

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет

вычислительной техники

Кафедра

МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Серверное приложение для логистического
сервиса**

Студент

(подпись, дата)

Антонов Илья Игоревич

(ФИО полностью)

Руководитель

(подпись, дата)

Балашова И.Ю.

(фамилия, инициалы)

Нормоконтролер

(подпись, дата)

Такташкин Д.В.

(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

Козлов А.Ю.

(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.

(фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент _____ Антонов Илья Иговерич _____ гр. 21ВП2 факультета _____ ВТ _____
<Фамилия, имя, отчество полностью>

2. Тема работы _____ Серверное приложение
_____ для логистического сервиса

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

3. Руководитель работы _____ к.т.н., доцент, доцент кафедры «МОиПЭВМ» Балашова И.Ю. _____

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

1) Назначение: серверная часть логистического сервиса

2) Среда выполнения: Windows 10, Linux, Mac OS.

Языковые средства разработки: GoLang, SQL, UML.

Инструментальные средства: JetBrains GoLand, Swagger, Docker, PostgreSQL, Draw.io, Kafka, Redis.

Технологии разработки: функциональная парадигма, микросервисная архитектура

3) Основные функции приложения:

а) регистрация и авторизация;

б) добавление, изменение, удаление продукции со склада;

в) выбор продукции из каталога;

г) оформление заказов на доставку продукции;

д) отслеживание доставки в реальном времени;

е) расчет кратчайшего пути доставки между несколькими точками.

ж) оформление отчетов о продажах и покупках

5. Перечень подлежащих разработке вопросов:

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

5. Календарный график выполнения работы

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

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 120 лист, 28 рисунков, 12 таблиц, 20 использованных источников и 10 приложений.

ЛОГИСТИЧЕСКИЙ СЕРВИС, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, GOLANG, TYPESCRIPT, UML, WEB-ПРИЛОЖЕНИЕ, БАЗА ДАННЫХ, POSTGRESQL, ЗАДАЧА КОММИВОЯЖЕРА, ТРАСПОРТНАЯ ЗАДАЧА.

Объект исследования – сервисы обеспечения товарных и логистических услуг для различных клиентов.

Цель выпускной квалификационной работы – спроектировать и разработать программное обеспечение, организующее логистические процессы с нескольких сторон: со стороны пользователя сервиса и со стороны транспортной компании, организующей перевозку товара.

Технологии разработки – реализация программных средств для серверной части сервиса ведется в среде разработки JetBrains GoLand 2025.1 на языке программирования Golang. Конструирование базы данных производится в СУБД PostgreSQL 16. Развертывание приложения производится посредством контейнеризации в системе Docker на операционной системе Linux Ubuntu 22.04.

Среда исполнения – персональный компьютер пользователя.

Результаты работы – в ходе выполнения выпускной квалификационной работы была спроектирована и разработана серверная часть сервиса, моделирующего логистические процессы каждой участвующей стороны.

					ПГУ 09.03.04 – 08BP212.03 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Серверное приложение для логистического сервиса. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Антонов И. И.					4	120
Пров.		Балашова И.Ю.				Группа 21ВП2		
Реценз.								
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение	7
1 Анализ предметной области и постановка задачи на разработку программного обеспечения логистического сервиса.....	9
1.1 Анализ предметной области приложения	9
1.2 Анализ алгоритмов решения транспортной задачи.....	10
1.3 Анализ аналогичных программных средств	12
1.4 Постановка задачи на разработку серверной части приложения.....	17
1.4.1 Цели и задачи приложения	17
1.4.2 Классы пользователей логистического сервиса	18
1.5 Анализ требований	19
1.5.1 Анализ функциональных требований.....	19
1.5.2 Анализ нефункциональных требований.....	22
1.6 Планирование разработки и оценка бюджета	23
2 Проектирование серверной части логистического сервиса	25
2.1 Архитектура приложения.....	25
2.2 Проектирование структур данных	28
2.2.1 Концептуальная модель баз данных	28
2.2.2 Логическая модель баз данных	31
2.3 Обобщенная структура разрабатываемого сервиса.....	35
2.4 Требования к аппаратному обеспечению серверной части	36
3 Разработка программного обеспечения серверной части	38
3.1 Разработка архитектуры серверной части	38
3.2 Разработка физической модели базы данных	41
3.3 Разработка архитектурных слоев внутри микросервисов.....	44
3.3.1 Разработка слоя представления данных в микросервисах	45
3.3.2 Разработка слоя бизнес-логики в микросервисах.....	47
3.3.3 Разработка слоя обращения к базе данных в микросервисах.....	50
4 Контроль качества программного обеспечения	54
4.1 Обоснование выбора модели тестирования и метрик кода	54

4.2	Тестирование серверной части логистического сервиса	55
4.2.1	Функциональное тестирование добавления новой продукции.....	55
4.2.2	Модульное тестирование метода обновления заявки	59
4.2.3	Метрики Холстеда для создания нового заказа.....	60
	Заключение.....	63
	Список использованных источников.....	65
	Приложение А (справочное) Глоссарий.....	68
	Приложение Б (обязательное) Объекты обращения к базам данных.....	70
	Приложение В (обязательное) SQL-скрипты	74
	Приложение Г (обязательное) Структуры слоя представления данных.....	80
	Приложение Д (обязательное) Структуры слоя бизнес-логики.....	85
	Приложение Е (обязательное) Структуры слоя обращения к базе данных.....	91
	Приложение Ж (обязательное) Листинг модульных тестов	96
	Приложение И (рекомендуемое) Спецификация атрибутов таблиц	101
	Приложение К (рекомендуемое) Иллюстрационные материалы к тестированию API.....	107
	Приложение Л (рекомендуемое) Графическая часть	118

Введение

В современных условиях стремительного развития электронной коммерции наблюдается значительный рост объема грузоперевозок, что создает повышенную потребность в эффективных логистических сервисах [1]. Логистический сервис – это система обеспечения, позволяющая покупателю выбрать оптимальный вариант приобретения и потребления товара. Сервис неразрывно связан с процессом продажи и представляет собой комплекс услуг, оказываемых в процессе заказа, поставки покупки и дальнейшего обслуживания продукции [2]. Ключевой задачей транспортной логистики является построение оптимальных маршрутов доставки грузов, минимизирующих затраты на транспортировку. С решением этой задачи доставка товара в настоящее время может занимать не несколько недель, а сводиться к нескольким часам, позволяя клиентам получать необходимый товар непосредственно в день его заказа. Для нахождения решения этих задач зачастую применяются алгоритмы поиска и оптимизации маршрутов. С помощью этих алгоритмов можно планировать использование транспортных средств и других ресурсов при организации доставки грузов. Основным критерием выбора алгоритма в этой задаче является скорость выполнения. Алгоритм должен быстро находить оптимальный маршрут, адаптируясь к изменениям, например, добавлению новых заказов, и быть устойчивым при заданном числе логистических узлов. В контексте решения транспортной задачи для проведения логистических процессов неправильный выбор алгоритма может привести к усложнению маршрута, что уже сильно сказывается на затраченных времени и ресурсах [1].

Целью данной ВКР является проектирование и разработка сервиса для проведения логистических процессов, начиная от выбора пользователем интересующего товара и заканчивая организацией и расчетом оптимального маршрута для доставки между несколькими клиентами одновременно. Программное средство должно предоставлять гибкий и удобный интерфейс для взаимодействия со всеми разделами: клиентским (товарным) и транспортным. Основная задача ВКР – разработка серверной части логистического сервиса,

обеспечивающая все управление товарами и пользователями, а также построение и поиск оптимальных маршрутов на основе информации о доставке из заказов потребителей. В рамках данной работы планируется создать систему, способную обрабатывать большие объемы данных в реальном времени для построения оптимальных маршрутов доставки за минимальное затраченное время. Система должна включать в себя модули управления складом, обработки заказов, мониторинга доставки, управления пользователями и осуществления различных транзакций, что позволит создать полноценное решение для автоматизации процессов современно логистики.

1 Анализ предметной области и постановка задачи на разработку программного обеспечения логистического сервиса

1.1 Анализ предметной области приложения

В основе предметной области выпускной квалификационной работы лежит транспортная логистика и все объекты и процессы к ней причастные. Рассмотрим основные термины и понятия выбранной предметной области, чтобы погрузиться в нее более детальнее.

Логистика – наука о организации, планировании, контроле и регулировании движения материальных и информационных потоков в пространстве и во времени от их первичного источника до конечного потребителя.

Материальный поток – это продукция, перемещаемая от производства к потребителю за определенный период времени.

Информационный поток – это сообщения и документы, сопровождающие материальный поток и необходимые для управления [3].

Транспортная логистика – это та часть логистической науки, которая отвечает за организацию перевозки грузов в определенное место и время, заранее обговоренное с заказчиком, по наиболее выгодному и оптимальному пути, специально разработанным, чтобы уменьшить все издержки [4].

Логистическая цепь – это линейно упорядоченное множество отдельных элементов логистической системы (физических и юридических лиц, производителей, дистрибьюторов, дилеров, складов, транспорта и т. д.), осуществляющих по ходу технологического процесса логистические операции по продвижению товара от одной системы к другой, от производителя к потребителю.

Логистическая операция – действие, преобразующее материальный или информационный поток.

Логистический сервис – это система обеспечения, позволяющая покупателю выбрать оптимальный вариант приобретения и потребления товара.

Сервис неразрывно связан с процессом продажи и представляет собой комплекс услуг, оказываемых в процессе заказа, поставки, покупки и дальнейшего обслуживания продукции [2].

Таким образом, программные средства логистических сервисов разрабатываются для различных групп пользователей: от клиентов, желающих заказать тот или иной доступный в сервисе товар, до различных менеджеров, занимающиеся поставкой и продажей этого товара для клиентов – с целью облегчить, автоматизировать и оптимизировать логистические процессы в условиях постоянного изменения материальных и информационных потоков.

1.2 Анализ алгоритмов решения транспортной задачи

Ключевой задачей подобных сервисов является нахождение оптимального решения задачи коммивояжера для транспортной логистики при построении маршрута, затрачивающего минимальное количество как человеческих и материальных, так и технологических ресурсов.

В разрабатываемом логистическом сервисе маршруты строятся на основе взвешенного графа транспортной сети, где вершины представляют склады и магазины, а ребра – пути между ними, каждое ребро имеет вес, например, расстояние. Для рассматриваемой системы, где число складов и магазинов невелико (до 15 узлов), эффективными будут алгоритмы оптимизации маршрута, не требующие значительных ресурсов для обработки данных, но обеспечивающие высокую скорость вычислений и адаптацию к изменениям. Основным критерием выбора алгоритма в этой задаче является скорость выполнения. Алгоритм должен быстро находить оптимальный маршрут, адаптируясь к изменениям, например, добавлению новых заказов, и быть устойчивым при малом числе узлов. По заданным параметрам системы были выбраны следующие алгоритмы для анализа и дальнейшего выбора конкретного решения:

- алгоритм ближайшего соседа;
- жадный алгоритм;

- генетический алгоритм;
- алгоритм ветвей и границ.

Для сравнения выбранных алгоритмов были проведены вычислительные эксперименты для $N = 3, 5, 7, 9, 11, 13, 15, 17$, где N – количество точек маршрута. В качестве параметров сравнения были выбраны 2 показателя:

- $L(\text{км})$ – длина построенного маршрута;
- $T(\text{с})$ – время вычисления маршрута алгоритмом.

Все эксперименты проводились на ноутбуке Macbook Pro 14 с процессором M3 Pro под управлением операционной системы MacOS 15.5.

Таблица 1 – Результаты проведения экспериментов для анализа

N	Алгоритм ближайшего соседа		Жадный алгоритм		Генетический алгоритм		Алгоритм ветвей и границ	
	L(км)	T(с)	L(км)	T(с)	L(км)	T(с)	L(км)	T(с)
3	178,77136 42	0,0000017 92	178,77136 42	0,0000063 75	178,77136 42	0,0083698 75	178,77136 42	0,0000014 59
5	268,03513 29	0,0000030 83	70,435235 32	0,0002305 83	268,03513 29	0,0107402 08	268,03513 29	0,0000081 67
7	281,21618 4	0,0000042 09	56,562915 88	0,0000155 83	281,21618 4	0,0110823 33	281,21618 4	0,0001211 25
9	332,26833 09	0,0000102 5	64,642455 82	0,0000219 17	312,52854 26	0,0215144 16	312,52854 26	0,0036563 33
11	321,99453 87	0,0000072 91	22,846711 2	0,0000215 41	321,99453 87	0,0204771 67	321,99453 87	0,0443768 75
13	380,40790 96	0,0000099 59	120,55470 79	0,0000259 58	300,50432 26	0,0221273 33	295,84360 51	1,2602916 25
15	350,15110 47	0,0000146 25	81,786679 62	0,0000340 84	284,21102 93	0,0317025 83	276,12551 52	117,65061 87
17	411,68715 07	0,0000177 5	87,213182 41	0,0000414 17	336,97751 59	0,0344284 58	–	–

По итогам проведенных вычислительных экспериментов для анализа алгоритмов решения задачи коммивояжера были получены следующие результаты.

Алгоритм ближайшего соседа и жадный алгоритм обеспечивают высокую скорость выполнения, легко адаптируются к изменениям и требуют минимальных ресурсов. Это делает их подходящими для маршрутов с ограниченным числом узлов. Однако они менее точны при поиске глобально оптимальных решений, чем генетический алгоритм и алгоритм ветвей и границ, что может стать критичным в сложных условиях маршрутизации.

Генетический алгоритм обеспечивает высокую оптимальность маршрутов и хорошую адаптацию к изменениям. Однако высокая вычислительная сложность и потребность в значительных ресурсах делают его менее подходящим для данной системы с небольшим количеством узлов.

Алгоритм ветвей и границ обеспечивает максимальную точность маршрута, но его вычислительная ресурсоемкость снижает эффективность для малых сетей, что делает его неоптимальным выбором в данных условиях.

Таким образом, для поставленной задачи наиболее подходят алгоритм ближайшего соседа и жадный алгоритм, поскольку они в случае схемы с небольшим числом магазинов обеспечивают наиболее быструю маршрутизацию и простую интеграцию в систему.

1.3 Анализ аналогичных программных средств

Проанализировав рынок альтернатив, схожих по функциональности на разрабатываемое в рамках выпускной квалификационной работы приложение, были выделены следующие платформы для проведения сравнения: СДЭК, DNL Express, Яндекс Маршрутизация, СберЛогистика. В таблице 2 представлено краткое описание общих сведений о выделенных аналогах логистических платформ.

Таблица 2 – Краткая описание общих сведений об аналогах

Наименование	Производитель	Описание
СДЭК	ООО «СДЭК-Глобал»	Логистический оператор, предоставляет комплекс логистических услуг в соответствии с современными международными стандартами и требованиями. Позволяют контролировать прохождение грузов на всех этапах, существенно сокращать сроки доставки, выбирать самые надежные и проверенные маршруты и совершенствовать логистические схемы на больших расстояниях

Продолжение таблицы 2

Наименование	Производитель	Описание
DHL Express	ДХЛ Интернешнл	Предлагает широкий выбор логистических продуктов и услуг: от простой доставки и транспортировки до инновационных решений по организации логистической цепочки
Яндекс Маршрутизация	ООО «Яндекс»	Комплексная логистическая система, состоящая из двух основных сервисов: Маршрутизация и Мониторинг, которая охватывает все бизнес-задачи, связанные с логистикой. Сервис распределяет заказы между исполнителями и разрабатывает для них оптимальные маршруты.
СберЛогистика	ООО «СБЕРЛОГИСТИКА»	Агрегатор служб доставки для интернет-магазинов.

Каждое из представленных приложений включает в себя различный функционал и возможности для взаимодействия с сервисом. Анализ и сравнение с аналогами будет включать в себя выделение плюсов и минусов каждого из выбранных программных средств.

СДЭК – курьерская доставка грузов и документов для организаций и частных лиц по России и миру. Успешно работают на рынке курьерских услуг и логистики с 2000 года. Графический интерфейс сервиса СДЭК представлена на рисунке 1.

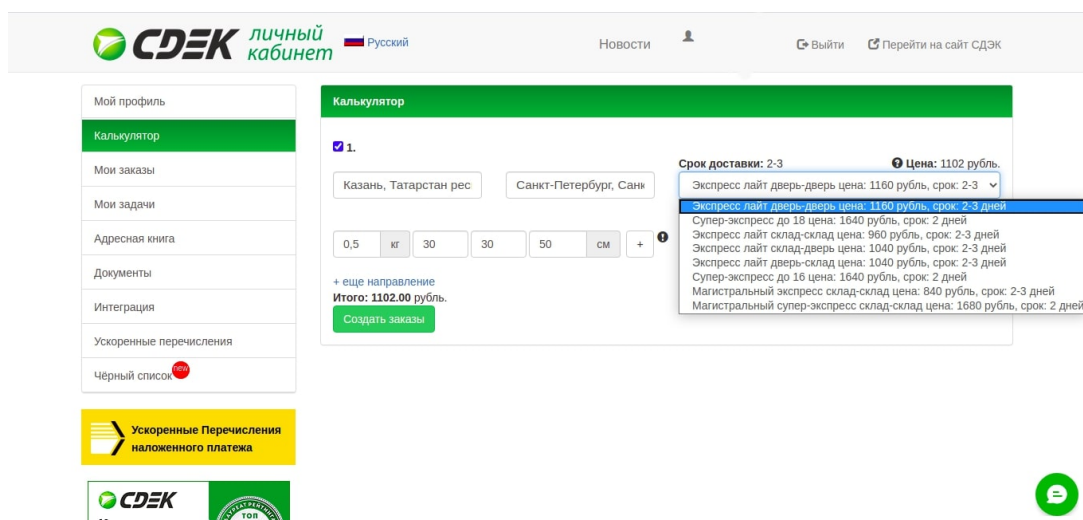


Рисунок 1 – Интерфейс сервиса СДЭК

К основным плюсам СДЭКА можно отнести:

- удобное онлайн-слежение доставок товара;
- автоматический расчет стоимости доставки товара с помощью встроенного калькулятора;
- возможность отправки грузов различных габаритов.

К минусам можно отнести:

- неоптимальная стоимость услуг логистики;
- относительно старый и нагруженный интерфейс сервиса.

DHL Express – один из лидеров международного рынка логистики.

Графический интерфейс сервиса представлен на рисунке 2.

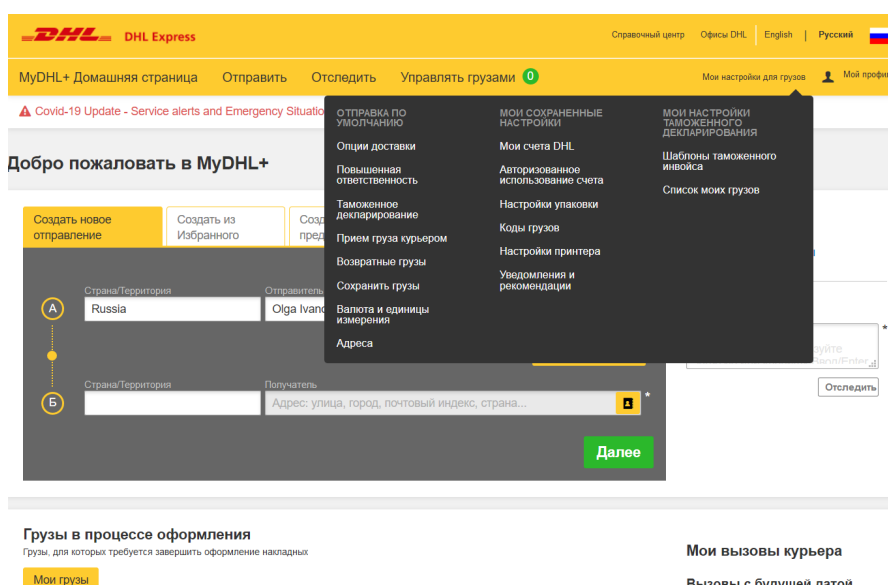


Рисунок 2 – Интерфейс сервиса DHL

Основные преимущества DHL Express:

- возможность формирования отчетов и адресных книг;
- удобный и ненагруженный интерфейс.

К основным недостаткам DHL Express можно отнести:

- высокая стоимость оказываемых сервисом услуг;
- строгие ограничения по весу и габаритам товара.

Яндекс Маршрутизация – логистическая платформа для планирования оптимальных маршрутов и контроля за их выполнением в режиме реального времени. Графический интерфейс сервиса представлен на рисунке 3.

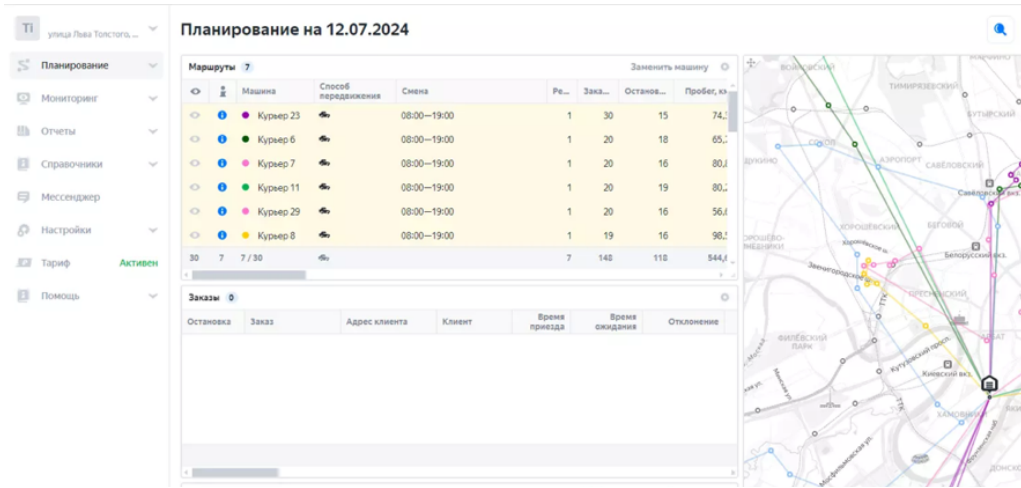


Рисунок 3 – Интерфейс сервиса Яндекс Маршрутизация

К основным преимуществам Яндекс Маршрутизации можно отнести:

- интеграция с картографическими сервисами для гибкого планирования маршрутов доставки;
- современный программный интерфейс.

Основными недостатками Яндекс Маршрутизации являются:

- высокая стоимость услуг сервиса;
- перегруженный функционал, что делает порог входы высоким для новичков.

СберЛогистика предоставляет низкие тарифы на экспресс-доставку в РФ и по миру и отслеживание посылок в личном кабинете на сайте или через API. Интерфейс сервиса представлен на рисунке 4.

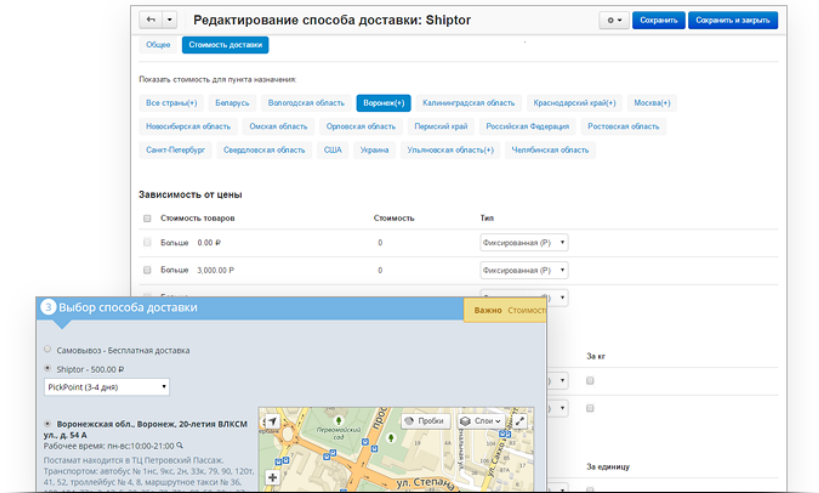


Рисунок 4 – Интерфейс «СберЛогистика»

К основным преимуществам «СберЛогистика» можно отнести:

- интеграция с экосистемой Сбера;
- отслеживание грузоперевозок в реальном времени.

Основными недостатками «СберЛогистика» являются:

- относительно старый и негибкий интерфейс сервиса;
- сильные ограничения на проведение грузоперевозок в некоторые регионы.

Для проведения сравнительного анализа было выведено несколько критериев для оценивания аналогов с разрабатываемым приложением. Для оценки каждого из критериев была предложена следующая система оценивания:

- 0 баллов – полностью не соответствует критерию;
- 1 балл – частично соответствует критерию;
- 2 балла – полностью соответствует критерию.

Результаты проведение сравнительного анализа приведены в таблице 3.

Таблица 3 – Сравнительный анализ аналогичных программных средств

Критерии	СДЭК	DHL Express	Яндекс Маршрутизация	СберЛогистика
Осуществление заказов доступных в сервисе товаров	2	0	0	0
Высокая скорость работы	1	2	1	1
Создание и просмотр документов отчетности	0	1	0	1
Отсутствие ограничений по объему работы	1	2	1	1
Взаимодействие с другими системами	0	1	2	1
Итого	6	6	6	5

Таким образом, по итогам сравнительного анализа можно сделать вывод о том, что рассмотренные сервисы предоставляют схожий функционал и обладают как рядом преимуществ, так и недостатков. Однако ни один из аналогов в полной мере не соответствует поставленным для сервиса критериям. Поэтому необходимо разработать сервис, который будет предоставлять следующий функционал:

- удобно взаимодействовать с товарами в сервисе;
- отслеживать доставку товаров в реальном времени;

- гибкую настройку необходимого пути доставки товара;
- генерировать отчеты о продажах/покупках товара;
- обеспечивать высокую скорость работы в сервисе.

1.4 Постановка задачи на разработку серверной части приложения

1.4.1 Цели и задачи приложения

Основой целью выпускной квалификационной работы является разработка серверной части программного обеспечения логистического сервиса. Основной его функцией должен быть поиск и расчет оптимального транспортного маршрута на основе данных о доставке заказов товаров пользователей. Кроме того, разрабатываемое приложение должно предоставлять пользователю удобный и гибкий интерфейс взаимодействия с доступными товарами, заказами и построенными логистическими маршрутами.

К основным задачам, которые должны решаться при разработке приложения, можно отнести следующие:

- авторизация и регистрация пользователей;
- осуществление заказов доступных в сервисе товаров;
- добавление, удаление и изменение товаров в сервисе;
- анализ осуществленных пользователями заказов и построение на их основе маршрута доставки товара;
- просмотр и формирование отчетов о совершенных заказах того или иного товара;
- предоставление возможности для связи с клиентской частью сервиса;
- организация связи серверной части с базой данных приложения.

В результате выполнения этих задач необходимо получить систему, которая будет состоять из микросервисов, работающих по технологии REST API и Kafka, а также иметь возможность сохранять необходимые объекты в базу данных.

1.4.2 Классы пользователей логистического сервиса

Для разрабатываемого логистического сервиса были выделены 3 класса пользователей: администратор, производитель, потребитель. Подробное описание их бизнес-процессов и с чем они взаимодействуют представлено в таблице 4.

Таблица 4 – Классы пользователей приложения

№	Наименование класса	Затрагиваемая часть сервиса	Бизнес-процессы	Используемые объекты
1	Администратор	Приложение администратора	Отслеживание и управление доставками	Товары, Заказы
			Утверждение новых пользователей и товаров	Товары, Пользователи, Клиенты
			Управление всеми пользователями	Пользователи, Клиенты
2	Потребитель	Приложение потребителя	Авторизация и регистрация	Пользователи
			Просмотр всех доступных товаров	Товары
			Оформление заказов	Заказы, Фин. операции
			Отслеживание собственных заказов	Заказы
			Получения отчета о своих заказах	Заказы, Фин. операции
3	Поставщик	Приложение поставщика	Авторизация и регистрация	Пользователи
			Управление собственной продукцией	Товары
			Получение отчета о проданной продукции	Заказы, Фин. операции

Разрабатываемое приложение логистического сервиса должно содержать в себе все описанные группы пользователей и реализовывать все их бизнес-процессы.

1.5 Анализ требований

1.5.1 Анализ функциональных требований

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе. Важно понимать, что диаграммы вариантов использования не предназначены для отображения проекта и не могут описывать внутреннее устройство системы [5]. Таким образом, диаграмма вариантов использования помогает верхнеуровнево отобразить связи между актерами и необходимыми сценариями для работы приложения. Подобные диаграммы говорят о том, что разрабатываемая система должна выполнять, а не как именно это будет реализовано. Это позволяет на ранних этапах проектирования определить основные функции системы и взаимодействие с пользователями.

Во время анализа функциональных требования логистического сервиса для выпускной квалификационной работы была разработана диаграмма вариантов использования, отражающая следующие основные сценарии работы сервиса:

- войти в систему;
- выйти из системы;
- управлять продукцией;
- управлять пользователями;
- управлять клиентами;
- управлять доставками;
- перейти в корзину;
- оформить заказ;
- сгенерировать отчет;
- получать уведомления на почту.

Каждый из этих сценариев представляет собой отдельный вариант использования, который может быть детализирован в дальнейшем процессе разработки.

Диаграмма вариантов использования представлена на рисунке 5.

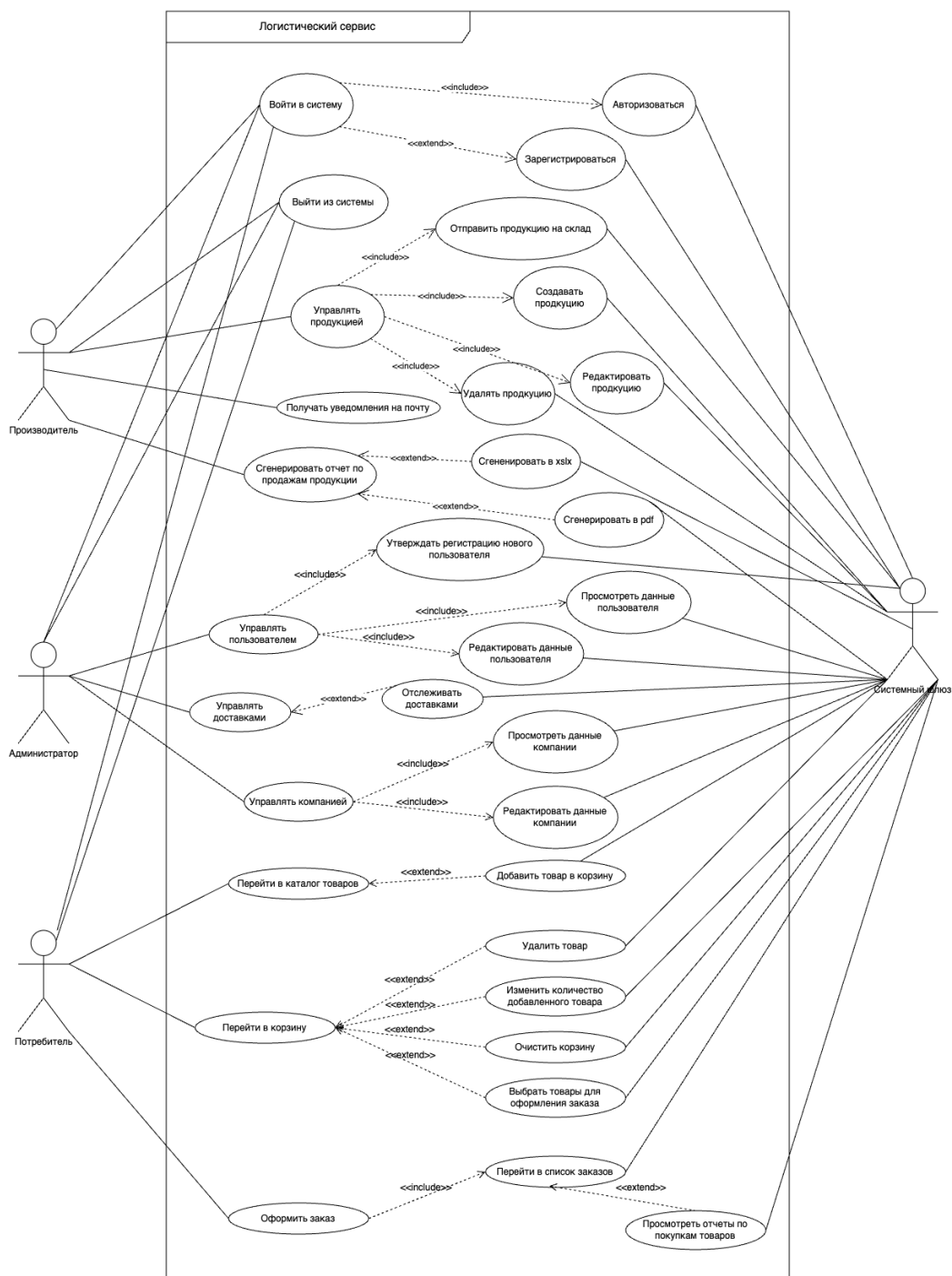


Рисунок 5 – Диаграмма вариантов использования логистического сервиса

Основным действующим лицом каждого из разделов сервиса является пользователь, относящийся к своему классу пользователей. На рисунках ниже описаны наиболее важные бизнес-процессы, относящиеся к каждой их групп пользователей.

На рисунке 6 представлена спецификация варианта использования «Войти в систему».

Прецедент: Войти в систему
ID: 1
Краткое описание: Пользователь регистрируется или авторизуется в приложении
Главные актеры: Пользователь любой группы
Второстепенные актеры: Нет
Предусловия: 1. Пользователь не был зарегистрирован/авторизован
Основной поток: 1. Пользователь переходит на страницу входа в приложение 2. Пользователь вводит данные для входа в существующий аккаунт (логин и пароль) 3. Пользователь получает доступ к функционалу приложения
Постусловия: 1. Пользователь вошел в приложение, ему доступен функционал
Альтернативные потоки: 2.а. Пользователь вводит данные для регистрации (логин и пароль, роль (поставщик или покупатель)) 2.б. Пользователь переходит к форме заполнения дополнительной информации о себе

Рисунок 6 – Спецификация варианта использования «Войти в систему»

На рисунке 7 представлена спецификация варианта использования «Управлять продукцией».

Прецедент: Управлять продукцией
ID: 2
Краткое описание: Пользователь может создавать, редактировать и удалять продукцию
Главные актеры: Пользователь класса производитель
Второстепенные актеры: Нет
Предусловия: 1. Пользователь прошел авторизацию
Основной поток: 1. Пользователь открывает форму по продукту 2. Пользователь заполняет все обязательные поля 3. Пользователь создает продукт
Постусловия: 1. Был создан новый продукт или отредактирован/удален существующий
Альтернативные потоки: 3а. Пользователь редактирует продукт 3б. Пользователь удаляет продукт

Рисунок 7 – Спецификация варианта использования «Управлять продукцией»

На рисунке 8 представлена спецификация варианта использования «Оформить заказ».

Прецедент: Оформить заказ
ID: 3
Краткое описание: Пользователь оформляет заказ на товары, находящиеся в корзине
Главные актеры: Пользователь класса покупатель
Второстепенные актеры: Нет
Предусловия: 1. Пользователь прошел авторизацию
Основной поток: 1. Пользователь переходит в корзину 2. Пользователь нажимает кнопку "Оформить заказ" 3. Пользователь заполняет данные для доставки (адрес) 4. Пользователь подтверждает оформление заказа 5. Пользователь получает уведомление об успешно созданном заказе
Постусловия: 1. Заказ успешно оформлен, данные сохранены, пользователь может отслеживать статус заказа
Альтернативные потоки: 2а. Корзина пуста, пользователь получает уведомление о том, что невозможно оформить заказ без товаров 3б. Данные для доставки неполные, пользователь получает сообщение при валидации о том, что нужно дополнить адрес

Рисунок 8 – Спецификация варианта использования «Оформить заказ»

Таким образом, на основе проведенного анализа функциональных требований, был выведен минимальный набор разрабатываемых функций, необходимый для корректной работы всех бизнес-сценариев логистического сервиса.

1.5.2 Анализ нефункциональных требований

При проведении анализа нефункциональных требований к разработке серверной части логистического сервиса для выпускной квалификационной работы, одной из ключевых задач было выбрать правильный архитектурный подход, который обеспечил выполнение всех необходимых требований. Был выделен следующий список требований, которому должно следовать разрабатываемая система:

- гибкая;
- масштабируемая;
- отказоустойчивая;
- производительная.

Такие требования были выведены на основе специфики приложения, так как придется обрабатывать большие массивы данных для построения оптимальных маршрутов доставки с минимальным количеством затраченного на это времени.

Опираясь на все поставленные требования, микросервисная архитектура стала наилучшим решением, благодаря своей модульности и независимости отдельно взятых компонентов системы.

Ниже приведен анализ всех нефункциональных требований, обосновывающий выбранную архитектуру приложения.

Гибкость – микросервисная архитектура, благодаря своей независимости компонентов, предоставляет возможность вносить изменения и добавлять новый функционал без влияния на уже существующий функционал, что упрощает разработку системы.

Масштабируемость и производительность – система должна адаптироваться к росту нагрузки с увеличением количества узлов в итоговом маршруте. Микросервисная архитектура позволяет увеличить ресурсы только сильно нагруженного модуля, что существенно снижает затраты на всю систему.

Отказоустойчивость – в выбранной архитектуре, каждый из разрабатываемых микросервисов работает изолированно друг от друга, что становится существенным плюсом при отказе какой-то одной системы, так как весь остальной сервис доступен и продолжает функционировать в штатном режиме.

1.6 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета являются критически важными этапами для успешной реализации программного продукта [19].

Управление проектом было решено проводить в программе Project Libre, которая позволяет планировать разработку, отслеживать выполнение поставленных задач и контролировать ресурсы на разработку.

После внесения данных о стадиях разработки приложения в программу было осуществлено планирование процесса создания программного обеспечения, а также проведена оценка финансовых затрат. Для наглядного представления хода выполнения работ используется диаграмма Ганта, которая отображает ключевые этапы разработки программного обеспечения.

Диаграмма Ганта – это график работ по проекту, который строится в виде таблицы с этапами и ответственными за их выполнение. Диаграмма состоит из двух осей: вертикальной со списком задач и горизонтальной со сроками [20].

Полученная диаграмма Ганта представлена на рисунке Л.1 в приложении Л.

В ходе планирования работы над проектом была выделена 21 задача. Началом работы была выбрана дата 8 апреля 2025 года. Окончание работы назначено на 13 июня 2025 года, согласно полученной диаграмме.

Список поставленных задач представлен на рисунке Л.2 в приложении Л. Используемые ресурсы отражены на рисунке Л.3 в приложении Л.

Как видно из представленного списка все задачи выполняются одним ресурсом – разработчиком

По итогу временные затраты на разработку серверной части логистического сервиса составили 408 часов, а полная стоимость проекта – 196000 рублей.

2 Проектирование серверной части логистического сервиса

2.1 Архитектура приложения

После проведенного анализа функциональных и нефункциональных требований, поставленных для разрабатываемого программного средства, была выбрана архитектура на основе микросервисов как основа системы. Такая архитектура обладает умеренной сложностью при разработке и прекрасной гибкостью в выборе решения для написания различных частей сервиса, позволяя выбирать наиболее подходящее средство для обеспечения работы каждого необходимого для работы компонента. Все это делает микросервисную архитектуру отличным выбором для бизнес-приложений, где является необходимым разработать распределенную, но тем временем хорошо управляемую архитектуру.

Для лучшего понимания особенностей системы и того, какие компоненты системы необходимы для работы разрабатываемого приложения, была разработана диаграмма компонентов серверной части логистического сервиса. Диаграмма компонентов описывает особенности физического представления системы. Она позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых могут быть как различные сервисы, написанные на различных языках программирования, так и базы данных, используемые для хранения данные приложений [6]. Диаграмма компонентов предоставляет общую картину архитектуры системы, а также помогает лучше понять ее структуру и взаимосвязи. Более того является удобным решением для документирования и коммуникации выбранных архитектурных решений при разработке приложений. Диаграмма компонентов предоставляет общую картину архитектуры системы, а также помогает лучше понять ее структуру и взаимосвязи.

Диаграмма компонентов для серверной части разрабатываемого логистического сервиса представлена на рисунке 9.

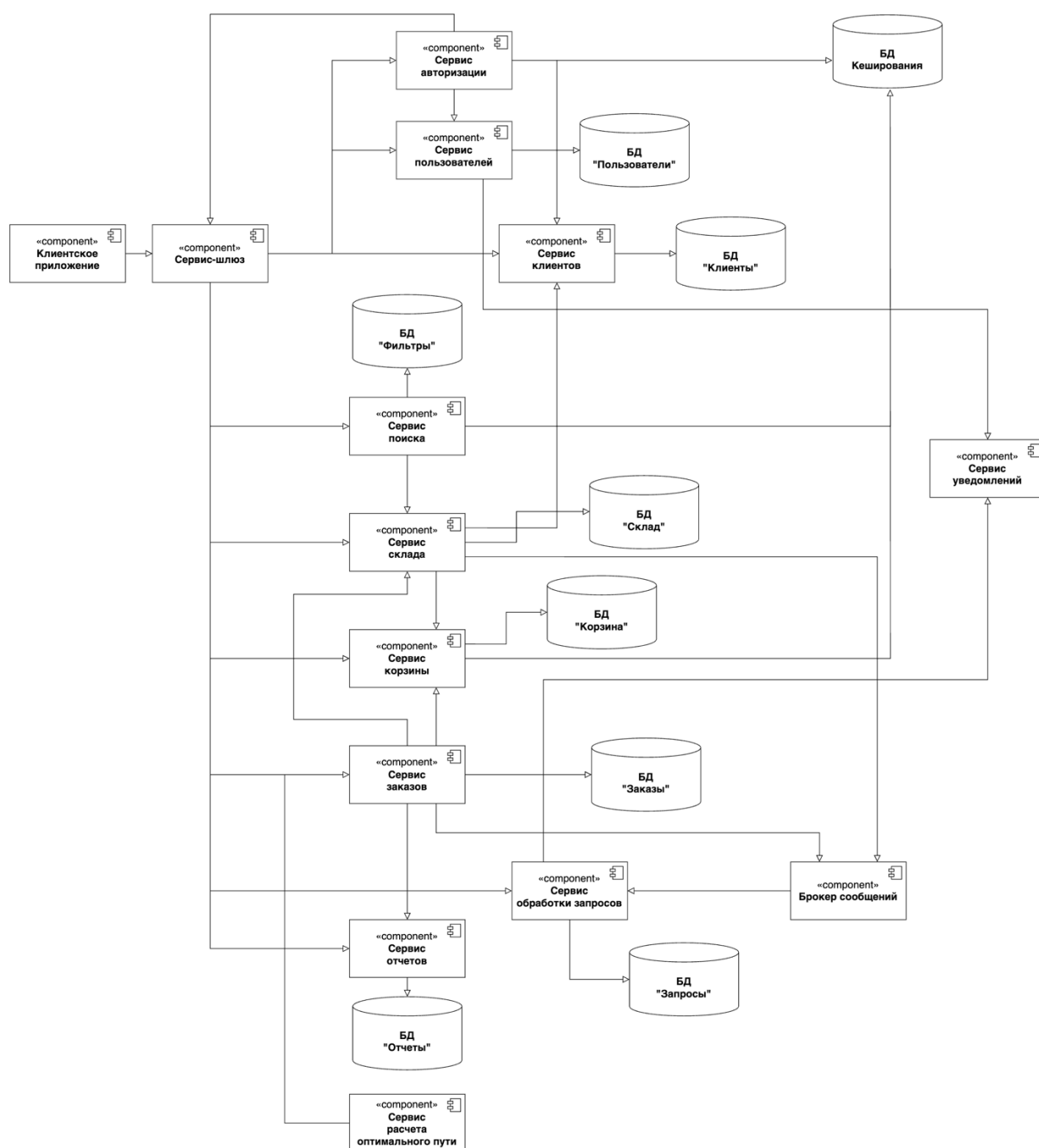


Рисунок 9 – Диаграмма компонентов

Каждый из представленных на диаграмме микросервисов отвечает за свою конкретную предметную область:

- сервис-шлюз – общая точка входа в систему для интеграции с клиентской частью сервиса;
- сервис авторизации – авторизация, аутентификация и регистрация пользователей;
- сервис пользователей – управление пользовательской информацией;

- сервис клиентов – учет информации о клиентах сервиса;
- сервис поиска – формирование каталога товаров для пользователя по выбранным фильтрам;
- сервис склада – управление товарной информацией;
- сервис корзины – управление содержимыми в корзине пользователей товарами;
- сервис заказов – осуществление заказов и проведение финансовых транзакций;
- сервис обработки запросов – управление запросами на регистрацию пользователей, добавление нового товара или поставку недостающего товара на склад;
- сервис отчетов – создание и наполнение различных отчетов;
- сервис расчета оптимального пути – агрегация информации о доставках и расчет оптимального пути на основе выбранного алгоритма решения транспортной задачи;
- сервис уведомлений – отправка и формирование сообщений пользователям, которые отправляются на почту, указанную при регистрации.

Такое разделение на предметные области позволяет максимально гибко взаимодействовать с системой, упрощая процессы модификации и масштабирования разработанного сервиса.

Для организации взаимодействия между микросервисами внутри серверной части используются два подхода: REST API и обмен сообщениями через брокера сообщений Apache Kafka.

REST API – стиль разработки, обеспечивающий простоту и гибкость во всех взаимодействиях. Используется в системе для простого обмена сообщениями в реальном времени, используя следующие глаголы для четкого разграничения выполняемой работы:

- GET для получения данных;
- POST для создания или изменения;
- PUT для обновления;

– DELETE для удаления.

Apache Kafka в свою очередь используется для асинхронного выполнения различных операций, например отправка сообщений пользователям или создание новых запросов при добавлении пользователя/товара в систему. Это позволяет не останавливать выполнение операций, выполняемых в основном потоке приложения, делая работу приложения более быстрой. Сама Kafka организует данные в разделы (topics) с упорядочением сообщений, что позволяет производителям (producers) отправлять данные, а потребителям (consumers) – считывать их в нужной последовательности [7].

Более того, для ускорения работы некоторых частей сервиса и временного хранения информации используется база данных ключ-значение Redis (БД кеширования на диаграмме компонентов). Ее использование позволяет меньше нагружать основную базу данных приложения, кешируя большие и частые запросы за информацией, которая не часто обновляется.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель баз данных

Концептуальная модель – это модель, отображающая данные и связи предметной области с учетом их семантического содержания, независимо от представления в компьютере. Такая модель представления данных строится для отображения вида хранения данных в общем виде и лучшего понимания какие данные необходимы для хранения в системе [8].

Концептуальная модель позволяет структурировать информацию на высоком уровне абстракции, что упрощает процесс проектирования базы данных. Она помогает выявить ключевые сущности, их атрибуты и взаимосвязи, что особенно важно на начальных этапах разработки системы

Согласно всем определенным требованиям по организации хранения и управления данными для серверной части разрабатываемого логистического сервиса была разработана диаграмма типа «сущность-связь» концептуальной модели базы данных. Диаграмма представлена на рисунке 10.



Рисунок 10 – Концептуальная модель БД

Диаграмма, разработанная в процессе проектирования концептуальной модели базы данных, наглядно отображает все сущности и связи между ними. В таблице 5 представлено описание сущностей, которые были выделены для организации работы бизнес-процессов логистического сервиса.

Таблица 5 – Описание сущностей

Название сущности	Идентификатор	Назначение
Роль	role	Хранит сведения о доступных в системе ролях пользователей
Пользователь	user	Хранит сведения о имеющихся пользователях
Клиент	client	Хранит сведения о имеющихся клиентах сервиса
Продукт	product	Хранит общую информации о продукте, добавленных в сервис
Модель продукта	product_item	Хранит более детальную информацию о каждой модели существующего продукта
Корзина	cart	Хранит информацию о корзине пользователя
Содержимое корзины	cart_item	Хранит информацию о содержимом корзины пользователя
Заказ	order	Хранит информацию о совершенных в сервисе заказах
Содержимое заказа	order_item	Хранит сведения о содержимом проведенных заказов
Финансовая операция	operation	Хранит общую информацию о совершенных операциях
Деталь операции	operation_detail	Хранит детали совершенных операций
Заявка	request	Хранит информацию о пришедших заявках, возможных в системе

В таблице 6 представлено описание всех возможных связей между выделенными сущностями

Таблица 6 – Описание связей между сущностями

Название связи	Описание	Мощность
role-user	Определяет принадлежность роли к пользователю	1:1
user-client	Определяет принадлежность клиента к пользователю	1:1
user-cart	Определяет принадлежность пользователя к корзине	1:1
user-order	Определяет принадлежность пользователя к заказу	1:M

Продолжение таблицы 6

Название связи	Описание	Мощность
cart-cart_item	Определяет принадлежность корзины и ее содержимого	1:M
order-order_item	Определяет принадлежность заказа и его содержимого	1:M
client-product	Определяет принадлежность клиента к добавленному им продукту	1:M
product-product_item	Определяет принадлежность продукта с его моделями	1:M
cart_item-product_item	Определяет принадлежность содержимого корзины с моделью продукта	1:M
order_item-product_item	Определяет принадлежность содержимого заказа с моделью продукта	1:M
order-operation	Определяет принадлежность заказа с осуществленной операцией	1:1
operation-operation_detail	Определяет принадлежность операции с ее деталями	1:M

Данная концептуальная модель базы данных была разработана на основе нотации IDEF1X [9] в приложении draw.io.

2.2.2 Логическая модель баз данных

Логическая модель данных – это расширение концептуальной модели данных. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи, которые представляют бизнес-информацию и определяют бизнес-правила [10]. Также, эта модель является основой для физической модели базы данных.

Для разрабатываемой серверной части логистического сервиса было выделено 6 баз данных, разделенных по предметным областям: «Пользователи», «Клиенты», «Склад», «Корзина», «Заказы», «Запросы».

Такое количество баз данных и их разделение было сделано по следующим причинам:

- при таком разделении можно управлять конкретно по предметным областям, независимо от других;
- снижение нагрузки на базу данных, распределяя ее по нескольким сразу;

– возможность масштабировать БД и хранить резервные копии отдельно.

На рисунке 11 представлена логическая модель базы данных «Пользователи».



Рисунок 11 – Логическая модель БД «Пользователи»

На рисунке 12 представлена логическая модель базы данных «Клиенты».

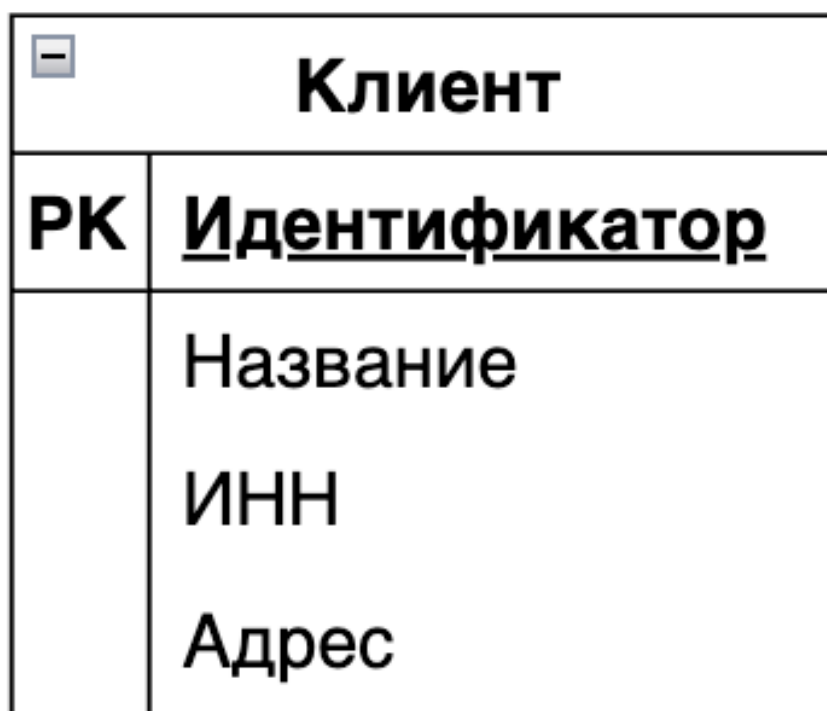


Рисунок 12 – Логическая модель БД «Клиенты»

На рисунке 13 представлена логическая модель базы данных «Склад».

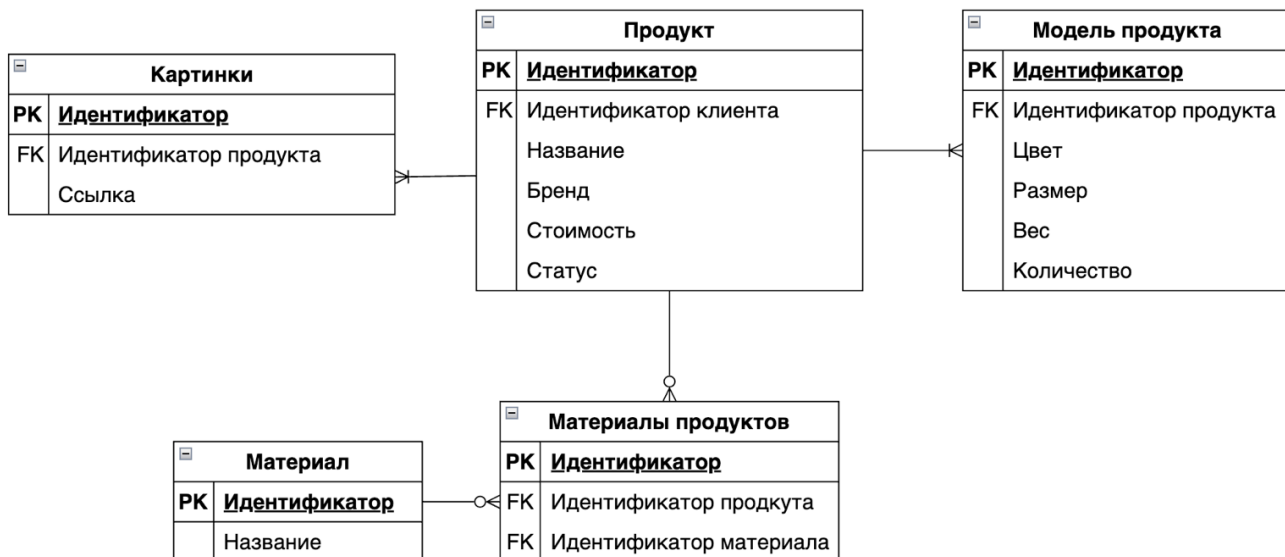


Рисунок 13 – Логическая модель БД «Склад»

На рисунке 14 представлена логическая модель базы данных «Корзина».



Рисунок 14 – Логическая модель БД «Корзина»

На рисунке 15 представлена логическая модель базы данных «Заказы».

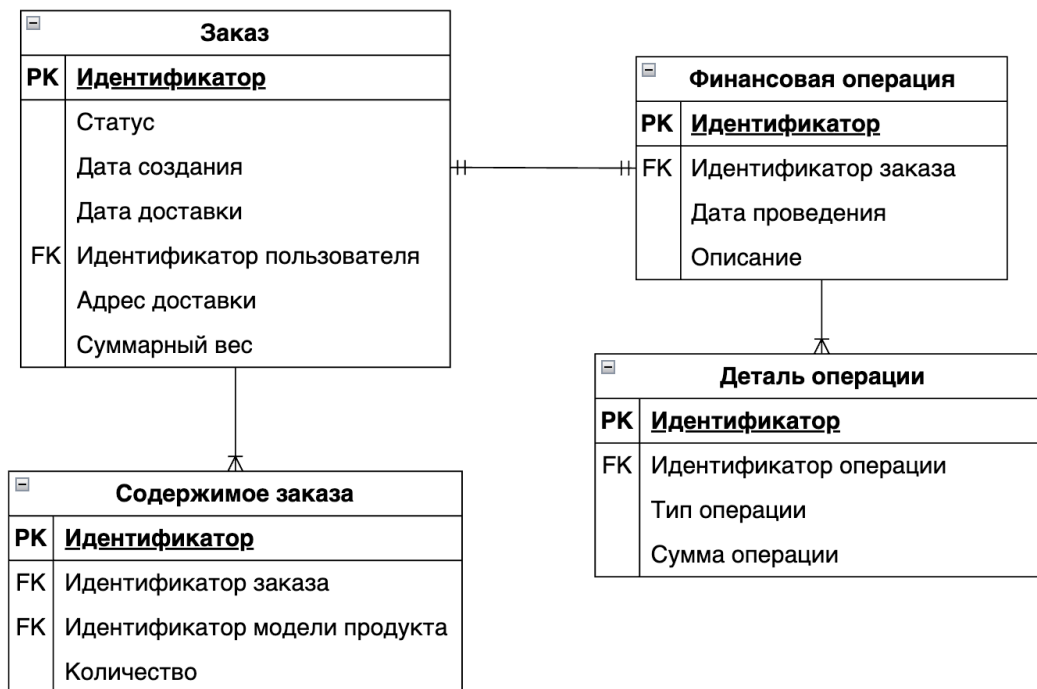


Рисунок 15 – Логическая модель БД «Заказы»

На рисунке 16 представлена логическая модель базы данных «Заявки».

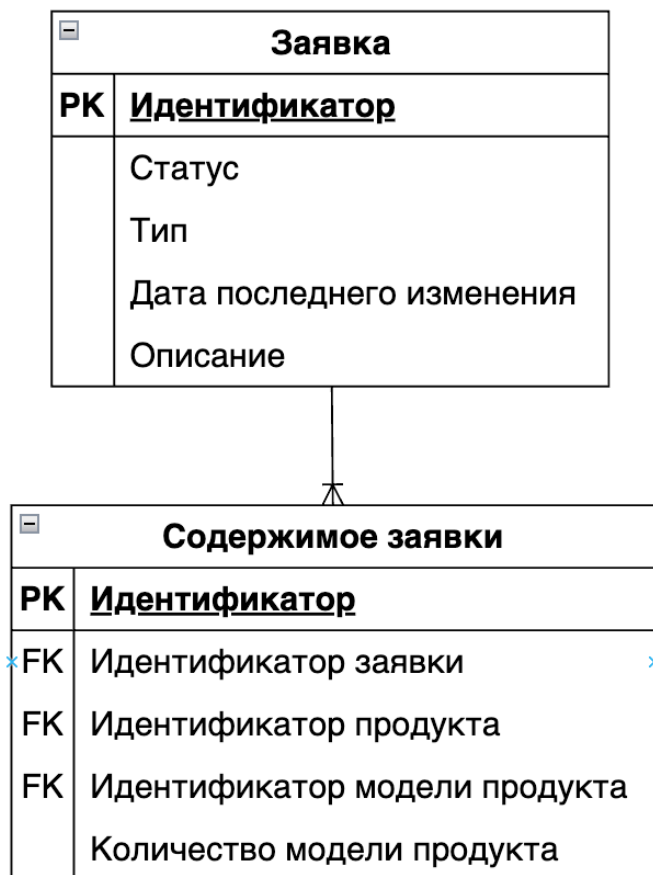


Рисунок 16 – Логическая модель БД «Заявки»

Разработанные логические модели баз данных для серверной части логистического сервиса в дальнейшем будут использованы для разработки физической модели базы данных и ее формирования.

2.3 Обобщенная структура разрабатываемого сервиса

В общем виде разрабатываемый логистический сервис будет представлять собой клиент-серверное приложение. Клиент подразумевает под собой приложение, которое будет использовать пользователь для взаимодействия со всем функционалом сервиса. В большинстве случаев таким приложением выступает браузер, установленный на персональном компьютере пользователя. Сервер, в свою очередь, представляет собой такой же компьютер или другое программное средство, на котором храниться и исполняется разработанное приложение. Также, в клиент-серверных приложениях используется отдельная машина (сервер), на которой отдельно хранятся все данные необходимые для приложения – сервер баз данных.

Основные плюсы таких приложений:

- отсутствие дублирования: весь сайт или приложение хранится на одном компьютере-сервере;
- минимальные требования к пользователю: от него требуется только наличие программы-клиента;
- безопасность: данные хранятся в базе данных, и пользователи не могут их просматривать.

Основные минусы подобных приложений:

- выход из строя сервера или базы данных: это сделает сервис недоступным для всех пользователей;
- перегрузка сервера: возможные DDOS-атаки могут привести к деградации производительности и замедлить работу приложения [11].

Архитектура приложения в общем виде представлена на диаграмме развертывания на рисунке 17.

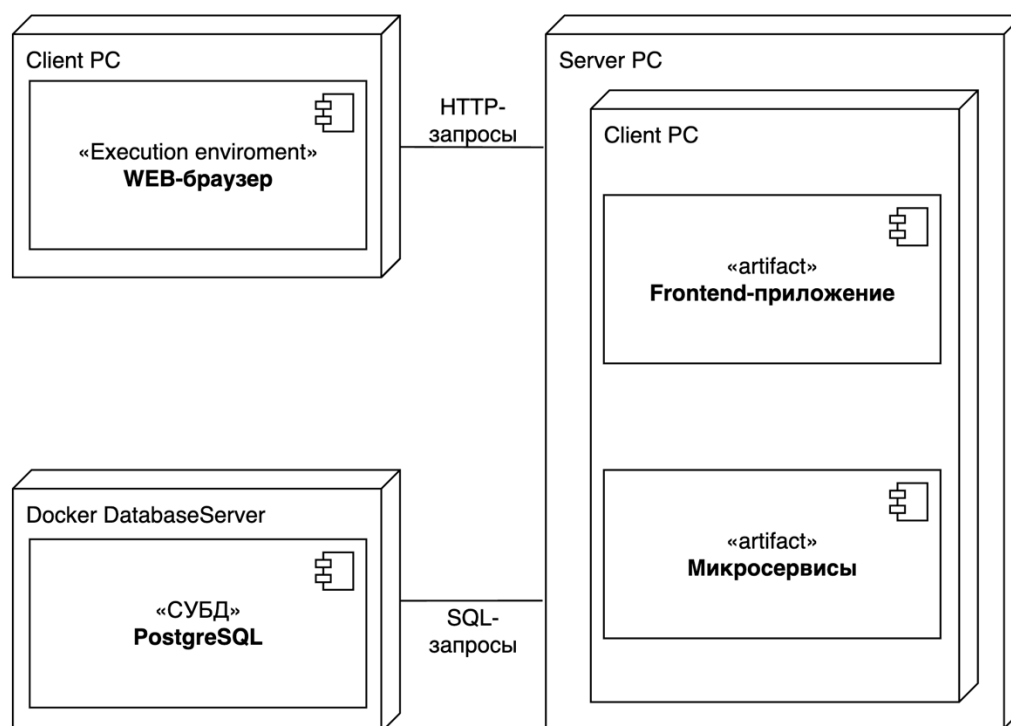


Рисунок 17 – Диаграмма развертывания

На диаграмме отражены три основных компонента для работы системы: компьютер пользователя, удаленный компьютер-сервер, на котором находятся все файлы для запуска и работы приложения, и сервер базы данных.

Пользователь по средством выполнения http-запросов из web-браузера будет обращаться к серверу с приложения для выполнения различных операций. Сервер с приложением, в свою очередь, при необходимости будет обращаться к серверу базы данных, используя SQL-запросы для получения данных логистического сервиса.

Также для обеспечения работы на сервере с приложением и сервере с базой данных должен быть установлен Docker для возможности развертывания всего приложения и базы данных с помощью инструкций, описанных dockerfile и в docker-compose файлах.

2.4 Требования к аппаратному обеспечению серверной части

Для обеспечения корректной работы серверной части разрабатываемого логистического сервиса были выделены минимальные и рекомендованные

требования к техническому оснащению программных средств, на которых будет запускаться приложение. Данные требования описаны в таблице 7.

Таблица 7 – Требования к программному обеспечению

Характеристика	Минимальные требования	Рекомендуемые требования
Процессор	x64 с 1.5 ГГц	x64 с 3 ГГц и более
Оперативная память	4 Гб	8 Гб и более
Тип и объем жесткого диска	HDD со 100 Гб	SSD с 1 Тб и более
Пропуская способность сети	1 Гбит/сек	1 Гбит/сек
Операционная система	Ubuntu 18.04	Ubuntu 20.04 и выше
Обязательное ПО	Docker 3.7	Docker 3.9 и выше

Таким образом, в процессе проектирования серверной части логистического сервиса были определены как минимальные, так и рекомендуемые требования к техническому оснащению, необходимому для корректной работы приложения. Минимальные требования позволяют обеспечить базовую функциональность системы, в то время как рекомендованные параметры направлены на достижение оптимальной производительности и масштабируемости. Учет этих требований при развертывании серверной части гарантирует стабильную работу логистического сервиса и эффективное выполнение задач, производимых в приложении.

3 Разработка программного обеспечения серверной части

3.1 Разработка архитектуры серверной части

Для разработки программного обеспечения серверной части логистического сервиса была выбрана микросервисная архитектура.

Микросервисы – это независимо развертываемые службы, моделируемые вокруг бизнес-домена. Они общаются друг с другом через сети и предлагают на выбор целый ряд вариантов архитектуры для решения задач. С технологической точки зрения микросервисы выставляют наружу возможности бизнеса, которые они инкапсулируют через одну или несколько конечных точек в сети. Микросервисы взаимодействуют друг с другом через эти сети, что делает их разновидностью распределенной системы. Они также инкапсулируют хранение и извлечение данных, предоставляя данные через четко определенные интерфейсы. И поэтому базы данных скрыты в пределах контура службы [12].

Серверная часть разрабатываемой системы состоит из 12 микросервисов, каждый из которых отвечает за свою конкретную предметную область. Для каждой части системы были разработаны свои интерфейсы для взаимодействия с ними по средством HTTP-запросов. Кроме того, для каждого микросервиса, что оперирует и обрабатывает какие-то данные, были выделенные отдельные базы данных и описаны интерфейсы для взаимодействия с данными.

Для обеспечения взаимодействия клиентской части разработанного приложения с серверной был выделен специальный сервис – API-Gateway. Он является общей точкой входа в систему, где описаны все контракты для взаимодействия с серверной частью. Он также инкапсулирует в себе модели, которые опрашиваются пользовательскими запросами и передаются далее в целевой микросервис, не раскрывая всех деталей разработанной системы пользователю.

Для разработки каждого из микросервисов в качестве типового решения для написания архитектуры было принято решение использовать многослойную архитектуру. Многоуровневая архитектура (или многослойная архитектура) –

это подход, при котором приложение разделяется на несколько слоев, каждый из которых отвечает за определенный функциональный аспект. Преимущества многоуровневой архитектуры включают легкость поддержки и изменения, а также улучшенную модульность и переиспользуемость кода [13]. На рисунке 18 представлен типовой вид многослойной архитектуры, используемый в каждом разработанном микросервисе системы.

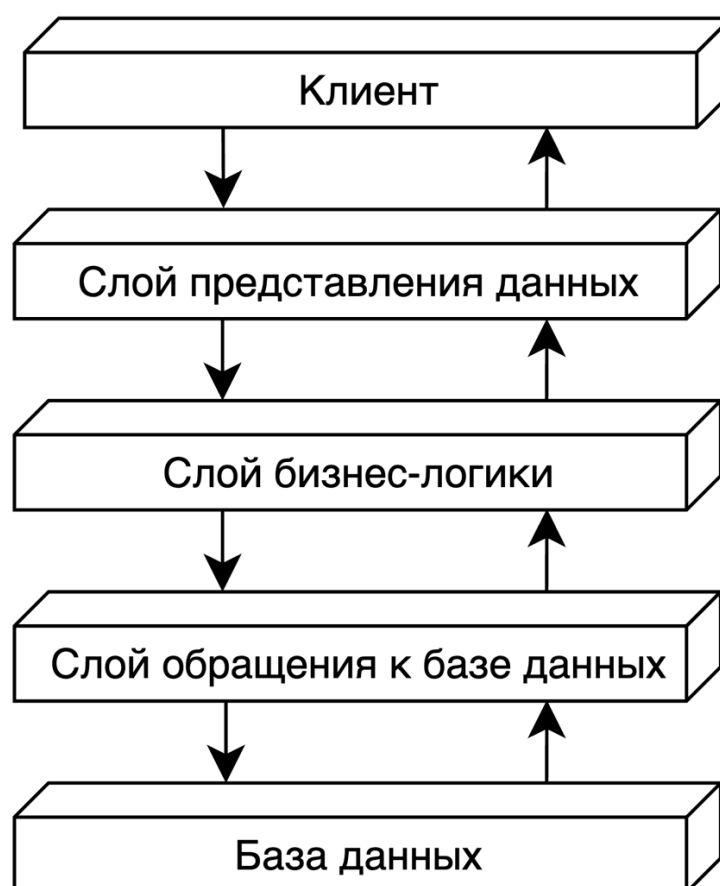


Рисунок 18 – Типовая архитектура микросервисов

Каждый из представленных слоев архитектуры выполняет свою роль в выполнении клиентских запросов.

Изначально все запросы пользователя (Клиент) попадают в слой представления данных (Controller), проходят валидацию и отправляются на дальнейшую обработку в слой бизнес-логики.

Слой бизнес-логики (Service) отвечает за всю обработку данных, пришедших от пользователя, и выполнение необходимых бизнес-функций для успешного выполнения запроса. Этот слой оперирует доменными (Domain)

объектами системы – ключевыми объектами, вокруг которых строится предметная область микросервиса. При необходимости получения каких-то данных их базы данных, этот слой обращается к следующему слою – слою обращения к базе данных.

Слой обращения к базе данных (Repository) описывает и определяет, каким образом микросервис будет взаимодействовать с базой данных. На данном уровне также происходит инициализация и настройка подключения к базе данных. Слой обращения к БД в свою очередь оперирует уже моделями баз данных (Data Access Object), которые по своей структуре повторяют сущности, описанные в таблицах баз данных, со всеми присущими им атрибутами.

На уровне базы данных хранятся все данные, относящиеся к предметной области микросервиса, а также описанные функции и триггеры для выполнения некоторых сложных агрегационных запросов.

Разработанное приложение было написано с учетом всех требований и стандартов REST API и обеспечивает единый интерфейс для взаимодействия HTTP-запросами между клиентом и сервером.

Также для выполнения асинхронной обработки данных и отложенного вызова некоторых функций системы, был использован брокер сообщений Apache Kafka. В определенную очередь (Topic) доставляется посредством определенного отправителя (Producer) сообщение с некоторым типом и содержанием, которое через какое-то время будет доставлено и обработано получателем (Consumer).

Кроме того, чтобы снять нагрузку с основной базы данных, находящейся под управлением PostgreSQL, было принято решение использовать базу данных ключ-значение Redis для кеширования выдачи каталога, имеющейся на складе продукции. Такой подход позволяет не только разгрузить основную базу, но ускорить выполнение запросов, так как Redis хранит свои данные в оперативной памяти и при получении данных их кеша мы не выполняем хождения за данными в какие-то другие микросервисы, а только достаем и фильтруем подсохраненные данные.

3.2 Разработка физической модели базы данных

Физическая модель базы данных – финальный этап моделирование базы данных, сделанный на основе логической модели базы данных. Эта модель данных включает в себя все необходимые таблицы, столбцы, связи, свойства базы данных для физической реализации баз данных. Производительность базы данных, стратегия индексации, физическое хранилище и денормализация – важные параметры физической модели [14]. Физическая модель разрабатывается уже для конкретной СУБД, выбранной для программного обеспечения.

На рисунке 19 представлена физическая модель базы данных «Пользователи».

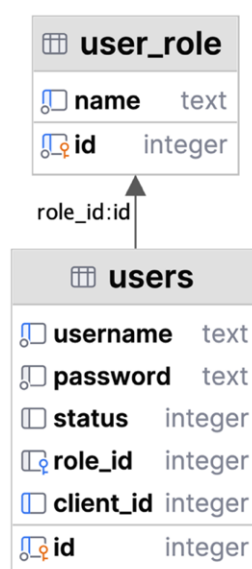


Рисунок 19 – Физическая модель БД «Пользователи»

На рисунке 20 представлена физическая модель базы данных «Клиенты».



Рисунок 20 – Физическая модель БД «Клиенты»

На рисунке 21 представлена физическая модель базы данных «Склад».

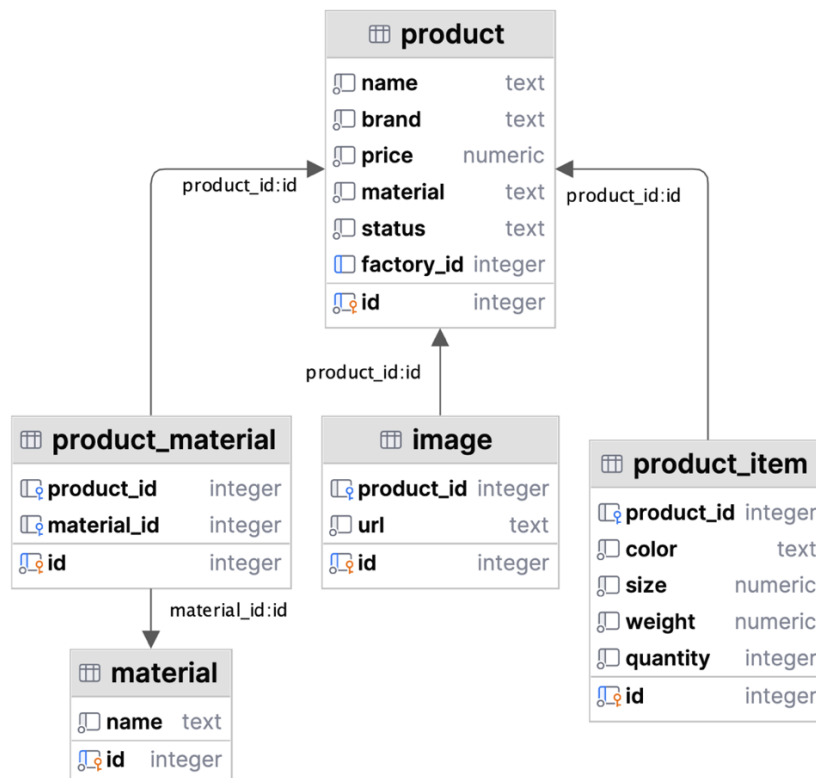


Рисунок 21 – Физическая модель БД «Склад»

На рисунке 22 представлена физическая модель базы данных «Корзина».

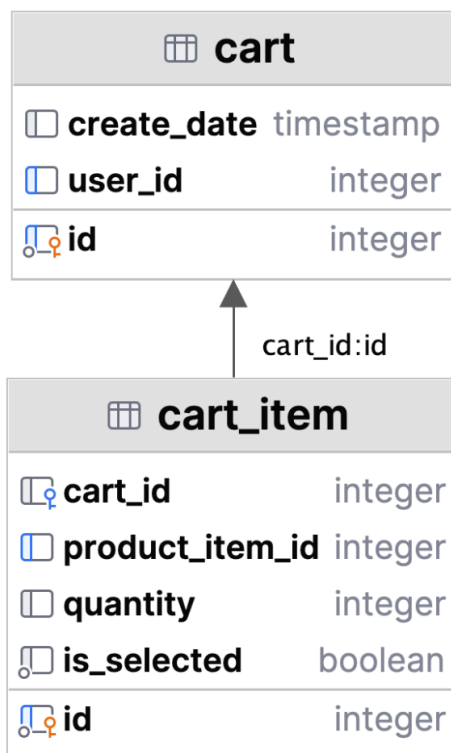


Рисунок 22 – Физическая модель БД «Корзина»

На рисунке 23 представлена физическая модель базы данных «Заказы».

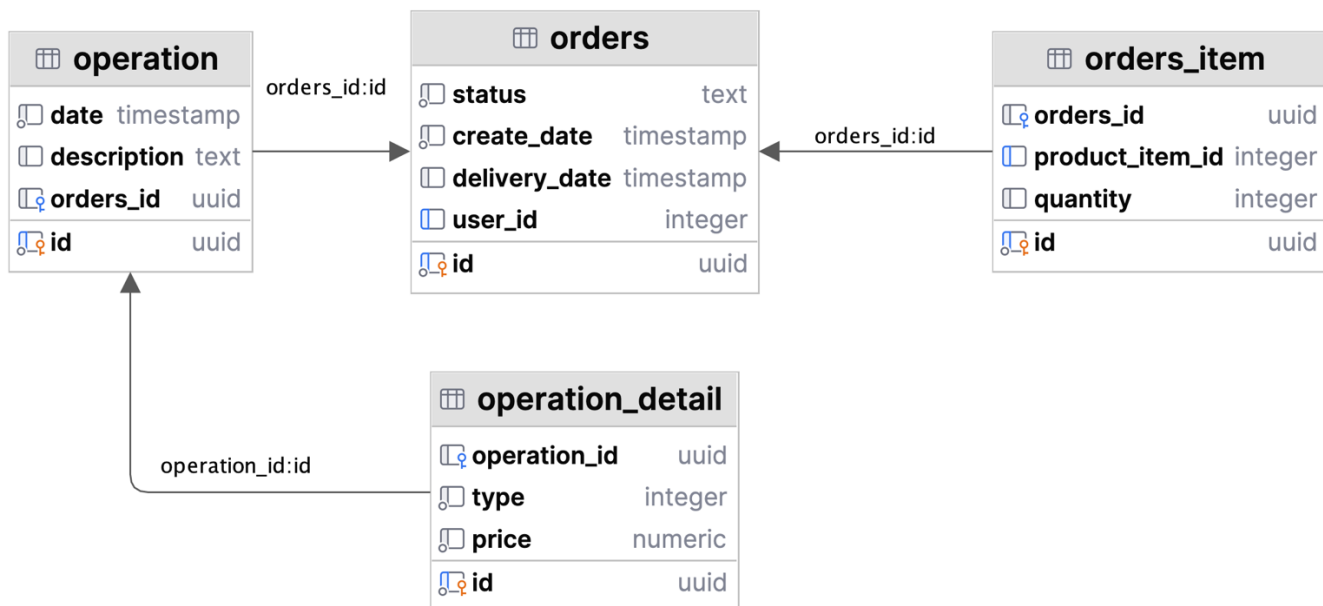


Рисунок 23 – Физическая модель БД «Заказы»

На рисунке 24 представлена физическая модель базы данных «Запросы».

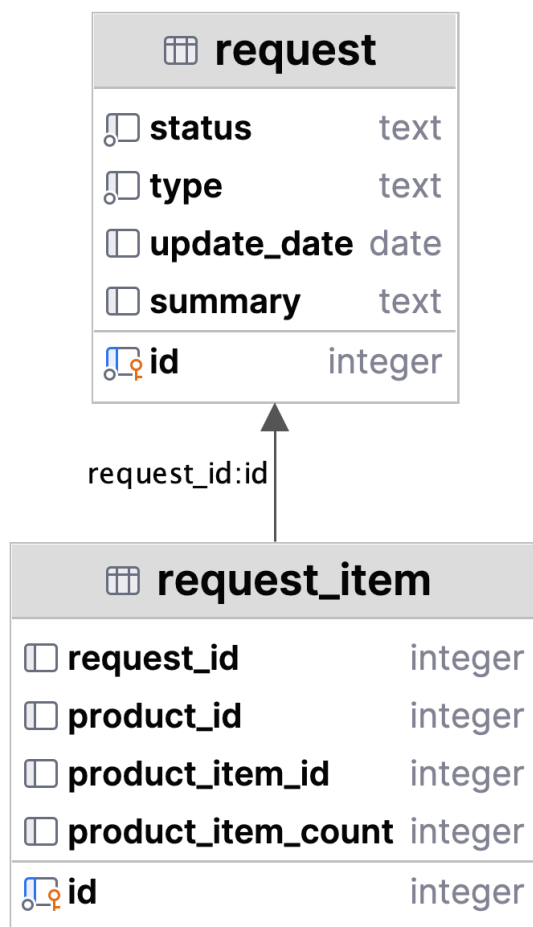


Рисунок 24 – Физическая модель БД «Запросы»

В таблице 7 отражено табличное описание всех сущностей всех баз данных логистического сервиса.

Таблица 7 – Описание сущностей баз данных

№ п/п	Наименование	Описание	Категория
1	users	Список всех пользователей системы	Стержневая
2	user_role	Данные о ролях пользователей	Кодовая
3	client	Список всех клиентов системы	Стержневая
4	product	Список всех клиентов системы	Стержневая
5	product_item	Детализированная информация о моделях продукции	Характеристическая
6	material	Список всех материалов	Стержневая
7	product_material	Детализированная информация о моделях продукции	Ассоциативная
8	image	Список всех изображений	Характеристическая
9	cart	Список корзин пользователей	Стержневая
10	cart_item	Список элементов корзины пользователя	Характеристическая
11	orders	Список всех заказов	Стержневая
12	orders_item	Список элементов заказа	Характеристическая
13	operation	Информация о финансовых транзакциях	Стержневая
14	operation_detail	Подробности операций	Характеристическая
15	request	Общая информация о запросах	Стержневая
16	request_item	Детальная информация о содержимом запроса	Характеристическая

Спецификации атрибутов всех сущностей таблиц баз данных представлены в приложении И.

Структуры данных, используемые в микросервисах в слоях обращения баз данных, описаны в приложении Б.

SQL-скрипты для создания и наполнения таблиц представлены в приложении В.

3.3 Разработка архитектурных слоев внутри микросервисов

В современных микросервисных архитектурах критически важным является четкое разделение ответственности между компонентами системы. Для обеспечения масштабируемости, тестируемости и поддерживаемости кода каждый микросервис построен по принципу многослойной архитектуры

В качестве примера для детального рассмотрения структуры и взаимодействия слоев будет использоваться микросервис управления складом (warehouse-service). Данный выбор обусловлен тем, что структура и принципы построения остальных микросервисов аналогичны, а функционал warehouse-service является репрезентативным для демонстрации всех ключевых архитектурных паттернов.

Детальное описание реализации каждого слоя всех остальных микросервисов представлено в соответствующих приложениях:

- описание слоя представления данных в приложении Г;
- описание слоя бизнес-логики в приложении Д;
- описание слоя обращения к базе данных в приложении Е.

3.3.1 Разработка слоя представления данных в микросервисах

Слой представления данных ответственный за описание контрактов, по которым клиентская часть сервиса узнает, что можно получить от данного микросервиса.

Каждая функция, описанная на данном архитектурное слое, помечается суффиксом Handler и относится подключается при инициализации сервера к общему роутеру. Все эти методы принимают GET, POST, PUT и DELETE запросы, пришедшие по HTTP протоколу, проверяют пришедшие данные и отправляют их дальше на следующий слой – слой бизнес-логики.

Пример инициализации роутера и подключения к нему всех маршрутов представлен на листинге 1.

Листинг 1 – Инициализация роутера и подключение маршрутов

```
func (r *Router) SetupRoutes(logger goatlogger.Logger, warehouse
service.WareHouse) {
    r.router.HandleFunc("/products",
handlers.GetProductsHandler(logger,
warehouse)).Methods(http.MethodGet)
    r.router.HandleFunc("/product/client/{id}",
handlers.GetClientProductsHandler(logger,
warehouse)).Methods(http.MethodGet)
```

```

    r.router.HandleFunc("/product/{productId}",
handlers.GetDetailedProductHandler(logger,
warehouse)).Methods(http.MethodGet)
    r.router.HandleFunc("/materials/list",
handlers.GetMaterialsList(logger,
warehouse)).Methods(http.MethodGet)
    r.router.HandleFunc("/products",
handlers.AddProductsHandler(logger,
warehouse)).Methods(http.MethodPost)
    r.router.HandleFunc("/products",
handlers.UpdateProductsHandler(logger,
warehouse)).Methods(http.MethodPut)
    r.router.HandleFunc("/products",
handlers.DeleteProductsHandler(logger,
warehouse)).Methods(http.MethodDelete)
    r.router.HandleFunc("/products/items",
handlers.GetProductItemsInfoHandler(logger,
warehouse)).Methods(http.MethodPost)
    r.router.HandleFunc("/products/detailed",
handlers.GetDetailedProductsHandler(logger,
warehouse)).Methods(http.MethodPost)
}

```

Пример реализации функции для обработки запросов представлен на листинге 2.

Листинг 2 – Реализации функции обработки запроса

```

func GetProductsHandler(logger goatlogger.Logger, warehouseService
service.Warehouse) server.Handler {
    return func(w http.ResponseWriter, r *http.Request) {
        ctx, err := goatcontext.New(r)
        if err != nil || !ctx.IsAuthorized() {
            w.WriteHeader(http.StatusUnauthorized)
            return
        }

        products, err := warehouseService.GetProducts(ctx)
        if err != nil {
            logger.Error(fmt.Sprintf("не удалось получить список
продуктов: %v", err))
            w.WriteHeader(http.StatusBadRequest)
            return
        }

        if err = json.WriteResponse(w, http.StatusOK, products);
err != nil {
            logger.Error(fmt.Sprintf("не удалось записать ответ:
%v", err))
            w.WriteHeader(http.StatusBadRequest)
            return
        }
    }
}

```

На рисунке 25 отображено графическое отображение связи между роутером и функциями обработки пришедших HTTP-запросов.

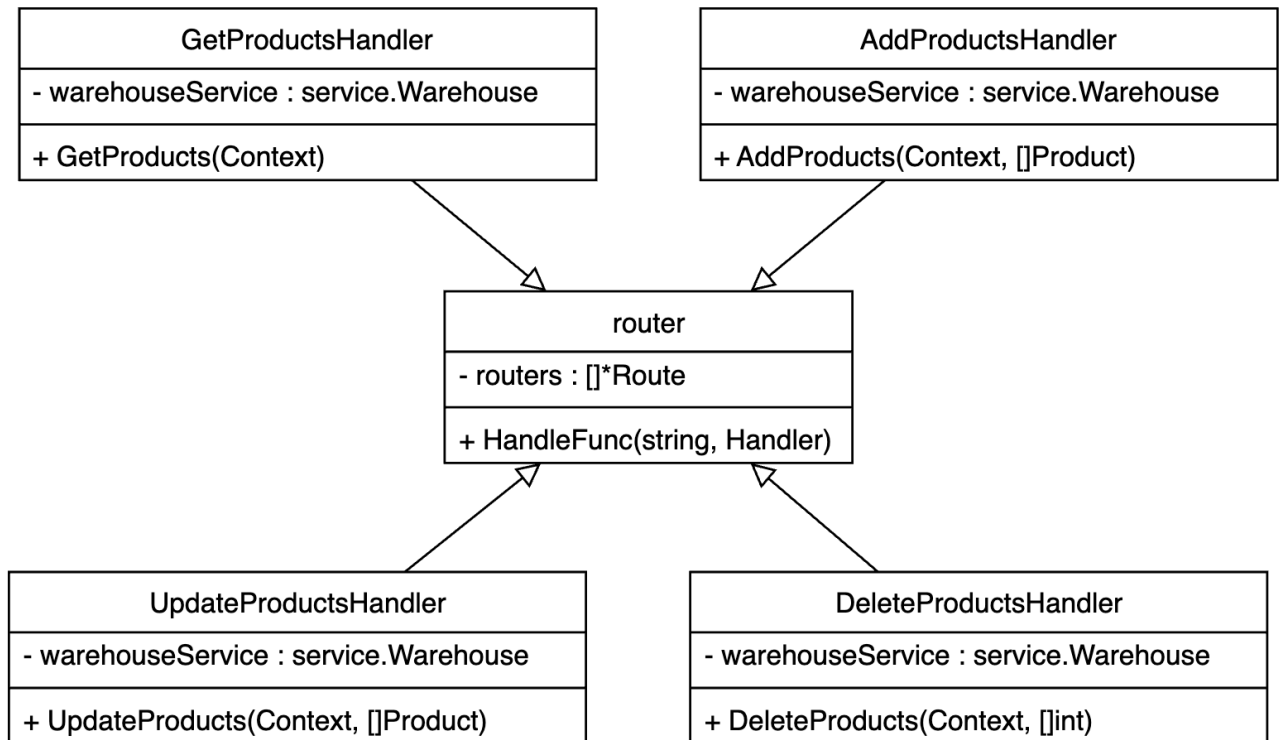


Рисунок 25 – Связь между роутером и функциями

Таким образом, слой представления данных играет ключевую роль в обеспечении взаимодействия между клиентской и серверной логикой. Он отвечает за валидацию входящих запросов, их маршрутизацию и передачу данных на слой бизнес-логики для дальнейшей обработки. Такой подход позволяет четко разделить ответственность между компонентами системы, что упрощает поддержку и масштабирование микросервиса.

3.3.2 Разработка слоя бизнес-логики в микросервисах

Слой бизнес-логики отвечает за выполнение и обработку данных, специфичных для конкретной предметной области внутри микросервиса. Все структуры, содержащие все функции, описывающие бизнес-процессы, помечаются ключевым словом `Service`, для обозначения принадлежности к слою бизнес-логики. Функции помечаются ключевым словом, в зависимости от того, что от них ожидается:

- Get – получение данных;
- Add – добавление данных;
- Update – обновление данных;
- Delete – удаление данных;

Пример инициализации сервиса и описание функций бизнес-процессов отражен на листинге 3.

Листинг 3 – Инициализация сервиса и описание функций

```
type WareHouse interface {
    GetProducts(ctx goatcontext.Context) ([]domain.Product, error)
    AddProducts(ctx goatcontext.Context, products
[]domain.Product) error
    UpdateProducts(ctx goatcontext.Context, products
[]domain.Product) error
    DeleteProducts(ctx goatcontext.Context, productIds []int)
error
}
type Impl struct {
    warehouseRepository repository.Warehouse
    clientServiceClient *clientservice.Client
    kafkaProducer      *broker.Producer
}
func NewWarehouseService(warehouseRepository repository.Warehouse,
clientServiceClient *clientservice.Client, kafkaProducer
*broker.Producer) WareHouse {
    return &Impl{
        warehouseRepository: warehouseRepository,
        clientServiceClient: clientServiceClient,
        kafkaProducer:      kafkaProducer,
    }
}
```

Вся обработка данных на этом уровне ведется уже с использованием доменных моделей предметной области, вокруг построена вся обработка. Такие модели хранятся в отдельном модуле с наименованием `domain`. Кроме того, данный слой уже взаимодействует со слоем обращения к базе данных для получения необходимой информации для выполнения бизнес-процессов. Для взаимодействия со следующим уровнем, например, при сохранении объектов в базу данных, предварительно перед передачи далее преобразует в объекты, характерные для базы данных, используя специальные функции-мапперы. Эти функции хранятся модуле с названием `mapping` и помечаются предложением «To» с

дальнейшим описанием в какой объект необходимо преобразовать данные. Также, на этом уровне используется специальные сервисы-клиенты из модуля `cluster` для взаимодействия с другими микросервисами.

Пример функции слоя бизнес-логики представлен на листинге 4.

Листинг 4 – Пример функции слоя бизнес-логики

```
func (s *Impl) GetProducts(ctx goatcontext.Context)
([]domain.Product, error) {
    dbProducts, err := s.warehouseRepository.GetProducts(ctx)
    if err != nil {
        return nil, err
    }

    factoryIds := lo.Map(dbProducts, func(item models.Product, _
int) int {
        return item.FactoryId
    })

    shortFactoryInfos, err :=
s.clientServiceClient.GetClientInfoShort(ctx, factoryIds)
    if err != nil {
        return nil, err
    }

    shortFactoryInfosMap := lo.Associate(shortFactoryInfos,
func(item clientservice.ClientInfoShort) (int,
clientservice.ClientInfoShort) {
        return item.Id, item
    })

    products := make([]domain.Product, 0, len(dbProducts))
    for _, dbProduct := range dbProducts {
        products = append(products,
mappings.ToDomainProduct(dbProduct,
shortFactoryInfosMap[dbProduct.FactoryId]))
    }

    return products, nil
}
```

В примере приведена реализация метода, для получения продукции из базы данных с информацией о производителях, которые их добавили.

В самом начале функции происходит взаимодействие со слоем обращения данных посредством вызова `warehouseRepository.GetProducts` и сохранения полученного результата в поле `dbProducts`.

Для получения информации о производителях, добавивших свою продукцию в сервис, используется функция `GetClientInfoShort`. Она связывает микросервис управления продукцией на складе с микросервисом управления клиентами с помощью HTTP-запроса. В сам запрос передаются идентификаторы необходимых для получения клиентов.

Также в функции получения продукции используется функция-маппер `ToDomainProduct` для преобразования объекта продукции из вида, полученного из базы данных, в доменный вид.

Диаграмма классов сервиса слоя бизнес-логики микросервиса управления складом представлена на рисунке 26.

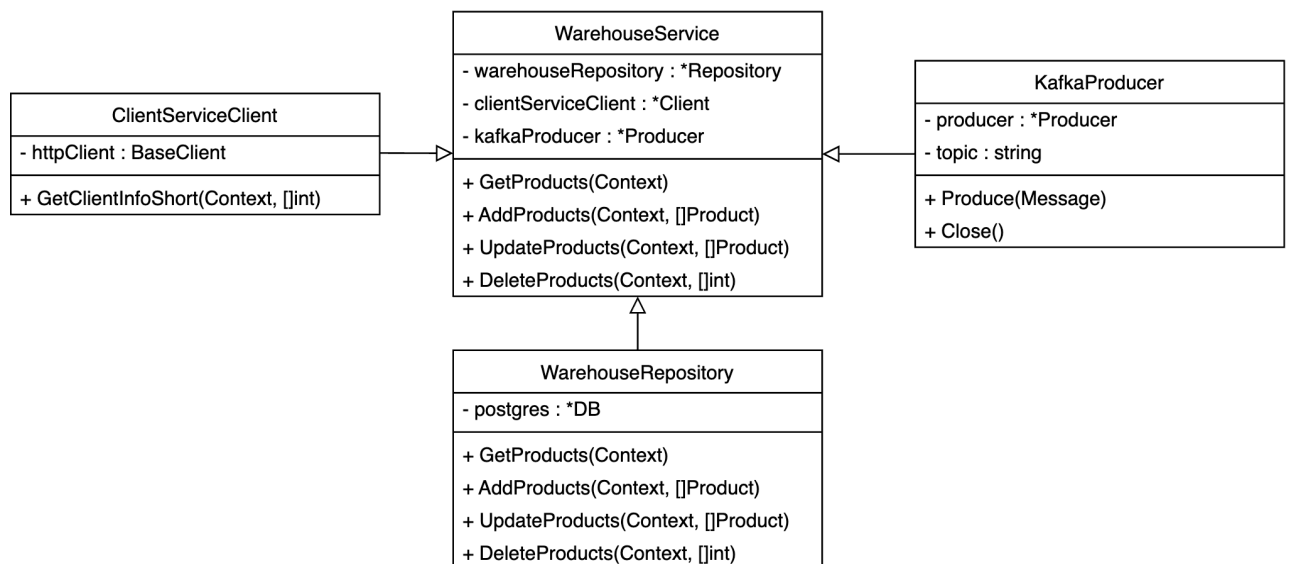


Рисунок 26 – Диаграмма классов сервиса

Слой бизнес-логики совмещает в себе обработку данных и связь с другими частями системы, что улучшает поддержку и масштабируемость приложения

3.3.3 Разработка слоя обращения к базе данных в микросервисах

Слой обращения к базе данных отвечает за подключение и взаимодействие с базой данных через функции, использующие SQL-скрипты. Структуры репозитория помечаются ключевым словом `Repository`. Данный слой не содержит в себе сложной логики обработки данных и предназначен только для

произведения различных операций над данными в БД. Пример инициализации репозитория и описание функций представлен на листинге 5.

Листинг 5 – Инициализация репозитория и описание функций

```
type Warehouse interface {
    GetProducts(ctx goatcontext.Context) ([]models.Product, error)
    AddProducts(ctx goatcontext.Context, products
[]models.Product) ([]models.Product, error)
    UpdateProducts(ctx goatcontext.Context, products
[]models.Product) error
    DeleteProducts(ctx goatcontext.Context, productIds []int)
error
}

type Init struct {
    postgres *sqlx.DB
}

func NewWarehouseRepository(postgres *sqlx.DB) Warehouse {
    return &Init{
        postgres: postgres,
    }
}
```

Пример функции слоя обращения к БД представлен на листинге 6.

Листинг 6 – Пример функции слоя обращения к базе данных

```
func (r *Init) GetProducts(ctx goatcontext.Context)
([]models.Product, error) {
    var products []models.Product
    if err := r.postgres.SelectContext(ctx, &products,
queries.GetProducts); err != nil {
        return nil, err
    }

    return products, nil
}
```

Функция из приведенного примера отвечает за получение продукции из базы данных. Для взаимодействия с базой данных используется функция `SelectContext`, в которую передается SQL-скрипт `GetProducts`. Все данные по выполнению метода сохраняются в поле `products`. SQL-скрипты, разработанные для логистического сервиса, отражены в приложении Г.

Рассмотрим взаимодействие всех компонентов и слоев между собой на примере получения доступной продукции. Для этого была разработана диаграмма последовательности, представленная на рисунке 27.

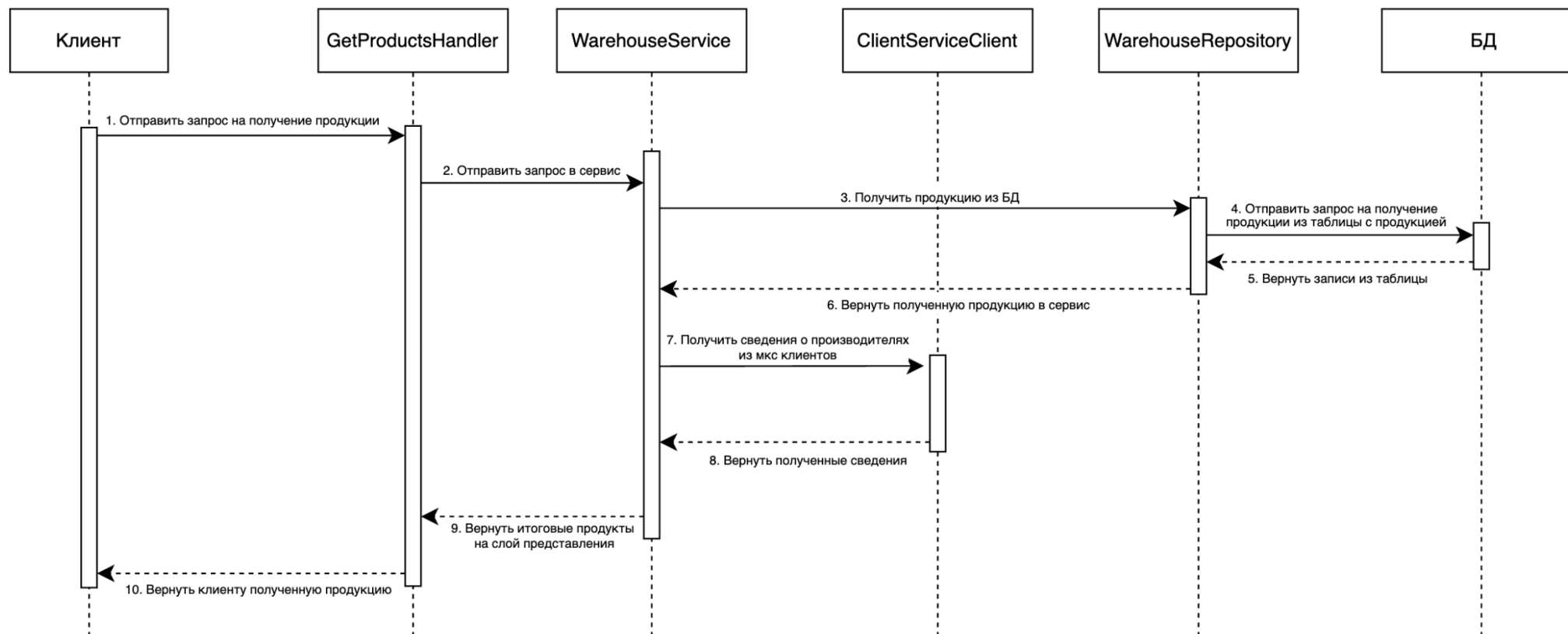


Рисунок 27 – Диаграмма последовательности получения продукции

Процесс получения продукции начинается с отправки клиентом запроса на получение продукции. Запрос получается и обрабатывается в `GetProductsHandler`, откуда дальше отправляется на слой бизнес-логики для проведения всех необходимых действий, направленных на получение итогового результата. В `WarehouseService` первоначально происходит обращение на слой обращения к базе данных (`WarehouseRepository`) для получения основной информации о продуктах из таблицы, хранящей всю добавленную продукцию. `WarehouseRepository` в свою очередь, связывается с базой данных посредством выполнения метода, использующего SQL-скрипт получения продукции, и возвращает данные из таблицы на дальнейшую обработку в сервис. Следующий этап – получение информации о производителях из микросервиса управления клиентами. `WarehouseService` вызывает функцию из `ClientServiceClient`, которая отправляется в целевой микросервис по HTTP-протоколу для получения необходимых данных. Получив все необходимые данные на слое бизнес-логики, данные объединяются в итоговую выдачу продуктов и возвращаются на слой представления данных. Полученные данные отправляются назад клиенту и отрисовываются в приложении клиента.

4 Контроль качества программного обеспечения

4.1 Обоснование выбора модели тестирования и метрик кода

В процессе разработки серверной части логистического сервиса особое внимание уделяется обеспечению качества программного кода. Комплексный подход к тестированию и анализу качеству кода позволяет гарантировать надежность, безопасность и эффективность функционирования частей микросервисной архитектуры.

Основные виды тестирования серверной части включают:

- модульное тестирование – проверка корректности работы отдельных компонентов и модулей системы в изоляции [15];
- интеграционного тестирование – верификация взаимодействия между микросервисами и компонентами системы [16];
- тестирование производительности – оценка скорости обработки запросов, нагрузки на систему и масштабируемости;
- функциональное тестирование – проверка соответствия реализованных функций заявленным требованиям [17];

Метрики качества кода играют важную роль в оценке надежности и эффективности серверной части логистического сервиса. В рамках данной выпускной квалификационной работы были выбраны следующие метрики: SLOC и метрика Холстеда.

SLOC (Source Lines of Code) – метрика, измеряющая количество строк исходного кода приложения. Она позволяет:

- оценить объем разработки;
- прогнозировать трудозатраты на поддержку;
- выявлять избыточную сложность кода.

Метрика Холстеда – комплексная метрика, основанная на анализе операторов и операндов в коде. Она обеспечивает:

- оценку сложности программы;
- прогнозирование трудоемкости разработки;

- анализ потенциальной надежности системы [18].

Выбор данных метрик обусловлен их способностью предоставить объективную оценку качества кода микросервисов, выявить потенциальные проблемы в архитектуре и обеспечить контроль над сложностью системы при ее развитии. Регулярный мониторинг выбранных метрик позволяет поддерживать высокий уровень качества серверной части логистического сервиса.

Комплексный подход к тестированию и анализу качества кода обеспечивает соответствие разрабатываемого программного обеспечения предъявляемым требованиям и гарантирует его надежную работу в условиях реальной эксплуатации.

4.2 Тестирование серверной части логистического сервиса

4.2.1 Функциональное тестирование добавления новой продукции

После завершения разработки программного обеспечения проводилось функциональное тестирование.

Этот процесс особенно важен для обеспечения качества разрабатываемого программного средства, поскольку функциональное тестирование позволяет контролировать, что приложение удовлетворяет всем поставленным требованиям и спецификациям. При тестировании выявляются и устраняются ошибки, что дает повышает надежность и стабильность работы системы.

Для проведения функционального тестирования был выбран пакет «Warehouse». Данный пакет отвечает за реализацию взаимодействия всех остальных частей системы со складом. Функционально «Warehouse» позволяет производить операции получения продукции и различные операции модификации над товарами: добавление новых, изменение и удаление уже имеющихся.

Для данного пакета были выявлены следующие требования:

- программа должна сообщить об ошибке, если вводимое название товара не соответствует поставленным ограничениям;

- программа должна сообщить об ошибке, если подобный товар уже существует в системе;
- программа должна сообщить об ошибке, если описание товара введено некорректно.

Для работы с данным пакетом было определено 5 причин и 6 следствий. Они представлены в таблице 8.

Таблица 8 – Причины и следствия

Причины	
П1	Корректная длина названия (5 – 20 символов)
П2	Корректные символы названия (А-Я, а-я, А-Z, а-z, 0 – 9, -, « »)
П3	Название товара не занято
П4	Корректная длина описания (10 – 250 символов)
П5	Корректные символы в описании (Все кроме спецсимволов (/, #, {} и т. д.)
Следствия	
С1	Некорректная длина названия товара
С2	Некорректные символы в названии товара
С3	Название товара уже занято
С4	Некорректная длина описания
С5	Некорректная символы в описании товара
С6	Товар успешно добавлен

На основе таблицы 8 создана функциональная диаграмма, которая описывает поведение программы с учетом возникающих во время работы приложения ошибок. Диаграмма представлена на рисунке 28

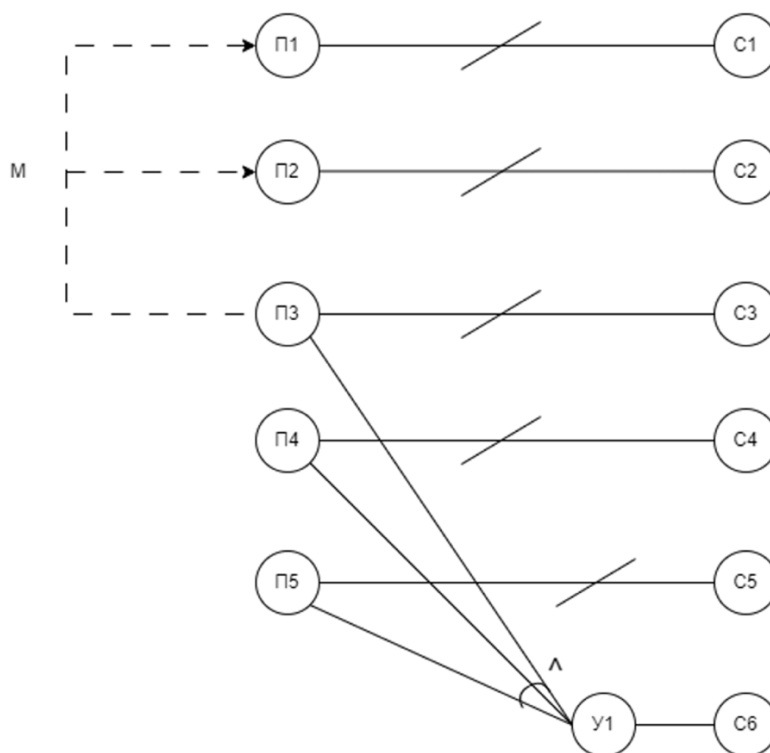


Рисунок 28 – Функциональная диаграмма

На основе функциональной диаграммы была составлена таблица решений для всех вариантов поведения, которые возможны в разрабатываемом приложении. Таблица решений представлена в таблице 9.

Таблица 9 – Таблица решений

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
П1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
П2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1
П3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
П4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
П5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
С1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
С2	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
С5	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
С4	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
С3	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
С6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

На основе таблицы решений были описаны следующие тест-кейсы для выявления ошибок и проверки реакции системы на возникающие во время работы приложения ситуации. Тест-кейсы представлены в таблице 10.

Таблица 10 – Тест-кейсы

№	Входные данные		Выходные данные			
1	T@рт	Оп№с@ние	Некорректная длина названия товара	Некорректные символы в названии товара	Некорректная символы в описании товара	Некорректная длина описания
2	T@рт	Верн(е)_оп№с@ние	Некорректная длина названия товара	Некорректные символы в названии товара	Некорректная символы в описании товара	
3	T@рт	Описание	Некорректная длина названия товара	Некорректные символы в названии товара		Некорректная длина описания
4	T@рт	Корректное описание товара	Некорректная длина названия товара	Некорректные символы в названии товара		
5	Торт	Оп№с@ние	Некорректная длина названия товара		Некорректная символы в описании товара	Некорректная длина описания

Продолжение таблицы 10

№	Входные данные		Выходные данные			
6	Торт	Верн()е_оп№с@ние	Некорректная длина названия товара		Некорректная символы в описании товара	
7	Торт	Описание	Некорректная длина названия товара			Некорректная длина описания
8	Торт	Корректное описание товара	Некорректная длина названия товара			
9	Шоколадный т@рт	Оп№с@ние		Некорректные символы в названии товара	Некорректная символы в описании товара	Некорректная длина описания
10	Шоколадный т@рт	Верн()е_оп№с@ние		Некорректные символы в названии товара	Некорректная символы в описании товара	
11	Шоколадный т@рт	Описание		Некорректные символы в названии товара		Некорректная длина описания
12	Шоколадный т@рт	Корректное описание товара		Некорректные символы в названии товара		
13	Шоколадный торт	Оп№с@ние			Некорректная символы в описании товара	Некорректная длина описания
14	Шоколадный торт	Верн()е_оп№с@ние			Некорректная символы в описании товара	
15	Шоколадный торт	Описание				Некорректная длина описания
16	Шоколадный торт	Корректное описание товара				

Окончание таблицы 10

№	Входные данные		Выходные данные			
17	Шоколадный торт	Оп№с@ние			Некорректная символы в описании товара	Некорректная длина описания
18	Шоколадный торт	Верн(е)_оп№с@ние			Некорректная символы в описании товара	
19	Шоколадный торт	Описание				Некорректная длина описания
20	Шоколадный торт	Корректное описание товара				Товар успешно добавлен

Программа успешно прошла все функциональные требования. Поведение разработанной системы соответствует всем требованиям.

4.2.2 Модульное тестирование метода обновления заявки

Для проверки корректности работы разработанного программного средства и проверки ожидаемого поведения программы было проведено модульное тестирование.

Тестирование пакета «Request» разработанного на языке Go проводится с помощью встроенного в язык модуля testing. Исходный код теста представлен в приложении Ж. Результаты проведения тестов представлены в таблице 11.

Таблица 11 – Результаты модульного тестирования

№	Входные данные	Ожидаемый результат	Фактический результат	Статус теста
1	Статус: reject	Статус заявки поменялся на отклонено, сообщение о обновлении статуса доставлено до пользователя	Статус заявки поменялся на отклонено, сообщение о обновлении статуса доставлено до пользователя	Пройден
2	Статус: approve	Статус заявки поменялся на подтверждено, сообщение о обновлении статуса доставлено до пользователя	Статус заявки поменялся на подтверждено, сообщение о обновлении статуса доставлено до пользователя	Пройден
3	Статус: пустой	Ошибка: статус не может быть пустым	Ошибка: статус не может быть пустым	Пройден

Продолжение таблицы 11

№	Входные данные	Ожидаемый результат	Фактический результат	Статус теста
4	Статус: nil	Ошибка: статус не может быть nil	Ошибка: статус не может быть nil	Пройден

Все этапы модульного тестирования прошли успешно. Программа корректно обрабатывает во всех возможных тестируемых ситуациях.

4.2.3 Метрики Холстеда для создания нового заказа

В качестве примера для расчета метрик рассмотрим сервис «Order», отвечающий за работу с заказами. Код данного модуля представлен в приложении Е.

Чтобы произвести оценку сложности необходимо рассчитать количество уникальных операторов и операндов и число их вхождений в коде. Операторы представлены в таблице 12, операнды – в таблице 13.

Таблица 11 – Операторы сервиса «Order»

Оператор	Количество
GetCartItem	1
return	14
lo.Map	1
fmt.Errorf	11
:=	14
!	11
=	30
string(	1
Authorize	2
CreateOrder	1
make	1
lo.Find	1
append	1
uuid.NewV7	4
CreateOrderItem	1
fmt.Sprintf	1
CreateOrderOperation	1
CreateOperationDetail	1
produceSupplyMessage	1
if	13
n1 = 20	N1 = 111

Таблица 12 – Операнды сервиса «Order»

Операнд	Количество вхождений
s	9

Продолжение таблицы 12

Операнд	Количество вхождений
ctx	9
cartItemsIds	2
itemsForOrder	4
warehouseProductItems	2
err	28
orderId	4
time.Now	2
UserId	2
ok	2
warehouseProductItem	4
totalOrderPrice	4
orderItemId	2
uErr	2
operationId	3
operationDetailId	2
nil	11
n2 = 17	N2 = 92

Длина программы рассчитывается по формуле:

$$N = N_1 + N_2,$$

где N_1 – общее число операторов в программе;

N – общее число операндов в программе.

Таким образом, на основе данных из таблицы 11, длина программы:

$$N = 111 + 93 = 203.$$

Словарь программы рассчитывается по формуле:

$$n = n_1 + n_2,$$

где n_1 – число уникальных операторов программы;

n_2 – число уникальных операндов программы.

Словарь программы на основе данных их таблицы 12:

$$n = 20 + 17 = 37.$$

Расчет вычислительная сложность программы находится по формуле:

$$HV = N * \log_2 n,$$

где N – длина программы;

n – словарь программы;

Таким образом, на основе ранее рассчитанных показателей N и n , вычислительная сложность программы составит:

$$HV = 203 * \log_2 37 = 1057,427.$$

Кроме того, был также рассчитан комплексный показатель качества кода – Maintainability Index. Этот показатель помогает оценить, насколько легко поддерживать и модифицировать код. Высокий индекс означает более легко поддерживаемый код. Расчет показателя происходит по формуле:

$$MI = 171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(LOC),$$

где HV – вычислительная сложность кода;

CC – цикломатическая сложность кода;

LOC – количество строк кода.

Показатель CC, для данного кода равен 4, количество строк кода насчитывает 105 строк, а вычислительная сложность была рассчитана выше и составляет 1057,427.

Таким образом комплексный показатель:

$$MI = 171 - 5.2 * \ln(1057,427) - 0.23 * 4 - 16.2 * \ln(105) = 58,477.$$

Результаты анализа находятся в пределах нормы, что соответствует среднему уровню сопровождаемости и сложности программного кода, а также соответствуют установленным критериям оценки качества кода. Это свидетельствует о невысокой сложности и прозрачности кодовой базы для разработчиков и простоты ее модификации.

Заключение

В рамках выпускной квалификационной работы спроектирована и разработана серверная часть для логистического сервиса.

Серверная часть логистического сервиса разработана на языке Go с использованием библиотеки Gorilla Mux для организации взаимодействия по стандартам REST. Вся система состоит из 12 микросервисов, каждый из которых отвечает за свою конкретную предметную область внутри системы.

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

На стадии проектирования определен архитектурный стиль, по которому велась разработка серверной части логистического сервиса. Спроектированы концептуальная и логическая модели базы данных приложения. Определены минимальные и рекомендуемые требования к аппаратному обеспечению, где должен будет разворачиваться логистический сервис.

Разработка серверной части логистического сервиса включала в себя реализацию 12 микросервисов на языке Go, разработку баз данных под управлением СУБД PostgreSQL, а также организацию межсервисного взаимодействия посредством использования Apache Kafka.

В рамках выпускной квалификационной работы проведен контроль качества разработанной серверной части логистического сервиса с помощью функционального и модульного тестирования. Также рассчитана метрика Холстеда для оценки качества написанного кода. По итогам проведенного контроля качества выявлена корректная работа разработанного приложения и достаточный уровень написанного кода, подтвержденный рассчитанными метриками.

Произведено планирование разработки и затраченного бюджета. Итоговые денежные затраты на разработку проекта составили 196000 рублей, а затраты трудовых ресурсов составили 408 часов. Начало работы поставлено на 8 апреля, а конец на 13 июня текущего года.

Таким образом, в результате выполнения выпускной квалификационной работы разработана серверная часть логистического сервиса, который способен в полной мере покрыть все необходимые критерии и поставленные цели и задачи. Логистический сервис обладает современным, актуальным дизайном с удобным для взаимодействия интерфейсом. Сервис обладает низкой стоимостью использования и быстротой выполнения логистических процессов, что положительно скажется на планировании затрат на различные ресурсы.

Список использованных источников

1. Медведев М.В., Антонов И.И., Балашова И.Ю. Визуализация сущностей логистического сервиса: отображение результата решения задачи маршрутизации в интерфейсе. // Вестник Пензенского государственного университета. 2024. С. 94 – 97. (Дата обращения: 17.05.2025)
2. Абдуллаев Н.А., Нажмутдинова С.А. Сервис в логистике // eor.dgu.ru. [Электронный ресурс]. URL: [https://eor.dgu.ru/lectures_f/Абдуллаев Н.А./project/p10aa1.html](https://eor.dgu.ru/lectures_f/Абдуллаев%20Н.А./project/p10aa1.html) (Дата обращения: 17.05.2025)
3. Семенов Ю.Е. Терминология логистики // Журнал об экономике и бизнесе. 2020. С. 152 – 154. [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/terminologiya-logistiki/viewer> (Дата обращения: 17.05.2025)
4. Щербакова Е.С. Транспортная логистика // Экономика и социум. 2021. №11(90)-2. С. 696 – 699. [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/transportnaya-logistika-1/viewer> (Дата обращения: 17.05.2025)
5. UML – диаграмма вариантов использования (use case diagram) // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/47940/> (дата обращения: 17.05.2025).
6. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 17.05.2025)
7. Просто о сложном. Концептуальная модель. // <https://mlkv.ru> [Электронный ресурс]. URL: <https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/> (дата обращения: 05.06.2025)
8. Верников Г. Основы методологии IDEF1X // Проект «Корпоративный менеджмент» [Электронный ресурс]. URL: <https://www.cfin.ru/vernikov/idef/idef1x.shtml/> (дата обращения 03.06.2024)

9. Клиент-серверная архитектура. // <https://ru.hexlet.io> [Электронный ресурс]. URL: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit (дата обращения: 05.06.2025)
10. Моделирование данных: обзор // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/556790/> (дата обращения: 05.06.2025)
11. Преимущества и недостатки клиент-серверной модели // <https://sky.pro> [Электронный ресурс]. URL: <https://sky.pro/wiki/gamedev/preimushhestva-i-nedostatki-klient-servernoj-modeli-v-igrah/> (дата обращения: 05.06.2025)
12. С. Ньюмен. От монолита к микросервисам. Эволюционные шаблоны для трансформации монолитной системы. // «БХВ-Петербург» 2022. С. 21. (дата обращения: 05.06.2025)
13. Д. Д. Чикилев. Виды архитектуры приложений. // Журнал «Научный лидер». 2024. С. 172. [Электронный ресурс]. URL: <https://scilead.ru/article/6673-vidi-arkhitekturi-prilozhenij> (дата обращения: 05.06.2025)
14. Моделирование данных: что это и зачем нужно // <https://practicum.yandex.ru> [Электронный ресурс]. URL: <https://practicum.yandex.ru/blog/chto-takoe-modelirovanie-dannyh-i-kak-ego-ispolzovat/> (дата обращения: 05.06.2025)
15. Юнит-тестирование для чайников // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/169381/> (дата обращения: 05.06.2025)
16. Интеграционное тестирование: виды, примеры и инструменты // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/843534/> (дата обращения: 05.06.2025)
17. Фундаментальная теория тестирования // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/549054/> (дата обращения: 05.06.2025)
18. Программный код и его метрики // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 05.06.2025)

19. Иванов, П.В. Управление проектами: Учебное пособие / П.В. Иванов. – Рн/Д: Феникс, 2018. –102 с
20. Создаем диаграмму Ганта // UNISENDER [Электронный ресурс]
URL: <https://www.unisender.com/ru/blog/что-такое-диаграмма-ганта-и-как-ее-сделать/> (Дата обращения: 05.06.2025)

Приложение А
(справочное)
Глоссарий

Серверное приложение – это программное обеспечение, которое работает на сервере и предоставляет различные сервисы или функциональность для клиентских приложений. Серверные приложения обычно используются в клиент-серверной архитектуре.

API (Application Programming Interface) – это набор правил и протоколов, которые позволяют различным программным приложениям взаимодействовать друг с другом.

REST (Representational State Transfer) – это стиль архитектуры программного обеспечения для распределенных систем, который, как правило, используется для построения веб-служб.

Слой представления данных – это первый и самый верхний уровень, который присутствует в приложении. Этот уровень представляет собой UI, то есть представление контента конечному пользователю, через интерфейс.

HTTP-запрос – это запрос, который отправляется от клиента к веб-серверу для получения или отправки данных.

Контроллер – это компонент приложения, который обрабатывает входящие запросы от пользователей или других систем.

Сервис – это слой бизнес-логики приложения.

Бизнес-логика – это набор правил, необходимых для запуска приложения в соответствии с руководящими принципами, установленными организацией.

Репозиторий – это компонент, отвечающий за взаимодействие с хранилищем данных.

Приложение Б
(обязательное)
Объекты обращения к базам данных

Листинг Б.1 – Объекты корзины

```
type Cart struct {
    Id          int          `db:"id"`
    CreateDate  time.Time   `db:"create_date"`
    UserId      int          `db:"user_id"`
    Items       []CartItem
}
type CartItem struct {
    Id          int          `db:"id"`
    ProductItemId int       `db:"product_item_id"`
    Quantity    int          `db:"quantity"`
    CartId      int          `db:"cart_id"`
    IsSelected  bool         `db:"is_selected"`
}
```

Листинг Б.2 – Объекты клиентов

```
type Client struct {
    Id      int      `db:"id"`
    Name    string   `db:"name"`
    INN     string   `db:"inn"`
    Address string   `db:"address"`
}
```

Листинг Б.3 – Объекты заказов

```
type Order struct {
    Id          uuid.UUID `db:"id"`
    Status      string    `db:"status"`
    CreateDate  time.Time `db:"create_date"`
    DeliveryDate time.Time `db:"delivery_date"`
    UserId      int       `db:"user_id"`
}
type OrderItem struct {
    Id          uuid.UUID `db:"id"`
    OrderId     uuid.UUID `db:"orders_id"`
    ProductItemId int       `db:"product_item_id"`
    Quantity    int       `db:"quantity"`
}
```

Листинг Б.4 – Объекты операций

```
type Operation struct {
    Id          uuid.UUID `db:"id"`
    Date        time.Time `db:"date"`
    Description string    `db:"description"`
    OrderId     uuid.UUID `db:"orders_id"`
}
type OperationDetail struct {
    Id          uuid.UUID `db:"id"`
    OperationId uuid.UUID `db:"operation_id"`
    Type        OperationDetailType `db:"type"`
    Price       decimal.Decimal    `db:"price"`
}
```

Листинг Б.5 – Объекты запросов

```

type Request struct {
    Id            int            `db:"id" json:"id"`
    Status        string          `db:"status" json:"status"`
    Type          string          `db:"type" json:"type"`
    UpdateDate    time.Time        `db:"update_date" json:"update_date"`
    Summary       string          `db:"summary" json:"summary"`
    Items         []RequestItem  `db:"- " json:"items"`
}

type RequestItem struct {
    Id            int            `db:"id" json:"id"`
    RequestId     int            `db:"request_id" json:"request_id"`
    ProductId     int            `db:"product_id" json:"product_id"`
    ProductItemId int            `db:"product_item_id"
json:"product_item_id"`
    ProductItemCount int        `db:"product_item_count"
json:"product_item_count"`
}

```

Листинг Б.6 – Объекты пользователей

```

type User struct {
    Id            int            `db:"id"`
    Username      string         `db:"username"`
    Password      string         `db:"password"`
    Status        int            `db:"status"`
    RoleId        int            `db:"role_id"`
    ClientId      int            `db:"client_id"`
}

type Role struct {
    Id            int            `db:"id"`
    Name          string         `db:"name"`
}

```

Листинг Б.7 – Объекты склада

```

type Product struct {
    Id            int            `db:"id" json:"id"`
    BrandName     string         `db:"brand" json:"brand"`
    Name          string         `db:"name" json:"name"`
    Price         decimal.Decimal `db:"price" json:"price"`
    Status        string         `db:"status" json:"status"`
    FactoryId     int            `db:"factory_id" json:"factory_id"`

    Items         []ProductItem
    Materials     []ProductMaterial
    Images        []ProductImages
}

type ProductMaterial struct {
    Id            int            `db:"id" json:"id"`
    MaterialId    int            `db:"material_id" json:"material_id"`
    ProductId     int            `db:"product_id" json:"product_id"`
    Material      string         `db:"name" json:"material"`
}

```

```

}
type ProductImages struct {
    Id          int          `db:"id" json:"id"`
    ProductId   int          `db:"product_id" json:"product_id"`
    ImageUrl    string       `db:"url" json:"image_url"`
}
type ProductItem struct {
    Id          int          `db:"id" json:"id"`
    ProductId   int          `db:"product_id" json:"productId"`
    StockCount  int          `db:"stock_count"
json:"stockCount"`
    Size        int          `db:"size" json:"size"`
    Weight       decimal.Decimal `db:"weight" json:"weight"`
    Color       string       `db:"color" json:"color"`
}

```

Приложение В
(обязательное)
SQL-скрипты

Листинг В.1 – Скрипты создания таблиц

```

create table if not exists cart
(
    id          serial primary key,
    create_date timestamp,
    user_id     int
);

create table if not exists cart_item
(
    id          serial primary key,
    cart_id     int references cart (id),
    product_item_id int,
    quantity    int,
    is_selected bool not null default false
);

create table if not exists client
(
    id          serial primary key,
    name        text not null default '',
    inn         text,
    address     text
);

create table if not exists orders
(
    id          uuid primary key,
    status       text not null,
    create_date  timestamp not null,
    delivery_date timestamp,
    user_id     int
);

create table if not exists orders_item
(
    id          uuid primary key,
    orders_id   uuid references orders (id),
    product_item_id int,
    quantity    int
);

create table if not exists operation
(
    id          uuid primary key,
    date        timestamp not null,
    description text,
    orders_id   uuid references orders (id)
);

create table if not exists operation_detail
(

```

```

        id          uuid primary key,
        operation_id uuid references operation (id),
        type        int      not null,
        price       decimal not null
    );

create table if not exists request
(
    id          serial primary key,
    status      text not null default 0,
    type        text not null default 0,
    update_date date,
    summary     text
);

create table if not exists request_item
(
    id          serial primary key,
    request_id  int,
    product_id  int,
    product_item_id int,
    product_item_count int
);

create table if not exists user_role
(
    id  serial primary key,
    name text not null default ''
);

create table if not exists users
(
    id          serial primary key,
    username    text not null default '',
    password    text not null default '',
    status      int,
    role_id     int references user_role (id),
    client_id   int
);

create table if not exists product
(
    id          serial primary key,
    name        text      default ''::text not null,
    brand       text      default ''::text not null,
    price       numeric default 0.0      not null,
    material    text      default ''::text not null,
    status      text      default ''::text not null,
    factory_id integer
);

create table if not exists product_item
(
    id          serial primary key,

```

```

        product_id integer references product(id),
        color      text      default ''::text not null,
        size       numeric default 0.0      not null,
        weight     numeric default 0.0      not null,
        quantity   integer default 0        not null
    );

```

```

create table if not exists material
(
    id      serial primary key,
    name text not null
);

```

```

create table if not exists product_material
(
    id          serial primary key,
    product_id integer references product(id),
    material_id integer references material(id)
);

```

```

create table if not exists image
(
    id          serial primary key,
    product_id integer references product(id),
    url         text default ''::text not null
);

```

Листинг В.2 – Скрипты вставки данных

```

INSERT INTO product (name, brand, price, status, factory_id)
VALUES (:name, :brand, :price, :status, :factory_id)
RETURNING id;

```

```

INSERT INTO product_item (product_id, color, size, weight,
quantity)
VALUES (:product_id, :color, :size, :weight, :stock_count)
RETURNING id;

```

```

INSERT INTO image (product_id, url)
VALUES (:product_id, :url)
RETURNING id;

```

```

INSERT INTO product_material (product_id, material_id)
VALUES (:product_id, :material_id)
RETURNING id;

```

```

insert into users (username, password, status, role_id, client_id)
values ($1, $2, $3, $4, $5)
returning id

```

```

insert into request (status, type, update_date, summary)
values (:status, :type, :update_date, :summary)
returning id

```

```

insert into request_item (request_id, product_id, product_item_id,
product_item_count)
values (:request_id, :product_id, :product_item_id,
:product_item_count)

insert into orders (id, status, create_date, delivery_date,
user_id)
values (:id, :status, :create_date, :delivery_date, :user_id)

insert into orders_item (id, orders_id, product_item_id, quantity)
values (:id, :orders_id, :product_item_id, :quantity)

insert into operation (id, date, description, orders_id)
values (:id, :date, :description, :orders_id)

insert into operation_detail (id, operation_id, type, price)
values (:id, :operation_id, :type, :price)

insert into client (name, inn, address)
VALUES ($1, $2, $3)
returning id

insert into cart (create_date, user_id)
values (:create_date, :user_id)

insert into cart_item (cart_id, product_item_id, quantity,
is_selected)
values (:cart_id, :product_item_id, :quantity, :is_selected)
returning id

```

Листинг В.3 – Скрипты получения данных

```

select id,
       create_date,
       user_id
from cart
where user_id = $1

select id,
       name,
       inn,
       address
from client
where id = any($1)

select id,
       date,
       description,
       orders_id
from operation
where orders_id = $1

select id,

```

```

        status,
        create_date,
        delivery_date,
        user_id
from orders
where user_id = $1

```

```

select id,
       status,
       type,
       update_date,
       summary
from request
where status = 'pending'

```

```

select id,
       username,
       password,
       status,
       role_id,
       client_id
from users
where username = $1

```

```

select pit.id      as id,
       p.id        as product_id,
       p.factory_id as factory_id,
       p.name       as name,
       p.price      as price,
       pit.color    as color,
       pit.size     as size,
       pit.weight   as weight,
       pit.quantity as count
from product_item pit
  join product p on p.id = pit.product_id
where pit.id = any ($1)

```

Приложение Г
(обязательное)
Структуры слоя представления данных

Листинг Г.1 – Слой представления auth-service

```
func (s *Server) Setup(authService service.AuthService,
registrationService service.Registration) {
    s.server.Post("/sign-up",
handlers.SignupHandler(registrationService))
    s.server.Post("/login", handlers.LoginHandler(authService))
    s.server.Post("/logout", handlers.LogoutHandler(authService))
    s.server.Post("/refresh-token",
handlers.RefreshHandler(authService))
    s.server.Post("/password/new",
handlers.UpdatePasswordHandler(authService))
}
```

Листинг Г.2 – Слой представления cart-service

```
func (r *Router) SetupRoutes(logger goatlogger.Logger, cartService
service.Cart) {
    r.router.HandleFunc("/cart", handlers.GetCartHandler(logger,
cartService)).Methods(http.MethodGet)
    r.router.HandleFunc("/cart/item",
handlers.AddCartItemHandler(logger,
cartService)).Methods(http.MethodPost)
    r.router.HandleFunc("/cart/item",
handlers.UpdateCartItemHandler(logger,
cartService)).Methods(http.MethodPut)
    r.router.HandleFunc("/cart/item/{id}",
handlers.DeleteItemHandler(logger,
cartService)).Methods(http.MethodDelete)
    r.router.HandleFunc("/cart/clear",
handlers.ClearCartHandler(logger,
cartService)).Methods(http.MethodDelete)
    r.router.HandleFunc("/cart/items",
handlers.GetCartItemsHandler(logger,
cartService)).Methods(http.MethodPost)
}
```

Листинг Г.3 – Слой представления client-service

```
func (r *Router) SetupRoutes(logger goatlogger.Logger,
clientService service.Client) {
    r.router.HandleFunc("/clients",
handlers.GetClientsHandler(logger,
clientService)).Methods(http.MethodGet)
    r.router.HandleFunc("/client/{id}",
handlers.GetClientHandler(logger,
clientService)).Methods(http.MethodGet)
    r.router.HandleFunc("/client",
handlers.AddClientHandler(logger,
clientService)).Methods(http.MethodPost)
    r.router.HandleFunc("/client",
handlers.UpdateClientHandler(logger,
clientService)).Methods(http.MethodPut)
```

```

    r.router.HandleFunc("/client/{id}",
handlers.DeleteClientHandler(logger,
clientService)).Methods(http.MethodDelete)
    r.router.HandleFunc("/info/short",
handlers.GetInfoShort(logger,
clientService)).Methods(http.MethodPost)
}

```

Листинг Г.4 – Слой представления notifier-service

```

func (r *Router) SetupRoutes(logger goatlogger.Logger, sender
service.Sender) {
    r.router.HandleFunc("/mail", handlers.MainSendHandler(logger,
sender)).Methods(http.MethodPost)
}

```

Листинг Г.5 – Слой представления order-service

```

func (r *Router) SetupRoutes(logger goatlogger.Logger,
orderService service.Order) {
    r.router.HandleFunc("/order",
handlers.CreateOrderHandler(logger,
orderService)).Methods(http.MethodPost)
    r.router.HandleFunc("/orders", handlers.GetUserOrders(logger,
orderService)).Methods(http.MethodGet)
    r.router.HandleFunc("/orders/latest",
handlers.GetLatestOrdersHandler(logger,
orderService)).Methods(http.MethodGet)
}

```

Листинг Г.6 – Слой представления report-service

```

func (r *Router) SetupRoutes(logger goatlogger.Logger,
reportService service.Report) {
    r.router.HandleFunc("/report/user/{id}/{date}",
handlers.GetUserReportHandler(logger,
reportService)).Methods(http.MethodGet)
    r.router.HandleFunc("/report/factory/{id}/{date}",
handlers.GetFactoryReportHandler(logger,
reportService)).Methods(http.MethodGet)
}

```

Листинг Г.7 – Слой представления request-service

```

func (r *Router) SetupRoutes(logger goatlogger.Logger,
requestService service.Request) {
    r.router.HandleFunc("/requests",
handlers.GetAllRequestsHandler(logger,
requestService)).Methods(http.MethodGet)
    r.router.HandleFunc("/request/{id}/{status}",
handlers.UpdateRequestStatusHandler(logger,
requestService)).Methods(http.MethodPut)
}

```

Листинг Г.8 – Слой представления search-service

```
func (r *Router) SetupRoutes(logger goatlogger.Logger,
searchService service.Search) {
    r.router.HandleFunc("/filters/active",
handlers.GetFiltersHandlers(logger,
searchService)).Methods(http.MethodGet)
    r.router.HandleFunc("/catalog",
handlers.GetCatalogHandler(logger,
searchService)).Methods(http.MethodGet)
    r.router.HandleFunc("/catalog/{productId}",
handlers.GetProductCatalog(logger,
searchService)).Methods(http.MethodGet)
}
```

Листинг Г.9 – Слой представления user-service

```
func (r *Router) SetupRoutes(logger goatlogger.Logger, userService
service.User) {
    r.router.HandleFunc("/users", handlers.GetUsersHandler(logger,
userService)).Methods(http.MethodGet)
    r.router.HandleFunc("/user",
handlers.GetUserByUsername(logger,
userService)).Methods(http.MethodGet)
    r.router.HandleFunc("/user/{id}",
handlers.GetUserHandler(logger,
userService)).Methods(http.MethodGet)
    r.router.HandleFunc("/user", handlers.AddUserHandler(logger,
userService)).Methods(http.MethodPost)
    r.router.HandleFunc("/user",
handlers.UpdateUserHandler(logger,
userService)).Methods(http.MethodPut)
    r.router.HandleFunc("/user/{id}",
handlers.DeleteUserHandler(logger,
userService)).Methods(http.MethodDelete)
    r.router.HandleFunc("/user/registration",
handlers.RegisterHandler(logger,
userService)).Methods(http.MethodPost)
    r.router.HandleFunc("/user/password/new",
handlers.UpdateUserPasswordHandler(logger,
userService)).Methods(http.MethodPut)
}
```

Листинг Г.10 – Слой представления warehouse-service

```
func (r *Router) SetupRoutes(logger goatlogger.Logger, warehouse
service.WareHouse) {
    r.router.HandleFunc("/products",
handlers.GetProductsHandler(logger,
warehouse)).Methods(http.MethodGet)
    r.router.HandleFunc("/product/client/{id}",
handlers.GetClientProductsHandler(logger,
warehouse)).Methods(http.MethodGet)
```

```

        r.router.HandleFunc("/product/{productId}",
handlers.GetDetailedProductHandler(logger,
warehouse)).Methods(http.MethodGet)
        r.router.HandleFunc("/materials/list",
handlers.GetMaterialsList(logger,
warehouse)).Methods(http.MethodGet)
        r.router.HandleFunc("/products",
handlers.AddProductsHandler(logger,
warehouse)).Methods(http.MethodPost)
        r.router.HandleFunc("/products",
handlers.UpdateProductsHandler(logger,
warehouse)).Methods(http.MethodPut)
        r.router.HandleFunc("/products",
handlers.DeleteProductsHandler(logger,
warehouse)).Methods(http.MethodDelete)
        r.router.HandleFunc("/products/items",
handlers.GetProductItemsInfoHandler(logger,
warehouse)).Methods(http.MethodPost)
        r.router.HandleFunc("/products/detailed",
handlers.GetDetailedProductsHandler(logger,
warehouse)).Methods(http.MethodPost)
    }

```

Приложение Д
(обязательное)
Структуры слоя бизнес-логики

Листинг Д.1 – Слой бизнес-логики auth-service

```

type AuthService interface {
    Login(username, password string) (domain.TokenResponse, error)
    Logout(refreshToken string)
    RefreshToken(token string) (domain.TokenResponse, error)
    UpdatePassword(request domain.UpdatePasswordRequest) error
}
type AuthServiceImpl struct {
    hasher      PasswordHasher
    jwtService   JwtService
    userService *userservice.Client
}
func NewAuthService(hasher PasswordHasher, jwtService JwtService,
    userClient *userservice.Client) AuthService {
    return &AuthServiceImpl{
        hasher:      hasher,
        jwtService:   jwtService,
        userService: userClient,
    }
}

```

Листинг Д.2 – Слой бизнес-логики cart-service

```

type Cart interface {
    GetCart(ctx goatcontext.Context) (domain.Cart, error)
    AddCartItem(ctx goatcontext.Context, item domain.CartItem)
    (domain.CartItem, error)
    RemoveCartItem(ctx goatcontext.Context, id int) error
    UpdateCartItem(ctx goatcontext.Context, cartItem
    domain.CartItem) error
    ClearCartItems(ctx goatcontext.Context) error
    GetCartItems(ctx goatcontext.Context, ids []int)
    ([]domain.CartItem, error)
}
type CartServiceImpl struct {
    cartRepository repository.Cart
    warehouseClient *warehouse.Client
}
func NewCartServiceImpl(cartRepository repository.Cart,
    warehouseClient *warehouse.Client) Cart {
    return &CartServiceImpl{
        cartRepository: cartRepository,
        warehouseClient: warehouseClient,
    }
}

```

Листинг Д.3 – Слой бизнес-логики client-service

```

type Client interface {
    GetClients(ctx goatcontext.Context) ([]domain.Client, error)
    GetClient(ctx goatcontext.Context, id int) (domain.Client,
    error)
}

```

```

        AddClient(ctx goatcontext.Context, client domain.Client)
        (domain.Client, error)
        UpdateClient(ctx goatcontext.Context, client domain.Client)
    error
        DeleteClient(ctx goatcontext.Context, id int) error
        GetClientsByIds(ctx goatcontext.Context, ids []int)
        ([]domain.Client, error)
    }
    type ClientServiceImpl struct {
        clientRepository repository.Client
    }
    func NewClientService(clientRepository repository.Client) Client {
        return &ClientServiceImpl{
            clientRepository: clientRepository,
        }
    }
}

```

Листинг Д.4 – Слой бизнес-логики notifier-service

```

type Sender interface {
    Send(message domain.Mail) error
}
type SenderImpl struct {
    receiver    string
    sender      string
    smtpClient  *client.Smtп
}
func NewSender(receiver string, sender string, smtpClient
*client.Smtп) Sender {
    return &SenderImpl{
        receiver: receiver,
    }
}

```

Листинг Д.5 – Слой бизнес-логики order-service

```

type Order interface {
    Order(ctx goatcontext.Context, cartItemsIds []int) error
    GetUserOrders(ctx goatcontext.Context) ([]domain.Order, error)
    GetLatestOrders(ctx goatcontext.Context) ([]domain.ReportItem,
error)
}
type OrderServiceImpl struct {
    cartService          *cart.Client
    warehouseService     *warehouse.Client
    orderRepository      repository.Order
    financeRepository    repository.Finance
    kafkaProducer        *kafka.Producer
}
func NewOrderService(cartService *cart.Client, warehouseService
*warehouse.Client, orderRepository repository.Order,
financeRepository repository.Finance) Order {
    return &OrderServiceImpl{
        cartService:      cartService,
    }
}

```

```

        warehouseService: warehouseService,
        orderRepository:  orderRepository,
        financeRepository: financeRepository,
    }
}

```

Листинг Д.6 – Слой бизнес-логики report-service

```

type Report interface {
    GetFactoryReport(ctx goatcontext.Context, factoryId int, date
time.Time) (*excelize.File, error)
    GetUserReport(ctx goatcontext.Context, userId int, date
time.Time) (*excelize.File, error)
}
type ReportService struct {
    reportRepository repository.Report
}
func NewReportService(reportRepository repository.Report) Report {
    return &ReportService{
        reportRepository: reportRepository,
    }
}

```

Листинг Д.7 – Слой бизнес-логики request-service

```

type Request interface {
    GetRequests(ctx goatcontext.Context) ([]domain.Request, error)
    UpdateStatus(ctx goatcontext.Context, id int, status
domain.Status) error
}
type Impl struct {
    requestRepository repository.Request
    warehouseClient    *warehouse.Client
    notifierClient      *notifier.Client
}
func NewRequestService(requestRepository repository.Request,
warehouseClient *warehouse.Client, notifierClient
*notifier.Client) Request {
    return &Impl{
        requestRepository: requestRepository,
        warehouseClient:   warehouseClient,
        notifierClient:     notifierClient,
    }
}

```

Листинг Д.8 – Слой бизнес-логики search-service

```

type Search interface {
    GetFilters(ctx goatcontext.Context) ([]domain.Filter, error)
    GetCatalog(ctx goatcontext.Context, searchId string,
appliedFilters domain.AppliedFilters) (domain.Catalog, error)
    GetProductCatalog(ctx goatcontext.Context, productId int)
(domain.Product, error)
}
type SearchService struct {

```

```

    filterRepository repository.Filter
    cacheRepository repository.Cache
    warehouseClient *warehouse.Client
}
func New(filterRepository repository.Filter, cacheRepository
repository.Cache, warehouseClient *warehouse.Client) Search {
    return &SearchService{
        filterRepository: filterRepository,
        cacheRepository: cacheRepository,
        warehouseClient: warehouseClient,
    }
}

```

Листинг Д.9 – Слой бизнес-логики user-service

```

type User interface {
    GetUsers(ctx goatcontext.Context) ([]domain.User, error)
    GetUserById(ctx goatcontext.Context, userId int) (domain.User,
error)
    GetUserByUsername(ctx goatcontext.Context, userName string)
(domain.User, error)
    AddUser(ctx goatcontext.Context, user domain.User)
(domain.User, error)
    UpdateUser(ctx goatcontext.Context, user domain.User) error
    DeleteUser(ctx goatcontext.Context, userId int) error
    UpdateUserPassword(ctx goatcontext.Context, request
domain.UpdatePasswordRequest) error
}
type UserServiceImpl struct {
    userRepository repository.User
    roleRepository repository.Role
    notifierClient *notifier.Client
}
func NewUserService(userRepository repository.User, roleRepository
repository.Role, notifierClient *notifier.Client) User {
    return &UserServiceImpl{
        userRepository: userRepository,
        roleRepository: roleRepository,
        notifierClient: notifierClient,
    }
}

```

Листинг Д.10 – Слой бизнес-логики warehouse-service

```

type WareHouse interface {
    GetProducts(ctx goatcontext.Context) ([]domain.Product, error)
    AddProducts(ctx goatcontext.Context, products
[]domain.Product) error
    UpdateProducts(ctx goatcontext.Context, products
[]domain.Product) error
    DeleteProducts(ctx goatcontext.Context, productIds []int)
error
}
type Impl struct {

```

```
    warehouseRepository repository.Warehouse
    clientServiceClient *clientservice.Client
    kafkaProducer       *broker.Producer
}
func NewWarehouseService(warehouseRepository repository.Warehouse,
    clientServiceClient *clientservice.Client, kafkaProducer
    *broker.Producer) WareHouse {
    return &Impl{
        warehouseRepository: warehouseRepository,
        clientServiceClient: clientServiceClient,
        kafkaProducer:       kafkaProducer,
    }
}
```

Приложение Е
(обязательное)
Структуры слоя обращения к базе данных

Листинг Е.1 – Слой обращения к базе данных cart-service

```

type Cart interface {
    GetCart(ctx goatcontext.Context) (cart database.Cart, err
error)
    CreateCart(ctx goatcontext.Context, cart database.Cart) (id
int64, err error)
    GetCartItems(ctx goatcontext.Context, cartId int) (items
[]database.CartItem, err error)
    AddCartItem(ctx goatcontext.Context, cartItem
database.CartItem) (id int64, err error)
    UpdateCartItem(ctx goatcontext.Context, cartItem
database.CartItem) error
    DeleteCartItems(ctx goatcontext.Context, cartItemIds []int)
error
    ClearCartItems(ctx goatcontext.Context, cartId int) error
    GetCartItemsByIds(ctx goatcontext.Context, ids []int) (items
[]database.CartItem, err error)
}
type CartRepositoryImpl struct {
    postgres *sqlx.DB
}
func NewCartRepository(postgres *sqlx.DB) Cart {
    return &CartRepositoryImpl{
        postgres: postgres,
    }
}

```

Листинг Е.2 – Слой обращения к базе данных client-service

```

type Client interface {
    GetClients(ctx goatcontext.Context) (clients
[]database.Client, err error)
    GetClient(ctx goatcontext.Context, id int) (client
database.Client, err error)
    AddClient(ctx goatcontext.Context, client database.Client) (id
int, err error)
    UpdateClient(ctx goatcontext.Context, client database.Client)
error
    DeleteClient(ctx goatcontext.Context, id int) error
    GetClientsByIds(ctx goatcontext.Context, ids []int) (clients
[]database.Client, err error)
}
type ClientRepositoryImpl struct {
    postgres *sqlx.DB
}
func NewClientRepository(postgres *sqlx.DB) Client {
    return &ClientRepositoryImpl{
        postgres: postgres,
    }
}

```

Листинг Е.3 – Слой обращения к базе данных order-service

```

type Order interface {
    GetUserOrders(ctx goatcontext.Context, userId int) (orders
[]database.Order, err error)
    CreateOrder(ctx goatcontext.Context, order database.Order)
error
    GetOrderItems(ctx goatcontext.Context, orderId uuid.UUID)
(items []database.OrderItem, err error)
    CreateOrderItem(ctx goatcontext.Context, orderItem
database.OrderItem) error
    GetLatestOrders(ctx goatcontext.Context, startDate, endDate
time.Time) (orders []database.LatestOrder, err error)
}
type OrderRepositoryImpl struct {
    postgres *sqlx.DB
}
func NewOrderRepository(postgres *sqlx.DB) Order {
    return &OrderRepositoryImpl{
        postgres: postgres,
    }
}
type Finance interface {
    GetOrderOperation(ctx goatcontext.Context, orderId uuid.UUID)
(operation database.Operation, err error)
    CreateOrderOperation(ctx goatcontext.Context, operation
database.Operation) error
    GetOperationDetails(ctx goatcontext.Context, operationId
uuid.UUID) (details []database.OperationDetail, err error)
    CreateOperationDetail(ctx goatcontext.Context, operation
database.OperationDetail) error
}
type FinanceRepositoryImpl struct {
    postgres *sqlx.DB
}
func NewFinanceRepository(postgres *sqlx.DB) Finance {
    return &FinanceRepositoryImpl{
        postgres: postgres,
    }
}

```

Листинг Е.4– Слой обращения к базе данных request-service

```

type Request interface {
    GetRequestById(ctx goatcontext.Context, id int) (request
database.Request, err error)
    GetPendingRequests(ctx goatcontext.Context)
([]database.Request, error)
    AddRequest(ctx goatcontext.Context, request database.Request)
error
    UpdateRequestStatus(ctx goatcontext.Context, id int, status
string) error
}
type Impl struct {
    postgres *sqlx.DB
}

```

```
func NewRequestRepository(postgres *sqlx.DB) Request {
    return &Impl{
        postgres: postgres,
    }
}
```

Листинг Е.5 – Слой обращения к базе данных user-service

```
type User interface {
    GetUsers(ctx goatcontext.Context) ([]database.User, error)
    GetUserById(ctx goatcontext.Context, userId int)
(database.User, error)
    GetUserByUsername(ctx goatcontext.Context, username string)
(database.User, error)
    AddUser(ctx goatcontext.Context, user database.User) (int,
error)
    UpdateUser(ctx goatcontext.Context, user database.User) error
    DeleteUser(ctx goatcontext.Context, userId int) error
}
type UserRepositoryImpl struct {
    postgres *sqlx.DB
}
func NewUserRepository(postgres *sqlx.DB) User {
    return &UserRepositoryImpl{
        postgres: postgres,
    }
}
type Role interface {
    GetRoleIdByName(ctx goatcontext.Context, name string) (int,
error)
    GetRoleById(ctx goatcontext.Context, id int) (name string, err
error)
}
type RoleRepository struct {
    postgres *sqlx.DB
}
func NewRoleRepository(postgres *sqlx.DB) Role {
    return &RoleRepository{
        postgres: postgres,
    }
}
```

Листинг Е.6 – Слой обращения к базе данных warehouse-service

```
type Warehouse interface {
    GetProducts(ctx goatcontext.Context) ([]models.Product, error)
    AddProducts(ctx goatcontext.Context, products
[]models.Product) ([]models.Product, error)
    UpdateProducts(ctx goatcontext.Context, products
[]models.Product) error
    DeleteProducts(ctx goatcontext.Context, productIds []int)
error
}
type Init struct {
```

```
    postgres *sqlx.DB
}
func NewWarehouseRepository(postgres *sqlx.DB) Warehouse {
    return &Init{
        postgres: postgres,
    }
}
```

Приложение Ж
(обязательное)
Листинг модульных тестов

Листинг Ж.1 – Файл request_test.go

```

package service
import (
    "errors"
    "request-service/cluster/notifier"
    "request-service/cluster/warehouse"
    "request-service/database"
    "request-service/domain"
    "testing"
    "time"

    "github.com/GOAT-prod/goatcontext"
    "github.com/stretchr/testify/assert"
    "github.com/stretchr/testify/mock"
)

type MockRequestRepository struct {
    mock.Mock
}

func (m *MockRequestRepository) GetPendingRequests(ctx
goatcontext.Context) ([]database.Request, error) {
    args := m.Called(ctx)
    return args.Get(0).([]database.Request), args.Error(1)
}

func (m *MockRequestRepository) UpdateRequestStatus(ctx
goatcontext.Context, id int, status string) error {
    args := m.Called(ctx, id, status)
    return args.Error(0)
}

func (m *MockRequestRepository) GetRequestById(ctx
goatcontext.Context, id int) (database.Request, error) {
    args := m.Called(ctx, id)
    return args.Get(0).(database.Request), args.Error(1)
}

type MockWarehouseClient struct {
    mock.Mock
}

func (m *MockWarehouseClient) GetDetailedProduct(ctx
goatcontext.Context, productId string) (domain.Product, error) {
    args := m.Called(ctx, productId)
    return args.Get(0).(domain.Product), args.Error(1)
}

type MockNotifierClient struct {
    mock.Mock
}

```

```

func (m *MockNotifierClient) SendMail(ctx goatcontext.Context,
mail notifier.Mail) error {
    args := m.Called(ctx, mail)
    return args.Error(0)
}

func TestImpl_UpdateStatus(t *testing.T) {
    ctx := goatcontext.NewContext()
    requestId := 123
    productId := "product-1"

    testProduct := domain.Product{
        Id:    productId,
        Name:  "Test Product",
        Items: []domain.ProductItem{
            {
                Id:    productId,
                Color: "Red",
                Size:  "M",
            },
        },
    }

    testRequest := database.Request{
        Id:          requestId,
        Status:      "PENDING",
        Type:        "SUPPLY",
        UpdateDate: time.Now(),
        Items: []database.RequestItem{
            {
                ProductId:    productId,
                ProductItemCount: 5,
            },
        },
    }

    t.Run("успешное обновление статуса (не ApprovedStatus)",
func(t *testing.T) {
    mockRepo := new(MockRequestRepository)
    mockWarehouse := new(MockWarehouseClient)
    mockNotifier := new(MockNotifierClient)

    service := NewRequestService(mockRepo,
&warehouse.Client{Client: mockWarehouse}, &notifier.Client{Client:
mockNotifier})

    mockRepo.On("UpdateRequestStatus", ctx, requestId,
"REJECTED").Return(nil)

    err := service.UpdateStatus(ctx, requestId,
domain.Status("REJECTED"))

    assert.NoError(t, err)

```

```

        mockRepo.AssertExpectations(t)
        mockWarehouse.AssertNotCalled(t, "GetDetailedProduct")
        mockNotifier.AssertNotCalled(t, "SendMail")
    })

    t.Run("успешное обновление статуса до ApprovedStatus", func(t
    *testing.T) {
        mockRepo := new(MockRequestRepository)
        mockWarehouse := new(MockWarehouseClient)
        mockNotifier := new(MockNotifierClient)

        service := NewRequestService(mockRepo,
        &warehouse.Client{Client: mockWarehouse}, &notifier.Client{Client:
        mockNotifier})

        mockRepo.On("UpdateRequestStatus", ctx, requestId,
        "APPROVED").Return(nil)
        mockRepo.On("GetRequestById", ctx,
        requestId).Return(testRequest, nil)
        mockWarehouse.On("GetDetailedProduct", ctx,
        productId).Return(testProduct, nil)
        mockNotifier.On("SendMail", ctx,
        mock.Anything).Return(nil)

        err := service.UpdateStatus(ctx, requestId,
        domain.ApprovedStatus)

        assert.NoError(t, err)
        mockRepo.AssertExpectations(t)

        time.Sleep(100 * time.Millisecond)
        mockWarehouse.AssertExpectations(t)
        mockNotifier.AssertExpectations(t)
    })

    t.Run("ошибка при обновлении статуса", func(t *testing.T) {
        mockRepo := new(MockRequestRepository)
        mockWarehouse := new(MockWarehouseClient)
        mockNotifier := new(MockNotifierClient)

        service := NewRequestService(mockRepo,
        &warehouse.Client{Client: mockWarehouse}, &notifier.Client{Client:
        mockNotifier})

        expectedErr := errors.New("database error")
        mockRepo.On("UpdateRequestStatus", ctx, requestId,
        "APPROVED").Return(expectedErr)

        err := service.UpdateStatus(ctx, requestId,
        domain.ApprovedStatus)

        assert.Error(t, err)
        assert.Equal(t, expectedErr, err)
    })

```

```

        mockRepo.AssertExpectations(t)
        mockWarehouse.AssertNotCalled(t, "GetDetailedProduct")
        mockNotifier.AssertNotCalled(t, "SendMail")
    })
    t.Run("ошибка при отправке сообщения после обновления до
ApprovedStatus", func(t *testing.T) {
        mockRepo := new(MockRequestRepository)
        mockWarehouse := new(MockWarehouseClient)
        mockNotifier := new(MockNotifierClient)

        service := NewRequestService(mockRepo,
&warehouse.Client{Client: mockWarehouse}, &notifier.Client{Client:
mockNotifier})

        mockRepo.On("UpdateRequestStatus", ctx, requestId,
"APPROVED").Return(nil)
        mockRepo.On("GetRequestById", ctx,
requestId).Return(testRequest, nil)
        mockWarehouse.On("GetDetailedProduct", ctx,
productId).Return(testProduct, nil)
        mockNotifier.On("SendMail", ctx,
mock.Anything).Return(errors.New("notification error"))

        err := service.UpdateStatus(ctx, requestId,
domain.ApprovedStatus)

        assert.NoError(t, err)
        mockRepo.AssertExpectations(t)

        time.Sleep(100 * time.Millisecond)
        mockWarehouse.AssertExpectations(t)
        mockNotifier.AssertExpectations(t)
    })
}

```

Приложение И
(рекомендуемое)
Спецификация атрибутов таблиц

Таблица И.1 – Спецификация атрибутов таблицы users

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор пользователя	integer	15	PK	Да		
2	username	Имя пользователя	text	255		Да		пустая строка
3	password	Пароль	text	255		Да		пустая строка
4	status	Статус пользователя	integer	1				
5	role_id	Идентификатор роли	integer	15	FK			
6	client_id	Идентификатор клиента	integer	15	FK			

Таблица И.2 – Спецификация атрибутов таблицы user_role

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор роли	integer	15	PK	Да		
2	name	Наименование роли	text	20		Да		shop

Таблица И.3 – Спецификация атрибутов таблицы client

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор клиента	integer	15	PK	Да		
2	name	Наименование клиента	text	255		Да		пустая строка
3	inn	Инд клиента	text	12		Да		пустая строка
4	address	Адрес клиента	text	255		Да		пустая строка

Таблица И.4 – Спецификация атрибутов таблицы product

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор продукта	integer	15	PK	Да		
2	name	Название продукта	text	255		Да		пустая строка
3	brand	Бренд производитель	text	30		Да		пустая строка
4	price	Стоимость продукта	decimal	1		Да		пустая строка
5	material	Краткое описание материала	text	100	FK			
6	status	Статус продукта	text	10	FK	Да		
7	factory_id	Идентификатор производителя	integer	15	FK			

Таблица И.5 – Спецификация атрибутов таблицы product_item

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор варианта продукта	integer	15	PK	Да		
2	color	Цвет	text	30				
3	size	Размер	decimal	6				
4	weight	Вес	decimal	6				
5	quantity	Количество на складе	integer	3				
6	product_id	Идентификатор продукта	integer	15	FK			

Таблица И.6 – Спецификация атрибутов таблицы material

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор материала	integer	15	PK	Да		
2	name	Наименование материала	text	20		Да		

Таблица И.7 – Спецификация атрибутов таблицы product_material

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор записи	integer	15	PK	Да		
2	product_id	Идентификатор продукта	integer	15	FK	Да		
3	material_id	Идентификатор материала	integer	15	FK	Да		

Таблица И.8 – Спецификация атрибутов таблицы image

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор картинки	integer	15	PK	Да		
2	product_id	Идентификатор продукта	integer	15	FK	Да		
3	url	Ссылка на картинку	text	255		Да		

Таблица И.9 – Спецификация атрибутов таблицы cart

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
-------	--------------	----------	------------	--------	------	--------------	-------------	-----------------------

Продолжение таблицы И.9

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор корзины	integer	15	PK	Да		
2	user_id	Идентификатор пользователя	integer	15	FK	Да		
3	create_date	Дата создания	timestamp	20		Да		0001-01-01

Таблица И.10 – Спецификация атрибутов таблицы cart_item

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор содержимого корзины	integer	15	PK	Да		
2	cart_id	Идентификатор корзины	integer	15	FK	Да		
3	product_item_id	Идентификатор варианта продукта	integer	15	FK	Да		
4	quantity	Количество	integer	15				
5	is_selected	Выбран	boolean	1				

Таблица И.11 – Спецификация атрибутов таблицы orders

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор заказа	uuid	20	PK	Да		
2	status	Статус заказа	text	20		Да		created
3	create_date	Дата создания	timestamp	20		Да		0001-01-01
4	delivery_date	Дата доставки	timestamp	20				0001-01-01
5	user_id	Идентификатор пользователя	integer	15	FK	Да		

Таблица И.12 – Спецификация атрибутов таблицы orders_item

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор содержимого заказа	uuid	20	PK	Да		

Продолжение таблицы И.12

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
2	orders_id	Идентификатор заказа	uuid	20	FK	Да		
3	product_item_id	Идентификатор варианта продукта	integer	15	FK	Да		
4	quantity	Количество	integer	3				

Таблица И.13 – Спецификация атрибутов таблицы operation

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор операции	uuid	20	PK	Да		
2	orders_id	Идентификатор заказа	uuid	20	FK	Да		
3	date	Дата операции	timestamp	20		Да		0001-01-01
4	description	Описание операции	text	255				

Таблица И.14 – Спецификация атрибутов таблицы operation_detail

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор детали операции	uuid	20	PK	Да		
2	operation_id	Идентификатор операции	uuid	20	FK	Да		
3	type	Тип операции	integer	2		Да		0
4	price	Сумма операции	decimal	20				

Таблица И.15 – Спецификация атрибутов таблицы request

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор заявки	integer	15	PK	Да		
2	status	Статус заявки	text	20		Да		undefined
3	type	Тип заявки	text	20		Да		undefined
4	update_date	Дата последнего изменения	timestamp	20				0001-01-01
5	summary	Описание	text	255				

Таблица И.16 – Спецификация атрибутов таблицы request_item

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор содержимого заявки	integer	15	PK	Да		
2	request_id	Идентификатор заявки	integer	15	FK	Да		
3	product_id	Идентификатор продукта	integer	15	FK	Да		
4	product_item_id	Идентификатор варианта продукта	integer	15	FK	ДА		
5	product_item_count	Количество варианта продукта	integer	3				0

Приложение К
(рекомендуемое)

Иллюстрационные материалы к тестированию API

Curl

curl -X 'POST' \
'http://localhost:8080/auth/register' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
 "address": "Пенза",
 "inn": "42453634",
 "name": "ООО Умня",
 "password": "12341234",
 "role": "shop",
 "username": "ilya@yandex.ru"
}'

Request URL

http://localhost:8080/auth/register

Server response

Code	Details
200	<div>Response body</div> <div><pre>{ "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE5NjYyODY0YjZmZm9wIiwidXN1c19pZCI6NSwiZXN1cm5hbWUiOiJpbHlhQ1hbmRleC5ydSJ9.K3BM_qq0b8moYBf6qzcc6eHTMauKEA3mxkkmz6xLw0", "refresh_token": "eyJ1c2VyX2lkIjo1LlC1c2VyX25hbWUiOiJpbHlhQ1hbmRleC5ydSI6InVzZXJfcm9sZSI6Inob3AiFQ==.edc8d887f5754c93e8875e7b78dcca38fd9e1de6063a2dad301209f0"}</pre></div> <div>Response headers</div> <div><pre>access-control-allow-headers: Authorization, Content-Type, Accept, Origin, User-Agent, DNT, Cache-Control, X-Mx-ReqToken, Keep-Alive, X-Requested-With, If-Modified-Since, x-referer access-control-allow-methods: GET, POST, PUT, DELETE, OPTIONS access-control-allow-origin: * content-length: 365 content-type: application/json date: Sat, 07 Jun 2025 08:55:34 GMT</pre></div>

Рисунок К.1 – Корректная регистрация пользователя

Curl

curl -X 'POST' \
'http://localhost:8080/auth/login' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
 "password": "12341234",
 "username": "ilya@yandex.ru"
}'

Request URL

http://localhost:8080/auth/login

Server response

Code	Details
200	<div>Response body</div> <div><pre>{ "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE5NjYyODY0YjZmZm9wIiwidXN1c19pZCI6NSwiZXN1cm5hbWUiOiJpbHlhQ1hbmRleC5ydSJ9.2P81eW7oKnbZwGvt6v_SsCWmW1r1W76uSlizl6Pk-E", "refresh_token": "eyJ1c2VyX2lkIjo1LlC1c2VyX25hbWUiOiJpbHlhQ1hbmRleC5ydSI6InVzZXJfcm9sZSI6Inob3AiFQ==.85d41308f91a2839fb1b543cae0519979cacc70a9483a01c83005581"}</pre></div> <div>Response headers</div> <div><pre>access-control-allow-headers: Authorization, Content-Type, Accept, Origin, User-Agent, DNT, Cache-Control, X-Mx-ReqToken, Keep-Alive, X-Requested-With, If-Modified-Since, x-referer access-control-allow-methods: GET, POST, PUT, DELETE, OPTIONS access-control-allow-origin: * content-length: 365 content-type: application/json date: Sat, 07 Jun 2025 08:58:07 GMT</pre></div>

Рисунок К.2 – Корректная авторизация пользователя

Curl

curl -X 'GET' \
'http://localhost:8080/cart' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE5NjYyODY0YjZmZm9wIiwidXN1c19pZCI6NSwiZXN1cm5hbWUiOiJpbHlhQ1hbmRleC5ydSJ9.2P81eW7oKnbZwGvt6v_SsCWmW1r1W76uSlizl6Pk-E'

Request URL

http://localhost:8080/cart

Server response

Code	Details
200	<div>Response body</div> <div><pre>{ "id": 2, "total": 0, "items": [] }</pre></div> <div>Response headers</div> <div><pre>access-control-allow-headers: Authorization, Content-Type, Accept, Origin, User-Agent, DNT, Cache-Control, X-Mx-ReqToken, Keep-Alive, X-Requested-With, If-Modified-Since, x-referer access-control-allow-methods: GET, POST, PUT, DELETE, OPTIONS access-control-allow-origin: * content-length: 30 content-type: application/json date: Sat, 07 Jun 2025 09:06:38 GMT</pre></div>

Рисунок К.3 – Корректное получение корзины пользователя

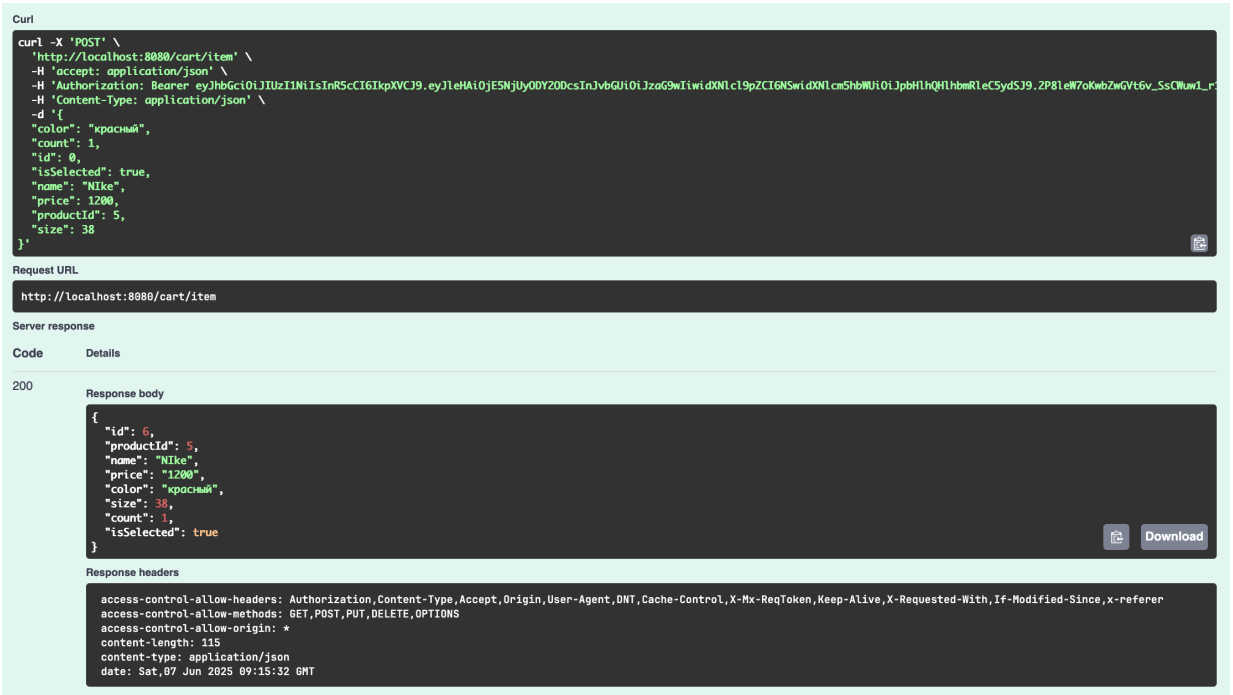


Рисунок К.4 – Корректное добавление продукции в корзину



Рисунок К.5 – Корректное удаление продукта из корзины

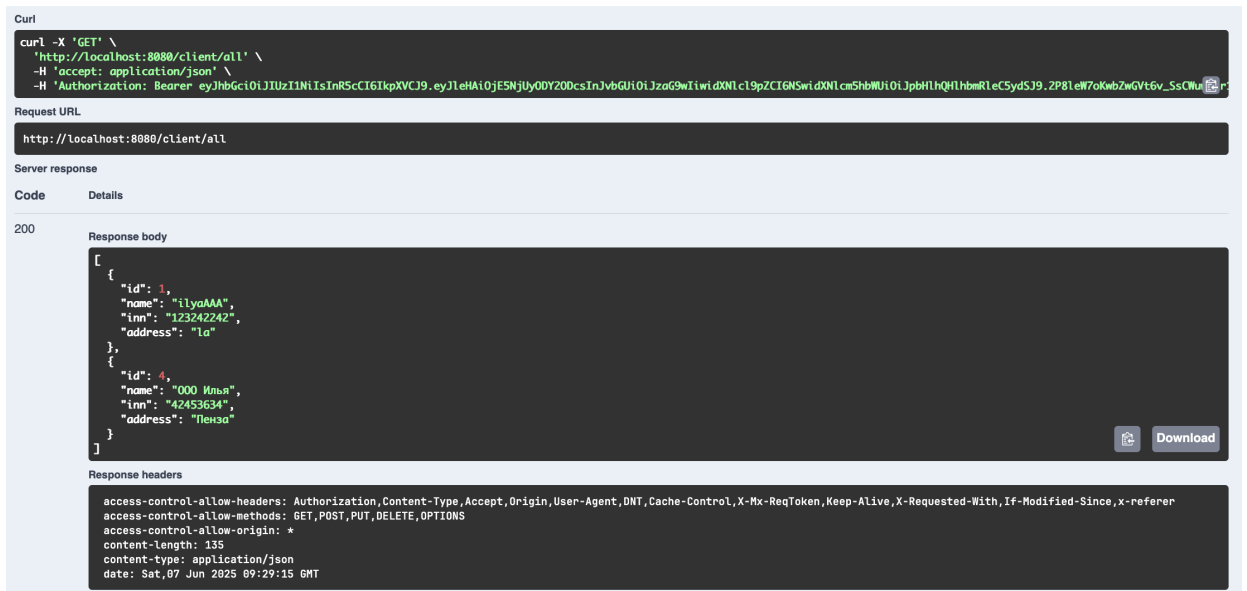


Рисунок К.6 – Корректное получение клиентов

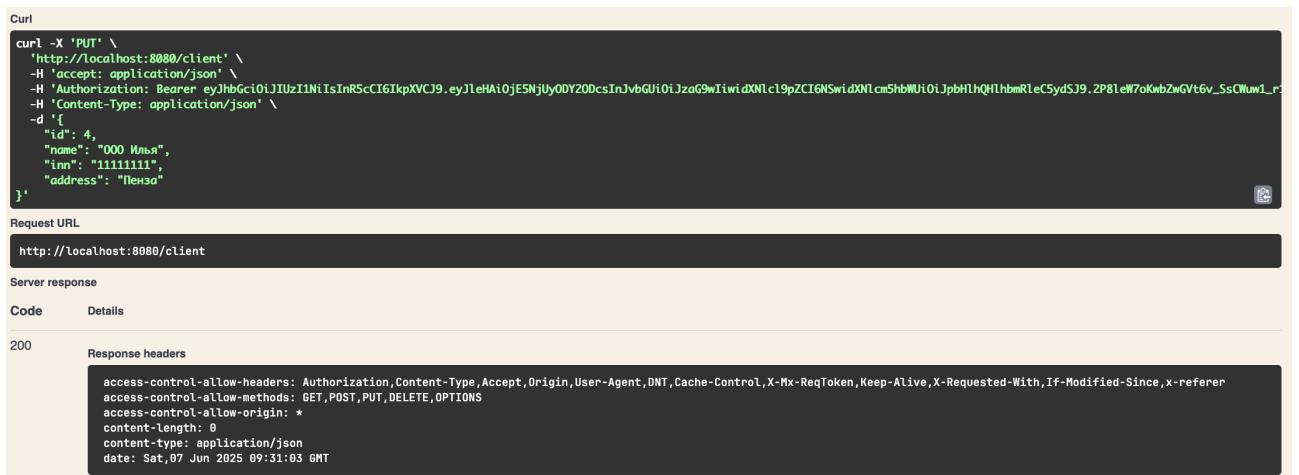


Рисунок К.7 – Корректное изменение клиента



Рисунок К.8 – Корректное удаление клиента



Рисунок К.9 – Корректное создание заказа



Рисунок К.10 – Корректное получение заказов пользователя

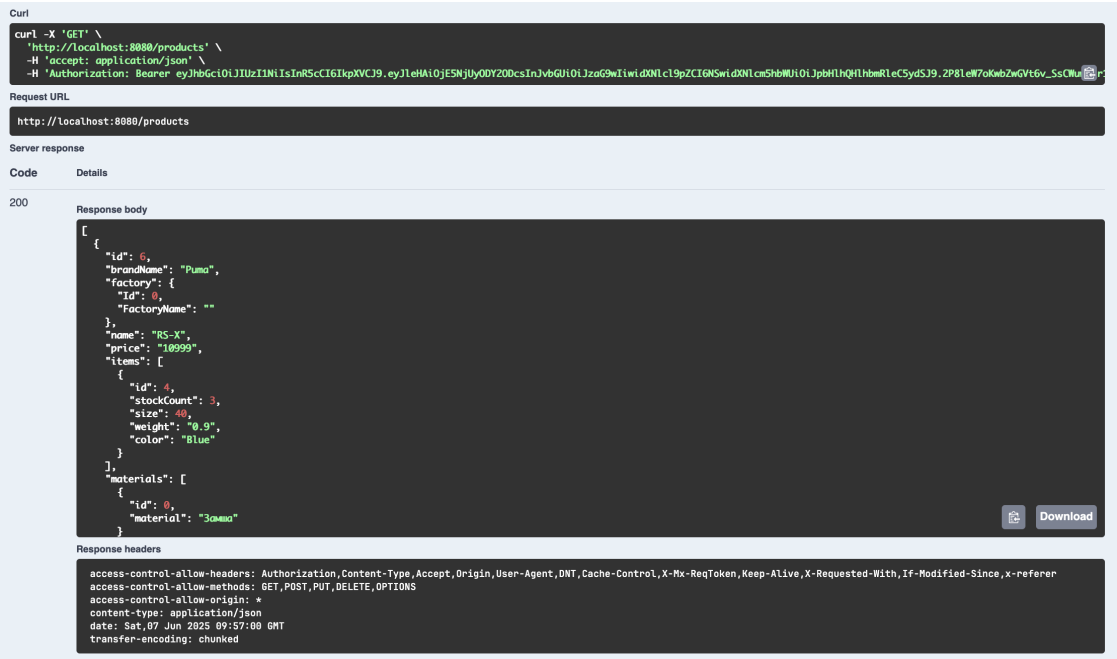


Рисунок К.11 – Корректное получение продукции со склада

Curl

```
curl -X 'POST' \
'http://localhost:8080/products' \
-H 'accept: application/json' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE5NjYyODY2ODcsInZvGUiOiJzaG9wIiwidXNlc19pZCI6NSwiZXNlcmShbWUiOiJpbHlhQ1hbmRleC5ydSJ9.2P8leW7oKmbZwGvt6v_SsCWm1_r' \
-H 'Content-Type: application/json' \
-d '[
{
  "brandName": "Nike",
  "factory": {
    "factoryName": "Nike Vietnam Factory",
    "id": 101
  },
  "id": 1001,
  "images": [
    {
      "imageUrl": "https://example.com/images/nike_shoe_1.jpg"
    },
    {
      "imageUrl": "https://example.com/images/nike_shoe_2.jpg"
    }
  ],
  "items": [
    {
      "color": "Black",
```

Request URL

```
http://localhost:8080/products
```

Server response

Code	Details
200	<div>Response headers</div> <div>access-control-allow-headers: Authorization,Content-Type,Accept,Origin,User-Agent,DNT,Cache-Control,X-Mx-ReqToken,Keep-Alive,X-Requested-With,If-Modified-Since,x-referer access-control-allow-methods: GET,POST,PUT,DELETE,OPTIONS access-control-allow-origin: * content-length: 0 content-type: application/json date: Sat,07 Jun 2025 10:01:18 GMT</div>

Рисунок К.12 – Корректное добавление продукции

Curl

```
},
{
  "color": "White",
  "id": 2002,
  "size": 41,
  "stockCount": 30,
  "weight": 0.5
}
],
"materials": [
  {
    "id": 1,
    "material": "Кожа"
  },
  {
    "id": 2,
    "material": "Текстиль"
  }
],
"name": "Air Max Running Shoes",
"price": 129.99,
"status": "approved"
}]'
```

Request URL

```
http://localhost:8080/products
```

Server response

Code	Details
200	<div>Response headers</div> <div>access-control-allow-headers: Authorization,Content-Type,Accept,Origin,User-Agent,DNT,Cache-Control,X-Mx-ReqToken,Keep-Alive,X-Requested-With,If-Modified-Since,x-referer access-control-allow-methods: GET,POST,PUT,DELETE,OPTIONS access-control-allow-origin: * content-length: 0 content-type: application/json date: Sat,07 Jun 2025 10:19:23 GMT</div>

Рисунок К.13 – Корректное изменение продукции

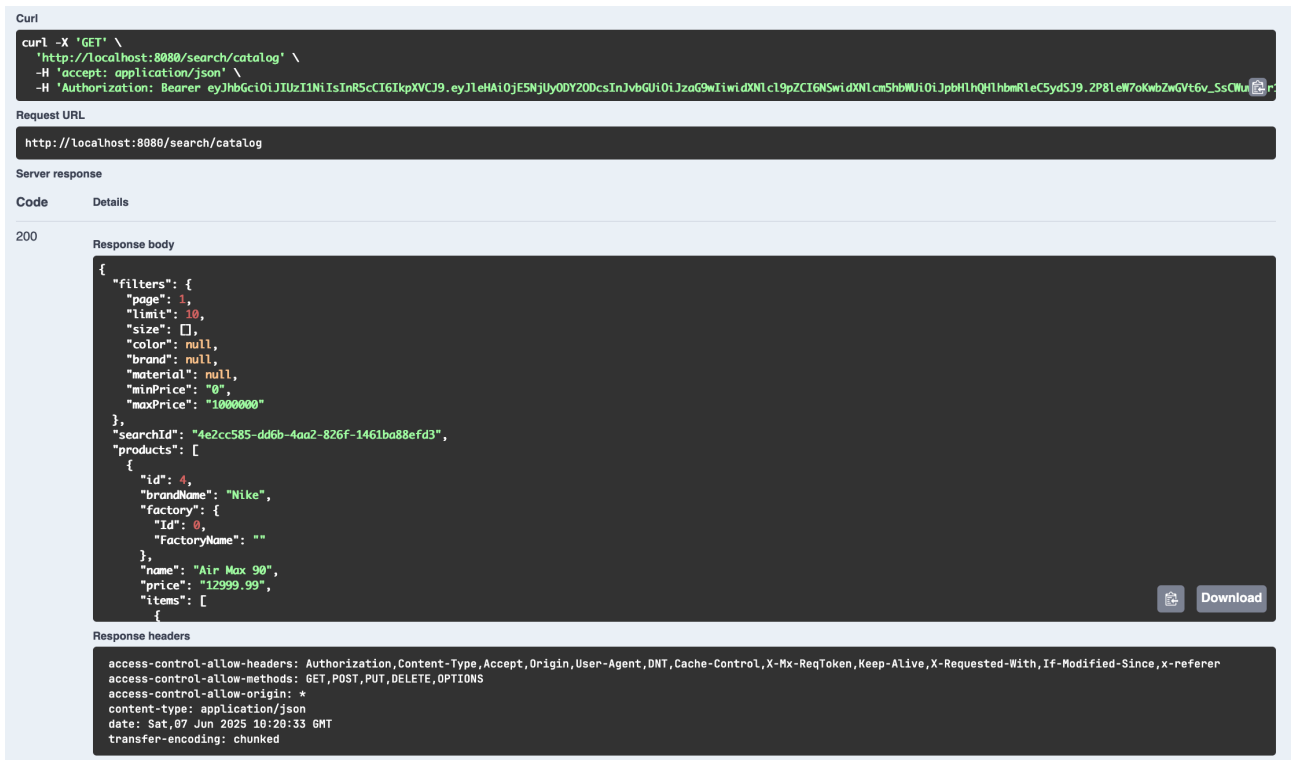


Рисунок К.14 – Корректное получение каталога продукции

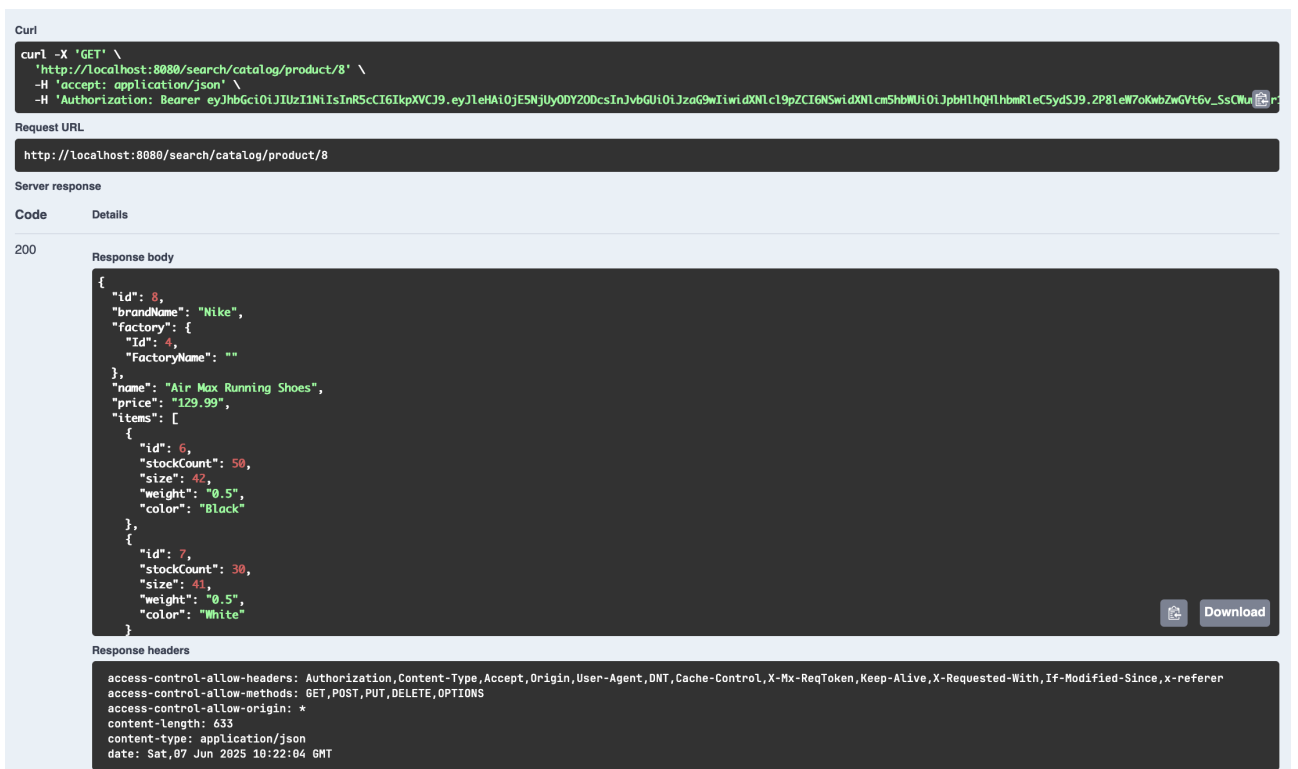


Рисунок К.15 – Корректное получение каталога по одной продукции

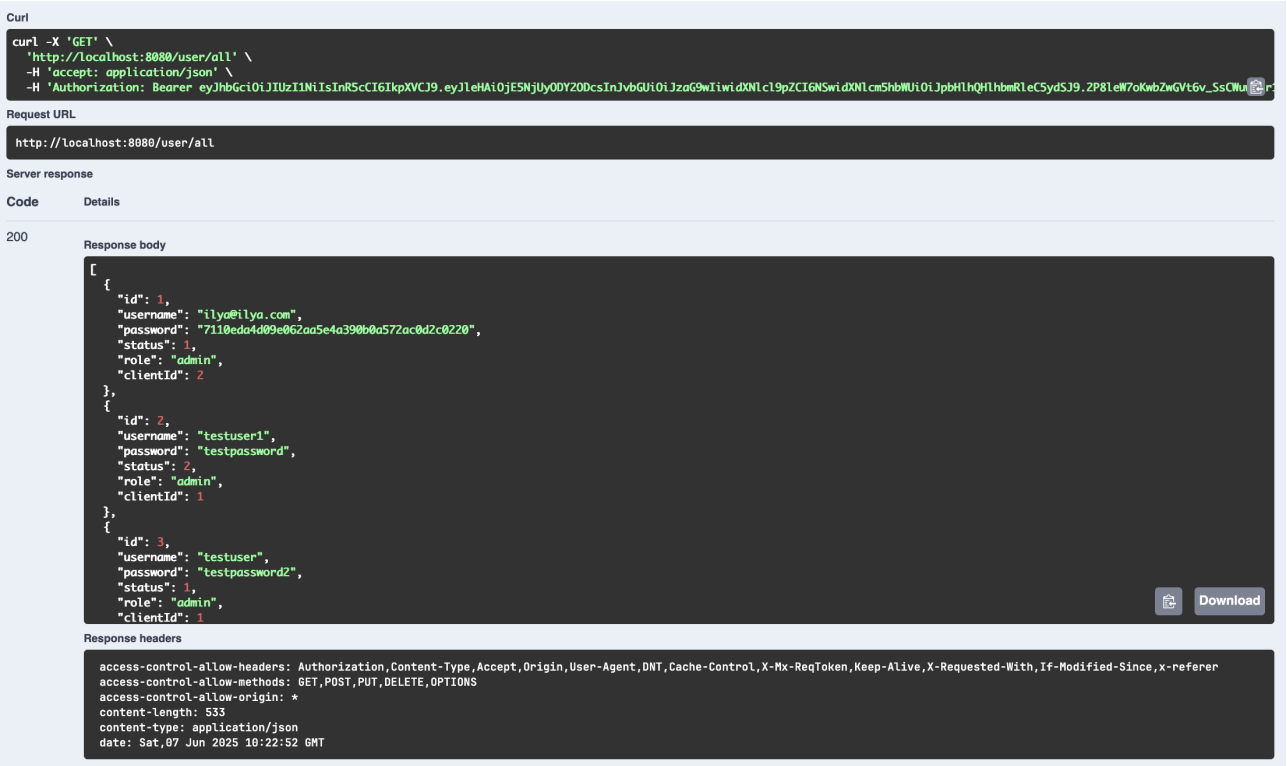


Рисунок К.16 – Корректное получение списка пользователей



Рисунок К.17 – Корректное изменение информации пользователя

Приложение Л
(рекомендуемое)
Графическая часть

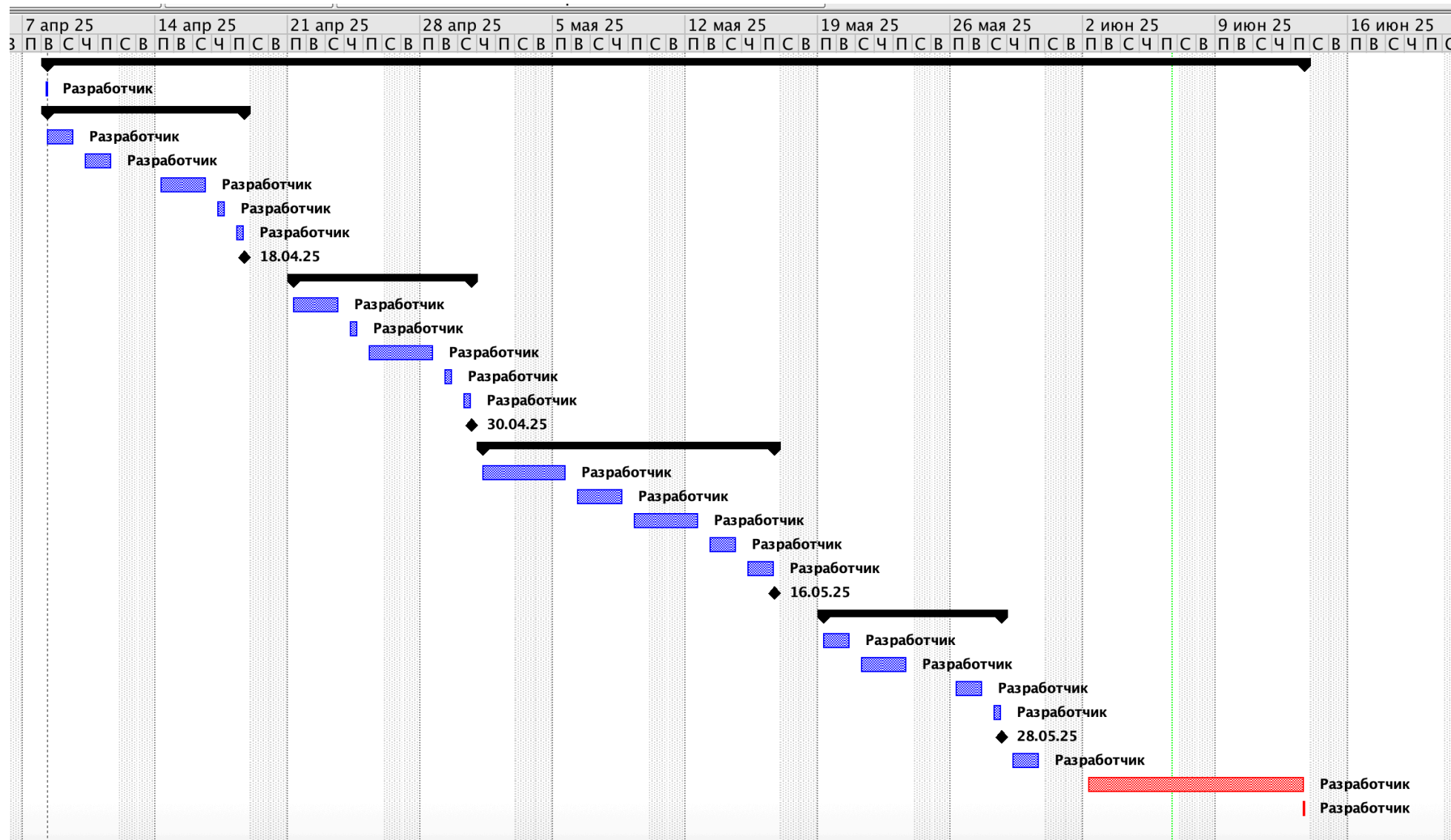


Рисунок Л.1 – Диаграмма Ганта

№	Название	Продолжительность	Начало	Окончание	Работа	Стоимость	Название ресу
	[-] Разработка серверного приложения	50 дней?	08.04.25 8:00	16.06.25 17:00	408 часов	196000 руб.	
	Начало проекта	1 день?	08.04.25 8:00	08.04.25 17:00	8 часов	0 руб.	
	[-] Анализ предметной области приложения	9 дней?	08.04.25 8:00	18.04.25 17:00	72 часов	36000 руб.	
	Изучение предметной области	2 дней?	08.04.25 8:00	09.04.25 17:00	16 часов	8000 руб.	Разработчик
	Изучение алгоритмов решения транспортной задачи	2 дней?	10.04.25 8:00	11.04.25 17:00	16 часов	8000 руб.	Разработчик
	Сравнение аналогов	3 дней?	14.04.25 8:00	16.04.25 17:00	24 часов	12000 руб.	Разработчик
	Составление требований к системе	1 день?	17.04.25 8:00	17.04.25 17:00	8 часов	4000 руб.	Разработчик
	Постановка задачи на разработку	1 день?	18.04.25 8:00	18.04.25 17:00	8 часов	4000 руб.	Разработчик
	Конец анализа предметной области	0 дней?	18.04.25 17:00	18.04.25 17:00	0 часов	0 руб.	
	[-] Проектирование приложения	8 дней?	21.04.25 8:00	30.04.25 17:00	64 часов	32000 руб.	
	Проектирование архитектуры	3 дней?	21.04.25 8:00	23.04.25 17:00	24 часов	12000 руб.	Разработчик
	Проектирование концептуальной модели данных	1 день?	24.04.25 8:00	24.04.25 17:00	8 часов	4000 руб.	Разработчик
	Проектирование логической модели данных	2 дней?	25.04.25 8:00	28.04.25 17:00	16 часов	8000 руб.	Разработчик
	Выбор технологий разработки	1 день?	29.04.25 8:00	29.04.25 17:00	8 часов	4000 руб.	Разработчик
	Составление требований к аппаратному обеспечению	1 день?	30.04.25 8:00	30.04.25 17:00	8 часов	4000 руб.	Разработчик
	Конец проектирования приложения	0 дней?	30.04.25 17:00	30.04.25 17:00	0 часов	0 руб.	
	[-] Разработка приложения	12 дней?	01.05.25 8:00	16.05.25 17:00	96 часов	48000 руб.	
	Разработка архитектуры	3 дней?	01.05.25 8:00	05.05.25 17:00	24 часов	12000 руб.	Разработчик
	Разработка физической модели данных	3 дней?	06.05.25 8:00	08.05.25 17:00	24 часов	12000 руб.	Разработчик
	Разработка слоев представления данных	2 дней?	09.05.25 8:00	12.05.25 17:00	16 часов	8000 руб.	Разработчик
	Разработка слоев бизнес-логики	2 дней?	13.05.25 8:00	14.05.25 17:00	16 часов	8000 руб.	Разработчик
	Разработка слоев обращения к базе данных	2 дней?	15.05.25 8:00	16.05.25 17:00	16 часов	8000 руб.	Разработчик
	Конец разработки приложения	0 дней?	16.05.25 17:00	16.05.25 17:00	0 часов	0 руб.	
	[-] Контроль качества приложения	8 дней?	19.05.25 8:00	28.05.25 17:00	64 часов	32000 руб.	
	Выбор метрик программного кода	2 дней?	19.05.25 8:00	20.05.25 17:00	16 часов	8000 руб.	Разработчик
	Функциональное тестирование	3 дней?	21.05.25 8:00	23.05.25 17:00	24 часов	12000 руб.	Разработчик
	Модульное тестирование	2 дней?	26.05.25 8:00	27.05.25 17:00	16 часов	8000 руб.	Разработчик
	Вычисление метрик программного кода	1 день?	28.05.25 8:00	28.05.25 17:00	8 часов	4000 руб.	Разработчик
	Конец контроля качества приложения	0 дней?	28.05.25 17:00	28.05.25 17:00	0 часов	0 руб.	
	Составление глоссария	2 дней?	29.05.25 8:00	30.05.25 17:00	16 часов	8000 руб.	Разработчик
	Составление отчета	10 дней?	02.06.25 8:00	13.06.25 17:00	80 часов	40000 руб.	Разработчик
	Конец проекта	1 день?	13.06.25 17:00	16.06.25 17:00	8 часов	0 руб.	

Рисунок Л.2 – Список задач проекта

№	Название	E-mail адрес	Стандартная ставка	Начислять	Основной календарь
	Разработчик	ilya.anton0v@vk.com	500 /час	Пропорционально	Пятидневка

Рисунок Л.3 – Список ресурсов проекта