МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Автоматизация процессов оптовых закупок и продаж путем создания специализированной площадки

Студент _____ Шаипов Камиль Гяряевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Афонин А. Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 03.06.2026г № 12)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

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

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2026

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«26» декабря 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Шаипов Камиль Гяряевич гр. 22ВП1 факультета ВТ

2.Тема работы Автоматизация процессов оптовых закупок и продаж путем создания специализированной площадки

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

3. Руководитель работы Афонин Александр Юрьевич

4. Задание на работу (назначение разработки, исходные данные и т.п.)
1) Автоматизировать процесс оптовых закупок путем создания специализированной площадки
2) Среда выполнения приложения: семейство ОС UNIX
Языковые средства: Golang, TypeScript
Инструментальные средства разработки: IDE JetBrains Goland
Технологии разработки: Docker, PostgreSQL, Chi, MinIO, React, Tailwind
3) Основные функции программного обеспечения:
а) автоматизация оформления заказов;
б) ведение участников площадки;
в) контроль сделок и разрешение споров;
г) документооборот;
д) аналитика и прогнозирование.

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области
2) Проектирование программных средств
3) Реализация программных средств
4) Тестирование программных средств
5) Оформление пояснительной записки

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%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

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 119 листов, 38 рисунков, 16 таблиц, 20 использованных источников, 5 приложений.

WEB-ПРИЛОЖЕНИЕ, ИНФОРМАЦИОННАЯ СИСТЕМА, GOLANG, REACT, DOCKER, POSTGRESQL, MINIO, CHI, TYPESCRIPT, JAVASCRIPT, B2B, ЮРИДИЧЕСКОЕ ЛИЦО, БАЙЕСОВСКАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ, UML

Объектом разработки является специализированная площадка для автоматизации процессов оптовых закупок.

Предметом разработки является подсистема электронной торговли, обеспечивающая взаимодействие покупателей, продавцов и модераторов платформы, управление каталогом товаров и категориями, оформление и сопровождение заказов, формирование счетов и закрывающих документов, обработку претензий, аналитику и прогнозирование спроса, а также администрирование пользователей и бизнес-процессов площадки.

Цель работы – разработка информационной системы автоматизации оптовых закупок посредством создания специализированной B2B-площадки.

Средства разработки: языки программирования Go и TypeScript, среда разработки IDE JetBrains Goland.

Среда исполнения – семейство ОС UNIX.

Результаты работы: проведён обзор и анализ существующих информационных систем и электронных площадок оптовых закупок, определены требования к разрабатываемой системе, выполнено проектирование микросервисной архитектуры и пользовательских сценариев для покупателей, продавцов и модераторов, разработаны и протестированы серверные сервисы и пользовательские интерфейсы.

Содержание

Введение 7
1 Анализ предметной области и постановка задачи на разработку приложения для оптовых закупок 8
1.1 Анализ предметной области приложения 8
1.2 Анализ моделей организации оптовых закупок 10
1.3 Анализ аналогичных приложений 11
1.4 Постановка задачи на разработку программного обеспечения 16
1.4.1 Цели и задачи приложения 16
1.4.2 Категории пользователей B2B-маркетплейса 17
1.5 Анализ требований 18
1.5.1 Анализ функциональных требований 18
1.5.2 Анализ нефункциональных требований 23
2 Проектирование программного обеспечения 25
2.1 Архитектура приложения 25
2.2 Проектирование структур данных 27
2.2.1 Концептуальная модель баз данных 27
2.2.2 Логическая модель баз данных 31
2.3 Обобщённая структура разрабатываемого сервиса 37
2.4 Рекомендуема конфигурация аппаратного обеспечения 38
3 Разработка приложения 40
3.1 Разработка архитектуры 40
3.2 Разработка физической модели базы данных 43
3.3 Разработка архитектурных слоев внутри микросервисов 50
3.3.1 Разработка слоя представления данных 50
3.3.2 Разработка слоя бизнес-логики в микросервисах 52
3.3.3 Разработка слоя обращения к базе данных 54
4 Контроль качества программного обеспечения 58
4.1 Выбор модели тестирования и метрик кода 58
4.2 Тестирование B2B-маркетплейса 59

4.2.1 Функциональное тестирование создания товара 59
4.2.2 Модульное тестирование создания заказа 61
4.2.3 Метрики Холстеда для метода создания заказа 62
5 Планирование разработки и оценка бюджета 64
Заключение 66
Список использованных источников 67
Приложение А (Справочное) Глоссарий 70
Приложение Б (обязательное) Исходный код 72
Приложение В (рекомендуемое) SQL-скрипты 94
Приложение Г (рекомендуемое) Объекты обращения к базам данных 105
Приложение Д (рекомендуемое) Исходный код модульного тестирования 112

Введение

В современных условиях цифровой трансформации экономики и электронной коммерции оптовые закупки и продажи всё чаще организуются с использованием специализированных электронных площадок, объединяющих юридических лиц в видео компаний-покупателей и компаний-продавцов в едином информационном пространстве [1]. B2B-маркетплейс – это программно-аппаратный комплекс, позволяющий юридическим лицам размещать коммерческие предложения, формировать заказы, получать счета на оплату и контролировать исполнение сделок без необходимости вести переговоры по каждой позиции вручную. Автоматизация таких процессов снижает издержки документооборота, повышает прозрачность взаимодействия сторон и ускоряет принятие управленческих решений.

Целью данной выпускной квалификационной работы является проектирование и разработка информационной системы – специализированной B2B-площадки для автоматизации процессов оптовых закупок и продаж. Программное средство должно обеспечивать закрытый доступ участников рынка, единый каталог товаров, оформление заказов по контуру RFQ (Request for Quotation – запрос коммерческого предложения), генерацию счетов в формате PDF, управление статусами исполнения заказов, механизм претензий и аналитическое прогнозирование цен и спроса по категориям товаров.

Объектом исследования являются бизнес-процессы оптовой торговли между организациями на электронной площадке. Предметом исследования – методы и средства автоматизации указанных процессов посредством микросервисной информационной системы с веб-клиентом. Для достижения поставленной цели необходимо проанализировать предметную область, сформулировать требования к серверной и клиентской частям, спроектировать архитектуру и структуры данных, а также реализовать программные средства.

1 Анализ предметной области и постановка задачи на разработку приложения для оптовых закупок

1.1 Анализ предметной области приложения

Предметная область выпускной квалификационной работы строится на процессе оптовых закупок и продаж между юридическими лицами на специализированной электронной площадке. Для обеспечения единства трактовки используемых категорий в работе рассмотрим основные термины и понятия, относящиеся к выбранной предметной области.

B2B – модель торговых отношений, в которой субъектами торговли являются юридические лица.

B2C – модель торговых отношений, в котором субъектом-покупателем является конечный потребитель (физическое лицо), а продавцом юридическое лицо.

B2G – модель торговых отношений, при котором поставщиком является организация или предприниматель (юридическое лицо), а заказчиком – государство, орган государственной власти или муниципальное образование.

SaaS – бизнес-модель, при которой компания предоставляет доступ к контенту по подписке.

Маркетплейс – программная платформа, позволяющая совершать онлайн и оффлайн сделки. Подобное программное обеспечение включает в себя автоматизацию сделок, документооборот и централизованное управление процессом торговли.

B2B-маркетплейс (business-to-business marketplace) – программная платформа, на которой компании-продавцы размещают товары и коммерческие предложения, а компании-покупатели формируют заказы на оптовые партии. В отличие от розничных маркетплейсов (B2C-маркетплейсы), участники представляют не физических лиц, а организации с реквизитами (ИНН, КПП, полное наименование).

Контур RFQ (Request for Quotation) – модель организации торговли, при которой покупатель выбирает товар из каталога, формирует заказ, получает счёт на оплату и осуществляет расчёт с продавцом вне системы. Площадка не выполняет функции платёжного агрегатора и не подтверждает факт доставки – она автоматизирует информационный обмен и документооборот между сторонами сделки, предоставляя централизованный документооборот и централизованное управление процессом торговли [2].

Коммерческое предложение – предложение продавца на конкретный товар с указанием цены, доступного количества и привязкой к позиции корзины покупателя. При оформлении заказа система формирует связку «товар – предложение – позиция заказа», фиксируя цену и количество на момент сделки.

Закрытая площадка – модель доступа, при которой регистрация новых участников осуществляется администратором (модератором) платформы, а не самостоятельно пользователями. Такой подход характерен для корпоративных закупочных систем и отраслевых B2B-порталов, где требуется верификация контрагентов.

Претензия – документ, инициируемый покупателем по конкретной позиции заказа при выявлении несоответствия товара, нарушения сроков или иных спорных ситуаций. Модератор платформы рассматривает претензию и формирует резолюцию, фиксируемую в информационной системе.

Исходя из этого, функциональность программных средств B2B-маркетплейса формируется под нужды различных типов пользователей:

– покупателей, использующих систему для оформления оптовыъ заказов;

– продавцов, управляющих ассортиментом товаров и исполнением заказов;

– модераторов, обеспечивающих доступ и разрешение споров – с целью автоматизировать, ускорить и стандартизировать процессы оптовых закупок и продаж.

1.2 Анализ моделей организации оптовых закупок

Ключевой задачей при проектировании специализированной площадки является выбор модели организации торговли, определяющей состав функций системы и границы автоматизации. На рынке B2B-решений выделяют несколько базовых подходов.

Каталожная модель (catalog-based) предполагает размещение продавцами фиксированных цен и остатков; покупатель выбирает товар и оформляет заказ без этапа торга. Данная модель обеспечивает высокую скорость оформления сделки и подходит для стандартизированных оптовых позиций с прозрачным ценообразованием.

Тендерная модель (RFQ/RFP) предполагает публикацию запроса покупателем и сбор конкурирующих предложений от нескольких продавцов. Она повышает конкуренцию, но увеличивает длительность цикла сделки и требует развитого механизма сравнения предложений. Данная модель наиболее характерна для B2G-сегмента, где конкурентные процедуры закупок регулируются законодательством, а также широко применяется в B2B при корпоративных и отраслевых закупках.

ERP-интеграционная модель связывает площадку с учётными системами участников (1C, SAP и др.) через EDI/API. Такой подход минимизирует ручной ввод данных, но требует существенных затрат на интеграцию и последующее сопровождение.

Для разрабатываемой системы выбран RFQ-контур с элементами каталожной модели:

– продавец заранее публикует товары с ценами и остатками, покупатель формирует корзину и оформляет заказ;

– счёт генерируется автоматически в PDF, оплата производится вне площадки.

Сравнение рассмотренных моделей по ключевым критериям приведено в таблице 1.

Таблица 1 – Сравнение моделей организации оптовых закупок

Критерий	Каталожная	Тендерная	ERP-интеграция	Выбранная модель
Скорость оформления заказа	Высокая	Низкая	Средняя	Высокая
Критерий	Каталожная	Тендерная	ERP-интеграция	Выбранная модель
Контроль контрагентов	Средний	Средний	Высокий	Высокий (закрытая площадка)
Прозрачность цены	Высокая	Средняя	Высокая	Высокая
Сложность внедрения	Низкая	Средняя	Высокая	Низкая
Автоматизация документов	Средняя	Средняя	Высокая	Высокая

По итогам сравнения можно сделать вывод, что комбинация каталожной модели с RFQ-контуром и закрытым доступом наилучшим образом соответствует цели работы – автоматизации оптовых закупок и продаж на специализированной площадке без избыточной сложности корпоративной интеграции на этапе MVP.

1.3 Анализ аналогичных приложений

На основании анализа рынка программных решений, для сопоставления отобраны платформы B2B-Center, ОптЛист и SAP Ariba. Сводные сведения о выбранных аналогах приведены в таблице 2.

Таблица 2 – Краткое описание аналогов

Наименование	Производитель	Описание
B2B-Center	ООО «Б2Б-Center»	Крупнейшая российская электронная торговая площадка для B2B-закупок: тендеры, аукционы, каталоги поставщиков, электронный документооборот.
ОптЛист	ИП Халяпин А. Н.	B2B оптовый портал: база поставщиков и производителей, каталог товаров, связь с контрагентами

Продолжение таблицы 2

Наименование	Производитель	Описание
SAP Ariba	SAP SE	Международная облачная платформа управления закупками: supplier management, контракты, аналитика цепочки поставок.

B2B-Center – флагман российского рынка электронных торгов для бизнеса. Платформа поддерживает широкий спектр процедур закупок, однако ориентирована преимущественно на тендерные процедуры, а не на прямое взаимодействие покупателя и продавца через единый товарный каталог. Подключение новых клиентов требует длительной процедуры аккредитации участников. [3] С точки зрения сравнительного анализа площадка имеет зрелую экосистему оформлений корпоративных сделок, но не предоставляет полноценную модель закрытого B2B-маркетплейса с оформление оптовых заказов через каталог. Интерфейс функционально насыщен, что повышает порог входа для небольших компаний. Внешний вид пользовательского интерфейса сервиса B2B-Center показан на рисунке 1.

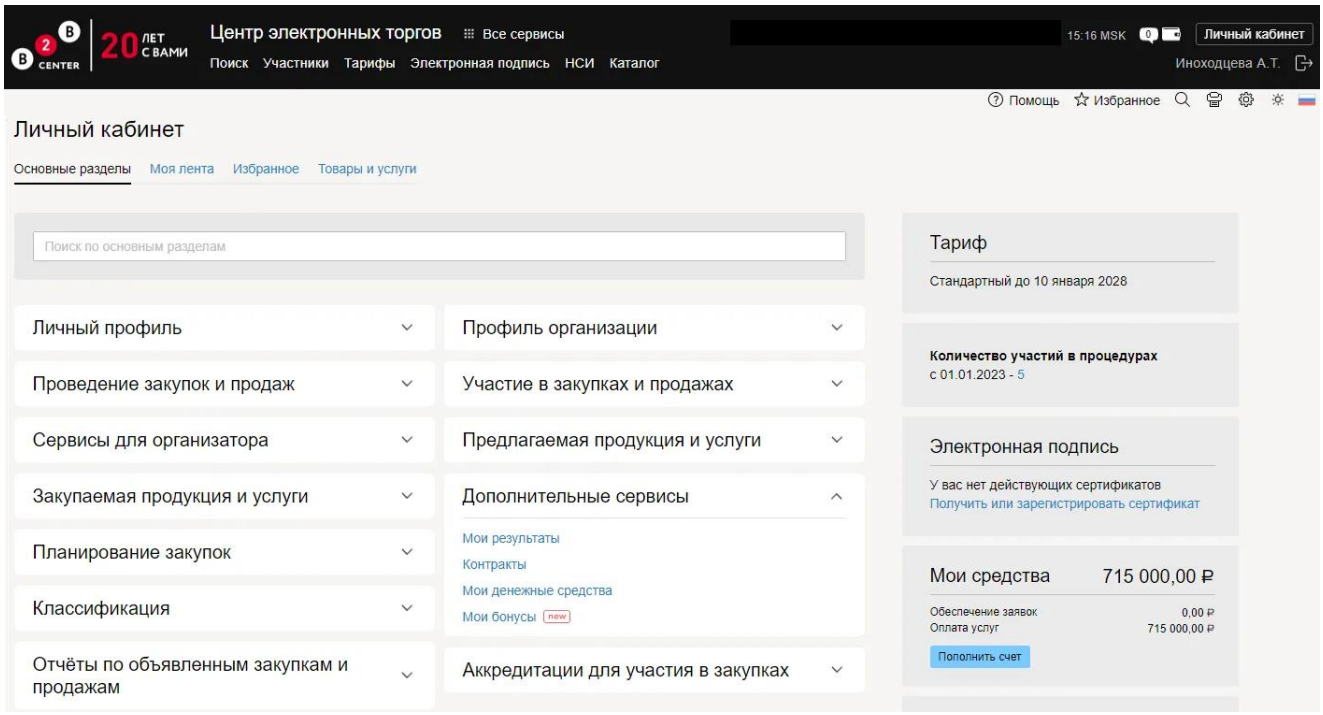

Рисунок 1 – Пользовательский интерфейс сервиса B2B-Center

Среди ключевых достоинств B2B-Center следует выделить:

– развитый функционал электронных торгов;

– соответствие требованиям законодательства о закупках;

– большую базу поставщика.

К недостаткам:

– высокую стоимость участия для малых предприятий;

– сложность настройки под задачи простого каталожного маркетплейса.

ОптЛист – российский B2B оптовый портал, работающий с 2011 года. Площадка ориентирована на средний и малый бизнес. Платформа объединяет базу поставщиков и производителей, каталог товаров и доску объявлений о закупках, предоставляя участникам возможность публикации коммерческих предложений и поиска партнеров по отраслевым категориям. Основной сценарий – поиск контрагента и согласование условий напрямую; сервис не обеспечивает полноценный закрытый контур с ролевым доступом, автоматической генерацией счетов и встроенным механизмом претензий. Графический интерфейс сервиса ОптЛист представлен на рисунке 2.

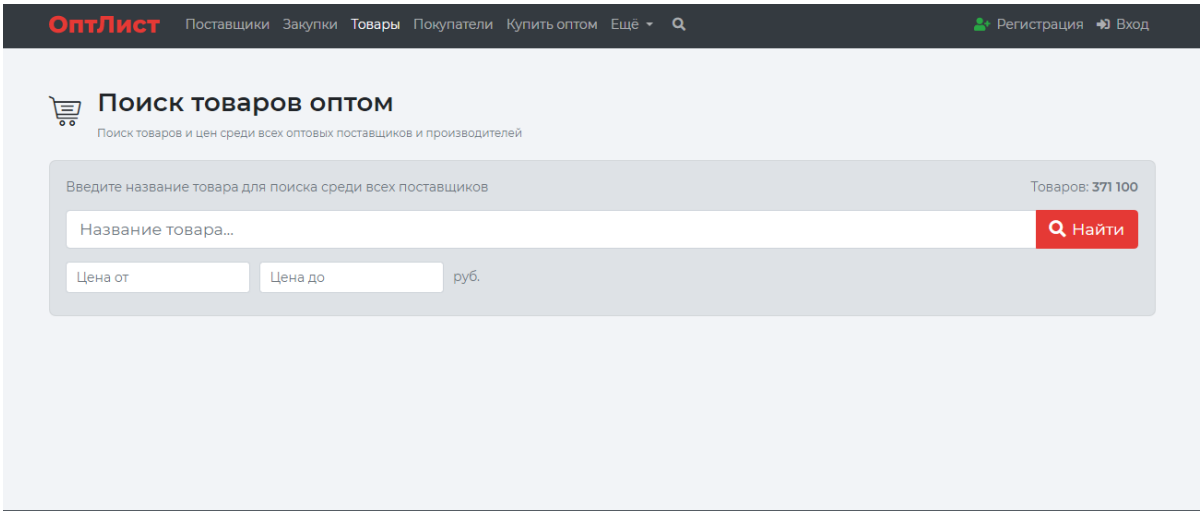

Рисунок 2 – Пользовательский интерфейс сервиса ОптЛист

SAP Ariba – enterprise-решение мирового уровня для управления закупками. Обеспечивает глубокую аналитику, интеграцию с ERP и управление глобальной базой поставщиков. Основные недостатки для российского среднего бизнеса – высокая стоимость лицензирования, сложность внедрения и

ограниченная адаптация под локальную специфику документооборота. Платформа реализует сквозной цикл закупочной деятельности: от формирования потребности и согласования заявок до проведения конкурентных процедур и контроля исполнения договорных обязательств. Для организаций, не использующих SAP в качестве корпоративной ERP-системы, экономическая целесообразность внедрения Ariba существенно снижается. Графический интерфейс сервиса представлен на рисунке 3.

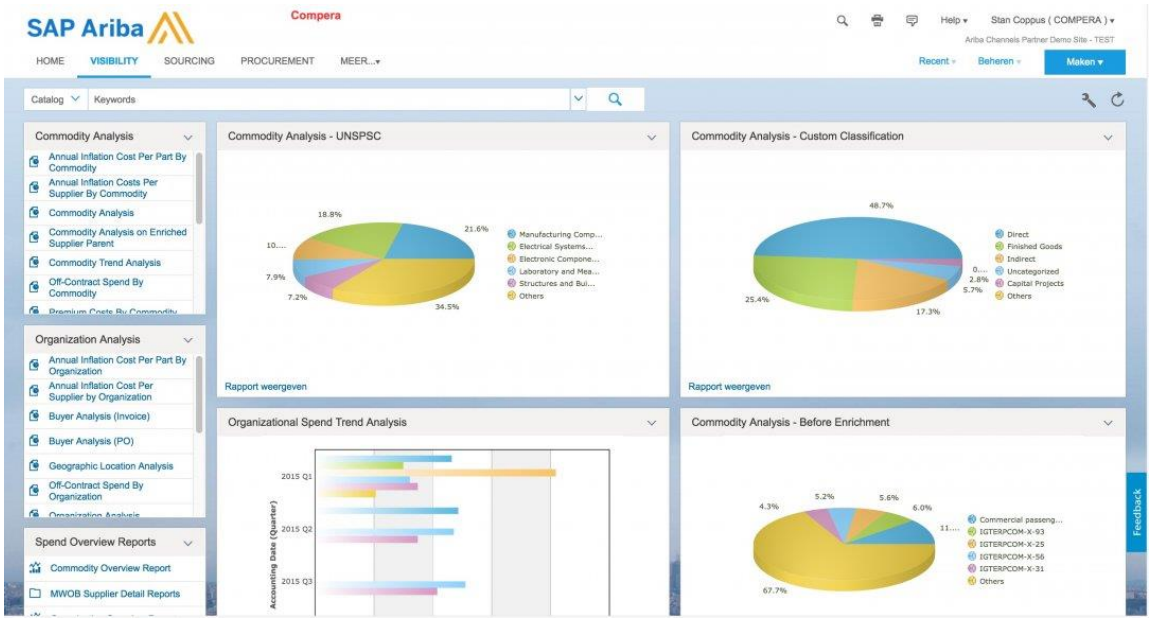

Рисунок 3 – Пользовательский интерфейс сервиса SAP Ariba

Для сопоставления аналогов с разрабатываемым приложением сформирован набор основных критериев оценки, отражающих ключевые функциональные требования к системе. Каждый из критериев оценивается по трёхбалльной шкале:

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

Суммарная оценка по всем критериям даёт обобщённое представление о возможностях рассматриваемых решений и их пригодности для реализации проектируемой B2B-платформы. Итоги сравнения, разрабатываемого приложения с аналогами представлены в таблице 3.

Таблица 3 – Результаты сравнения аналогичных программных средств

Критерий	B2B-Center	ОптЛист	SAP Ariba	Разрабатываемая система
Единый каталог товаров	2	2	2	2
Корзина и оформление заказа	1	1	2	2
Аналитика и прогнозирование	1	0	2	2
Закрытый доступ	1	0	2	2
Ориентация на российский рынок	2	2	0	2
Управление статусом заказа	1	1	2	2
Система претензий	1	0	2	2
Генерация счета	0	0	2	2
Итого	9	6	14	16

На основании проведённого сравнения можно заключить, что рассмотренные сервисы реализуют схожий, но не идентичный функционал и, наряду с достоинствами, имеют отдельные ограничения. Ни одна из рассмотренных платформ не сочетает простоту каталожного маркетплейса с закрытым контуром, автоматической генерацией счетов, претензионным механизмом и встроенной аналитикой для среднего бизнеса. Поэтому в рамках выпускной квалификационной работы необходимо разработать специализированную площадку, в которой будут реализованы следующие разделы:

- единый каталог товаров с фильтрацией по категориям и ценам;
- оформление заказов и автоматическую генерацию счетов PDF;
- отслеживание статусов исполнения заказов;
- механизм подачи и разрешения претензий;
- прогнозирование цен и спроса по категориям товаров;

– микросервисную архитектуру для масштабирования и сопровождения.

1.4 Постановка задачи на разработку программного обеспечения

1.4.1 Цели и задачи приложения

Целью, выполняемой выпускной квалификационной работы, является проектирование и разработка информационной системы в виде специализированной B2B-платформы, обеспечивающей автоматизацию процессов оптовых закупок и продаж. Основной функцией системы должно быть обеспечение полного цикла взаимодействия покупателя и продавца: от просмотра каталога до получения счёта и закрытия сделки, с участием модератора платформы в управлении доступом и разрешении споров.

При создании приложения необходимо обеспечить выполнение следующих ключевых задач:

– реализация входа пользователей в систему и администрирование учётных записей организаций;

– публикация товарных позиций и их поиск в общем каталоге;

– ведение корзины покупок и оформление заказов;

– формирование платёжных счетов в формате PDF;

– отслеживание и изменение статусов позиций заказа;

– подача и разрешение претензий покупателями и модератором;

– сбор аналитики и построение прогнозов цен и спроса;

– монетизация площадки (SaaS-подписка, комиссия со сделок, закрывающие документы);

– организация безопасной и отказоустойчивой инфраструктуры приложения.

По итогам выполнения перечисленных задач предполагается создание системы, архитектура которой включает микросервисные компоненты, веб-

приложение, единый шлюз доступа с проверкой прав пользователей, отдельные базы данных и хранилище объектов для графических материалов товаров.

1.4.2 Категории пользователей B2B-маркетплейса

Для разрабатываемого B2B-маркетплейса были выделены 3 категории пользователей: покупатель, продавец и модератор. Ролевая модель обеспечивает разграничение полномочий: каждый участник взаимодействует с системой только в рамках функций, соответствующих его должностным обязанностям в цепочке оптовых закупок. Такое распределение ролей позволяет исключить несанкционированный доступ к данным и обеспечить прозрачность информационных потоков на всех этапах взаимодействия. Сведения о выполняемых бизнес-процессах и используемых объектах по каждой категории пользователей отражены в таблице 4.

Таблица 4 – Сведения о категориях пользователей приложения

№	Наименование класса	Затрагиваемая часть сервиса	Бизнес-процессы	Используемые объекты
1	Покупатель	Клиентское приложение покупателя	Просмотр каталога, добавление товара в корзину, оформление заказа, скачивание счёта, подача претензии, просмотр прогнозов	Товары, корзина, заказы, счета, претензии
2	Продавец	Клиентское приложение продавца	Редактирование товаров, загрузка изображений, управление категориями, изменение статусов заказов, просмотр отчетности и закрывающих документов	Товары, предложения, категории, заказы, подписки

Продолжение таблицы 4

№	Наименование класса	Затрагиваемая часть сервиса	Бизнес-процессы	Используемые объекты
3	Модератор	Панель модератора	Создание пользователей, рассмотрение претензий, формирование резолюций, управление подписками и ЗД	Пользователи, претензии, резолюции, документы

Как следует из представленных данных, каждая роль оперирует собственным набором сущностей и выполняет непересекающиеся операции, что исключает дублирование функций между участниками платформы. Выделенная структура ролей послужит основой для формирования функциональных требований к клиентскому приложению и проектированию пользовательских интерфейсов соответствующих рабочих мест.

1.5 Анализ требований

1.5.1 Анализ функциональных требований

Диаграмма вариантов использования показывает перечень функций, доступных пользователям в рамках информационной системы, и то, как между собой связаны отдельные сценарии работы. На ней отображаются действующие лица – участники предметной области – и варианты использования, которыми они могут воспользоваться, а также отношения включения и расширения между ними. Каждый актор связан с теми сценариями, которые он инициирует или в которых участвует; граница системы отделяет внешнее окружение от функционала, реализуемого внутри приложения. Отношение включения указывает на обязательный фрагмент поведения, входящий в состав базового сценария, а отношение расширения описывает дополнительную, необязательную ветвь, которая может выполняться при определённых условиях.

Следует учитывать, что данный вид диаграмм описывает систему исключительно с внешней, пользовательской точки зрения: он фиксирует

доступный функционал, не раскрывая деталей внутренней реализации, структуры модулей или способа хранения данных. Такой подход позволяет на раннем этапе проекта сформировать единое представление о возможностях системы и проверить полноту охвата потребностей пользователей. Для детального описания отдельных сценариев диаграмма дополняется текстовыми спецификациями вариантов использования, в которых уточняются предусловия, основной и альтернативные потоки, а также ожидаемый результат. Поэтому диаграммы вариантов использования применяются на этапе формулирования требований и согласования ожиданий заинтересованных сторон, тогда как для проектирования внутренней структуры, взаимодействия компонентов и логики обработки данных используются другие средства моделирования [4]. Связь актора с вариантом использования показывает, какие цели пользователь может достичь при работе с системой, не уточняя при этом порядок шагов и технические детали выполнения операции.

При анализе функциональных требований построена диаграмма вариантов использования, описывающая ключевые сценарии функционирования сервиса:

- войти в систему;
- выйти из системы;
- перейти в каталог товаров;
- перейти в корзину;
- оформить заказ;
- скачать счёт на оплату;
- управлять товарами;
- управлять заказами;
- управлять пользователями;
- подать претензию;
- рассмотреть претензию;
- просмотреть прогнозы цен и спроса.

Диаграмма вариантов использования представлена на рисунке 4.

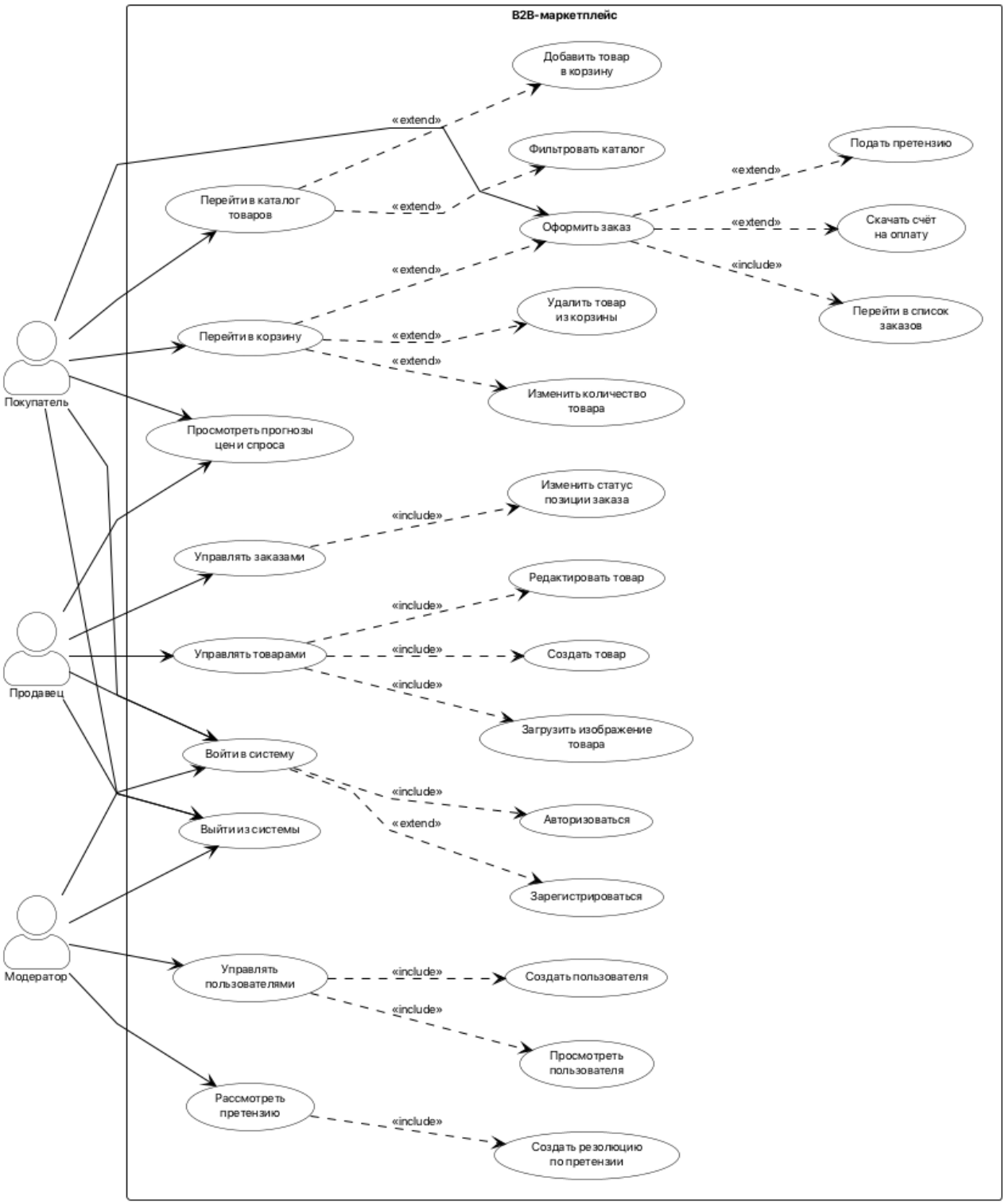

Рисунок 4 – Диаграмма вариантов использования B2B-маркетплейса

В каждом функциональном разделе сервиса основным актором выступает пользователь соответствующей категории. Далее приведены спецификации ключевых прецедентов B2B-маркетплейса.

Спецификация прецедента «Войти в систему» показана на рисунке 5.

Прецедент: Войти в систему
ID: 1
Краткое описание: Пользователь авторизуется в приложении B2B-маркетплейса.
Главные актеры: Пользователь любой группы (покупатель, продавец модератор).
Второстепенные актеры: Нет.
Предусловия: 1. Учетная запись пользователя создана модератором 2. Пользователь не авторизован в текущей сессии
Основной поток: 1. Пользователь переходит на страницу входа в приложение 2. Пользователь вводит логин и пароль существующей учетной запись 3. Система проверяет учетные данные пользователя и выдаем ему пару JWT-токенов
Постусловия: 1. Пользователь авторизован, ему доступны функции согласно роли.
Альтернативные потоки: 2а. Пользователь вводит неверные данные для авторизации – система отражает сообщение об ошибке и доступ к приложению не предоставляется

Рисунок 5 – Спецификация прецедента «Войти в систему»

Спецификация прецедента «Управлять товарами» показана на рисунке 6.

Прецедент: Управлять товарами
ID: 2
Краткое описание: Пользователь создает, редактирует и публикует товары в каталоге площадки
Главные актеры: Пользователь класса продавец
Второстепенные актеры: Нет
Предусловия: 1. Продавец авторизован в системе
Основной поток: 1. Продавец открывает раздел управления товарами 2. Продавец заполняет обязательные поля карточки товара 3. Продавец загружает изображение товара 4. Система проверяет и сохраняет объект в базу данных
Постусловия: 1. Товар создан или изменен и доступен покупателям в каталоге.
Альтернативные потоки: 4а. Пользователь редактирует существующий товар 4б. Ошибка проверка, система показывает ошибку пользователю, товар не сохранен

Рисунок 6 – Спецификация прецедента «Управлять товарами»

Спецификация прецедента «Оформить заказ» представлена на рисунке 7.

Прецедент: Оформить заказ
ID: 3
Краткое описание: Пользователь оформляет заказ товаров, находящихся в корзине
Главные актеры: Пользователь класса покупатель
Второстепенные актеры: Нет
Предусловия: 1. Покупатель авторизован в системе 2. В корзине есть неоформленные позиции
Основной поток: 1. Покупатель переходит в корзину 2. Покупатель нажимает кнопку оформить заказ 3. Система создает заказ в системе 4. Позиции корзины помечаются как оформленные 5. Покупатель получает номер созданного заказа
Постусловия: 1. Заказ успешно создан 2. Покупатель может отслеживать статус заказа 3. Покупатель может скачать счёт
Альтернативные потоки: 2а. Корзина пуста – кнопка оформления заказа недоступна

Рисунок 7 – Спецификация прецедента «Оформить заказ»

Спецификация прецедента «Подать претензию» представлена на рисунке 8.

Прецедент: Подать претензию
ID: 4
Краткое описание: Покупатель инициирует претензию по позиции ранее оформленного заказа
Главные актеры: Пользователь класса покупатель
Второстепенные актеры: Нет
Предусловия: 1. Покупатель авторизован в системе 2. Существует заказ с позицией по которой возник спор
Основной поток: 1. Покупатель открывает детали заказа 2. Покупатель выбирает позицию и нажимает кнопку «Подать претензию» 3. Покупатель заполняет описание проблемы 4. Система сохраняет претензию и передает ее модератору системы 5. Покупатель получает номер созданного заказа
Постусловия: 1. Претензия зарегистрирована, модератор может принять решение
Альтернативные потоки: 3а. Описание не заполнено – система запрашивает обязательные данные

Рисунок 8 – Спецификация прецедента «Подать претензию»

Таким образом, по результатам анализа функциональных требований сформирован минимальный перечень реализуемых функций, достаточный для корректного выполнения всех бизнес-сценариев B2B-маркетплейса.

1.5.2 Анализ нефункциональных требований

При анализе нефункциональных требований к информационной системе B2B-маркетплейса одной из приоритетных задач являлся выбор архитектурного подхода, способного удовлетворить предъявляемые ограничения. Были определены следующие требования:

- гибкость;
- масштабируемость;
- отказоустойчивость;
- производительность;
- безопасность.

Перечисленные требования обусловлены особенностями предметной области: система должна одновременно обслуживать запросы множества организаций, хранить графические материалы товаров, формировать PDF-документы и выполнять аналитические вычисления без ухудшения качества взаимодействия с пользователем.

С учётом сформулированных требований наиболее целесообразным решением признана микросервисная архитектура [5], обеспечивающая модульность и автономность отдельных компонентов системы.

Гибкость. Благодаря слабой связанности компонентов микросервисная архитектура позволяет модифицировать отдельные модули и расширять функциональность без перестройки уже работающих частей системы, что облегчает разработку и последующую поддержку системы.

Масштабируемость и производительность. Система должна сохранять работоспособность при увеличении числа участников и росте товарного каталога. В микросервисной архитектуре допускается наращивание

вычислительных ресурсов только для наиболее нагруженного сервиса, что снижает общие эксплуатационные затраты.

Отказоустойчивость. Каждый модуль системы работает независимо: отказ одного модуля не блокирует работу других модулей.

Безопасность – доступ к функциям разграничен по ролям (покупатель, продавец модератор), проверка авторизации выполняется через единую точку входа в серверную часть приложения, пароли хранятся в зашифрованном виде, межсервисное взаимодействие осуществляется во внутренней сети.

2 Проектирование программного обеспечения

2.1 Архитектура приложения

По итогам анализа функциональных и нефункциональных требований к разрабатываемому приложению в качестве базового решения принята микросервисная архитектура. Данный подход характеризуется приемлемой трудоёмкостью реализации и широкими возможностями при подборе технологий для отдельных предметных областей, позволяя создавать независимые модули приложения, тем самым повышая отказоустойчивость программного средства.

Для наглядного представления структуры системы и состава программных модулей построена диаграмма компонентов B2B-маркетплейса [6]. Диаграмма компонентов описывает физическое представление системы и позволяет определить границы ответственности каждого модуля. Диаграмма компонентов представлена на рисунке 9.

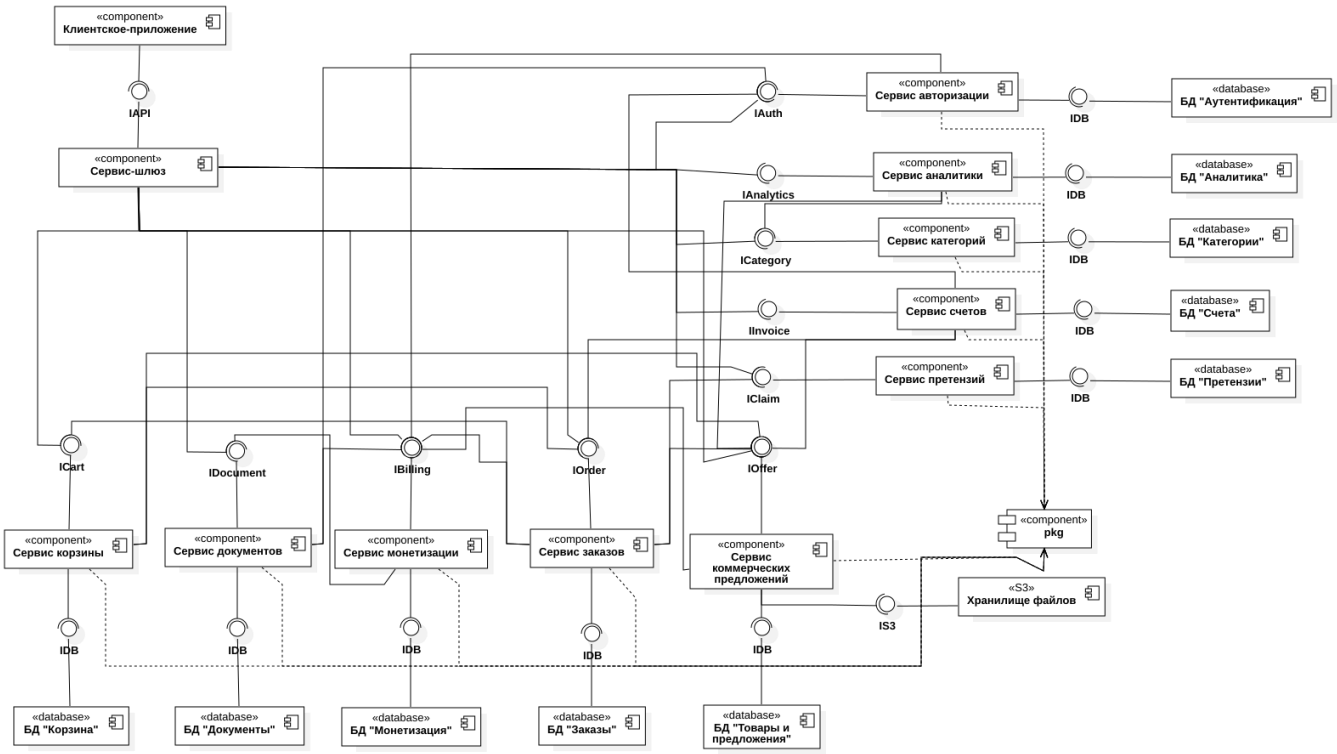

Рисунок 9 – Диаграмма компонентов

Представленные на диаграмме компоненты разделены по зонам ответсвтенности:

– сервис-шлюз – единая точка входа, JWT-авторизация (JSON Web Tokens), CORS, маршрутизация запросов к внутренним сервисам;

– сервис авторизации – вход пользователей в систему, выдача токенов и ведение учётных записей организаций;

– сервис коммерческих предложений – товары, коммерческие предложения, хранение изображение в MinIO;

– сервис корзины – корзина покупателя и оформление заказа;

– сервис заказов – создание заказов, статусы позиций, закрытие сделок;

– сервис счетов – генерация счетов на оплату в PDF;

– сервис претензий – претензии и резолюции модератора;

– сервис категорий – справочник категорий товаров;

– сервис аналитики – сбор метрик и прогнозирование цен и спроса;

– сервис монетизации – тарифы, подписки SaaS, комиссии со сделок;

– сервис документов – закрывающие документы (PDF, email, личный кабинет).

Такое разделение по предметным областям позволяет гибко модифицировать и масштабировать разрабатываемый сервис. Каждый микросервис использует собственную базу данных PostgreSQL, что исключает прямые SQL-запросы между сервисами и обеспечивает слабую связанность компонентов [7].

Обмен данными между микросервисами организован по протоколу HTTP в формате REST: для вызова удалённых операций применяются клиенты, вынесенные в общую библиотеку. Все серверные модули доступны друг другу во внутренней сети развёртывания.

Для однозначного описания действий над ресурсами используется стандартный набор HTTP-методов:

– GET – чтение и запрос существующих данных;

– POST – добавление новых сущностей;

– PUT – изменение ранее созданных сущностей;

– DELETE – удаление объектов из системы.

Асинхронные операции (генерация закрывающих документов, начисление комиссий) реализуются через паттерн Outbox и фоновые процессы.

Клиентская часть – одностраничное приложение (Single Page Application) на React 19 на языке программирования TypeScript, собранное через Vite и доступное через nginx. Все запросы направляются на сервис-шлюз, который добавляет заголовки X-User-ID и X-User-Type после успешной JWT-верификации.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель баз данных

Концептуальная модель данных фиксирует перечень сущностей предметной области на уровне их смыслового назначения, без привязки к конкретной СУБД, формату таблиц и способу физического размещения записей [8]. В рамках проектируемого B2B-маркетплейса такая модель используется для фиксации тех сущностей, которые обеспечивают выполнение основных операций платформы: идентификацию участников торгов, ведение товарного каталога, формирование заказов, расчётно-документальное сопровождение сделок, рассмотрение претензий и накопление аналитических показателей [9].

В соответствии с установленными требованиями к организации информационного обеспечения B2B-маркетплейса была сформирована ER-диаграмма концептуального уровня. На диаграмме отражены ключевые объекты предметной области и зависимости между ними. Схема используется при переходе к логическому и физическому проектированию баз данных микросервисов и для проверки того, что в модели отражены все сущности, необходимые для работы B2B-маркетплейса. Концептуальная ER-диаграмма приведена на рисунке 10.

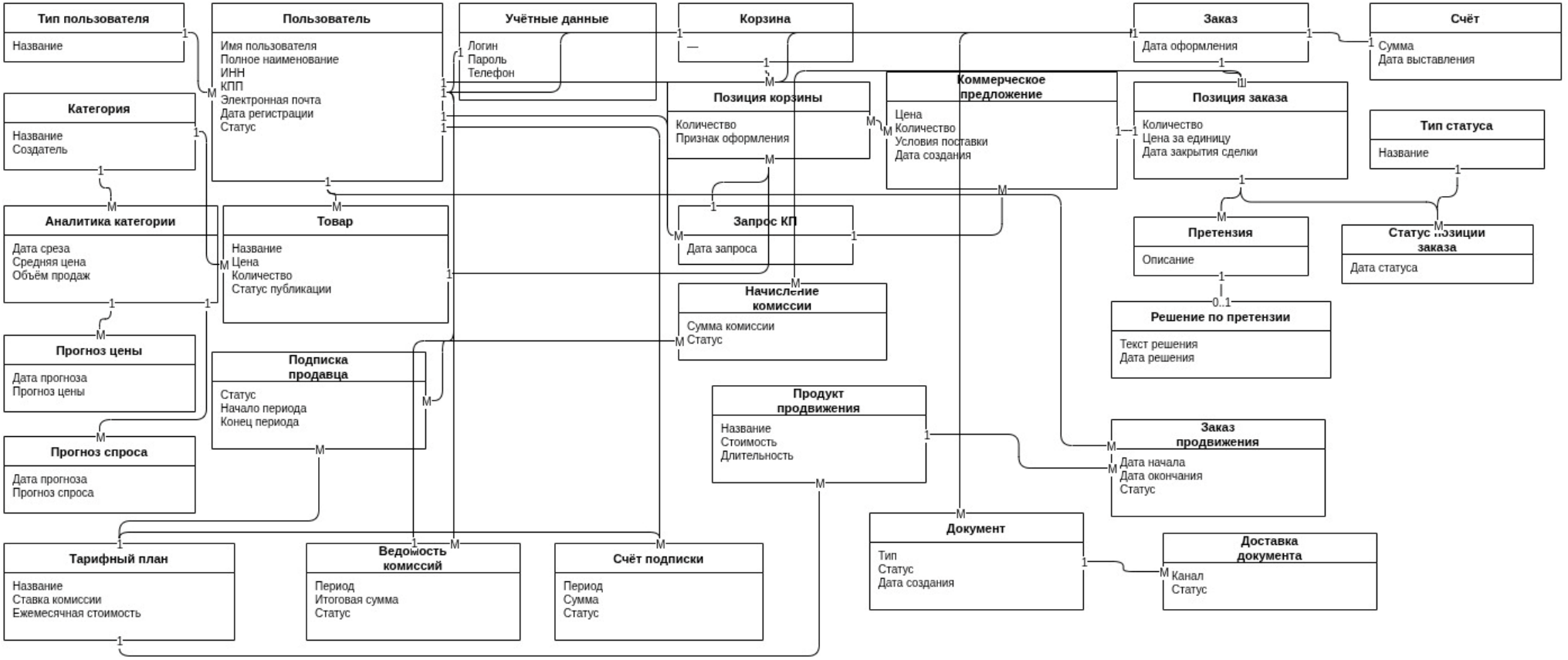

Рисунок 10 – Концептуальная модель базы данных

На диаграмме концептуальной модели показаны все информационные объекты системы и отношения, связывающие их между собой. Подробные сведения о сущностях, задействованных в бизнес-процессах B2B-маркетплейса, приведены в таблице 5.

Таблица 5 – Характеристика сущностей концептуальной модели

Название сущности	Идентификатор	Назначение
Тип пользователя	user_type	Хранит сведения о ролях пользователей: покупатель, продавец, модератор
Пользователь	user	Хранит сведения о пользователях системы
Учётные данные	credential	Хранит сведения об учетных данных пользователей системы
Категория	category	Хранит сведения категориях товаров
Аналитика категории	category_analytics	Хранит сведения об аналитическом срезе категории
Прогноз спроса	demand_forecast	Хранит сведения о прогнозе объема спроса по категории на дату
Прогноз цены	price_forecast	Хранит сведения о прогнозе средней цены по категории на дату
Товар	product	Хранит сведения о карточке товара продавца
Корзина	cart	Хранит сведения о корзине покупателя
Позиция корзины	cart_item	Хранит сведения о позиции корзины
Запрос коммерческого предложения	quote_request	Хранит сведения о запросе коммерческого предложения покупателя
Коммерческое предложение	offer	Хранит сведения о коммерческом предложении
Заказ	order	Хранит сведения об оформленном заказе
Позиция заказа	order_item	Хранит сведения о позиции оформленного заказа
Статус позиции заказа	order_item_status	Хранит сведения об истории смены статусов позиции заказа
Тип статуса позиции заказа	order_item_status	Хранит сведения о возможных статусах заказа

Продолжение таблицы 5

Название сущности	Идентификатор	Назначение
Счёт	invoice	Хранит сведения о счёте по заказу
Претензия	claim	Хранит сведения об обращении покупателя по позиции заказа
Решение по претензии	claim_resolution	Хранит сведения о решении модератора по претензии

В таблице 6 содержится описание ключевых связей между сущностями концептуальной модели.

Таблица 6 – Характеристика связей концептуальной модели

Название связи	Описание	Мощность
user_type – user	Принадлежность типа пользователя к пользователю	1:M
user – credential	Учётные данные организации к пользователю	1:1
user – cart	Принадлежность корзины к организации-покупателю	1:1
user – order	Принадлежность заказа к пользователю	1:M
user – product	Принадлежность товара к организации-продавцу	1:M
user – quote_request	Принадлежность запроса коммерческого предложения к пользователю	1:M
category – product	Принадлежность категории товара к товару	1:M
category – category_analytics	Принадлежность аналитики категории к категории	1:M
category – demand_forecast	Принадлежность прогноза спроса к категории	1:M
category – price_forecast	Принадлежность прогноза средней цены категории к категории	1:M
cart – cart_item	Принадлежность позиции корзины к корзине	1:M
product – cart_item	Принадлежность позиции корзины к товару	1:M
quoute_request – cart_item	Связь запроса коммерческого предложения с позициями корзины	M:M
cart_item – offer	Принадлежность коммерческого предложения к позиции корзины	1:M

Продолжение таблицы 6

Название связи	Описание	Мощность
quote_request – offer	Принадлежность запроса коммерческого предложения к коммерческим предложениям	1:M
order – order_item	Принадлежность позиции заказа к заказу	1:M
order – invoice	Принадлежность заказа к счету по заказу	1:1
status_type – order-item_status	Принадлежность статуса позиции заказа к возможным статусам	1:M
order_item – order_item_status	Принадлежность статуса позиции заказа к заказу	1:M
order_item – claim	Принадлежность претензии по позиции к позиции заказа	1:M
claim – claim_resolution	Принадлежность резолюции по претензии к претензии	1:0..1

Концептуальная модель разработана с учётом принципа «база данных на сервис»: логически связанные сущности распределены по изолированным базам данных соответствующих модулей приложения.

2.2.2 Логическая модель баз данных

На логическом уровне концептуальная модель детализируется за счёт добавления атрибутов, первичных и внешних ключей [10], а также правил связывания сущностей, отражающих бизнес-логику системы [11]. Полученное описание используется при проектировании физических таблиц базы данных [12].

Для серверной части разрабатываемого приложения спроектировано десять изолированных баз данных, каждая из которых соответствует отдельной предметной области: «Аутентификация», «Корзина», «Заказы», «Товары и предложения», «Счета», «Претензии», «Категории», «Аналитика», «Монетизация», «Документы».

Такое разделение обусловлено следующими причинами:

– автономное ведение данных в рамках каждой функциональной зоны;
– снижение нагрузки на единую БД за счёт распределения;

– возможность масштабирования и резервного копирования раздельно;

– соответствие принципам микросервисной архитектуры.

На рисунке 11 приведена логическая модель базы данных «Аутентификация».

Рисунок 11 – Логическая модель базы данных «Аутентификация»

На рисунке 12 приведена логическая модель базы данных «Товары и предложения»

Рисунок 12 – Логическая модель базы данных «Товары и предложения»

На рисунке 13 приведена логическая модель базы данных «Заказы».

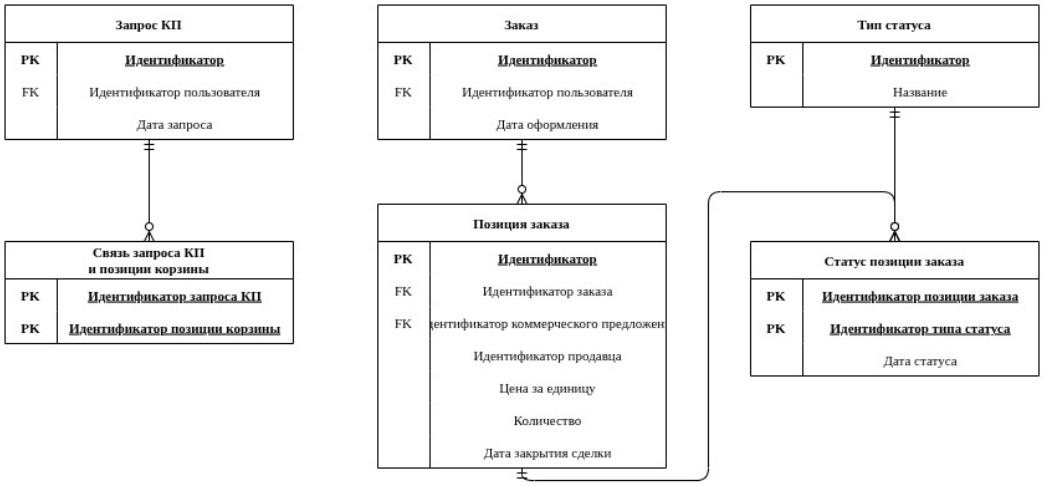

Рисунок 13 – Логическая модель базы данных «Заказы»

На рисунке 14 приведена логическая модель базы данных «Претензии».

Рисунок 14 – Логическая модель базы данных «Претензии»

На рисунке 15 приведена логическая модель базы данных «Аналитика»

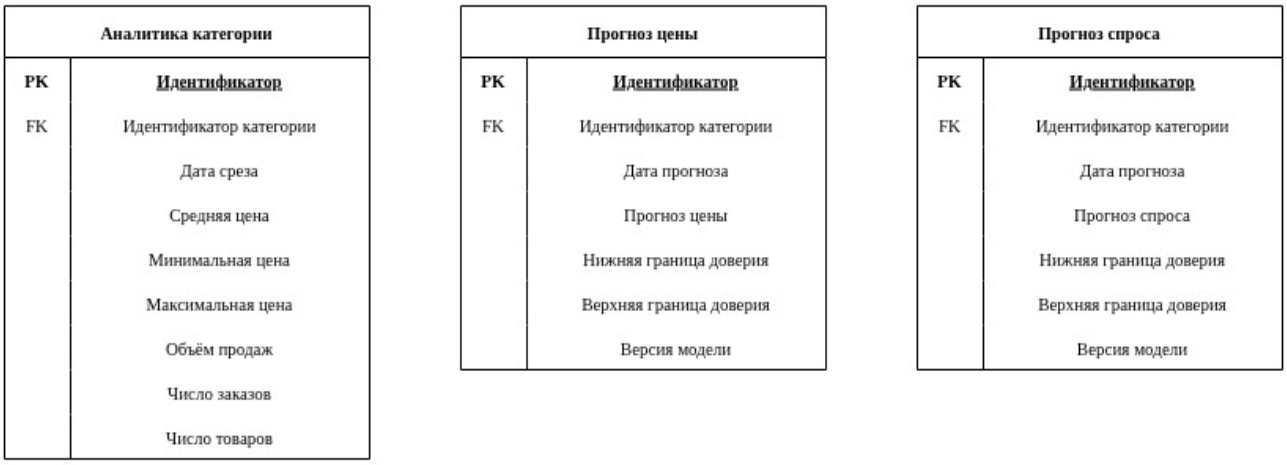

Рисунок 15 – Логическая модель базы данных «Аналитика»

На рисунке 16 приведена логическая модель базы данных «Монетизация».

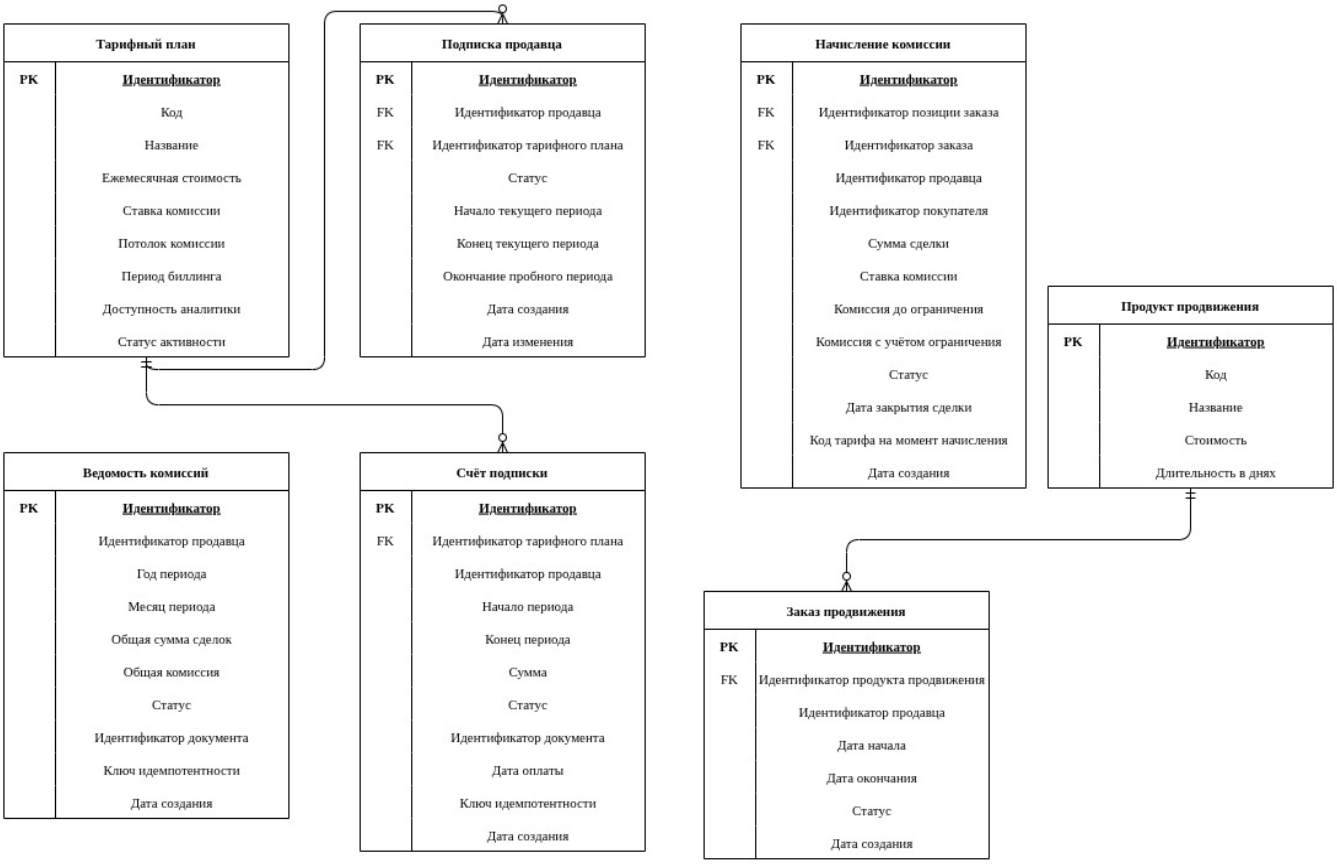

Рисунок 16 – Логическая модель базы данных «Монетизация»

На рисунке 17 приведена логическая модель базы данных «Корзина».

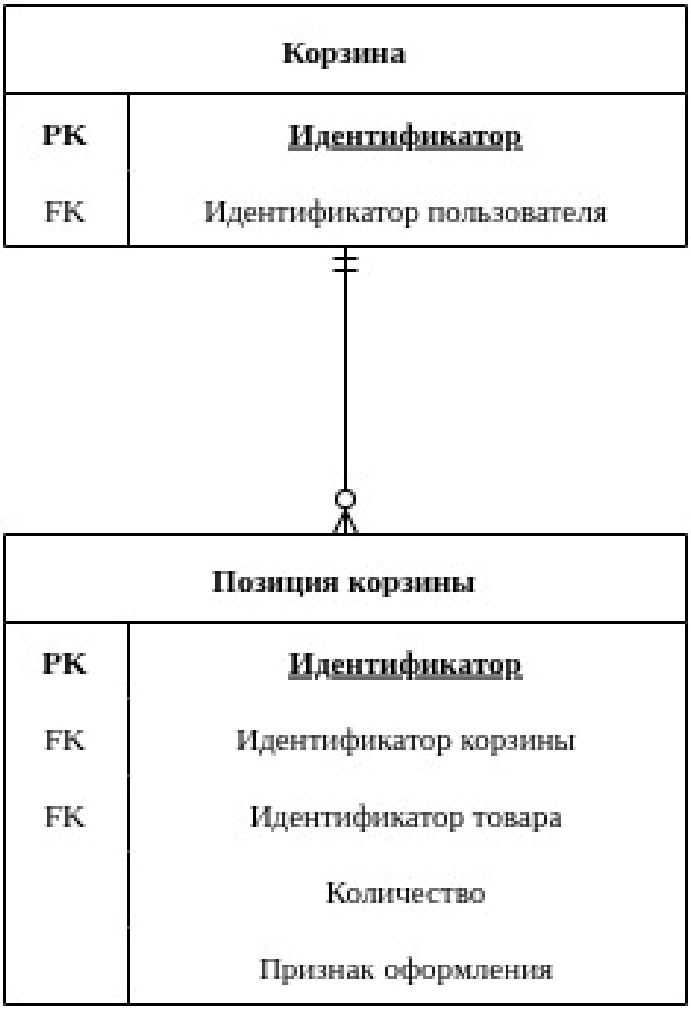

Рисунок 17 – Логическая модель базы данных «Корзина»

На рисунке 18 приведена логическая модель базы данных «Категории».

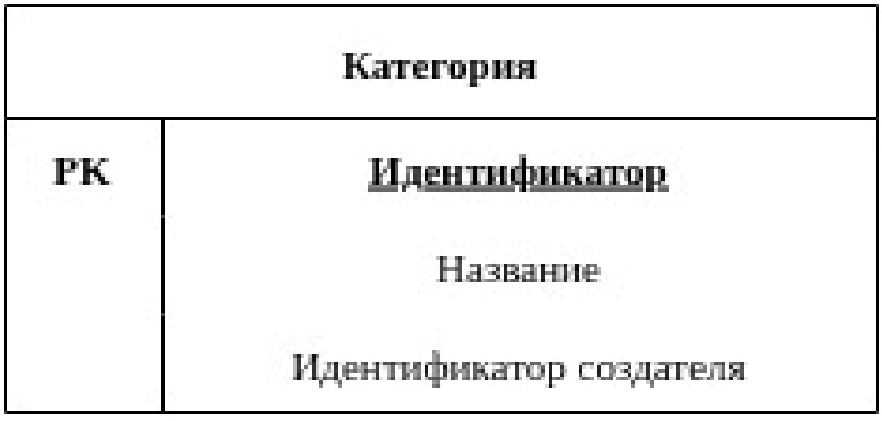

Рисунок 18 – Логическая модель базы данных «Категории»

На рисунке 19 приведена логическая модель базы данных «Документы».

Рисунок 19 – Логическая модель базы данных «Документы»

На рисунке 20 приведена логическая модель базы данных «Счета».

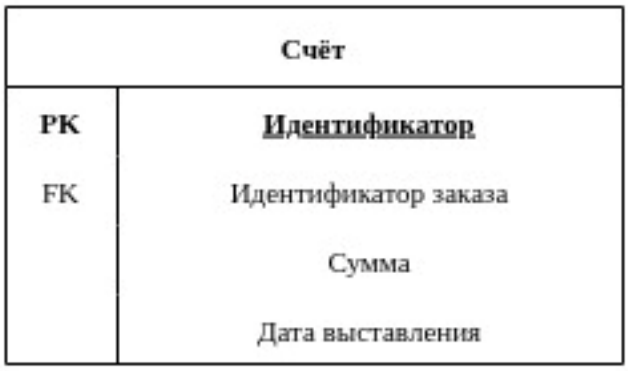

Рисунок 20 – Логическая модель БД «Счета»

Разработанные логические модели баз данных послужат основой для последующего проектирования физической структуры хранения данных и подготовки SQL-скриптов миграции, размещаемых в каталоге migrations/per-service/ (приложение Б).

2.3 Обобщённая структура разрабатываемого сервиса

B2B-маркетплейс функционирует по принципу клиент-серверного взаимодействия: пользовательский интерфейс работает на стороне клиента, бизнес-логика – на сервере [13]. Клиентская часть представляет из себя одностраничное веб-приложение, функционирующее в браузере пользователя. Серверная часть – набор микросервисов, обеспечивающих обработку бизнес-логики, взаимодействие с базами данных и обмен данными с клиентским приложением. Между клиентской и серверной частями реализован механизм аутентификации на основе JSON Web Token (JWT), обеспечивающий безопасную передачу сведений о пользователе и его роли при обращении к микросервисам. Серверные компоненты информационной системы развёртываются в контейнеризованной среде, что обеспечивает изоляцию микросервисов, единообразие конфигурации исполнения и упрощает сопровождение системы при изменении состава программных модулей.

Ключевые преимущества такой организации:

– низкие требования к рабочему месту пользователя – достаточно современного веб-браузера;

– централизация бизнес-логики и данных на сервере, без их дублирования на стороне клиента;

– разграничение доступа с помощью JWT-токенов и ролевой модели;

– упрощение развёртывания за счёт контейнеризации компонентов системы.

Основные недостатки [14]:

– при отказе серверного узла или СУБД система становится недоступной для всех пользователей;

– увеличение нагрузки требует горизонтального масштабирования отдельных микросервисов.

Общая архитектура развёртывания приложения показана на диаграмме развёртывания (рисунок 21).

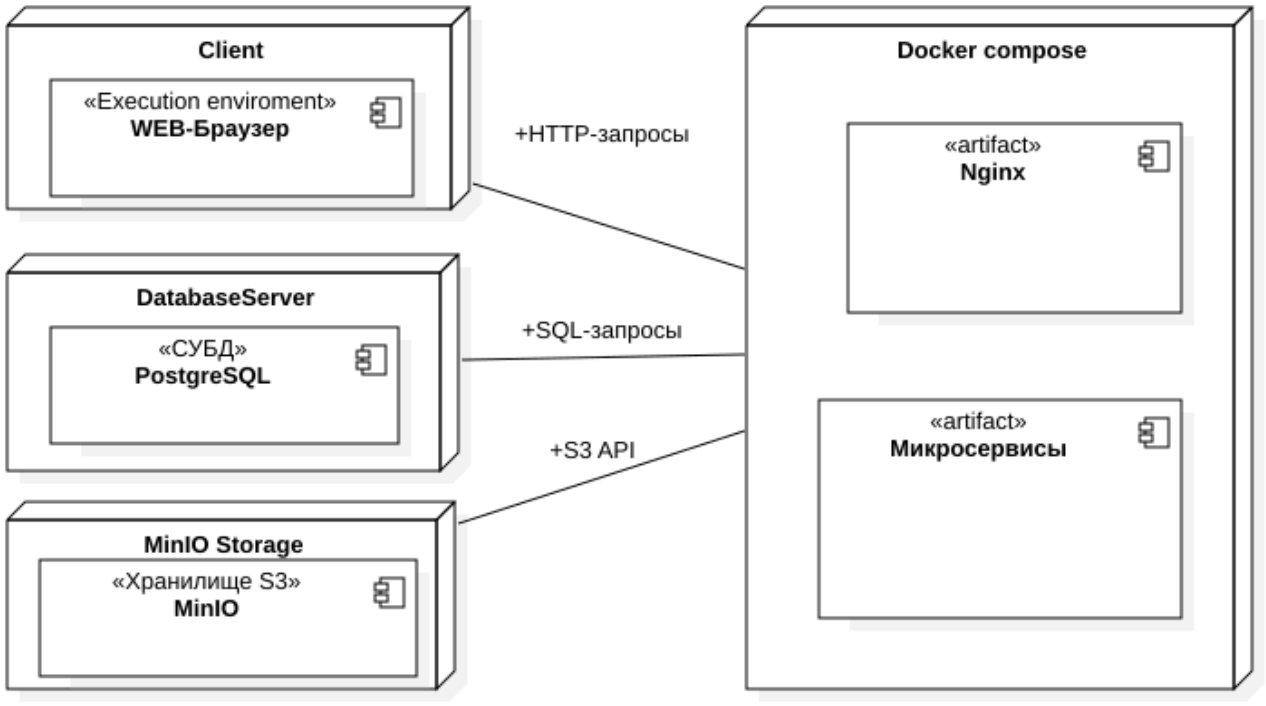

Рисунок 21 – Схема развёртывания компонентов B2B-маркетплейса

На схеме показаны четыре ключевых элемента инфраструктуры: рабочее место пользователя с установленным веб-браузером, узел приложений, сервер СУБД PostgreSQL и объектное хранилище MinIO. Взаимодействие начинается с того, что клиент отправляет HTTP-запросы из браузера на API-шлюз; тот, в свою очередь, направляет обращения к нужным микросервисам. При выполнении операций сервисы используют собственные базы PostgreSQL, а для работы с графическими материалами товаров – хранилище MinIO.

2.4 Рекомендуема конфигурация аппаратного обеспечения

Для развёртывания и эксплуатации серверной части B2B-маркетплейса в среде разработки и тестирования предъявляются следующие минимальные требования к аппаратному обеспечению рабочей станции или сервера,

обеспечивающего стабильную работу всей системы. Требования приведены на таблице 8.

Таблица 8 – Требования к аппаратному обеспечению серверной части

Компонент	Минимальные требования	Рекомендуемые требования
Процессор	4 ядра, 2.0 ГГц	8 ядер, 2.5 ГГц и выше
Оперативная память	4 ГБ	8 ГБ и выше
Дисковое пространство	20 ГБ SSD	50 ГБ SSD и выше
Сеть	100 Мбит/с	1 Гбит/с
ОС сервера	Linux	Linux

Предъявляемые требования к аппаратным ресурсам определяются необходимостью параллельного функционирования совокупности программных компонентов: клиентского веб-приложения, сервиса-шлюза, десяти микросервисов, а также средств обратного проксирования, хранения файлов и системы управления базами данных, обслуживающей десять отдельных баз. Совокупный объём вычислительной нагрузки формируется одновременной работой указанных контейнеров в единой среде развёртывания. Для условий промышленной эксплуатации целесообразно размещение СУБД PostgreSQL и объектного хранилища MinIO на самостоятельных вычислительных узлах. Дополнительно рекомендуется организовать контроль работоспособности сервисов посредством маршрутов проверки состояния (health-check) и выполнять регулярное резервное копирование баз данных с целью обеспечения непрерывности функционирования информационной системы.

3 Разработка приложения

3.1 Разработка архитектуры

Для реализации программных средств B2B-маркетплейса в рамках квалификационной работы принята микросервисная архитектурная модель. Выбор данного подхода обусловлен возможностью раздельного развёртывания сервисов, их функциональной автономностью и привязкой к конкретным бизнес-доменам предметной области. Взаимодействие между сервисами осуществляется по сетевым протоколам, при этом каждый компонент предоставляет внешний интерфейс, инкапсулирующий соответствующую прикладную функциональность [6].

Под микросервисами в контексте разрабатываемой системы понимаются автономные программные компоненты, каждый из которых решает ограниченный круг задач в рамках одной предметной области и может разрабатываться, тестироваться и развёртываться независимо от остальных модулей. В отличие от монолитной архитектуры, где вся бизнес-логика сосредоточена в едином приложении, микросервисный подход предполагает декомпозицию системы на совокупность слабо связанных служб, обменивающихся данными по сети посредством стандартизированных программных интерфейсов. Такая организация повышает масштабируемость отдельных функциональных блоков, упрощает сопровождение программного комплекса и позволяет локализовать изменения в пределах одного сервиса без переработки всей серверной части.

Серверный контур включает десять микросервисов, реализованных на языке Go, а также сервис-шлюз, выполняющий функции API Gateway. Для взаимодействия с внешними компонентами каждый микросервис предоставляет REST-интерфейс – набор HTTP-методов, обеспечивающих обмен данными в формате JSON. Такой подход позволяет клиентскому приложению обращаться к серверной подсистеме через унифицированные сетевые запросы без привязки к внутренней реализации отдельных служб.

API Gateway выступает единой точкой входа в распределённую систему: он принимает входящие запросы от клиента, выполняет предварительную проверку прав доступа и перенаправляет обращение к микросервису, ответственному за обработку конкретной операции. Использование шлюза снижает связанность клиентской части с внутренней топологией серверных компонентов и централизует функции аутентификации, маршрутизации и контроля доступа.

Микросервисы, оперирующие персистентными данными, используют изолированные экземпляры схем в СУБД PostgreSQL 15 по принципу «база данных на сервис». Данное решение ограничивает взаимное влияние сервисов при сбоях и изменениях структуры хранения, а также обеспечивает независимое развитие программных модулей.

Внутренняя организация каждого микросервиса построена по трёхслойной схеме: слой представления данных (Handler), уровень бизнес-логики (Service) и уровень доступа к данным (Repository). Типовая структура проекта включает каталоги Графическое представление принятой слоистой организации приведено на рисунке 22.

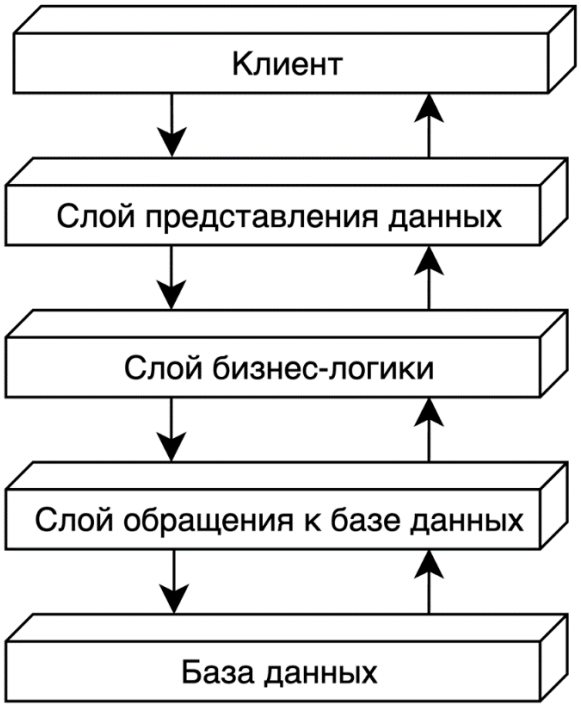

Рисунок 22 – Архитектура микросервиса

Слой представления данных отвечает за приём и первичную обработку входящих запросов. На нём выполняется проверка корректности передаваемых данных и извлечение служебной информации о пользователе, необходимой для дальнейшей обработки.

Слой бизнес-логики содержит правила предметной области. Именно на этом уровне принимаются решения, определяющие допустимость и последовательность выполнения операций.

Доступ к данным, хранящимся в PostgreSQL, инкапсулирован слоем обращения к базе данных (Repository). SQL-операции выполняются через стандартный интерфейс database/sql с использованием драйвера lib/pq. Изменения структуры баз данных оформляются в виде миграционных скриптов, их применение осуществляется в процессе развёртывания системы.

Программные интерфейсы микросервисов спроектированы в соответствии с архитектурным стилем REST:

– GET – чтение и запрос существующих данных;
– POST – добавление новых сущностей;
– PUT – изменение ранее созданных сущностей;
– DELETE – удаление объектов из системы.

Операции, не требующие немедленного ответа пользователю – в частности, начисление комиссионных вознаграждений и формирование закрывающих документов – реализованы по шаблону outbox. Фоновый обработчик последовательно извлекает необработанные записи и выполняет соответствующие действия. Механизм блокировки при чтении гарантирует, что одно и то же событие не будет обработано повторно, даже если в системе работает несколько экземпляров службы одновременно.

Клиентская подсистема представляет из себя одностраничное веб-приложение, обеспечивающие работу в рамках одной страницы, без нужды перезагрузки для обновления контента. Взаимодействие с серверной частью выполняется по архитектурному стилю REST через HTTP-запросы.

3.2 Разработка физической модели базы данных

Физическая модель данных представляет собой практическую реализацию логической схемы. Для каждой сущности подготовлен отдельный SQL-скрипт миграции. Сводные сведения о базах данных приведены в таблице 9.

Таблица 9 – Физические базы данных

База данных	Микросервис	Основные таблицы
БД «Аутентификация»	Сервис авторизации	user_types, users, credentials
БД «Корзина»	Сервис корзины	carts, cart_items
БД «Заказы»	Сервис заказов	orders, order_items, order_item_statuses, status_types, outbox_events
БД «Товары и предложения»	Сервис товаров и предложений	products, offers, quote_requests, quote_requests_cart_items
БД «Счета»	Сервис счетов	invoices
БД «Претензии»	Сервис претензий	claims, claim_resolutions
БД «Категории»	Сервис категорий	categories
БД «Аналитика»	Сервис аналитики	category_analytics, price_forecast, demand_forecast
БД «Монетизация»	Сервис монетизации	tariffs, subscriptions, commission_ledger, commission_statements
БД «Документы»	Сервис документов	documents, delivery_log

Описание всех сущностей представлено на таблице 10.

Таблица 10 – Описание всех сущностей системы

№	Наименование	Описание	Категория
1	user_types	Данные о типах пользователей	Кодовая
2	users	Список всех пользователей системы	Стержневая
3	credentials	Учетные данные пользователей	Характеристическая
4	categories	Справочник товарных категорий	Стержневая

Продолжение таблицы 10

№	Наименование	Описание	Категория
5	products	Список всех товаров продавцов	Стержневая
6	carts	Список корзин	Стержневая
7	cart_items	Позиции корзины	Характеристическая
8	quote_request	Запрос коммерческих предложений от покупателей	Стержневая
9	quote_request_cart_items	Связь запроса коммерческого предложения	Ассоциативная
10	offers	Коммерческие предложения продавцов	Стержневая
11	orders	Список всех заказов	Стержневая
12	order_items	Позиции заказа	Характеристическая
13	status_types	Справочник статусов позиций заказа	Кодовая
14	order_items_statuses	История статусов позиций заказа	Характеристическая
15	outbox_events	Очередь исходящих событий для межсервисной синхронизации	Характеристическая
16	invoices	Счета на оплату по заказам	Стержневая
17	claims	Претензии покупателей по позициям заказа	Стержневая
18	claim_resolutions	Решения по претензиям	Характеристическая
19	subscription_plans	Справочник тарифных планов подписки	Кодовая
20	seller_subscriptions	Подписки продавцов на тарифные планы	Стержневая
21	commission_accrullas	Начисления по закрытым сделкам	Стержневая
22	commission_statements	Сводные ведомости комиссий за период	Стержневая
23	subscription_invoices	Счета на оплату подписки	Стержневая
24	billing_outbox_events	Очередь событий монетизации для асинхронной обработки	Характеристическая
25	documents	Сгенерированные документы	Стержневая

Продолжение таблицы 10

№	Наименование	Описание	Категория
26	document_deliveries	Статусы доставки документов по каналам	Характеристическая
27	category_analytics	Агрегированная аналитика по категориям	Характеристическая
28	price_forecast	Прогнозы цен по категориям	Характеристическая
29	demand_forecast	Прогнозы спроса по категориям	Характеристическая

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

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

На рисунке 23 приведена физическая модель базы данных «Аутентификация».

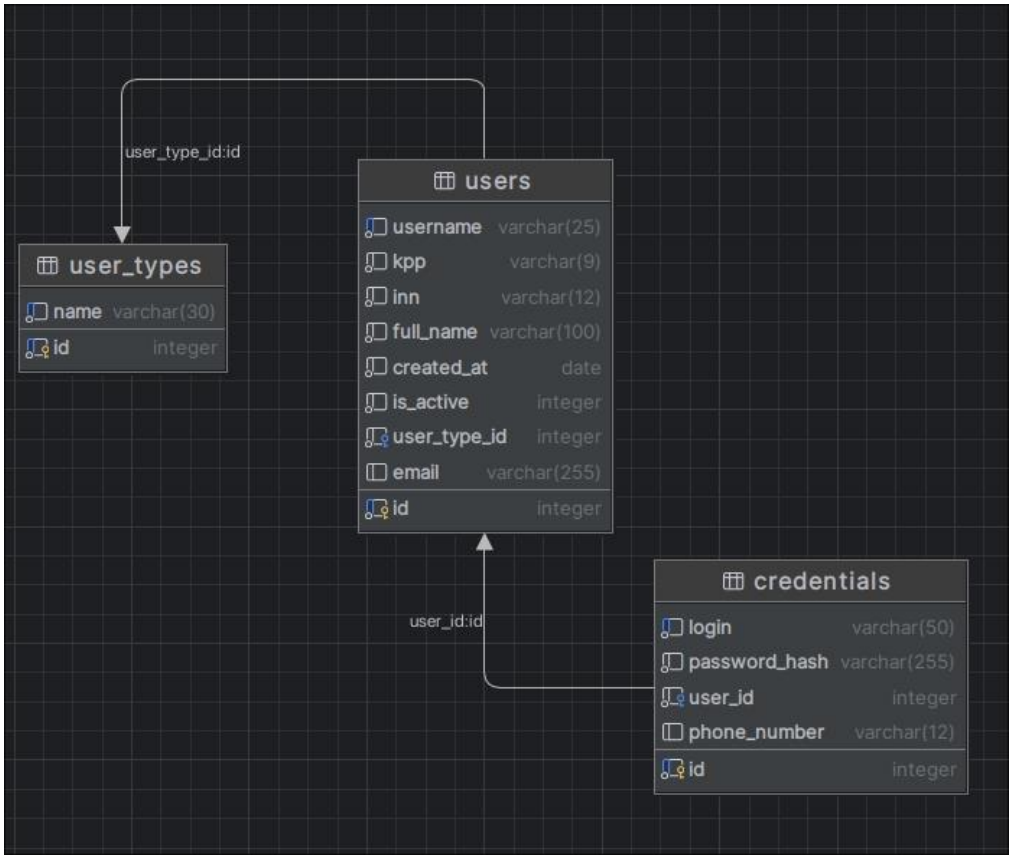

Рисунок 23 – Физическая модель БД «Аутентификация»

На рисунке 24 приведена физическая модель базы данных «Монетизация».

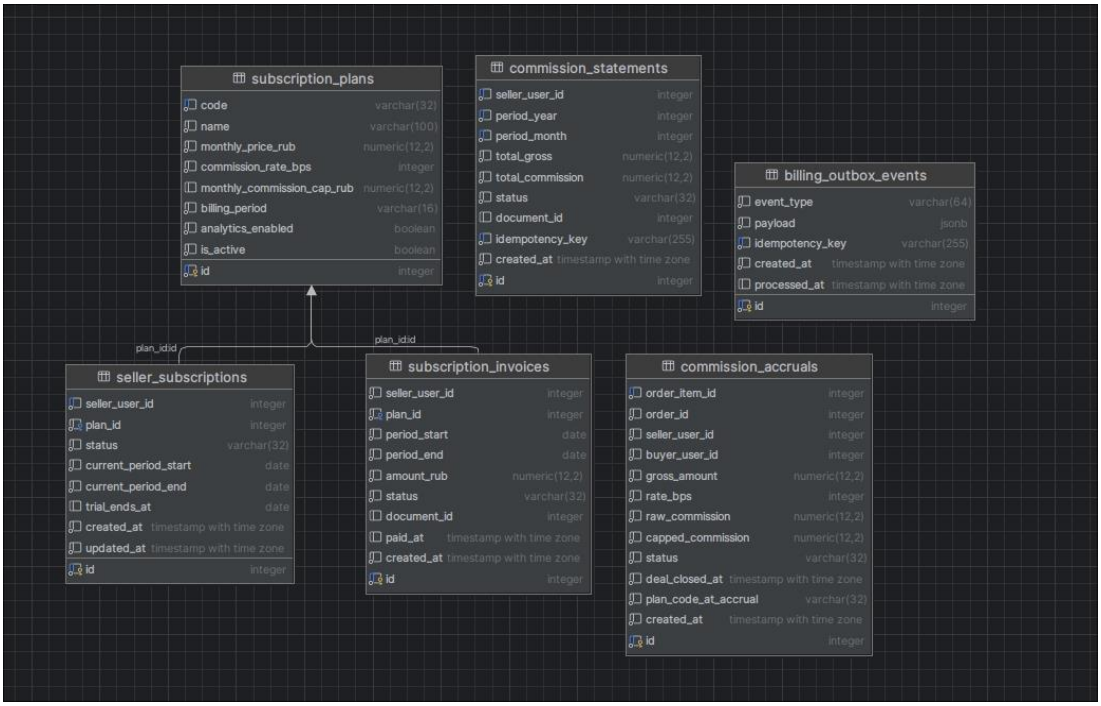

Рисунок 24 – Физическая модель БД «Монетизация»

На рисунке 25 приведена физическая модель базы данных «Аналитика».

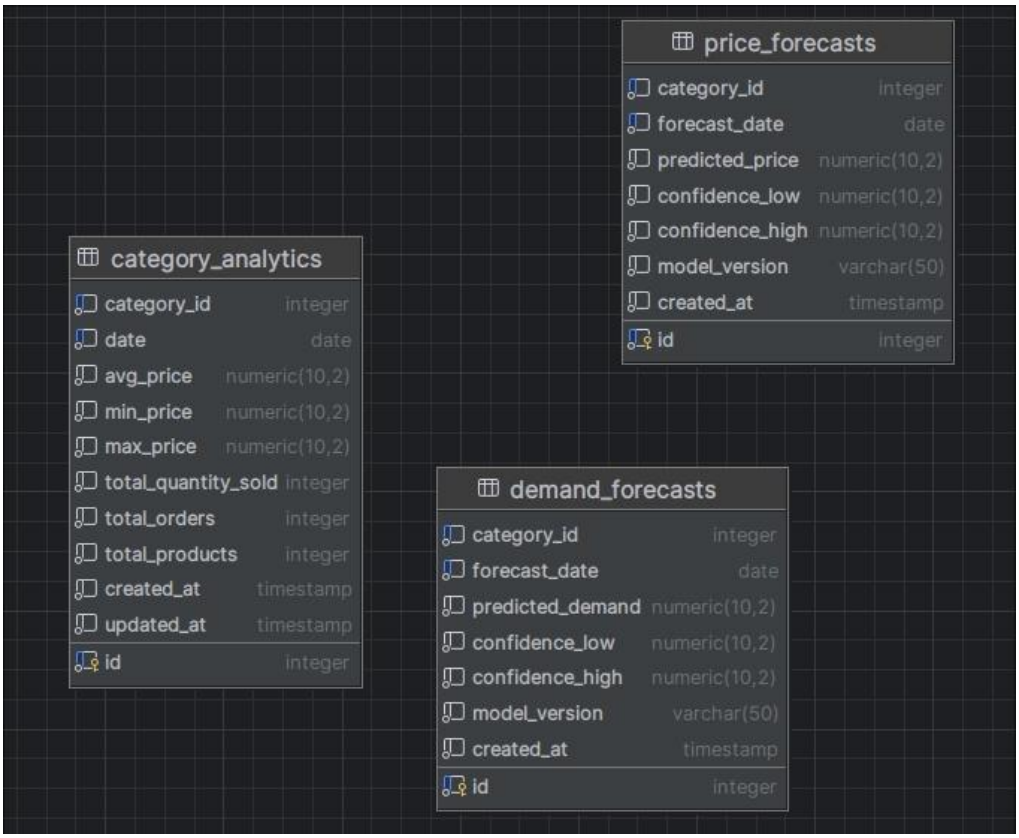

Рисунок 25 – Физическая модель БД «Аналитика»

На рисунке 26 приведена физическая модель базы данных «Корзина».

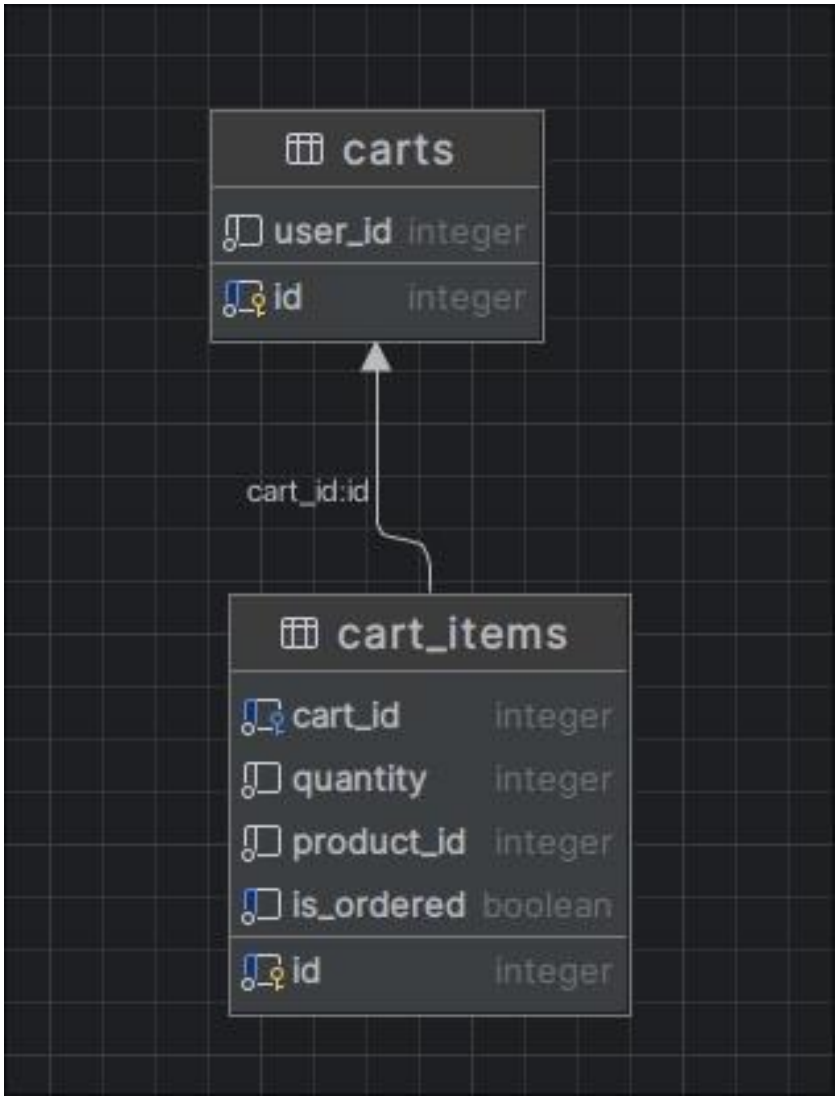

Рисунок 26 – Физическая модель БД «Корзина»

На рисунке 27 приведена физическая модель базы данных «Категории».

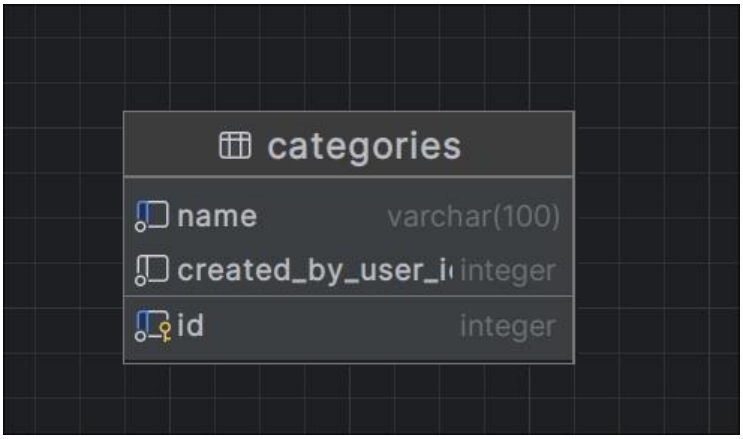

Рисунок 27 – Физическая модель БД «Категории»

На рисунке 28 приведена физическая модель базы данных «Претензии».

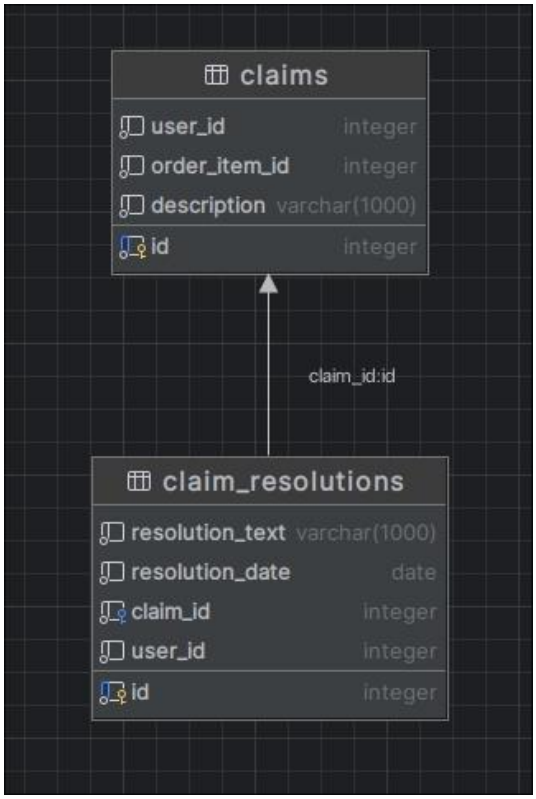

Рисунок 28 – Физическая модель БД «Претензии»

На рисунке 29 приведена физическая модель базы данных «Документы».

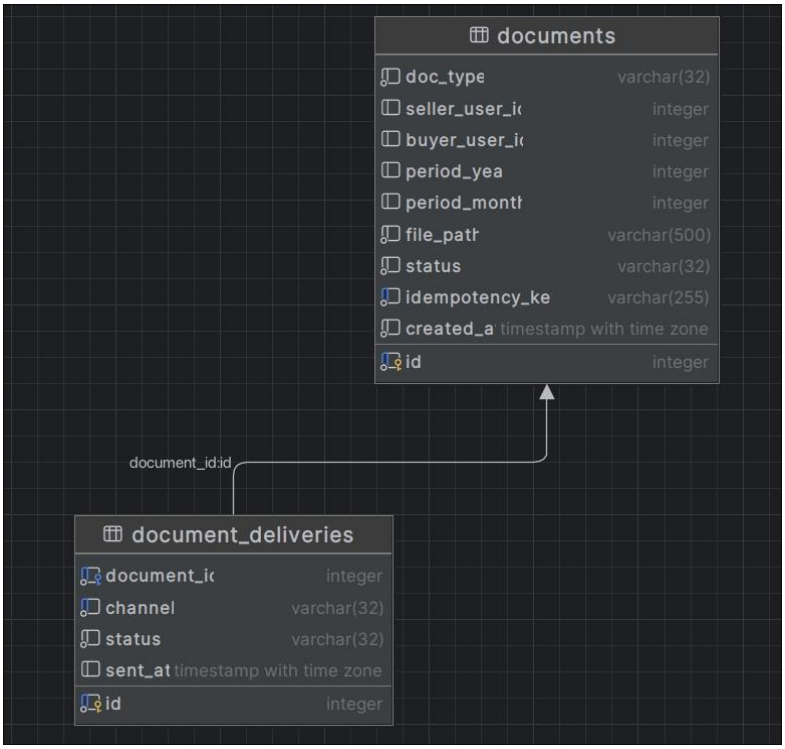

Рисунок 29 – Физическая модель БД «Документы»

На рисунке 30 приведена физическая модель базы данных «Счета».

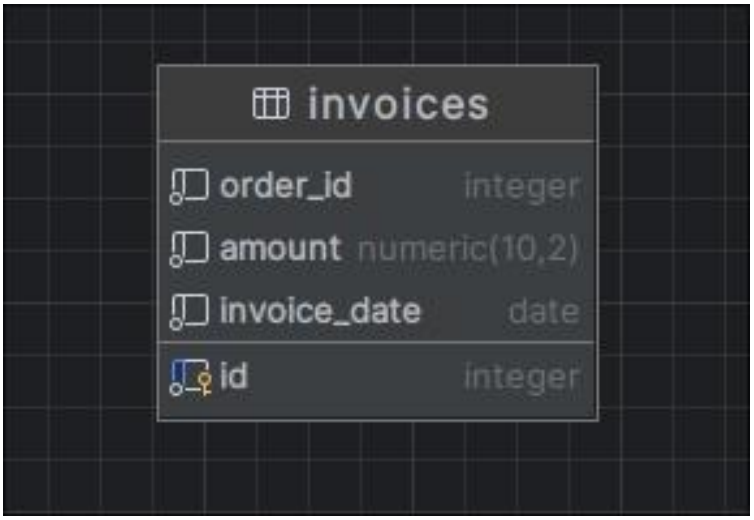

Рисунок 30 – Физическая модель БД «Счета»

На рисунке 31 приведена физическая модель базы данных «Товары и предложения».

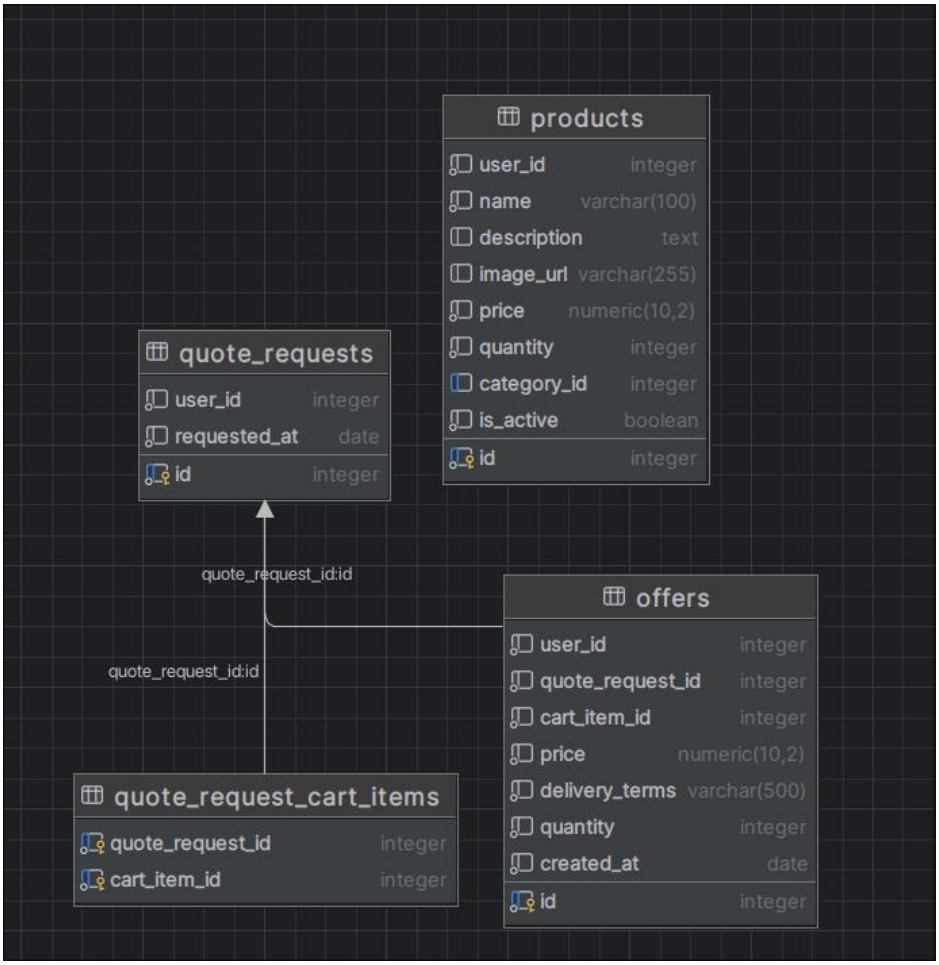

Рисунок 31 – Физическая модель БД «Товары и предложения»

На рисунке 32 приведена физическая модель базы данных «Заказы».

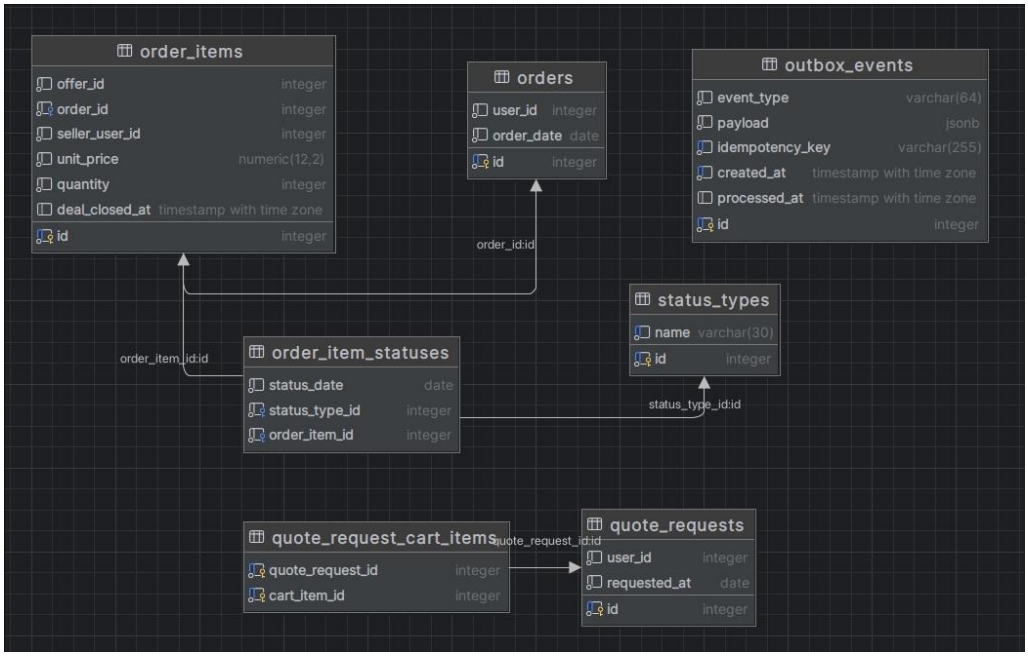

Рисунок 32 – Физическая модель БД «Заказы»

Графические материалы товаров хранятся в объектном хранилище MinIO, совместимом со стандартом S3. Ответственность за размещение файлов и формирование адресов для их отображения в интерфейсе возложена на сервис товаров и предложений. Параметры публичного доступа задаются при конфигурировании системы.

3.3 Разработка архитектурных слоев внутри микросервисов

3.3.1 Разработка слоя представления данных

Слой представления данных реализован в составе каждого микросервиса и выполняет функции приёма входящих HTTP-обращений, направления их в нужные модули и возврата результата на сторону клиента. В момент инициализации сервиса подключаются обработчики запросов, формирующие согласованный способ взаимодействия с внешними элементами системы.

В качестве иллюстрации работы слоя представления рассмотрена операция оформления заказа. При поступлении запроса обработчик извлекает идентификатор пользователя из служебных заголовков, переданных сервисом-

шлюзом, и передаёт управление уровню бизнес-логики. Далее формируется перечень позиций корзины, после чего сведения о заказе направляются в сервис заказов (order-service) посредством межсервисного сетевого вызова. Результат операции возвращается клиенту в структурированном формате JSON. Фрагмент реализации указанного метода представлен на листинге 1.

Листинг 1 – Реализация оформления заказа

```
func (h *RestHandler) PurchaseCart(w http.ResponseWriter, r
*http.Request) {
     userID, err := middleware.ExtractUserID(r)
     if err != nil {
          response.Error(w, err)
          return
     }

     items, err := h.cartService.PurchaseCart(userID)
     if err != nil {
          response.Error(w, err)
          return
     }

     cartItemIDs := make([]int64, len(items))
     for i, item := range items {
          cartItemIDs[i] = item.ID
     }

     _, err = h.orderClient.CreateOrder(r.Context(), userID,
cartItemIDs)
     if err != nil {
          response.Error(w, err)
          return
     }

     if err := h.cartService.RemoveItems(userID, cartItemIDs); err
!= nil {
          fmt.Printf("Warning: failed to remove cart items: %v\n",
err)
     }

     response.JSON(w, http.StatusOK, map[string]interface{}{
          "success": true,
          "message": "Order created successfully",
     })
}
```

Сервис-шлюз выполняет две ключевые функции: проверку подлинности JWT-токена на промежуточном уровне обработки запроса и маршрутизацию

обращений к целевым микросервисам. Входящие запросы распределяются по функциональным областям системы – аутентификация, корзина, заказы, товары, счета, претензии, категории, аналитика, монетизация и документооборот – и перенаправляются к соответствующим серверным компонентам.

Регистрация маршрутов выполняется в точке входа приложения и определяет полный перечень операций, доступных клиентской части в рамках данного микросервиса. Каждый маршрут связывает HTTP-метод и шаблон адреса с конкретным обработчиком слоя представления данных. Пример регистрации маршрутов сервиса корзины представлен на листинге 2.

Листинг 2 – Регистрация маршрутов

```
r := chi.NewRouter()
r.Get("/api/health", func(w http.ResponseWriter, r *http.Request)
{
     w.Header().Set("Content-Type", "application/json")
     w.WriteHeader(http.StatusOK)
     w.Write([]byte(`{"status":"ok"}`))
})
r.Route("/api/cart", func(r chi.Router) {
     r.Get("/", restHandler.GetCart)
     r.Post("/items", restHandler.AddItem)
     r.Put("/items/{id}", restHandler.UpdateItemQuantity)
     r.Delete("/items/{id}", restHandler.RemoveItem)
     r.Delete("/items", restHandler.ClearCart)
     r.Post("/checkout", restHandler.PurchaseCart)
     r.Get("/items/{id}", restHandler.GetCartItem)
})
srv := &http.Server{
     Addr:    fmt.Sprintf(":%s", *restPort),
     Handler: r,
}
```

3.3.2 Разработка слоя бизнес-логики в микросервисах

На бизнес-слое (сервисный слой) микросервиса принимаются решения о допустимости операций и содержимом операций, выполняется согласование данных между компонентами и формируется результат, передаваемый на уровень хранения. Слой представления данных ограничивается приёмом запросов и делегированием – вся предметная логика сосредоточена именно в сервисном уровне.

Сервис корзины отвечает за подготовительный этап закупки: накопление выбранных позиций и их последующую передачу в контур оформления заказа. Перед добавлением товара система запрашивает актуальные сведения у сервиса управления предложениями и последовательно проверяет три условия: позитивность запрашиваемого количества, активность карточки товара и наличие достаточного остатка с учётом уже отложенных в корзине единиц. При оформлении заказа формируется полный перечень позиций, который направляется в сервис управления заказами; после успешного создания заказа соответствующие записи исключаются из корзины покупателя. Листинг 3 иллюстрирует проверки при добавлении позиции.

Листинг 3 – Проверка условий при добавлении товара в корзину

```
product, err := s.offerClient.GetProduct(context.Background(),
productID)
if !product.IsActive {
    return fmt.Errorf("product is not available")
}
if product.Quantity < quantity {
    return fmt.Errorf("not enough stock: available %d, requested
%d",
        product.Quantity, quantity)
}
existingQty, err := s.cartRepo.GetItemQuantityByProduct(cartID,
productID)
if existingQty+quantity > product.Quantity {
    return fmt.Errorf("not enough stock: available %d, requested
%d",
        product.Quantity, existingQty+quantity)
}
return s.cartRepo.AddItem(cartID, productID, quantity)
```

Сервис аналитики выполняет расчёт прогнозных значений цен и объёмов спроса в разрезе товарных категорий. За основу взят метод байесовской линейной регрессии: на вход подаётся временной ряд исторических наблюдений, на выходе – ожидаемое значение показателя и границы доверительного интервала, отражающие разброс возможных исходов. Расчёт опирается на признаки календарного характера – день недели, сезонность, праздничные дни. Фоновый процесс с интервалом 24 часа пересчитывает статистику за последние 90 суток и строит прогноз на 30 дней; полученные данные выводятся на странице

категорий в виде графика. Фрагмент цикла построения прогноза приведён в листинге 4.

Листинг 4 – Формирование прогнозных значений с доверительным интервалом

```
for i := 0; i < forecastDays; i++ {
    forecastDate := lastDate.AddDate(0, 0, i+1)
    forecastFeatures := extractFeatures(forecastDate, startDate)
    predicted := predictWithMultipleRegression(forecastFeatures,
coefficients)
    stdErr := math.Sqrt(posteriorVariance * (1.0 + 1.0/n))
    confidenceInterval := 1.96 * stdErr
    results[i] = ForecastResult{
        PredictedValue: math.Max(0, predicted),
        ConfidenceLow:  math.Max(0, predicted-confidenceInterval),
        ConfidenceHigh: predicted + confidenceInterval,
    }
}
```

Описанные сервисные компоненты охватывают два различных класса задач: операционное сопровождение закупочного процесса со стороны покупателя и аналитическую поддержку принятия решений на основе статистических данных. Разделение указанной логики по самостоятельным микросервисам позволяет изменять алгоритмы прогнозирования и правила работы с корзиной независимо друг от друга.

3.3.3 Разработка слоя обращения к базе данных

Слой доступа к данным обеспечивает установку соединения с СУБД PostgreSQL и выполнение операций чтения, записи, изменения и удаления записей. В микросервисной архитектуре платформы этот слой вынесен в отдельные структуры – репозитории. Их имена заканчиваются на Repository, что позволяет однозначно отнести компонент к уровню работы с хранилищем. Репозиторий не реализует правила предметной области: он лишь переводит вызовы сервисного слоя в SQL-запросы и возвращает результат в виде структур данных Go.

В сервисе корзины (cart-service) за взаимодействие с базой данных «Коризна» отвечает CartRepository. При старте приложения в функции main

создаётся пул соединений, после чего репозиторий передаётся в конструктор сервиса.

Инициализация и перечень методов приведены в листинге 5.

Листинг 5 – Инициализация репозитория корзины и описание методов

```
type CartRepository struct {
     db *sql.DB
}
func NewCartRepository(db *sql.DB) *CartRepository {
     return &CartRepository{db: db}
}
```

Основные методы CartRepository:

– GetOrCreateCart – поиск корзины пользователя или её создание;

– GetCartItems – выборка позиций с пагинацией;

– AddItem – добавление товара в корзину;

– RemoveItem – удаление товара из корзины;

– UpdateItemQuantity – изменение количество товара в корзине;

– ClearCart – очистка корзины;

– RemoveItems – очистка корзины после оформления заказа.

SQL-запросы размещены непосредственно в теле методов и выполняются через стандартный пакет database/sql драйвера lib/pq.

На листинге 6 показан фрагмент метода получения содержимого корзины.

Листинг 6 – Пример метода слоя доступа к данным

```
func (r *CartRepository) GetCartItems(cartID int64, limit, offset
int) ([]models.CartItem, int, error) {
     var total int
     countQuery := `SELECT COUNT(*) FROM cart_items WHERE cart_id =
$1 AND is_ordered = FALSE`
     err := r.db.QueryRow(countQuery, cartID).Scan(&total)
     if err != nil {
          return nil, 0, fmt.Errorf("failed to count cart items:
%w", err)
     }
     query := `
          SELECT ci.id, ci.product_id, ci.quantity
          FROM cart_items ci
          WHERE ci.cart_id = $1 AND is_ordered = FALSE
          ORDER BY ci.id
          LIMIT $2 OFFSET $3
```

```
    `
    rows, err := r.db.Query(query, cartID, limit, offset)
    // ...
    return items, total, nil
}
```

Метод сначала подсчитывает общее число неоформленных позиций, затем выбирает страницу записей с учётом параметров. В результат попадают только идентификатор позиции, идентификатор товара и количество – сведения о самом товаре в таблицах корзины не дублируются и подгружаются на следующем уровне.

Чтобы показать, как согласованно работают слои при типовом сценарии, рассмотрим получение корзины покупателя. Для него подготовлена диаграмма последовательности (рисунок 33).

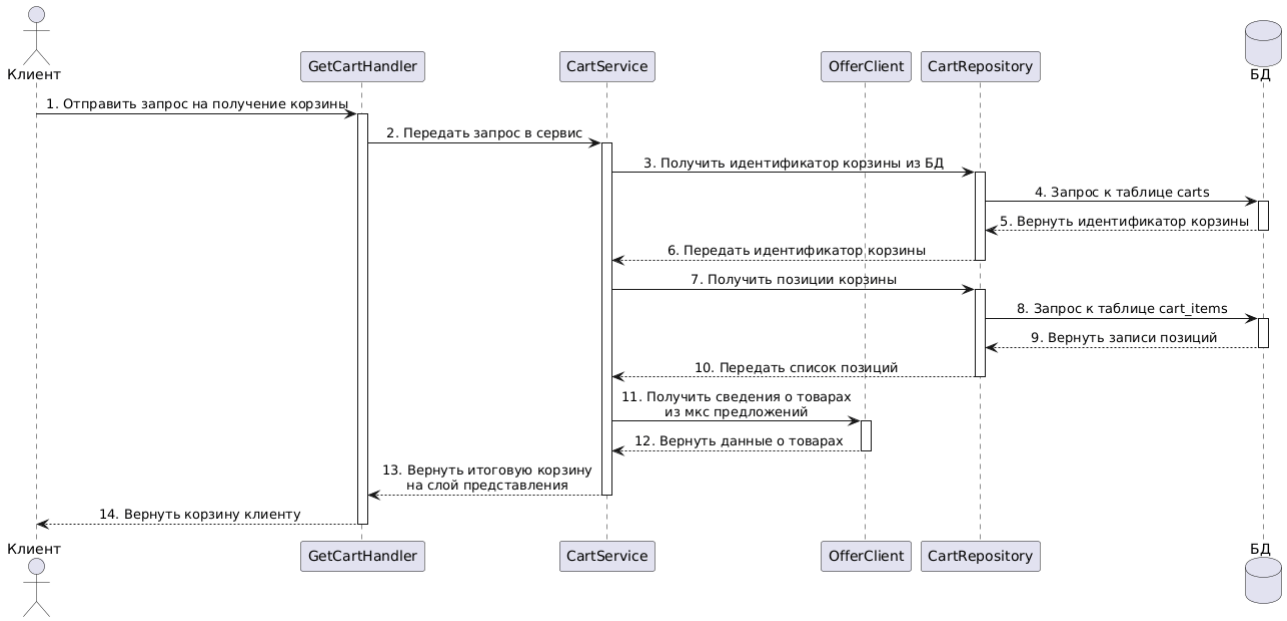

Рисунок 33 – Диаграмма последовательности получения корзины покупателя

Цепочка начинается с HTTP-запроса. Его принимает обработчик GetCart слоя представления: извлекается идентификатор пользователя из контекста запроса, разбираются параметры постраничного вывода, далее управление передаётся в CartService.

На уровне бизнес-логики GetCart обращается к CartRepository. Репозиторий вызывает GetOrCreateCart, чтобы получить идентификатор

корзины, после чего – GetCartItems для выборки позиций из PostgreSQL. На этом этапе в ответе содержатся лишь ссылки на товары, без наименований и цен.

Далее сервис для каждой позиции вызывает enrichCartItem, который через HTTP-клиент offer.Client запрашивает микросервис предложений (offer-service). Там подтягиваются название, цена, изображение и остаток на складе. Собранные данные объединяются в структуру Cart и возвращаются обработчику. RestHandler формирует JSON-ответ и отправляет его клиентскому приложению, где корзина отображается в интерфейсе.

Такое разделение обязанностей позволяет хранить в базе корзины только ссылки на товары, а актуальные сведения о продукции получать из сервиса, которому они принадлежат. Репозиторий остаётся тонким слоем над СУБД, а вся проверка наличия и доступности товара сосредоточена в CartService.

4 Контроль качества программного обеспечения

4.1 Выбор модели тестирования и метрик кода

При создании web-приложения существенный акцент делается на поддержании высокого качества программного кода. Сочетание различных видов тестирования и процедур контроля качества создаёт условия для стабильной и эффективной работы микросервисной архитектуры.

К тестированию серверной части в работе относятся следующие направления:

– модульное тестирование – контроль правильности функционирования отдельных компонентов в изолированной среде [15];

– интеграционное тестирование – проверка корректности обмена данными между микросервисами [16];

– тестирование производительности – измерение времени обработки запросов и способности системы к масштабированию;

– функциональное тестирование – подтверждение того, что реализованные функции соответствуют сформулированным требованиям [17].

Показатели качества кода занимают важное место при оценке надёжности решения. В выпускной квалификационной работе для анализа использованы метрика SLOC и метрики Холстеда [18].

SLOC (Source Lines of Code) – показатель, характеризующий размер исходного кода. По итогам подсчёта объём серверной части (Go, без Frontend) составляет около 10890 строк; клиентская часть (TypeScript/React) – около 6434 строк. Распределение SLOC по микросервисам приведено в таблице 11.

Таблица 11 – Метрики кода

Микросервис	Строк кода	Назначение
analytics-service	1461	Аналитика и прогнозы
billing-service	1455	Тарифы, комиссии, подписки
order-service	993	Заказы и закрытие сделок
offer-service	907	Товары, предложения

Продолжение таблицы 11

Микросервис	Строк кода	Назначение
document-service	888	Закрывающие документы, счета
cart-service	808	Корзина и оплата
auth-service	683	Пользователи и учетные данные
claim-service	434	Претензии
category-service	434	Категории
auth-gateway	427	Сервис-шлюз
invoice-service	612	Счета

Метрика Холстеда обеспечивает оценку сложности программы, прогнозирование трудоёмкости и анализ потенциальной надёжности [17].

4.2 Тестирование B2B-маркетплейса

4.2.1 Функциональное тестирование создания товара

Помимо того проводилось функциональное тестирование. Этот процесс позволяет контролировать соответствие приложения поставленным требованиям и спецификациям [16].

Для функционального тестирования выбран сценарий «Создание товара». Сценарий позволяет проверить загрузку изображения, проверку полей и публикацию товара в каталоге.

Выделены следующие требования:

– при пустом наименовании товара либо при его длине менее трёх символов система должна выводить сообщение об ошибке пользователю с ролью продавец;

– если цена или количество не заданы положительными числами, пользователю должно отображаться уведомление об ошибке пользователю с ролью продавец;

– при незаполненном описании товара программа обязана сигнализировать об ошибке;

– при вводе корректных данных созданная позиция должна отображаться в каталоге продавца.

Для сценария определено 4 причины и 5 следствий (таблица 12).

Таблица 12 – Причины и следствия (создание товара)

Обозначение	Описание
П1	Корректное название (3–100 символов)
П2	Корректная цена (> 0)
П3	Корректное количество (целое > 0)
П4	Заполнено описание
С1	Ошибка: некорректное название
С2	Ошибка: некорректная цена
С3	Ошибка: некорректное количество
С4	Ошибка: пустое описание
С5	Товар успешно создан

Иллюстрация экрана создания товара – на рисунке 34.

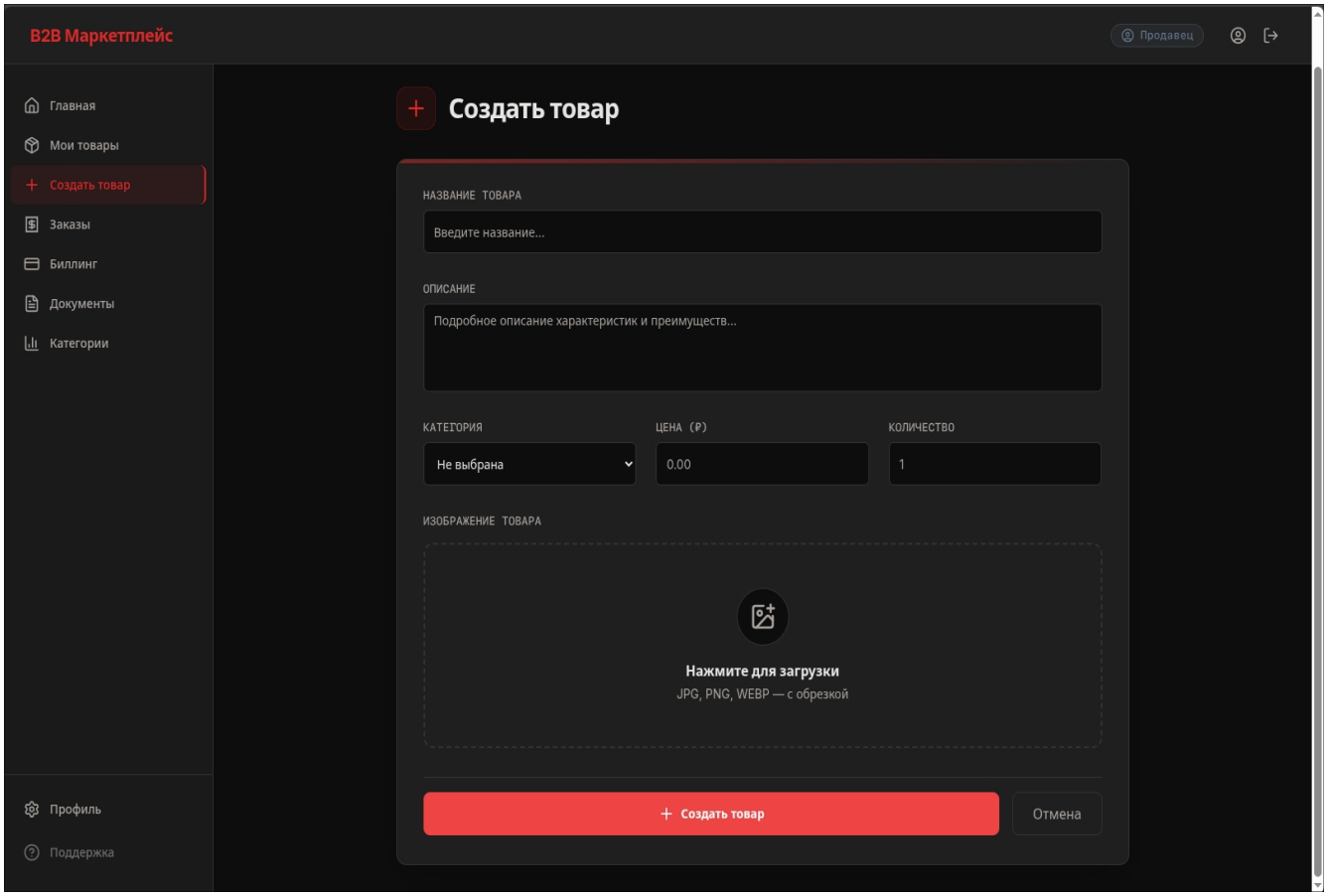

Рисунок 34 – Экран создания товара (функциональное тестирование)

Результаты функционального тестирования представлены в таблице 13.

Таблица 13 – Результаты функционального тестирования создания товара

№	Название	Цена	Кол-во	Описание	Ожидаемый результат	Фактический результат
1		100	10	Описание товара	C1	C1
2	A	100	10	Описание товара	C1	C1
3	Кабель UTP	-5	10	Описание	C2	C2
4	Кабель UTP	100	0	Описание	C3	C3
5	Кабель UTP	100	10		C4	C4
6	Кабель UTP	100	10	Описание кабеля	C5	C5

Для приложения успешно проведено функциональное тестирование, по результатам тестирования программное обеспечение соблюдает все функциональные требования. Поведение системы соответствует заявленным спецификациям.

4.2.2 Модульное тестирование создания заказа

Для сервиса заказов проведено модульное тестирование. Проверяемый метод: метод создания заказа (CreateOrder) с использованием пакета testing языка Go. Тесты выполняются с mock-объектами HTTP-клиентов корзины (cart) и предложения (offer). Исходный код тестов – в приложении Г. Результат тестирования представлен на таблице 14.

Таблица 14 – Результаты модульного тестирования CreateOrder

№	Входные данные	Ожидаемый результат	Фактический результат	Статус
1	Корректные cart_item_ids, достаточный остаток	order_id > 0	order_id > 0	Пройден
2	Пустой список cart_item_ids	Ошибка валидации	Ошибка валидации	Пройден
3	Количество > остатка на складе	Ошибка not enough quantity	Ошибка not enough quantity	Пройден
4	offer-service недоступен	Ошибка failed to create offer	Ошибка failed to create offer	Пройден

Модульное тестирование прошло успешно. Программное обеспечение корректно отрабатывает.

4.2.3 Метрики Холстеда для метода создания заказа

Метрики Холстеда рассчитываются на основе анализа операторов и операндов исходного кода. В качестве примера рассмотрен метод создания заказа (CreateOrder) исходный код в приложении Е. Операторы представлены на таблице 15.

Таблица 15 – Операторы метода CreateOrder

Оператор	Количество
If	3
For	2
Return	8
:=	12
=	4
fmt.Errorf	7
Append	1
Break	1
n1	9
N1	38

Операнды представлены на таблице 16.

Таблица 16 – Операнды метода CreateOrder

Операнд	Количество вхождений
S	14
Err	18
cartItemID	4
cartItem	6
Product	8
offerID	3
Product	8
offerID	3
itemInputs	2
quoteRequestID	3
Offers	3
Nil	0
n2	10
N2	64

Длина программы: $N = N1 + N2 = 102$. Словарь: $n = n1 + n2 = 19$. Объём: $V = N \times \log2(n) \approx 433$. Вычислительная сложность: $D = (n1 \times N2) / (2 \times n2) \approx 3{,}38$. Сводное значение метрики качества кода укладывается в допустимые границы, что указывает на умеренную сложность и достаточную читаемость исходного кода для дальнейшей поддержки.

5 Планирование разработки и оценка бюджета

Планирование работ и расчёт бюджета выступают ключевыми условиями успешного создания программного продукта [19].

Для ведения проекта выбрана среда Project Libre: она используется для составления плана разработки, мониторинга выполнения задач и контроля задействованных ресурсов.

После занесения сведений об этапах создания приложения в систему сформирован календарный план работ и выполнена оценка планируемых финансовых расходов. Наглядное отображение последовательности и сроков выполнения задач обеспечивает диаграмма Ганта, на которой отмечены основные стадии разработки ПО.

Диаграмма Ганта представляет собой табличный график проектных работ с указанием этапов и исполнителей. Её структура включает две координатные оси: по вертикали размещается перечень задач, по горизонтали – временная шкала [20].

Полученная диаграмма приведена на рисунке 36.

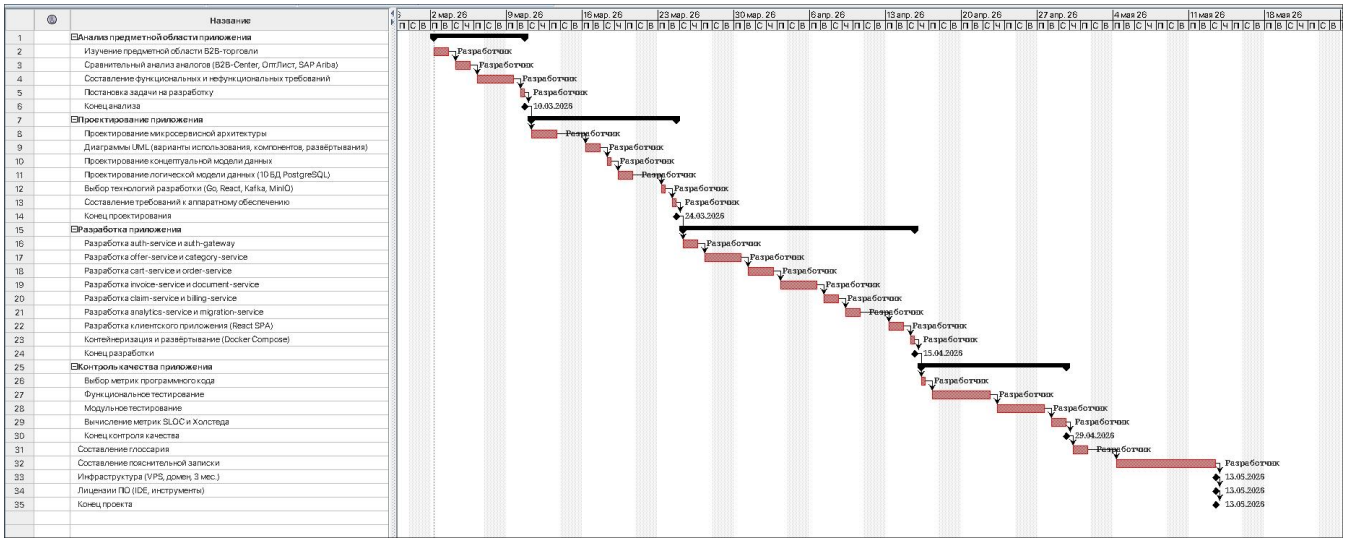

Рисунок 36 – Диаграмма Ганта B2B-маркетплейса

В рамках проекта сформирован перечень из 30 задач. Их типы, трудозатраты и задействованные ресурсы приведены на рисунке 37.

		Название	Продолжит...	Начало	Окончание	Предшествующие
1		**Анализ предметной области приложения**	**7 дней**	**02.03.2026, 08:00**	**10.03.2026, 17:00**	
2		Изучение предметной области B2B-торго...	2 дней	02.03.2026, 08:00	03.03.2026, 17:00	
3		Сравнительный анализ аналогов (B2B-Се...	2 дней	04.03.2026, 08:00	05.03.2026, 17:00	2
4		Составление функциональных и нефункци...	2 дней	06.03.2026, 08:00	09.03.2026, 17:00	3
5		Постановка задачи на разработку	1 день	10.03.2026, 08:00	10.03.2026, 17:00	4
6		Конец анализа	0 дней	10.03.2026, 17:00	10.03.2026, 17:00	5
7		**Проектирование приложения**	**10 дней**	**11.03.2026, 08:00**	**24.03.2026, 17:00**	
8		Проектирование микросервисной архите...	3 дней	11.03.2026, 08:00	13.03.2026, 17:00	6
9		Диаграммы UML (варианты использовани...	2 дней	16.03.2026, 08:00	17.03.2026, 17:00	8
10		Проектирование концептуальной модели ...	1 день	18.03.2026, 08:00	18.03.2026, 17:00	9
11		Проектирование логической модели данн...	2 дней	19.03.2026, 08:00	20.03.2026, 17:00	10
12		Выбор технологий разработки (Go, React, ...	1 день	23.03.2026, 08:00	23.03.2026, 17:00	11
13		Составление требований к аппаратному о...	1 день	24.03.2026, 08:00	24.03.2026, 17:00	12
14		Конец проектирования	0 дней	24.03.2026, 17:00	24.03.2026, 17:00	13
15		**Разработка приложения**	**16 дней**	**25.03.2026, 08:00**	**15.04.2026, 17:00**	
16		Разработка auth-service и auth-gateway	2 дней	25.03.2026, 08:00	26.03.2026, 17:00	14
17		Разработка offer-service и category-service	2 дней	27.03.2026, 08:00	30.03.2026, 17:00	16
18		Разработка cart-service и order-service	3 дней	31.03.2026, 08:00	02.04.2026, 17:00	17
19		Разработка invoice-service и document-ser...	2 дней	03.04.2026, 08:00	06.04.2026, 17:00	18
20		Разработка claim-service и billing-service	2 дней	07.04.2026, 08:00	08.04.2026, 17:00	19
21		Разработка analytics-service и migration-se...	2 дней	09.04.2026, 08:00	10.04.2026, 17:00	20
22		Разработка клиентского приложения (Rea...	2 дней	13.04.2026, 08:00	14.04.2026, 17:00	21
23		Контейнеризация и развёртывание (Docke...	1 день	15.04.2026, 08:00	15.04.2026, 17:00	22
24		Конец разработки	0 дней	15.04.2026, 17:00	15.04.2026, 17:00	23
25		**Контроль качества приложения**	**10 дней**	**16.04.2026, 08:00**	**29.04.2026, 17:00**	
26		Выбор метрик программного кода	1 день	16.04.2026, 08:00	16.04.2026, 17:00	24
27		Функциональное тестирование	4 дней	17.04.2026, 08:00	22.04.2026, 17:00	26
28		Модульное тестирование	3 дней	23.04.2026, 08:00	27.04.2026, 17:00	27
29		Вычисление метрик SLOC и Холстеда	2 дней	28.04.2026, 08:00	29.04.2026, 17:00	28
30		Конец контроля качества	0 дней	29.04.2026, 17:00	29.04.2026, 17:00	29
31		Составление глоссария	2 дней	30.04.2026, 08:00	01.05.2026, 17:00	30
32		Составление пояснительной записки	7,5 дней	04.05.2026, 08:00	13.05.2026, 13:00	31
33		Инфраструктура (VPS, домен, 3 мес.)	0 дней	13.05.2026, 13:00	13.05.2026, 13:00	32
34		Лицензии ПО (IDE, инструменты)	0 дней	13.05.2026, 13:00	13.05.2026, 13:00	33

Разработка B2B-маркетплейса- стр.1

Рисунок 37 – Задачи проекта

Все работы выполнял один специалист – разработчик. Состав ресурсов показан на рисунке 38.

	Название	Тип	E-mail адрес	Стандартная ставка	Инициалы
	Разработчик	Работа	k-shaipov@mail.ru	500/час	Р

Рисунок 38 – Ресурс «Разработчик»

Таким образом было проведено планирование бюджета, на разработку приложения по итогам планировая выделяется 420 часов на сумму в 230000 рублей.

Заключение

В рамках выпускной квалификационной работы была спроектировано и разработано web-приложение – специализированная B2B-маркетплейс для автоматизации процессов оптовых закупок и продаж.

Система реализована на языке Go (серверная часть, 10 микросервисов) и клиентское приложение. Взаимодействие организовано по REST через сервис-шлюз с JWT-авторизацией. Данные хранятся в 10 изолированных базах PostgreSQL 15, изображения товаров – в MinIO.

На этапе анализа проведён разбор предметной области оптовой торговли, сравнительный анализ аналогов (B2B-Center, ОптЛист, SAP Ariba), сформулированы функциональные и нефункциональные требования, определены категории пользователей: покупатель, продавец, модератор.

На стадии проектирования выбрана микросервисная архитектура, разработаны концептуальная и логическая модели данных, определены требования к аппаратному обеспечению.

Разработка включала реализацию RFQ-контура, механизма претензий, аналитики с байесовской регрессией, монетизации и генерации закрывающих документов.

Проведён контроль качества: функциональное и модульное тестирование, расчёт метрик SLOC и Холстеда. По итогам тестирования подтверждена корректность работы ключевых сценариев. Затраты на разработку – 420 часов, бюджет – 230 000 рублей.

Таким образом, разработанная B2B-площадка обеспечивает автоматизацию оптовых закупок и продаж, соответствует поставленным целям и задачам и может быть использована как основа для промышленного внедрения и дальнейшего развития (интеграция с ЭДО, расширение аналитики).

Список использованных источников

1. Дербенёв А.М., Столярова А.Н. Эволюция корпоративных B2B ландшафтов в Российской Федерации и Европейском Союзе: сравнительный анализ B2B-маркетплейсов // Прогрессивная экономика. 2026. № 3. С. 232 – 252. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/evolyutsiya-korporativnyh-b2b-landshaftov-v-rossiyskoy-federatsii-i-evropeyskom-soyuze-sravnitelnyy-analiz-b2b-marketpleysov (Дата обращения: 17.05.2025)

2. Дегтев Г.В., Сергеева С.А., Абдалова Т.Ю., Поляков Р.Л. Совершенствование информационного пространства сферы закупок в части электронного документооборота как фактор повышения эффективности и прозрачности закупочной деятельности // Финансовые рынки и банки. 2021. № 2. С. 3 – 7. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/sovershenstvovanie-informatsionnogo-prostranstva-sfery-zakupok-v-chasti-elektronnogo-dokumentooborota-kak-faktor-povysheniya (Дата обращения: 17.05.2025)

3. Рахматуллина В.Р., Горшенин В.Ф. Особенности закупочной деятельности Российской промышленной корпорации на электронных торговых площадках // Вестник Челябинского государственного университета. 2018. № 8 (418). Экономические науки. Вып. 62. С. 143 – 150. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/osobennosti-zakupochnoy-deyatelnosti-rossiyskoy-promyshlennoy-korporatsii-na-elektronnyh-torgovyh-ploschadkah (Дата обращения: 17.05.2025)

4. UML - диаграмма вариантов использования (use case diagram) // https://habr.com [Электронный ресурс]. URL: https://habr.com/ru/articles/47940/ (дата обращения: 17.05.2025)

5. Чикилев Д.Д. Виды архитектуры приложений // Научный лидер. 2024. № 22 (172). [Электронный ресурс]. URL: https://scilead.ru/article/6673-vidi-arkhitekturi-prilozhenij (дата обращения: 05.06.2025)

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. Ньюмен С. От монолита к микросервисам. Эволюционные шаблоны для трансформации монолитной системы / Пер. с англ. – СПб.: БХВ-Петербург, 2021. – 266 с.

8. 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)

9. Уварова Л.А., Иванов Д.Ю. Моделирование процессов сбыта в системе «производитель – маркетплейс» // Вестник Пермского университета. Серия «Экономика». 2024. Т. 19, № 3. С. 284 – 299. [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/modelirovanie-protsessov-sbyta-v-sisteme-proizvoditel-marketpleys (Дата обращения: 17.05.2025)

10. 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)

11. Верников Г. Основы методологии IDEF1X // Проект «Корпоративный менеджмент» [Электронный ресурс]. URL: https://www.cfin.ru/vernikov/idef/idef1x.shtml/ (дата обращения: 03.06.2024)

12. Моделирование данных: обзор // https://habr.com [Электронный ресурс]. URL: https://habr.com/ru/articles/556790/ (дата обращения: 05.06.2025)

13. 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)

14. Преимущества и недостатки клиент-серверной модели // https://sky.pro [Электронный ресурс]. URL: https://sky.pro/wiki/gamedev/preimushestva-i-nedostatki-klient-servernoj-modeli-v-igrah/ (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/chto-takoe-diagramma-ganta-i-kak-ee-sdelat/ (Дата обращения: 05.06.2025)

Приложение А

(Справочное)

Глоссарий

B2B – модель торговых отношений, в которой субъектами торговли являются юридические лица.

B2C – модель торговых отношений, в котором субъектом-покупателем является конечный потребитель (физическое лицо), а продавцом юридическое лицо.

B2G – модель торговых отношений, при котором поставщиком является организация или предприниматель (юридическое лицо), а заказчиком – государство, орган государственной власти или муниципальное образование.

SaaS – бизнес-модель, при которой компания предоставляет доступ к контенту по подписке.

Маркетплейс – программная платформа, позволяющая совершать онлайн и оффлайн сделки. Подобное программное обеспечение включает в себя автоматизацию сделок, документооборот и централизованное управление процессом торговли.

B2B-маркетплейс (business-to-business marketplace) – программная платформа, на которой компании-продавцы размещают товары и коммерческие предложения, а компании-покупатели формируют заказы на оптовые партии. В отличие от розничных маркетплейсов (B2C-маркетплейсы), участники представляют не физических лиц, а организации с реквизитами (ИНН, КПП, полное наименование).

Контур RFQ (Request for Quotation) – модель организации торговли, при которой покупатель выбирает товар из каталога, формирует заказ, получает счёт на оплату и осуществляет расчёт с продавцом вне системы. Площадка не выполняет функции платёжного агрегатора и не подтверждает факт доставки – она автоматизирует информационный обмен и документооборот между сторонами сделки, предоставляя централизованный документооборот и централизованное управление процессом торговли.

Приложение Б

(обязательное)

Исходный код

Листинг Б.1 – auth-service/internal/service/auth_service.go

```
package service

import (
	"auth-service/internal/models"
	"auth-service/internal/repository"
	"errors"
	"fmt"
	"time"

	"github.com/golang-jwt/jwt/v5"
	"golang.org/x/crypto/bcrypt"
)

type authService struct {
	userRepo   *repository.UserRepository
	jwtSecret  []byte
	accessTTL  time.Duration
	refreshTTL time.Duration
}

func NewAuthService(userRepo *repository.UserRepository, jwtSecret
string) AuthService {
	return &authService{
		userRepo:   userRepo,
		jwtSecret:  []byte(jwtSecret),
		accessTTL:  15 * time.Minute,
		refreshTTL: 7 * 24 * time.Hour,
	}
}

type Claims struct {
	UserID   int64  `json:"user_id"`
	UserType string `json:"user_type"`
	jwt.RegisteredClaims
}

func (s *authService) Login(login, password string) (string,
string, error) {
	user, cred, err := s.userRepo.GetUserByLogin(login)
	if err != nil {
		return "", "", fmt.Errorf("invalid credentials")
	}

	if user.IsActive == 0 {
		return "", "", fmt.Errorf("user is not active")
	}

	err = bcrypt.CompareHashAndPassword([]byte(cred.PasswordHash),
[]byte(password))
	if err != nil {
		return "", "", fmt.Errorf("invalid credentials")
```

```
	}

	accessToken, err := s.generateAccessToken(user.ID,
user.UserType)
	if err != nil {
		return "", "", fmt.Errorf("failed to generate access
token: %w", err)
	}

	refreshToken, err := s.generateRefreshToken(user.ID,
user.UserType)
	if err != nil {
		return "", "", fmt.Errorf("failed to generate refresh
token: %w", err)
	}

	return accessToken, refreshToken, nil
}

func (s *authService) CreateUser(user *models.User, login,
password, phoneNumber string) (int64, error) {
	hashedPassword, err :=
bcrypt.GenerateFromPassword([]byte(password), bcrypt.DefaultCost)
	if err != nil {
		return 0, fmt.Errorf("failed to hash password: %w", err)
	}
	return s.userRepo.CreateUser(user, login,
string(hashedPassword), phoneNumber)
}

func (s *authService) ChangePassword(userID int64, oldPassword,
newPassword string) error {
	cred, err := s.userRepo.GetCredentialsByUserID(userID)
	if err != nil {
		return fmt.Errorf("credentials not found")
	}

	err = bcrypt.CompareHashAndPassword([]byte(cred.PasswordHash),
[]byte(oldPassword))
	if err != nil {
		return fmt.Errorf("invalid old password")
	}

	hashedPassword, err :=
bcrypt.GenerateFromPassword([]byte(newPassword),
bcrypt.DefaultCost)
	if err != nil {
		return fmt.Errorf("failed to hash password: %w", err)
	}

	return s.userRepo.UpdatePassword(userID,
string(hashedPassword))
```

```
}

func (s *authService) RefreshToken(refreshToken string) (string,
string, error) {
     claims, err := s.validateToken(refreshToken)
     if err != nil {
          return "", "", fmt.Errorf("invalid refresh token: %w",
err)
     }

     user, err := s.userRepo.GetUserByID(claims.UserID)
     if err != nil {
          return "", "", fmt.Errorf("user not found")
     }

     if user.IsActive == 0 {
          return "", "", fmt.Errorf("user is not active")
     }

     accessToken, err := s.generateAccessToken(user.ID,
user.UserType)
     if err != nil {
          return "", "", fmt.Errorf("failed to generate access
token: %w", err)
     }

     newRefreshToken, err := s.generateRefreshToken(user.ID,
user.UserType)
     if err != nil {
          return "", "", fmt.Errorf("failed to generate refresh
token: %w", err)
     }

     return accessToken, newRefreshToken, nil
}

func (s *authService) ValidateToken(token string) (*Claims, error)
{
     return s.validateToken(token)
}

func (s *authService) GetUserByID(userID int64) (*models.User,
error) {
     return s.userRepo.GetUserByID(userID)
}

func (s *authService) validateToken(tokenString string) (*Claims,
error) {
     token, err := jwt.ParseWithClaims(tokenString, &Claims{},
func(token *jwt.Token) (interface{}, error) {
          if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
```

```
                return nil, fmt.Errorf("unexpected signing method:
%v", token.Header["alg"])
          }
          return s.jwtSecret, nil
     })

     if err != nil {
          return nil, err
     }

     claims, ok := token.Claims.(*Claims)
     if !ok || !token.Valid {
          return nil, errors.New("invalid token")
     }

     return claims, nil
}

func (s *authService) generateAccessToken(userID int64, userType
string) (string, error) {
     claims := &Claims{
          UserID:   userID,
          UserType: userType,
          RegisteredClaims: jwt.RegisteredClaims{
               ExpiresAt:
jwt.NewNumericDate(time.Now().Add(s.accessTTL)),
               IssuedAt:  jwt.NewNumericDate(time.Now()),
          },
     }
     token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
     return token.SignedString(s.jwtSecret)
}

func (s *authService) generateRefreshToken(userID int64, userType
string) (string, error) {
     claims := &Claims{
          UserID:   userID,
          UserType: userType,
          RegisteredClaims: jwt.RegisteredClaims{
               ExpiresAt:
jwt.NewNumericDate(time.Now().Add(s.refreshTTL)),
               IssuedAt:  jwt.NewNumericDate(time.Now()),
          },
     }
     token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
     return token.SignedString(s.jwtSecret)
}
```

Листинг Б.2 – auth-gateway/internal/middleware/auth_middleware.go

```
package middleware
import (
     "context"
```

```
     "net/http"
     "strconv"
     "strings"

     "github.com/golang-jwt/jwt/v5"
)

type AuthMiddleware struct {
     jwtSecret []byte
}

func NewAuthMiddleware(jwtSecret string) *AuthMiddleware {
     return &AuthMiddleware{
          jwtSecret: []byte(jwtSecret),
     }
}

type Claims struct {
     UserID   int64  `json:"user_id"`
     UserType string `json:"user_type"`
     jwt.RegisteredClaims
}

func (m *AuthMiddleware) RequireAuth(next http.Handler)
http.Handler {
     return http.HandlerFunc(func(w http.ResponseWriter, r
*http.Request) {
          authHeader := r.Header.Get("Authorization")
          if authHeader == "" {
               http.Error(w, "Authorization header required",
http.StatusUnauthorized)
               return
          }
          parts := strings.Split(authHeader, " ")
          if len(parts) != 2 || parts[0] != "Bearer" {
               http.Error(w, "Invalid authorization header format",
http.StatusUnauthorized)
               return
          }
          tokenString := parts[1]
          token, err := jwt.ParseWithClaims(tokenString, &Claims{},
func(token *jwt.Token) (interface{}, error) {
               if _, ok := token.Method.(*jwt.SigningMethodHMAC);
!ok {
                    return nil, jwt.ErrSignatureInvalid
               }
               return m.jwtSecret, nil
          })
          if err != nil || !token.Valid {
               http.Error(w, "Invalid token",
http.StatusUnauthorized)
               return
```

```
          }
          claims, ok := token.Claims.(*Claims)
          if !ok {
               http.Error(w, "Invalid token claims",
http.StatusUnauthorized)
               return
          }
          ctx := context.WithValue(r.Context(), "user_id",
claims.UserID)
          ctx = context.WithValue(ctx, "user_type",
claims.UserType)
          r = r.WithContext(ctx)
          r.Header.Set("X-User-ID",
strconv.FormatInt(claims.UserID, 10))
          r.Header.Set("X-User-Type", claims.UserType)
          next.ServeHTTP(w, r)
     })
}
```

Листинг Б.3 – order-service/internal/worker/outbox_worker.go

```
package worker
import (
     "log"
     "time"

     "order-service/internal/service"
)

func StartOutboxWorker(orderService service.OrderService, interval
time.Duration) {
     go func() {
          ticker := time.NewTicker(interval)
          defer ticker.Stop()
          for range ticker.C {
               if err := orderService.ProcessOutbox(20); err != nil
{
                    log.Printf("outbox worker error: %v", err)
               }
          }
     }()
}
```

Листинг Б.4 – billing-service/internal/service/billing_service.go

```
package service
import (
     "bytes"
     "context"
     "database/sql"
     "encoding/json"
     "fmt"
     "io"
```

```
	"log"
	"net/http"
	"time"

	"b2b-marketplace/pkg/billing"
	"b2b-marketplace/pkg/httperr"

	"billing-service/internal/repository"
)

type BillingService struct {
	repo               *repository.BillingRepository
	documentServiceURL string
	internalToken      string
}

func NewBillingService(repo *repository.BillingRepository,
documentURL, internalToken string) *BillingService {
	return &BillingService{repo: repo, documentServiceURL:
documentURL, internalToken: internalToken}
}

func (s *BillingService) EnsureTrial(sellerUserID int64) error {
	sub, err := s.repo.GetSubscription(sellerUserID)
	if err != nil {
		return err
	}
	if sub != nil {
		return nil
	}
	return s.repo.CreateTrialSubscription(sellerUserID)
}

func (s *BillingService) ProcessDealClosed(ev
billing.DealClosedEvent) error {
	if err := s.EnsureTrial(ev.SellerID); err != nil {
		return err
	}
	sub, err := s.repo.GetSubscription(ev.SellerID)
	if err != nil || sub == nil {
		return httperr.NewValidation("no subscription")
	}
	if sub.Status != "trial" && sub.Status != "active" {
		return httperr.NewValidation("subscription not active")
	}
	closedAt, err := time.Parse(time.RFC3339, ev.ClosedAt)
	if err != nil {
		closedAt = time.Now().UTC()
	}
	year, month := repository.MonthKey(closedAt)
	raw := ev.GrossAmount * float64(sub.Plan.CommissionRateBps) /
10000
```

```
	used, err := s.repo.SumCappedCommissionForMonth(ev.SellerID,
year, month)
	if err != nil {
		return err
	}
	cap := repository.CapValue(sub.Plan)
	remaining := cap - used
	if remaining < 0 {
		remaining = 0
	}
	capped := raw
	if capped > remaining {
		capped = remaining
	}
	return s.repo.InsertAccrual(ev.OrderItemID, ev.OrderID,
ev.SellerID, ev.BuyerID, ev.GrossAmount,
		sub.Plan.CommissionRateBps, raw, capped, sub.Plan.Code,
closedAt)
}

func (s *BillingService) GetUsage(sellerUserID int64)
(map[string]interface{}, error) {
	sub, err := s.GetSubscription(sellerUserID)
	if err != nil || sub == nil {
		return nil, err
	}
	now := time.Now()
	used, err := s.repo.SumCappedCommissionForMonth(sellerUserID,
now.Year(), int(now.Month()))
	if err != nil {
		return nil, err
	}
	cap := repository.CapValue(sub.Plan)
	remaining := cap - used
	if remaining < 0 {
		remaining = 0
	}
	return map[string]interface{}{
		"month":                now.Format("2006-01"),
		"commission_accrued":   used,
		"commission_cap":       cap,
		"commission_remaining": remaining,
		"cap_reached":          used >= cap,
	}, nil
}

func (s *BillingService) GenerateStatement(sellerUserID int64,
year, month int, force, runNow bool) (*GenerateStatementResult,
error) {
	if sellerUserID == 0 {
		sellers, err := s.repo.ListSellersWithAccruals(year,
month)
```

```
          if err != nil {
               return nil, err
          }
          var last *GenerateStatementResult
          for _, sid := range sellers {
               res, err := s.generateStatementForSeller(sid, year,
month, force, runNow)
               if err == nil {
                    last = res
               }
          }
          if last == nil {
               return nil, httperr.NewValidation("no accruals for
period")
          }
          return last, nil
     }
     return s.generateStatementForSeller(sellerUserID, year, month,
force, runNow)
}

func (s *BillingService) GenerateSaaSInvoicesForPeriod(year, month
int, runNow bool) error {
     subs, err := s.repo.ListActiveSubscriptions()
     if err != nil {
          return err
     }
     for _, sub := range subs {
          if sub.Plan.Code == "trial" {
               continue
          }
          periodStart := time.Date(year, time.Month(month), 1, 0,
0, 0, 0, time.UTC)
          periodEnd := periodStart.AddDate(0, 1, 0)
          invoiceID, err := s.repo.CreateSubscriptionInvoice(
               sub.SellerUserID, sub.Plan.ID, periodStart,
periodEnd, sub.Plan.MonthlyPriceRub,
          )
          if err != nil {
               log.Printf("saas-invoice: skip seller %d: %v",
sub.SellerUserID, err)
               continue
          }
          if runNow {
               docID, err := s.postSaaSInvoiceDocument(invoiceID,
sub.SellerUserID, sub.Plan, periodStart, periodEnd)
               if err != nil {
                    log.Printf("saas-invoice: doc gen failed for
seller %d: %v", sub.SellerUserID, err)
                    continue
               }
               if docID > 0 {
```

```
                    _ = s.repo.UpdateInvoiceDocument(invoiceID,
docID)
               }
          } else {
               payload, _ := json.Marshal(map[string]interface{}{
                    "invoice_id":     invoiceID,
                    "seller_user_id": sub.SellerUserID,
                    "plan_code":      sub.Plan.Code,
                    "plan_name":      sub.Plan.Name,
                    "amount_rub":     sub.Plan.MonthlyPriceRub,
                    "period_start":   periodStart.Format("2006-01-
02"),
                    "period_end":     periodEnd.Format("2006-01-
02"),
               })
               _ = s.repo.InsertOutbox("GenerateSaaSInvoice",
                    fmt.Sprintf("saas:%d:%s", sub.SellerUserID,
periodStart.Format("2006-01-02")), payload)
          }
     }
     return nil
}

func (s *BillingService) IsSubscriptionActive(sellerUserID int64)
(bool, error) {
     sub, err := s.GetSubscription(sellerUserID)
     if err != nil || sub == nil {
          return false, err
     }
     return sub.Status == "trial" || sub.Status == "active", nil
}

func (s *BillingService) GetStatement(id int64) (sellerUserID
int64, year, month int, documentID sql.NullInt64, err error) {
     return s.repo.GetStatement(id)
}
```

Листинг Б.5 – claim-service/internal/service/claim_service.go

```
package service
import (
     "claim-service/internal/models"
     "claim-service/internal/repository"
)

type ClaimServiceImpl struct {
     claimRepo *repository.ClaimRepository
}

func NewClaimService(claimRepo *repository.ClaimRepository)
ClaimService {
     return &ClaimServiceImpl{claimRepo: claimRepo}
}
```

```
func (s *ClaimServiceImpl) CreateClaim(userID, orderItemID int64,
description string) (int64, error) {
     return s.claimRepo.CreateClaim(userID, orderItemID,
description)
}

func (s *ClaimServiceImpl) GetClaim(claimID int64) (*models.Claim,
error) {
     return s.claimRepo.GetClaim(claimID)
}

func (s *ClaimServiceImpl) GetClaims(userID int64, limit, offset
int) ([]models.Claim, int, error) {
     return s.claimRepo.GetClaims(userID, limit, offset)
}

func (s *ClaimServiceImpl) CreateResolution(claimID, moderatorID
int64, resolutionText string) (int64, error) {
     return s.claimRepo.CreateResolution(claimID, moderatorID,
resolutionText)
}

func (s *ClaimServiceImpl) HasOpenClaim(orderItemID int64) (bool,
error) {
     return s.claimRepo.HasOpenClaim(orderItemID)
}
```

Листинг Б.6 – analytics-service/internal/service/analytics_service.go

```
package service
import (
     "analytics-service/internal/models"
     "analytics-service/internal/repository"
     "context"
     "fmt"
     "math"
     "time"

     "b2b-marketplace/pkg/category"
     "b2b-marketplace/pkg/offer"
     "b2b-marketplace/pkg/pagination"
)

type AnalyticsService struct {
     repo               *repository.AnalyticsRepository
     bayesianRegression *BayesianLinearRegression
     categoryClient     category.Client
     offerClient        offer.Client
}

func NewAnalyticsService(
     repo *repository.AnalyticsRepository,
```

```
     categoryClient category.Client,
     offerClient offer.Client,
) *AnalyticsService {
     return &AnalyticsService{
          repo:               repo,
          bayesianRegression: NewBayesianLinearRegression(),
          categoryClient:     categoryClient,
          offerClient:        offerClient,
     }
}

func (s *AnalyticsService) UpdateAnalyticsData(days int) error {
     ctx, cancel := context.WithTimeout(context.Background(),
30*time.Second)
     defer cancel()

     categories, err := s.categoryClient.GetAllCategories(ctx)
     if err != nil {
          return fmt.Errorf("failed to get categories: %w", err)
     }
     today := todayUTC()
     for _, cat := range categories {
          products, _, err := s.offerClient.SearchProducts(ctx,
&pagination.Params{Limit: 1000, Offset: 0},
map[string]interface{}{
               "category_id": cat.ID,
          })
          if err != nil {
               fmt.Printf("Warning: failed to get products for
category %d: %v\n", cat.ID, err)
               continue
          }
          if len(products) == 0 {
               continue
          }
          var sum, minP, maxP float64
          var totalQty int
          minP = math.MaxFloat64
          for _, p := range products {
               sum += p.Price
               totalQty += p.Quantity
               if p.Price < minP {
                    minP = p.Price
               }
               if p.Price > maxP {
                    maxP = p.Price
               }
          }
          avgPrice := sum / float64(len(products))
          analytics := &models.CategoryAnalytics{
               CategoryID:        cat.ID,
               Date:              today,
```

```
                AvgPrice:          avgPrice,
                MinPrice:          minP,
                MaxPrice:          maxP,
                TotalQuantitySold: totalQty,
                TotalOrders:       0,
                TotalProducts:     len(products),
            }
            if err := s.repo.UpsertCategoryAnalytics(analytics); err
!= nil {
                return fmt.Errorf("failed to upsert analytics for
category %d: %w", cat.ID, err)
            }
        }
        return nil
}

func (s *AnalyticsService) GenerateForecasts(categoryID int64,
forecastDays int) error {
        endDate := time.Now()
        startDate := endDate.AddDate(0, 0, -90)
        analytics, err := s.repo.GetCategoryAnalytics(categoryID,
startDate, endDate)
        if err != nil {
            return fmt.Errorf("failed to get category analytics: %w",
err)
        }
        analytics = s.expandAnalyticsForForecast(categoryID,
analytics)
        if len(analytics) < 2 {
            return fmt.Errorf("недостаточно данных для генерации
прогнозов для категории %d", categoryID)
        }
        dates := make([]time.Time, len(analytics))
        prices := make([]float64, len(analytics))
        demand := make([]float64, len(analytics))
        for i, a := range analytics {
            dates[i] = a.Date
            prices[i] = a.AvgPrice
            demand[i] = float64(a.TotalQuantitySold)
        }
        priceForecasts, err :=
s.bayesianRegression.PredictPriceForecast(prices, dates,
forecastDays)
        if err != nil {
            return fmt.Errorf("failed to predict prices: %w", err)
        }
        demandForecasts, err :=
s.bayesianRegression.PredictDemandForecast(demand, dates,
forecastDays)
        if err != nil {
            return fmt.Errorf("failed to predict demand: %w", err)
        }
```

```
	lastHist := calendarDateUTC(dates[len(dates)-1])
	today := todayUTC()
	forecastStart := lastHist.AddDate(0, 0, 1)
	if forecastStart.Before(today) {
		forecastStart = today
	}
	for i, pf := range priceForecasts {
		forecastDate := forecastStart.AddDate(0, 0, i)
		forecast := &models.PriceForecast{
			CategoryID:     categoryID,
			ForecastDate:   forecastDate,
			PredictedPrice: pf.PredictedValue,
			ConfidenceLow:  pf.ConfidenceLow,
			ConfidenceHigh: pf.ConfidenceHigh,
			ModelVersion:   "v1.0",
		}
		if err := s.repo.SavePriceForecast(forecast); err != nil
{
			return fmt.Errorf("failed to save price forecast:
%w", err)
		}
	}
	for i, df := range demandForecasts {
		forecastDate := forecastStart.AddDate(0, 0, i)
		forecast := &models.DemandForecast{
			CategoryID:      categoryID,
			ForecastDate:    forecastDate,
			PredictedDemand: df.PredictedValue,
			ConfidenceLow:   df.ConfidenceLow,
			ConfidenceHigh:  df.ConfidenceHigh,
			ModelVersion:    "v1.0",
		}
		if err := s.repo.SaveDemandForecast(forecast); err != nil
{
			return fmt.Errorf("failed to save demand forecast:
%w", err)
		}
	}
	return nil
}
```

Листинг Б.7 – analytics-service/internal/service/bayesian_regression.go

```
package service
import (
	"fmt"
	"math"
	"time"
)

type BayesianLinearRegression struct {
	PriorMean     float64
	PriorVariance float64
```

```
	NoiseVariance float64
}

func NewBayesianLinearRegression() *BayesianLinearRegression {
	return &BayesianLinearRegression{
		PriorMean:     0.0,
		PriorVariance: 1.0,
		NoiseVariance: 1.0,
	}
}

type ForecastResult struct {
	PredictedValue float64
	ConfidenceLow  float64
	ConfidenceHigh float64
}

func linearRegression(x, y []float64) (alpha, beta float64) {
	if len(x) != len(y) || len(x) == 0 {
		return 0, 0
	}
	meanX := mean(x)
	meanY := mean(y)
	var numerator float64
	var denominator float64
	for i := 0; i < len(x); i++ {
		dx := x[i] - meanX
		dy := y[i] - meanY
		numerator += dx * dy
		denominator += dx * dx
	}
	if denominator == 0 {
		beta = 0
	} else {
		beta = numerator / denominator
	}
	alpha = meanY - beta*meanX
	return alpha, beta
}

func mean(values []float64) float64 {
	if len(values) == 0 {
		return 0
	}
	var sum float64
	for _, v := range values {
		sum += v
	}
	return sum / float64(len(values))
}

func variance(values []float64, mean float64) float64 {
```

```
     if len(values) <= 1 {
          return 0
     }
     var sumSqDiff float64
     for _, v := range values {
          diff := v - mean
          sumSqDiff += diff * diff
     }
     return sumSqDiff / float64(len(values)-1)
}

func (blr *BayesianLinearRegression) PredictPriceForecast(
     historicalPrices []float64,
     historicalDates []time.Time,
     forecastDays int,
) ([]ForecastResult, error) {
     if len(historicalPrices) < 2 {
          return nil, fmt.Errorf("недостаточно исторических данных
для прогноза")
     }
     if len(historicalPrices) != len(historicalDates) {
          return nil, fmt.Errorf("несоответствие размеров массивов
данных")
     }
     startDate := historicalDates[0]
     features := make([]FeatureVector, len(historicalDates))
     for i, date := range historicalDates {
          features[i] = extractFeatures(date, startDate)
     }
     coefficients, err := multipleLinearRegression(features,
historicalPrices)
     if err != nil {
          return nil, fmt.Errorf("ошибка множественной регрессии:
%w", err)
     }
     var residualSumSq float64
     for i := range features {
          predicted := predictWithMultipleRegression(features[i],
coefficients)
          residual := historicalPrices[i] - predicted
          residualSumSq += residual * residual
     }
     n := float64(len(historicalPrices))
     numFeatures := 5.0
     var sampleVariance float64
     if n > numFeatures+1 {
          sampleVariance = residualSumSq / (n - numFeatures - 1)
     } else {
          sampleVariance = residualSumSq / n
     }
```

```
     posteriorVariance :=
blr.computePosteriorVarianceMultiple(sampleVariance, n,
numFeatures)
     results := make([]ForecastResult, forecastDays)
     lastDate := historicalDates[len(historicalDates)-1]
     for i := 0; i < forecastDays; i++ {
          forecastDate := lastDate.AddDate(0, 0, i+1)
          forecastFeatures := extractFeatures(forecastDate,
startDate)
          predicted :=
predictWithMultipleRegression(forecastFeatures, coefficients)
          stdErr := math.Sqrt(posteriorVariance * (1.0 + 1.0/n))
          confidenceInterval := 1.96 * stdErr
          results[i] = ForecastResult{
               PredictedValue: math.Max(0, predicted),
               ConfidenceLow:  math.Max(0, predicted-
confidenceInterval),
               ConfidenceHigh: predicted + confidenceInterval,
          }
     }
     return results, nil
}

func (blr *BayesianLinearRegression)
computePosteriorVarianceMultiple(sampleVariance, n, numFeatures
float64) float64 {
     if n == 0 {
          return blr.PriorVariance
     }
     if sampleVariance < 1e-10 {
          sampleVariance = 1e-10
     }
     priorWeight := 1.0 / blr.PriorVariance
     dataWeight := n / sampleVariance
     posteriorVariance := 1.0 / (priorWeight + dataWeight)
     posteriorVariance *= (1.0 + (numFeatures+1.0)/n)
     if posteriorVariance < 0 {
          posteriorVariance = sampleVariance
     }
     return posteriorVariance
}
```

Листинг Б.8 – migration-service/internal/migrator/migrator.go

```
package migrator
import (
     "database/sql"
     "errors"
     "fmt"
     "log"
     "os"
     "path/filepath"
     "sort"
```

```
     "strconv"
     "strings"

     "github.com/golang-migrate/migrate/v4"
     _ "github.com/golang-migrate/migrate/v4/database/postgres"
     _ "github.com/golang-migrate/migrate/v4/source/file"
)

var serviceDatabases = []string{
     "b2b_auth", "b2b_cart", "b2b_order", "b2b_offer",
     "b2b_invoice", "b2b_claim", "b2b_category",
     "b2b_analytics", "b2b_billing", "b2b_document",
}

type Config struct {
     Host           string
     Port           string
     User           string
     Password       string
     MigrationsPath string
     AdminDatabase  string
}

func Run(cfg Config) error {
     if cfg.AdminDatabase == "" {
          cfg.AdminDatabase = "postgres"
     }
     adminDSN := fmt.Sprintf(
          "host=%s port=%s user=%s password=%s dbname=%s
sslmode=disable",
          cfg.Host, cfg.Port, cfg.User, cfg.Password,
cfg.AdminDatabase,
     )
     adminDB, err := sql.Open("postgres", adminDSN)
     if err != nil {
          return fmt.Errorf("open admin connection: %w", err)
     }
     defer adminDB.Close()

     if err := adminDB.Ping(); err != nil {
          return fmt.Errorf("ping postgres: %w", err)
     }
     for _, dbName := range serviceDatabases {
          if err := ensureDatabase(adminDB, dbName); err != nil {
               return err
          }
          if err := migrateDatabase(cfg, dbName); err != nil {
               return err
          }
     }
     log.Println("All migrations applied successfully")
     return nil
```

```
}

func ensureDatabase(adminDB *sql.DB, dbName string) error {
     var exists int
     err := adminDB.QueryRow(
          "SELECT 1 FROM pg_database WHERE datname = $1", dbName,
     ).Scan(&exists)
     if err == nil {
          log.Printf("database %s already exists", dbName)
          return nil
     }
     if !errors.Is(err, sql.ErrNoRows) {
          return fmt.Errorf("check database %s: %w", dbName, err)
     }
     log.Printf("creating database %s", dbName)
     if _, err := adminDB.Exec(fmt.Sprintf("CREATE DATABASE %q",
dbName)); err != nil {
          return fmt.Errorf("create database %s: %w", dbName, err)
     }
     return nil
}

func migrateDatabase(cfg Config, dbName string) error {
     migrationsDir := filepath.Join(cfg.MigrationsPath, dbName)
     if _, err := os.Stat(migrationsDir); err != nil {
          return fmt.Errorf("migrations for %s: %w", dbName, err)
     }
     sourceURL := fmt.Sprintf("file://%s",
filepath.ToSlash(migrationsDir))
     targetURL := fmt.Sprintf(
          "postgres://%s:%s@%s:%s/%s?sslmode=disable",
          cfg.User, cfg.Password, cfg.Host, cfg.Port, dbName,
     )
     m, err := migrate.New(sourceURL, targetURL)
     if err != nil {
          return fmt.Errorf("init migrator for %s: %w", dbName,
err)
     }
     defer func() {
          srcErr, dbErr := m.Close()
          if srcErr != nil {
               log.Printf("close source for %s: %v", dbName,
srcErr)
          }
          if dbErr != nil {
               log.Printf("close database for %s: %v", dbName,
dbErr)
          }
     }()

     if version, dirty, verErr := m.Version(); verErr == nil &&
dirty {
```

```
          return fmt.Errorf(
               "database %s is in dirty state at version %d; fix
manually before retrying",
               dbName, version,
          )
     }
     targetVersion, err := latestVersion(migrationsDir)
     if err != nil {
          return err
     }
     schemaVersion, err := latestSchemaVersion(migrationsDir)
     if err != nil {
          return err
     }
     dbDSN := fmt.Sprintf(
          "host=%s port=%s user=%s password=%s dbname=%s
sslmode=disable",
          cfg.Host, cfg.Port, cfg.User, cfg.Password, dbName,
     )
     serviceDB, err := sql.Open("postgres", dbDSN)
     if err != nil {
          return fmt.Errorf("open service connection for %s: %w",
dbName, err)
     }
     defer serviceDB.Close()

     baseline, err := needsBaseline(serviceDB)
     if err != nil {
          return fmt.Errorf("baseline check for %s: %w", dbName,
err)
     }
     if baseline {
          log.Printf("baselining existing schema for %s at version
%d", dbName, schemaVersion)
          if err := m.Force(int(schemaVersion)); err != nil {
               return fmt.Errorf("baseline %s: %w", dbName, err)
          }
     }
     if err := m.Up(); err != nil {
          if errors.Is(err, migrate.ErrNoChange) {
               log.Printf("database %s is up to date (version %d)",
dbName, targetVersion)
               return nil
          }
          return fmt.Errorf("migrate up %s: %w", dbName, err)
     }
     version, dirty, err := m.Version()
     if err != nil && !errors.Is(err, migrate.ErrNilVersion) {
          return fmt.Errorf("read migration version for %s: %w",
dbName, err)
     }
     if dirty {
```

```
          return fmt.Errorf("database %s ended in dirty state at
version %d", dbName, version)
     }
     log.Printf("database %s migrated to version %d", dbName,
version)
     return nil
}

func needsBaseline(db *sql.DB) (bool, error) {
     var migrationTableExists bool
     if err := db.QueryRow(`
          SELECT EXISTS (
               SELECT 1 FROM information_schema.tables
               WHERE table_schema = 'public' AND table_name =
'schema_migrations'
          )`).Scan(&migrationTableExists); err != nil {
          return false, err
     }
     if migrationTableExists {
          var migrationRows int
          if err := db.QueryRow(`SELECT COUNT(*) FROM
schema_migrations`).Scan(&migrationRows); err != nil {
               return false, err
          }
          if migrationRows > 0 {
               return false, nil
          }
     }
     var appTables int
     if err := db.QueryRow(`
          SELECT COUNT(*) FROM information_schema.tables
          WHERE table_schema = 'public'
            AND table_type = 'BASE TABLE'
            AND table_name <>
'schema_migrations'`).Scan(&appTables); err != nil {
          return false, err
     }
     return appTables > 0, nil
}
```

Приложение В

(рекомендуемое)

SQL-скрипты

Листинг В.1 – Скрипты создания таблиц

```
CREATE TABLE user_types
(
    id   SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL UNIQUE
);

CREATE TABLE users
(
    id           SERIAL PRIMARY KEY,
    username     VARCHAR(25)  NOT NULL UNIQUE,
    kpp          VARCHAR(9)   NOT NULL,
    inn          VARCHAR(12)  NOT NULL,
    full_name    VARCHAR(100) NOT NULL,
    email        VARCHAR(255),
    created_at   DATE         NOT NULL,
    is_active    INT          NOT NULL,
    user_type_id INT          NOT NULL,
    CONSTRAINT fk_users_user_type
        FOREIGN KEY (user_type_id) REFERENCES user_types (id)
);

CREATE TABLE credentials
(
    id            SERIAL PRIMARY KEY,
    login         VARCHAR(50)  NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    user_id       INT          NOT NULL,
    phone_number  VARCHAR(12),
    CONSTRAINT fk_credentials_user
        FOREIGN KEY (user_id) REFERENCES users (id)
);

ALTER TABLE users ADD COLUMN IF NOT EXISTS email VARCHAR(255);

CREATE TABLE carts
(
    id      SERIAL PRIMARY KEY,
    user_id INT NOT NULL
);

CREATE TABLE cart_items
(
    id         SERIAL PRIMARY KEY,
    cart_id    INT     NOT NULL,
    quantity   INT     NOT NULL,
    product_id INT     NOT NULL,
    is_ordered BOOLEAN NOT NULL DEFAULT FALSE,
    CONSTRAINT fk_cart_items_cart FOREIGN KEY (cart_id) REFERENCES
carts (id)
);
```

```
CREATE INDEX idx_cart_items_is_ordered ON cart_items(cart_id,
is_ordered) WHERE is_ordered = FALSE;

CREATE TABLE categories
(
    id                 SERIAL PRIMARY KEY,
    name               VARCHAR(100) NOT NULL UNIQUE,
    created_by_user_id INT          NOT NULL
);

CREATE INDEX idx_categories_name ON categories(name);

CREATE TABLE products
(
    id          SERIAL PRIMARY KEY,
    user_id     INT            NOT NULL,
    name        VARCHAR(100)   NOT NULL,
    description TEXT,
    image_url   VARCHAR(255),
    price       NUMERIC(10, 2) NOT NULL,
    quantity    INT            NOT NULL,
    category_id INT,
    is_active   BOOLEAN        NOT NULL DEFAULT TRUE
);

CREATE INDEX idx_products_category_id ON products(category_id);

CREATE TABLE quote_requests
(
    id           SERIAL PRIMARY KEY,
    user_id      INT  NOT NULL,
    requested_at DATE NOT NULL
);

CREATE TABLE quote_request_cart_items
(
    quote_request_id INT NOT NULL,
    cart_item_id     INT NOT NULL,
    PRIMARY KEY (quote_request_id, cart_item_id)
);

CREATE TABLE offers
(
    id               SERIAL PRIMARY KEY,
    user_id          INT            NOT NULL,
    quote_request_id INT            NOT NULL,
    cart_item_id     INT            NOT NULL,
    price            NUMERIC(10, 2) NOT NULL,
    delivery_terms   VARCHAR(500)   NOT NULL,
    quantity         INT            NOT NULL,
    created_at       DATE           NOT NULL
```

```
);

CREATE TABLE quote_requests
(
    id           SERIAL PRIMARY KEY,
    user_id      INT  NOT NULL,
    requested_at DATE NOT NULL
);

CREATE TABLE quote_request_cart_items
(
    quote_request_id INT NOT NULL,
    cart_item_id     INT NOT NULL,
    PRIMARY KEY (quote_request_id, cart_item_id)
);

CREATE TABLE orders
(
    id         SERIAL PRIMARY KEY,
    user_id    INT  NOT NULL,
    order_date DATE NOT NULL
);

CREATE TABLE order_items
(
    id             SERIAL PRIMARY KEY,
    offer_id       INT            NOT NULL,
    order_id       INT            NOT NULL,
    seller_user_id INT            NOT NULL DEFAULT 0,
    unit_price     NUMERIC(12, 2) NOT NULL DEFAULT 0,
    quantity       INT            NOT NULL DEFAULT 0,
    deal_closed_at TIMESTAMPTZ,
    CONSTRAINT fk_order_items_order FOREIGN KEY (order_id)
REFERENCES orders (id)
);

CREATE TABLE status_types
(
    id   SERIAL PRIMARY KEY,
    name VARCHAR(30) NOT NULL UNIQUE
);

CREATE TABLE order_item_statuses
(
    status_date    DATE NOT NULL,
    status_type_id INT  NOT NULL,
    order_item_id  INT  NOT NULL,
    CONSTRAINT fk_order_item_statuses_type FOREIGN KEY
(status_type_id) REFERENCES status_types (id),
    CONSTRAINT fk_order_item_statuses_item FOREIGN KEY
(order_item_id)  REFERENCES order_items (id)
);
```

```
CREATE TABLE outbox_events
(
    id              SERIAL PRIMARY KEY,
    event_type      VARCHAR(64)  NOT NULL,
    payload         JSONB        NOT NULL,
    idempotency_key VARCHAR(255) NOT NULL UNIQUE,
    created_at      TIMESTAMPTZ  NOT NULL DEFAULT NOW(),
    processed_at    TIMESTAMPTZ
);

CREATE INDEX idx_outbox_unprocessed ON outbox_events (created_at)
WHERE processed_at IS NULL;

CREATE TABLE subscription_plans
(
    id                          SERIAL PRIMARY KEY,
    code                        VARCHAR(32)    NOT NULL UNIQUE,
    name                        VARCHAR(100)   NOT NULL,
    monthly_price_rub           NUMERIC(12, 2) NOT NULL DEFAULT 0,
    commission_rate_bps         INT            NOT NULL,
    monthly_commission_cap_rub NUMERIC(12, 2),
    billing_period              VARCHAR(16)    NOT NULL DEFAULT
'month',
    analytics_enabled           BOOLEAN        NOT NULL DEFAULT
FALSE,
    is_active                   BOOLEAN        NOT NULL DEFAULT
TRUE
);

CREATE TABLE seller_subscriptions
(
    id                    SERIAL PRIMARY KEY,
    seller_user_id        INT         NOT NULL UNIQUE,
    plan_id               INT         NOT NULL REFERENCES
subscription_plans (id),
    status                VARCHAR(32) NOT NULL,
    current_period_start DATE        NOT NULL,
    current_period_end   DATE        NOT NULL,
    trial_ends_at         DATE,
    created_at            TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at            TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE commission_accruals
(
    id                    SERIAL PRIMARY KEY,
    order_item_id         INT            NOT NULL UNIQUE,
    order_id              INT            NOT NULL,
    seller_user_id        INT            NOT NULL,
    buyer_user_id         INT            NOT NULL,
    gross_amount          NUMERIC(12, 2) NOT NULL,
```

```
    rate_bps              INT            NOT NULL,
    raw_commission        NUMERIC(12, 2) NOT NULL,
    capped_commission     NUMERIC(12, 2) NOT NULL,
    status                VARCHAR(32)    NOT NULL DEFAULT
'accrued',
    deal_closed_at        TIMESTAMPTZ    NOT NULL,
    plan_code_at_accrual VARCHAR(32)    NOT NULL,
    created_at            TIMESTAMPTZ    NOT NULL DEFAULT NOW()
);

CREATE TABLE commission_statements
(
    id                SERIAL PRIMARY KEY,
    seller_user_id    INT            NOT NULL,
    period_year       INT            NOT NULL,
    period_month      INT            NOT NULL,
    total_gross       NUMERIC(12, 2) NOT NULL DEFAULT 0,
    total_commission NUMERIC(12, 2) NOT NULL DEFAULT 0,
    status            VARCHAR(32)    NOT NULL DEFAULT 'draft',
    document_id       INT,
    idempotency_key   VARCHAR(255)   NOT NULL UNIQUE,
    created_at        TIMESTAMPTZ    NOT NULL DEFAULT NOW(),
    UNIQUE (seller_user_id, period_year, period_month)
);

CREATE TABLE subscription_invoices
(
    id               SERIAL PRIMARY KEY,
    seller_user_id   INT            NOT NULL,
    plan_id          INT            NOT NULL REFERENCES
subscription_plans (id),
    period_start     DATE           NOT NULL,
    period_end       DATE           NOT NULL,
    amount_rub       NUMERIC(12, 2) NOT NULL,
    status           VARCHAR(32)    NOT NULL DEFAULT 'draft',
    document_id      INT,
    paid_at          TIMESTAMPTZ,
    idempotency_key VARCHAR(255)   UNIQUE,
    created_at       TIMESTAMPTZ    NOT NULL DEFAULT NOW()
);

CREATE TABLE promotion_products
(
    id             SERIAL PRIMARY KEY,
    code           VARCHAR(32)    NOT NULL UNIQUE,
    name           VARCHAR(100)   NOT NULL,
    price_rub      NUMERIC(12, 2) NOT NULL,
    duration_days INT            NOT NULL
);

CREATE TABLE promotion_orders
(
```

```
    id              SERIAL PRIMARY KEY,
    seller_user_id INT         NOT NULL,
    product_id      INT         NOT NULL REFERENCES
promotion_products (id),
    starts_at       DATE        NOT NULL,
    ends_at         DATE        NOT NULL,
    status          VARCHAR(32) NOT NULL DEFAULT 'active',
    created_at      TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

CREATE TABLE billing_outbox_events
(
    id               SERIAL PRIMARY KEY,
    event_type       VARCHAR(64)  NOT NULL,
    payload          JSONB        NOT NULL,
    idempotency_key VARCHAR(255) NOT NULL UNIQUE,
    created_at       TIMESTAMPTZ  NOT NULL DEFAULT NOW(),
    processed_at     TIMESTAMPTZ
);

CREATE TABLE claims
(
    id             SERIAL PRIMARY KEY,
    user_id        INT           NOT NULL,
    order_item_id INT           NOT NULL,
    description    VARCHAR(1000) NOT NULL
);

CREATE TABLE claim_resolutions
(
    id               SERIAL PRIMARY KEY,
    resolution_text VARCHAR(1000) NOT NULL,
    resolution_date DATE          NOT NULL,
    claim_id         INT           NOT NULL,
    user_id          INT           NOT NULL,
    CONSTRAINT fk_claim_resolutions_claim FOREIGN KEY (claim_id)
REFERENCES claims (id)
);

CREATE TABLE category_analytics
(
    id                   SERIAL PRIMARY KEY,
    category_id          INT            NOT NULL,
    date                 DATE           NOT NULL,
    avg_price            NUMERIC(10, 2) NOT NULL,
    min_price            NUMERIC(10, 2) NOT NULL,
    max_price            NUMERIC(10, 2) NOT NULL,
    total_quantity_sold INT            NOT NULL DEFAULT 0,
    total_orders         INT            NOT NULL DEFAULT 0,
    total_products       INT            NOT NULL DEFAULT 0,
    created_at           TIMESTAMP      NOT NULL DEFAULT
CURRENT_TIMESTAMP,
```

```
    updated_at          TIMESTAMP      NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    CONSTRAINT uk_category_analytics_category_date UNIQUE
(category_id, date)
);

CREATE TABLE price_forecasts
(
    id              SERIAL PRIMARY KEY,
    category_id     INT            NOT NULL,
    forecast_date   DATE           NOT NULL,
    predicted_price NUMERIC(10, 2) NOT NULL,
    confidence_low  NUMERIC(10, 2) NOT NULL,
    confidence_high NUMERIC(10, 2) NOT NULL,
    model_version   VARCHAR(50)    NOT NULL DEFAULT 'v1.0',
    created_at      TIMESTAMP      NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    CONSTRAINT uk_price_forecasts_category_date UNIQUE
(category_id, forecast_date)
);

CREATE TABLE demand_forecasts
(
    id               SERIAL PRIMARY KEY,
    category_id      INT            NOT NULL,
    forecast_date    DATE           NOT NULL,
    predicted_demand NUMERIC(10, 2) NOT NULL,
    confidence_low   NUMERIC(10, 2) NOT NULL,
    confidence_high  NUMERIC(10, 2) NOT NULL,
    model_version    VARCHAR(50)    NOT NULL DEFAULT 'v1.0',
    created_at       TIMESTAMP      NOT NULL DEFAULT
CURRENT_TIMESTAMP,
    CONSTRAINT uk_demand_forecasts_category_date UNIQUE
(category_id, forecast_date)
);

CREATE INDEX idx_category_analytics_category_id ON
category_analytics(category_id);
CREATE INDEX idx_category_analytics_date        ON
category_analytics(date);
CREATE INDEX idx_price_forecasts_category_id    ON
price_forecasts(category_id);
CREATE INDEX idx_price_forecasts_date           ON
price_forecasts(forecast_date);
CREATE INDEX idx_demand_forecasts_category_id   ON
demand_forecasts(category_id);
CREATE INDEX idx_demand_forecasts_date          ON
demand_forecasts(forecast_date);

CREATE TABLE documents
(
    id              SERIAL PRIMARY KEY,
```

```
    doc_type         VARCHAR(32)  NOT NULL,
    seller_user_id   INT,
    buyer_user_id    INT,
    period_year      INT,
    period_month     INT,
    file_path        VARCHAR(500) NOT NULL,
    status           VARCHAR(32)  NOT NULL DEFAULT 'draft',
    idempotency_key VARCHAR(255) NOT NULL UNIQUE,
    created_at       TIMESTAMPTZ  NOT NULL DEFAULT NOW()
);

CREATE TABLE document_deliveries
(
    id          SERIAL PRIMARY KEY,
    document_id INT         NOT NULL REFERENCES documents (id),
    channel     VARCHAR(32) NOT NULL,
    status      VARCHAR(32) NOT NULL,
    sent_at     TIMESTAMPTZ,
    UNIQUE (document_id, channel)
);

CREATE TABLE invoices
(
    id           SERIAL PRIMARY KEY,
    order_id     INT            NOT NULL,
    amount       NUMERIC(10, 2) NOT NULL,
    invoice_date DATE           NOT NULL
);
```

Листинг В.2 – Скрипты наполнения базы данных

```
INSERT INTO user_types (id, name) VALUES (1, 'buyer')     ON
CONFLICT (name) DO NOTHING;
INSERT INTO user_types (id, name) VALUES (2, 'seller')    ON
CONFLICT (name) DO NOTHING;
INSERT INTO user_types (id, name) VALUES (3, 'moderator') ON
CONFLICT (name) DO NOTHING;

INSERT INTO users (username, kpp, inn, full_name, created_at,
is_active, user_type_id)
VALUES ('buyer1', '987654321', '987654321098', 'Покупатель
Петров', CURRENT_DATE, 1, 1)
ON CONFLICT (username) DO NOTHING;

INSERT INTO credentials (login, password_hash, user_id,
phone_number)
SELECT 'buyer1',
'$2a$10$q1PLOf9JiboKqmq.VZznOu8FiAnkvmEPtvDaF7Fg.fcVA8iwnWn.e',
u.id, '+79001234567'
FROM users u WHERE u.username = 'buyer1'
ON CONFLICT (login) DO UPDATE SET password_hash =
EXCLUDED.password_hash;
```

```
INSERT INTO users (username, kpp, inn, full_name, created_at,
is_active, user_type_id)
VALUES ('seller1', '111222333', '111222333444', 'Продавец Иванов',
CURRENT_DATE, 1, 2)
ON CONFLICT (username) DO NOTHING;

INSERT INTO credentials (login, password_hash, user_id,
phone_number)
SELECT 'seller1',
'$2a$10$q1PLOf9JiboKqmq.VZznOu8FiAnkvmEPtvDaF7Fg.fcVA8iwnWn.e',
u.id, '+79007654321'
FROM users u WHERE u.username = 'seller1'
ON CONFLICT (login) DO UPDATE SET password_hash =
EXCLUDED.password_hash;

INSERT INTO users (username, kpp, inn, full_name, created_at,
is_active, user_type_id)
VALUES ('moderator', '555666777', '555666777888', 'Модератор
Сидоров', CURRENT_DATE, 1, 3)
ON CONFLICT (username) DO NOTHING;

INSERT INTO credentials (login, password_hash, user_id,
phone_number)
SELECT 'moderator',
'$2a$10$q1PLOf9JiboKqmq.VZznOu8FiAnkvmEPtvDaF7Fg.fcVA8iwnWn.e',
u.id, '+79009876543'
FROM users u WHERE u.username = 'moderator'
ON CONFLICT (login) DO UPDATE SET password_hash =
EXCLUDED.password_hash;

INSERT INTO carts (id, user_id) VALUES (1, 1) ON CONFLICT DO
NOTHING;

SELECT setval(
    pg_get_serial_sequence('carts', 'id'),
    COALESCE((SELECT MAX(id) FROM carts), 1)
);

INSERT INTO categories (name, created_by_user_id)
VALUES
    ('Электроника',              2),
    ('Одежда',                   2),
    ('Продукты питания',         2),
    ('Строительные материалы',   2),
    ('Офисные принадлежности',   2)
ON CONFLICT (name) DO NOTHING;

INSERT INTO products (id, user_id, name, description, price,
quantity)
VALUES
    (1, 2, 'Ноутбук Lenovo',          'Ноутбук для бизнеса',
45000.00, 50),
```

```
    (2, 2, 'Мышь беспроводная',       'Беспроводная мышь',
1500.00, 200),
    (3, 2, 'Клавиатура механическая', 'Механическая клавиатура',
3500.00, 100),
    (4, 2, 'Монитор 27"',             'Монитор для офиса',
25000.00, 30),
    (5, 2, 'Кабель HDMI',             'HDMI кабель 2м',
500.00, 500)
ON CONFLICT DO NOTHING;

SELECT setval(
    pg_get_serial_sequence('products', 'id'),
    COALESCE((SELECT MAX(id) FROM products), 1)
);

INSERT INTO status_types (id, name) VALUES (1, 'В обработке')
ON CONFLICT (name) DO NOTHING;
INSERT INTO status_types (id, name) VALUES (2, 'Отправлен')
ON CONFLICT (name) DO NOTHING;
INSERT INTO status_types (id, name) VALUES (3, 'Доставлен')
ON CONFLICT (name) DO NOTHING;
INSERT INTO status_types (id, name) VALUES (4, 'Отменен')
ON CONFLICT (name) DO NOTHING;
INSERT INTO status_types (id, name) VALUES (5, 'Сделка закрыта')
ON CONFLICT (name) DO NOTHING;

SELECT setval(
    pg_get_serial_sequence('status_types', 'id'),
    COALESCE((SELECT MAX(id) FROM status_types), 1)
);

INSERT INTO subscription_plans
    (code, name, monthly_price_rub, commission_rate_bps,
monthly_commission_cap_rub, billing_period, analytics_enabled)
VALUES
    ('trial',   'Пробный',    0,     300, 30000,  'month', true),
    ('starter', 'Стартовый',  5000,  250, 50000,  'month', true),
    ('pro',     'Про',        15000, 150, 200000, 'month', true)
ON CONFLICT (code) DO NOTHING;

INSERT INTO promotion_products (code, name, price_rub,
duration_days)
VALUES
    ('top_week',       'Топ категории (7 дней)',  3000,  7),
    ('highlight_month','Выделение (30 дней)',     10000, 30)
ON CONFLICT (code) DO NOTHING;
```

Приложение Г

(рекомендуемое)

Объекты обращения к базам данных

Листинг Г.1 – Модели сервиса аутентификации (auth-service)

```
type User struct {
    ID         int64  `json:"id"`
    Username   string `json:"username"`
    KPP        string `json:"kpp"`
    INN        string `json:"inn"`
    FullName   string `json:"full_name"`
    Email      string `json:"email"`
    CreatedAt  string `json:"created_at"`
    IsActive   int    `json:"is_active"`
    UserTypeID int    `json:"user_type_id"`
    UserType   string `json:"user_type"`
}

type Credential struct {
    ID           int64  `json:"id"`
    Login        string `json:"login"`
    PasswordHash string `json:"password_hash"`
    UserID       int64  `json:"user_id"`
    PhoneNumber  string `json:"phone_number"`
}

type UserType struct {
    ID   int    `json:"id"`
    Name string `json:"name"`
}
```

Листинг Г.2 – Модель корзины (cart-service)

```
type Cart struct {
    ID    int64      `json:"id"`
    Items []CartItem `json:"items"`
}

type CartItem struct {
    ID                int64   `json:"id"`
    ProductID         int64   `json:"product_id"`
    Quantity          int     `json:"quantity"`
    ProductName       string  `json:"product_name"`
    Price             float64 `json:"price"`
    ImageURL          string  `json:"image_url,omitempty"`
    AvailableQuantity int     `json:"available_quantity"`
}
```

Листинг Г.3 – Модели товара и предложения (offer-service)

```
type Product struct {
    ID          int64   `json:"id"`
    UserID      int64   `json:"user_id"`
    Name        string  `json:"name"`
    Description string  `json:"description"`
    ImageURL    string  `json:"image_url,omitempty"`
```

```
    Price       float64 `json:"price"`
    Quantity    int     `json:"quantity"`
    IsActive    bool    `json:"is_active"`
    CategoryID  *int64  `json:"category_id,omitempty"`
}

type Offer struct {
    ID             int64   `json:"id"`
    UserID         int64   `json:"user_id"`
    QuoteRequestID int64   `json:"quote_request_id"`
    CartItemID     int64   `json:"cart_item_id"`
    Price          float64 `json:"price"`
    DeliveryTerms  string  `json:"delivery_terms"`
    Quantity       int     `json:"quantity"`
    CreatedAt      string  `json:"created_at"`
}
```

Листинг Г.4 – Модели заказа (order-service)

```
type Order struct {
    ID        int64       `json:"id"`
    OrderDate time.Time   `json:"order_date"`
    Items     []OrderItem `json:"items"`
}

type OrderItem struct {
    ID           int64      `json:"id"`
    OfferID      int64      `json:"offer_id"`
    Status       string     `json:"status"`
    Price        float64    `json:"price"`
    UnitPrice    float64    `json:"unit_price"`
    Quantity     int        `json:"quantity"`
    UserID       int64      `json:"user_id,omitempty"`
    SellerUserID int64      `json:"seller_user_id"`
    DealClosedAt *time.Time `json:"deal_closed_at,omitempty"`
    OrderID      int64      `json:"order_id,omitempty"`
    BuyerUserID  int64      `json:"buyer_user_id,omitempty"`
}

type OrderItemCloseInfo struct {
    OrderItemID  int64
    OrderID      int64
    BuyerUserID  int64
    SellerUserID int64
    UnitPrice    float64
    Quantity     int
    DealClosedAt *time.Time
}

type ItemCreateInput struct {
    OfferID      int64
    SellerUserID int64
    UnitPrice    float64
```

```
    Quantity     int
}
```

Листинг Г.5 – Модели биллинга (billing-service)

```
type Plan struct {
    ID                    int64           `json:"id"`
    Code                  string          `json:"code"`
    Name                  string          `json:"name"`
    MonthlyPriceRub       float64
`json:"monthly_price_rub"`
    CommissionRateBps     int
`json:"commission_rate_bps"`
    MonthlyCommissionCapRub sql.NullFloat64 `json:"-"`
    BillingPeriod         string
`json:"billing_period"`
    AnalyticsEnabled      bool
`json:"analytics_enabled"`
}

type Subscription struct {
    SellerUserID       int64
    Plan               Plan
    Status             string
    CurrentPeriodStart time.Time
    CurrentPeriodEnd   time.Time
    TrialEndsAt        sql.NullTime
}

type Accrual struct {
    ID               int64     `json:"id"`
    OrderItemID      int64     `json:"order_item_id"`
    OrderID          int64     `json:"order_id"`
    GrossAmount      float64   `json:"gross_amount"`
    RawCommission    float64   `json:"raw_commission"`
    CappedCommission float64   `json:"capped_commission"`
    Status           string    `json:"status"`
    DealClosedAt     time.Time `json:"deal_closed_at"`
}

type AccrualPeriod struct {
    Year            int     `json:"year"`
    Month           int     `json:"month"`
    Count           int     `json:"count"`
    TotalCommission float64 `json:"total_commission"`
    TotalGross      float64 `json:"total_gross"`
}

type SubscriptionInvoice struct {
    ID           int64         `json:"id"`
    SellerUserID int64         `json:"seller_user_id"`
    PlanID       int64         `json:"plan_id"`
    PeriodStart  time.Time     `json:"period_start"`
```

```
    PeriodEnd    time.Time     `json:"period_end"`
    AmountRub    float64       `json:"amount_rub"`
    Status       string        `json:"status"`
    DocumentID   sql.NullInt64 `json:"document_id"`
    PaidAt       sql.NullTime  `json:"paid_at"`
    CreatedAt    time.Time     `json:"created_at"`
}

type PromotionProduct struct {
    ID           int64   `json:"id"`
    Code         string  `json:"code"`
    Name         string  `json:"name"`
    PriceRub     float64 `json:"price_rub"`
    DurationDays int     `json:"duration_days"`
}
```

Листинг Г.6 – Модели претензий (claim-service)

```
type Claim struct {
    ID          int64            `json:"id"`
    UserID      int64            `json:"user_id"`
    OrderItemID int64            `json:"order_item_id"`
    Description string           `json:"description"`
    Resolution  *ClaimResolution `json:"resolution,omitempty"`
}

type ClaimResolution struct {
    ID             int64  `json:"id"`
    ResolutionText string `json:"resolution_text"`
    ResolutionDate string `json:"resolution_date"`
    ModeratorID    int64  `json:"moderator_id"`
}
```

Листинг Г.7 – Модели аналитики (analytics-service)

```
type CategoryAnalytics struct {
    ID                int64     `json:"id"`
    CategoryID        int64     `json:"category_id"`
    Date              time.Time `json:"date"`
    AvgPrice          float64   `json:"avg_price"`
    MinPrice          float64   `json:"min_price"`
    MaxPrice          float64   `json:"max_price"`
    TotalQuantitySold int       `json:"total_quantity_sold"`
    TotalOrders       int       `json:"total_orders"`
    TotalProducts     int       `json:"total_products"`
    CreatedAt         time.Time `json:"created_at"`
    UpdatedAt         time.Time `json:"updated_at"`
}

type PriceForecast struct {
    ID             int64     `json:"id"`
    CategoryID     int64     `json:"category_id"`
    ForecastDate   time.Time `json:"forecast_date"`
```

```
    PredictedPrice float64   `json:"predicted_price"`
    ConfidenceLow  float64   `json:"confidence_low"`
    ConfidenceHigh float64   `json:"confidence_high"`
    ModelVersion   string    `json:"model_version"`
    CreatedAt      time.Time `json:"created_at"`
}

type DemandForecast struct {
    ID              int64     `json:"id"`
    CategoryID      int64     `json:"category_id"`
    ForecastDate    time.Time `json:"forecast_date"`
    PredictedDemand float64   `json:"predicted_demand"`
    ConfidenceLow   float64   `json:"confidence_low"`
    ConfidenceHigh  float64   `json:"confidence_high"`
    ModelVersion    string    `json:"model_version"`
    CreatedAt       time.Time `json:"created_at"`
}

type ForecastCalendar struct {
    CategoryID      int64            `json:"category_id"`
    CategoryName    string           `json:"category_name"`
    PriceForecasts  []PriceForecast  `json:"price_forecasts"`
    DemandForecasts []DemandForecast `json:"demand_forecasts"`
    StartDate       time.Time        `json:"start_date"`
    EndDate         time.Time        `json:"end_date"`
}
```

Листинг Г.8 – Модели счёта-фактуры (invoice-service)

```
type Invoice struct {
    ID          int64   `json:"id"`
    OrderID     int64   `json:"order_id"`
    Amount      float64 `json:"amount"`
    InvoiceDate string  `json:"invoice_date"`
}

type InvoiceData struct {
    InvoiceID   int64
    OrderID     int64
    OrderDate   string
    InvoiceDate string
    Buyer       CompanyInfo
    Seller      CompanyInfo
    Items       []InvoiceItem
    TotalAmount float64
}

type CompanyInfo struct {
    FullName string
    INN      string
    KPP      string
}
```

```
type InvoiceItem struct {
    ProductName string
    Quantity    int
    Price       float64
    Total       float64
}
```

Листинг Г.9 – Модель категории (category-service)

```
type Category struct {
    ID                int64  `json:"id"`
    Name              string `json:"name"`
    CreatedByUserID   int64  `json:"created_by_user_id"`
}
```

Приложение Д

(рекомендуемое)

Исходный код модульного тестирования

Листинг Д.1 – Модели сервиса аутентификации (auth-service)

```
package service_test

import (
          "context"
          "errors"
          "strings"
          "testing"

          sqlmock "github.com/DATA-DOG/go-sqlmock"

          billingpkg "b2b-marketplace/pkg/billing"
          cartpkg "b2b-marketplace/pkg/cart"
          claimpkg "b2b-marketplace/pkg/claim"
          offerpkg "b2b-marketplace/pkg/offer"
          "b2b-marketplace/pkg/pagination"

          "order-service/internal/repository"
          "order-service/internal/service"
)

type mockCartClient struct {
          GetCartItemFn func(ctx context.Context, cartItemID
int64) (*cartpkg.CartItem, error)
}

func (m *mockCartClient) GetCart(ctx context.Context, userID
int64, params *pagination.Params) (*cartpkg.Cart, int, error) {
          return nil, 0, nil
}
func (m *mockCartClient) GetCartItem(ctx context.Context,
cartItemID int64) (*cartpkg.CartItem, error) {
          return m.GetCartItemFn(ctx, cartItemID)
}
func (m *mockCartClient) AddItem(ctx context.Context, userID,
productID int64, quantity int) error {
          return nil
}
func (m *mockCartClient) RemoveItem(ctx context.Context, userID,
cartItemID int64) error {
          return nil
}
func (m *mockCartClient) Checkout(ctx context.Context, userID
int64) (map[string]interface{}, error) {
          return nil, nil
}

type mockOfferClient struct {
          GetProductFn               func(ctx context.Context,
productID int64) (*offerpkg.Product, error)
```

```
        SearchOffersFn             func(ctx context.Context,
filters map[string]interface{}) ([]offerpkg.Offer, error)
        CreateOfferFn              func(ctx context.Context,
userID, quoteRequestID, cartItemID int64, price float64,
deliveryTerms string, quantity int) (int64, error)
        DecrementProductQuantityFn func(ctx context.Context,
productID int64, quantity int) error
}

func (m *mockOfferClient) GetProduct(ctx context.Context,
productID int64) (*offerpkg.Product, error) {
        return m.GetProductFn(ctx, productID)
}
func (m *mockOfferClient) SearchProducts(ctx context.Context,
params *pagination.Params, filters map[string]interface{})
([]offerpkg.Product, int, error) {
        return nil, 0, nil
}
func (m *mockOfferClient) SetProductActive(ctx context.Context,
userID, productID int64, isActive bool) error {
        return nil
}
func (m *mockOfferClient) DecrementProductQuantity(ctx
context.Context, productID int64, quantity int) error {
        if m.DecrementProductQuantityFn != nil {
            return m.DecrementProductQuantityFn(ctx, productID,
quantity)
        }
        return nil
}
func (m *mockOfferClient) GetOffer(ctx context.Context, offerID
int64) (*offerpkg.Offer, error) {
        return nil, nil
}
func (m *mockOfferClient) SearchOffers(ctx context.Context,
filters map[string]interface{}) ([]offerpkg.Offer, error) {
        return m.SearchOffersFn(ctx, filters)
}
func (m *mockOfferClient) CreateOffer(ctx context.Context, userID,
quoteRequestID, cartItemID int64, price float64, deliveryTerms
string, quantity int) (int64, error) {
        return m.CreateOfferFn(ctx, userID, quoteRequestID,
cartItemID, price, deliveryTerms, quantity)
}

type mockClaimClient struct{}

func (m *mockClaimClient) CreateClaim(ctx context.Context, userID,
orderItemID int64, description string) (*claimpkg.Claim, error) {
        return nil, nil
}
```

```
func (m *mockClaimClient) GetClaim(ctx context.Context, claimID
int64) (*claimpkg.Claim, error) {
          return nil, nil
}
func (m *mockClaimClient) GetClaims(ctx context.Context, userID
int64) ([]claimpkg.Claim, error) {
          return nil, nil
}
func (m *mockClaimClient) HasOpenClaim(ctx context.Context,
orderItemID int64, internalToken string) (bool, error) {
          return false, nil
}

type mockBillingClient struct{}

func (m *mockBillingClient) GetSubscription(ctx context.Context,
sellerUserID int64) (*billingpkg.SellerSubscription, error) {
          return nil, nil
}
func (m *mockBillingClient) GetSubscriptionInternal(ctx
context.Context, sellerUserID int64, internalToken string)
(*billingpkg.SellerSubscription, error) {
          return nil, nil
}
func (m *mockBillingClient) ProcessDealClosed(ctx context.Context,
event billingpkg.DealClosedEvent, internalToken string) error {
          return nil
}
func (m *mockBillingClient) CreateTrial(ctx context.Context,
sellerUserID int64, internalToken string) error {
          return nil
}

func expectQuoteRequest(mock sqlmock.Sqlmock, userID int64,
cartItemID int64, quoteRequestID int64) {
          mock.ExpectBegin()
          mock.ExpectQuery(`INSERT INTO quote_requests`).

          WillReturnRows(sqlmock.NewRows([]string{"id"}).AddRow(qu
oteRequestID))
          mock.ExpectExec(`INSERT INTO quote_request_cart_items`).
              WillReturnResult(sqlmock.NewResult(1, 1))
          mock.ExpectCommit()
}

func expectCreateOrder(mock sqlmock.Sqlmock, orderID int64) {
          mock.ExpectBegin()
          mock.ExpectQuery(`INSERT INTO orders`).

          WillReturnRows(sqlmock.NewRows([]string{"id"}).AddRow(or
derID))
          mock.ExpectQuery(`INSERT INTO order_items`).
```

```
            WillReturnRows(sqlmock.NewRows([]string{"id"}).AddRow(in
t64(1)))
            mock.ExpectExec(`INSERT INTO order_item_statuses`).
                WillReturnResult(sqlmock.NewResult(1, 1))
            mock.ExpectCommit()
}
func TestCreateOrder_ValidInput_ReturnsPositiveOrderID(t
*testing.T) {
            db, mock, err := sqlmock.New()
            if err != nil {
                t.Fatalf("failed to open sqlmock: %v", err)
            }
            defer db.Close()

            expectQuoteRequest(mock, 1, 7, 10)
            expectCreateOrder(mock, 42)

            cartC := &mockCartClient{
                GetCartItemFn: func(_ context.Context, id int64)
(*cartpkg.CartItem, error) {
                    return &cartpkg.CartItem{ID: id, ProductID:
100, Quantity: 5}, nil
                },
            }
            offerC := &mockOfferClient{
                GetProductFn: func(_ context.Context, productID
int64) (*offerpkg.Product, error) {
                    return &offerpkg.Product{ID: productID, UserID:
2, Price: 99.99, Quantity: 10}, nil
                },
                SearchOffersFn: func(_ context.Context, _
map[string]interface{}) ([]offerpkg.Offer, error) {
                    return []offerpkg.Offer{}, nil
                },
                CreateOfferFn: func(_ context.Context, _, _, _
int64, _ float64, _ string, _ int) (int64, error) {
                    return 55, nil
                },
            }

            svc := service.NewOrderService(
                repository.NewOrderRepository(db), cartC, offerC,
                &mockClaimClient{}, &mockBillingClient{}, "tok",
            )

            orderID, err := svc.CreateOrder(1, []int64{7})

            if err != nil {
                t.Fatalf("ожидался успех, получена ошибка: %v", err)
            }
            if orderID <= 0 {
```

```
            t.Errorf("ожидался order_id > 0, получено %d",
orderID)
        }
}

func TestCreateOrder_EmptyCartItems_ReturnsValidationError(t
*testing.T) {
        db, _, err := sqlmock.New()
        if err != nil {
            t.Fatalf("failed to open sqlmock: %v", err)
        }
        defer db.Close()

        svc := service.NewOrderService(
            repository.NewOrderRepository(db),
            &mockCartClient{}, &mockOfferClient{},
            &mockClaimClient{}, &mockBillingClient{}, "tok",
        )

        _, err = svc.CreateOrder(1, []int64{})

        if err == nil {
            t.Fatal("ожидалась ошибка валидации, получено nil")
        }
        if !strings.Contains(err.Error(), "validation error") {
            t.Errorf("ожидалась 'validation error', получено:
%v", err)
        }
}

func
TestCreateOrder_InsufficientStock_ReturnsNotEnoughQuantityError(t
*testing.T) {
        db, mock, err := sqlmock.New()
        if err != nil {
            t.Fatalf("failed to open sqlmock: %v", err)
        }
        defer db.Close()

        expectQuoteRequest(mock, 1, 7, 10)

        cartC := &mockCartClient{
            GetCartItemFn: func(_ context.Context, id int64)
(*cartpkg.CartItem, error) {
                return &cartpkg.CartItem{ID: id, ProductID:
100, Quantity: 20}, nil // запрошено 20
            },
        }
        offerC := &mockOfferClient{
            GetProductFn: func(_ context.Context, productID
int64) (*offerpkg.Product, error) {
```

```
                    return &offerpkg.Product{ID: productID,
Quantity: 5}, nil // в наличии только 5
                },
            }

            svc := service.NewOrderService(
                repository.NewOrderRepository(db), cartC, offerC,
                &mockClaimClient{}, &mockBillingClient{}, "tok",
            )

            _, err = svc.CreateOrder(1, []int64{7})

            if err == nil {
                t.Fatal("ожидалась ошибка, получено nil")
            }
            if !strings.Contains(err.Error(), "not enough quantity")
{
                t.Errorf("ожидалась ошибка 'not enough quantity',
получено: %v", err)
            }
}

func
TestCreateOrder_OfferServiceUnavailable_ReturnsFailedToCreateOffer
Error(t *testing.T) {
            db, mock, err := sqlmock.New()
            if err != nil {
                t.Fatalf("failed to open sqlmock: %v", err)
            }
            defer db.Close()

            expectQuoteRequest(mock, 1, 7, 10)

            cartC := &mockCartClient{
                GetCartItemFn: func(_ context.Context, id int64)
(*cartpkg.CartItem, error) {
                    return &cartpkg.CartItem{ID: id, ProductID:
100, Quantity: 5}, nil
                },
            }
            offerC := &mockOfferClient{
                GetProductFn: func(_ context.Context, productID
int64) (*offerpkg.Product, error) {
                    return &offerpkg.Product{ID: productID, UserID:
2, Price: 99.0, Quantity: 10}, nil
                },
                SearchOffersFn: func(_ context.Context, _
map[string]interface{}) ([]offerpkg.Offer, error) {
                    return []offerpkg.Offer{}, nil
                },
                CreateOfferFn: func(_ context.Context, _, _, _
int64, _ float64, _ string, _ int) (int64, error) {
```

```
                    return 0, errors.New("connection refused")
                },
            }

            svc := service.NewOrderService(
                repository.NewOrderRepository(db), cartC, offerC,
                &mockClaimClient{}, &mockBillingClient{}, "tok",
            )

            _, err = svc.CreateOrder(1, []int64{7})

            if err == nil {
                t.Fatal("ожидалась ошибка, получено nil")
            }
            if !strings.Contains(err.Error(), "failed to create
offer") {
                t.Errorf("ожидалась ошибка 'failed to create offer',
получено: %v", err)
            }
}
```