
        const weight = parseFloat(values.size.weight) || 1;
        const value = parseFloat(values.value) || 0;
        const deliveryType = values.deliveryType;
          const createOrder = await create(user.user.id, values.from, values.to, length, width, height, weight, value, deliveryType, `${values.passportSerie} ${values.passportNumber}`, deliveryCost);
         setError({type: "success", message: `Заказ успешно создан, код отслеживания: ${createOrder.track_code} `});
    }


    return (
        <div className="flex items-center justify-center h-full">
            <Card title="Оформление посылки" className="w-full max-w-[1600px]">
                <Form layout="vertical" className="mt-4" onFinish={createOrder} form={form}>
                    <Row gutter={[16, 16]} wrap>
                        <Col xs={24} md={12} lg={8}>
                            <Form.Item label="Точка отправки" name="from"
                                rules={[
                                    {required: true, message: 'Выберите точку отправки' }
                                ]}>
                                <Select showSearch placeholder="Выберите точку отправки" options={
                                    deliveryPoints.map((point) => (
                                        {
                                            value: point.id,
                                            label: point.address
                                        }
                                    ))}
                                    optionFilterProp="label"
                                    filterSort={(optionA, optionB) => {
                                        var _a, _b;
                                        return (
                                            (_a = optionA === null || optionA === void 0 ? void 0 : optionA.label) !== null &&
                                            _a !== void 0
                                                ? _a
                                                : ''
                                        )
                                            .toLowerCase()
```

```
                        .localeCompare(
                            ((_b = optionB === null || optionB === void 0 ? void 0 : optionB.label) !== null
&&
                            _b !== void 0
                              ? _b
                              : ''
                          ).toLowerCase(),
                        );
                    }}
                />
            </Form.Item>
        </Col>
        <Col xs={24} md={12} lg={8}>
            <Form.Item label="Точка назначения" name="to" rules={[
                {required: true, message: "Выберите точку назначения"}
            ]}>
                <Select showSearch placeholder="Выберите точку назначения" options={
                    deliveryPoints.map((point) => (
                        {
                            value: point.id,
                            label: point.address
                        }
                    ))}
                    optionFilterProp="label"
                    filterSort={(optionA, optionB) => {
                        var _a, _b;
                        return (
                            (_a = optionA === null || optionA === void 0 ? void 0 : optionA.label) !== null &&
                            _a !== void 0
                              ? _a
                              : ''
                        )
                            .toLowerCase()
                            .localeCompare(
                                ((_b = optionB === null || optionB === void 0 ? void 0 : optionB.label) !== null
&&
                                _b !== void 0
                                  ? _b
                                  : ''
                              ).toLowerCase(),
```

```
                    );
                }}
            />
        </Form.Item>
    </Col>
    <Col xs={24} md={12} lg={8}>
        <Form.Item label="Размер посылки" name="size"
        rules={[
            {required: true, message: 'Введите размер посылки' }
        ]}>
            <Popover open={open}
                onOpenChange={handleOpenChange}
                placement={'bottom'}
                trigger={'click'}
                content={
                    <div onClick={(e) => e.stopPropagation()}>
                        <Tabs items={
                            [
                                {
                                    key: '1',
                                    label: "Выбрать",
                                    children: (
                                        <Radio.Group
                                            style={
                                                {
                                                    display: 'flex',
                                                    flexDirection: 'column',
                                                    gap: 8
                                                }
                                            }
                                            value={rValue}
                                            onChange={onRChange}
                                            options={sizes}/>
                                    )
                                },
                                {
                                    key: '2',
                                    label: "Вручную",
                                    children: (
```

```
                    <Form layout={"vertical"} onFinish={onSizeFinish}>
                      <Row gutter={[16, 16]} className="max-w-[400px]" wrap>
                        <Col span={12} >
                                <Form.Item label="Длина" name="length" required={true}
initialValue={1}>
                            <InputNumber min={1} max={199} suffix={"см"}>

                            </InputNumber>
                          </Form.Item>
                        </Col>
                        <Col span={12} >
                                <Form.Item label="Ширина" name="width" required={true}
initialValue={1}>
                            <InputNumber min={1} max={199} suffix={"см"}>

                            </InputNumber>
                          </Form.Item>
                        </Col>
                        <Col span={12} >
                                <Form.Item label="Высота" name="height" required={true}
initialValue={1}>
                            <InputNumber min={1} max={199} suffix={"см"}>

                            </InputNumber>
                          </Form.Item>
                        </Col>
                        <Col span={12} >
                                <Form.Item label="Вес" name="weight" required={true}
initialValue={1}>
                            <InputNumber min={1} max={199} suffix={"кг"}>

                            </InputNumber>
                          </Form.Item>
                        </Col>
                        <Col span={24} >
                          <Form.Item className="mt-6">
                                <Button type="primary" size="large" className="w-full"
htmlType={"submit"}>
                              Подтвердить
                            </Button>
                          </Form.Item>
```

```
                                    </Col>
                                </Row>
                            </Form>
                        )
                    },
                ]
            }>
            </Tabs>
        </div>
    }>
        <Input placeholder="Выберите размер посылки" className="w-full max-w-[400px]" value={sizeValue} readOnly>
        </Input>
    </Popover>
</Form.Item>
</Col>
</Row>

<Title level={5} className="mt-4">Тип доставки</Title>
<Form.Item name="deliveryType" className="mt-2" rules={[
  {required: true, message: 'Выбирите тип доставки' }
]}>
  <Radio.Group>
    {
      deliveryTypes.map((deliveryType) => (
                                    <Radio.Button key={deliveryType.id} value={deliveryType.id}>{deliveryType.name}</Radio.Button>
      ))
    }
  </Radio.Group>
</Form.Item>

<Title level={5} className="mt-4">Ценность посылки, руб.</Title>
<Form.Item name="value" className="mt-2" rules={[
  {required: true, message: 'Введите ценность посылки' }
]}>
    <InputNumber min={0} max={100000} style={{width: '100%'}} placeholder="Введите сумму" className="w-full max-w-[400px]" />
</Form.Item>
```

```
<Row gutter={24}>
  <Col span={12} xs={24} sm={12}>
    <Form.Item
      name="passportSerie"
      label="Серия паспорта получателя"
      rules={[
        { required: true, message: 'Введите серию паспорта' },
        { pattern: /^\d{4}$/, message: 'Серия должна состоять из 4 цифр' },
      ]}>
      <Input.OTP
        length={4}
        size="large"
      />
    </Form.Item>
  </Col>
  <Col span={12} xs={24} sm={12}>
    <Form.Item
      name="passportNumber"
      label="Номер паспорта получателя"
      rules={[
      { required: true, message: 'Введите номер паспорта' },
      { pattern: /^\d{6}$/, message: 'Номер должен состоять из 6 цифр' }]}>
      <Input.OTP
        length={6}
        size="large"
      />
    </Form.Item>
  </Col>
</Row>

<Form.Item className="mt-6">
  <Popconfirm
    title="Подтвердить оформление посылки?"
    description={`Стоимость доставки: ${deliveryCost || '0'} руб.`}
    onOpenChange={() => {
      onCalculate(form.getFieldsValue());
    }}
    onConfirm={() => form.submit()}
    okText="Да"
```

```
                    cancelText="Нет">
                    <Button type="primary" size="large" className="w-[200px]">
                        Оформить
                    </Button>
                </Popconfirm>
            </Form.Item>
        </Form>
        {error && (
            <Alert
                message=""
                description={error.message}
                type={error.type}
            />
        )}
    </Card>
  </div>
 );
});

export default CreateOrder;
```

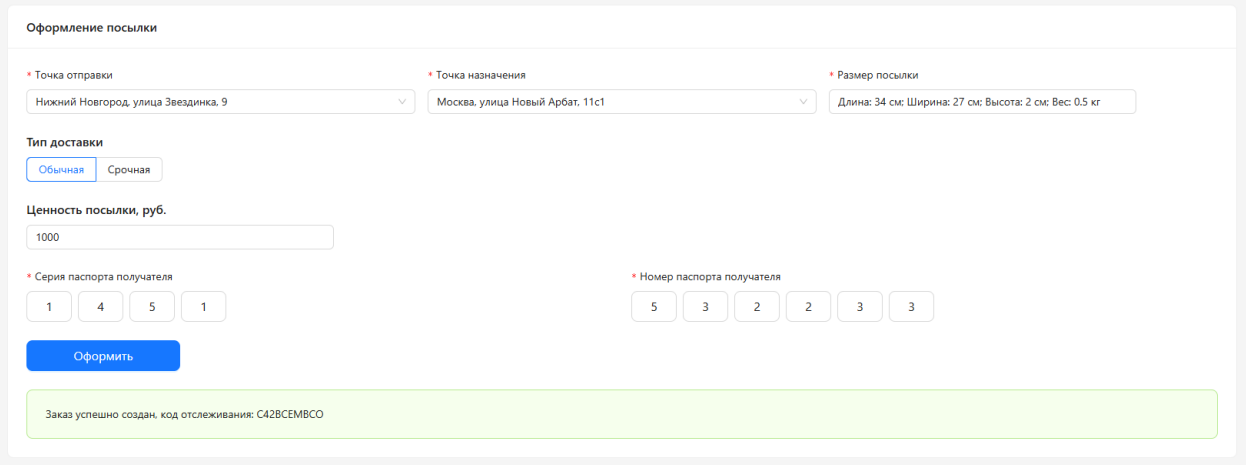

Рисунок 4.3.11 Страница создания новой посылки

5 Тестирование информационной системы

5.1 Автоматизированное тестирование с использованием Selenium IDE

Selenium IDE — это инструмент для записи и воспроизведения тестов веб-интерфейсов. Он позволяет автоматизировать проверку функциональности сайтов без написания кода (или с минимальным его количеством).

Основные особенности:

- Простота использования — тесты создаются через запись действий в браузере (клики, ввод текста, навигация).
- Поддержка браузеров — работает как расширение для Chrome, Firefox и Edge.

5.2 Тестовые сценарии

Были выбраны три места для тестирования: авторизация, запись на курс и изменение личной информации. Созданы и выполнены следующие тестовые сценарии с использованием Selenium IDE:

Таблица 5.2.1

Тестовый сценарий «Авторизация»

Действующее лицо		Неавторизованный пользователь	
Цель		Проверить процесс авторизации пользователя	
Предусловие		Открыта страница авторизации	
№	Команда Selenium	Параметры	Ожидаемый результат
1	open	http://localhost:5174/ authorization	Открывается страница авторизации
2	click	id=login_email	Нажатие на поле email
3	type	id=login_email	В поле email вводится адрес
4	click	id=login_password	Нажатие на поле password
5	type	id=login_password	В поле password вводится пароль
6	click	xpath=// button[@type='submit']	Происходит отправка формы

Результат: тестовый сценарий успешно выполняется, все проверки пройдены.

Таблица 5.2.2

Тестовый сценарий «Расчет стоимости доставки»

Действующее лицо		Пользователь	
Цель		Проверить процесс расчета стоимости доставки	
Предусловие		Открыта страница калькулятора стоимости доставки	
№	Команда Selenium	Параметры	Ожидаемый результат
1	open	http://localhost:5174/ calculator	Открывается страница калькулятора стоимости доставки
2	click	id=from	Нажатие на поле выбора пункта отправки
3	click	xpath=(.//*[normalize-space(text()) and normalize-space(.)='Москва, улица Новый Арбат, 11с1'])[1]/following::div[2]	Выбор пункта
4	click	id=to	Нажатие на поле выбора пункта нащначения
5	click	xpath=(.//*[normalize-space(text()) and normalize-space(.)='Новосибирск, Красный проспект, 182']) [2]/following::div[2]	Выбор пункта
6	click	xpath=//div[@id='root']/div/ div/main/div/div/div[2]/ form/div/div[3]/div/div/ div[2]/div/div/input	Нажатие на форму выбора размера посылки
7	click	xpath=//div[@id='rc-tabs-1-panel-1']/div/label	Выбор вкладки предустановленных размеров
9	click	xpath=// div[@id='deliveryType']/label	Выбор типа доставки
10	click	id=value	Нажатие на поле

			ввода ценности посылки
11	type	id=value	Ввод ценности посылки
	click	xpath=//div[@id='root']/div/div/main/div/div/div[2]/form/div[4]/div/div/div/div/button/span	Нажатие на кнопку « Рассчитать»

Результат: тестовый сценарий успешно выполняется, все проверки пройдены.

5.3 Проверка адаптивного дизайна

Адаптивный веб-дизайн гарантирует оптимальное отображение сайта на любых устройствах — от смартфонов до десктопов. В рамках тестирования был разработан специализированный тест-кейс, который автоматически проверяет корректность визуального представления интерфейса при динамическом изменении размеров окна браузера.

Таблица 5.2.1

Тестовый сценарий «Проверка адаптивного дизайна»

Действующее лицо		Клиент	
Цель		Проверить адаптивность дизайна	
Предусловие		Открыта страница личного	
№	Команда Selenium	Параметры	Ожидаемый результат
1	open	http://localhost:5174/profile	Открывается страница личного кабинета
2	setWindowSize	1552x928	Проверка отображения на десктопе
3	setWindowSize	820x1180	Проверка отображения на планшете
4	setWindowSize	390x844	Проверка отображения на мобильном устройстве

На рисунках 5.3.1, 5.3.2 и 5.3.3 представлены результаты проверки отображения главной страницы при различных размерах окна браузера:

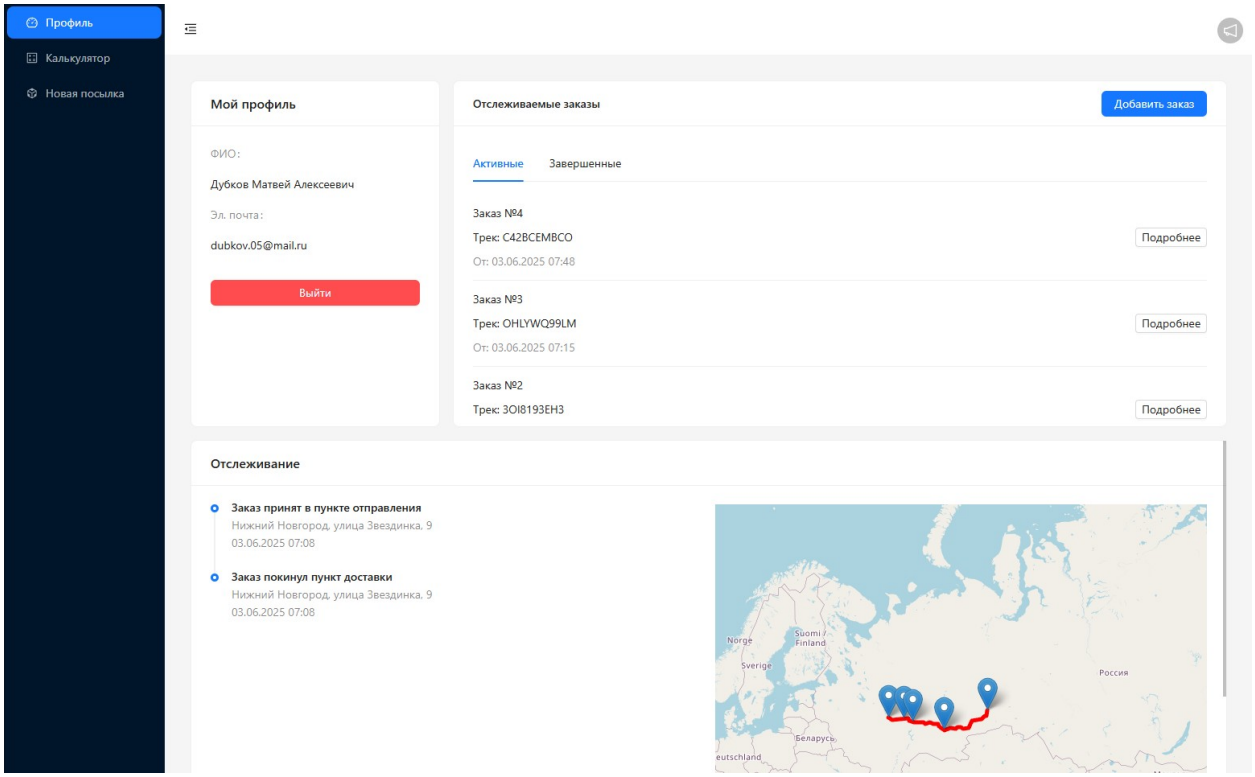

Рисунок 5.3.1 Отображение на десктопе (1552x928)

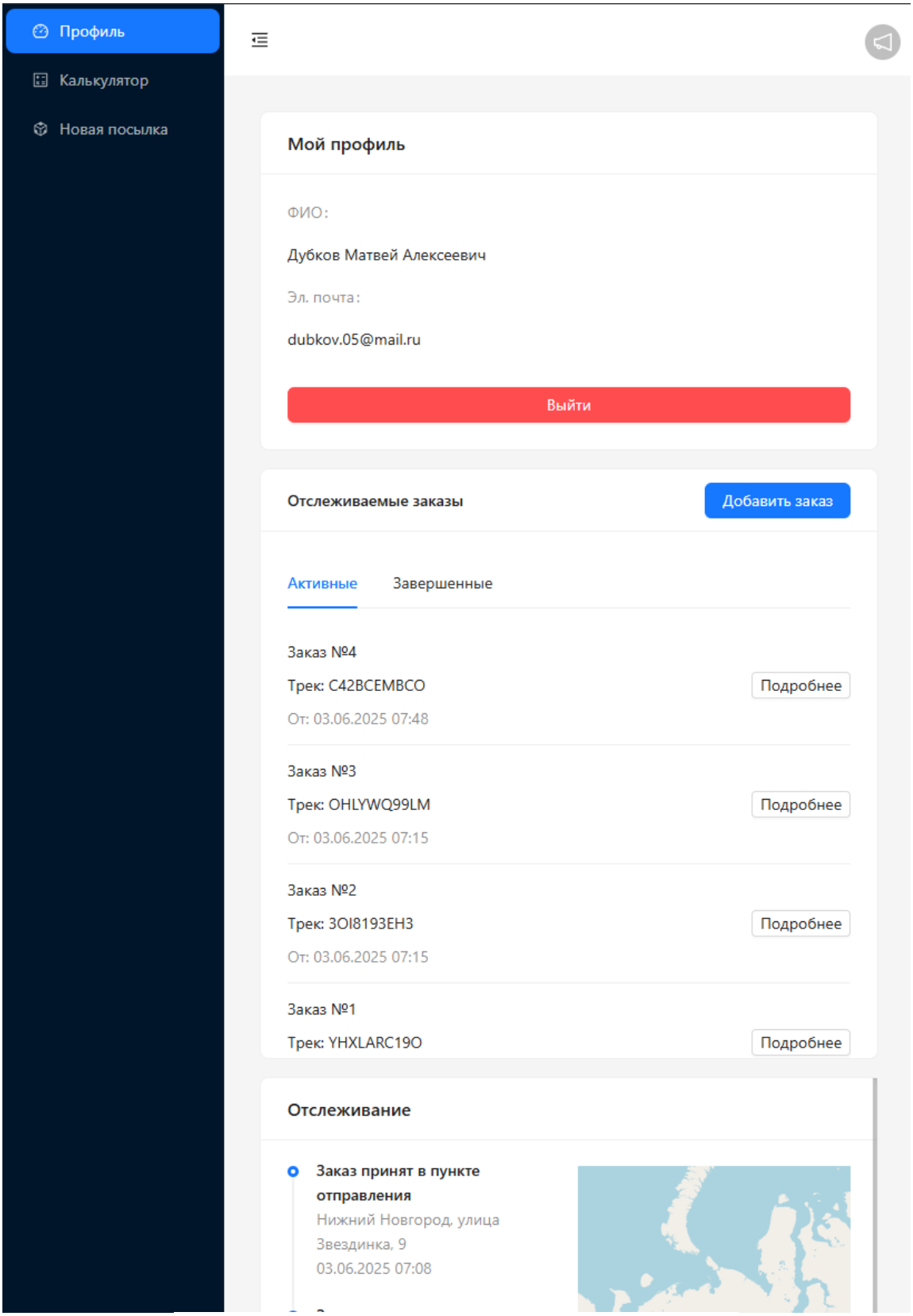

Рисунок 5.3.2 Отображение на планшете (820x1180)

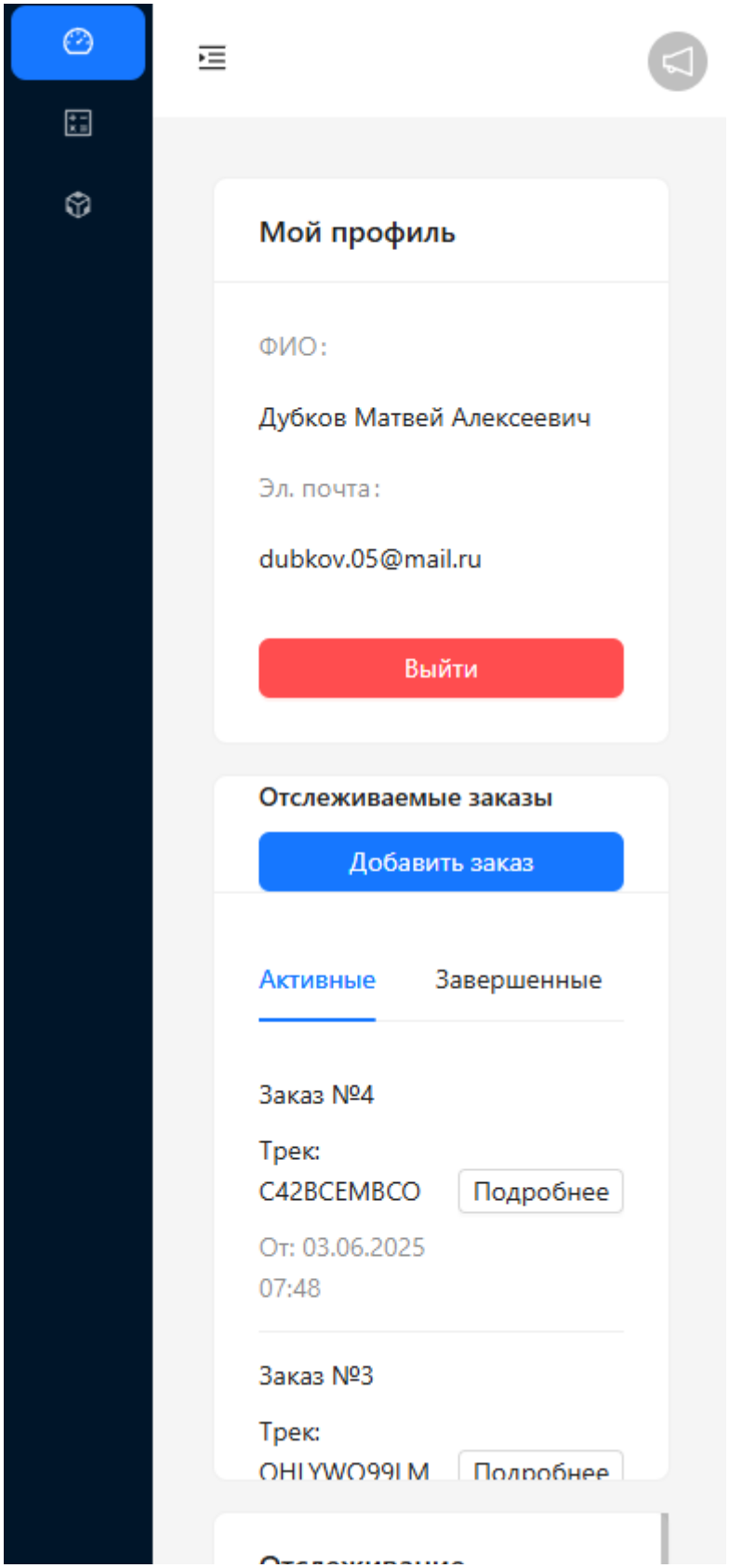

Рисунок 5.3.3 Отображение на мобильном устройстве (390x844)

6 Руководство пользователя

6.1 Введение

6.1.1 Назначение руководства

Данное руководство пользователя предназначено для ознакомления как новых, так и действующих пользователей с возможностями веб-приложения службы доставки. Его основная задача — облегчить освоение интерфейса, объяснить принципы работы с ключевыми разделами системы и помочь эффективно применять все обучающие инструменты платформы.

Руководство ориентировано на широкую аудиторию: от начинающих разработчиков до опытных специалистов, стремящихся расширить свои знания. Документ подготовлен с акцентом на доступность и наглядность, содержит понятные и пошаговые инструкции.

6.1.2 Обзор ролей

1) Пользователь — имеет доступ к личной информации, к общей информации на сайте, может создавать посылки, отслеживать посылки, обучаться.

2) Управляющий - имеет доступ к личной информации, может принимать посылки в пункт сортировки, а также отправлять их дальше по маршруту.

3) Администратор — имеет доступ данным статистики работы пунктов доставки.

6.1.3 Использование руководства

Руководство разделено на разделы(пункты) в соответствии с ролями. Выберите нужный раздел и следуйте инструкциям.

6.2 Общее руководство для всех ролей

6.2.1 Общая информация

При открытии веб-приложения пользователь попадает на страницу авторизации(Рисунок 2.1.1), но он также доступна страница калькулятора стоимости доставки(Рисунок 2.1.2). Навигация по доступным страницам происходит в меню, находящимся в левой части экрана(Рисунок 2.1.3).

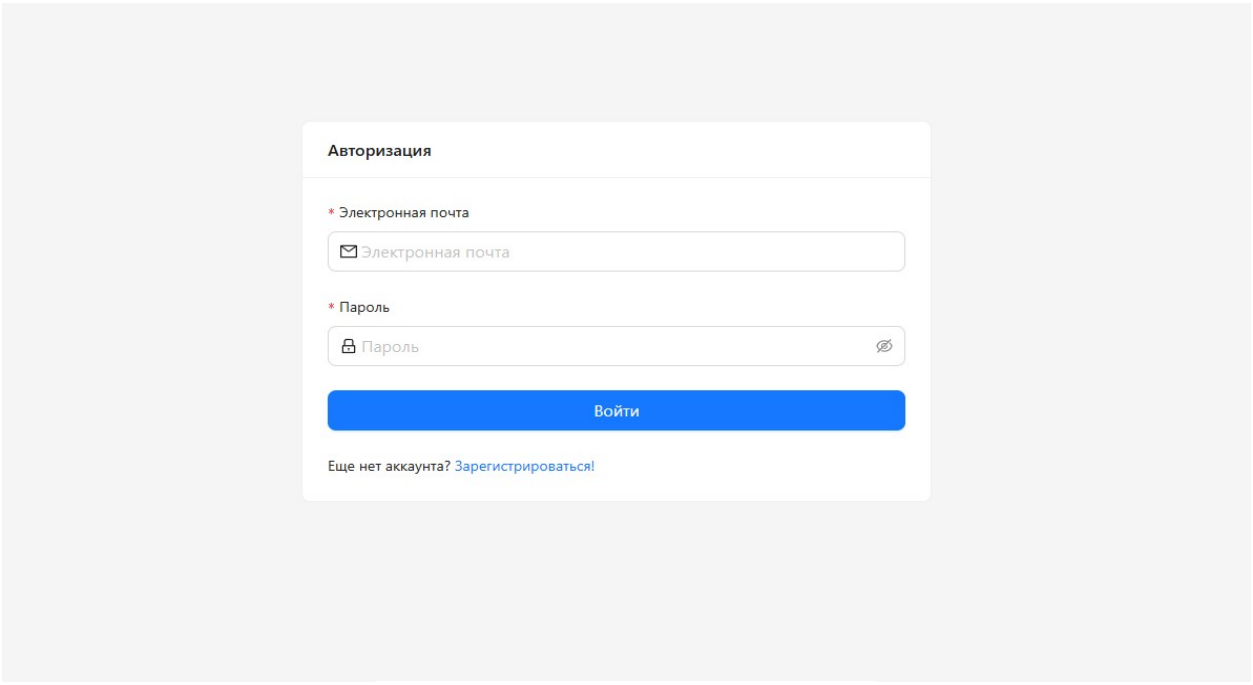

Рисунок 2.1.1 Страница авторизации

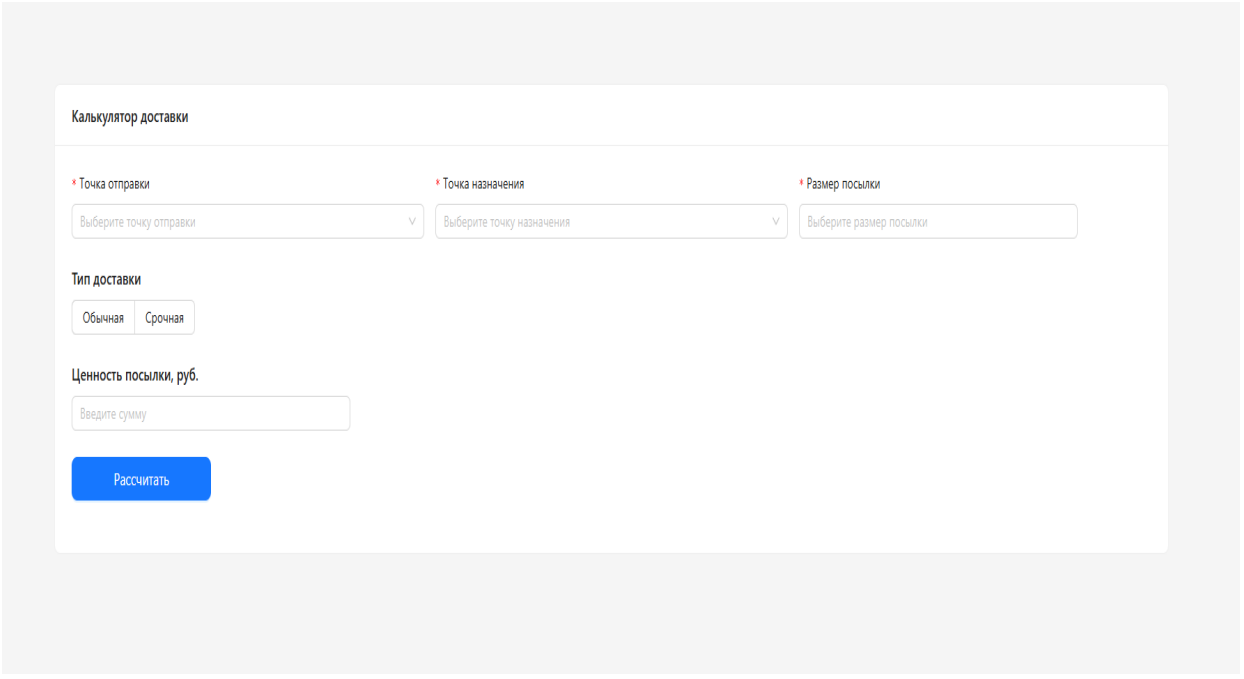

Рисунок 2.1.2 Страница калькулятора стоимости доставки

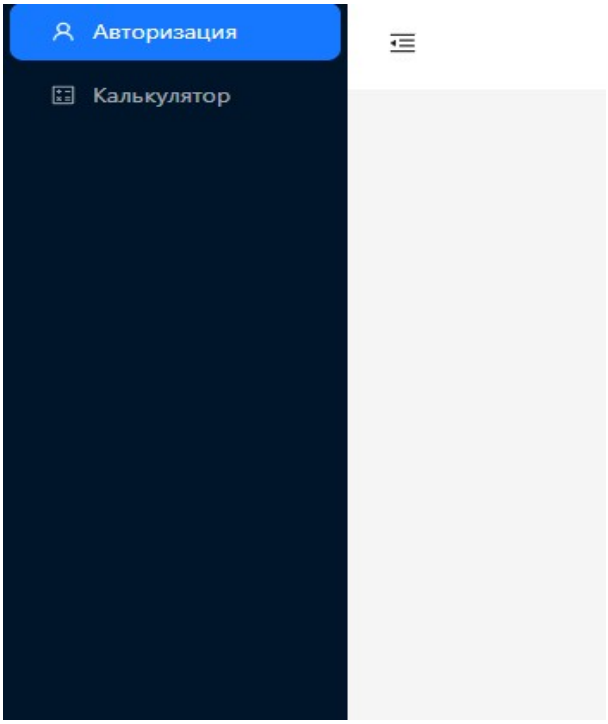

Рисунок 2.1.3 Меню

6.2.2 Регистрация

На странице авторизации нужно перейти по ссылке «Зарегистрироваться!», таким образом мы попадаем на страницу регистрации(Рисунок 2.2.1).

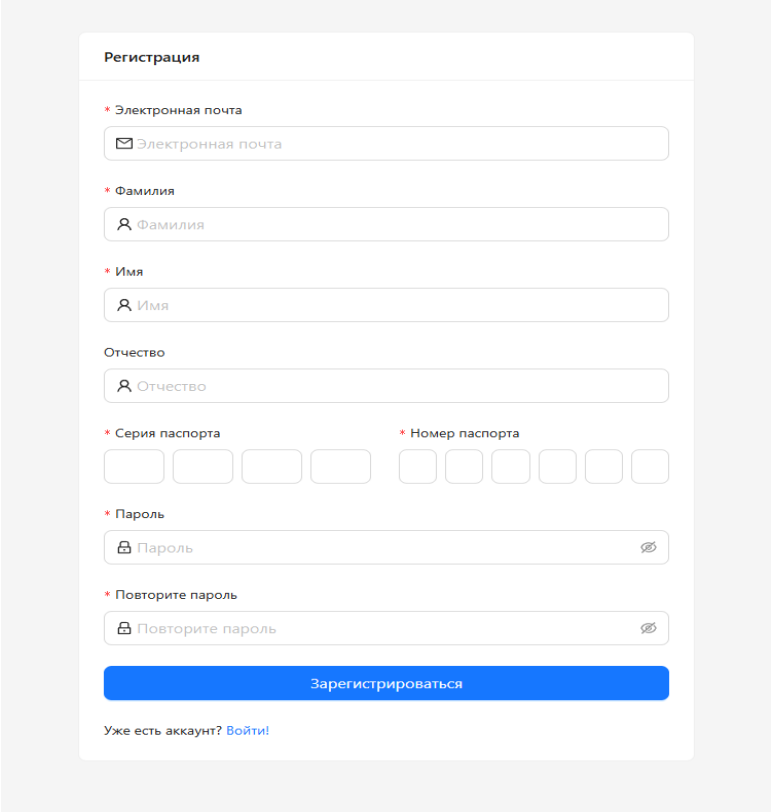

Рисунок 2.2.1 Страница регистрации

Далее необходимо ввести данные в форму: электронная почта, фамилия, имя, отчество, серия и номер паспорта, два раза пароль(Рисунок 2.2.2).

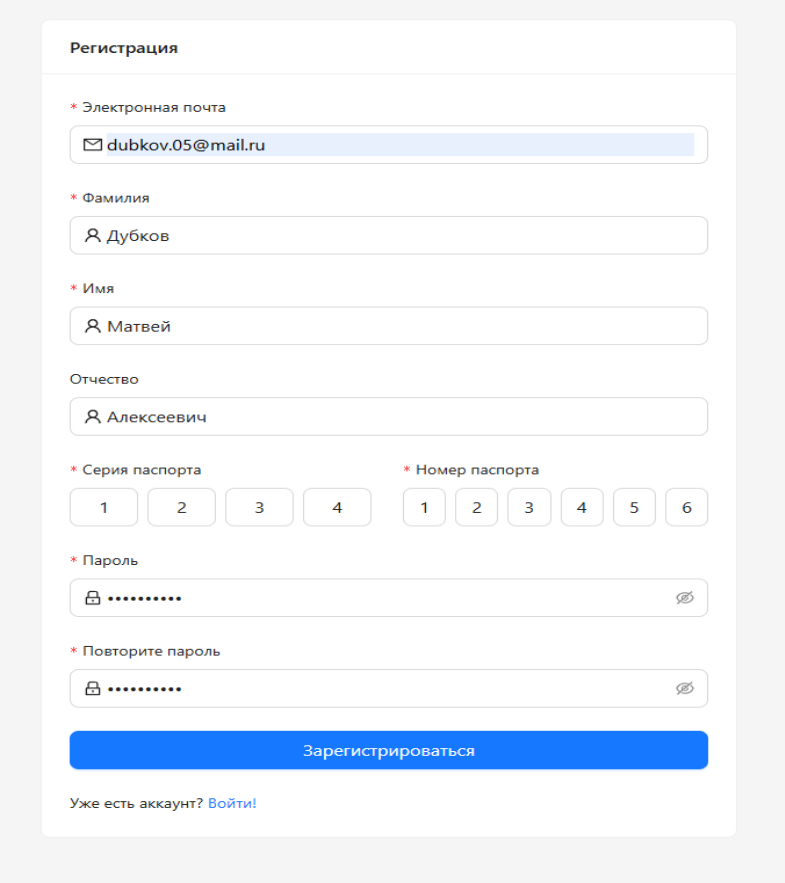

Рисунок 2.2.2 Заполненная форма регистрации

Далее нажимаем кнопку «Зарегистрироваться» и нас перенаправляет на страницу личного кабинета(Рисунок 2.2.3).

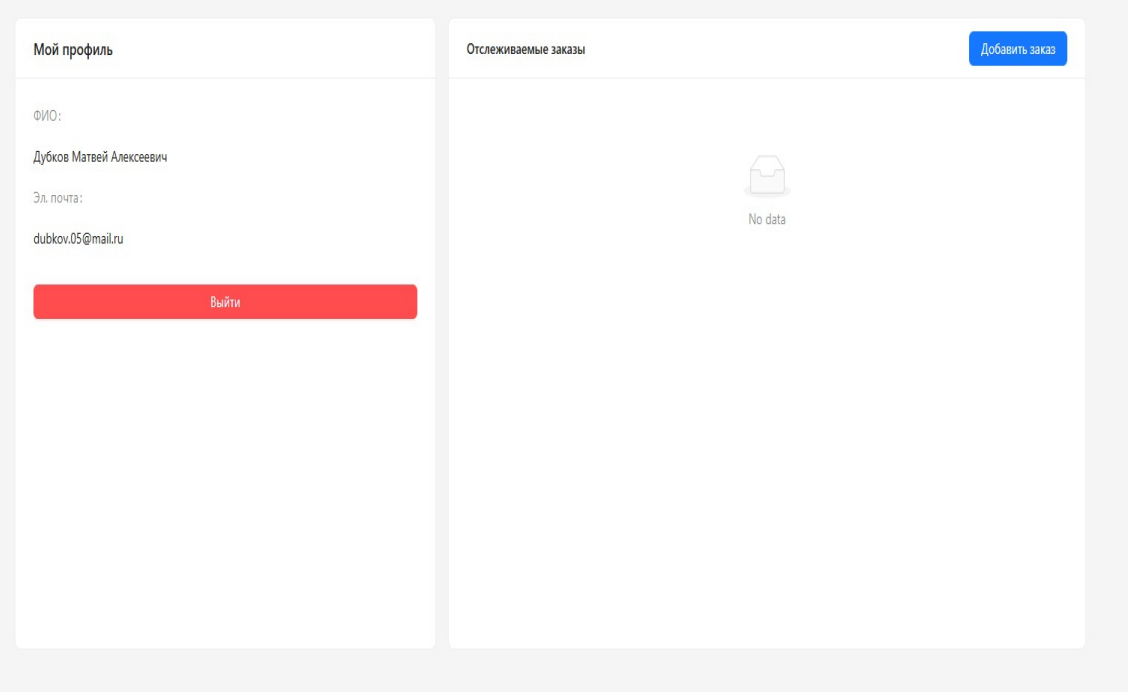

Рисунок 2.2.3 Личный кабинет

6.2.3 Авторизация

На странице авторизации(Рисунок 2.1.1) необходимо заполнить форму(Рисунок 2.3.1).

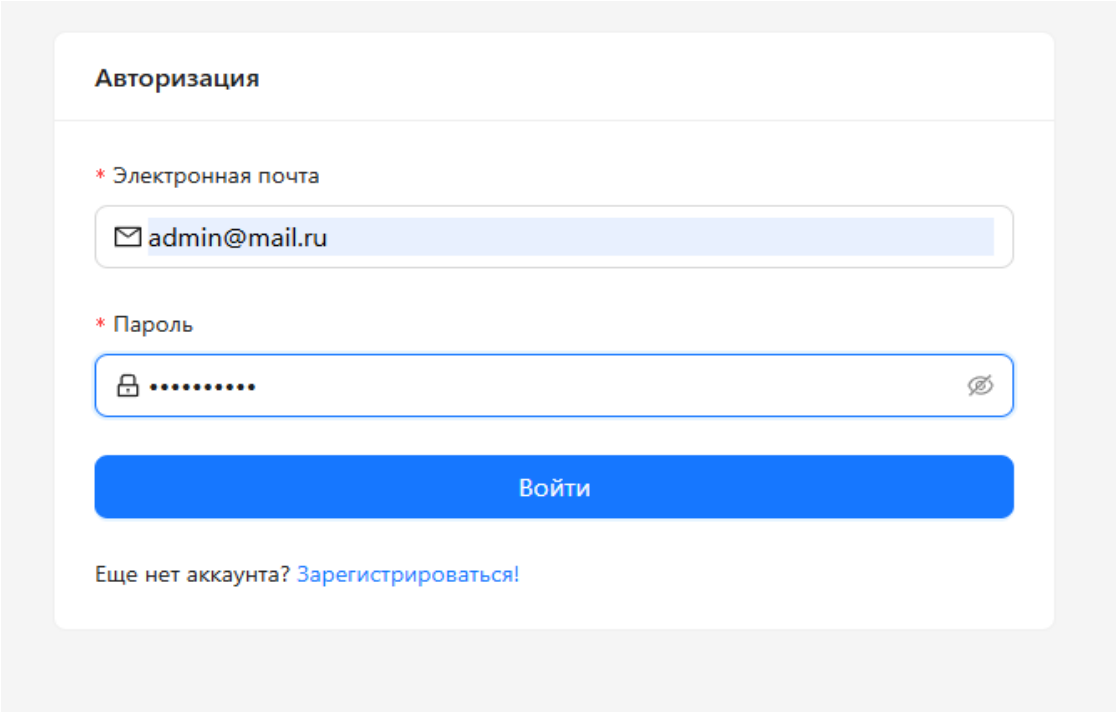

Рисунок 2.3.1 Заполненная форма авторизации

Далее нажимаем кнопку «Войти» и попадаем в личный кабинет(Рисунок 2.2.3).

6.2.4 Калькулятор стоимости доставки

Чтобы попасть на страницу калькулятора(Рисунок 2.1.2) необходимо нажать соответствующую кнопку в меню(Рисунок 2.1.3). Далее заполняем форму(Рисунок 2.4.1) и нажимаем кнопку «Рассчитать». Стоимость доставки выведется под кнопкой(Рисунок 2.4.2).

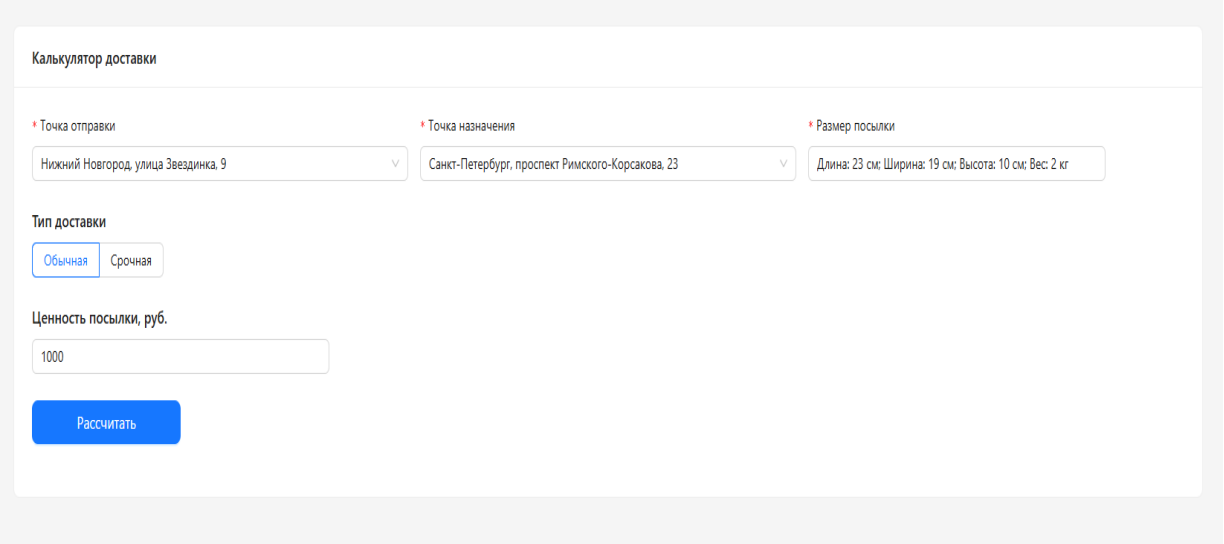

Рисунок 2.4.1 Заполненная форма калькулятора

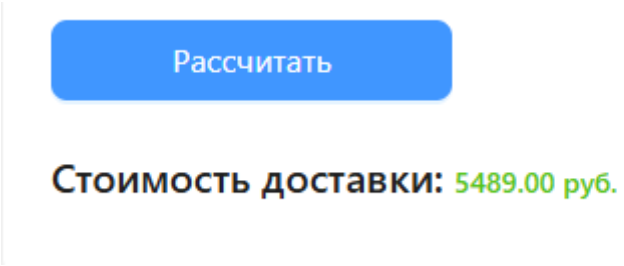

Рисунок 2.4.2 Вывод стоимости доставки

6.3 Руководство для клиента

6.3.1 Создание посылки

Для создания посылки необходимо нажать на соответствующую кнопку в меню.(Рисунок 2.1.3). Далее необходимо заполнить форму(Рисунок 3.1.1) и нажать кнопку «Рассчитать», далее в окне подтверждение нажать «Да»(Рисунок 3.1.2).

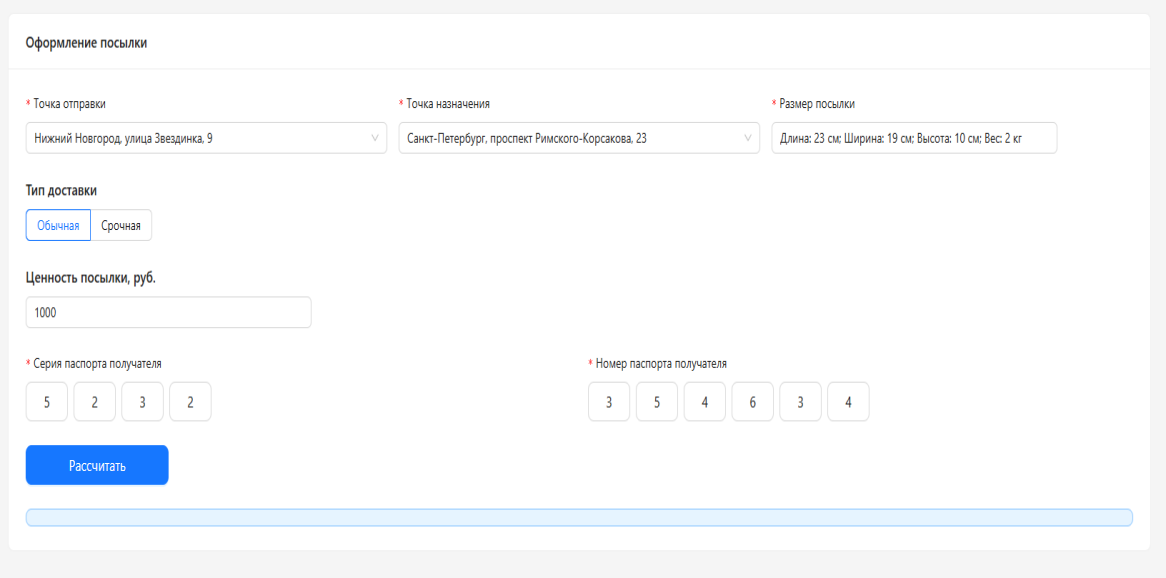

Рисунок 3.1.1 Заполненная форма создания посылки

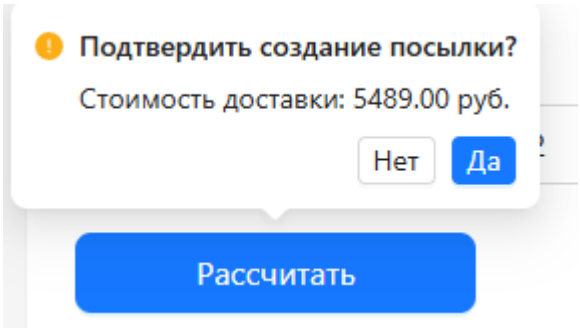

Рисунок 3.1.2 Окно подтверждения создания посылки

6.3.2 Отслеживание посылок

После оформления посылки, он автоматически появится в профиле, в разделе «Отслеживаемые посылки»(Рисунок 3.2.1). Также можно добавить в отслеживаемые посылку, которая не создана Вами. Для этого необходимо нажать кнопку «Добавить посылку» и в выпадающем окне(Рисунок 3.2.2) ввести трек-код и нажать клавишу добавить.

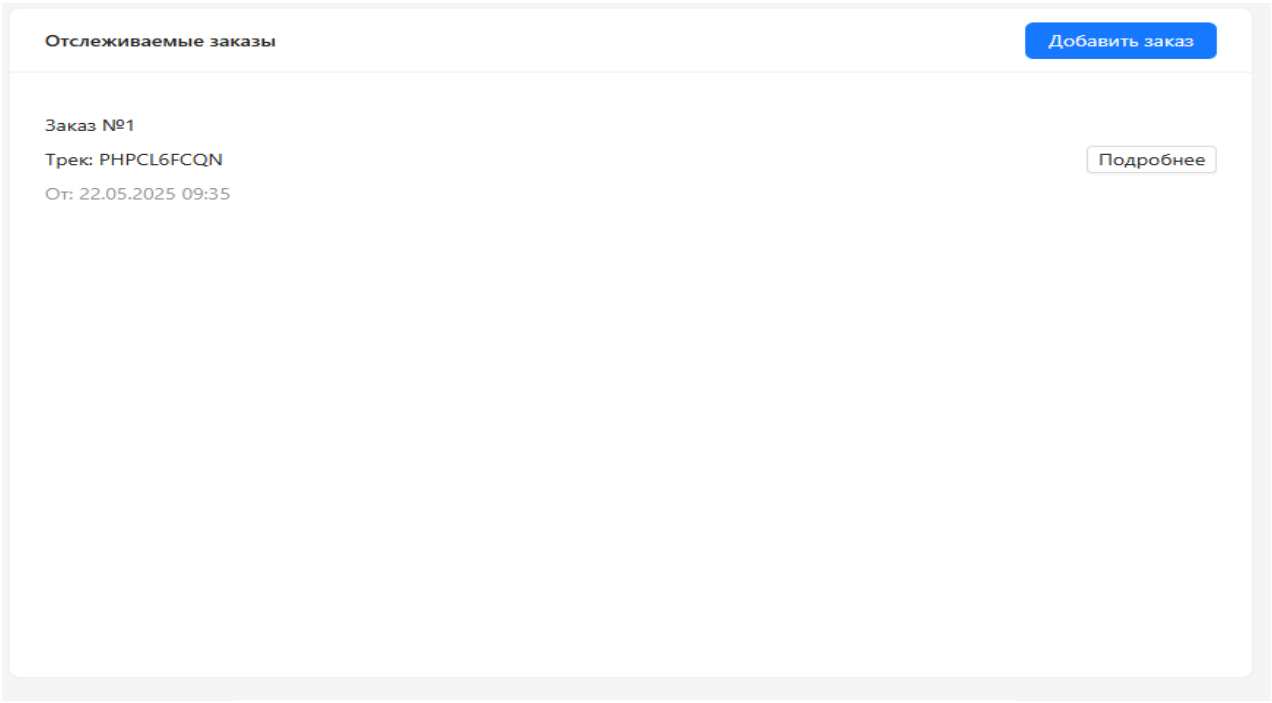

Рисунок 3.2.1 Раздел отслеживаемых посылок в профиле

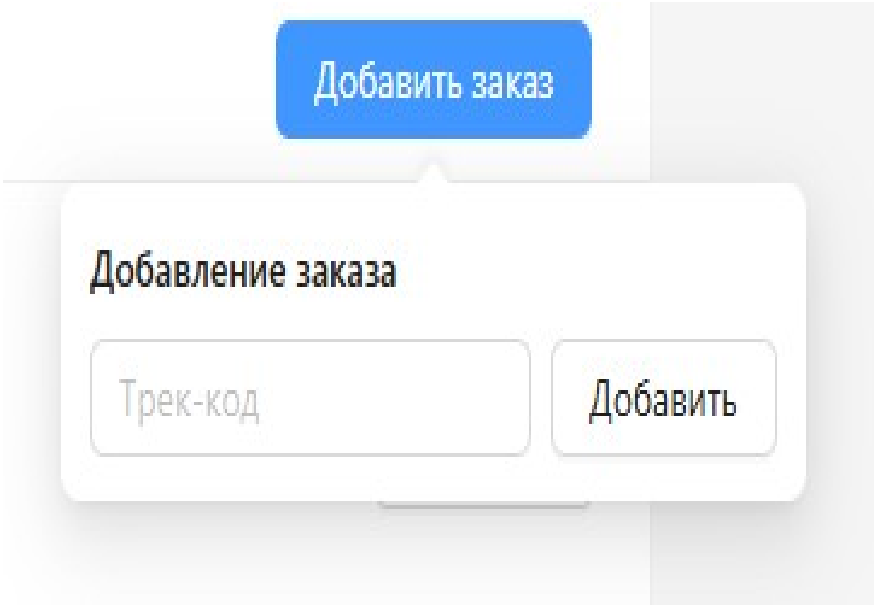

Рисунок 3.2.2 Выпадающее меню добавления посылки

Для получения подробной информации о посылке, необходимо нажать кнопку «Подробнее» у нужной посылки. Появится раздел, отображающий информацию о доставке посылки(Рисунок 3.2.3).

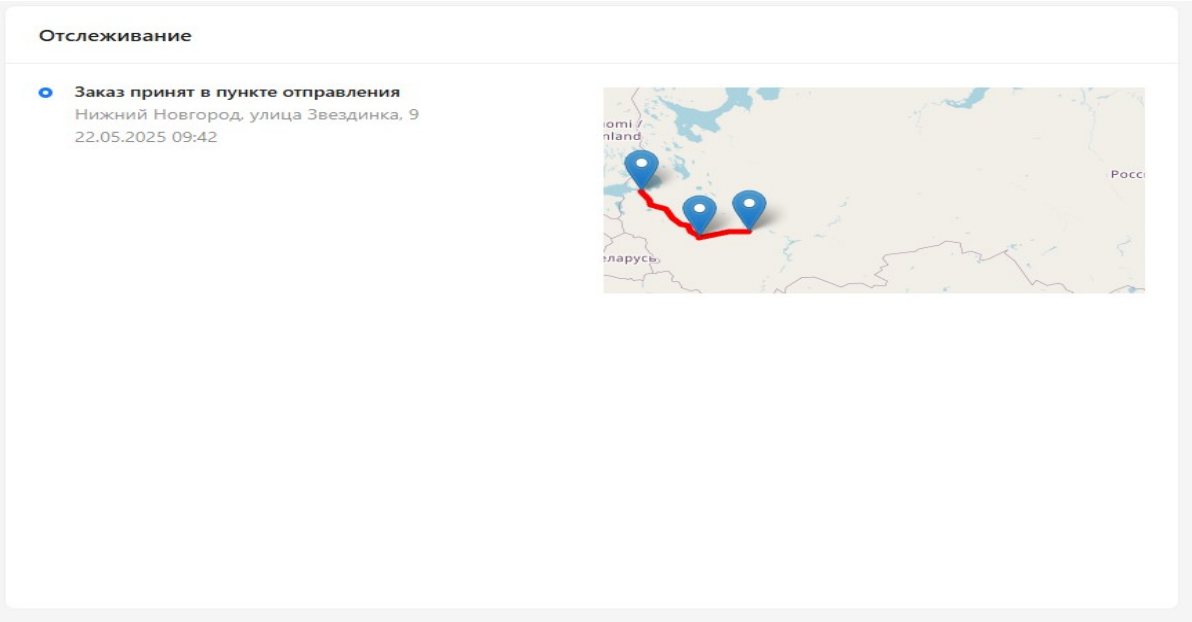

Рисунок 3.2.3 Раздел отслеживания

6.4 Руководство управляющего

6.4.1 Принятие и отправка посылок

В личном кабинете управляющего есть раздел «Ожидаемые посылки»(Рисунок 4.1.1) и «Посылки, ожидающие отправки»(Рисунок 4.1.2). Пользователь может найти необходимые посылки. Для этого необходимо воспользоваться вводом для поиска(Рисунок 4.1.3).

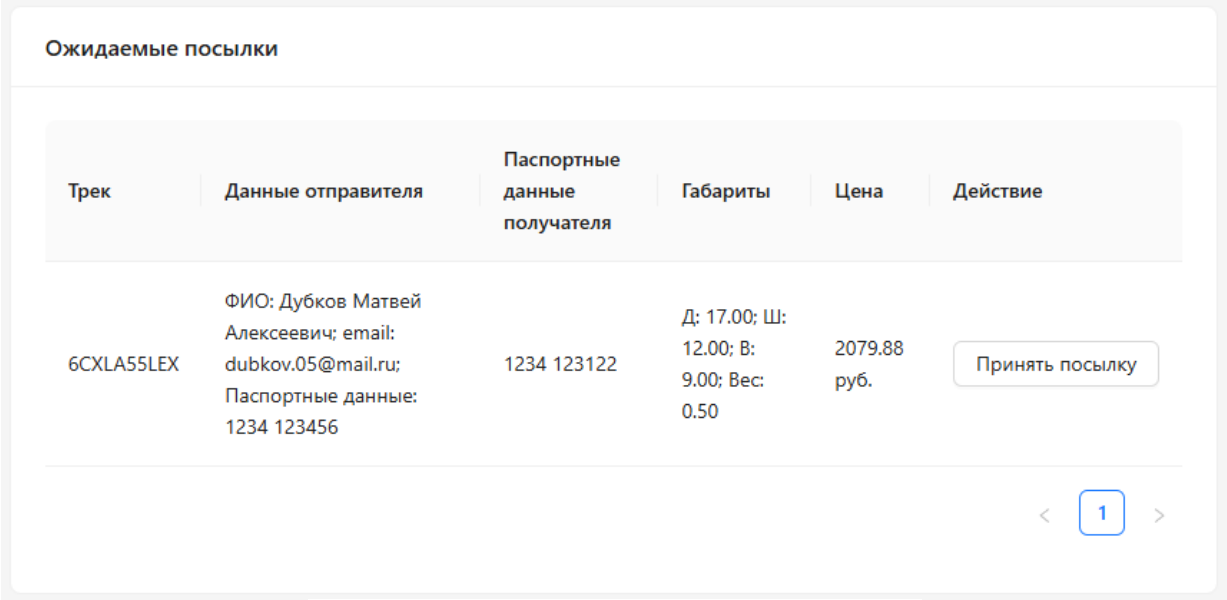

Рисунок 4.1.1 Раздел «Ожидаемые посылки»

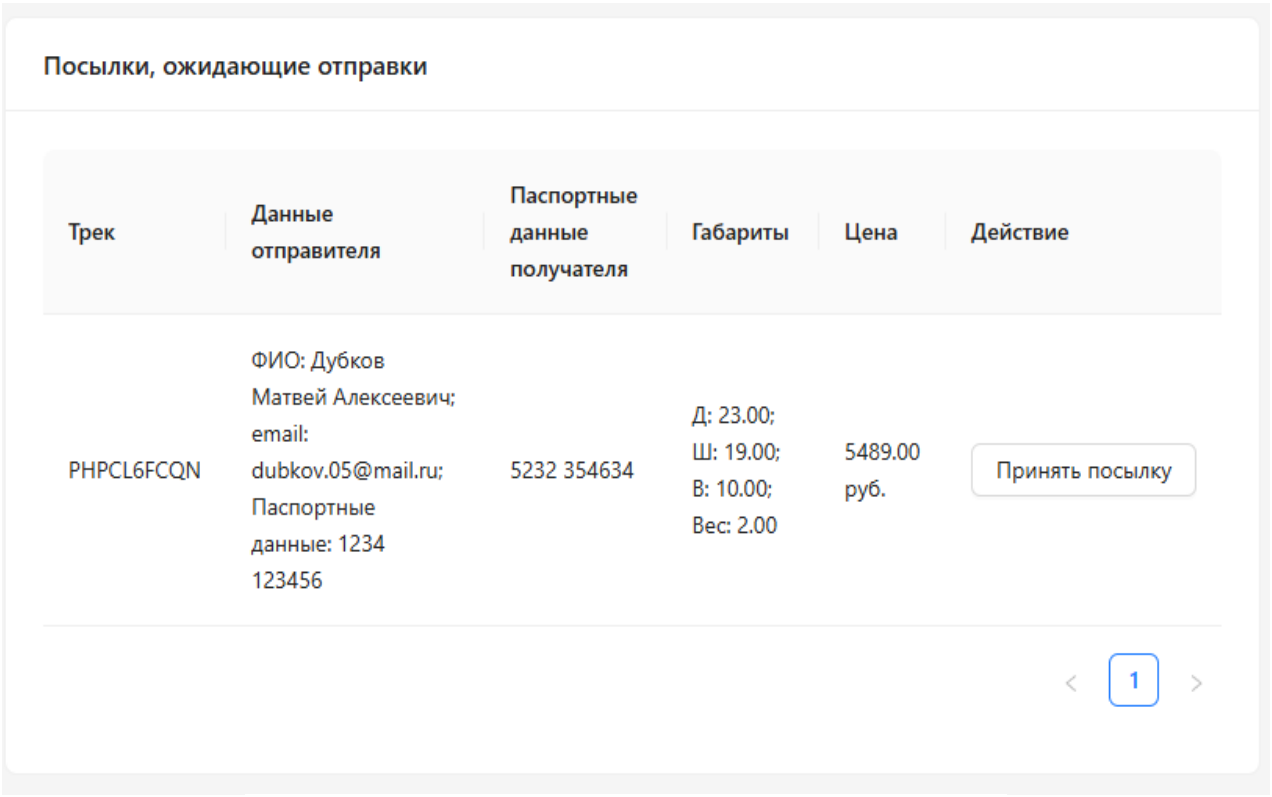

Рисунок 4.1.2 Раздел «Посылки, ожидающие отправки»

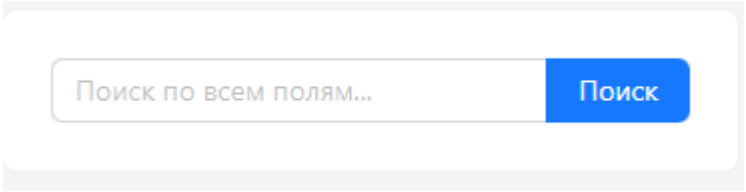

Рисунок 4.1.3 Ввод поиска

6.5 Руководство администратора

6.5.1 Просмотр статистики работы пунктов доставки.

В личном кабинете администратора доступны график, отображающий информацию о эффективности пунктов доставки(Рис 5.1.1).

Рисунок 5.1.1 График эффективности пунктов доставки

Помимо этого, администратору доступны другие графики(Рисунок 5.1.2).

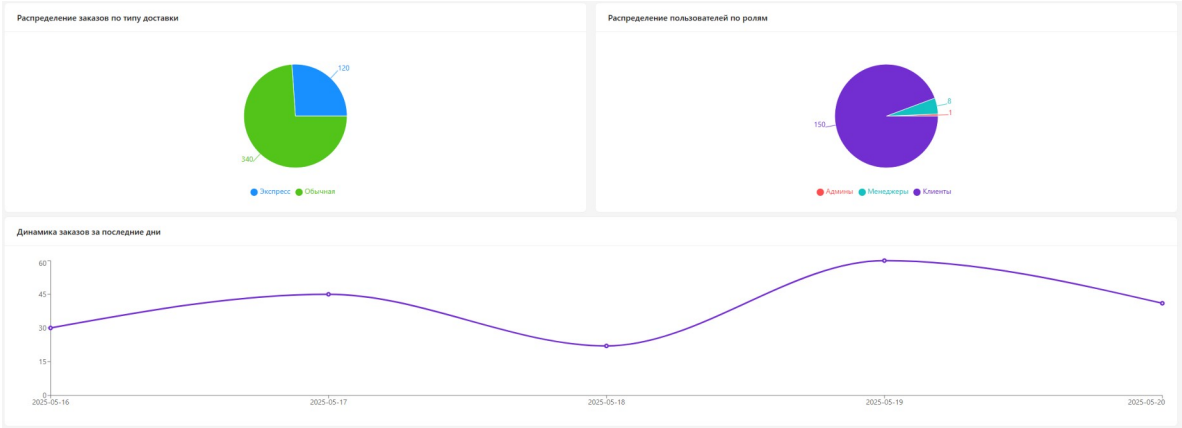

Рисунок 5.1.2 Прочие графики

7 Экономический анализ

В данной части дипломного проекта по «Разработке веб-приложения для службы доставки» необходимо оценить разработанное приложение для определения его рентабельности и срока окупаемости.

Для работы данной информационной системы необходимы следующие минимальные требования:

Таблица 7.1

Основные характеристики системы

Наименование	Характеристики
Операционная система	MS Windows 98/Me, MS Windows NT 4.0/2000/XP/Vista/7/8/Server 2003 (рекомендуется MS Windows 7/8/10/Server 2012)
Среда выполнения JavaScript (Node.js)	Версия 22.15.0
Процессор	Intel Core i5 или AMD Ryzen 5 и последующие модели
Оперативная память	Минимально: 8ГБ, рекомендуется: 16Гб
Жёсткий диск	От 10 Гб и выше
USB-порт	2.0
Видеокарта	SVGA

7.1 Расчёт полезного времени работы

Баланс рабочего времени — это количественный показатель, отражающий среднее число часов, которое работник должен отработать за установленный плановый период (месяц, квартал, год). Он рассчитывается как произведение количества рабочих дней в периоде на среднюю продолжительность рабочего дня в часах.

Рабочее время — это период, в течение которого работник в соответствии с правилами внутреннего распорядка и условиями трудового договора выполняет свои должностные обязанности. К рабочему времени также относятся иные временные интервалы, которые согласно Трудовому кодексу Российской Федерации и другим нормативным актам включаются в его состав.

В рамках данной работы необходимо произвести расчёт баланса рабочего

времени на апрель и май 2025 года.

Таблица 7.1.1

Баланс рабочего времени

Показатели	**Дни/часы**
1. Календарный фонд времени	61
2. Количество нерабочих дней	21
2.1. Праздничные	2
2.2. Выходные	19
3. Количество календарных рабочих дней (номинальный фонд)	40
4. Потери рабочего времени в связи с сокращением продолжительности рабочего дня (часов)	1
5. Средняя продолжительность рабочего дня (часов)	7,975
6. Полезный эффективный фонд рабочего времени (часов)	319

Календарный фонд времени — это общее количество дней в рассматриваемом периоде. Номинальный фонд времени рассчитывается как календарное количество дней за вычетом выходных и праздничных. Полезный (или эффективный) фонд рабочего времени определяется путем умножения числа рабочих дней в месяце на среднюю продолжительность рабочего дня.

1. Номинальный фонд:

$Ф_{н.}=Ф_{календ.}-Д_{нераб.}$,

$Ф_{н.}$=61 – 21 = 40 (дней);

2. Потери рабочего времени в день:

ПРВ = ТП + ПД * ВС / Фн

Где:

ПД – предпраздничные дни;

ВС – время сокращенного рабочего дня

ПРВ = 1 * 1 / 40 = 0,025 (часа)

3. Средняя продолжительность рабочего дня:

Ср.прод.раб.дня = длит.раб.дня – ПРВ (час)

СрПрРабД = 8 – 0,025 = 7,975 (часа)

4. Полезный (эффективный) фонд времени рассчитывается произведением числа рабочих дней в месяце на среднюю продолжительность рабочего дня.

Fвр = число.раб.дн*ср.прод.раб.дня (час)

Fвр = 40 * 7,975 = 319 (часов)

7.2 Обоснование выбора специалистов для разработки

Для выполнения работ по разработке веб-приложения была выбрана должность инженера-программиста (веб-разработчика). В соответствии с квалификационными требованиями, инженер-программист должен обладать компетенциями в области разработки клиент-серверных приложений с использованием Node.js на серверной стороне и React — на клиентской. Также необходим опыт работы с REST API, базами данных, а также знание принципов современной веб-разработки.

Должностные обязанности инженера-программиста:

1. Разрабатывает архитектуру веб-приложения и определяет технологический стек.
2. Осуществляет выбор и реализацию методов взаимодействия между клиентской и серверной частями приложения.
3. Разрабатывает пользовательский интерфейс (UI) с использованием современных фреймворков (в частности, React).
4. Разрабатывает и реализует серверную логику на базе Node.js, включая маршрутизацию, обработку запросов, взаимодействие с базой данных.
5. Определяет формат, структуру и объем данных, подлежащих обработке, а также схемы их хранения и передачи.
6. Выполняет тестирование, отладку и устранение ошибок как на клиентской, так и на серверной части.
7. Проводит интеграцию клиентского и серверного модулей, обеспечивает безопасность и защиту данных.
8. Создает и сопровождает техническую и пользовательскую документацию.
9. Осуществляет сопровождение и поддержку разрабатываемого программного продукта после внедрения.

Дополнительно, в рамках специфики проекта, в обязанности веб-разработчика входит:

1. Проектирование REST API и обеспечение их корректной работы с клиентским интерфейсом.
2. Подключение и настройка СУБД MySQL, а также взаимодействие с базой данных с использованием ORM-библиотеки Sequelize.
3. Реализация адаптивной и интерактивной клиентской части, в том числе с использованием компонентов UI-библиотеки Ant Design.
4. Настройка среды разработки, сборки и развертывания проекта.
5. Оптимизация производительности приложения и обеспечение его масштабируемости.

Требования к квалификации:

Высшее профессиональное (техническое или инженерно-экономическое) образование или стаж работы в должности веб-разработчика не менее двух лет, или среднее профессиональное (техническое или инженерно-

экономическое) образование и стаж работы в должности программиста веб-разработчика не менее двух лет.

Таблица 7.2.1

Основные этапы разработки

Этап	Проводит	Время на этап/час
Анализ требований и предметной области	Инженер-программист	5
Проектирование архитектуры веб-приложения	Инженер-программист	20
Разработка веб-приложения	Инженер-программист	220
1. Разработка серверной части	Инженер-программист	60
2. Разработка клиентской части		120
3. Интеграция и настройка взаимодействия клиент–сервер		40
Тестирование функциональности	Инженер-программист	15
Внесение доработок по результатам тестирования	Инженер-программист	20
Разработка пользовательской документации	Инженер-программист	10
Проведение экономического анализа проекта	Инженер-программист	10
Всего		300

7.3 Расчёт заработной платы и отчислений во внебюджетные фонды

Заработная плата (оплата труда работника) представляет собой

совокупность экономических и правовых отношений, связанных с вознаграждением за выполненную работу. Размер оплаты зависит от квалификации сотрудника, сложности, объема и качества выполняемых задач, а также условий труда. В структуре заработной платы выделяют две основные части: компенсационные и стимулирующие выплаты.

Компенсационные выплаты предназначены для возмещения особых условий труда, таких как работа в неблагоприятных или экстремальных климатических зонах, на территориях с радиационным загрязнением и других отклонениях от нормальных условий. Эти выплаты компенсируют дополнительные риски и неудобства, связанные с выполнением профессиональных обязанностей в таких условиях.

Стимулирующие выплаты направлены на повышение мотивации и эффективности труда работников. В эту категорию входят доплаты и надбавки стимулирующего характера, премии и другие поощрительные выплаты, которые способствуют повышению производительности и достижению лучших результатов.

Заработная плата складывается из основной и дополнительной:

ЗПобщ = ЗПосн + ЗПдоп

Выделяют 2 основные формы оплаты труда:

- повременная – оплата труда производится за фактически отработанное время, независимо от результатов работы;
- сдельная - оплата труда производится за объём выполненных работ, независимо от потраченного времени.

Сдельная зарплата бывает:

- сдельно-премиальной;
- аккордной, простой сдельной;
- сдельно-прогрессивной.

Повременная состоит из:

- простой;
- повременно-премиальной.

7.4 Расчёт заработной платы для рабочих повременщиков

Для определения оклада воспользуемся статистическими данными. Средний месячный оклад по Нижегородской области для инженера-программиста с опытом работы от 2 лет составляет 76800 рублей.

Программист отработал не полных два месяца. Его оплата составляет:

Оплата = $(З_н*2)*(Т_ф/Т_н)$

$З_н$ - Оклад

$Т_ф$ - Фактическая продолжительность работы за месяц, ч.

$Т_н$ – Нормативная продолжительность работы за месяц, ч.

Оплата = (76.800*2)*(300/320,5) = 143213,728 (руб).

Расчёт фонда оплаты труда и отчислений во внебюджетные фонды

Размер совокупных страховых взносов равен 30,2%:

10. единая предельная величина базы для исчисления страховых взносов (ФОМС, СФР) - 30%;
11. по страхованию от несчастных случаев («на травматизм») (СФР) – 0,2%.

Расчёты производятся по одной формуле:

Ф = ФОТ * Ф%

Где:

Ф – сумма отчислений в определённый фонд;

ФОТ – заработная плата работника повременщика;

Ф% - процент отчислений в определённый фонд.

Отчисления от зарплаты программиста 1С :

Отч. во внебюдж.фонды = 143213,728 * 0,302 = 43250,545(руб).

7.5 Расчёт материальных затрат разработки

Материальные затраты на разработку и внедрение программного продукта представляют собой сумму денежных средств, затраченных на приобретение необходимых материалов, непосредственно задействованных в процессе создания и внедрения системы. Оценка стоимости таких материалов производится исходя из актуальных рыночных цен. Материальные затраты (расходы) образуют часть себестоимости продукции.

Таблица 7.5.1

Расчёт материальных затрат

Наименование расходного материала	**Единица измерения**	**Норма расхода на проект**	**Оптовая цена за единицу материала, руб.**	**Сумма, руб.**
Интернет-трафик	Месяц	2	870*	1740
Пачка бумаги	Листы	100	1,5	150
Флеш-накопитель на 32 гб	Штук	1	400	400
Итого				**2290**

*Интернет – провайдер Ростелеком, тариф 870 рублей.

7.6 Расчёт суммы платежей за электроэнергию

Таблица 7.6.1

Расчёт суммы платежей за электроэнергию

Наименование оборудования	Кол-во оборудования Q (шт.)	Потребляемая мощность, кВт	Время работы, час	Потребляемая энергия, кВт/ч
Ноутбук Realme Book Prime 14	1	0,4	300	120
Принтер	1	0,3	3	0,9
Итого				**120,9**

Информация по стоимости электроэнергии тариф на электроэнергию, дифференцированный по двум зонам суток, дневная зона (с 7 до 23 часов), соц норма 5,47 руб за 1 кВт.ч.

Эл.эн = стоимость 1кВт/ч * потр.мощн

Эл.эн = 5,47 * 120,9 = 661,323 (руб).

7.7 Расчёт сумм амортизационных отчислений

Амортизация представляет собой процесс поэтапного переноса стоимости основных средств и нематериальных активов на себестоимость выпускаемой продукции, выполняемых работ или оказываемых услуг.

Начисление амортизации осуществляется ежемесячно, начиная с месяца, следующего за датой ввода объекта в эксплуатацию. Прекращение начислений происходит с первого числа месяца, следующего за полным списанием или износом актива.

Норма амортизации отражает ежегодный процент стоимости актива, подлежащий списанию, и используется для расчёта суммы амортизационных отчислений. Совокупность этих отчислений формирует амортизационный фонд на протяжении всего срока службы имущества.

К основным средствам относятся объекты, используемые в производственной, хозяйственной или управленческой деятельности организации сроком более 12 месяцев. В настоящее время амортизация начисляется на активы, первоначальная стоимость которых составляет не менее 100 000 рублей.

Формула для расчёта амортизации:

На =1/Тн* 100%

Где Тн - нормативный срок службы оборудования,

На - норма амортизации;

Таблица 7.7.1

Расчёт амортизации

Оборудование	Первоначальная стоимость, руб.	Норм. срок службы, лет	На год, %	На мес, %	А.О мес., руб.	∑ А.О, руб.
Ноутбук Realme Book Prime 14	59900	-	-	-	-	-
Принтер Pantum P2502	10500	-	-	-	-	-
Итого	**70400**					

Так как стоимость ноутбука меньше, чем 100 000 рублей, то амортизация не рассчитывается.

7.8 Расчет сметы затрат

Смета затрат представляет собой совокупность всех расходов, понесённых компанией в ходе разработки и продвижения своей продукции. Она играет ключевую роль в управленческом учёте и служит основой для формирования цен на товары и услуги.

Уровень себестоимости оказывает влияние на:

- рентабельность компании;
- прибыль организации.

Таблица 7.7.1

Расчёт сметы затрат

№	Статьи затрат	Сумма, руб.
1	Основные и вспомогательные материалы	2290
2	Электроэнергия	661,323
4	Основная заработная плата (зарплата веб-программисту)	143213,728
5	Отчисления во внебюджетные фонды (страховые выплаты)	43250,545
6	Управленческие расходы	10000
	Себестоимость	**199415,596**

Необходим получить прибыль от разработки информационного продукта, которая будет равняться 20% от полученной суммы себестоимости. Поэтому, стоимость разработки системы будет составлять:

Прибыль = ТехСебС + П%

Где ТехСебС – технологическая себестоимость; П% - процент прибыли

Прибыль = 199415,596 * 0,2 = 39883,119 (руб).

Необходимо выделить налог на добавленную стоимость (НДС), с 1 января 2019, НДС в России стал официально равняться 20%.

НДС = (ТехСебС+Прибыль)*НДС (%)

НДС = (199415,596 + 39883,119)*0,2 = 47859,743 (руб).

СуммаСНДС = ТехСебС + Прибыль + НДС

СуммаСНДС = 199415,596 + 39883,119 + 47859,743 = 287158,458 (руб).

7.9 Расчёт дохода от использования системы

Доход — это совокупная сумма поступивших за определённый период денежных средств, включая поступления не только от основного вида деятельности.

Доходы от использования разработанной системы формируются за счёт сокращения времени работы логиста и аналитика, занимающихся построением маршрутов доставки и анализом эффективности пунктов.

Средняя заработная плата логиста составляет 76 048 рублей в месяц, что соответствует 475,3 руб./час (при 160 рабочих часах в месяц).

До внедрения системы логист вручную строил один маршрут за 5 минут. В день он обрабатывал до 50 маршрутов, при 20 рабочих днях в месяц:

5*50*20= 5000 минут или 83,33 часа.

После внедрения веб-приложения маршруты строятся мгновенно, участие логиста не требуется.

Экономия = 83,33 * 475,3 = 39589,17 (руб/мес) или 475070,04 рублей в год. В квартал: 118767,51 рублей.

Средняя зарплата аналитика составляет 85 264 руб./мес, что эквивалентно 532,9 руб./час (при 160 рабочих часах в месяц).

До внедрения системы аналитик тратил в среднем 3 часа в день на сбор, обработку и визуализацию данных по эффективности пунктов доставки.

После внедрения системы аналитика получает автоматически сформированные графики и таблицы. На их анализ уходит не более 30 минут в день

ЭВ – экономия времени при подсчёте доходов и расходов по проектам, начислению зарплаты сотрудников и распределения её на проекты.

ЭВ = 3 * 60 – 0,5 * 60 = 180 – 30 = 150 минут или 2,5 часа.

Экономия за месяц составит:

20*2,5 = 50 часов в месяц.

Экономия за квартал:

50 * 3 = 150 часов.

Таким образом выходит, что экономия за 3 месяца составляет:

532,9 * 150 = 79935 руб.

Если сложить экономию за три месяца от логиста и аналитика, получается:

118767,51 + 79935 **= 198702,51 руб.**

7.10 Расчет срока окупаемости

Окупаемость проекта — это период времени, за который чистая прибыль от реализации проекта (доходы за вычетом всех затрат и вложений) компенсирует первоначальные инвестиции. Этот показатель, как правило, выражается в месяцах или годах и позволяет оценить эффективность вложений.

Выгода за 1 месяц составит:

Выгода = 198702,51 / 3 = 66234,17 руб.

Экономия за период, равный одному году, составляет:

Экономия = 66234,17 * 12 = 794810,04 руб.

Таким образом, период окупаемости:

Ток = Sпп / ΔЭ,

Где Sпп – стоимость программного продукта;

ΔЭ – экономия средств от внедрения;

Ток = 287158,458 / 66234,17 = 4,3 (месяца).

7.11 Выводы

В экономической части дипломного проекта произведен расчёт себестоимости веб-приложения для службы доставки. В расчет себестоимости вошли следующие затраты:

- Заработная плата программиста за два месяца — 143213,728 рублей.
- Отчисления страховых взносов за два месяца — 43250,545 рублей.
- Стоимость материалов — 2290 рублей.
- Расходы на электроэнергию за два месяца — 661,323 рубля.
- Управленческие расходы — 10 000 рублей.

Общая стоимость разработки составила 287158,458 рублей.

Разработанное веб-приложение окупится примерно через 4,3 месяца. Результаты проведенных расчётов показывают – проектируемая информационная система является экономически обоснованной и рентабельной.

Заключение

В рамках данного дипломного проекта было разработано веб-приложение для службы доставки, обеспечивающее эффективное взаимодействие между клиентами, управляющими пунктов доставки и администраторами системы. Реализованы ключевые функции: создание и отслеживание заказов клиентами, приём и отправка посылок в пунктах доставки, а также получение статистики и аналитики администраторами. Система включает в себя как клиентскую, так и серверную части, с использованием современных технологий веб-разработки.

Перспективы развития:

- Создание мобильного приложения для клиентов и сотрудников.
- Оптимизация маршрутов доставки с использованием алгоритмов машинного обучения.
- Расширение функциональности личного кабинета и интеграция с внешними службами логистики.

Список использованных источников

1. Богачев А.В. Node.js. Разработка серверных веб-приложений. — СПб.: Питер, 2021. — 352 с.

2. Дакетт Дж. HTML и CSS. Разработка и дизайн веб-сайтов. — М.: Эксмо, 2020. — 480 с.

3. Евсеев Д.А., Трофимов В.В. Web-дизайн в примерах и задачах: учебное пособие. — М.: КноРус, 2018. — 263 с.

4. Кузнецов М.В., Симдянов И.В. PHP 8. Профессиональное программирование. — СПб.: БХВ-Петербург, 2022. — 1088 с.

5. Кумскова И.А. Базы данных: учебник. — М.: КноРус, 2019. — 488 с

6. Фриман Э., Робсон Э. Изучаем HTML, XHTML и CSS. — СПб.: Питер, 2017. — 608 с.

7. Ant Design — UI библиотека компонентов — https://ant.design

8. OpenStreetMap. Географические данные и координаты — https://www.openstreetmap.org

9. React документация — https://reactjs.org

10. Sequelize ORM документация — https://sequelize.org