

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Программные средства персонализированного подбора и
бронирования санаторно-курортных услуг**

Студент

(подпись, дата)

Пилюгин Артём Эдуардович
(ФИО полностью)

Руководитель

(подпись, дата)

Балашова И.Ю.
(фамилия, инициалы)

Нормоконтролёр

(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

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

Заведующий кафедрой

(подпись)

Козлов А.Ю.
(фамилия, инициалы)

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

Секретарь ГЭК

(подпись)

Попова Н.А.
(фамилия, инициалы)

Пенза, 2026

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

Факультет

Вычислительной техники

Кафедра

МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Пилюгин Артём Эдуардович гр. 22ВП2 факультета ФВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Программные средства персонализированного подбора и бронирования санаторно-курортных услуг

Тема утверждена приказом ПГУ №1583/о от "30" декабря 2025г.

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

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

1) *Назначение: информационная система клиентского бронирования санаториев*

2) *Среда выполнения: Windows 11.*

Языковые средства разработки: Go, TypeScript, UML.

Инструментальные средства: Visual Studio Code, React, Docker, PostgreSQL, Swagger, NATS, Visual Paradigm 17.0.

Технологии разработки: микросервисная архитектура, многослойная архитектура, компонентный подход.

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) просмотр каталога санаториев;

в) создание, просмотр, изменение и отмена бронирований клиентов;

г) оформление и ведение договоров администратором;

д) выставление счетов и проведение оплаты;

е) мониторинг состояния сервисов;

ж) предоставление Swagger-документации.

5. Перечень подлежащих разработке вопросов:

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

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	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 _____ 26.12.2025г.
(подпись, дата)

Задание к исполнению принял студент _____ 26.12.2025г.
(подпись, дата)

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

Реферат

Пояснительная записка содержит 114 страниц, 29 рисунков, 19 таблиц, 20 использованных источников и 9 приложений.

САНАТОРНО-ГОСТИНИЧНЫЙ БИЗНЕС, КЛИЕНТСКОЕ БРОНИРОВАНИЕ, ИНФОРМАЦИОННАЯ СИСТЕМА, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, REST API, JWT, POSTGRESQL, NATS, GO, TYPESCRIPT, REACT, DOCKER, МОНИТОРИНГ ДАННЫХ, UML

Объект исследования – информационная система клиентского бронирования малого санаторного-гостиничного бизнеса.

Цель выпускной квалификационной работы – спроектировать и разработать программное обеспечение для автоматизации клиентского и служебного контуров бронирования санаторно-гостиничных услуг.

Технологии разработки – языки программирования Go и TypeScript; клиентская часть на библиотеке REACT со средством сборки Vite; серверная часть в архитектурном стиле микросервисов с REST API; аутентификация на основе JWT; реляционная СУБД PostgreSQL; брокер сообщений NATS для асинхронного обмена событиями; контейнеризация Docker и оркестрация Docker Compose; документирование API средствами Swagger; моделирование UML в Visual Paradigm; среда разработки Visual Studio Code.

Среда исполнения – Windows 11, веб-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы разработана информационная система клиентского бронирования с веб-интерфейсом для клиентов и администраторов, обеспечивающая подбор санатория, оформление бронирований, ведение договоров, проведение оплат и мониторинг состояния сервисов.

					ПГУ 09.03.04 – 08BP222.24 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Пилюгин А.Э.			Программные средства персонализированного подбора и бронирования санаторно-курортных услуг Пояснительная записка	Лит.	Лист	Листов
Пров.		Балашова И.Ю.					4	114
Реценз.						Группа 22ВП2		
Н. Контр.		Такташкин Д.В						
Утв.		Козлов А.Ю.						

Содержание

Введение.....	8
1 Анализ предметной области и постановка задачи на разработку программных средств персонализированного подбора и бронирования санаторно-курортных услуг	10
1.1 Постановка задачи на разработку программных средств	10
1.2 Анализ аналогов разрабатываемого приложения.....	12
1.3 Анализ нефункциональных требований	15
1.4 Анализ функциональных требований	16
1.5 Планирование разработки и оценка бюджета.....	21
2 Проектирование программных средств персонализированного подбора и бронирования санаторно-курортных услуг	23
2.1 Архитектура и состав системы	23
2.2 Проектирование структур данных	27
2.2.1 Концептуальная модель данных	27
2.2.2 Логическая модель данных.....	31
2.2.3 Проектирование физического уровня	33
2.3 Проектирование пользовательского интерфейса	35
2.4 Проектирование серверной части	35
2.4.1 Проектирование микросервиса аутентификации.....	36
2.4.2 Проектирование микросервиса бронирований и каталога.....	38
2.4.3 Проектирование микросервиса оплаты.....	42
2.5 Проектирование межсервисного взаимодействия.....	44
3 Реализация программных средств персонализированного подбора и бронирования санаторно-курортных услуг	48
3.1 Реализация базы данных	48
3.2 Реализация серверной части	51
3.3 Реализация клиентской части	52

3.4	Реализация пользовательского интерфейса веб-приложения персонализированного подбора и бронирования санаторно-курортных услуг	56
4	Тестирование программных средств персонализированного подбора и бронирования санаторно-курортных услуг	63
4.1	Обоснование выбора видов тестирования и метрик кода	63
4.2	Функциональное тестирование сценария создания бронирования	65
4.2.1	Причины и следствия	65
4.2.2	Функциональная диаграмма	66
4.2.3	Таблица решений	68
4.2.4	Тест-кейсы	69
4.3	Нагрузочное тестирование	71
4.4	Оценка качества программного кода	73
4.4.1	Метрики размера программы	74
4.4.2	Метрики Холстеда	74
4.4.3	Цикломатическая сложность	75
4.4.4	Метрика Чепина	76
4.4.5	Объектная метрика WMC	78
4.4.6	Автоматизированный подсчет	79
	Заключение	80
	Список использованных источников	82
	Приложение А (справочное) Глоссарий	85
	Приложение Б (рекомендуемое) Спецификация атрибутов таблиц	88
	Приложение В (обязательное) Доменные объекты	95
	Приложение Г (обязательное) SQL-скрипты	98
	Приложение Д (обязательное) Структуры слоя представления данных	100
	Приложение Е (обязательное) Слой бизнес-логики	102
	Приложение Ж (обязательное) Слой доступа к данным	105
	Приложение И (обязательное) Модульные тесты	107

Приложение К (рекомендуемое) Иллюстрационные материалы к тестированию API	109
--	-----

Введение

Современный этап цифровизации сферы санаторно-курортного и гостиничного обслуживания характеризуется переходом от разрозненных инструментов (отдельных сайтов, электронных таблиц, локальных учетных программ) к единым цифровым платформам, обеспечивающим клиенту сквозной сценарий выбора, бронирования и оплаты размещения. Малые санаторно-гостиничные предприятия испытывают потребность в информационных системах, которые сочетают в себе персонализированный подбор варианта размещения, прозрачный учет бронирований и контролируемый платежный контур, и при этом допускают дальнейшее расширение без полной переработки программного продукта.

Актуальность темы выпускной квалификационной работы обусловлена необходимостью построения подобной платформы, ориентированной не на крупные сетевые гостиницы, а на сегмент малых санаторно-курортных учреждений. Такие предприятия, как правило, не располагают собственным цифровым контуром и вынуждены опираться на универсальные агрегаторы, не учитывающие специфику медицинского профиля, длительности оздоровительных программ и сезонных особенностей размещения. Это снижает эффективность подбора варианта размещения и затрудняет принятие клиентом обоснованного решения [1].

Объектом исследования являются процессы клиентского взаимодействия с санаторно-курортными учреждениями, включающие подбор санатория, оформление и сопровождение бронирования, а также расчетно-платежные операции.

Предметом исследования выступают методы и средства проектирования и разработки информационной системы, обеспечивающей персонализированный подбор санаториев по совокупности критериев (город, медицинский профиль, удаленность от моря, ценовой диапазон, период проживания) и сквозное оформление бронирования [2] с последующей оплатой.

Целью данной выпускной квалификационной работы является проектирование и разработка информационной системы персонализированного подбора и бронирования санаторно-курортных услуг, начиная от выбора клиентом санатория по совокупности критериев (город, медицинский профиль, удаленность от моря, ценовой диапазон, период проживания) и заканчивая оформлением бронирования, выставлением счета и оплатой. Программное средство должно предоставлять удобный интерфейс для взаимодействия со всеми разделами: клиентским (каталог, бронирования, оплата) и служебным (управление санаториями и контроль бронирований). Основная задача выпускной квалификационной работы – разработка серверной части распределенного программного комплекса на основе микросервисной архитектуры, обеспечивающая управление учетными записями пользователей, ведение каталога санаториев, обработку бронирований и проведение расчетно-платежных операций с учетом специфики санаторно-курортного сегмента. В рамках данной работы планируется создать систему, способная согласованно обрабатывать запросы клиентского и административного контуров при проверке доступности размещения, изменении параметров бронирования и фиксации оплаты. Система должна включать в себя модули аутентификации, каталога и бронирований, оплаты, а также клиентское веб-приложение для персонализированного подбора и сопровождения заявок, что позволит сформировать полноценное решение для автоматизации процессов малого санаторно-гостиничного бизнеса.

Для достижения поставленной цели в работе последовательно решаются задачи анализа предметной области и формирования функциональных и нефункциональных требований; проектирования микросервисной архитектуры, моделей данных и пользовательского интерфейса; программной реализации серверных модулей аутентификации, каталога и бронирований, расчетно-платёжного контура и клиентского веб-приложения; а также тестирования и оценки качества разработанного программного комплекса.

1 Анализ предметной области и постановка задачи на разработку программных средств персонализированного подбора и бронирования санаторно-курортных услуг

1.1 Постановка задачи на разработку программных средств

В рамках выпускной квалификационной работы необходимо разработать информационную систему, предоставляющую клиенту инструмент персонализированного подбора и бронирования санаторно-курортных услуг. Система должна охватывать сквозной сценарий: от регистрации и поиска подходящего санатория до оформления договора и подтверждения оплаты. Архитектурно решение разделяется на серверную часть, реализованную на основе микросервисного подхода, и клиентскую часть, представленную одностраничным веб-приложением.

Разрабатываемая система должна реализовывать следующие функции:

- регистрация и аутентификация клиента, выдача токена доступа для последующих защищенных операций;
- персонализированный подбор санаториев по совокупности критериев: город, медицинский профиль, расстояние до моря, ценовой диапазон, период проживания, число гостей;
- предоставление детальной информации о санатории, включая состав услуг, медицинский профиль, расположение и фотогалерею;
- проверка фактической доступности санатория на заданные даты с учетом существующих бронирований и вместимости;
- создание, просмотр, изменение и отмена клиентских бронирований;
- оформление договора с клиентом сотрудником служебного контура (менеджером);
- автоматическое формирование счета на оплату по факту оформления договора;

- идемпотентная обработка платежей с контролем соответствия суммы и автоматическим обновлением платежного статуса договора;
- публикация доменных событий жизненного цикла бронирования и платежа для асинхронной согласованности данных между сервисами;
- разграничение доступа по ролям (клиент, менеджер, бухгалтер, администратор);
- предоставление REST-API с документацией в формате Swagger и служебных точек доступа мониторинга.

В качестве основного языка программирования серверной части необходимо использовать Go (Golang), так как он сочетает высокую производительность, встроенную поддержку конкурентности и эффективную работу с сетевыми протоколами, что соответствует требованиям микросервисной архитектуры. В качестве реляционной системы управления базами данных применяется PostgreSQL – широко используемое промышленное решение, обеспечивающее необходимый уровень изоляции транзакций (Serializable) для критичных финансовых операций [3].

Для асинхронного взаимодействия между сервисами должен использоваться брокер сообщений NATS [4]. Он отличается легкостью развертывания, низкими накладными расходами на доставку сообщений и поддержкой режима JetStream для гарантированной доставки доменных событий.

Клиентская часть системы должна быть реализована в виде одностраничного веб-приложения (Single-Page Application) на базе библиотеки React и языка программирования TypeScript. Такой подход обеспечивает строгую типизацию, повышает сопровождаемость кода и снижает количество ошибок на этапе разработки. В качестве сборщика проекта применяется Vite, обеспечивающий быструю горячую перезагрузку и оптимизированную production-сборку. Для управления состоянием серверных данных используется библиотека React Query, для глобального состояния клиента – Zustand, для маршрутизации – React Router, для стилизации – Tailwind CSS.

В качестве протокола межсервисного синхронного взаимодействия применяется REST поверх HTTP, обмен данными – в формате JSON. Этот формат является универсальным, поддерживается всеми современными языками программирования и удобен для чтения, отладки и документирования.

Все компоненты системы должны быть контейнеризированы с помощью Docker и оркестрированы средствами Docker Compose. Это обеспечивает воспроизводимость среды выполнения, упрощает локальную разработку и подготавливает основу для последующего развертывания в кластерных средах.

В качестве среды разработки используется Visual Studio Code как универсальный кроссплатформенный редактор с обширной экосистемой расширений для языков Go и TypeScript, а также для работы с Docker и базами данных. Целевой операционной системой разработки выбрана Microsoft Windows 11, обеспечивающая совместимость со всеми перечисленными инструментами.

При выполнении всех вышеприведенных требований разрабатываемая информационная система будет обеспечивать реализацию всех заявленных возможностей и формировать целостный клиентский контур персонализированного подбора и бронирования санаторно-курортных услуг.

1.2 Анализ аналогов разрабатываемого приложения

Проведен анализ существующих программных решений, обладающих функциональностью, частично или полностью пересекающейся с разрабатываемой системой. Основное назначение разрабатываемой системы заключается в персонализированном подборе санаториев по медицинским и потребительским критериям, оформлении бронирования и сопровождении платежного контура для сегмента малых санаторно-курортных предприятий.

В качестве рассматриваемых аналогов выбраны следующие платформы:

- Booking.com – международный агрегатор средств размещения, поддерживающий поиск и бронирование гостиниц, апартаментов и иных объектов;

- sanatory.ru – специализированный российский сервис подбора санаторно-курортных учреждений с возможностью фильтрации по медицинскому профилю;
- TUI Россия – туристический оператор с собственной платформой бронирования туров и санаторно-курортного отдыха;
- tripadvisor – международная платформа отзывов и бронирования средств размещения с агрегированной оценкой пользователей.

Для сравнения рассмотренных платформ выделен ряд критериев, который приведен в таблице 1.

Таблица 1 – Критерии сравнения

Код	Критерий	Пояснение
K1	Фильтрация по медицинскому профилю	Наличие возможности подбора санатория по профилю лечения
K2	Фильтрация по расстоянию до моря	Наличие параметра удаленности объекта размещения от моря при поиске
K3	Фильтрация по ценовому диапазону	Наличие фильтра по стоимости размещения
K4	Сквозное оформление бронирования и оплаты	Наличие полного цикла: выбор → бронирование → оплата в рамках одной платформы
K5	Ориентация на малые санаторно-курортные предприятия	Наличие инструментов, рассчитанных на небольшие учреждения, а не на сетевые отели
K6	Открытый REST API и Swagger-документация	Наличие задокументированного API для интеграции со сторонними системами

Наличие реализованной возможности обозначается знаком «+», отсутствие – знаком «-».

Далее, в таблице 2, приведено сравнение функциональных возможностей рассматриваемых платформ.

Таблица 2 – Сравнительная таблица приложений-аналогов

Критерий	Booking .com	Sanatory .ru	TUI	Trip- advisor
K1	-	+	-	-
K2	-	+	-	-
K3	+	+	+	+
K4	+	-	+	-
K5	-	-	-	-
K6	-	-	-	-

Проведенное сравнение показывает, что существующие платформы реализуют только отдельные аспекты задачи.

Каждая из рассмотренных платформ обладает собственным набором сильных сторон, однако ни одна из них не обеспечивает одновременно специализированный подбор по медицинскому профилю, ориентацию на сегмент малых санаторно-курортных предприятий и возможность независимого развертывания решения в инфраструктуре конкретного учреждения.

Таким образом, Booking.com и Tripadvisor ориентированы на массовое размещение и не учитывают медицинскую специфику санаторно-курортных услуг. Sanatory.ru обеспечивает специализированный подбор, однако работает в формате агрегатора и не предоставляет малым учреждениям самостоятельного программного продукта. TUI Россия ориентирован на туристические пакеты и собственную сеть партнеров.

В отличие от рассмотренных аналогов, разрабатываемая система ориентирована именно на сегмент малых санаторно-курортных предприятий и

предоставляет решение, ориентированное на развертывание в инфраструктуре конкретного учреждения посредством контейнеризации (Docker Compose). Помимо специализированного подбора по медицинскому профилю и расстоянию до моря, разрабатываемая система предоставляет открытый REST API со Swagger-документацией, что позволяет интегрировать ее со сторонними учетными системами учреждения.

Разрабатываемая система направлена на обеспечение более широких функциональных возможностей за счет сочетания персонализированного подбора по медицинскому профилю, сквозного оформления бронирования и оплаты, что позволит сформировать универсальный инструмент цифровизации клиентского контура санаторно-курортных предприятий.

1.3 Анализ нефункциональных требований

При проектировании информационной системы необходимо определить нефункциональные требования, которые задают условия надежной, безопасной и сопровождаемой эксплуатации программного комплекса. Ниже сформулированы основные группы таких требований и способы их удовлетворения в выбранной архитектуре.

Требования к безопасности предполагают ограничение доступа к защищенным функциям системы. Аутентификация клиентов и сотрудников служебного контура выполняется на основе JWT-токенов. Межсервисное взаимодействие защищается внутренним API-ключом. Пароли пользователей хранятся в виде bcrypt-хеша.

Масштабируемость обеспечивается возможностью независимого расширения и развертывания отдельных подсистем. Для этого в архитектуре выделены самостоятельные микросервисы аутентификации, бронирований и каталога, а также оплаты. Каждый из них размещается в отдельном Docker-контейнере.

Надежность системы выражается в том, что отказ одной подсистемы не должен приводить к полному отказу приложения. Критичная бизнес-логика

оплаты счета защищена от гонок данных. Для этого применяются транзакции PostgreSQL с уровнем изоляции Serializable и идемпотентные ключи платежей.

Наблюдаемость реализуется за счет единообразных точек мониторинга в каждом микросервисе: эндпоинтов /health, /health/ready и /metrics в формате Prometheus. Это позволяет подключать внешние средства мониторинга и алертинга без изменения прикладного кода.

Сопровождаемость достигается применением единой многослойной архитектуры сервисов. Контракты внешнего взаимодействия формализованы в виде REST API и сопровождаются Swagger-документацией.

Совместимость клиентской части обеспечивается корректной работой веб-приложения в актуальных версиях браузеров Google Chrome, Mozilla Firefox и Microsoft Edge на персональных компьютерах и мобильных устройствах.

Удобство использования предполагает интуитивно понятный клиентский интерфейс, адаптивную верстку и минимальное число шагов при оформлении бронирования.

Соблюдение перечисленных нефункциональных требований обеспечивает стабильную, надежную и сопровождаемую работу разрабатываемой системы.

1.4 Анализ функциональных требований

В процессе анализа функциональных требований на основе поставленной задачи были выделены следующие сценарии работы пользователей системы:

- зарегистрироваться в системе;
- пройти аутентификацию и получить токен доступа;
- открыть каталог санаториев;
- задать параметры персонализированного подбора (город, медицинский профиль, расстояние до моря, ценовой диапазон, даты проживания, число гостей);

- получить список санаториев, соответствующих заданным критериям;
- просмотреть детальную информацию о выбранном санатории;
- проверить доступность санатория на заданный период;
- создать бронирование на выбранные даты;
- просмотреть список собственных бронирований;
- изменить параметры существующего бронирования;
- отменить ранее созданное бронирование;
- оформить договор с клиентом (сценарий служебного контура);
- зафиксировать оплату счета по договору (сценарий служебного контура);
- получить документацию по REST API системы;
- получить сведения о работоспособности компонентов системы (для администратора).

Перечисленные сценарии охватывают полный цикл работы клиента: от регистрации и входа до подбора санатория по заданным критериям, оформления бронирования и последующего сопровождения заявки, включая изменение параметров и отмену. Отдельно выделены служебные операции административного контура – оформление договора и фиксация оплаты, а также вспомогательные возможности получения документации REST API и контроля работоспособности компонентов системы.

Такое разделение отражает двухконтурную модель приложения: клиентский контур ориентирован на самостоятельное взаимодействие пользователя с каталогом и бронированиями, служебный – на операции персонала предприятия. Для наглядного представления взаимосвязей между актёрами и перечисленными функциями была составлена диаграмма вариантов использования [5] (рисунок 1).

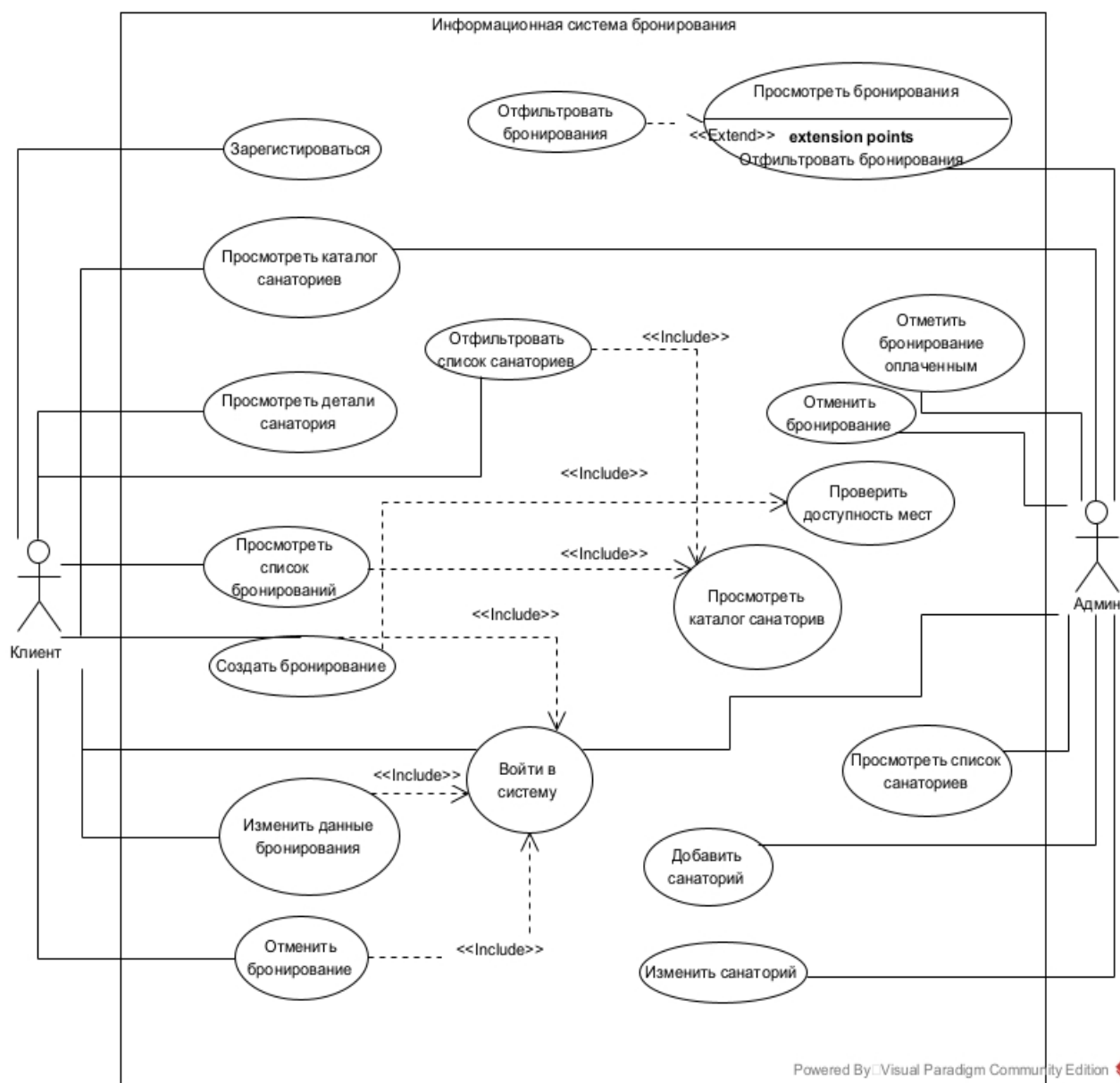


Рисунок 1 – Диаграмма вариантов использования

На рисунках 2-3 представлено описание некоторых наиболее важных сценариев вариантов использования (далее ВИ) разрабатываемого приложения. В текстовых спецификациях для выбранных ВИ фиксируются цель сценария, последовательность действий актёров и системы, возможные альтернативные ветви и условия завершения, что позволяет формализовать требования к поведению программы до этапа реализации. Описанные сценарии далее используются как основа для проектирования интерфейса, построения тестовых кейсов и проверки полноты реализованного функционала.

Прецедент: создать бронирование
ID: 7
Краткое описание: клиент выбирает санаторий, указывает даты и создаёт бронирование. Включает проверку доступности мест, расчёт стоимости и оплату.
Действующие лица: клиент
Предусловие: клиент аутентифицирован; санаторий выбран в каталоге
Основной поток: <ol style="list-style-type: none"> 1. Клиент указывает дату заезда, дату выезда и количество гостей. 2. Система проверяет корректность дат и наличие свободных мест на указанный период. 3. Система рассчитывает стоимость бронирования исходя из цены санатория и длительности проживания. 4. Клиент подтверждает бронирование и переходит к оплате. 5. Система создаёт бронирование в статусе confirmed и публикует событие в брокер сообщений. 6. Система возвращает клиенту подтверждение с деталями бронирования.
Постусловие: бронирование создано и отображается в списке бронирований клиента.

Рисунок 2 – Описание ВИ «Создать бронирование»

Создание бронирования является центральным сценарием клиентского контура. Вариант использования включает проверку доступности мест, расчет стоимости и оплату бронирования.

Прецедент: изменить данные бронирования
ID: 3
Краткое описание: клиент пожелал изменить данные уже созданного бронирования: даты заезда/выезда или количество гостей
Действующие лица: клиент
Предусловие: клиент аутентифицирован; бронирование существует, принадлежит клиенту и находится в статусе confirmed
Основной поток: <ol style="list-style-type: none"> 1. Клиент открывает список своих бронирований 2. Клиент выбирает бронирование и переходит к его редактированию 3. Клиент изменяет параметры: дату заезда, дату выезда или количество гостей 4. Система проверяет корректность новых дат и наличие свободных мест на изменённый период с учётом существующих бронирований. 5. Система сохраняет обновлённые параметры бронирования 6. Система возвращает клиенту подтверждение с обновлёнными данными
Постусловие: Параметры бронирования обновлены и отражены в личном кабинете клиента.

Рисунок 3 – Описание ВИ «Изменить данные бронирования»

После создания бронирования клиент имеет возможность скорректировать его параметры. На рисунке 4 представлен сценарий первичного взаимодействия клиента с системой.

Далее, на рисунках 4 и 5 приводятся прецеденты, связанные с авторизацией клиента.

Прецедент: зарегистрироваться
ID: 1
Краткое описание: клиент создаёт учётную запись в системе, вводя адрес электронной почты, пароль и полное имя.
Действующие лица: клиент
Предусловие: клиент открыл страницу регистрации
Основной поток: <ol style="list-style-type: none"> 1. Клиент вводит адрес электронной почты, пароль и полное имя. 2. Система проверяет корректность формата email и наличие обязательных полей. 3. Система передаёт данные в сервис аутентификации, который сохраняет учётную запись (пароль хранится в виде bcrypt-хеша). 4. Система уведомляет клиента об успешной регистрации и предлагает перейти ко входу в систему.
Постусловие: учётная запись клиента создана; клиент может выполнить вход в систему.

Рисунок 4 – Описание ВИ «Зарегистрироваться»

Регистрация является обязательным предварительным условием для доступа к защищенным функциям системы. После успешной регистрации клиент выполняет вход в систему – рисунок 5.

Прецедент: войти в систему
ID: 2
Краткое описание: клиент вводит учётные данные и получает JWT-токен для доступа к защищённым функциям.
Действующие лица: клиент
Предусловие: учётная запись клиента создана
Основной поток: <ol style="list-style-type: none"> 1. Клиент вводит адрес электронной почты и пароль. 2. Система проверяет учётные данные через сервис аутентификации. 3. Сервис аутентификации проверяет пароль и генерирует JWT-токен с идентификатором пользователя и ролью. 4. Клиент получает токен доступа и переходит в личный кабинет.
Постусловие: клиент аутентифицирован; токен доступа сохранён и используется при последующих операциях.

Рисунок 5 – Описание ВИ «войти в систему»

Вход в систему обеспечивает клиенту получение JWT-токена, необходимого для создания и управления бронированиями. Помимо авторизованных операций, система предоставляет возможность поиска санаториев без входа в систему – рисунок 6.

Прецедент: отфильтровать список санаториев
ID: 4
Краткое описание: клиент задаёт параметры фильтрации и получает список санаториев, соответствующих заданным критериям.
Действующие лица: клиент
Предусловие: клиент открыл страницу каталога санаториев
<p>Основной поток:</p> <ol style="list-style-type: none"> 1. Клиент задаёт один или несколько параметров фильтрации: город, медицинский профиль, расстояние до моря, ценовой диапазон, даты проживания, количество гостей. 2. Система отправляет запрос с параметрами фильтрации на сервер. 3. Сервер возвращает список санаториев, соответствующих критериям, с пагинацией. 4. Клиент просматривает результаты и при необходимости уточняет параметры подбора.
Постусловие: клиент получил список подходящих санаториев и может перейти к просмотру деталей.

Рисунок 6 – Описание ВИ «Отфильтровать список санаториев»

Спроектированные варианты использования будут реализованы в ходе разработки приложения.

1.5 Планирование разработки и оценка бюджета

Для отражения основных этапов разработки информационной системы персонализированного подбора и бронирования санаторно-курортных услуг в соответствии с заданными требованиями и технологиями составлена диаграмма Ганта плана разработки в среде Microsoft Office Project [6] (рисунок 7).

На диаграмме представлена календарная последовательность работ от анализа предметной области и формирования требований до тестирования программного комплекса и оформления пояснительной записки.

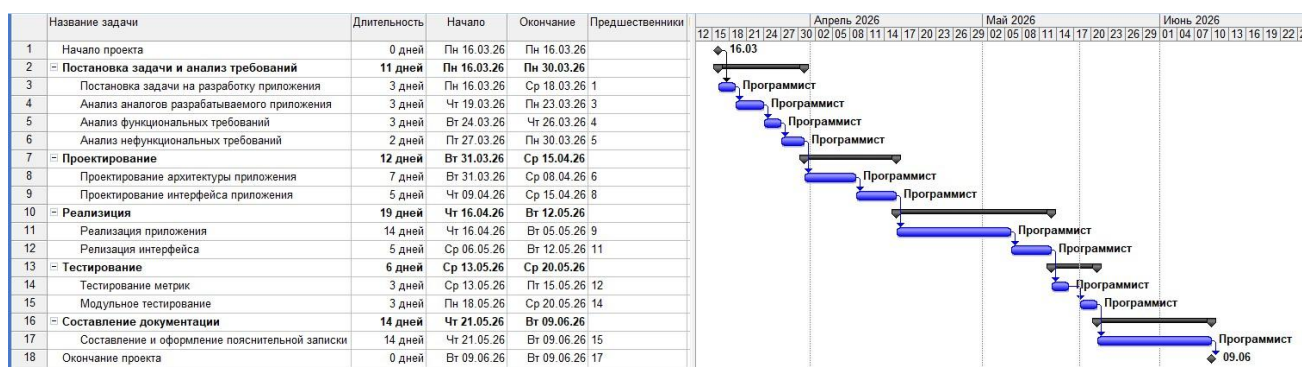


Рисунок 7 – Диаграмма Ганта плана разработки

На диаграмме можно увидеть все этапы разработки приложения, начиная от постановки задачи и анализа требований, и заканчивая тестированием и составлением пояснительной записки.

В разработке приложения участвует только один человек – программист, со ставкой 200 рублей в час.

Итоговая статистика разработки проекта представлена на рисунке 8.

	Длительность	Трудозатраты	Затраты
Текущие	62д	496ч	р.99 200,00
Базовые	0д?	0ч	р.0,00
Фактические	0д	0ч	р.0,00
Оставшиеся	62д	496ч	р.99 200,00

Рисунок 8 – Статистика проекта

Стоимость работ определяется исходя из почасовой ставки, закреплённой в ресурсном плане Microsoft Office Project, и автоматически агрегируется по завершённому перечню работ диаграммы Ганта. Полученные оценки согласованы с объёмом функциональности, предусмотренной для клиентского и служебного контуров системы, и отражают реалистичный срок создания микросервисного программного комплекса с веб-интерфейсом в условиях индивидуальной разработки.

Подводя итог, можно видеть, что разработка информационной системы персонализированного подбора и бронирования санаторно-курортных услуг займет 62 рабочих дня, что соответствует 496 часам. Бюджет разработки составит почти 100 тысяч рублей.

2 Проектирование программных средств персонализированного подбора и бронирования санаторно-курортных услуг

Проектирование программных средств представляет собой этап, на котором формализуются архитектура, структура данных, состав программных компонентов и правила их взаимодействия. Для разрабатываемой системы данный этап опирается на результаты анализа предметной области и требований, а также на принятое технологическое решение о построении распределенного приложения в виде набора слабо связанных микросервисов на языке Go с веб-клиентом для конечных пользователей.

2.1 Архитектура и состав системы

При проектировании архитектуры учитывались масштабируемость, возможность независимого развертывания отдельных частей приложения, разграничение доступа клиентов и администратора, а также необходимость согласованной работы подсистем каталога, бронирований и оплаты.

Монолитная архитектура для описываемого класса задач обеспечивает простоту первоначальной разработки, однако связывает масштабирование, сопровождение и выпуск обновлений в единый цикл. В условиях, когда каталог санаториев, клиентские бронирования и платежный контур развиваются с разной скоростью и предъявляют различные требования к надежности, предпочтительным является выделение самостоятельных сервисов с четкими границами ответственности.

В качестве базового архитектурного стиля принята микросервисная архитектура (MSA): совокупность автономно развертываемых компонентов, каждый из которых инкапсулирует отдельный фрагмент бизнес-логики и взаимодействует с другими компонентами по сетевым протоколам. Клиентское приложение реализует роль внешнего потребителя REST API и не обращается напрямую к внутренним сервисам оплаты, что снижает поверхность атаки и упрощает эволюцию серверных контрактов.

Для наглядного представления размещения программных и инфраструктурных элементов подготовлена диаграмма развертывания (рисунок 9).

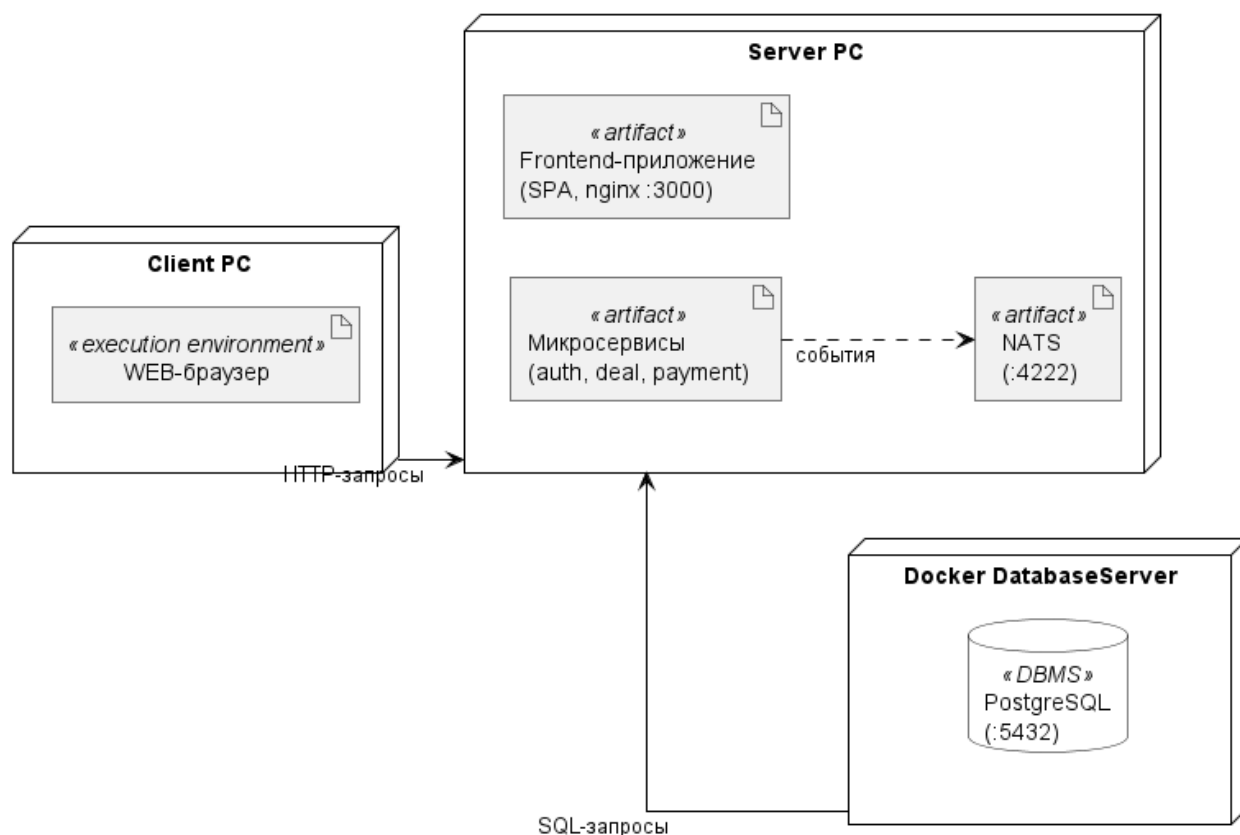


Рисунок 9 – Архитектура (развертывание) информационной системы персонализированного подбора и бронирования санаторно-курортных услуг

Клиентская часть представлена одностраничным веб-приложением (SPA), размещаемым в контейнере с HTTP-сервером. Через него клиент выполняет подбор санатория по параметрам, оформляет и сопровождает бронирования, инициирует оплату. Администратор предприятия использует тот же интерфейс в служебном режиме: ведет справочник санаториев и контролирует бронирования клиентов, включая выставление счета и подтверждение оплаты.

Микросервис аутентификации (auth-service) отвечает за регистрацию клиентов, проверку учетных данных и формирование JWT-токенов доступа.

Микросервис бронирований и каталога (deal-service) является центральным узлом прикладной логики: публикует REST API для каталога и бронирований, проксирует операции входа и регистрации в auth-service,

реализует административный API, выполняет синхронные вызовы payment-service при выставлении счета и обрабатывает асинхронные уведомления об оплате.

Микросервис оплаты (payment-service) управляет счетами и платежами, обеспечивает идемпотентную обработку повторных запросов на оплату и публикует доменные события о завершенных платежах.

Общий модуль platform-common концентрирует инфраструктурные возможности: структурированное логирование, обертку над брокером NATS, middleware для метрик Prometheus и передачи идентификатора трассировки в цепочке запросов.

СУБД PostgreSQL используется как единое реляционное хранилище с логическим разделением на схемы auth, deal и payment, что соответствует границам микросервисов и упрощает последующее физическое разделение данных при росте нагрузки.

Брокер сообщений NATS применяется для асинхронной доставки доменных событий жизненного цикла бронирований и платежей.

Клиентское одностраничное приложение обращается к серверной части через REST API микросервиса бронирований и каталога, выступающего единой точкой входа для браузера. Микросервис аутентификации обеспечивает регистрацию, проверку учётных данных и выдачу JWT; микросервис оплаты ведёт счета и платежи, согласование жизненного цикла бронирования и оплаты дополняется обменом событиями через NATS.

Таким образом, обеспечивается надежное и стабильное функционирование всей системы, соответствие микросервисной архитектуре и современным требованиям рынка. Построенная архитектура позволяет расширять программный код, без необходимости глубокой переработки системы, что в свою очередь снижает расходы на дальнейшую поддержку.

Взаимосвязь прикладных модулей и направление основных потоков данных отражены на диаграмме компонентов [7] (рисунок 10).

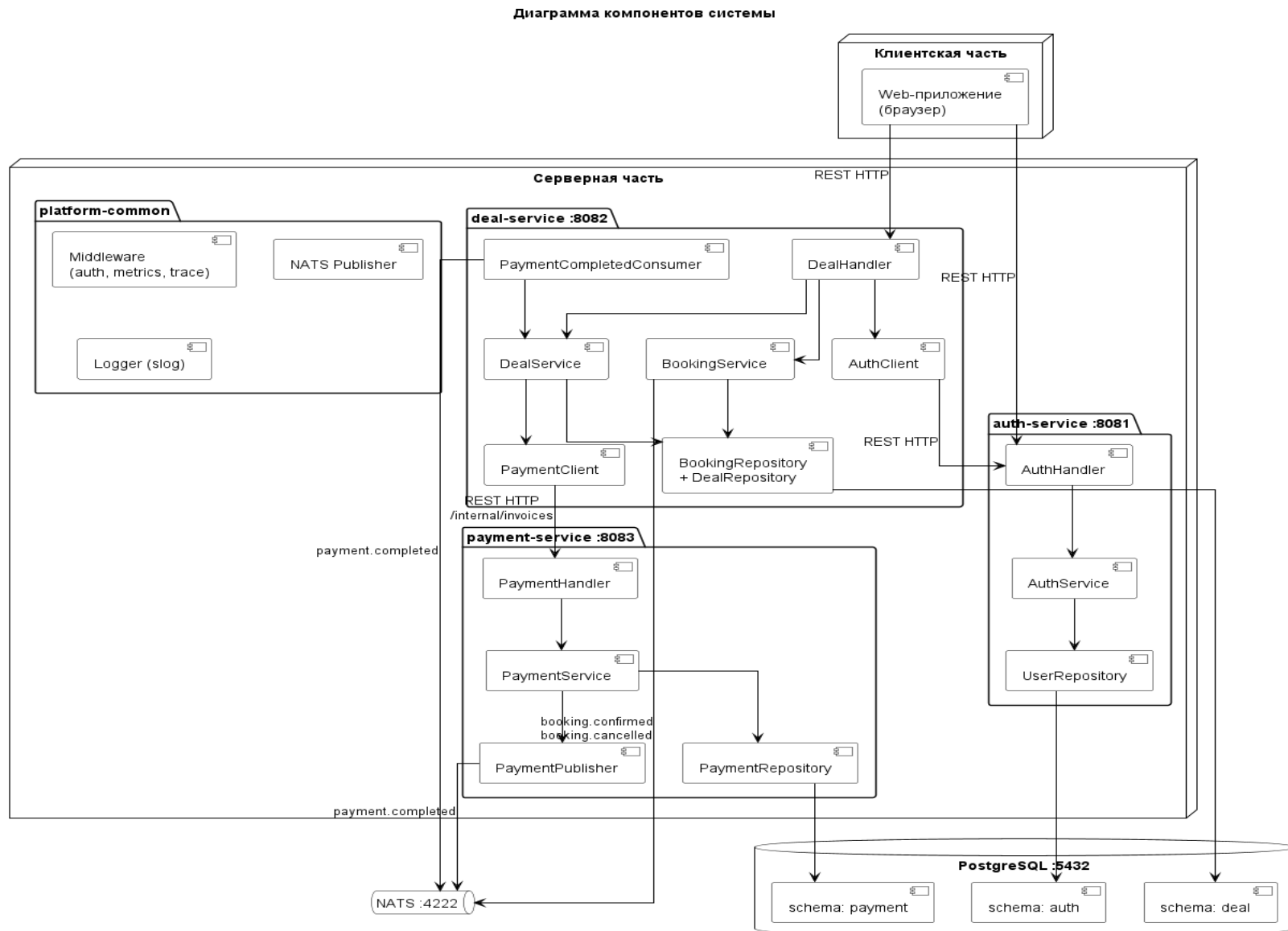


Рисунок 10 – Диаграмма компонентов информационной системы

Принятая схема позволяет масштабировать отдельные сервисы, локализовать отказы и развивать клиентский и служебный контуры без полной переработки монолитного ядра.

2.2 Проектирование структур данных

2.2.1 Концептуальная модель данных

Концептуальная модель отражает сущности предметной области и связи между ними без привязки к конкретной СУБД. Перед построением модели выделены основные объекты учета, соответствующие бизнес-процессам подбора, бронирования и оплаты санаторно-курортных услуг. Сущности концептуальной модели данных приведены в таблице 3.

Таблица 3 – Сущности концептуальной модели данных

Название сущности	Условный идентификатор	Назначение сущности
Пользователь	User	Учетная запись клиента или администратора системы
Санаторий	Sanatorium	Описание объекта размещения и лечения, параметры для фильтрации
Медицинский профиль	MedicalProfile	Справочник профилей лечения для персонализированного подбора
Бронирование	Booking	Заявка клиента на размещение в санатории на период дат
Счет на оплату	Invoice	Документ на оплату, связанный с бронированием
Платеж	Payment	Факт оплаты счета с ключом идемпотентности
Договор (служебный контур)	Contract	Резервная сущность служебного учета размещения (при расширении B2B-сценариев)

Разграничение блоков позволяет зафиксировать границы ответственности подсистем разрабатываемой информационной системы. Каталог формирует справочную основу для фильтрации и сравнения вариантов размещения; при этом связь санатория с медицинскими профилями отражает отраслевую специфику санаторно-курортных услуг, отличающую предметную область от универсальных гостиничных агрегаторов. Клиентский контур связывает учётную запись пользователя с конкретной заявкой на проживание и тем самым обеспечивает персонализацию сценария: каждое бронирование относится к определённому клиенту, санаторию и календарному периоду. Платёжный контур замыкает бизнес-процесс, переводя подтверждённое намерение разместиться в юридически и финансово значимую операцию – выставление счёта и фиксацию оплаты.

При переходе от перечня сущностей к графическому описанию на концептуальном уровне фиксируются состав объектов учёта и бизнес-смысл связей между ними без привязки к реляционным таблицам, типам данных и первичным ключам. В модель клиентского контура включены пользователь, санаторий, медицинский профиль, бронирование, счёт на оплату и платёж; для каждой сущности указываются атрибуты, необходимые для описания предметной области, учётные данные клиента, параметры объекта размещения, сведения о заявке, сумме и статусе расчётов. Связь санатория с медицинским профилем отражает персонализированный подбор по направлениям лечения, а цепочка «бронирование-счёт-платёж» – сквозной сценарий оформления и оплаты услуги. Служебная сущность договора на диаграмме не показывается, поскольку не входит в основной клиентский процесс.

Для наглядного представления состава объектов учета и характера связей между ними построена концептуальная диаграмма в нотации «сущность–связь» [8]. На рисунке 11 приведено графическое представление модели предметной области «Персонализированный подбор и бронирование санаторно-курортных услуг».

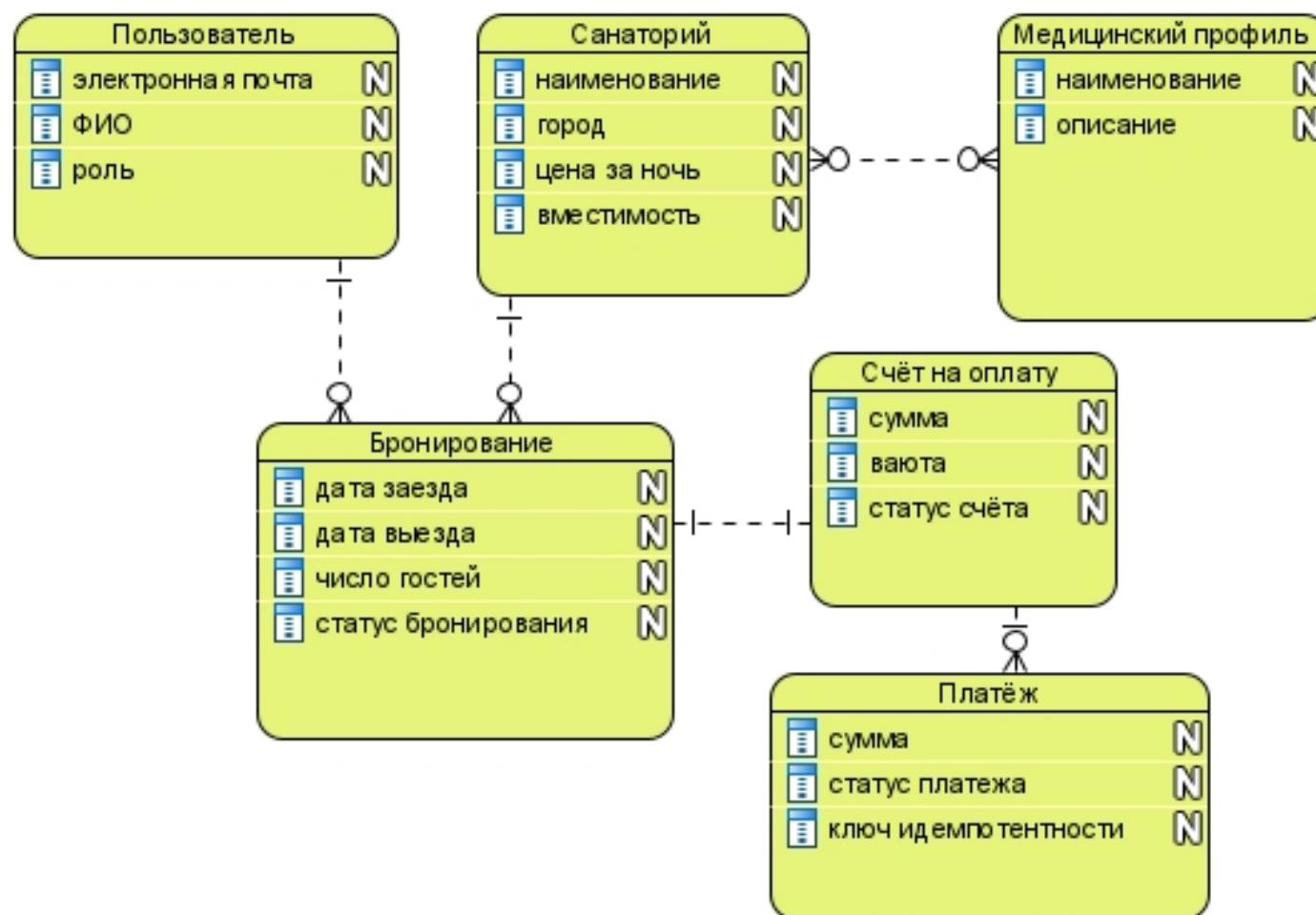


Рисунок 11 – Концептуальная модель данных предметной области «Персонализированный подбор и бронирование санаторно-курортных услуг»

На диаграмме отражены центральные объекты учета и направление связей между ними: от каталога санаториев и профилей лечения – к бронированию клиента, от бронирования – к выставлению счета и фиксации платежа. Графическое представление позволяет оценить полноту модели относительно сквозного сценария подбора, бронирования и оплаты.

Для однозначной фиксации характера связей и их кратности (один-ко-многим, многие-ко-многим, один-к-одному) составлена таблица 4. В ней приведены пары сущностей, тип связи и краткое описание бизнес-смысла каждого отношения; данные таблицы 4 используются при переходе к логическому проектированию реляционной схемы.

Таблица 4 – Связи между сущностями концептуальной модели

Связь	Сущность 1	Сущность 2	Тип связи	Описание
Пользователь – Бронирование	Пользователь	Бронирование	1:N	Один клиент может иметь множество бронирований
Санаторий – Бронирование	Санаторий	Бронирование	1:N	Один санаторий принимает множество бронирований на разные периоды
Санаторий – Медицинский профиль	Санаторий	Медицинский профиль	M:N	Санаторий специализируется на нескольких профилях лечения
Бронирование – Счет	Бронирование	Счет	1:1	Для клиентского сценария счет выставляется на конкретное бронирование
Счет – Платеж	Счет	Платеж	1:N	По счету допускается учет попыток оплаты; успешная фиксируется идиempотентно

Концептуальная модель служит основой для логического проектирования реляционной базы данных и согласования бизнес-правил с ограничениями целостности на уровне СУБД [9].

2.2.2 Логическая модель данных

Логическая модель данных – это расширение концептуальной модели данных. Она включает в себя все сущности, атрибуты, ключи и взаимосвязи, которые представляют бизнес-информацию и определяют бизнес-правила [10]. В разрабатываемой системе применяется подход «схема на домен»: данные группируются в пространства имен PostgreSQL, что соответствует границам микросервисов и облегчает разграничение доступа.

В схеме `auth` размещаются отношения, связанные с учетными записями: `users` (пользователи с полями `admin` и `client`), `refresh_tokens` (техническая таблица для возможного расширения механизма обновления токенов).

В схеме `deal` размещаются отношения предметной области каталога и бронирований: `medical_profiles`, `sanatoriums`, `sanatorium_medical_profiles` (связь многие-ко-многим), `bookings`. Дополнительно в схеме `deal` сохраняется отношение `contracts` для служебного контура договоров, что обеспечивает преемственность архитектуры при подключении корпоративных сценариев.

В схеме `payment` размещаются отношения `invoices` и `payments`. Счет (`invoices`) может ссылаться либо на бронирование (поле `booking_id`), либо на договор (поле `contract_id`); на уровне СУБД закреплено ограничение, требующее ровно одну из ссылок, что исключает неоднозначность предмета оплаты.

При переходе к логическому уровню сущности концептуальной модели отображаются в отношения с первичными и внешними ключами, атрибутами и типами данных PostgreSQL. На диаграмме отражён клиентский контур без служебных таблиц обновления токенов.

Логическая модель данных представлена на рисунке 12.

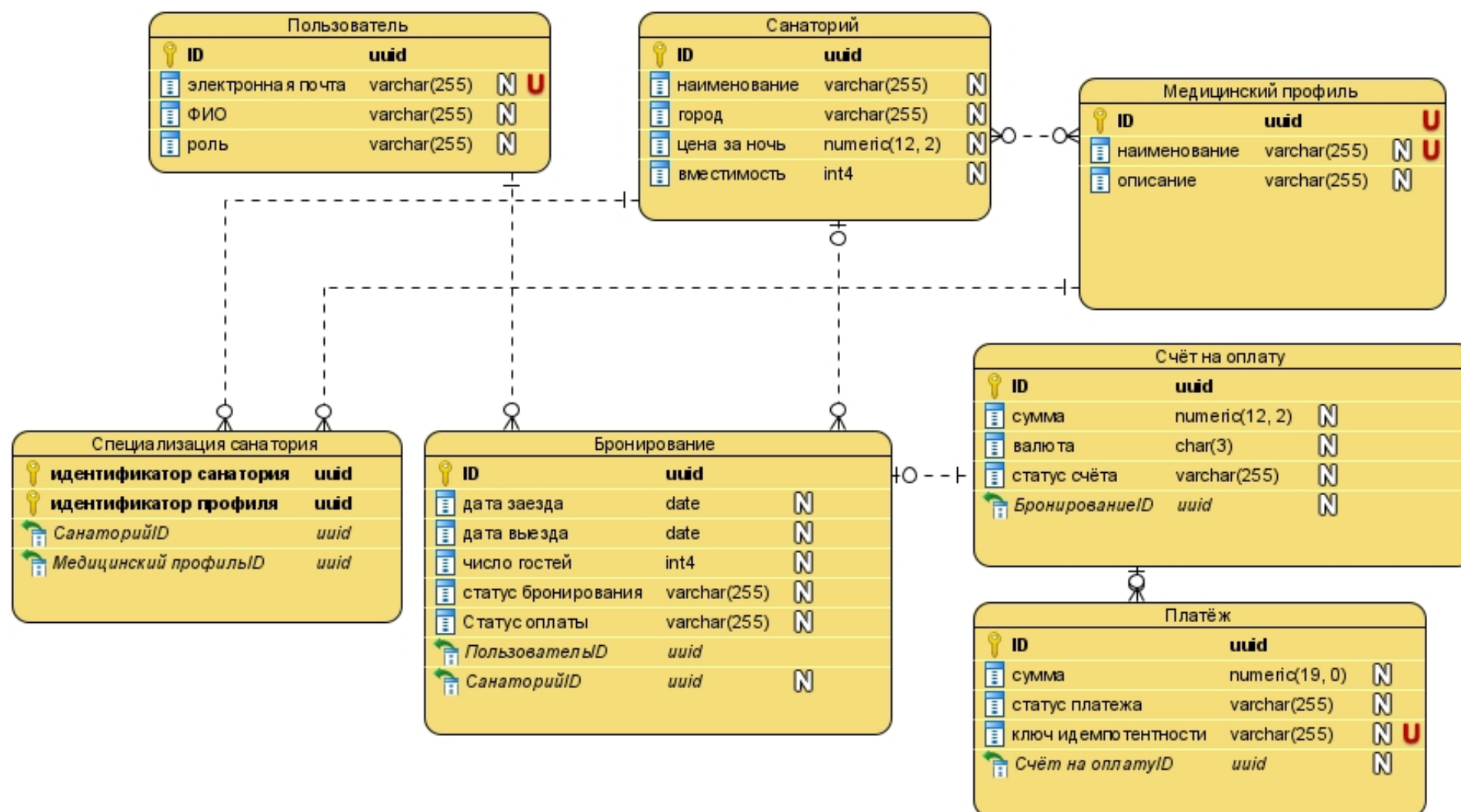


Рисунок 12 – Логическая модель данных (схемы auth, deal, payment)

Ключевые бизнес-правила, отраженные в логической модели и подкрепленные ограничениями:

- дата заезда в бронировании строго меньше даты выезда;
- число гостей и стоимость размещения положительны;
- платежный статус бронирования принимает значения `unpaid`, `invoice_issued`, `invoice_failed`, `paid`;
- ключ идемпотентности платежа уникален в пределах сервиса оплаты;
- уникальность пары «наименование санатория – город» предотвращает дублирование записей каталога при администрировании.

Полная спецификация атрибутов ключевых отношений схем `auth`, `deal` и `payment` приведена в приложении Б (таблицы Б.1-Б.5).

2.2.3 Проектирование физического уровня

Физическая модель данных показывает, как будет организована работа с данными: как они будут собираться, где и в каком виде храниться и как обрабатываться [11]. На физическом уровне для атрибутов задаются конкретные типы PostgreSQL, индексы и внешние ключи. Для ускорения фильтрации каталога предусмотрены индексы по городу, цене за ночь и расстоянию до моря; для проверки пересечения периодов бронирований используется расширение `btree_gist` и ограничения на диапазоны дат.

На рисунке показан фрагмент физической структуры: таблицы `deal.sanatoriums`, `deal.bookings`, `payment.invoices` и `payment.payments` с типами полей PostgreSQL. Связь бронирования и счёта задаётся полями `invoice_id` и `booking_id`; полные спецификации атрибутов приведены в приложении К.

Физическая модель (фрагмент схем `deal` и `payment`) приведена на рисунке 13; полное описание атрибутов основных таблиц вынесено в приложение К.



Рисунок 13 – Физическая модель данных (фрагмент)

На фрагменте представлены отношения `deal.bookings`, `payment.invoices` и `payment.payments` с указанием типов атрибутов PostgreSQL, первичных и внешних ключей, а также ограничений целостности (проверка периода проживания, допустимые значения платежного статуса, уникальность ключа идемпотентности). Связь бронирования со счетом и платежами отражает реализацию сквозного сценария оплаты на физическом уровне при разделении данных по схемам микросервисов `deal` и `payment`.

2.3 Проектирование пользовательского интерфейса

Интерфейс веб-приложения проектировался с учетом двух ролей: клиента, заинтересованного в подборе санатория и управлении собственными бронированиями, и администратора, обслуживающего каталог и контролирующего оплату.

В клиентском контуре предусмотрены следующие экраны: страница входа и регистрации; каталог санаториев с панелью фильтров (город, медицинские профили, расстояние до моря, ценовой диапазон, даты проживания, сортировка); карточка санатория с описанием, галереей, списком профилей и формой расчета бронирования; раздел «Мои бронирования» с операциями изменения, отмены и оплаты.

В служебном контуре администратора выделены вкладки «Бронирования» и «Санатории». На вкладке бронирований отображается реестр заявок с фильтрацией по статусу бронирования, статусу оплаты и городу; администратор может отменить бронирование, выставить счет и зафиксировать оплату. На вкладке санаториев реализованы просмотр списка, создание и редактирование записей в модальном окне, удаление с проверкой наличия активных бронирований.

2.4 Проектирование серверной части

Серверная часть разделена на три микросервиса [12]. Для каждого сервиса применяется единая многослойная организация кода:

- transport layer – HTTP-обработчики, middleware аутентификации, сериализация запросов и ответов REST API;
- service layer – валидация доменных правил, оркестрация операций, публикация и обработка событий;
- repository layer – доступ к PostgreSQL, транзакции, маппинг записей на доменные структуры;
- domain layer – типы и структуры бизнес-сущностей;
- integration layer – HTTP-клиенты к смежным сервисам и адаптеры брокера сообщений.

Такое разделение повышает тестируемость и сопровождаемость: бизнес-правила концентрируются в сервисном слое, а детали SQL и сетевых вызовов изолируются в инфраструктурных пакетах [13].

В качестве примера для детального рассмотрения структуры и взаимодействия слоев используется микросервис бронирований и каталога (deal-service). Выбор обусловлен тем, что принципы построения auth-service и payment-service аналогичны, а функционал deal-service репрезентативен для демонстрации ключевых архитектурных паттернов (каталог, бронирование, оплата, администрирование).

Детальное описание реализации слоев микросервисов представлено в приложениях:

- доменные объекты – в приложении В;
- SQL-скрипты создания и наполнения таблиц – в приложении Г;
- слой HTTP-представления (маршруты REST API) – в приложении Д;
- слой бизнес-логики – в приложении Е;
- слой обращения к базе данных – в приложении Ж.

2.4.1 Проектирование микросервиса аутентификации

Микросервис аутентификации инкапсулирует регистрацию клиентов и выдачу JWT. Диаграмма классов сервиса представлена на рисунке 14.

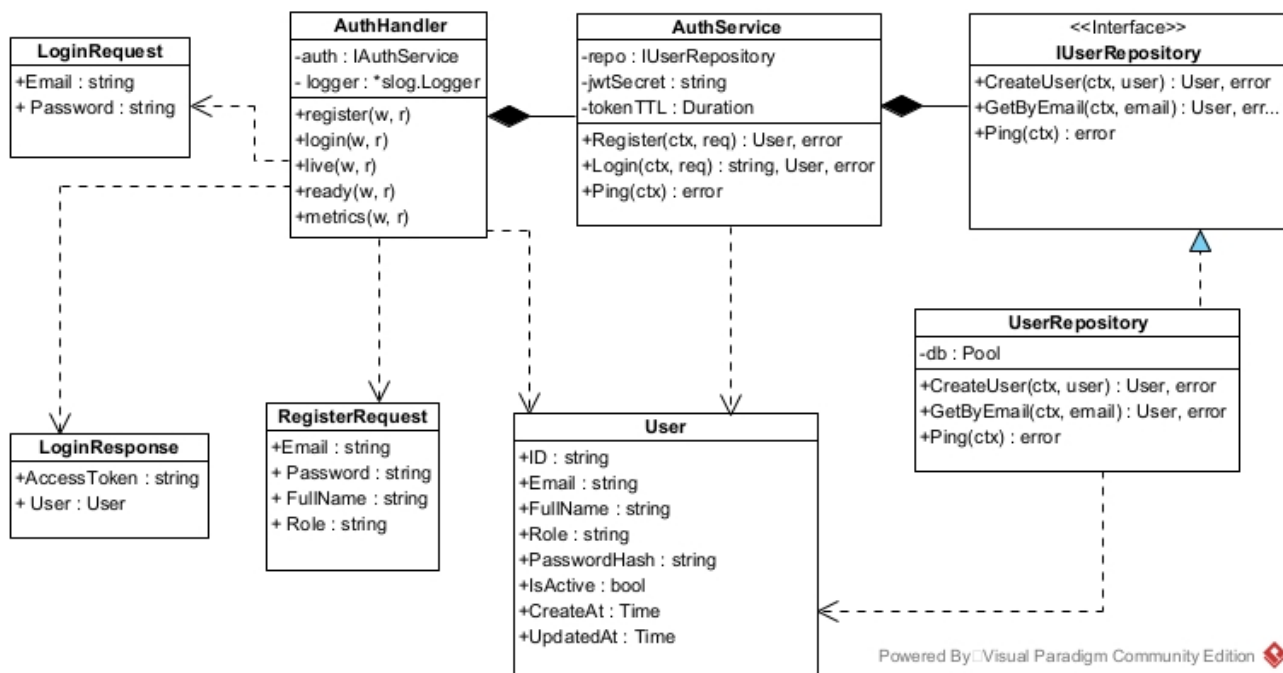


Рисунок 14 – Диаграмма классов микросервиса аутентификации (auth-service)

На диаграмме классов микросервиса аутентификации зафиксировано разделение на транспортный, сервисный и уровень доступа к данным в соответствии с принятой многослойной архитектурой. Такое построение изолирует HTTP-контракт и правила выдачи токенов от деталей хранения учётных записей в PostgreSQL.

Основными элементами проектного решения являются AuthHandler, обрабатывающий HTTP-запросы регистрации, входа и служебных проверок готовности; AuthService, выполняющий проверку входных данных, хеширование пароля алгоритмом bcrypt и генерацию JWT по алгоритму HS256; UserRepository, реализующий операции с таблицей auth.users; а также доменная модель User, описывающая пользователя через идентификатор, адрес электронной почты, полное имя, роль и хеш пароля.

Взаимодействие компонентов соответствует принятой многослойной схеме: AuthHandler принимает HTTP-запрос и передает его в AuthService, где выполняется проверка данных и формирование токена; для сохранения и чтения учетной записи сервис обращается к UserRepository, оперирующему доменной моделью User. Внешний контракт микросервиса, доступный смежным модулям по REST, приведен в таблице 5.

Таблица 5 – Методы REST API микросервиса аутентификации

Метод	Точка доступа	Описание	Параметры
POST	/api/v1/users/register	Зарегистрировать нового клиента	Body: { email, password, full_name, role }
POST	/api/v1/auth/login	Пройти аутентификацию и получить JWT	Body: { email, password }
GET	/health	Базовая проверка доступности сервиса	-
GET	/health/live	Liveness-проверка для оркестратора	-
GET	/health/ready	Readiness-проверка с контролем БД	-
GET	/metrics	Метрики в формате Prometheus	-

Для клиентского SPA прямой вызов auth-service не используется: deal-service выступает единой точкой входа и проксирует операции регистрации и входа, что упрощает настройку CORS и сокращает число адресов, известных браузеру.

2.4.2 Проектирование микросервиса бронирований и каталога

Микросервис deal-service объединяет публичный API каталога, API клиентских бронирований, административный API и интеграции с auth-service и payment-service. Диаграмма классов приведена на рисунке 15.

Внутренняя структура deal-service построена по многослойной схеме: транспортный слой обслуживает публичные, клиентские и административные маршруты, сервисный слой реализует правила каталога и жизненный цикл бронирований, репозитории обеспечивают доступ к схеме deal в PostgreSQL, а интеграционные клиенты и middleware связывают модуль с auth-service и payment-service. На рисунке 15 отражены основные классы и направление зависимостей между слоями при обработке типовых запросов.

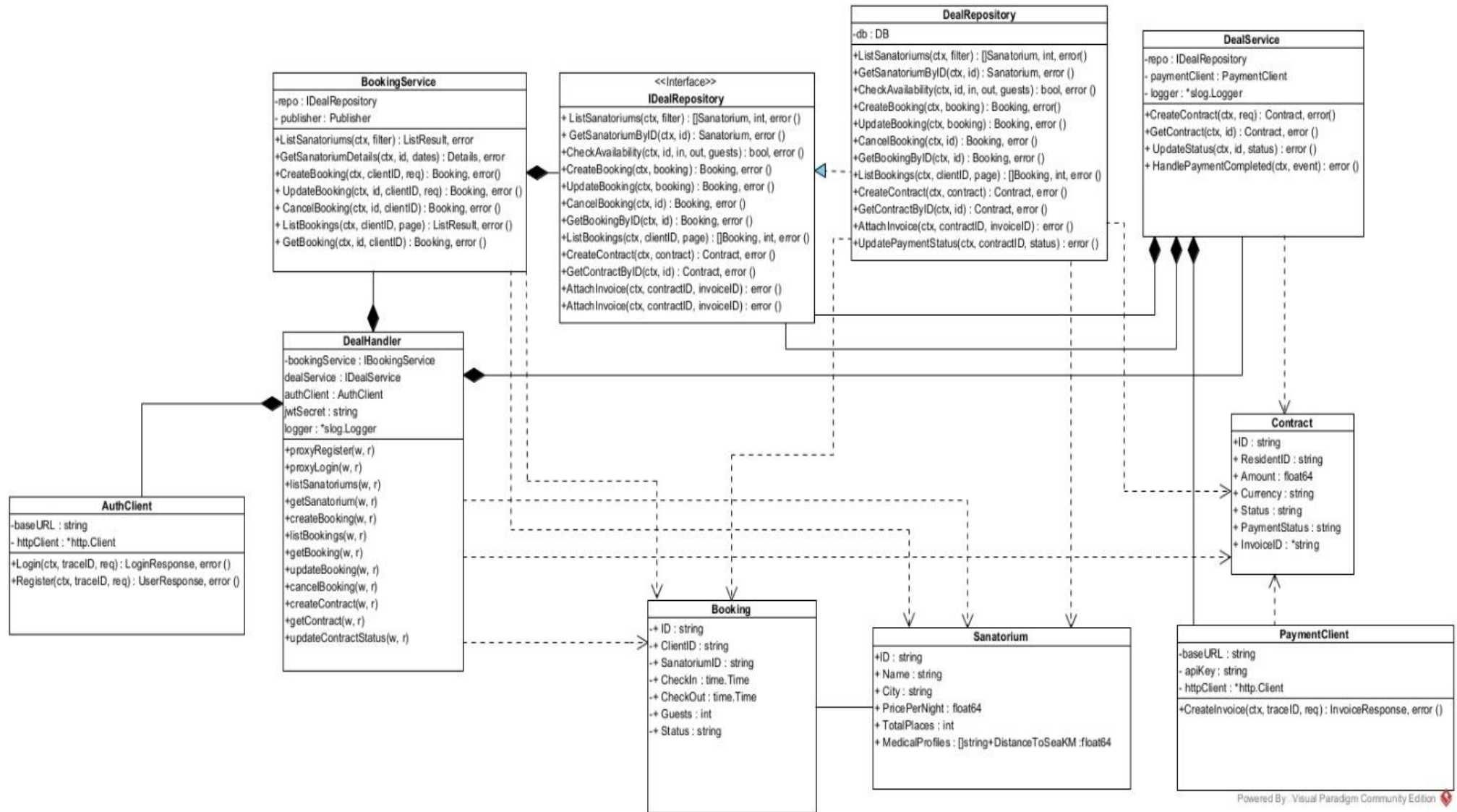


Рисунок 15 – Диаграмма классов микросервиса бронирований и каталога (deal-service)

Среди проектируемых компонентов выделяются Handler пакета httptransport, обеспечивающий маршрутизацию запросов, применение middleware и формирование ответов; BookingService, реализующий жизненный цикл бронирований, проверку доступности мест, расчёт суммы к оплате и обработку события payment.completed для бронирований; SanatoriumAdminService, выполняющий CRUD-операции над каталогом санаториев и привязками медицинских профилей; BookingCatalogRepository, обеспечивающий выборку каталога с фильтрами и пагинацией; клиенты AuthClient и PaymentClient для синхронной интеграции с соседними микросервисами; а также AuthMiddleware, осуществляющий проверку JWT, извлечение роли (client, admin) и идентификатора субъекта. Внешнее взаимодействие с deal-service в публичном и клиентском контуре реализуется через REST API: Handler маршрутизирует запросы к сервисному слою, а защищенные операции дополнительно проходят проверку JWT в AuthMiddleware. Перечень основных методов, доступных неавторизованному пользователю и клиенту с ролью client, приведен в таблице 6.

Таблица 6 – Основные методы REST API микросервиса бронирований (публичный и клиентский контур)

Метод	Точка доступа	Описание	Параметры
POST	/api/auth/register	Проксирование регистрации в auth-service	Body: { email, password, full_name }
POST	/api/auth/login	Проксирование входа в auth-service	Body: { email, password }
GET	/api/sanatoriums	Список санаториев с фильтрацией и пагинацией	Query: page, page_size, city, profiles, max_distance_to_sea, price_min, price_max, check_in, check_out, sort
GET	/api/sanatoriums/{id}	Детали санатория с опциональной проверкой доступности	Path: id; Query: check_in, check_out

Продолжение таблицы 6

Метод	Точка доступа	Описание	Параметры
POST	/api/bookings	Создать бронирование (клиент)	Body: { sanatorium_id, check_in, check_out, guests }
GET	/api/bookings	Получить список собственных бронирований клиента	Query: page, page_size
GET	/api/bookings/{id}	Получить бронирование по идентификатору	Path: id
PUT	/api/bookings/{id}	Изменить параметры бронирования	Path: id; Body: { check_in, check_out, guests }
DELETE	/api/bookings/{id}	Отменить бронирование	Path: id
POST	/api/v1/contracts	Оформить договор и выставить счет (служебный контур)	Body: { resident_id, room_id, manager_id, start_date, end_date, amount, currency }
GET	/api/v1/contracts/{id}	Получить договор по идентификатору	Path: id
PATCH	/api/v1/contracts/{id}/status	Обновить статус договора	Path: id; Body: { status }
GET	/health	Базовая проверка доступности	-
GET	/health/ready	Readiness-проверка с контролем БД	-
GET	/metrics	Метрики Prometheus	-
GET	/swagger/*	Swagger UI документации	-

Перечисленные методы обеспечивают публичный доступ к каталогу, аутентификацию клиента и сопровождение собственных бронирований; служебные точки поддерживают эксплуатацию и документирование API. Операции административного контура вынесены в отдельную группу маршрутов с префиксом /api/admin и доступны только пользователю с ролью admin; их состав приведен в таблице 7.

Таблица 7 – Методы REST API административного контура (deal-service)

Метод	Точка доступа	Назначение
GET	/api/admin/bookings	Реестр бронирований с фильтрами
DELETE	/api/admin/bookings/{id}	Отмена бронирования администратором
POST	/api/admin/bookings/{id}/checkout	Выставление счета (служебно)
POST	/api/admin/bookings/{id}/pay	Подтверждение оплаты (служебно)
GET	/api/admin/sanatoriums	Список санаториев (администратор)
POST	/api/admin/sanatoriums	Создание санатория
PUT	/api/admin/sanatoriums/{id}	Изменение данных санатория

Проверка доступности при создании и изменении бронирования выполняется в два этапа: на уровне приложения проверяются корректность дат и числа гостей, на уровне SQL – суммарная занятость санатория по пересекающимся подтвержденным бронированиям относительно поля total_places.

2.4.3 Проектирование микросервиса оплаты

Микросервис payment-service отвечает за счета и платежи. Внутренние точки доступа защищаются заголовком X-Internal-API-Key; клиентский браузер к сервису не обращается. Данные платёжного контура размещаются в схеме payment реляционной СУБД PostgreSQL и изолируются от схем auth и deal в соответствии с границами микросервиса. Обращение к payment-service инициируется со стороны deal-service при выставлении счёта и со стороны клиентского контура – опосредованно через единую точку входа при проведении оплаты. Диаграмма классов – на рисунке 16.

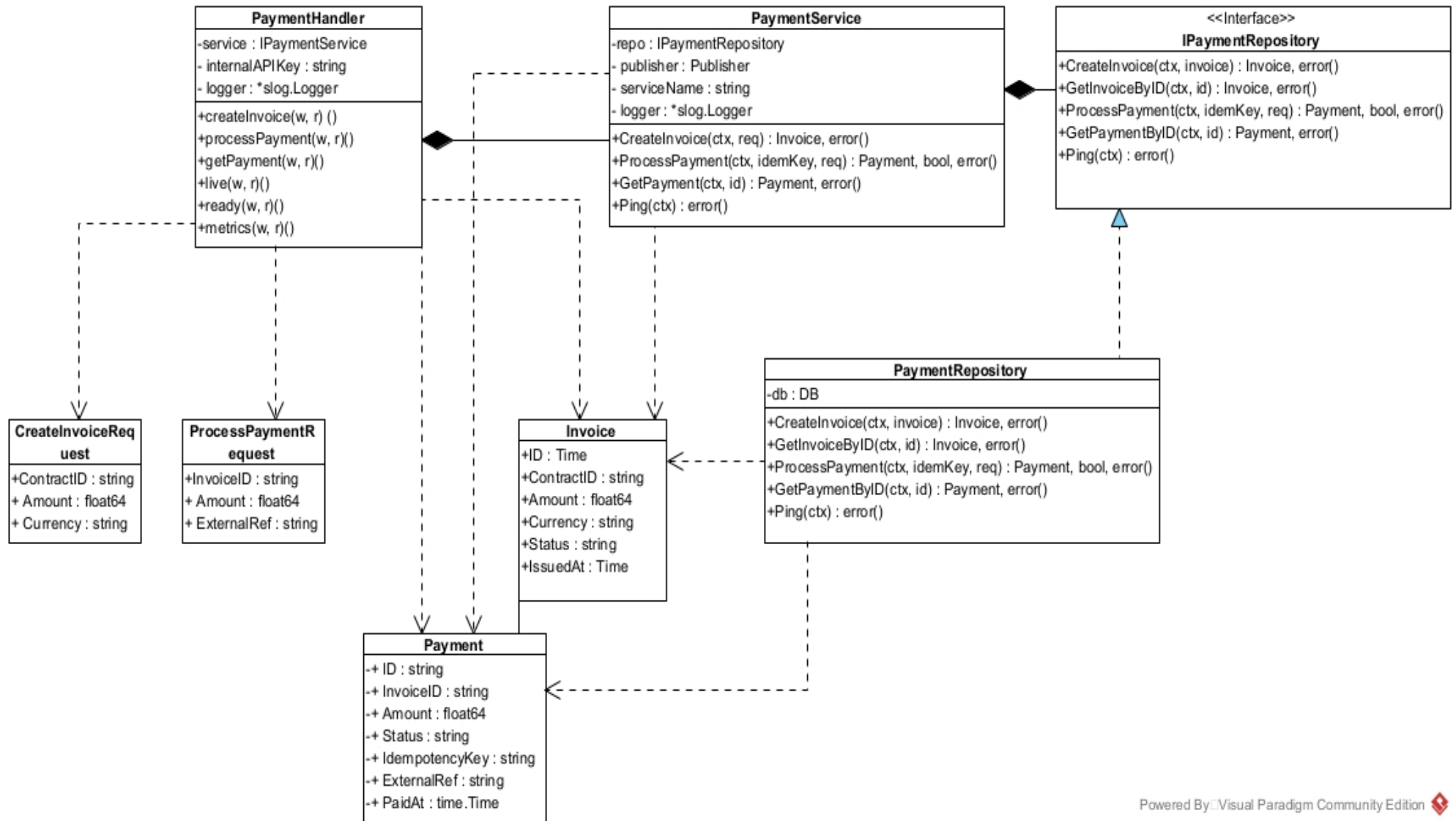


Рисунок 16 – Диаграмма классов микросервиса оплаты (payment-service)

PaymentService обеспечивает создание счета, проведение платежа в транзакции с уровнем изоляции Serializable, блокировку строки счета (SELECT FOR UPDATE), контроль совпадения суммы платежа и суммы счета, а также публикацию события payment.completed. Описанная логика PaymentService доступна смежным модулям через REST API payment-service. Создание счета выполняется по внутреннему маршруту с проверкой заголовка X-Internal-API-Key; проведение платежа и запрос сведений о платеже реализованы в версионированном API /api/v1. Перечень точек доступа и их назначение приведены в таблице 8.

Таблица 8 – Методы REST API микросервиса оплаты

Метод	Точка доступа	Описание	Параметры
POST	/internal/invoices	Создать счет по договору (только служебный вызов)	Header: X-Internal-API-Key; Body: { contract_id, amount, currency }
POST	/api/v1/payments	Провести платеж по счету	Header: Idempotency-Key; Body: { invoice_id, amount, external_ref }
GET	/api/v1/payments/{id}	Получить платеж по идентификатору	Path: id
GET	/health	Базовая проверка доступности	-
GET	/health/ready	Readiness-проверка с контролем БД	-
GET	/metrics	Метрики Prometheus	-

Как следует из таблицы 8, прямых обращений из клиентского браузера к payment-service не предусмотрено: выставление счета инициирует deal-service, а операция оплаты для пользователя проходит через маршруты /api/bookings/{id}/checkout и /api/bookings/{id}/pay с проксированием на сервис оплаты. Служебные endpoints обеспечивают мониторинг состояния микросервиса в составе Docker Compose.

2.5 Проектирование межсервисного взаимодействия

Взаимодействие между микросервисами разделено на синхронное и асинхронное в соответствии с характером операции [14].

Синхронные вызовы по REST применяются, когда инициатору необходим немедленный результат:

- deal-service – auth-service при регистрации и входе;
- deal-service – payment-service при выставлении счета (POST /internal/invoices);
- клиент – deal-service при всех пользовательских сценариях каталога, бронирований и оплаты.

Асинхронный обмен событиями через NATS используется для слабосвязанного оповещения:

- deal-service публикует booking.confirmed и booking.cancelled при изменении статуса бронирования;
- payment-service публикует payment.completed после успешной транзакции оплаты;
- deal-service обрабатывает payment.completed и обновляет payment_status у бронирования (или договора при служебном сценарии).

Формат сообщения унифицирован: конверт содержит идентификатор события, тип, метку времени, источник, trace_id и полезную нагрузку. Такой подход упрощает расширение числа подписчиков без изменения контракта публикации [15].

Сквозной клиентский сценарий «бронирование-счёт-оплата» сочетает синхронные REST-вызовы и асинхронное оповещение через брокер сообщений. После создания бронирования клиент инициирует выставление счёта: deal-service рассчитывает сумму проживания, обращается к payment-service по внутреннему API и сохраняет идентификатор счёта в записи бронирования. Проведение оплаты выполняется с передачей ключа идиempotentности; payment-service в транзакции фиксирует платёж и публикует событие payment.completed, после чего deal-service обновляет платёжный статус бронирования.

На рисунке 17 показана диаграмма последовательности основного клиентского сценария «бронирование-счет-оплата».

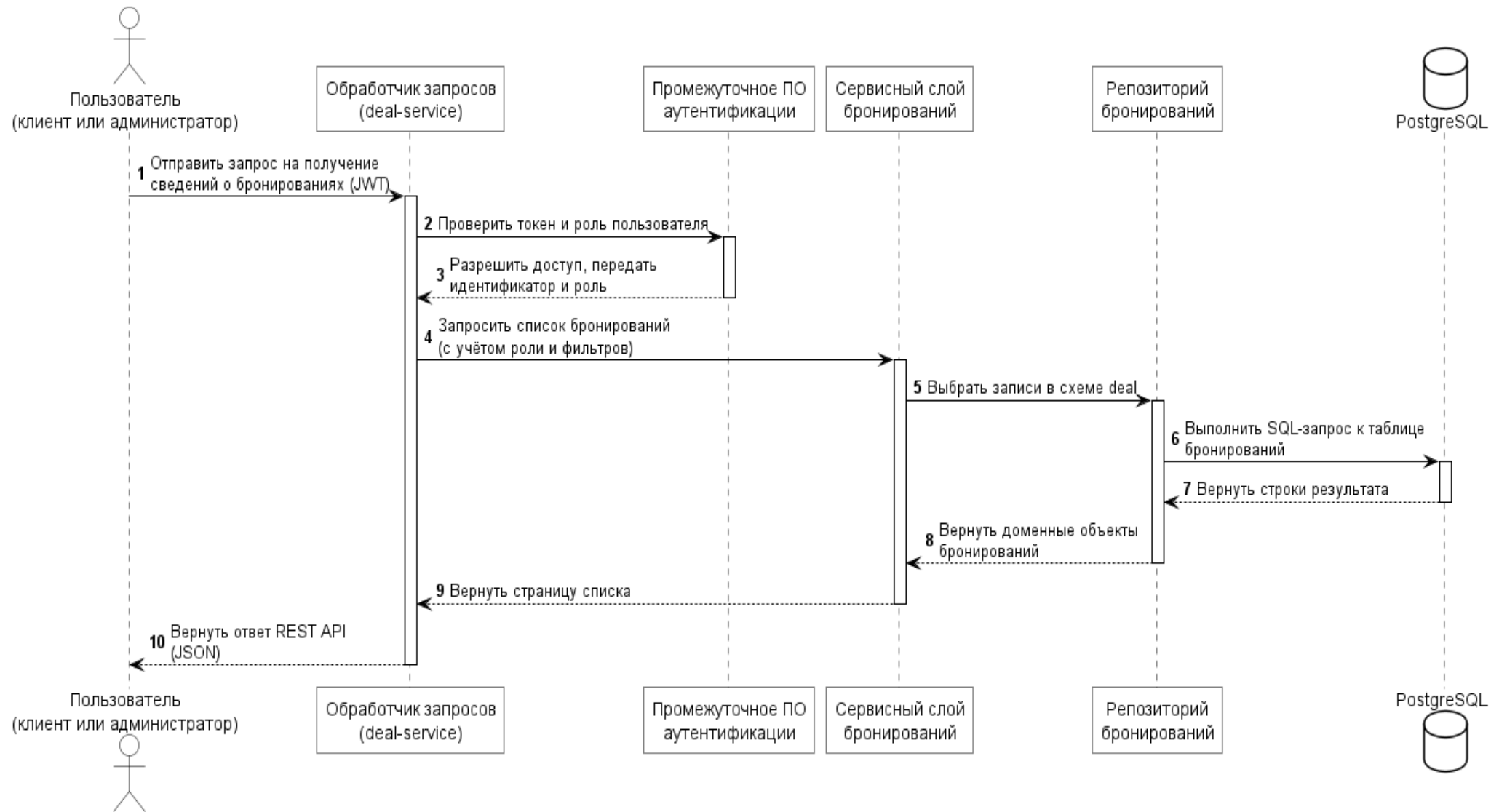


Рисунок 17 – Диаграмма последовательности сценария бронирования и оплаты

На диаграмме показан сквозной сценарий оформления и оплаты бронирования: клиент обращается к deal-service, который при выставлении счета синхронно вызывает payment-service и сохраняет идентификатор счета в записи бронирования; при проведении оплаты запрос с ключом идемпотентности повторно проходит через deal-service к payment-service, где в транзакции фиксируется платеж. Завершение оплаты асинхронно передается событием payment.completed через NATS, после чего deal-service обновляет платежный статус бронирования без обратного блокирующего вызова к сервису оплаты. Таким образом сочетаются немедленный ответ клиенту на критичных шагах и слабосвязанное согласование состояния между микросервисами.

Форматы запросов и ответов на каждом шаге цепочки закреплены в REST-контрактах deal-service и payment-service: для защищённых операций используется JWT, при оплате — заголовок Idempotency-Key, а обращения к payment-service выполняются по внутреннему API со служебным ключом. Это обеспечивает согласованность синхронной части сценария с проектными решениями по безопасности и идемпотентности платежей.

Для микросервиса deal-service сформирована спецификация OpenAPI (Swagger). Документация доступна по маршруту /swagger/* и включает описание публичных, клиентских и административных endpoints, кодов ответов и схем тел запросов. Наличие формализованного контракта облегчает согласование фронтенда и бэкенда, а также демонстрацию системы на защите выпускной квалификационной работы

3 Реализация программных средств персонализированного подбора и бронирования санаторно-курортных услуг

На этапе реализации проектные решения, сформулированные в разделе 2, были воплощены в работающий программный комплекс. Реализация охватывает развертывание базы данных PostgreSQL, серверную часть в виде набора микросервисов на языке Go, клиентское одностраничное приложение и пользовательский интерфейс для клиентского и административного контуров. При разработке сохранена многослойная структура сервисов, согласованные контракты REST API и разделение синхронного и асинхронного взаимодействия между компонентами.

Программные артефакты организованы в едином репозитории: исходный код микросервисов и общего модуля платформы, клиентское приложение, SQL-скрипты инициализации и миграций, а также конфигурация контейнеров для локального развертывания. Такая организация обеспечивает воспроизводимость сборки, сопровождение системы и возможность независимой проверки результатов работы на защите выпускной квалификационной работы.

3.1 Реализация базы данных

Физическая база данных развернута в PostgreSQL. Структура создается скриптами инициализации из каталога `sql/postgres`: выделение схем, создание таблиц, индексов и ограничений, загрузка демонстрационных данных каталога и учетной записи администратора.

Миграционный подход реализован в виде нумерованных SQL-файлов (`00_init_schemas.sql`, `01_booking_catalog.sql`, `04_booking_payments.sql` и последующие корректирующие скрипты), что позволяет воспроизводимо поднимать окружение в Docker Compose и фиксировать эволюцию модели (например, добавление платежных полей в `deal.bookings`, поддержка `booking_id` в `payment.invoices`, расширение справочника `medical_profiles`).

При старте контейнера PostgreSQL в среде разработки автоматически выполняется монтирование каталога `sql/postgres` в `docker-entrypoint-initdb.d`, благодаря чему схемы `auth`, `deal` и `payment` создаются без ручного вмешательства.

Фрагмент реализованной физической структуры (таблицы `deal.bookings` и `payment.invoices`) иллюстрируется рисунком 18; детальные таблицы атрибутов для всех сущностей целесообразно поместить в приложение Б к пояснительной записке.

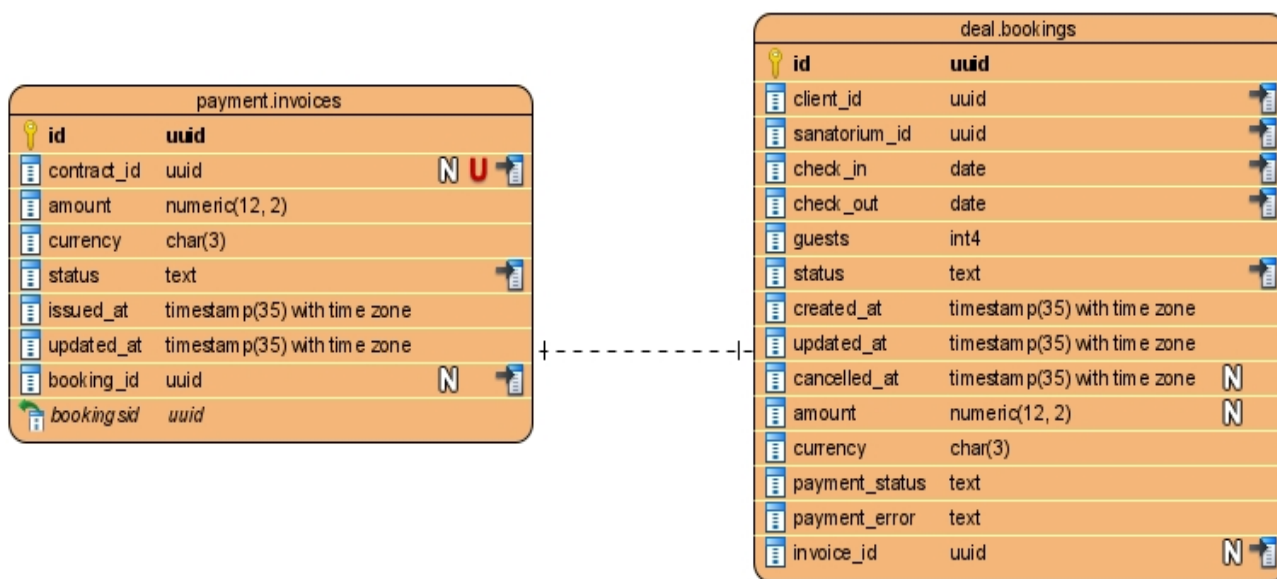


Рисунок 18 – Реализованная физическая структура данных (фрагмент)

На рисунке 18 представлены ключевые отношения платежного контура: в таблице `deal.bookings` фиксируются параметры бронирования, сумма к оплате, статус оплаты и ссылка `invoice_id` на счет в схеме `payment`; таблица `payment.invoices` хранит сумму, валюту и жизненный цикл счета, а таблица `payment.payments` – факт проведения платежа с поддержкой идемпотентности по полю `idempotency_key`. Такое разделение позволяет сервису бронирований оперировать доменными данными клиента, а сервису оплаты – обеспечивать транзакционную целостность финансовых операций.

Полная спецификация атрибутов основных таблиц (`auth.users`, `deal.sanatoriums`, `deal.bookings`, `payment.invoices`, `payment.payments`) с указанием

типов данных, ролей полей, ограничений и значений по умолчанию приведена в приложении Б.

Таблица 9 – Атрибуты отношения deal.bookings (фрагмент)

Содержание поля	Имя в БД	Тип	Ключ	Обязательность
Идентификатор бронирования	id	UUID	PK	Да
Идентификатор клиента	client_id	UUID	-	Да
Идентификатор санатория	sanatorium_id	UUID	FK	Да
Дата заезда	check_in	DATE	-	Да
Дата выезда	check_out	DATE	-	Да
Число гостей	guests	INT	-	Да
Статус бронирования	status	TEXT	-	Да
Сумма к оплате	amount	NUMERIC(12,2)	-	Нет
Статус оплаты	payment_status	TEXT	-	Да
Ссылка на счет	invoice_id	UUID	-	Нет

Отношение deal.bookings является центральным объектом хранения клиентских заявок: в нем фиксируются связь с санаторием (sanatorium_id), период проживания, число гостей и статус бронирования. Поля amount, payment_status и invoice_id поддерживают платежный сценарий – расчет суммы к оплате, отражение этапов выставления счета и оплаты, а также ссылку на запись в схеме payment без нарушения границы микросервисов на уровне прикладного кода. Остальные служебные атрибуты (метки времени, текст ошибки платежа) и ограничения целостности (проверка дат, допустимые значения статусов) задаются в SQL-скриптах и при необходимости уточняются в приложении Б.

3.2 Реализация серверной части

Серверная часть реализована на языке Go (модули auth-service, deal-service, payment-service, platform-common). В качестве HTTP-маршрутизатора используется chi; доступ к PostgreSQL – через драйвер pgx. Контейнеризация выполнена средствами Docker; оркестрация локального стенда – Docker Compose (сервисы auth-service, deal-service, payment-service, postgres, nats, frontend).

Каждый микросервис собирается с флагом `-mod=vendor`, что фиксирует зависимости и обеспечивает воспроизводимость сборки в изолированной среде CI.

Проверка JWT выполняется в deal-service и payment-service на основе общего секрета без обращения к auth-service на каждый запрос. Роли client и admin разграничивают доступ к маршрутам `/api/bookings` и `/api/admin/*`.

В разделе реализации серверной части описаны ключевые сценарии, обеспечивающие работу клиентского и служебного контуров. Регистрация и аутентификация выполняются через единую точку входа deal-service: данные регистрации проксируются в auth-service, где проверяются корректность адреса электронной почты, длина пароля (не менее восьми символов) и полное имя, после чего пароль хешируется и учётная запись сохраняется с ролью client. При входе клиенту выдаётся JWT-токен, срок действия которого задаётся параметром конфигурации `TOKEN_TTL_MINUTES`.

Персонализированный подбор санаториев реализован обработчиком `GET /api/sanatoriums`, формирующим SQL-запрос с учётом фильтров по городу, списку медицинских профилей, максимальному расстоянию до моря, ценовому диапазону, датам проживания и режиму сортировки. При передаче параметров `check_in` и `check_out` в выборку включаются только объекты с достаточной вместимостью на указанный период. Справочник медицинских профилей доступен по `GET /api/medical-profiles` и используется как в клиентской панели фильтров, так и в формах администратора.

Создание, изменение и отмена бронирования доступны аутентифицированному клиенту – владельцу заявки. При успешном создании публикуется событие `booking.confirmed`, при отмене – `booking.cancelled`; изменение параметров повторно инициирует проверку пересечения дат. Оплата оформляется в два этапа: операция `checkout` рассчитывает сумму как произведение числа ночей на цену за ночь, создаёт счёт в `payment-service` и сохраняет `invoice_id` в записи бронирования; операция `pay` принимает заголовок `Idempotency-Key`, после чего `payment-service` в транзакции с уровнем изоляции `Serializable` фиксирует платёж, публикует событие `payment.completed`, а `deal-service` переводит поле `payment_status` в значение `paid`.

В служебном контуре пользователь с ролью `admin` получает доступ к реестру бронирований и операциям CRUD над каталогом санаториев. Удаление санатория при наличии активных подтверждённых бронирований отклоняется с кодом ответа `409 Conflict`, что предотвращает нарушение ссылочной целостности с точки зрения бизнес-правил. Для контроля эксплуатации каждый микросервис предоставляет служебные точки доступа `/health`, `/health/ready` и `/metrics`; проверки готовности `auth-`, `deal-` и `payment-service` включают доступность PostgreSQL и используются в `healthcheck` контейнеров Docker Compose.

3.3 Реализация клиентской части

Клиентская часть реализована как SPA на TypeScript с библиотекой React 18 и сборщиком Vite. Маршрутизация страниц выполняется `react-router-dom`; состояние аутентификации хранится в `zustand`; запросы к API – через `axios` с перехватчиком, подставляющим JWT. Для кэширования и повторного получения данных применяется `@tanstack/react-query`.

Структура исходного кода организована по функциональным каталогам: `pages` (экранные формы), `components` (переиспользуемые элементы интерфейса), `api` (обертки над REST), `lib` и `hooks` (вспомогательная логика, локализация, форматирование). Стилизация выполнена с Tailwind CSS.

Корневой компонент `App.tsx` задает дерево маршрутов и разделяет публичный контур (каталог, карточка санатория, вход и регистрация) от защищенных разделов. Доступ клиента к разделу «Мои бронирования» и администратора к панели `/admin` контролируется компонентом `ProtectedRoute`, который проверяет наличие действующего токена и соответствие роли, закодированной в JWT. Административный интерфейс вложен в `AdminLayout` и включает вкладки реестра бронирований и управления каталогом санаториев.

Общая оболочка приложения формируется компонентами `Layout` и `Navbar`: через них реализованы навигация между экранами, отображение текущей роли пользователя и переключение языка интерфейса. Страницы каталога и детальной карточки санатория опираются на переиспользуемые элементы `CatalogFiltersPanel`, `SanatoriumCard` и `MedicalProfileChip`; раздел «Мои бронирования» использует `BookingCard` и хук `useBookingPayment`, инкапсулирующий последовательность операций `checkout` и `pay`. Состояние аутентификации сосредоточено в хранилище `authStore` на основе `zustand` и передаётся в API-слой, а загрузка и кэширование серверных данных выполняются средствами `react-query` при обращении к модулям `api/auth.ts`, `api/sanatoriums.ts`, `api/bookings.ts` и `api/admin.ts`.

Взаимодействие с серверной частью осуществляется через единый HTTP-клиент на базе `axios`: перехватчик запросов автоматически добавляет заголовок `Authorization` с JWT для защищённых операций, а все вызовы направляются на `deal-service` как на единую точку входа. Такое разделение позволяет страницам оставаться сфокусированными на представлении данных и реакции на действия пользователя, тогда как сетевые контракты, повторное получение ответов и обработка ошибок сосредоточены в API-слое и слое состояния. На диаграмме компонентов (рисунок 19) отражены связи между маршрутизацией, экранными формами, UI-компонентами, хранилищем аутентификации и модулями REST API. Диаграмма компонентов клиентской части представлена на рисунке 19.

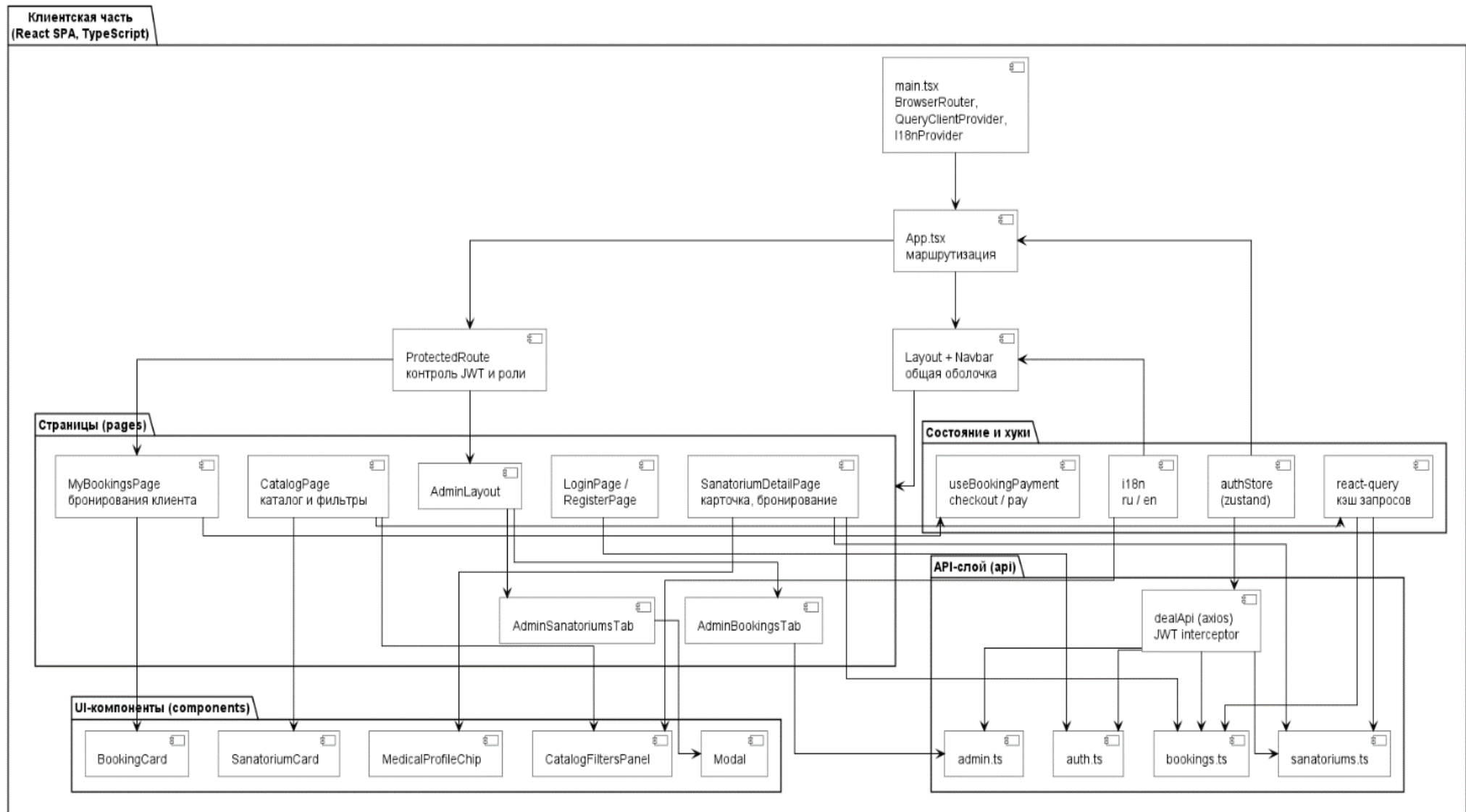


Рисунок 19 – Диаграмма компонентов клиентской части веб-приложения

На рисунке 19 отражена модульная структура клиентской части: корневой маршрутизатор и защита маршрутов, экранные формы, переиспользуемые UI-компоненты, слой обращений к REST API, а также хранилище аутентификации и кэш запросов. Для наглядного сопоставления диаграммы с исходным кодом ключевые компоненты и их функциональное назначение сведены в таблице 10.

Таблица 10 – Основные компоненты клиентской части

Компонент	Назначение
App.tsx	Корневая маршрутизация, разделение публичных и защищенных маршрутов
CatalogPage	Каталог санаториев и панель фильтров
SanatoriumDetailPage	Детали санатория и форма бронирования
MyBookingsPage	Список бронирований клиента, оплата и отмена
AdminLayout	Контейнер административного контура
AdminBookingsTab	Реестр бронирований, служебные операции оплаты
AdminSanatoriumsTab	Управление каталогом санаториев

Клиент обращается к единому базовому URL deal-service; payment-service для браузера недоступен, что соответствует спроектированной модели безопасности.

Асинхронное согласование статусов бронирования и оплаты выполняется через NATS; инфраструктурные функции логирования, метрик и работы с брокером сосредоточены в общем модуле platform-common.

3.4 Реализация пользовательского интерфейса веб-приложения персонализированного подбора и бронирования санаторно-курортных услуг

Клиентская часть представляет собой одностраничное веб-приложение с общей панелью навигации, переключением языка (русский и английский) и разграничением маршрутов для гостя, клиента и администратора.

Согласно сформированным требованиям к разработке интерфейса информационной системы персонализированного подбора и бронирования санаторно-курортных услуг был получен следующий список необходимых страниц, которые спроектированы и реализованы:

- страница входа в систему;
- страница регистрации клиента;
- страница каталога санаториев с панелью фильтров;
- страница детальной информации о санатории с формой бронирования;
- страница просмотра собственных бронирований клиента;
- административная панель – вкладка реестра бронирований;
- административная панель – вкладка управления каталогом санаториев.

Рассмотрим каждую из страниц в контексте её содержания и взаимодействия с другими экранами. Взаимодействие пользователя с системой начинается с операций аутентификации и регистрации, после чего последовательно задействуются страницы каталога, детальной информации о санатории и управления бронированиями; для сотрудников предусмотрены дополнительные административные разделы с расширенным набором функций. Для доступа к защищённым разделам неавторизованный пользователь переходит на страницу входа. Интерфейс страницы входа представлен на рисунке 20.

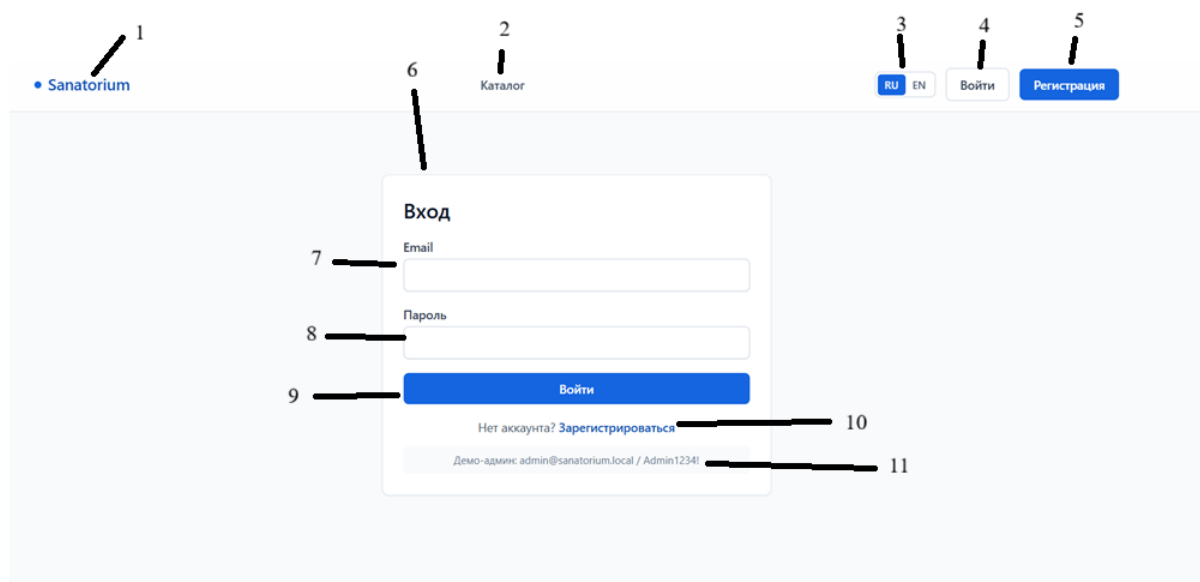


Рисунок 20 – Страница входа в систему

На рисунке 20 отображаются логотип и наименование приложения Sanatorium (1); ссылка перехода в каталог (2); переключатель языка RU/EN (3); кнопка «Войти» в общей панели (4); кнопка «Регистрация» в общей панели (5). В центральной форме размещены заголовок «Вход» (6); поле адреса электронной почты (7); поле пароля (8); кнопка подтверждения входа (9); ссылка «Нет аккаунта? Зарегистрироваться» (10); информационный блок с демонстрационными учётными данными администратора (11).

Перед отправкой формы выполняется клиентская проверка заполненности полей и формата адреса электронной почты; при подтверждении входа запрос направляется через deal-service в auth-service, после успешного ответа JWT сохраняется в хранилище клиента и выполняется переход в каталог для роли client или в административный раздел для роли admin.

Пользователь, не имеющий учётной записи, может перейти к регистрации по ссылке в форме входа либо по кнопке «Регистрация» в общей панели навигации.

Регистрация нового клиента выполняется на отдельной странице. Интерфейс страницы регистрации представлен на рисунке 21.

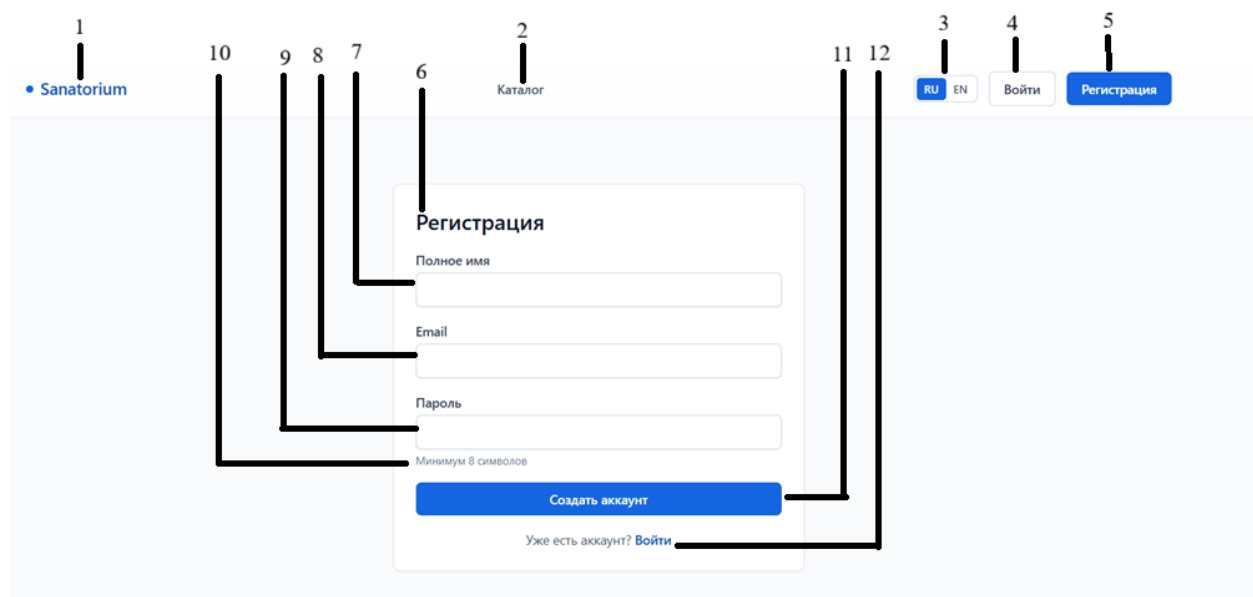


Рисунок 21 – Страница регистрации клиента

На рисунке 21 пользователь видит общую панель навигации: логотип Sanatorium (1); ссылку «Каталог» (2); переключатель языка (3); кнопку «Войти» (4); выделенную кнопку «Регистрация» (5). В форме регистрации размещены заголовок (6); поле полного имени (7); поле электронной почты (8); поле пароля (9); подсказка о минимальной длине пароля (10); кнопка «Создать аккаунт» (11); ссылка «Уже есть аккаунт? Войти» (12). Перед отправкой запроса выполняется клиентская проверка заполненности полей, формата адреса и длины пароля.

После успешного создания учётной записи запрос регистрации, направленный через deal-service в auth-service, завершается автоматическим входом: клиент получает JWT, сессия сохраняется в хранилище, и выполняется переход в каталог без повторного ввода данных. Это сокращает число шагов при первом обращении к системе и позволяет сразу перейти к подбору санатория.

После входа или при первом обращении к системе клиент работает с каталогом санаториев. Интерфейс страницы каталога представлен на рисунке 22.

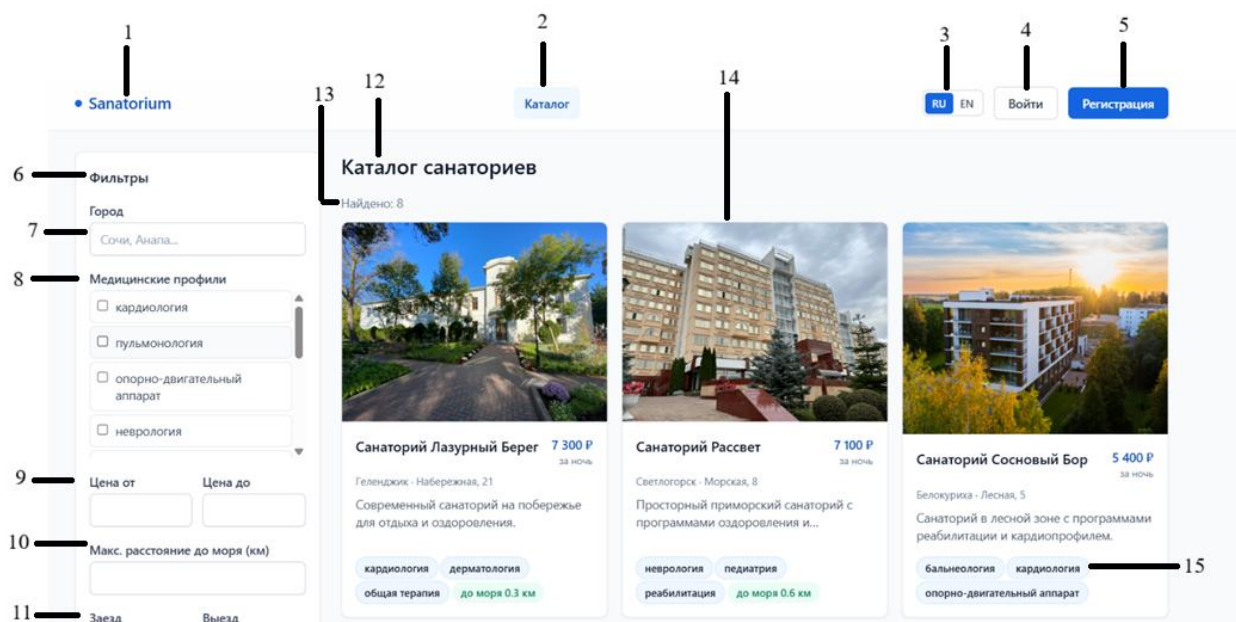


Рисунок 22 – Страница каталога санаториев с панелью фильтров

На рисунке 22 в верхней части экрана расположены логотип Sanatorium (1); ссылка «Каталог» (2); переключатель языка (3); кнопки «Войти» и «Регистрация» (4)–(5). В левой колонке размещена панель «Фильтры» (6) с полем города (7); списком медицинских профилей с флажками выбора (8); полями ценового диапазона «Цена от» и «Цена до» (9); полем максимального расстояния до моря (10); полями дат заезда и выезда (11). В основной области отображаются заголовок «Каталог санаториев» (12); счётчик найденных объектов (13); сетка карточек санаториев (14) с изображением, наименованием, ценой за ночь, адресом, кратким описанием и метками медицинских профилей (15).

Изменение параметров фильтрации инициирует запрос к REST API deal-service; список карточек обновляется с учётом города, медицинских профилей, ценового диапазона, расстояния до моря и выбранного периода проживания. Таким образом экран каталога объединяет персонализированный подбор и переход к детальному просмотру выбранного объекта.

Для перехода к подробному описанию выбранного объекта пользователь открывает карточку санатория в каталоге. Интерфейс страницы детальной информации представлен на рисунке 23.

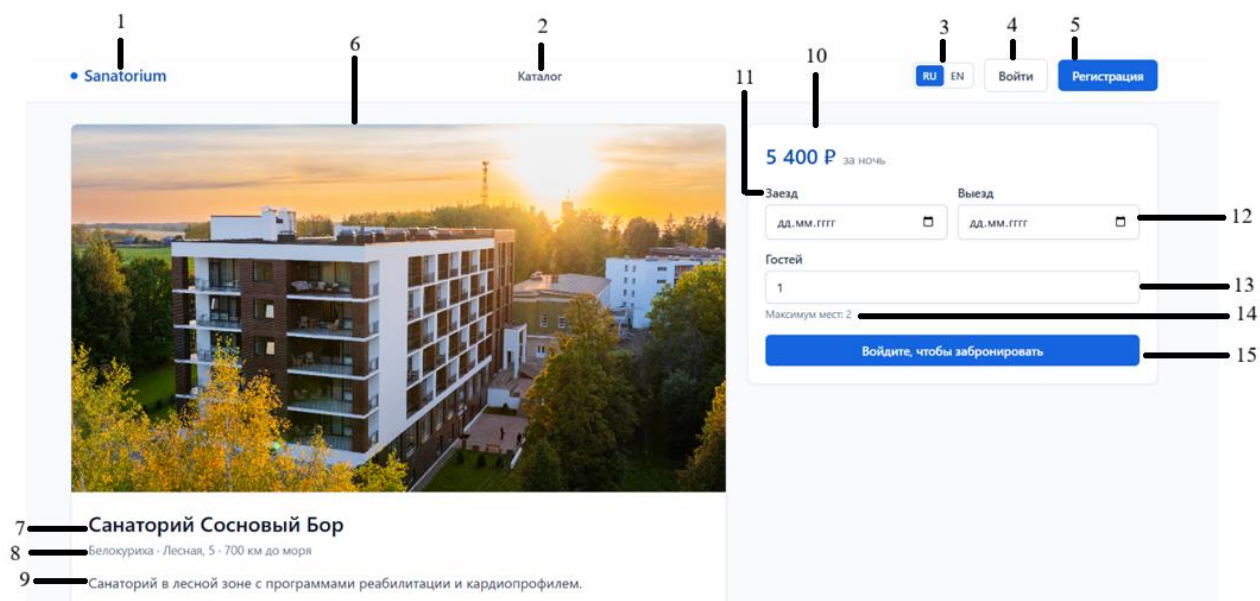


Рисунок 23 – Страница детальной информации о санатории

На рисунке 23 в общей панели отображаются логотип (1); ссылка «Каталог» (2); переключатель языка (3); кнопки входа и регистрации (4)-(5). В основной части страницы размещены фотография санатория (6); наименование объекта (7); строка с городом, адресом и расстоянием до моря (8); краткое описание (9). В правой колонке находится панель бронирования: цена за ночь (10); поле даты заезда (11); поле даты выезда (12); поле числа гостей (13); подсказка о максимальной вместимости (14); кнопка «Войдите, чтобы забронировать» для неавторизованного пользователя (15).

Авторизованный клиент управляет своими заявками в разделе «Мои бронирования». Интерфейс раздела представлен на рисунке 24.

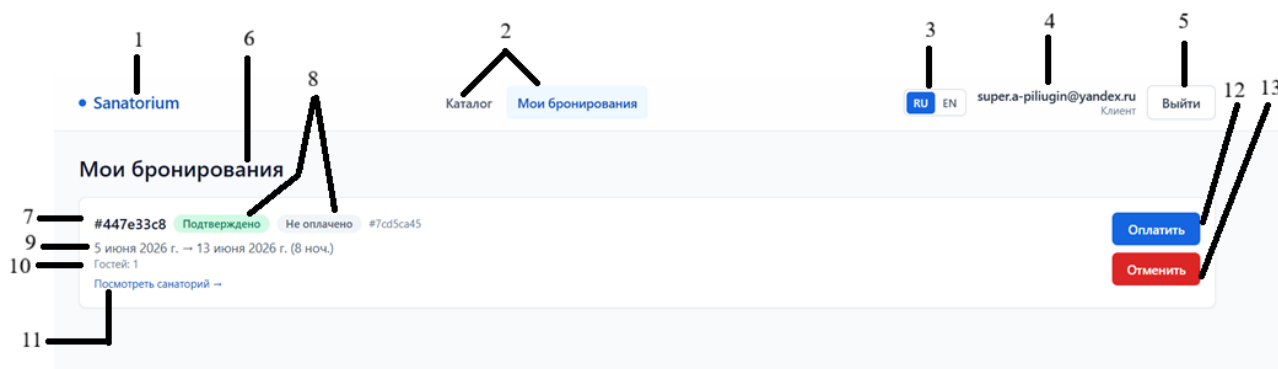


Рисунок 24 – Раздел просмотра и оплаты собственных бронирований клиента

На рисунке 24 показаны логотип Sanatorium (1); пункты навигации «Каталог» и «Мои бронирования» (2); переключатель языка (3); адрес электронной почты и роль клиента (4); кнопка выхода (5); заголовок раздела (6). В карточке бронирования отображаются наименование санатория и идентификатор заявки (7); статусы «Подтверждено» и «Не оплачено» (8); период проживания и число ночей (9); число гостей (10); ссылка «Посмотреть санаторий» (11); кнопка «Оплатить» (12); кнопка «Отменить» (13).

Пользователь с ролью admin после входа перенаправляется в служебную панель. По умолчанию открывается вкладка реестра бронирований. Интерфейс вкладки представлен на рисунке 25.

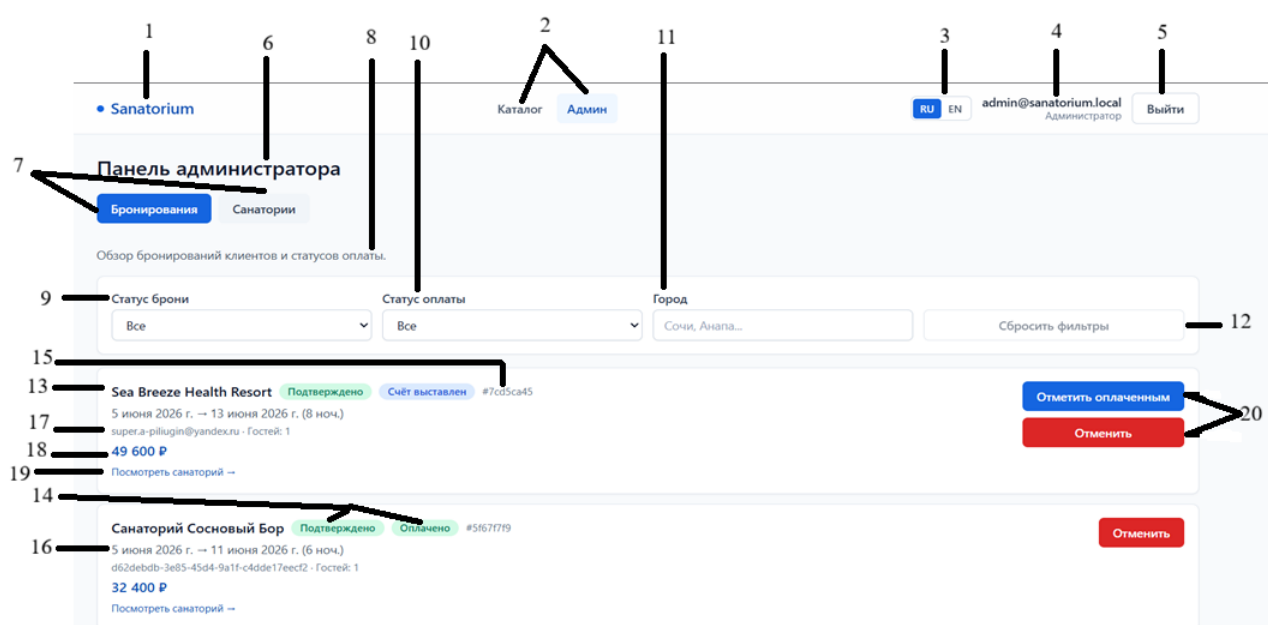


Рисунок 25 – Административный реестр бронирований

На рисунке 25 в общей панели размещены логотип (1); ссылки «Каталог» и «Админ» (2); переключатель языка (3); учётная запись администратора (4); кнопка выхода (5). Ниже расположены заголовок «Панель администратора» (6); вкладки «Бронирования» и «Санатории» (7); пояснительный текст раздела (8). Панель фильтров включает поля «Статус брони» (9), «Статус оплаты» (10), «Город» (11) и кнопку «Сбросить фильтры» (12). В карточке бронирования отображаются наименование санатория (13); статусы заявки и оплаты (14); идентификатор бронирования (15); даты проживания (16); контакт клиента и

число гостей (17); сумма к оплате (18); ссылка просмотра санатория (19); кнопки «Отметить оплаченным» и «Отменить» (20).

Управление справочником санаториев выполняется на соседней вкладке административной панели. Интерфейс вкладки представлен на рисунке 26.

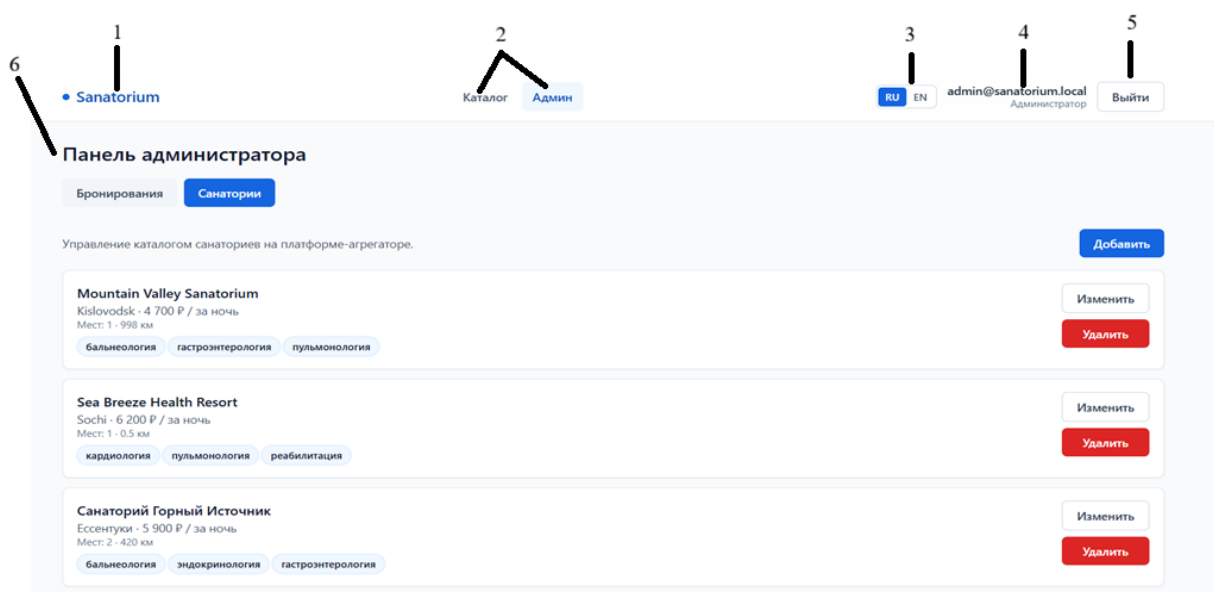


Рисунок 26 – Административное управление каталогом санаториев

На рисунке 26 показаны основные элементы административной вкладки управления каталогом: логотип приложения (1); навигационные ссылки «Каталог» и «Админ» (2); переключатель языка RU/EN (3); блок учётной записи администратора (4) и кнопка выхода (5). Заголовок «Панель администратора» (6) обозначает текущий служебный режим, в рамках которого выполняются операции ведения справочника санаториев.

Локализация подписей интерфейса и медицинских профилей согласована со справочником slug-имён в базе данных, что обеспечивает единообразие персонализированного подбора на клиентской панели фильтров и в административных формах при сохранении неизменных контрактов REST API.

4 Тестирование программных средств персонализированного подбора и бронирования санаторно-курортных услуг

В рамках верификации разработанной информационной системы проведены функциональное тестирование методом черного ящика, нагрузочное тестирование и статический анализ метрик кода. Основной акцент сделан на серверную часть микросервиса бронирований и каталога (deal-service), реализующую критичный сценарий создания бронирования; сквозной пользовательский поток проверен под нагрузкой, сложность центрального метода сервисного слоя оценена количественными метриками.

4.1 Обоснование выбора видов тестирования и метрик кода

Тестирование программного обеспечения является неотъемлемой частью жизненного цикла разработки и направлено на проверку соответствия реализованного продукта заявленным функциональным и нефункциональным требованиям, а также на выявление дефектов до ввода системы в эксплуатацию [16].

Для распределенных приложений, построенных в стиле микросервисной архитектуры, применяются следующие виды тестирования.

Модульное тестирование проверяет отдельные функции и методы в изоляции от остальных компонентов [17]. Интеграционное тестирование верифицирует взаимодействие модулей внутри сервиса и обмен данными между микросервисами [18]. Функциональное тестирование подтверждает соответствие поведения системы спецификации с точки зрения внешнего наблюдателя (метод «черного ящика»). Нагрузочное тестирование оценивает время отклика и устойчивость приложения при имитации параллельной работы пользователей [18]. Полный листинг модульных тестов приведен в приложении И.

В рамках настоящей работы в качестве основного объекта детального анализа выбран метод BookingService.CreateBooking микросервиса deal-service, поскольку он концентрирует проверки обязательных идентификаторов,

валидацию дат и числа гостей, обращение к репозиторию и публикацию события подтверждения бронирования. Сквозные сценарии с участием клиентского веб-приложения, сервиса аутентификации и сервиса оплаты отражены в нагрузочном тестировании. Исходный текст метода CreateBooking приведен в приложении Е (листинг Е.1).

Для функциональной верификации применена техника построения функциональной диаграммы (cause-effect graph) и сокращенной таблицы решений. Для оценки эксплуатационных характеристик использован инструмент к6. Для количественной оценки сложности и сопровождаемости кода рассчитаны метрики объема (SLOC), Холстеда, цикломатической сложности по Маккейбу, метрики Чепина и объектной метрики WMC [19]; результаты автоматизированной проверки цикломатической сложности получены с помощью gosyclo.

Среда и инструменты тестирования сведены в таблице 11.

Таблица 11 – Среда и инструменты тестирования

Компонент	Назначение
Docker Compose	Развертывание микросервисов auth-service, deal-service, payment-service, PostgreSQL, NATS и клиентского SPA
Инструмент к6	Имитация регистрации, входа, просмотра каталога и создания бронирования под нагрузкой
gosyclo	Автоматизированный подсчет цикломатической сложности по файлу booking.go

Перечисленные средства обеспечивают воспроизводимость проверок на локальном стенде, идентичном конфигурации Docker Compose, используемой при разработке и демонстрации системы. Docker Compose задает единое окружение для функциональной и нагрузочной верификации сквозных

сценариев; k6 применяется для имитации параллельной работы пользователей; gosucl – для автоматизированного расчета цикломатической сложности при статическом анализе кода. На этой основе далее рассматривается функциональное тестирование сценария создания бронирования.

4.2 Функциональное тестирование сценария создания бронирования

Функциональное тестирование проводилось методом черного ящика с применением техники функциональных диаграмм и таблиц решений. Объектом тестирования является сценарий CreateBooking – создание бронирования санатория на уровне сервисного слоя.

4.2.1 Причины и следствия

В рамках анализа сценария выделены 7 входных условий (причин) и 5 возможных результатов (следствий). Полные формулировки приведены в таблицах 12 и 13.

Таблица 12 – Причины (входные условия) для сценария CreateBooking

ID	Формулировка
П1	client_id заполнен (не пустая строка)
П2	sanatorium_id заполнен (не пустая строка)
П3	check_in задана (дата въезда указана, не нулевое значение)
П4	check_out задана (дата выезда указана, не нулевое значение)
П5	check_out > check_in (корректный диапазон дат)
П6	guests > 0 (количество гостей положительное)
П7	Репозиторная проверка пройдена (санаторий существует, есть свободные места, даты доступны)

Причины в таблице 12 описывают входные условия и проверки, выполняемые при создании бронирования на сервисном слое. Для построения функциональной диаграммы и таблицы решений необходимо зафиксировать возможные исходы обработки запроса; они оформлены как следствия и приведены в таблице 13. Каждое следствие соответствует определенной

комбинации выполнения или невыполнения условий П1 – П7 (ошибка валидации, доменная ошибка репозитория либо успешное создание записи со статусом confirmed).

Таблица 13 – Следствия (выходные результаты) для сценария CreateBooking

ID	Формулировка
C1	Ошибка client_id and sanatorium_id are required
C2	Ошибка guests must be greater than zero
C3	Ошибка invalid booking date range
C4	Доменная ошибка репозитория (sanatorium not found / guests exceed capacity / not available)
C5	Успешное создание бронирования (status = confirmed)

Между причинами действует ограничение E1: если П3 = 0 или П4 = 0 (хотя бы одна из дат не задана), то П5 не вычисляется. П5 имеет смысл только при П3 = 1 и П4 = 1 одновременно.

4.2.2 Функциональная диаграмма

На основе таблиц 12 и 13 построена функциональная диаграмма, отражающая связи между причинами, ограничением E1 и следствиями (см. рисунок 27).

Узлы диаграммы соответствуют этапам проверки в методе CreateBooking: совместное выполнение П1 и П2 (заполненность идентификаторов клиента и санатория), проверка П6 (положительное число гостей), валидация дат П3 – П5 с учетом ограничения E1 (условие П5 определяется только при одновременно заданных датах заезда и выезда), а также репозиторная проверка П7 (существование санатория, вместимость и доступность на выбранный период). Каждая ветвь диаграммы завершается одним из следствий: сообщением об ошибке валидации (C1 – C3), доменной ошибкой при обращении к данным (C4) либо успешным созданием бронирования (C5). Таким образом формируется функциональная диаграмма.

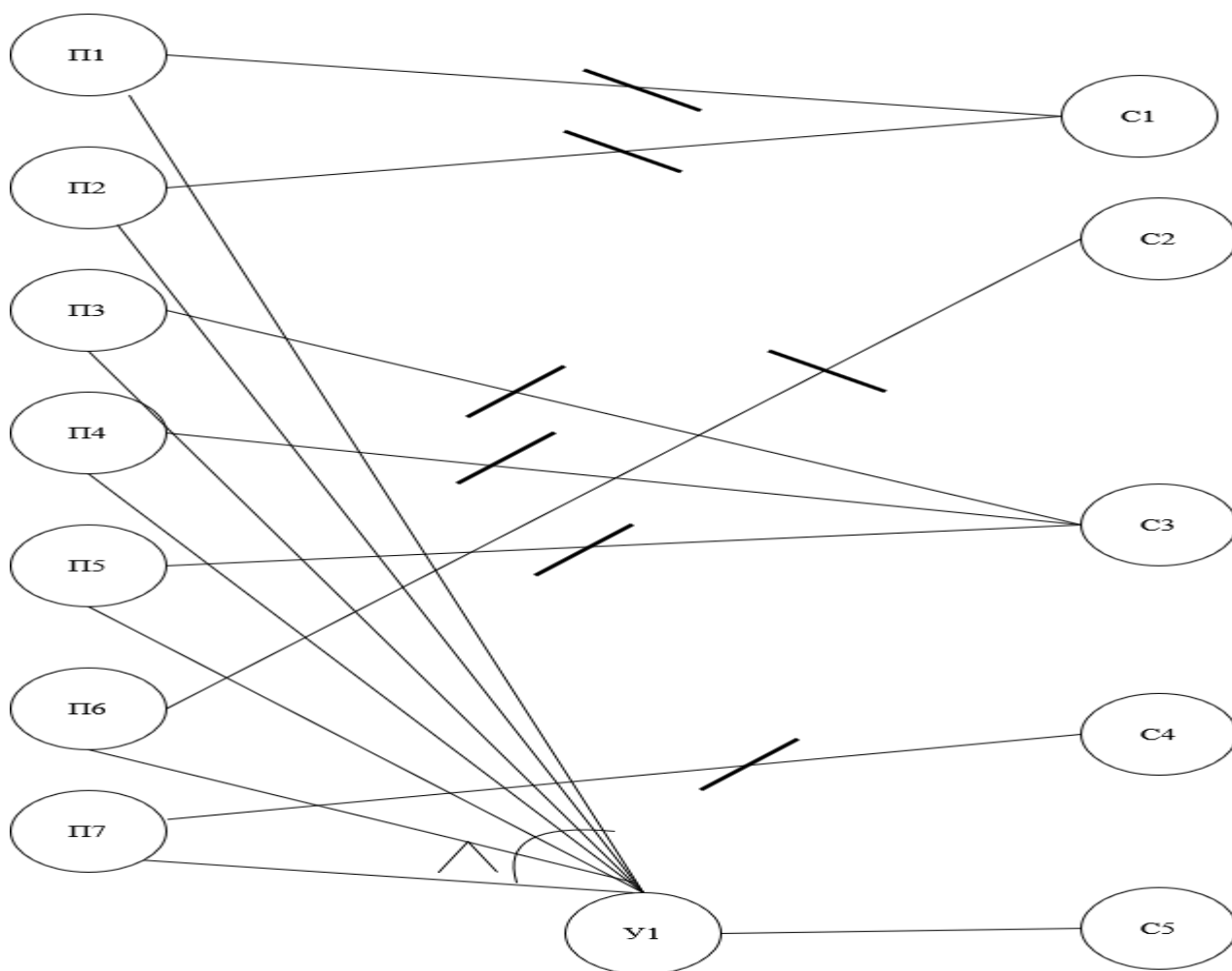


Рисунок 27 – Функциональная диаграмма для сценария CreateBooking

Порядок проверок в коде метода BookingService.CreateBooking (листинг E.1 в приложении E):

- проверка П1 ∧ П2 – обязательные идентификаторы клиента и санатория.
- проверка П6 – количество гостей.
- проверка П3, П4, П5 через ValidateBookingDateRange – валидность дат.
- проверка П7 – обращение к репозиторию (существование санатория, вместимость, доступность на запрашиваемые даты).

4.2.3 Таблица решений

Полный комбинаторный перебор 7 причин дает $2^7 = 128$ комбинаций. С учетом ограничения E1 число уникальных допустимых комбинаций равно 80. Сокращение таблицы выполняется путем объединения строк с одинаковым следствием, в которых значения части причин не влияют на результат (обозначаются «-»). Результаты приведены в таблице 14.

Таблица 14 – Сокращенная таблица решений для сценария CreateBooking

№	П1	П2	П3	П4	П5	П6	П7	С1	С2	С3	С4	С5	Покрыто комбинаций
1	0	-	-	-	-	-	-	1	0	0	0	0	40
2	1	0	-	-	-	-	-	1	0	0	0	0	20
3	1	1	-	-	-	0	-	0	1	0	0	0	10
4	1	1	0	-	-	1	-	0	0	1	0	0	4
5	1	1	1	0	-	1	-	0	0	1	0	0	2
6	1	1	1	1	0	1	-	0	0	1	0	0	2
7	1	1	1	1	1	1	0	0	0	0	1	0	1
8	1	1	1	1	1	1	1	0	0	0	0	1	1

В результате сокращения 80 допустимых комбинаций сведены к 8 правилам (сокращение в 10 раз). Правила №1-2 объединяют случаи с незаполненными идентификаторами, правило №3 – случаи с некорректным числом гостей, правила № 4 – 6 – случаи с невалидными датами (три различных варианта), правило №7 – случай с ошибкой репозитория, правило №8 – успешное выполнение.

4.2.4 Тест-кейсы

Для каждого правила таблицы 14 сформирован тест-кейс с конкретными входными данными и ожидаемым результатом (таблица 15).

Таблица 15 – Тест-кейсы по таблице решений для CreateBooking

№	Входные данные	Ожидаемый результат	Фактический результат
1	client_id="", sanatorium_id=uuid-san, check_in=2026-07-10, check_out=2026-07-15, guests=2	Ошибка client_id and sanatorium_id are required	HTTP 400 {"error": "client_id and sanatorium_id are required"} (воспроизводится через unit- тест, т.к. через API client_id берется из JWT токена)
2	client_id=uuid-cli, sanatorium_id="", check_in=2026-07-10, check_out=2026-07-15, guests=2	Ошибка client_id and sanatorium_id are required	HTTP 400 {"error": "client_id and sanatorium_id are required"}
3	client_id=uuid-cli, sanatorium_id=uuid-san, check_in=2026-07-10, check_out=2026-07-15, guests=0	Ошибка guests must be greater than zero	HTTP 400 {"error": "guests must be greater than zero"}
4	client_id=uuid-cli, sanatorium_id=uuid-san, check_in не задана, check_out=2026-07-15, guests=2	Ошибка invalid booking date range	HTTP 400 {"error": "check_in format must be YYYY-MM-DD"} (HTTP- хэндлер отклоняет запрос до вызова сервиса; через unit-тест с time.Time{} ошибка не возникает, т.к. нулевая дата меньше любой реальной check_out)
5	client_id=uuid-cli, sanatorium_id=uuid-san, check_in=2026-07-10, check_out не задана, guests=2	Ошибка invalid booking date range	HTTP 400 {"error": "check_out format must be YYYY-MM-DD"} (HTTP- хэндлер отклоняет запрос до вызова сервиса; через unit-тест с time.Time{} как check_out вернется invalid booking date range)

Продолжение таблицы 15

№	Входные данные	Ожидаемый результат	Фактический результат
6	client_id=uuid-cli, sanatorium_id=uuid-san, check_in=2026-07-15, check_out=2026-07-10, guests=2	Ошибка invalid booking date range	HTTP 400 {"error": "invalid booking date range"}
7	client_id=uuid-cli, sanatorium_id=uuid-san, check_in=2026-07-10, check_out=2026-07-15, guests=2, repo → FAIL	Доменная ошибка (sanatorium not found / guests exceed capacity / not available)	HTTP 404 {"error": "sanatorium not found"} (при несуществующем sanatorium_id)
8	client_id=uuid-cli, sanatorium_id=uuid-san, check_in=2026-07-10, check_out=2026-07-15, guests=2, repo → OK	Успешное создание бронирования (status=confirmed)	HTTP 201

Все 8 тест-кейсов охватывают каждый из выявленных сценариев выполнения метода CreateBooking, включая граничные случаи валидации входных данных и взаимодействия с репозиторием. Проверка всех тест-кейсов показала, что система функционирует штатно, согласно заложенному поведению. Несоответствий не выявлено. Поведение системы при выполнении тест-кейсов соответствует заданным требованиям к сценарию создания бронирования. Таким образом, выявлено полное соответствие ожидаемому поведению.

4.3 Нагрузочное тестирование

Для оценки поведения разработанного приложения под нагрузкой проведено нагрузочное тестирование с использованием инструмента k6. Целью тестирования являлась проверка корректности работы системы при выполнении типового пользовательского сценария, а также оценка времени отклика основных HTTP-запросов.

В качестве тестируемого сценария выбран пользовательский поток, включающий регистрацию пользователя, вход в систему, получение списка санаториев и выполнение операции создания бронирования. В рамках одного тестового прохода последовательно выполнялись действия, характерные для практического использования приложения. Сценарий нагрузки реализован средствами k6.

Нагрузка в ходе тестирования изменялась поэтапно. На первом этапе в течение 30 секунд количество виртуальных пользователей увеличивалось до 10. Далее в течение 1 минуты нагрузка возрастала до 50 виртуальных пользователей, после чего в течение еще 1 минуты поддерживалась на этом уровне. На завершающем этапе в течение 30 секунд нагрузка снижалась до 0.

В ходе тестирования заданы пороговые значения для ключевых запросов. Для запроса получения списка санаториев установлено требование: 99% запросов выполняются менее чем за 1 секунду. Для запроса создания бронирования – менее чем за 2 секунды. Для общего числа ошибочных HTTP-запросов задано ограничение не более 5%.

Испытания выполнялись на развёрнутом в Docker комплексе микросервисов; скрипт k6 для каждого виртуального пользователя проверяет коды HTTP-ответов и корректность данных на шагах сценария. Для запроса создания бронирования задаются уникальные даты проживания, что снижает влияние параллельной нагрузки на результаты прогона.

Результат выполнения нагрузочного тестирования представлен на рисунке 28.

■ THRESHOLDS

```

http_req_duration{name:create_booking}
✓ 'p(99)<2000' p(99)=204.91ms

http_req_duration{name:list_sanatoriums}
✓ 'p(99)<1000' p(99)=89.52ms

http_req_failed
X 'rate<=0.05' rate=20.26%

```

■ TOTAL RESULTS

```

checks_total.....: 43218 240.095456/s
checks_succeeded...: 86.48% 37379 out of 43218
checks_failed.....: 13.51% 5839 out of 43218

```

```

✓ register status is 200/201
✓ login status is 200
✓ login has access_token
✓ list status is 200
✓ list has items
X create booking status is 200/201
  ↳ 18% — ✓ 1364 / X 5839

```

HTTP

```

http_req_duration.....: avg=147.3ms min=1.42ms med=60.95ms max=1.03s p(90)=407.16ms p(95)=486.71ms
{ expected_response:true }...: avg=178.84ms min=1.42ms med=112.92ms max=1.03s p(90)=432.89ms p(95)=511.55ms
{ name:create_booking }.....: avg=24.25ms min=2.79ms med=14.43ms max=466.1ms p(90)=48.19ms p(95)=71.85ms
{ name:list_sanatoriums }....: avg=15.04ms min=1.42ms med=9.04ms max=515.1ms p(90)=31.57ms p(95)=46.43ms
http_req_failed.....: 20.26% 5839 out of 28812
http_req_failed.....: 20.26% 5839 out of 28812
http_reqs.....: 28812 160.063637/s

```

EXECUTION

```

iteration_duration.....: avg=792.14ms min=317.02ms med=789.59ms max=2.11s p(90)=1.23s p(95)=1.41s
iterations.....: 7203 40.015909/s
vus.....: 1 min=1 max=50
vus_max.....: 50 min=50 max=50

```

Рисунок 28 – Результат нагрузочного тестирования с использованием k6

По результатам тестирования установлено, что проверки корректности регистрации, аутентификации и получения списка санаториев были пройдены в полном объеме: по каждому из этих шагов зафиксировано 3 488 успешных выполнений из 3 488. Операция создания бронирования завершилась с частичными ошибками: из 3 488 попыток успешно выполнено 2 316, что составляет 66,4%. В совокупности из 20 928 проверок (checks) успешно пройдено 19 756 (94,40%), доля неуспешных проверок составила 5,60%.

Среднее время выполнения HTTP-запросов по всем операциям составило 358,71 мс, медианное значение – 129,59 мс, 90-й перцентиль – 1001,81 мс, 95-й перцентиль – 1109,77 мс. Для запроса получения списка санаториев среднее время отклика составило 25,44 мс, значение p(99) – 93,01 мс. Для запроса создания бронирования среднее время отклика составило 40,99 мс, значение p(99) – 135,28 мс.

Заданные пороговые значения по времени отклика выполнены со значительным запасом: для запроса получения санаториев p(99) = 93 мс при

установленном ограничении менее 1000 мс, для запроса создания бронирования $p(99) = 135$ мс при установленном ограничении менее 2000 мс. Вместе с тем пороговое значение по доле ошибочных HTTP-запросов не было достигнуто: показатель `http_req_failed` составил 8,40% при допустимом значении не более 5%. Отдельно отмечается, что доля неуспешных проверок кб (5,60%) и доля неуспешных HTTP-запросов (8,40%) отражают разные уровни агрегации: проверки фиксируют ожидаемые коды ответа по каждому шагу сценария, тогда как `http_req_failed` учитывает только HTTP-статусы, не относящиеся к классу 2xx.

Источником ошибок по созданию бронирования являются конфликты бизнес-правил при параллельной нагрузке: санаторий с ограниченной вместимостью уже занят на запрашиваемые даты, и сервис корректно отклоняет повторные попытки. Это поведение является штатным с точки зрения логики приложения.

Таким образом, результаты нагрузочного тестирования показали, что приложение обеспечивает высокую скорость отклика и успешно обрабатывает сценарии регистрации, аутентификации и просмотра данных без ошибок. Превышение порога по доле ошибочных HTTP-запросов обусловлено особенностями тестового сценария и может быть устранено путем расширения тестовых данных либо разделения временных диапазонов бронирования между виртуальными пользователями.

4.4 Оценка качества программного кода

Статический анализ метрик кода выполнен для метода `BookingService.CreateBooking` – центрального метода сервисного слоя, реализующего сценарий создания бронирования. Используются метрики объёма, Холстеда, цикломатической сложности и Чепина. Метод расположен в файле `src/go-microservices/deal-service/internal/service/booking.go`. Исходный текст приведен в приложении Е (листинг Е.1).

4.4.1 Метрики размера программы

Метрики размера программы рассматриваются как первый этап статического анализа качества кода серверной части. Показатели объёма исходного текста позволяют сопоставить масштаб реализации с цикломатической сложностью и метриками Холстеда, рассчитанными для того же фрагмента, и тем самым получить целостное представление о трудоёмкости сопровождения критичного сценария создания бронирования. Объектом измерения выбран метод `BookingService.CreateBooking` микросервиса `deal-service`, поскольку он концентрирует основную бизнес-логику оформления заявки: проверку обязательных идентификаторов, валидацию дат и числа гостей, обращение к репозиторию и публикацию доменного события. Подсчёт выполнен по исходному тексту метода в файле `booking.go` без учёта пустых строк; результаты сведены в таблице 16.

Таблица 16 – Метрики размера метода `CreateBooking`

Метрика	Значение
Общее число строк кода (без пустых)	31
Число строк-комментариев	0

Полученное значение объёма (31 строка исполняемого кода) указывает на компактную реализацию метода при сохранении полного набора проверок, описанных в функциональном тестировании. Отсутствие строк-комментариев в данном фрагменте не снижает читаемость: поведение метода выражается через именованные ошибки (`ErrInvalidGuests`, `ErrInvalidDateRange`) и типизированные структуры входных данных, которые задают контракт операции без дополнительных пояснений в тексте исходного кода.

4.4.2 Метрики Холстеда

Метрики Холстеда оценивают сложность программы через словарь и частоту использования операторов и операндов. Расчет выполнен по

стандартным формулам метрики Холстеда для исходного текста метода CreateBooking; сводные показатели приведены в таблице 17.

В расчёте различаются уникальные и суммарные показатели: $n1$ и $n2$ фиксируют разнообразие языковых конструкций и используемых сущностей, а $N1$ и $N2$ – их фактическую частоту в теле метода. На этой основе определяются производные величины: длина и объём программы (V), отражающие информационную насыщенность кода, сложность (D), характеризующая структурную запутанность, и интеллектуальные усилия (E), оценивающие трудоёмкость понимания и сопровождения фрагмента. Применение метрик Холстеда к методу CreateBooking позволяет количественно дополнить показатели размера и установить, насколько компактная по числу строк реализация сопровождается умеренной или повышенной внутренней сложностью.

Таблица 17 – Сводные метрики Холстеда для метода CreateBooking

Метрика	Обозначение	Значение
Число уникальных операторов	$n1$	15
Число уникальных операндов	$n2$	30
Словарь программы	$n = n1 + n2$	45
Суммарное число операторов	$N1$	26
Суммарное число операндов	$N2$	48
Длина программы	$N = N1 + N2$	74
Объём программы	$V = N \cdot \log_2(n)$	406,40
Сложность программы	$D = (n1/2) \cdot (N2/n2)$	12,00
Интеллектуальные усилия	$E = D \cdot V$	4 876,80

Значение объема $V = 406,40$ находится в диапазоне 20 – 1000, что соответствует умеренной сложности функции.

4.4.3 Цикломатическая сложность

Цикломатическая сложность (СС) по Маккейбу определяет число независимых путей выполнения через функцию и характеризует минимальное количество тест-кейсов, необходимых для полного покрытия ветвей.

Для метода CreateBooking: CC = 6.

Это значение соответствует умеренной сложности. Метод содержит четыре ветвления: проверку обязательных идентификаторов, проверку количества гостей, валидацию диапазона дат и обработку ошибки репозитория.

Цикломатическая сложность методов типа BookingService в файле booking.go приведена на таблице 18.

Таблица 18 – Цикломатическая сложность методов BookingService

Метод	CC
ListSanatoriums	10
GetSanatoriumDetails	6
CreateBooking	6
UpdateBooking	6
CancelBooking	4
publishBookingEvent	4
ListBookings	3
GetBooking	1
WMC (сумма)	40

По данным таблицы 18 наибольшее значение цикломатической сложности имеет метод ListSanatoriums (CC = 10) в связи с разветвленной логикой фильтрации каталога; методы CreateBooking, GetSanatoriumDetails и UpdateBooking характеризуются умеренной сложностью (CC = 6), что согласуется с результатами функционального тестирования сценария создания бронирования. Служебные операции GetBooking, ListBookings, CancelBooking и publishBookingEvent относятся к менее сложным путям выполнения. Суммарный показатель WMC = 40 отражает распределение ветвлений в пределах класса BookingService и используется при переходе к анализу функциональной сложности по метрике Чепина.

4.4.4 Метрика Чепина

Метрика Чепина дополняет цикломатический анализ оценкой функциональной сложности через роль переменных в методе: учитывается не

только число ветвлений, но и то, какие данные участвуют во входе, изменяются в ходе выполнения или управляют условиями. Для сценария создания бронирования объектом расчета выбран метод `CreateBooking`, поскольку в нем сосредоточены проверки входных параметров и обработка результата обращения к репозиторию, ранее отраженные в функциональном тестировании.

Для метода `CreateBooking` выполнена классификация переменных по четырем группам метрики Чепина: Р – входные параметры, передаваемые в метод; М – модифицируемые переменные, значение которых формируется в ходе выполнения; С – управляющие переменные, участвующие в условиях и ветвлениях; Т – внешние (в рассматриваемом методе отсутствуют). Сводка по группам приведена в таблице 19, на основании которой рассчитывается показатель Q.

Таблица 19 – Разбиение переменных по группам Чепина для метода `CreateBooking`

№	Название переменной	Группа
1	ctx	Р
2	traceID	Р
3	booking	М
4	in.ClientID	С
5	in.SanatoriumID	С
6	in.Guests	С
7	in.CheckIn	С
8	in.CheckOut	С
9	err	С

Для метода `CreateBooking` выполнена классификация переменных по четырём группам метрики Чепина: Р – входные параметры, передаваемые в метод; М – модифицируемые переменные, значение которых формируется в ходе выполнения; С – управляющие переменные, участвующие в условиях и ветвлениях; Т – внешние (в рассматриваемом методе отсутствуют). Сводка по

группам приведена в таблице 19, на основании которой рассчитывается показатель Q по формуле:

$$Q = 1 \cdot P + 2 \cdot M + 3 \cdot C + 0,5 \cdot T,$$

где P – число входных параметров;

M – число модифицируемых переменных;

C – число управляющих переменных;

T – число внешних переменных.

Для метода CreateBooking получено P = 2, M = 1, C = 6 и T = 0; при подстановке в формулу (4) значение Q составляет 22,00. Указанный результат показывает, что основная сложность метода формируется управляющими переменными валидации (группа C), что согласуется с результатами цикломатической сложности.

4.4.5 Объектная метрика WMC

Метрика WMC (Weighted Methods per Class) определяет суммарную сложность всех методов класса. В качестве веса отдельного метода принимается его цикломатическая сложность, поэтому WMC позволяет оценить не только «размер» типа по числу операций, но и степень ветвистости реализованных в нём сценариев. Показатель рассматривается совместно с метриками объёма, Холстеда и Чепина для метода CreateBooking: если последние характеризуют отдельный критичный фрагмент, то WMC даёт оценку распределения сложности по всему сервисному типу, на котором сосредоточена прикладная логика каталога и бронирований.

Для BookingService WMC = 40, что отражает концентрацию бизнес-логики в одном типе. Наибольший вклад вносят ListSanatoriums (CC = 10) и группа методов создания, обновления и получения деталей санатория (CC = 6 каждый).

Распределение весов по методам показывает, что суммарная сложность типа складывается из нескольких самостоятельных сценариев каталога. Это упрощает локализацию изменений и сопровождения справочника санаториев.

4.4.6 Автоматизированный подсчет

Метрики цикломатической сложности верифицированы с помощью инструмента gocyclo. Результат отображен на рисунке 29.

```
PS C:\Users\super\source\repos\SoftwareArchitectureCourseWorkProject> gocyclo -over 0 -avg src/go-microservices/deal-service/internal/service/booking.go
10 service (*BookingService).ListSanatoriums src/go-microservices/deal-service/internal/service/booking.go:98:1
6 service (*BookingService).UpdateBooking src/go-microservices/deal-service/internal/service/booking.go:198:1
6 service (*BookingService).CreateBooking src/go-microservices/deal-service/internal/service/booking.go:163:1
6 service (*BookingService).GetSanatoriumDetails src/go-microservices/deal-service/internal/service/booking.go:139:1
4 service normalizePagination src/go-microservices/deal-service/internal/service/booking.go:290:1
4 service (*BookingService).publishBookingEvent src/go-microservices/deal-service/internal/service/booking.go:276:1
4 service (*BookingService).CancelBooking src/go-microservices/deal-service/internal/service/booking.go:233:1
3 service (*BookingService).ListBookings src/go-microservices/deal-service/internal/service/booking.go:256:1
2 service DatesOverlap src/go-microservices/deal-service/internal/service/booking.go:312:1
303:1
1 service IsGuestsExceedCapacity src/go-microservices/deal-service/internal/service/booking.go:335:1
1 service IsSanatoriumNotAvailable src/go-microservices/deal-service/internal/service/booking.go:331:1
1 service IsSanatoriumNotFound src/go-microservices/deal-service/internal/service/booking.go:327:1
1 service IsBookingNotFound src/go-microservices/deal-service/internal/service/booking.go:323:1
1 service toDateUTC src/go-microservices/deal-service/internal/service/booking.go:318:1
1 service (*BookingService).GetBooking src/go-microservices/deal-service/internal/service/booking.go:252:1
1 service NewBookingService src/go-microservices/deal-service/internal/service/booking.go:89:1
Average: 3.18
PS C:\Users\super\source\repos\SoftwareArchitectureCourseWorkProject>
```

Рисунок 29 – Полученные результаты метрик при помощи gocyclo

Среднее значение цикломатической сложности по всем функциям файла: 3,18.

Полученные показатели метрик кода находятся в пределах, приемлемых для сопровождения и тестирования: умеренный объем по Холстеду, цикломатическая сложность центрального метода $CC = 6$, метрика Чепина $Q = 22$, суммарная $WMC = 40$. Результаты функционального и нагрузочного тестирования, а также статического анализа подтверждают соответствие реализованного поведения заявленным требованиям к сценарию бронирования.

Заключение

В рамках выпускной квалификационной работы решена задача разработки информационной системы персонализированного подбора и бронирования санаторно-курортных услуг для малого санаторно-гостиничного бизнеса. Целью работы являлось создание распределенного программного комплекса, обеспечивающего сквозной сценарий от регистрации клиента и подбора санатория по совокупности критериев до оформления бронирования и оплаты, с возможностью администрирования каталога и контроля бронирований со стороны служебного контура.

На этапе анализа предметной области и требований выделены ключевые бизнес-процессы клиентского и административного контуров, рассмотрены программные аналоги и обоснована целесообразность собственной разработки. Сформулированы функциональные требования к регистрации и аутентификации, персонализированному поиску санаториев, управлению бронированиями и платежным операциям. Определены нефункциональные требования к безопасности, масштабируемости, надежности, наблюдаемости, сопровождаемости, совместимости и удобству использования системы.

В разделе проектирования принята микросервисная архитектура на языке Go с выделением сервисов аутентификации, бронирований и каталога, а также оплаты. Спроектированы модели данных на концептуальном, логическом и физическом уровнях с размещением сущностей в схемах PostgreSQL auth, deal и payment. Описаны REST API микросервисов, правила синхронного и асинхронного межсервисного взаимодействия через брокер NATS, а также состав клиентского веб-приложения. Подготовлена спецификация OpenAPI для публичного и административного контуров deal-service.

На этапе реализации развернута база данных PostgreSQL с воспроизводимой инициализацией и миграционными SQL-скриптами. Серверная часть реализована в виде контейнеризованных микросервисов с единой многослойной структурой, JWT-аутентификацией, защитой внутренних

вызовов API-ключом, идемпотентной обработкой платежей и публикацией доменных событий. Клиентская часть выполнена как одностраничное приложение на React и TypeScript; реализованы каталог с панелью фильтров по медицинским профилям и параметрам размещения, оформление и сопровождение бронирований, а также административные разделы управления санаториями и реестром бронирований. Исходный код информационной системы (микросервисы, клиентское приложение, скрипты PostgreSQL, конфигурация Docker) размещен в репозитории на платформе GitHub [20].

Проведено тестирование разработанного программного обеспечения. Функциональная верификация критичного сценария создания бронирования выполнена методом черного ящика с применением функциональных диаграмм и таблиц решений. Нагрузочное тестирование с использованием k6 подтвердило работоспособность сквозного пользовательского потока при имитации параллельных обращений. Статический анализ метрик кода для центрального метода сервисного слоя показал приемлемый уровень сложности по показателям Холстеда, цикломатической сложности, метрике Чепина и объектной метрике WMC. Полученные результаты свидетельствуют о соответствии реализованного поведения заявленным требованиям.

Таким образом, поставленная цель выпускной квалификационной работы достигнута: разработан и реализован программный комплекс, ориентированный на сегмент малых санаторно-курортных предприятий и обеспечивающий персонализированный подбор размещения, сквозное бронирование и оплату при разграничении прав клиента и администратора.

Перспективы развития системы связаны с расширением интеграционного и модульного тестового покрытия, внедрением гарантированной доставки событий на основе NATS JetStream, подключением внешних платежных шлюзов и дальнейшим развитием аналитических возможностей для администратора предприятия.

Список использованных источников

1. Воронцова Г.Г., Воронцова А.В. Автоматизация санаторно-курортных предприятий // Проблемы современной экономики. 2020. С. 166 – 168. [Электронный ресурс]. URL: <https://www.m-economy.ru/art.php?nArtId=6838> (дата обращения: 06.06.2026).
2. Резникова О.С., Се Яцзин, Скляренко М.А. Цифровые технологии бронирования в индустрии гостеприимства // Экономика, предпринимательство и право. 2025. С. 2547 – 2558. [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/vliyanie-tsifrovyyh-tehnologiy-bronirovaniya-ispolzuemyh-v-sfere-servisa-industrii-gostepriimstva-na-effektivnost-truda> (дата обращения: 06.06.2026).
3. PostgreSQL Documentation // PostgreSQL [Электронный ресурс]. URL: <https://www.postgresql.org/docs/> (дата обращения: 06.06.2026).
4. NATS Documentation // NATS [Электронный ресурс]. URL: <https://docs.nats.io/> (дата обращения: 06.06.2026).
5. UML – диаграмма вариантов использования (use case diagram) // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/47940/> (дата обращения: 10.06.2026).
6. Создаем диаграмму Ганта // UNISENDER [Электронный ресурс] URL: <https://www.unisender.com/ru/blog/chto-takoe-diagramma-ganta-i-kak-ee-sdelat/> (Дата обращения: 06.06.2026)
7. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 06.06.2026)
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: 06.06.2026)
9. Верников Г. Основы методологии IDEF1X // Проект «Корпоративный менеджмент» [Электронный ресурс]. URL: <https://www.cfin.ru/vernikov/idef/idef1x.shtml/> (дата обращения 06.06.2026)

10. Моделирование данных: обзор // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/556790/> (дата обращения: 06.06.2026)
11. 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: 06.06.2026)
12. Микросервисы – что такое // <https://sky.pro> [Электронный ресурс]. URL: <https://skyeng.ru/magazine/wiki/it-industriya/chto-takoe-mikroservisy/> (дата обращения: 06.06.2026)
13. Клиент-серверная архитектура. SA для самых маленьких. // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/en/articles/836464/> (дата обращения: 06.06.2026)
14. С. Ньюмен. От монолита к микросервисам. Эволюционные шаблоны для трансформации монолитной системы. // «БХВ-Петербург» 2022. С. 21. (дата обращения: 06.06.2026)
15. Д. Д. Чикилев. Виды архитектуры приложений. // Журнал «Научный лидер». 2024. С. 172. [Электронный ресурс]. URL: <https://scilead.ru/article/6673-vidi-arkhitekturi-prilozhenij> (дата обращения: 06.06.2026)
16. Фундаментальная теория тестирования // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/549054/> (дата обращения: 06.06.2026)
17. Юнит-тестирование для чайников // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/169381/> (дата обращения: 06.06.2026)
18. Интеграционное тестирование: виды, примеры и инструменты // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/articles/843534/> (дата обращения: 06.06.2026)
19. Программный код и его метрики // <https://habr.com> [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 06.06.2026)

20. GitHub. Исходный код информационной системы персонализированного подбора и бронирования санаторно-курортных услуг [Электронный ресурс]. URL: <https://github.com/Ypsidshi/Sanatorium-Diploma-project>- (дата обращения: 10.06.2026).

Приложение А
(справочное)
Глоссарий

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

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

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

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

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

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

Слой представления данных – это первый и самый верхний уровень, который присутствует в приложении. Данный уровень представляет собой пользовательский интерфейс (UI), то есть представление содержимого конечному пользователю через интерфейс. В разрабатываемой системе клиентское SPA относится к этому уровню; на стороне микросервисов прием и маршрутизация HTTP-запросов выделены в отдельный слой (приложение Д).

Сервис – это слой бизнес-логики приложения.

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

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

Доменный объект – это структура данных, описывающая сущность предметной области (бронирование, санаторий, пользователь, счет, платеж) без привязки к SQL и без сетевых деталей; в коде размещается в пакете `internal/domain` (приложение В).

Микросервис – это автономно развертываемый компонент распределенной системы, инкапсулирующий отдельный фрагмент функциональности и взаимодействующий с другими компонентами по сети (в работе: `auth-service`, `deal-service`, `payment-service`).

JWT (JSON Web Token) – токен доступа, выдаваемый микросервисом аутентификации после успешного входа; используется клиентским приложением и проверяется микросервисами бронирований и оплаты.

Брокер сообщений NATS – программная шина для асинхронной публикации и доставки доменных событий (например, подтверждение бронирования, завершение платежа) между микросервисами.

СУБД PostgreSQL – реляционная система управления базами данных, в которой данные логически разделены по схемам `auth`, `deal` и `payment` в соответствии с границами микросервисов.

Приложение Б
(рекомендуемое)
Спецификация атрибутов таблиц

Таблица Б.1 – Спецификация атрибутов таблицы авторизация.пользователи

№ п/ п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор пользователя	UUID	16	PK	Да	Уникальность	gen_random_uuid()
2	email	Адрес электронной почты (логин)	TEXT	-	-	Да	UNIQUE, не пустой	-
3	full_name	ФИО или отображаемое имя	TEXT	-	-	Да	Не пустой	-
4	role	Роль в системе	TEXT	-	-	Да	admin, client	-
5	password_hash	Хэш пароля	TEXT	-	-	Да	Не пустой	-
6	is_active	Признак активной учетной записи	BOOLEAN	1	-	Да	-	TRUE
7	created_at	Дата и время создания записи	TIMESTAMP PTZ	8	-	Да	-	NOW()
8	updated_at	Дата и время последнего изменения	TIMESTAMP PTZ	8	-	Да	-	NOW()

Таблица Б.2 – Спецификация атрибутов таблицы бронь.санатории

№ п/ п	Наименование	Описание	Тип данных	Разме р	Рол ь	Обязательн ый	Ограничени я	Значение по умолчанию
1	id	Идентификат ор санатория	UUID	16	PK	Да	Уникальнос ть	gen_random_uui d()
2	name	Наименовани е санатория	TEXT	-	-	Да	Уникальнос ть в паре с city	-
3	description	Описание объекта	TEXT	-	-	Да	-	пустая строка
4	city	Город размещения	TEXT	-	-	Да	-	-
5	address	Адрес	TEXT	-	-	Да	-	-
6	distance_to_sea_ km	Расстояние до моря, км	NUMERI C	8,2	-	Да	≥ 0	0
7	amenities	Список удобств (JSON- массив)	JSONB	-	-	Да	-	[]
8	image_urls	URL изображений (JSON- массив)	JSONB	-	-	Да	-	[]
9	price_per_night	Цена за ночь	NUMERI C	12,2	-	Да	> 0	-
10	total_places	Вместимость (число мест)	INT	4	-	Да	> 0	1
11	latitude	Широта (для карты)	NUMERI C	9,6	-	Нет	-	NULL

Продолжение таблицы Б.2

№ п/ п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
12	longitude	Долгота (для карты)	NUMERIC	9,6	-	Нет	-	NULL
13	created_at	Дата создания	TIMESTAMP Z	8	-	Да	-	NOW()
14	updated_at	Дата изменения	TIMESTAMP Z	8	-	Да	-	NOW()

Таблица Б.3 – Спецификация атрибутов таблицы бронь.бронирования

№ п/ п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор бронирования	UUID	16	PK	Да	Уникальность	gen_random_uuid()
2	client_id	Идентификатор клиента	UUID	16	-	Да	Соответствует пользователю с ролью client	-
3	sanatorium_id	Ссылка на санаторий	UUID	16	FK	Да	deal.sanatoriums(id), RESTRICT	-
4	check_in	Дата заезда	DATE	4	-	Да	check_in < check_out	-

Продолжение таблицы Б.3

№ п/ п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
5	check_out	Дата выезда	DATE	4	-	Да	-	-
6	guests	Число гостей	INT	4	-	Да	> 0	-
7	status	Статус бронирования	TEXT	-	-	Да	created, confirmed, cancelled	confirmed
8	amount	Сумма к оплате	NUMERIC	12,2	-	Нет	-	NULL
9	currency	Валюта расчета	CHAR	3	-	Да	-	RUB
10	payment_statuses	Статус оплаты бронирования	TEXT	-	-	Да	unpaid, invoice_issued, invoice_failed, paid	unpaid
11	payment_error	Текст ошибки платежного контура	TEXT	-	-	Да	-	пустая строка
12	invoice_id	Ссылка на счет в payment	UUID	16	-	Нет	UNIQUE при не NULL	NULL
13	created_at	Дата создания	TIMESTAMP PTZ	8	-	Да	-	NOW()
14	updated_at	Дата изменения	TIMESTAMP PTZ	8	-	Да	-	NOW()

Таблица Б.4 – Спецификация атрибутов таблицы оплата.счета

№ п/ п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения	Значение по умолчанию
1	id	Идентификатор счета	UUID	16	PK	Да	Уникальность	gen_random_uuid()
2	contract_id	Ссылка на договор (служебный контур)	UUID	16	-	Нет*	UNIQUE; *ровно одно из contract_id, booking_id	NULL
3	booking_id	Ссылка на бронирование (клиентский контур)	UUID	16	-	Нет*	UNIQUE при не NULL; *см. ограничение строки 2	NULL
4	amount	Сумма счета	NUMERIC	12,2	-	Да	> 0	-
5	currency	Валюта	CHAR	3	-	Да	-	RUB
6	status	Статус счета	TEXT	-	-	Да	issued, paid, overdue, cancelled	issued
7	issued_at	Дата выставления	TIMESTAMP PTZ	8	-	Да	-	NOW()
8	updated_at	Дата изменения	TIMESTAMP PTZ	8	-	Да	-	NOW()

Таблица Б.5 – Спецификация атрибутов таблицы оплата.оплаты

№ п/ п	Наименовани е	Описание	Тип данных	Разме р	Рол ь	Обязательн ый	Ограничения	Значение по умолчани ю
1	id	Идентификато р платежа	UUID	16	PK	Да	-	-
2	invoice_id	Ссылка на счет	UUID	16	FK	Да	payment.invoices(id), CASCADE	-
3	amount	Сумма платежа	NUMERI С	12,2	-	Да	> 0	-
4	status	Статус платежа	TEXT	-	-	Да	completed, failed, refunded	-
5	idempotency_k ey	Ключ идемпотентнос ти	TEXT	-	-	Да	UNIQUE	-
6	external_ref	Внешний идентификато р провайдера	TEXT	-	-	Нет	-	NULL

Приложение В
(обязательное)
Доменные объекты

Листинг В.1 – Структура Booking (deal-service/internal/domain/booking.go)

```

type Booking struct {
    ID                string    `json:"id"`
    ClientID          string    `json:"client_id"`
    SanatoriumID      string    `json:"sanatorium_id"`
    CheckIn           time.Time `json:"check_in"`
    CheckOut          time.Time `json:"check_out"`
    Guests            int       `json:"guests"`
    Status            string    `json:"status"`
    Amount            *float64 `json:"amount,omitempty"`
    Currency          string    `json:"currency,omitempty"`
    PaymentStatus     string    `json:"payment_status,omitempty"`
    InvoiceID          *string   `json:"invoice_id,omitempty"`
    CreatedAt         time.Time `json:"created_at"`
    UpdatedAt         time.Time `json:"updated_at"`
    CancelledAt       *time.Time `json:"cancelled_at,omitempty"`
}

```

Листинг В.2 – Структура Sanatorium (deal-service/internal/domain/sanatorium.go)

```

type Sanatorium struct {
    ID                string    `json:"id"`
    Name              string    `json:"name"`
    City              string    `json:"city"`
    PricePerNight     float64   `json:"price_per_night"`
    TotalPlaces       int       `json:"total_places"`
    MedicalProfiles   []string  `json:"medical_profiles"`
    // ...
}

```

Листинг В.3 – Структуры Invoice и Payment (payment-service/internal/domain/payment.go)

```

type Invoice struct {
    ID                string    `json:"id"`
    ContractID       string    `json:"contract_id,omitempty"`
    BookingID        string    `json:"booking_id,omitempty"`
    Amount           float64   `json:"amount"`
    Currency         string    `json:"currency"`
    Status           string    `json:"status"`
}

type Payment struct {
    ID                string    `json:"id"`
    InvoiceID         string    `json:"invoice_id"`
    Amount           float64   `json:"amount"`
    Status           string    `json:"status"`
    IdempotencyKey   string    `json:"idempotency_key"`
    PaidAt           time.Time `json:"paid_at"`
}

```

Листинг В.4 – Структура User (auth-service/internal/domain/user.go)

```
type User struct {  
    ID            string    `json:"id"`  
    Email         string    `json:"email"`  
    FullName      string    `json:"full_name"`  
    Role          string    `json:"role"`  
    PasswordHash  string    `json:"-"`  
    CreatedAt     time.Time `json:"created_at"`  
}
```

Приложение Г
(обязательное)
SQL-скрипты

Листинг Г.1 – Создание схем и таблицы пользователей

(sql/postgres/00_init_schemas.sql)

```
CREATE SCHEMA IF NOT EXISTS auth;
CREATE SCHEMA IF NOT EXISTS deal;
CREATE SCHEMA IF NOT EXISTS payment;

CREATE TABLE IF NOT EXISTS auth.users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email TEXT NOT NULL UNIQUE,
    full_name TEXT NOT NULL,
    role TEXT NOT NULL CHECK (role IN ('admin', 'client')),
    password_hash TEXT NOT NULL,
    is_active BOOLEAN NOT NULL DEFAULT TRUE,
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);
```

Листинг Г.2 – Таблица бронирований (sql/postgres/01_booking_catalog.sql)

```
CREATE TABLE IF NOT EXISTS deal.bookings (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    client_id UUID NOT NULL,
    sanatorium_id UUID NOT NULL REFERENCES deal.sanatoriums(id) ON
DELETE RESTRICT,
    check_in DATE NOT NULL,
    check_out DATE NOT NULL,
    guests INT NOT NULL CHECK (guests > 0),
    status TEXT NOT NULL DEFAULT 'confirmed'
    CHECK (status IN ('created', 'confirmed', 'cancelled')),
    CONSTRAINT ck_booking_dates CHECK (check_in < check_out)
);
```

Листинг Г.3 – Связь счета с бронированием

(sql/postgres/04_booking_payments.sql)

```
ALTER TABLE payment.invoices ADD COLUMN IF NOT EXISTS booking_id
UUID NULL;
ALTER TABLE payment.invoices
    ADD CONSTRAINT invoices_reference_check
    CHECK (
        (contract_id IS NOT NULL AND booking_id IS NULL)
        OR (contract_id IS NULL AND booking_id IS NOT NULL)
    );
```

Приложение Д
(обязательное)
Структуры слоя представления данных

Листинг Д.1 – Маршруты auth-service (internal/transport/http/handlers.go)

```
func (h *Handler) Router(registry *prometheus.Registry)
http.Handler {
    r := chi.NewRouter()
    r.Get("/health", h.ready)
    r.Route("/api/v1", func(api chi.Router) {
        api.Post("/users/register", h.register)
        api.Post("/auth/login", h.login)
    })
    return r
}
```

Листинг Д.2 – Маршруты deal-service (internal/transport/http/handlers.go)

```
r.Route("/api", func(api chi.Router) {
    api.Post("/auth/login", h.loginViaAuthService)
    api.Post("/auth/register", h.registerViaAuthService)
    api.Get("/sanatoriums", h.listSanatoriums)
    api.Get("/sanatoriums/{id}", h.getSanatoriumByID)

    api.Group(func(authorized chi.Router) {
        authorized.Use(ClientAuthMiddleware(h.jwtSecret,
h.logger))
        authorized.Post("/bookings", h.createBooking)
        authorized.Get("/bookings", h.listBookings)
        authorized.Post("/bookings/{id}/checkout",
h.checkoutBooking)
        authorized.Post("/bookings/{id}/pay", h.payBooking)
    })

    api.Route("/admin", func(admin chi.Router) {
        admin.Use(AuthMiddleware(h.jwtSecret, h.logger, "admin"))
        admin.Get("/bookings", h.listBookingsAdmin)
        admin.Post("/bookings/{id}/checkout",
h.adminCheckoutBooking)
        admin.Post("/bookings/{id}/pay", h.adminPayBooking)
        admin.Get("/sanatoriums", h.listSanatoriumsAdmin)
        admin.Post("/sanatoriums", h.createSanatoriumAdmin)
    })
})
```

Листинг Д.3 – Маршруты payment-service (internal/transport/http/handlers.go)

```
func (h *Handler) Router(registry *prometheus.Registry)
http.Handler {
    r := chi.NewRouter()
    r.Post("/internal/invoices", h.createInvoice)
    r.Route("/api/v1", func(api chi.Router) {
        api.Post("/payments", h.processPayment)
        api.Get("/payments/{id}", h.getPayment)
    })
    return r
}
```

Приложение Е
(обязательное)
Слой бизнес-логики

Листинг E.1 – Метод BookingService.CreateBooking (deal-service/internal/service/booking.go)

```
func (s *BookingService) CreateBooking(ctx context.Context,
traceID string, in CreateBookingInput) (domain.Booking, error) {
    if strings.TrimSpace(in.ClientID) == "" ||
strings.TrimSpace(in.SanatoriumID) == "" {
        return domain.Booking{}, fmt.Errorf("client_id and
sanatorium_id are required")
    }
    if in.Guests <= 0 {
        return domain.Booking{}, ErrInvalidGuests
    }
    if err := ValidateBookingDateRange(in.CheckIn, in.CheckOut);
err != nil {
        return domain.Booking{}, err
    }

    booking, err := s.repo.CreateBooking(ctx,
repository.NewBooking{
        ClientID: in.ClientID, SanatoriumID: in.SanatoriumID,
        CheckIn: in.CheckIn, CheckOut: in.CheckOut, Guests:
in.Guests,
    })
    if err != nil {
        return domain.Booking{}, err
    }

    s.publishBookingEvent(ctx, traceID, "booking.confirmed",
map[string]any{
        "booking_id": booking.ID, "client_id": booking.ClientID,
    })
    return booking, nil
}
```

Листинг E.2 – Регистрация пользователя (auth-service/internal/service/auth.go)

```
func (s *AuthService) Register(ctx context.Context, email,
password, fullName, role string) (domain.User, error) {
    email = strings.ToLower(strings.TrimSpace(email))
    if !strings.Contains(email, "@") {
        return domain.User{}, ErrInvalidEmail
    }
    if len(password) < 8 {
        return domain.User{}, ErrWeakPassword
    }
    if role != "client" {
        return domain.User{}, ErrInvalidRole
    }
    hash, err := bcrypt.GenerateFromPassword([]byte(password),
bcrypt.DefaultCost)
    if err != nil {
        return domain.User{}, fmt.Errorf("hash password: %w", err)
    }
}
```

```

    }
    return s.repo.CreateUser(ctx, email, fullName, role,
string(hash))
}

```

Листинг Е.3 – Проведение платежа (payment-service/internal/service/payment.go)

```

func (s *PaymentService) ProcessPayment(ctx context.Context,
traceID string, in ProcessPaymentInput) (domain.Payment, bool,
error) {
    if in.InvoiceID == "" {
        return domain.Payment{}, false, fmt.Errorf("invoice_id is
required")
    }
    if strings.TrimSpace(in.IdempotencyKey) == "" {
        return domain.Payment{}, false, fmt.Errorf("idempotency
key is required")
    }
    payment, isDuplicate, err := s.repo.ProcessPayment(ctx,
repository.ProcessPaymentInput{
        InvoiceID: in.InvoiceID, Amount: in.Amount,
        IdempotencyKey: in.IdempotencyKey, ExternalRef:
in.ExternalRef,
    })
    // публикация payment.completed в NATS при успехе
    return payment, isDuplicate, err
}

```

Приложение Ж
(обязательное)
Слой доступа к данным

Листинг Ж.1 – Создание бронирования в репозитории (deal-service/internal/repository/booking_catalog_postgres.go)

```
func (r *Repository) CreateBooking(ctx context.Context, in
NewBooking) (domain.Booking, error) {
    tx, err := r.db.BeginTx(ctx, &sql.TxOptions{Isolation:
sql.LevelSerializable})
    if err != nil {
        return domain.Booking{}, err
    }
    defer tx.Rollback()

    if in.Guests > totalPlaces {
        return domain.Booking{}, ErrGuestsExceedCapacity
    }
    available, err := isSanatoriumAvailable(ctx, tx,
in.SanatoriumID, in.CheckIn, in.CheckOut, in.Guests, nil)
    if !available {
        return domain.Booking{}, ErrSanatoriumNotAvailable
    }

    const insertQuery = `
INSERT INTO deal.bookings (id, client_id, sanatorium_id, check_in,
check_out, guests, status)
VALUES ($1, $2, $3, $4, $5, $6, 'confirmed')
RETURNING ` + bookingRowColumns
    // ...
    return booking, tx.Commit()
}
```

Листинг Ж.2 – Создание пользователя (auth-service/internal/repository/postgres.go)

```
func (r *UserRepository) CreateUser(ctx context.Context, email,
fullName, role, passwordHash string) (domain.User, error) {
    const q = `
INSERT INTO auth.users (email, full_name, role, password_hash)
VALUES ($1, $2, $3, $4)
RETURNING id, email, full_name, role, password_hash, created_at`
    var user domain.User
    err := r.db.QueryRowContext(ctx, q, email, fullName, role,
passwordHash).Scan(
        &user.ID, &user.Email, &user.FullName, &user.Role,
&user.PasswordHash, &user.CreatedAt,
    )
    return user, err
}
```

Приложение И
(обязательное)
Модульные тесты

Листинг И.1 – Тесты валидации дат (deal-service/internal/service/booking_test.go)

```

func TestValidateBookingDateRange(t *testing.T) {
    checkIn := time.Date(2026, 6, 10, 15, 0, 0, 0, time.UTC)
    checkOut := time.Date(2026, 6, 15, 11, 0, 0, 0, time.UTC)

    if err := ValidateBookingDateRange(checkIn, checkOut); err !=
nil {
        t.Fatalf("expected valid range, got error: %v", err)
    }
    if err := ValidateBookingDateRange(checkIn, checkIn); err ==
nil {
        t.Fatalf("expected error for same start/end date")
    }
}

```

Листинг И.2 – Тесты регистрации (auth-service/internal/service/auth_test.go)

```

func TestRegisterAllowsClientOnly(t *testing.T) {
    svc := NewAuthService(stubUserRepo{}, "secret", time.Hour)

    if _, err := svc.Register(context.Background(),
"client@example.com", "Pass1234", "Client User", "client"); err !=
nil {
        t.Fatalf("expected client registration to succeed, got
%v", err)
    }
    if _, err := svc.Register(context.Background(),
"admin@example.com", "Pass1234", "Admin User", "admin"); err !=
ErrInvalidRole {
        t.Fatalf("expected ErrInvalidRole for admin registration,
got %v", err)
    }
}

```

Приложение К
(рекомендуемое)
Иллюстрационные материалы к тестированию API

Проверка выполнялась через Swagger UI (/swagger на deal-service), Postman или аналог при работающем стенде Docker Compose. Ниже – перечень ключевых эндпоинтов и места для скриншотов ответов.

Таблица К.1 – Ключевые эндпоинты REST API

Сервис	Метод	Путь	Назначение	Рисунок
auth-service	POST	/api/v1/users/register	Регистрация клиента	К.1
auth-service	POST	/api/v1/auth/login	Выдача JWT	К.2
deal-service	POST	/api/auth/login	Вход через BFF deal-service	К.2
deal-service	GET	/api/sanatoriums	Каталог санаториев	К.3
deal-service	GET	/api/sanatoriums/{id}	Детали и доступность	К.4
deal-service	POST	/api/bookings	Создание бронирования (JWT client)	К.5
deal-service	POST	/api/bookings/{id}/checkout	Выставление счета	К.6
deal-service	POST	/api/bookings/{id}/pay	Оплата бронирования	К.7
deal-service	GET	/api/admin/bookings	Реестр бронирований (JWT admin)	К.8
deal-service	POST	/api/auth/login	Ошибка 401 при неверном пароле	К.9

Code

Details

201

Response body

```
{
  "id": "2f42d5d7-9e35-4498-be17-177221d26406",
  "email": "test20260604@example.com",
  "full_name": "Тестов Тест Тестович",
  "role": "client",
  "created_at": "2026-06-09T14:06:28.193712Z"
}
```

 Download

Response headers

```
content-length: 193
content-type: application/json
date: Tue, 09 Jun 2026 14:06:28 GMT
x-trace-id: 5c427d90-8ac6-46d6-8c25-5e32b9402753
```

Рисунок К.1 – Успешная регистрация клиента (POST /api/v1/users/register, код 201)

Responses

Response content type application/json ▼

```
Curl

curl -X "POST" \
'http://localhost:8082/api/auth/login' \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
  "email": "test20260604@example.com",
  "password": "Pass1234"
}'
```

Request URL

```
http://localhost:8082/api/auth/login
```

Server response

Code	Details
200	Response body <pre>{ "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFnbnCI6ImNlcnQyMDIzMjYvNEBleGFtcGlzLnVybSIwV4CIGMTk4NTAyMTk4NDcwZnVsbnF9YWV1IjoieklQLTdkGB8BVLTqtCyINCiOLXRgdGCINClOLXRgdcGBL7qstC4BYcltCnpYXQIOEIE3ODEwMTQIDQqIe3YveGUOI3JjbGllbmQiLCJzdWIiOiIyZjgyZDVkaHkxMzYTMjltQMOTgtYmduXkhlcztcyFAh3jYWMDFiFQ.6fYhyzK6dm8SfmILzael7kvlpMa3jovoF_RcIQKsk", "created_at": "2026-06-09T14:06:28.193712Z", "email": "test20260604@example.com", "full_name": "Tectoe Tect Tectosw", "id": "2142d5d7-9e35-4498-be17-177221d26486", "role": "client" }</pre> Download

Response headers

```
content-length: 547
content-type: application/json
date: Tue, 09 Jun 2026 14:16:24 GMT
x-trace-id: fc01f69f-a609-4249-b508-0d2f57ab4667
```

Code

Details

200

Response body

```
{
  "items": [
    {
      "id": "02a2a25a-3a14-4152-8cca-e01b7f2c3357",
      "name": "Санаторий Лазурный Берег",
      "description": "Современный санаторий на побережье для отдыха и оздоровления.",
      "city": "Геленджик",
      "address": "Набережная, 21",
      "distance_to_sea_km": 0.3,
      "amenities": [
        "spa",
        "pool",
        "wifi",
        "medical_center"
      ],
      "image_urls": [
        "/images/sanatorium-lazurny-1.jpg",
        "/images/sanatorium-lazurny-2.jpg",
        "/images/sanatorium-lazurny-3.jpg"
      ],
      "price_per_night": 7300,
      "total_places": 2,
      "medical_profiles": [
        "cardiology",
        "dermatology",
        "general_therapy"
      ],
      "status": "active"
    }
  ]
}
```

Download

Response headers

```
content-type: application/json
date: Tue, 09 Jun 2026 14:18:13 GMT
transfer-encoding: chunked
x-trace-id: 18bcl8e2-fa48-4bb2-afe5-f22e55a97240
```


Server response	
Code	Details
200	<div>Response body</div> <pre>{ "id": "56300058-e799-4a49-a53b-b0640dda94df", "client_id": "2f42d5d7-9a35-4498-be17-177221d26406", "sanatorium_id": "02a2a25a-3a14-4152-8cca-e01b7f2c3357", "check_in": "2026-08-01T00:00:00Z", "check_out": "2026-08-08T00:00:00Z", "guests": 2, "status": "confirmed", "amount": 51100, "currency": "RUB", "payment_status": "invoice_issued", "invoice_id": "a8bc0b58-c95f-4617-8dd2-5a8c36951b7f", "created_at": "2026-06-09T14:26:49.590906Z", "updated_at": "2026-06-09T14:28:21.090066Z" }</pre> <div>   </div> <div>Response headers</div> <pre>content-length: 457 content-type: application/json date: Tue, 09 Jun 2026 14:28:21 GMT x-trace-id: 733a428e-bc92-4748-b252-a5087a634bd1</pre>

Рисунок К.6 – Выставление счета (POST /api/bookings/{id}/checkout, код 200)

Server response	
Code	Details
200	<div>Response body</div> <pre>{ "booking": { "id": "56300058-e799-4a49-a53b-b0640dda94df", "client_id": "2f42d5d7-9a35-4498-be17-177221d26406", "sanatorium_id": "02a2a25a-3a14-4152-8cca-e01b7f2c3357", "check_in": "2026-08-01T00:00:00Z", "check_out": "2026-08-08T00:00:00Z", "guests": 2, "status": "confirmed", "amount": 51100, "currency": "RUB", "payment_status": "paid", "invoice_id": "a8bc0b58-c95f-4617-8dd2-5a8c36951b7f", "created_at": "2026-06-09T14:26:49.590906Z", "updated_at": "2026-06-09T14:29:55.066022Z" }, "duplicate": false }</pre> <div>   </div>

Рисунок К.7 – Подтверждение оплаты бронирования (POST /api/bookings/{id}/pay, код 200)

Server response	
Code	Details
200	<div>Response body</div> <pre>{ "items": [{ "id": "56300058-e799-4a49-a53b-b0640dda94df", "client_id": "2f42d5d7-9a35-4498-be17-177221d26406", "client_email": "test20260604@example.com", "sanatorium_id": "02a2a25a-3a14-4152-8cca-e01b7f2c3357", "sanatorium_name": "Санаторий Лазурный Берег", "check_in": "2026-08-01T00:00:00Z", "check_out": "2026-08-08T00:00:00Z", "guests": 2, "status": "confirmed", "amount": 51100, "currency": "RUB", "payment_status": "paid", "invoice_id": "a8bc0b58-c95f-4617-8dd2-5a8c36951b7f", "created_at": "2026-06-09T14:26:49.590906Z", "updated_at": "2026-06-09T14:29:55.077273Z" }, { "id": "7cd5ca45-fcc8-4a09-b0d7-a1479b7350a", "client_id": "b551ce3d-1cec-44f3-90e3-ed52e72fcd", "client_email": "super.a-piliugin@yandex.ru", "sanatorium_id": "447e33c8-6ae9-4303-9fae-5d8edd2932e5", "sanatorium_name": "Sea Breeze Health Resort", "check_in": "2026-06-05T00:00:00Z", "check_out": "2026-06-13T00:00:00Z", }] }</pre> <div>   </div>

Рисунок К.8 – Административный реестр бронирований (GET /api/admin/bookings, код 200)

Server response	
Code	Details
401	<div><div>Error: Unauthorized</div><div><div>Response body</div><div><pre>{ "error": "invalid credentials" }</pre></div><div> Download</div></div><div><div>Response headers</div><div><pre>content-length: 32 content-type: application/json date: Tue, 09 Jun 2026 14:32:49 GMT x-trace-id: b2079de8-2038-4ee1-9c05-6d9277505060</pre></div></div></div>

Рисунок К.9 – Ошибка аутентификации (POST /api/auth/login, код 401)