

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

Информационная система оказания
психологической помощи

Студент

(подпись, дата)

Ушаков Михаил Андреевич
(ФИО полностью)

Руководитель

(подпись, дата)

Козлов А.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 _____ Ушаков Михаил Андреевич _____ гр. _____ 22ВП1 _____ факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы _____ Информационная система оказания психологической помощи _____

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

3. Руководитель работы _____ д.т.н., профессор кафедры «МО и ПЭВМ» Козлов А.Ю. _____

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

1) Назначение: информационная система оказания психологической помощи в виде веб-сайта с доступом через браузер.

2) Среда выполнения: Windows 11.

Языковые средства разработки: JAVA, TYPESCRIPT, SQL, UML.

Инструментальные средства: Visual Studio Code, pgAdmin, Visual Paradigm.

Технологии разработки: микросервисная архитектура, ООП, принципы SOLID, шаблоны проектирования.

3) Основные функции приложения:

а) регистрация и авторизация;

б) создание парной комнаты психологической терапии;

в) создание одиночной комнаты психологической терапии;

г) просмотр ленты публикаций и подбор специалистов;

д) создание и написание публикаций;

е) поиск пользователей и добавление в друзья;

ж) администрирование сайта;

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи;

2) Проектирование;

3) Реализация;

4) Контроль качества программного обеспечения;

5) Оформление пояснительной записки.

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 _____ 26.12.2025
(подпись, дата)

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

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

Реферат

Пояснительная записка содержит 135 листов, 43 рисунка, 24 таблицы, 22 использованных источника и 6 приложений.

ИНФОРМАЦИОННАЯ СИСТЕМА, WEB-ПРИЛОЖЕНИЕ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, REST, UML, POSTGRESQL, TYPESCRIPT, БАЗА ДАННЫХ, REACTJS, DOCKER, GIT, WEBSOCKET.

Объект исследования – информационная система оказания психологической помощи.

Цель выпускной квалификационной работы – разработать информационную систему для оказания психологической помощи, позволяющую получать психологические консультации от специалистов и ИИ.

Технологии разработки – разработка клиентской части приложения проводилась на языке программирования TypeScript с использованием библиотеки ReactJS в среде разработки VS Code. Разработка серверной части приложения производилась на языке программирования Java в среде разработки VS Code. Разработка базы данных проводилась в СУБД PostgreSQL.

Среда исполнения – Windows 11, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы были созданы клиентская и серверная часть информационной системы для оказания психологической помощи. Все требования к разработке приложения соблюдены.

					ПГУ 09.03.04 – 08ВР221.22 ПЗ					
Изм.	Лист	№ докум.	Подпись	Дата	Информационная система оказания психологической помощи. Пояснительная записка			Лит.	Лист	Листов
Разраб.		Ушаков М.А.								
Пров.		Козлов А.Ю.							4	135
Реценз.								Группа 22ВП1		
Н. Контр.		Такташкин Д.В.								
Утв.		Козлов А.Ю.								

Содержание

Введение.....	7
1 Анализ требований и постановка задачи.....	9
1.1 Анализ предметной области	9
1.2 Анализ программных аналогов.....	10
1.2.1 Анализ аналога «Б17».....	10
1.2.2 Анализ аналога «ПроДокторов».....	12
1.2.3 Анализ аналога «Ясно».....	13
1.2.4 Результаты анализа	14
1.3 Постановка задачи на разработку.....	15
1.4 Анализ требований.....	18
1.4.1 Анализ функциональных требований.....	18
1.4.2 Анализ нефункциональных требований	21
1.5 Выбор инструментальных средств.....	23
1.6 Планирование проекта и оценка бюджета	25
2 Проектирование информационной системы оказания психологической помощи	28
2.1 Проектирование архитектуры информационной системы	28
2.2 Проектирование клиентской части	31
2.3 Проектирование серверной части.....	47
2.3.1 Проектирование сервиса шлюза	47
2.3.2 Проектирование сервиса пользователей.....	50
2.3.3 Проектирование сервиса контента	53
2.3.4 Проектирование сервиса чата	56
2.3.5 Проектирование сервиса биллинга	60
2.4 Проектирование базы данных.....	63
2.4.1 Разработка концептуальной модели базы данных.....	63
2.4.2 Разработка логической модели базы данных	67
2.4.3 Разработка физической модели базы данных	68
2.4.4 Развертывание базы данных в среде pgAdmin 4	71
2.5 Проектирование взаимодействия со средствами контейнеризации и обмена сообщениями	72
3 Реализация информационной системы оказания психологической помощи ..	75
3.1 Реализация взаимодействия с базой данных	75

3.1.1 Реализация миграций схем и генерируемых столбцов	75
3.1.2 Реализация серверных запросов к базе данных	77
3.1.3 Наполнение базы данных начальными данными	78
3.2 Реализация серверной части	80
3.2.1 Реализация аутентификации и авторизации.....	80
3.2.2 Реализация очередности ходов и взаимодействия с ИИ.....	82
3.2.3 Реализация гарантированной доставки событий	83
3.3 Реализация клиентской части	85
3.3.1 Реализация взаимодействия в реальном времени.....	85
3.3.2 Реализация управления серверным состоянием	88
3.3.3 Реализация разграничения доступа на стороне клиента.....	91
3.3.4 Реализация сборки компонентов	93
4 Тестирование информационной системы оказания психологической помощи	96
4.1 Функциональное тестирование.....	96
4.1.1 Классы эквивалентности	96
4.1.2 Функциональная диаграмма и таблица решений	97
4.1.3 Диаграмма состояний и переходов	99
4.2 Тестирование API средствами Postman	100
4.3 Модульное тестирование клиентской части	103
4.4 Оценка качества программного кода	105
4.4.1 Метрика Холстеда.....	105
4.4.2 Метрика Чепина	107
Заключение	109
Список использованных источников	111
Приложение А (справочное) Глоссарий	113
Приложение Б (обязательное) Листинг серверной части приложения	116
Приложение В (обязательное) Листинг клиентской части приложения.....	121
Приложение Г (рекомендуемое) Атрибуты сущностей базы данных	126
Приложение Д (рекомендуемое) Скрипт для создания объектов базы данных.....	130
Приложение Е (рекомендуемое) Листинг модульных тестов	133

Введение

Современные тенденции в области охраны психического здоровья все чаще смещаются в сторону цифровых решений, предоставляющих доступ к психологической поддержке из любой точки мира. Рост уровня тревожности, эмоционального выгорания и межличностных конфликтов сопровождается нехваткой квалифицированных специалистов и территориальной неравномерностью их распределения, а традиционные форматы помощи ограничены по охвату, времени работы и степени анонимности. Онлайн-платформы психологической помощи позволяют пользователям получать поддержку, изучать профессиональный контент и работать с инструментами саморефлексии без привязки к месту и времени. Это значительно снижает барьер обращения за помощью, повышает доступность поддержки и способствует формированию культуры заботы о ментальном здоровье.

Разрабатываемая информационная система представляет собой веб-приложение, объединяющее инструменты ИИ-ассистированной терапии, контентную среду профессиональных публикаций и социальные механики в единой безопасной экосистеме. Основными функциональными возможностями системы являются: регистрация и авторизация пользователей с разграничением ролей, ведение психологического профиля, просмотр контентной ленты с оценками и комментариями, формирование круга общения, оформление платной подписки, а также создание и проведение индивидуальных и парных ИИ-модерируемых терапевтических сессий. Система ориентирована как на пользователей, ищущих психологическую поддержку, так и на специалистов, публикующих профессиональные материалы.

Основной целью проекта является создание доступного, интуитивного и функционально насыщенного инструмента, который объединяет удобный интерфейс, надежную серверную архитектуру и безопасное хранение чувствительных персональных данных. При реализации системы используются современные технологии и принципы модульной архитектуры, обеспечивающие масштабируемость и отказоустойчивость, а также особое внимание уделяется

защите психологических профилей и содержимого терапевтических сессий пользователей.

Основные термины и сокращения, используемые в настоящей работе, приведены в приложении А.

1 Анализ требований и постановка задачи

1.1 Анализ предметной области

Информационная система оказания психологической помощи – это веб-платформа, объединяющая инструменты ИИ-ассистированной саморефлексии, контентную среду профессиональных публикаций и социальные механики взаимодействия пользователей в единую безопасную экосистему. Система предоставляет доступ к психологической поддержке только при помощи браузера, без привязки к месту, времени и без необходимости установки дополнительного программного обеспечения.

Актуальность разработки обусловлена устойчивым ростом потребности населения в доступных инструментах психологической поддержки [1]. Увеличение уровня тревожности, эмоционального выгорания и межличностных конфликтов сопровождается нехваткой квалифицированных специалистов и территориальной неравномерностью их распределения. Традиционные форматы помощи – очные консультации и телефоны доверия – ограничены по охвату, времени работы и степени анонимности, что создает барьер для обращения. Цифровые платформы способны дополнить работу специалистов, предоставляя пользователям пространство для саморефлексии и модулируемого диалога.

В предметной области выделяются три основные роли пользователей. Клиент – пользователь, который ищет психологическую поддержку: ведет ИИ-модерируемые дневники, участвует в парных сессиях, читает профессиональный контент и формирует круг общения. Психолог-автор – верифицированный специалист, публикующий профессиональные материалы и наращивающий аудиторию через контентную ленту. Администратор – пользователь с привилегированным доступом, выполняющий модерацию контента, управление аккаунтами и контроль финансовых операций через отдельную панель управления.

Для понимания предметной области необходимо определить ее основные понятия. ИИ-терапевтическая сессия – модулируемый искусственным

интеллектом диалог, в котором система анализирует сообщения участников и управляет очередностью высказываний. Контентная лента – раздел платформы, в котором отображаются профессиональные публикации авторов с поддержкой тегов, комментариев и оценок. Подписка – платный уровень доступа, открывающий пользователю расширенные возможности платформы, такие как создание парных терапевтических сессий.

Таким образом, разрабатываемая информационная система должна объединить терапевтические, контентные и социальные функции в единой безопасной экосистеме с разграничением ролей пользователей и встроенной модерацией.

1.2 Анализ программных аналогов

В рамках анализа предметной области необходимо разграничить классы систем, схожих по назначению с разрабатываемой информационной системой. Существующие сервисы онлайн-психологической помощи делятся на несколько групп в зависимости от решаемых задач: контентные психологические порталы, анкетные агрегаторы специалистов и сервисы онлайн-психотерапии с алгоритмическим подбором. Далее приведен обзор наиболее характерных представителей каждой из категорий с выявлением их сильных и слабых сторон.

1.2.1 Анализ аналога «Б17»

Б17 – один из крупнейших психологических порталов рунета, относящийся к категории контентных сообществ и работающий с 2008 года [2]. Платформа объединяет тысячи специалистов: помимо записи на консультацию, пользователи могут читать авторские статьи психологов, участвовать в форумных обсуждениях и задавать анонимные вопросы. Именно контентная составляющая отличает Б17 от большинства конкурентов: прежде чем обратиться к специалисту, клиент может изучить его публикации и понять профессиональную позицию автора, что повышает осознанность выбора. Главная страница портала «Б17» представлена на рисунке 1.

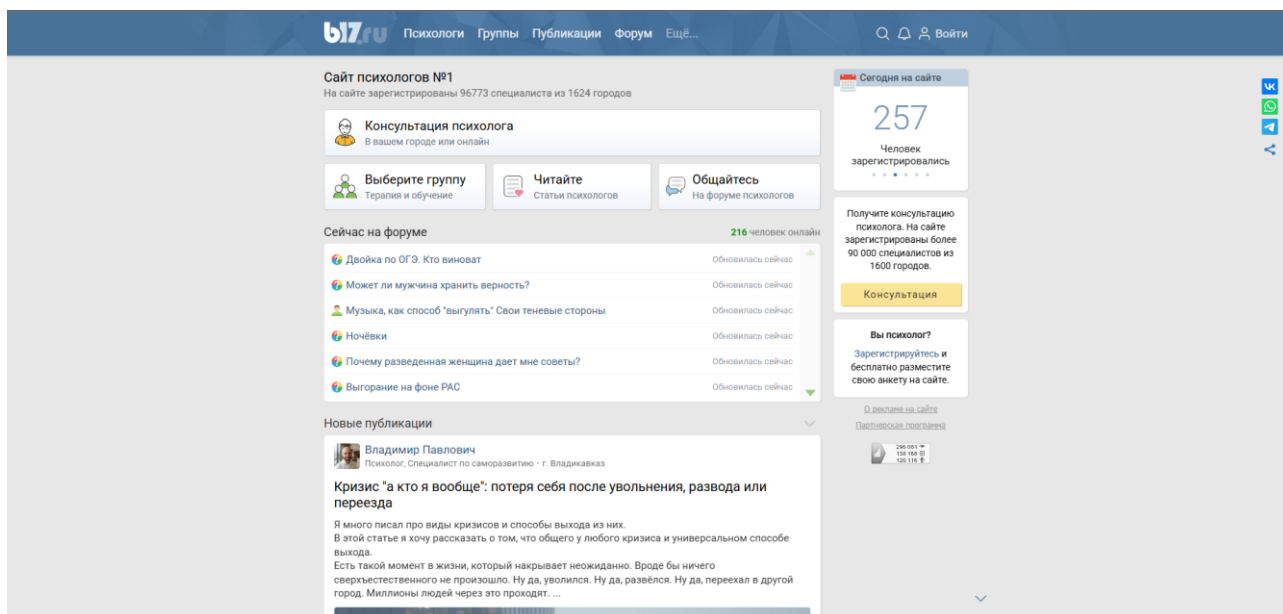


Рисунок 1 – Главная страница портала «Б17»

К положительным качествам данного сервиса можно отнести следующие возможности:

- глубина контентного наполнения и развитое сообщество с обратной связью;
- возможность познакомиться с подходом специалиста через его публикации;
- низкий порог входа – часть функций доступна без оплаты.

Несмотря на перечисленные достоинства, стоит отметить, что в сервисе присутствует ряд ограничений и недостатков:

- визуально устаревший и перегруженный интерфейс, затрудняющий навигацию и снижающий доверие новых пользователей;
- слабая поисковая оптимизация, из-за которой портал сложно обнаружить по типовым запросам;
- отсутствие инструментов саморефлексии и ИИ-ассистированной терапии.

Таким образом, портал Б17 обеспечивает глубокое контентное взаимодействие клиента со специалистом, однако устаревший интерфейс и отсутствие инструментов самопомощи ограничивают его применимость в качестве полноценной экосистемы психологической поддержки.

1.2.2 Анализ аналога «ПроДокторов»

ПроДокторов – универсальный агрегатор медицинских специалистов, включающий раздел психологов и относящийся к категории анкетных агрегаторов [3]. Сервис предоставляет каталог анкет с фильтрацией по стажу, образованию, стоимости и отзывам пациентов, а также возможность онлайн-записи на прием. Страница подбора психолога сервиса «ПроДокторов» представлена на рисунке 2.

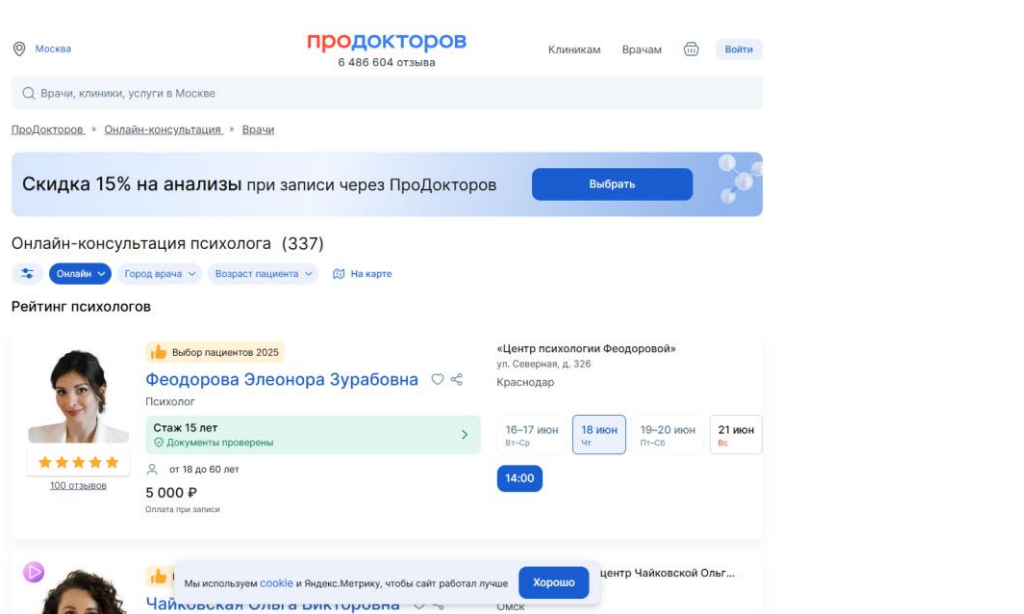


Рисунок 2 – Страница подбора психолога сервиса «ПроДокторов»

Среди ключевых преимуществ данного сервиса стоит выделить следующие пункты:

- широкая база специалистов, охватывающая не только психологов, но и врачей смежных направлений;
- развитая система верифицированных отзывов;
- привычный для пользователей интерфейс медицинского агрегатора.

Однако, несмотря на эти преимущества, сервис имеет существенные недостатки:

- анкетный принцип подбора: формальные критерии (стаж, образование, рейтинг) не отражают подход конкретного специалиста;

- отсутствие информации о методах работы и профессиональных взглядах психолога;
- отсутствие контентной и социальной составляющих.

Подводя промежуточный итог, можно отметить, что анкетные агрегаторы решают задачу формального поиска специалиста, однако вынуждают пользователя принимать решение исключительно на основе анкетных данных, без возможности заочного знакомства с подходом психолога.

1.2.3 Анализ аналога «Ясно»

Ясно – сервис онлайн-психотерапии, основанный в 2017 году и представляющий категорию платформ с алгоритмическим подбором специалиста [4]. После заполнения короткой анкеты алгоритм предлагает клиенту несколько подходящих психологов, из которых он выбирает самостоятельно. Консультации проходят в защищенном видеочате. Сервис проводит строгий отбор специалистов и регулярные супервизии, что повышает общий уровень качества услуг. Главная страница сервиса «Ясно» представлена на рисунке 3.

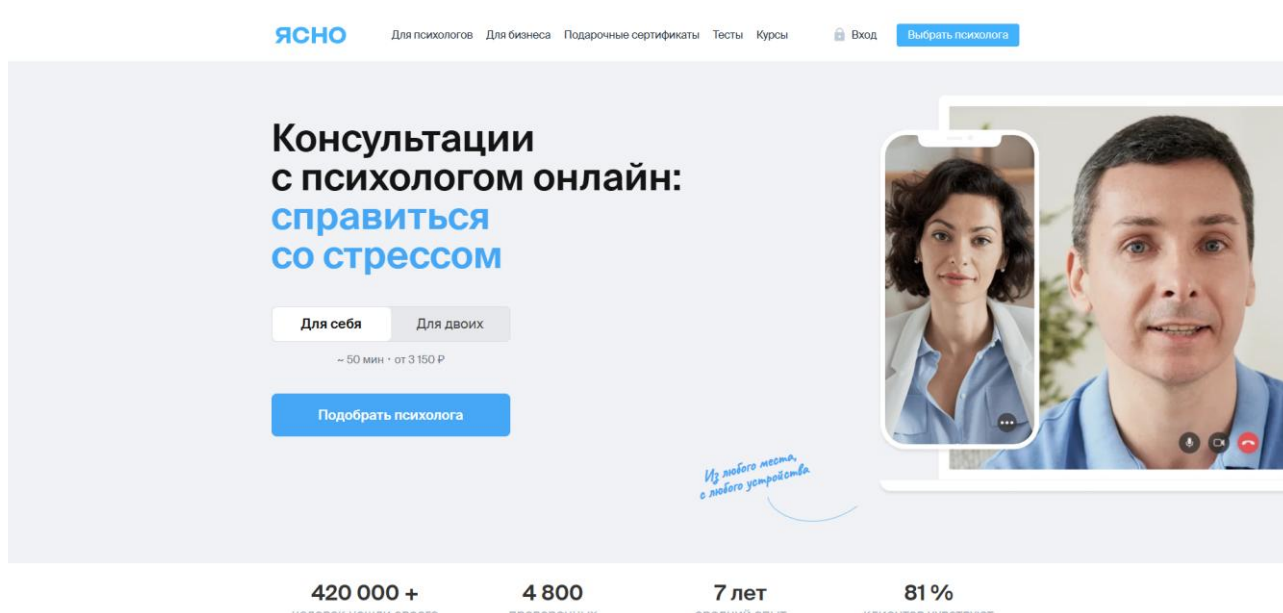


Рисунок 3 – Главная страница сервиса «Ясно»

Ключевые преимущества сервиса «Ясно»:

- современный и лаконичный интерфейс;

- строгий отбор специалистов и фиксированная стоимость сессии;
- наличие алгоритма подбора под конкретный запрос клиента.

Однако при всех преимуществах у платформы есть и недостатки:

- непрозрачность алгоритма подбора специалиста;
- итоговый выбор основан на анкетных данных – у пользователя нет возможности ознакомиться с профессиональным контентом специалиста до начала платной сессии;
- отсутствие бесплатных инструментов самопомощи и социальных механик.

Таким образом, сервисы онлайн-психотерапии обеспечивают качественный подбор и проведение консультаций, однако ограничивают пользователя рамками платного индивидуального формата и не формируют контентную среду вокруг специалистов.

1.2.4 Результаты анализа

Анализ существующих решений в области онлайн-психологической помощи позволяет выделить ряд ключевых проблем, характерных для различных типов платформ.

Проблемы анкетных агрегаторов:

- подбор специалиста ограничен формальными критериями (стаж, образование, рейтинг), которые не отражают профессиональную позицию и подход психолога;
- отсутствие контентной среды, позволяющей клиенту заочно познакомиться со специалистом до начала платного взаимодействия;
- отсутствие социальных механик, формирующих долгосрочную вовлеченность пользователя.

Проблемы контентных платформ:

- устаревший интерфейс, снижающий доверие и удобство использования;
- низкая поисковая видимость, затрудняющая привлечение новых пользователей;

– отсутствие алгоритмического подбора специалиста под конкретный запрос.

Общей проблемой всех рассмотренных решений является то, что ни одна из платформ не предоставляет пользователю инструментов самопомощи – ИИ-модерируемой саморефлексии, ведения терапевтического дневника или модерируемых парных сессий для разрешения конфликтов.

Обозначив перечисленные проблемы, становится очевидной потребность в создании интегрированной информационной системы, способной объединить в себе возможности контентного взаимодействия, социальных механик и ИИ-ассистированной терапии в едином веб-интерфейсе. Предлагаемое решение должно синтезировать сильные стороны рассмотренных аналогов: современность и удобство интерфейса сервиса «Ясно», контентную глубину и возможность знакомства с автором, характерные для портала «Б17», а также дополнить их принципиально новым функционалом – ИИ-модерируемыми терапевтическими сессиями для саморефлексии и разрешения межличностных конфликтов. В результате создается единая экосистема, охватывающая полный путь пользователя: от первого знакомства с темой ментального здоровья через контент – до выбора подходящего специалиста и самостоятельной работы с инструментами самопомощи.

1.3 Постановка задачи на разработку

Разрабатываемая информационная система оказания психологической помощи представляет собой веб-платформу, объединяющую инструменты ИИ-ассистированной терапии, контентную среду профессиональных публикаций и социальные механики в едином пользовательском интерфейсе. Основная цель системы – повысить доступность психологической поддержки: предоставить пользователям пространство для саморефлексии и модерируемого разрешения конфликтов, специалистам – среду публикации контента и привлечения аудитории, а администраторам – средства контроля и модерации.

Регистрация в системе является обязательным условием, так как для полноценной работы платформы требуется закрепление за пользователем психологического профиля, круга общения, подписок и истории терапевтических сессий. При этом чувствительные данные – психологическая анкета и содержимое сессий – скрыты от других пользователей и доступны только владельцу аккаунта.

Ключевой функциональной особенностью системы являются ИИ-модерируемые терапевтические сессии. В индивидуальном режиме («Соло-сессия») пользователь ведет диалог с искусственным интеллектом для саморефлексии и проработки проблем. В парном режиме («Парная комната») два участника при посредничестве ИИ разрешают межличностный конфликт: система анализирует каждое сообщение, управляет очередностью высказываний и обеспечивает экологичность диалога. Создание сессий доступно в рамках платной подписки, что формирует биллинговую составляющую системы.

Действующими лицами предметной области являются клиент, психолог-автор и администратор. Описание категорий пользователей и автоматизируемых бизнес-процессов представлено в таблице 1.

Таблица 1 – Категории пользователей информационной системы

№ к. п.	Категория пользователей	Наименование приложения	Автоматизируемые бизнес-процессы	Классы информационных объектов
1	Клиент	Информационная система оказания психологической помощи	Регистрация и авторизация, заполнение психологической анкеты, просмотр контентной ленты, поиск публикаций по тегам, оценка и комментирование публикаций, добавление друзей, подписка на авторов, оформление платной подписки, создание и проведение соло- и парных ИИ-терапевтических сессий, просмотр архива сессий, редактирование профиля	Пользователи, психологические анкеты, публикации, комментарии, оценки, теги, дружеские связи, подписки, терапевтические сессии, сообщения сессий

Продолжение таблицы 1

№ к. п.	Категория пользователей	Наименование приложения	Автоматизируемые бизнес-процессы	Классы информационных объектов
2	Психолог-автор	Информационная система оказания психологической помощи	Подача заявки на статус автора, создание и редактирование публикаций, присвоение тегов, публикация материалов, ответы на комментарии читателей, просмотр статистики аудитории	Публикации, черновики, теги, комментарии, подписчики
3	Администратор	Панель администрирования информационной системы	Модерация публикаций и комментариев, рассмотрение заявок на статус автора, управление статусами аккаунтов, блокировка пользователей, просмотр и контроль статусов финансовых транзакций	Пользователи, публикации, комментарии, заявки авторов, транзакции, подписки

Информационная система должна выполнять следующие информационные задачи:

- регистрация и авторизация пользователей с разграничением ролей;
- ведение профиля пользователя и психологической анкеты;
- формирование круга общения: добавление в друзья и подписка на авторов;
- создание, редактирование и публикация профессионального контента с поддержкой тегов;
- просмотр контентной ленты, оценка и комментирование публикаций в древовидных ветках;
- оформление и продление платной подписки;
- создание и проведение индивидуальных ИИ-терапевтических сессий;
- создание и проведение парных ИИ-модерируемых сессий с поочередной передачей хода;
- архивация сессий и просмотр истории диалогов;
- модерация контента, управление аккаунтами и контроль финансовых операций.

Сценарий выполнения ключевой информационной задачи – проведения парной ИИ-терапевтической сессии – представлен на рисунке 4 в виде диаграммы деятельности UML.

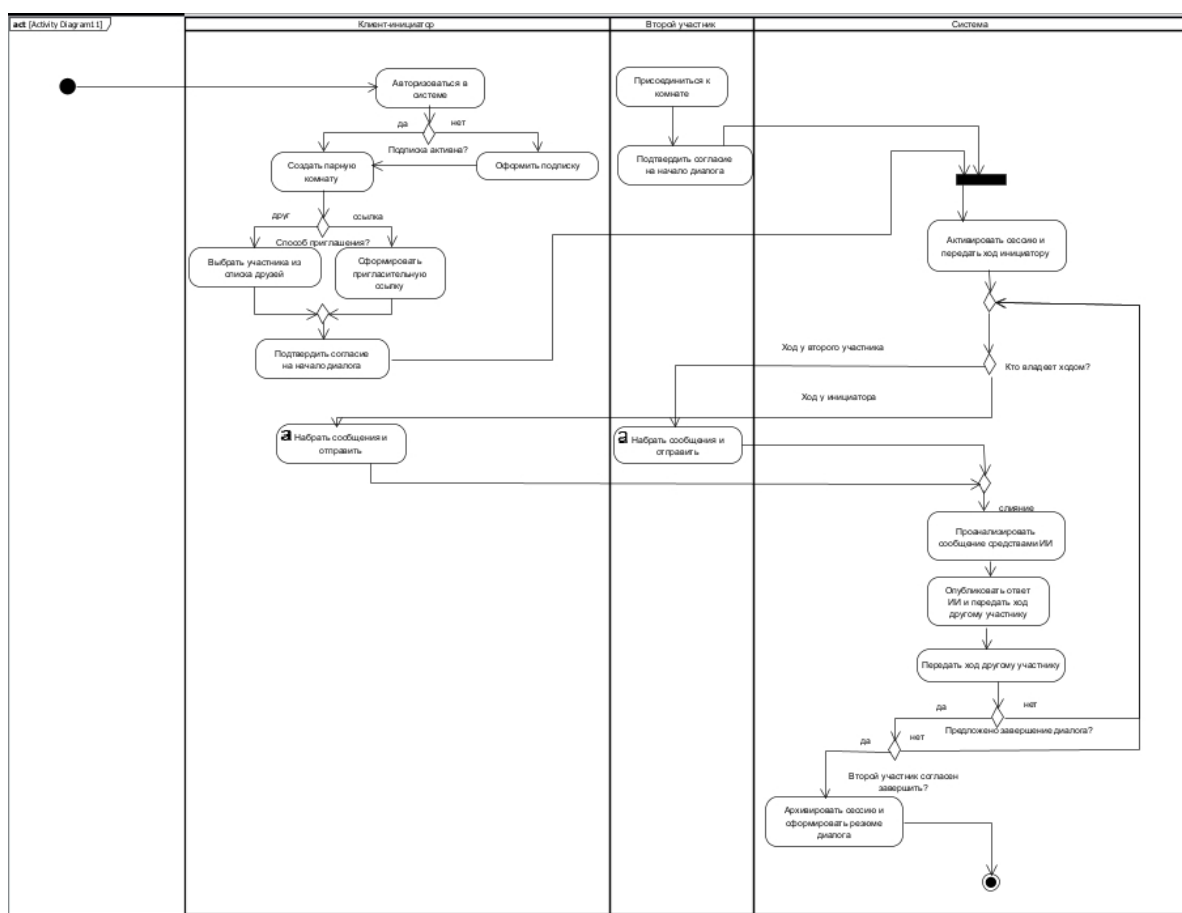


Рисунок 4 – Диаграмма деятельности проведения парной ИИ-терапевтической сессии

Полученные результаты являются исходными данными для анализа требований, проектирования и реализации информационной системы оказания психологической помощи.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

По результатам анализа предметной области, изучения программных аналогов и постановки задачи был сформирован перечень функциональных требований к разрабатываемой информационной системе.

Для визуализации полученных требований была разработана диаграмма вариантов использования в нотации UML [5]. Это визуальное представление взаимодействия пользователей с системой, описывающее функциональность с точки зрения конечного пользователя. Диаграмма определяет границы системы, внешних актеров и доступные им сценарии работы, что позволяет систематизировать требования и служит основой для последующего проектирования.

В разрабатываемой системе выделяются три основных актера: клиент, психолог-автор и администратор. Психолог-автор является специализацией клиента: ему доступны все клиентские функции, дополненные возможностями управления публикациями. Диаграмма вариантов использования представлена на рисунке 5.

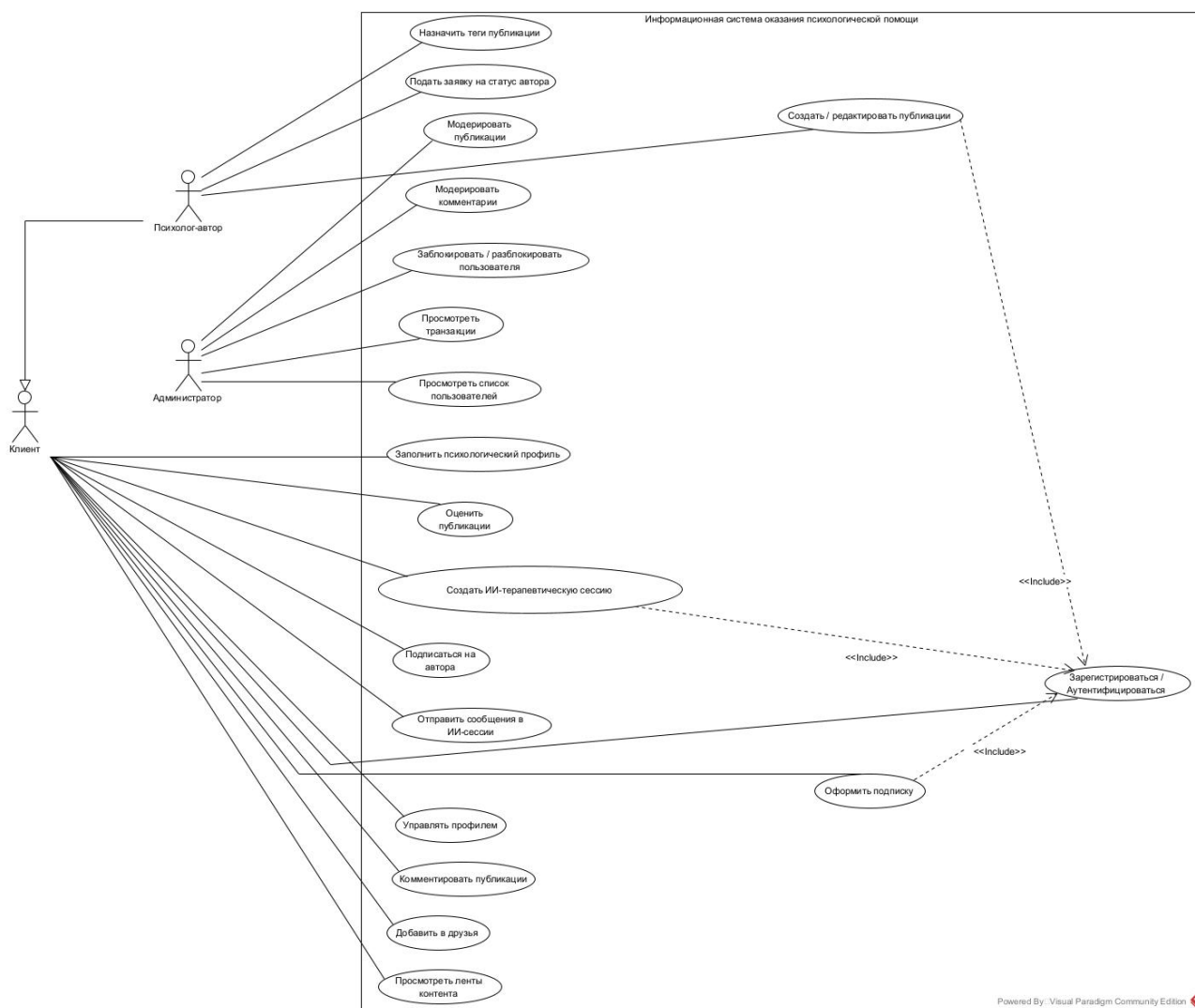


Рисунок 5 – Диаграмма вариантов использования информационной системы

Кроме того, были разработаны спецификации наиболее важных вариантов использования, которые приведены ниже.

Описание сценария варианта использования «Создание ИИ-терапевтической сессии» представлено в таблице 2.

Таблица 2 – Спецификация варианта использования «Создание ИИ-терапевтической сессии»

Название	Создание ИИ-терапевтической сессии
Актер	Клиент
Описание	Клиент создает индивидуальную или парную ИИ-модерируемую сессию для саморефлексии или разрешения конфликта
Предусловие	Клиент авторизован в системе; для парной сессии – оформлена активная подписка и выбранный участник добавлен в друзья
Основной поток	1. Клиент переходит в раздел «ИИ-Терапия». 2. Клиент выбирает тип сессии: «Соло-дневник» или «Парная комната». 3. При выборе парной комнаты клиент указывает участника из списка друзей. 4. Система создает комнату и назначает первый ход. 5. Отображается интерфейс сессии с полем ввода сообщения
Альтернативный поток	2а. Подписка не активна – система отображает предложение оформить подписку. 3а. Выбранный участник не найден в списке друзей – система предлагает сначала добавить пользователя в друзья
Постусловие	Создана новая ИИ-терапевтическая сессия; участники могут начать диалог

Описание сценария варианта использования «Публикация статьи» представлено в таблице 3.

Таблица 3 – Спецификация варианта использования «Публикация статьи»

Название	Публикация статьи
Актер	Психолог-автор
Описание	Автор создает и публикует профессиональную статью в контентной ленте платформы
Предусловие	Пользователь авторизован и имеет подтвержденную роль «Автор»
Основной поток	1. Автор переходит в раздел «Мои публикации». 2. Автор нажимает «Создать статью». 3. Автор заполняет заголовок и текст в блочном редакторе. 4. Автор добавляет обложку и назначает теги. 5. Автор нажимает «Опубликовать». 6. Система проверяет заполнение обязательных полей и публикует статью
Альтернативный поток	5а. Автор нажимает «Сохранить черновик» – статья сохраняется со статусом «Черновик» без публикации. 6а. Обязательные поля не заполнены – система выводит сообщение об ошибке
Постусловие	Статья опубликована и отображается в контентной ленте; подписчики автора видят новую публикацию

Описание сценария варианта использования «Блокировка пользователя» представлено в таблице 4.

Таблица 4 – Спецификация варианта использования «Блокировка пользователя»

Название	Блокировка пользователя
Актер	Администратор
Описание	Администратор блокирует аккаунт пользователя, нарушающего правила платформы
Предусловие	Администратор авторизован и имеет роль «Администратор»; целевой аккаунт существует и не заблокирован
Основной поток	1. Администратор переходит в раздел «Управление пользователями». 2. Администратор находит нужного пользователя. 3. Администратор нажимает «Заблокировать». 4. Система запрашивает подтверждение. 5. Администратор подтверждает действие. 6. Система меняет статус аккаунта на «Заблокирован» и скрывает контент пользователя из ленты
Альтернативный поток	5а. Администратор отменяет действие – статус аккаунта не изменяется
Постусловие	Аккаунт пользователя заблокирован; его публикации и комментарии скрыты из публичного доступа

По итогам анализа функциональных требований были сформулированы варианты использования информационной системы, с помощью которых у проекта появились четкие и однозначные рамки поведения, влияющие на дальнейший процесс проектирования и реализации. А для ряда ключевых вариантов использования были подготовлены их спецификации, позволяющие детально отобразить последовательность действий пользователя в системе.

1.4.2 Анализ нефункциональных требований

1.4.2.1 Анализ требований к серверной части

Информационная система оказания психологической помощи обрабатывает чувствительные персональные данные, в том числе психологические профили и содержимое терапевтических сессий, поэтому требования к безопасности и надежности серверной части являются приоритетными. При этом система должна сохранять доступность остальных функций даже при временной недоступности отдельных из них, а также рационально использовать вычислительные ресурсы и сохранять отзывчивость при возрастании нагрузки.

В ходе анализа были выделены следующие требования к серверной части:

- обеспечение доступа к защищенным разделам системы исключительно после прохождения процедур аутентификации и авторизации на основе механизма токенов доступа с ролевым разграничением прав на стороне сервера;
- хранение аутентификационных данных пользователей в криптографически защищенном виде с логической и физической изоляцией от публичных профилей;
- обеспечение строгой конфиденциальности психологических профилей пользователей и метаданных машинного анализа терапевтических сессий, исключающей несанкционированный доступ третьих лиц;
- реализация механизмов постраничной выдачи, денормализации данных и кэширования в оперативной памяти для оптимизации производительности высоконагруженных компонентов системы;
- обеспечение итоговой согласованности данных при выполнении распределенных транзакций с использованием асинхронного обмена сообщениями и механизмов компенсации;
- реализация отказоустойчивой обработки запросов с формированием информативных сообщений об ошибках и возможностью прозрачного для пользователя повторного выполнения операций при временной недоступности отдельных модулей;
- обеспечение возможности независимого развертывания и масштабирования модулей за счет использования изолированных хранилищ данных для каждого сервиса и применения глобально уникальных идентификаторов для кросс-сервисного взаимодействия.

Перечисленные требования определяют приоритеты последующего проектирования серверной части, прежде всего в части безопасности, надежности и производительности, и учитываются при выборе архитектурного подхода и инструментальных средств, рассматриваемых в подразделе 1.5 и во второй главе работы.

1.4.2.2 Анализ требований к интерфейсу

Интерфейс информационной системы должен обеспечивать пользователям удобную и комфортную работу. Учитывая специфику предметной области, интерфейс должен создавать ощущение безопасного и спокойного пространства: пользователь, обращающийся за психологической поддержкой, не должен сталкиваться с перегруженностью элементами и информационным шумом.

Для пользовательского интерфейса были выделены следующие требования:

- понятность пользователям без специальной технической подготовки;
- доступность основных разделов – контентной ленты, ИИ-терапии, личного кабинета – из единой навигации не более чем за два перехода;
- согласованность терминологии интерфейса с предметной областью психологической помощи, а не с техническими понятиями;
- отображение индикаторов загрузки на время выполнения запросов и блокировка повторного нажатия кнопок отправки;
- адаптивность к различным размерам экранов и кроссбраузерность.

Выполнение указанных требований позволит создать интерфейс, обеспечивающий высокую эффективность и удобство работы для всех категорий пользователей, а полученные в результате анализа данные потребуются при проектировании архитектуры и структур данных системы.

1.5 Выбор инструментальных средств

При создании информационной системы оказания психологической помощи должен задействоваться набор технологий, обеспечивающий кроссплатформенность, развитую экосистему библиотек и поддержку микросервисной архитектуры, выбранной по результатам анализа нефункциональных требований.

Серверная часть будет реализована на языке Java 21 с использованием фреймворка Spring Boot, который является промышленным стандартом

разработки корпоративных веб-приложений и предоставляет готовые решения для построения REST-сервисов, обеспечения безопасности и работы с данными [6]. Система разделяется на независимые микросервисы – сервис пользователей, сервис контента, сервис ИИ-сессий и сервис биллинга, – что соответствует принципам построения слабосвязанных распределенных систем [7]. Единой точкой входа выступает API-шлюз на основе Spring Cloud Gateway, выполняющий маршрутизацию запросов и проверку JWT-токенов.

Для управления данными используется система управления базами данных PostgreSQL – надежная объектно-реляционная СУБД с поддержкой транзакций и типа данных JSONB, что важно при хранении структурированного контента публикаций и метаданных ИИ-анализа [8]. Каждый микросервис работает с собственной схемой базы данных в соответствии с паттерном Schema-per-service. Для кэширования данных контентной ленты и хранения эфемерных черновиков сообщений терапевтических сессий применяется резидентное хранилище Redis. Асинхронный обмен событиями между сервисами обеспечивает брокер сообщений RabbitMQ, гарантирующий доставку событий при реализации распределенных транзакций по паттерну Saga.

Клиентская часть будет разрабатываться на библиотеке React 18 с использованием языка TypeScript, обеспечивающего статическую типизацию и снижающего количество ошибок на этапе компиляции [9]. Сборку проекта и мгновенную перезагрузку при изменениях обеспечивает инструмент Vite, что сокращает цикл внесения правок до нескольких секунд. Для управления серверным состоянием применяется библиотека TanStack Query, для клиентского состояния – Zustand.

Взаимодействие между клиентской и серверной частями осуществляется по протоколу HTTP при работе с запросами и по протоколу WebSocket (STOMP) во время работы с постоянным двусторонним потоком данных в ИИ-терапевтических сессиях, где требуется доставка сообщений и смена хода в реальном времени.

Развертывание системы выполняется средствами контейнеризации Docker: каждый микросервис, база данных, кэш и брокер сообщений упаковываются в отдельные контейнеры и оркестрируются с помощью Docker Compose, что обеспечивает воспроизводимость окружения и упрощает перенос системы между серверами.

Контроль версий кода реализован с использованием системы управления версиями Git. Хранилище кода синхронизировано с удаленным репозиторием GitHub, что позволяет организовать резервное хранение и отслеживание изменений.

Для построения диаграмм используется инструмент Visual Paradigm, поддерживающий стандарт UML 2.0 и позволяющий создавать диаграммы вариантов использования, деятельности, классов, последовательностей, а также архитектурные схемы развертывания.

Веб-приложение оптимизируется под современные браузеры: Google Chrome, Mozilla Firefox, Microsoft Edge, Safari, включая их последние версии, что обеспечивает корректное отображение интерфейса и стабильную работу на большинстве пользовательских устройств.

1.6 Планирование проекта и оценка бюджета

Управление сроками и затратами — неотъемлемая часть создания программного продукта. Для разрабатываемой информационной системы заранее нужно зафиксировать: какие работы выполняются, в какой последовательности, кто их делает и сколько времени на них уходит. Без такого плана сложно согласовать разработку и уложиться в установленный срок.

Для построения календарного графика и первичной оценки затрат был использован Microsoft Project [10]. Это программное обеспечение, предназначенное для планирования задач, распределения ресурсов, отслеживания прогресса выполнения работ, а также аналитики рабочих нагрузок, бюджетов и построения визуальных графиков.

Для выполнения данного проекта выделено три роли:

- аналитик (64 чел.ч);
- программист (528 чел.ч);
- тестировщик (32 чел.ч).

Суммарные трудозатраты по плану составляют 624 чел.ч. Для оценки бюджета проекта приняты ориентировочные почасовые ставки, представленные в таблице 5.

Таблица 5 – Почасовые ставки исполнителей

Исполнитель	Ставка, руб./час
Аналитик	300
Программист	380
Тестировщик	280

Расчет расходов на разработку приведен в таблице 6.

Таблица 6 – Расходы на разработку информационной системы

№	Этап	Аналитик, чел.ч	Программист, чел.ч	Тестировщик, чел.ч	Стоимость, руб.
1	Анализ предметной области	64	–	–	19200
2	Проектирование	–	144	–	54720
3	Реализация	–	264	–	100320
4	Тестирование	–	56	32	30420
5	Оформление документации	–	64	–	24320
	Итого	64	528	32	228800

Таким образом, планирование показало, что разработка информационной системы укладывается в два календарных месяца, а предварительная стоимость разработки составляет 228 800 руб. Полученные оценки носят плановый характер и могут уточняться при изменении состава команды или ставок, но служат основой для контроля хода выполнения работы. Результаты анализа требований и постановки задачи являются исходными данными для проектирования информационной системы оказания психологической помощи.

2 Проектирование информационной системы оказания психологической помощи

2.1 Проектирование архитектуры информационной системы

Для удовлетворения функциональных и нефункциональных требований, сформированных в первой главе, в качестве архитектурного шаблона серверной части информационной системы была выбрана микросервисная архитектура. Микросервисная архитектура – это архитектурный стиль, при котором приложение строится как совокупность независимо развертываемых сервисов, организованных вокруг бизнес-возможностей, обладающих собственным хранилищем данных и взаимодействующих между собой через легкие протоколы [7]. Такой подход обеспечивает гибкость в разработке, изоляцию сбоев, независимую масштабируемость отдельных функциональных модулей и возможность использования различных технологий в рамках одной системы.

Выбор микросервисной архитектуры обусловлен спецификой предметной области: модули ИИ-терапии, ленты контента, биллинга и администрирования имеют существенно различающиеся профили нагрузки, требования к согласованности данных и темпы развития, что делает нецелесообразным их объединение в монолитное приложение. Кроме того, разделение приватных психологических данных клиента и публичных данных профиля реализуется на уровне физического разделения схем баз данных, что соответствует требованиям безопасности, сформулированным в подразделе 1.4.2.

В качестве основного способа коммуникации между клиентским приложением и серверной частью используется REST API поверх HTTPS. Главная особенность REST – обмен сообщениями без сохранения состояния: каждое сообщение самодостаточно и содержит всю информацию, необходимую для его обработки, что делает сервисы слабо связанными и облегчает масштабирование системы. Для модуля ИИ-терапии, где требуется доставка сообщений и смена хода в реальном времени, дополнительно используется

протокол WebSocket – технология двунаправленной асинхронной связи между браузером и сервером [11].

На рисунке 7 приведена диаграмма компонентов, из которых состоит информационная система.

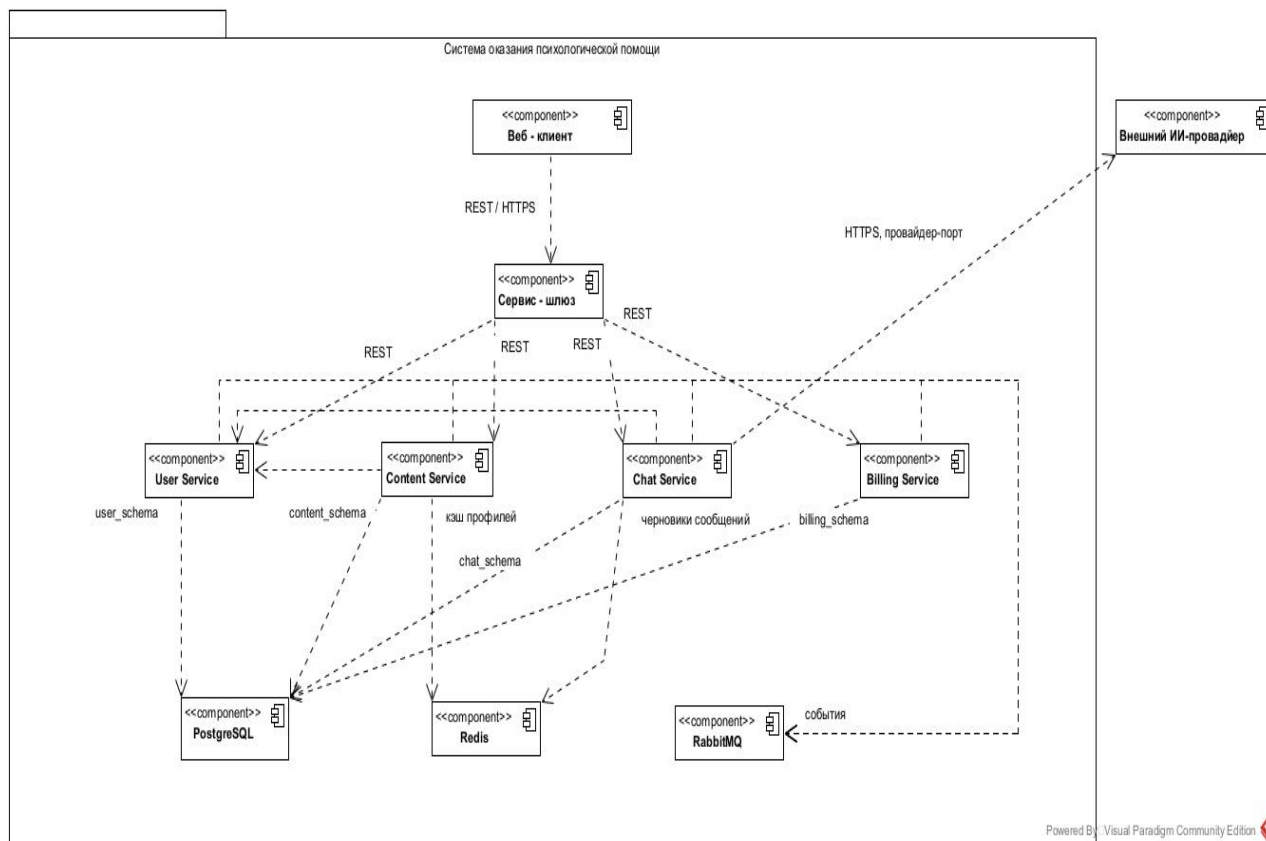


Рисунок 7 – Диаграмма компонентов информационной системы

Согласно представленной диаграмме, в системе выделяются следующие компоненты.

Клиентская часть представлена веб-клиентом (SPA), через который пользователь взаимодействует с контентной лентой, личным кабинетом и модулем ИИ-терапии, а пользователи с соответствующими правами – с административной панелью. Клиент общается с серверной частью посредством REST API, а в терапевтических сессиях дополнительно использует WebSocket-соединение.

Сервис-шлюз (API Gateway) является единой точкой входа для всего клиентского трафика. Шлюз выполняет маршрутизацию запросов по префиксам путей к соответствующим микросервисам, лимитирование частоты запросов и

сквозную передачу идентификатора трассировки. Шлюз не хранит состояние, не содержит бизнес-логики и не имеет собственной базы данных, что обеспечивает его горизонтальную масштабируемость.

Сервис пользователей (User Service) – единый источник истины по идентификации пользователей, их ролям и социальному графу. Сервис отвечает за регистрацию и аутентификацию с генерацией JWT-токенов, хранение и обновление профилей, реализацию подписок на авторов и дружеских связей, а также за выдачу пакетных ответов с публичными данными пользователей для обогащения ответов других сервисов.

Сервис контента (Content Service) управляет жизненным циклом публикаций, хранением древовидных комментариев, оценками постов и справочником тегов. Сервис ориентирован на высокую скорость чтения: счетчики оценок и комментариев денормализованы, а данные авторов запрашиваются у сервиса пользователей пакетными вызовами с кэшированием в Redis.

Сервис ИИ-сессий (Chat Service) реализует индивидуальные и парные ИИ-модерируемые терапевтические сессии: управляет жизненным циклом комнат, очередностью ходов участников, хранением сообщений и взаимодействием с внешним ИИ-провайдером. Доступ к внешней языковой модели абстрагирован интерфейсом провайдера, что позволяет заменять поставщика ИИ без изменения бизнес-логики сервиса.

Сервис биллинга (Billing Service) является финансовым ядром системы: владеет жизненным циклом подписок, журналом транзакций, промокодами и пробным периодом. Взаимодействие с внешним платежным провайдером также вынесено за интерфейс, а повышение роли пользователя выполняется только после подтверждения оплаты.

Реляционная СУБД PostgreSQL выступает основным персистентным хранилищем. Каждому микросервису соответствует собственная логическая схема (user_schema, content_schema, chat_schema, billing_schema), что

обеспечивает изоляцию данных: жесткие внешние ключи между схемами отсутствуют, межсхемные ссылки реализованы как мягкие ссылки по UUID.

Резидентное хранилище Redis используется как кэширующая прослойка для ускорения операций чтения, хранилище эфемерных черновиков сообщений терапевтических сессий и как хранилище состояния механизма лимитирования запросов на стороне шлюза.

Брокер сообщений RabbitMQ выступает в роли асинхронной шины событий между микросервисами. Гарантированная доставка обеспечивается реализацией паттерна Transactional Outbox: бизнес-событие сохраняется в таблицу исходящих событий в одной транзакции с основным бизнес-действием, после чего асинхронно публикуется в брокер фоновым обработчиком.

Взаимодействие между компонентами системы организовано по следующим правилам. Весь клиентский трафик проходит через API Gateway и маршрутизируется к соответствующему микросервису по префиксу пути. Межсервисное взаимодействие происходит во внутренней сети, минуя шлюз, и делится на два типа: синхронные REST-вызовы – исключительно для операций обогащения данных; асинхронный обмен событиями через RabbitMQ – для всех остальных случаев распространения изменений состояния (изменение роли пользователя, публикация поста, подтверждение оплаты, архивация сессии).

Выбранная архитектура удовлетворяет всем сформулированным ранее требованиям: микросервисный подход обеспечивает масштабируемость и независимую развертываемость компонентов; единая точка входа упрощает применение мер безопасности; разделение схем базы данных гарантирует требуемый уровень конфиденциальности чувствительных данных; шаблон Transactional Outbox в сочетании с RabbitMQ обеспечивает отказоустойчивость и гарантированную доставку событий между сервисами.

2.2 Проектирование клиентской части

Пользовательский интерфейс клиентского приложения должен обеспечивать привычное и удобное представление информации, простое и

эффективное выполнение основных функций пользователей, а с учетом специфики предметной области – создавать ощущение безопасного и спокойного пространства.

Клиентская часть информационной системы представляет собой одностраничное приложение с общим каркасом: верхней панелью навигации, обеспечивающей доступ ко всем разделам не более чем за два перехода, и основной рабочей областью, в которой отображается содержимое текущего раздела. Состав отображаемых пунктов навигации зависит от роли пользователя: раздел «Мои публикации» доступен только авторам, раздел администрирования – только администраторам.

Интерфейс приложения должен состоять из следующих компонентов:

- форма авторизации;
- форма регистрации;
- страница контентной ленты;
- страница поиска публикаций;
- страница просмотра публикации с комментариями;
- страница управления кругом общения «Друзья и подписки»;
- страница автора;
- личный кабинет пользователя с психологической анкетой;
- страница управления подпиской;
- студия автора с редактором публикаций;
- страница списка ИИ-терапевтических сессий;
- окно ИИ-терапевтической сессии;
- страницы административной панели: управление пользователями, тегами и биллингом.

Реализация перечисленных компонентов в едином визуальном стиле обеспечит смысловую целостность программного продукта и значительно снизит порог вхождения для новых пользователей системы.

На рисунке 8 представлен макет формы авторизации.

The diagram shows a login form titled "Вход" (Login). It contains the following elements:

- 1. A line pointing to the "Email" input field.
- 2. A line pointing to the "Пароль" (Password) input field.
- 3. A line pointing to the blue "Войти" (Login) button.
- 4. A line pointing to the link "Нет аккаунта? Зарегистрироваться" (No account? Register).

Рисунок 8 – Форма авторизации пользователя

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – поле для ввода почты пользователя;
- 2 – поле для ввода пароля;
- 3 – кнопка входа в систему;
- 4 – ссылка для перехода на форму регистрации.

На рисунке 9 представлен макет формы регистрации.

The diagram shows a registration form titled "Регистрация" (Registration). It contains the following elements:

- 1. A line pointing to the "Email" input field.
- 2. A line pointing to the "Имя пользователя (username)" input field.
- 3. A line pointing to the "Полное имя (Full Name)" input field.
- 4. A line pointing to the "Пароль" (Password) input field.
- 5. A line pointing to the blue "Зарегистрироваться" (Register) button.
- 6. A line pointing to the link "Уже есть аккаунт? Войти" (Already have an account? Login).

Рисунок 9 – Форма регистрации пользователя

На рисунке цифрами отмечены следующие компоненты формы:

- 1 – поле для ввода почты пользователя;
- 2 – поле для ввода имени пользователя (логина);
- 3 – поле для ввода полного имени;
- 4 – поле для ввода пароля;
- 5 – кнопка регистрации;
- 6 – ссылка для перехода на форму авторизации.

Данные формы позволяют пользователю быстро получить доступ к системе: регистрация требует минимального набора данных, а заполнение психологической анкеты вынесено в личный кабинет и выполняется по желанию пользователя.

На рисунке 10 представлен макет главного окна приложения со страницей контентной ленты.

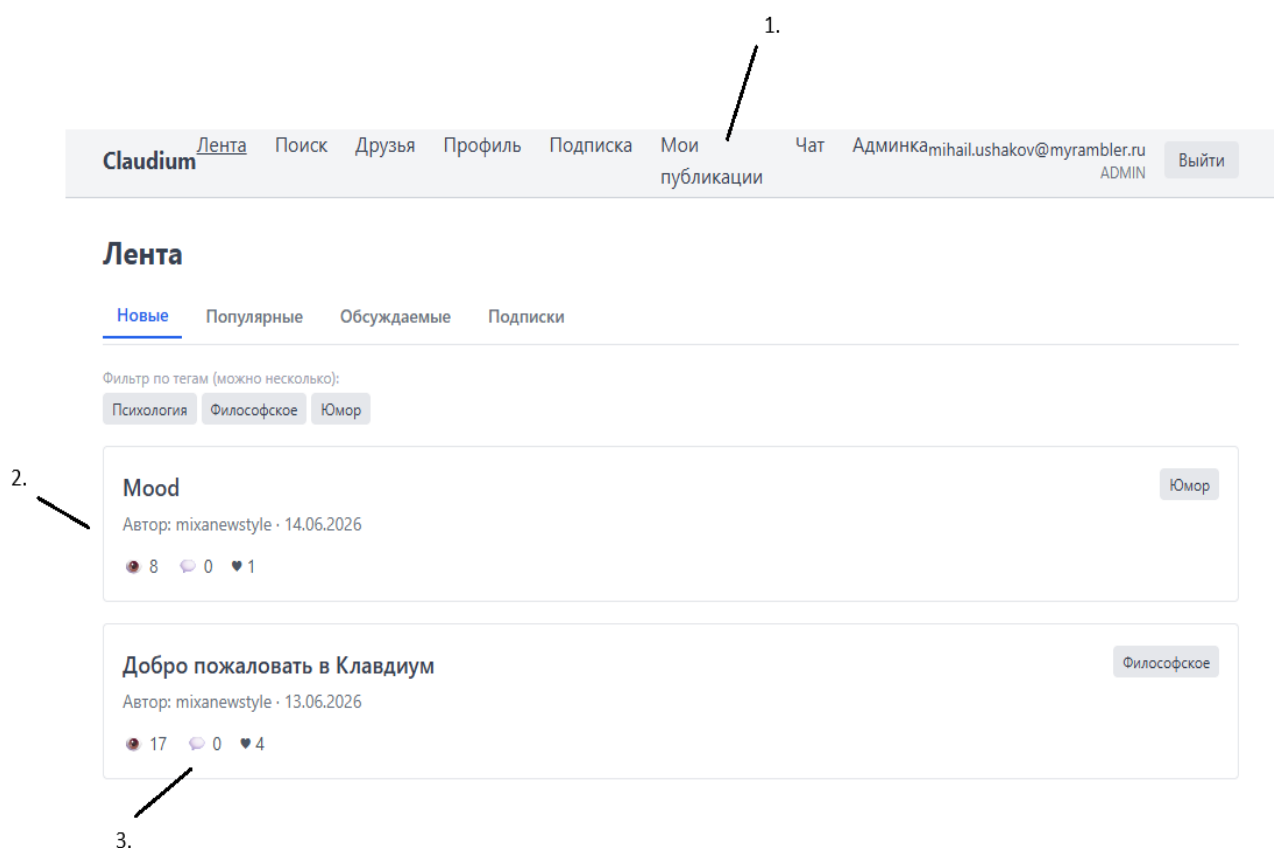


Рисунок 10 – Главное окно приложения со страницей контентной ленты

На рисунке цифрами отмечены следующие компоненты:

1 – боковая панель навигации с разделами «Лента», «Поиск», «Друзья», «Профиль», «Подписка», «Чат» и кнопкой выхода;

2 – список карточек публикаций с названием, заголовком и именем автора;

3 – счетчики оценок, просмотров и комментариев публикации;

Контентная лента является стартовой страницей приложения: после авторизации пользователь сразу попадает в пространство профессиональных публикаций, что соответствует пользовательскому пути знакомства с платформой.

На рисунке 11 представлен макет страницы поиска публикаций.

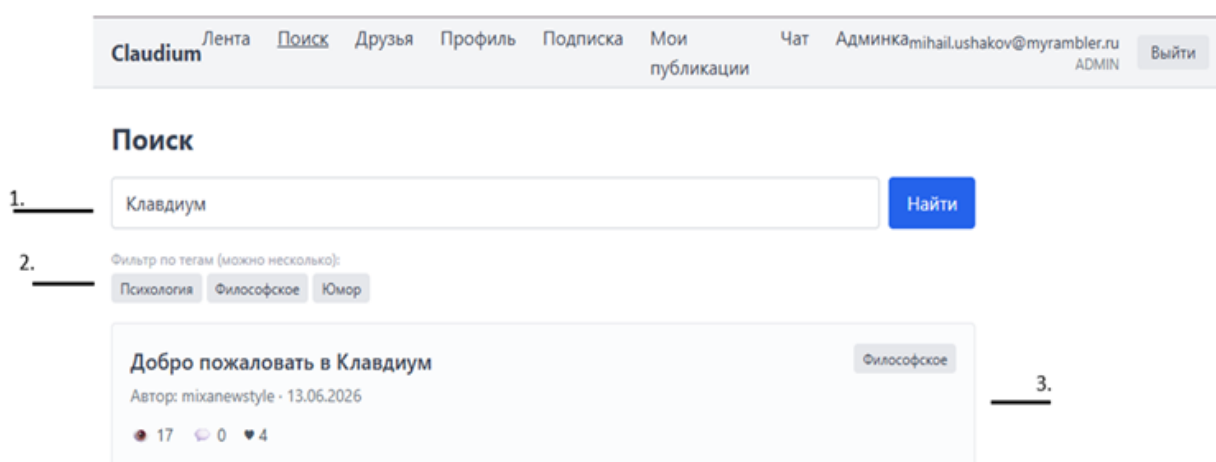


Рисунок 11 – Страница поиска публикаций

На рисунке цифрами отмечены следующие компоненты:

1 – поле ввода поискового запроса;

2 – панель выбора тегов для фильтрации;

3 – список найденных публикаций;

На рисунке 12 представлен макет страницы просмотра публикации.

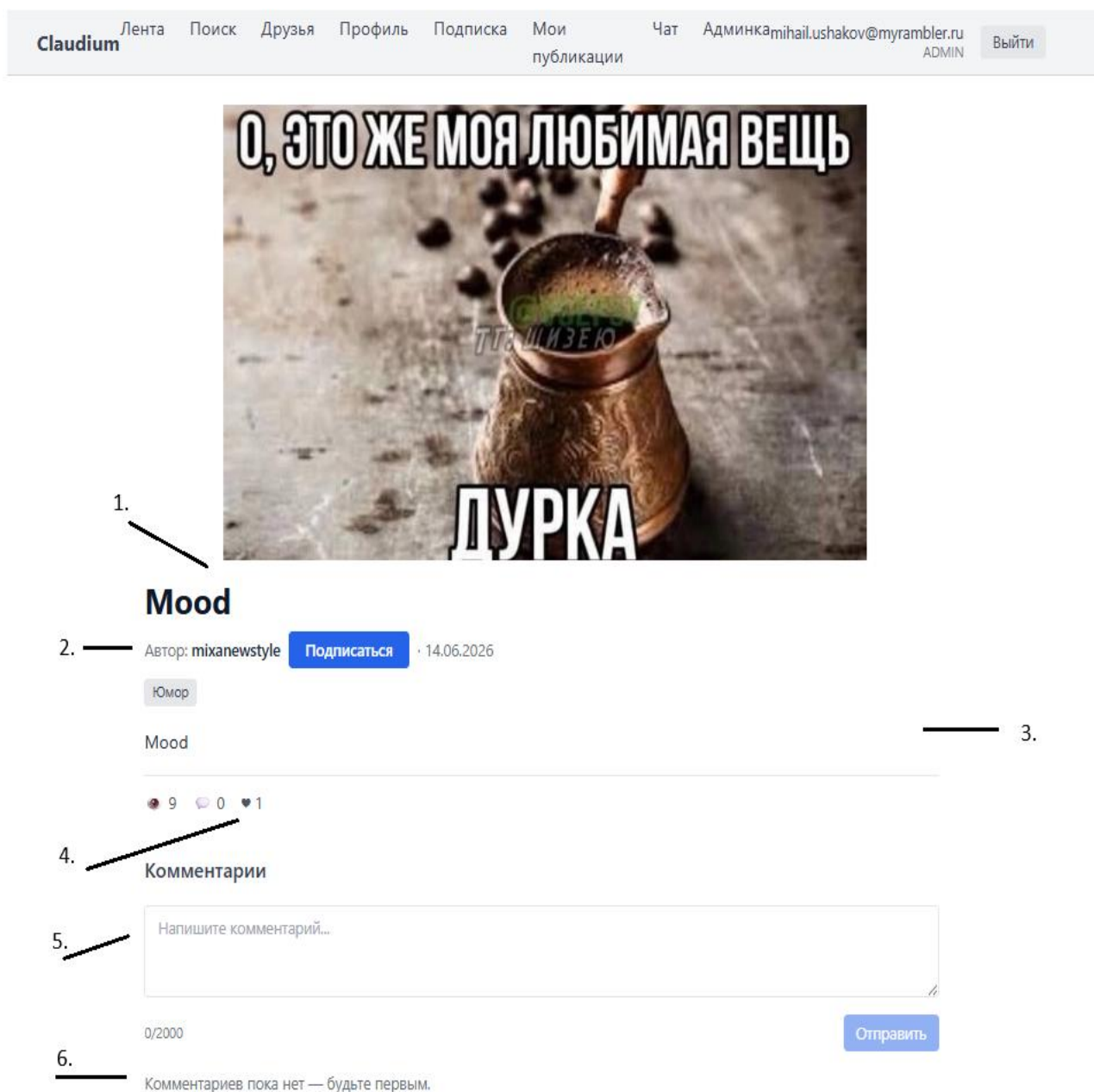


Рисунок 12 – Страница просмотра публикации

На рисунке цифрами отмечены следующие компоненты:

- 1 – заголовок и обложка публикации;
- 2 – блок информации об авторе с переходом на его страницу;
- 3 – содержимое публикации;
- 4 – кнопка оценки публикации;
- 5 – поле для комментария и его отправки;
- 6 – блок существующих комментариев.

Страница публикации объединяет чтение материала и социальное взаимодействие: пользователь может оценить статью, вступить в обсуждение в многоуровневых ветках комментариев и перейти к профилю автора для знакомства с другими его материалами.

На рисунке 13 представлен макет страницы «Друзья и подписки».

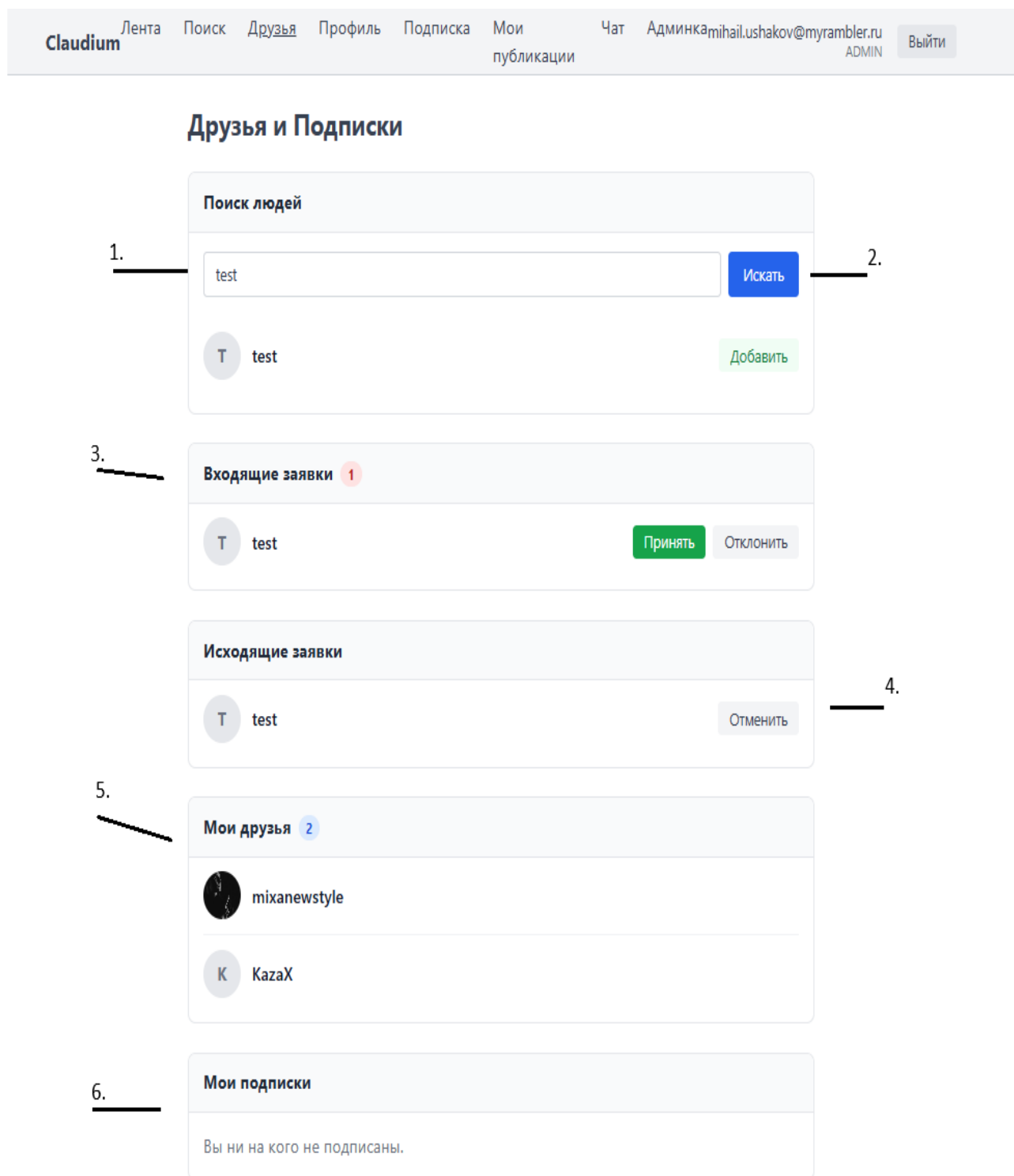


Рисунок 13 – Страница управления кругом общения «Друзья и подписки»

На рисунке цифрами отмечены следующие компоненты:

- 1 – поле поиска пользователей по имени;
- 2 – кнопка отправки заявки в друзья;
- 3 – блок входящих заявок с кнопками «Принять» и «Отклонить»;
- 4 – блок исходящих заявок с кнопкой «Отменить»;
- 5 – список друзей пользователя;
- 6 – список подписок на авторов.

На рисунке 14 представлен макет страницы автора.

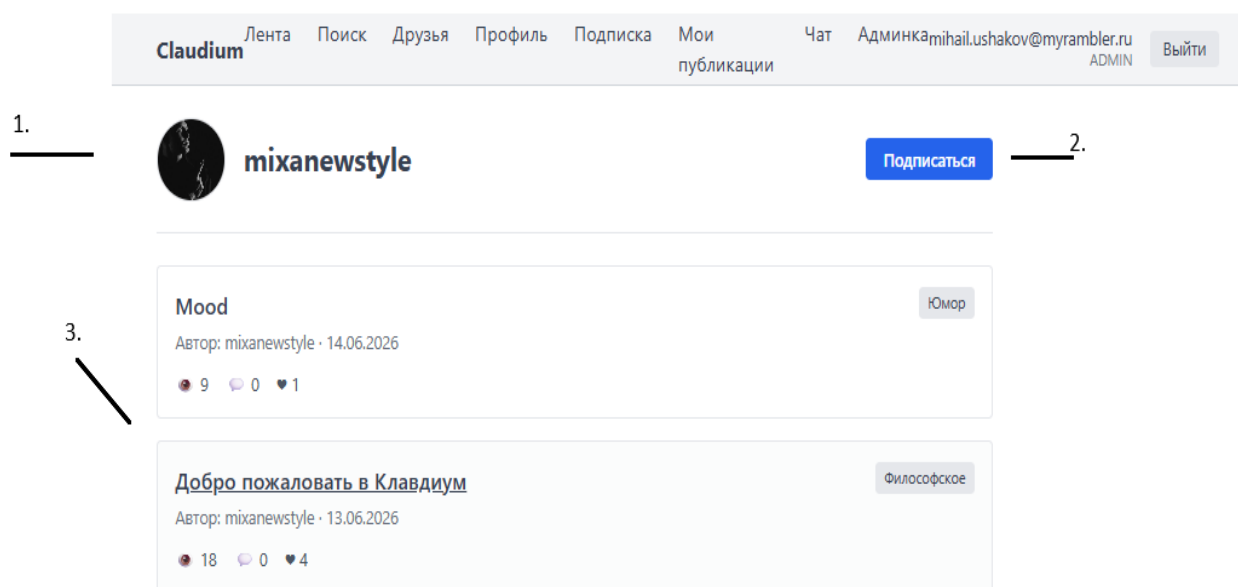


Рисунок 14 – Страница автора

На рисунке цифрами отмечены следующие компоненты:

- 1 – блок информации об авторе;
- 2 – кнопка подписки на автора;
- 3 – список публикаций автора.

Страница автора реализует ключевую идею платформы – знакомство со специалистом через его профессиональный контент: пользователь изучает публикации психолога и подписывается на его обновления.

На рисунке 15 представлен макет личного кабинета пользователя.

Мой профиль

Профиль [Редактировать](#)

Email mi hail.ushakov@myrambler.ru	Роль ADMIN
Имя пользователя (username) admin	Полное имя Михаил Ушаков
Дата рождения Не указана	Пол Не указан
Контактная информация Не указана	
О себе (bio) Не указано	

Анкета (контекст для ИИ)
Заполните эти поля, чтобы ИИ-терапевт лучше понимал вашу ситуацию. Это необязательно.

О себе

Расскажите о себе в свободной форме

Что привело

Над чем хотите поработать

Опыт терапии

Не указано

Символов: 2 / 1000

[Сохранить анкету](#)

Рисунок 15 – Личный кабинет пользователя

На рисунке цифрами отмечены следующие компоненты:

- 1 – блок данных профиля: имя пользователя, полное имя, дата рождения, пол, контактная информация, поле «О себе»;
- 2 – отображение текущей роли пользователя;

3 – кнопка перехода в режим редактирования профиля с кнопками «Сохранить» и «Отмена»;

4 – психологическая анкета (контекст для ИИ): поля «О себе», «Что привело», «Над чем хотите поработать», «Опыт терапии»;

5 – кнопка сохранения анкеты.

Психологическая анкета заполняется в свободной форме и используется исключительно как контекст для ИИ-модератора терапевтических сессий; в соответствии с требованиями безопасности ее содержимое недоступно другим пользователям системы.

На рисунке 16 представлен макет страницы управления подпиской.

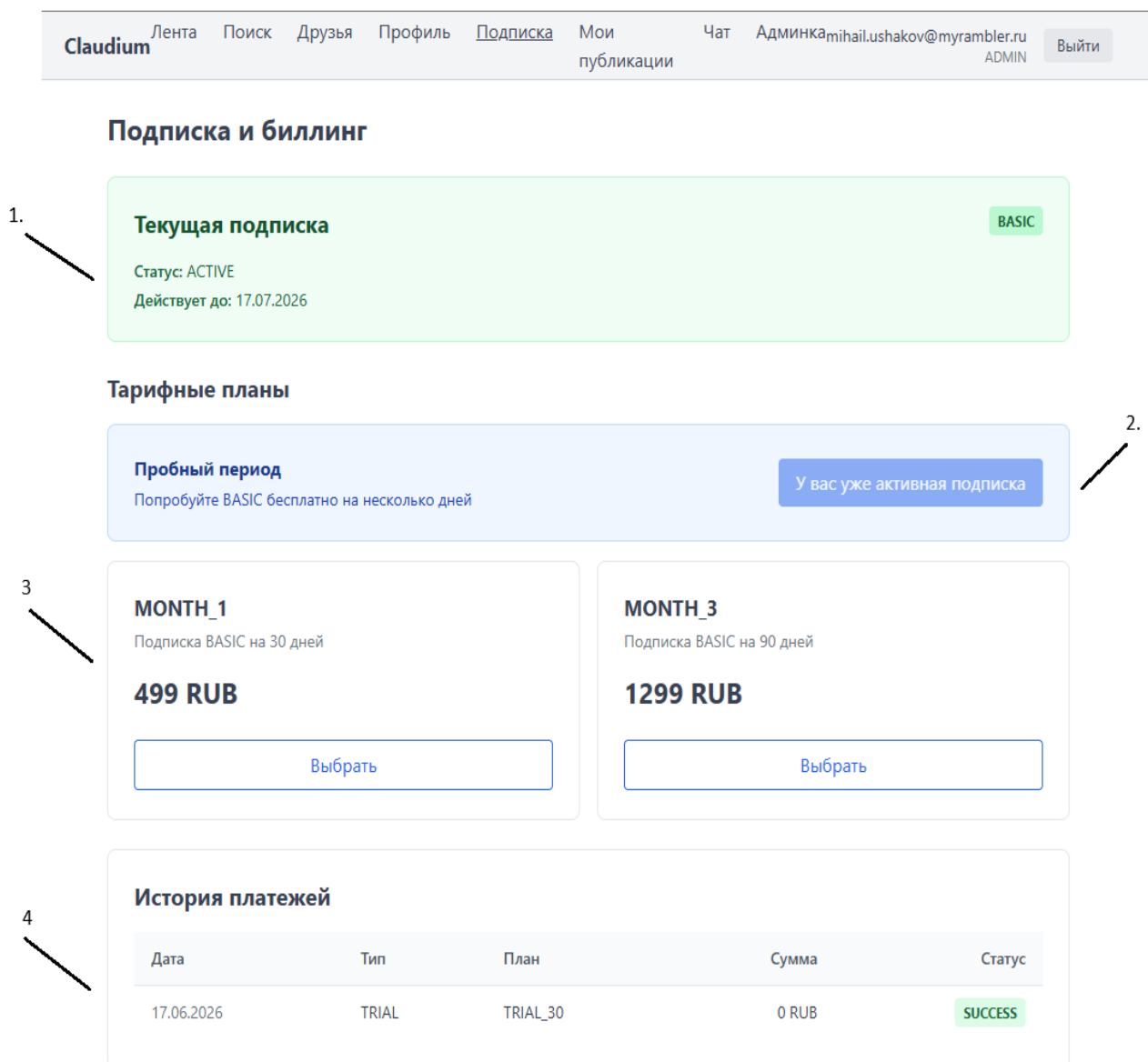


Рисунок 16 – Страница «Подписка и биллинг»

На рисунке цифрами отмечены следующие компоненты:

- 1 – карточка статуса текущей подписки с датой окончания действия;
- 2 – блок пробного периода с кнопкой активации;
- 3 – список тарифных планов с кнопками выбора;
- 4 – панель оформления подписки: длительность, сумма к оплате;
- 5 – поле ввода промокода с кнопкой проверки;
- 6 – история транзакций пользователя.

На рисунке 17 представлен макет студии автора.

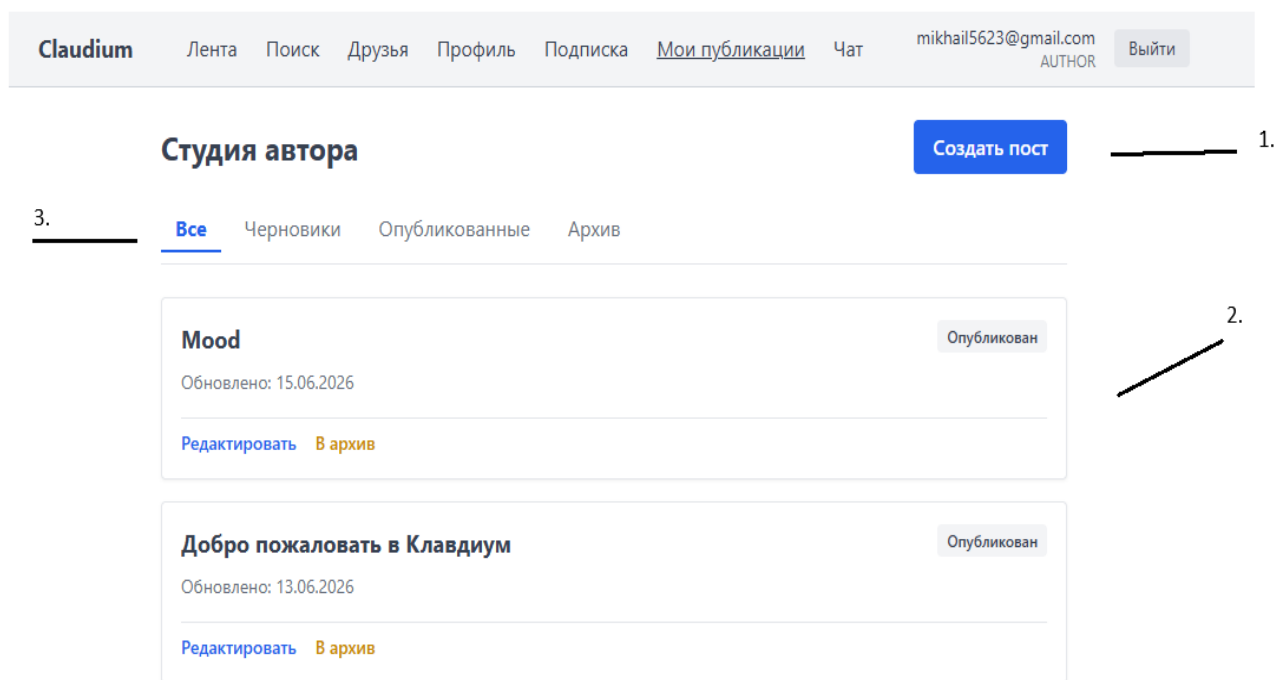


Рисунок 17 – Студия автора

На рисунке цифрами отмечены следующие компоненты:

- 1 – кнопка создания нового поста;
- 2 – список публикаций автора;
- 3 – кнопки сортировки публикаций автора.

Студия автора охватывает полный жизненный цикл публикации: материал создается как черновик, дорабатывается и затем публикуется в контентную ленту, после чего становится доступен подписчикам автора и пользователям сайта.

На рисунке 18 представлен макет страницы списка ИИ-терапевтических сессий.

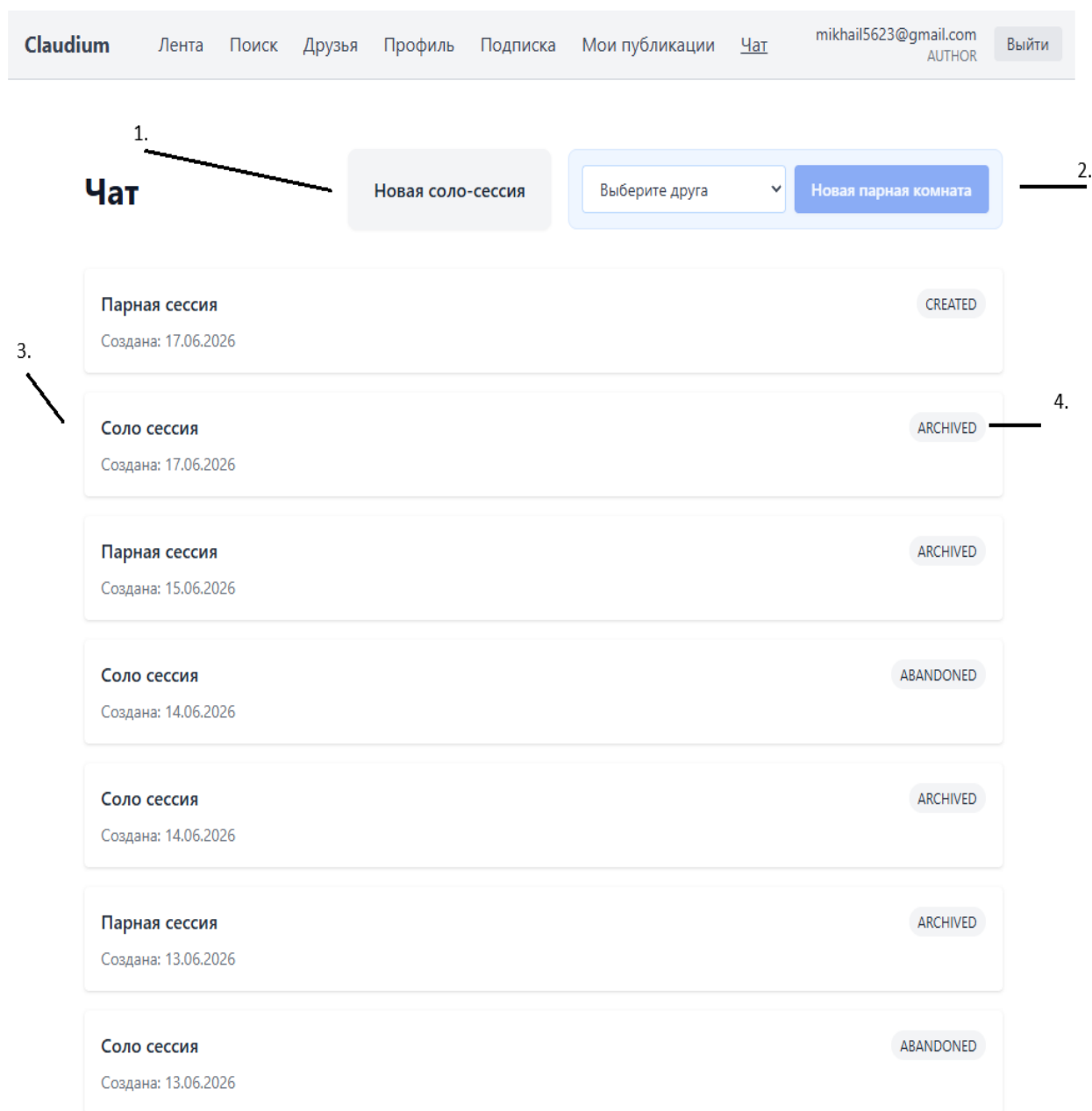


Рисунок 18 – Страница списка ИИ-терапевтических сессий

На рисунке цифрами отмечены следующие компоненты:

1 – кнопка создания индивидуальной сессии («Соло-дневник»);

2 – кнопка создания парной сессии с выбором друга из списка;

3 – список сессий пользователя с указанием типа и статуса;

4 – переход в архивированную сессию для чтения.

На рисунке 19 представлен макет окна ИИ-терапевтической сессии.

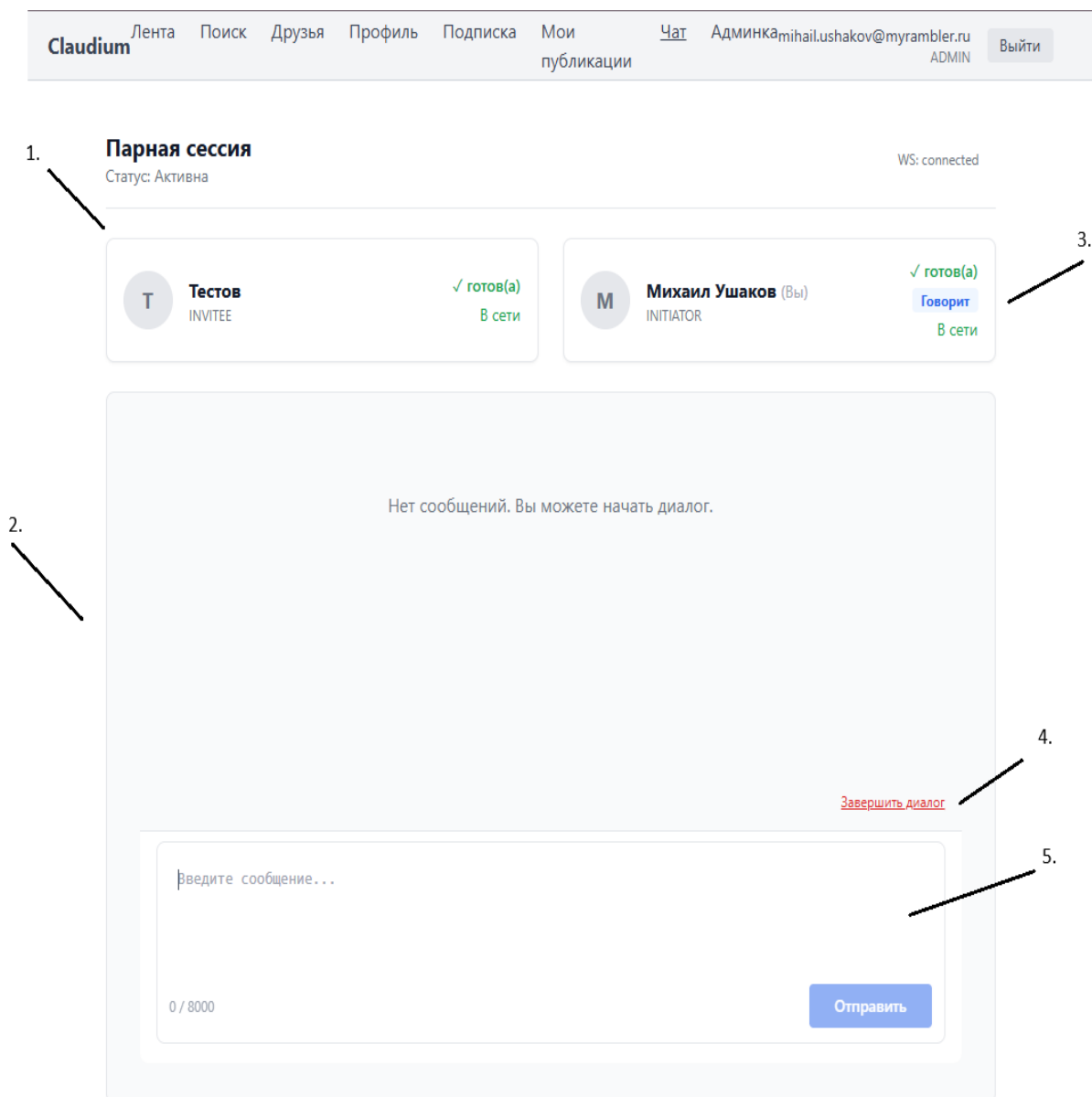


Рисунок 19 – Окно парной ИИ-терапевтической сессии

На рисунке цифрами отмечены следующие компоненты:

1 – заголовок сессии с индикаторами присутствия участников («В сети» / «Не в сети»);

2 – область переписки: сообщения участников и ответы ИИ-модератора;

3 – индикатор хода («Говорит...»);

4 – кнопка «Завершить диалог» и панель подтверждения завершения вторым участником;

5 – поле ввода сообщения с кнопкой отправки, активное только для участника, владеющего ходом.

Окно сессии отражает спроектированный в разделе 1.3 сценарий: интерфейс визуализирует очередность ходов, блокирует ввод для наблюдающего участника и проводит обоих участников через взаимные подтверждения начала и завершения диалога.

На рисунке 20 представлен макет страницы управления пользователями административной панели.

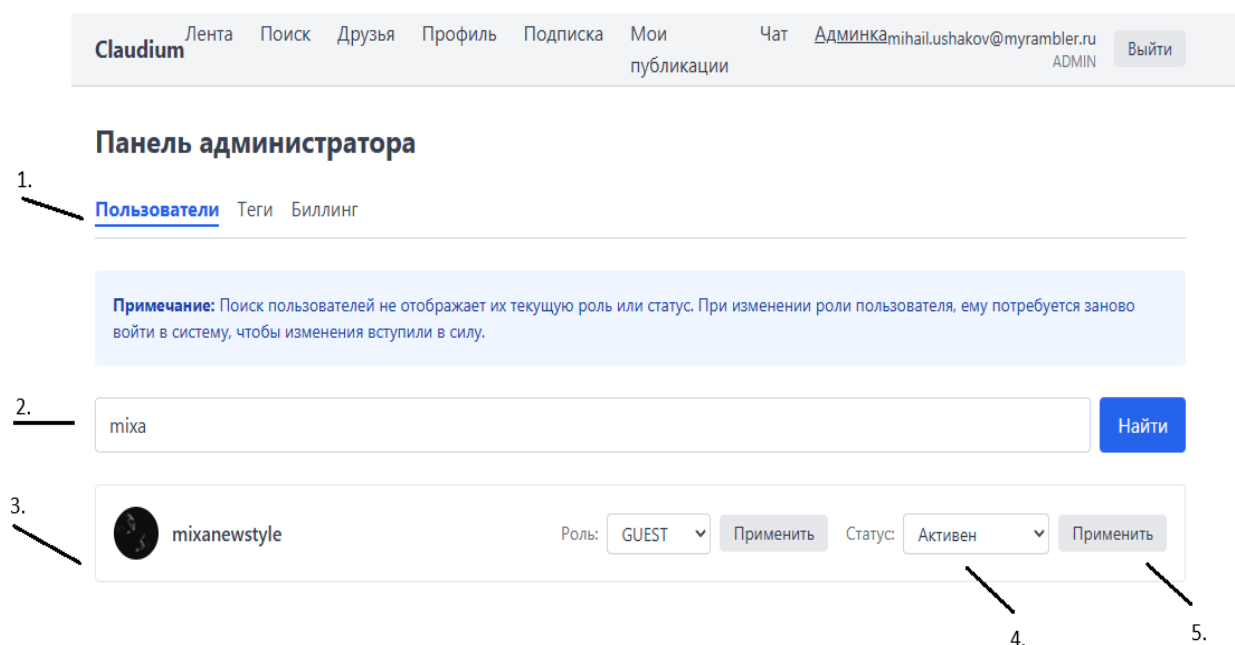


Рисунок 20 – Страница управления пользователями

На рисунке цифрами отмечены следующие компоненты:

1 – вкладки административной панели: «Пользователи», «Теги», «Биллинг»;

2 – поле поиска пользователя по имени;

3 – список найденных пользователей;

4 – элементы изменения роли пользователя с кнопкой «Применить»;

5 – элементы изменения статуса аккаунта (активен, заблокирован).

На рисунке 21 представлен макет страницы управления тегами.

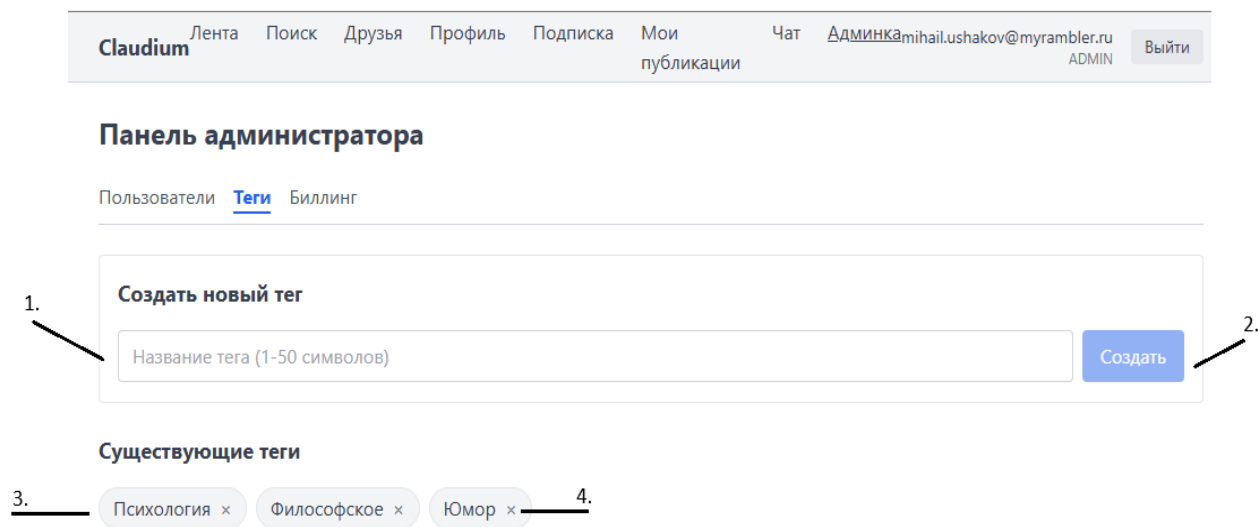


Рисунок 21 – Страница управления тегами

На рисунке цифрами отмечены следующие компоненты:

1 – поле ввода названия нового тега;

2 – кнопка создания тега;

3 – список существующих тегов;

4 – кнопка удаления тега.

На рисунке 22 представлен макет страницы управления биллингом.

Панель администратора

Пользователи Теги [Биллинг](#)

1.

Промокоды

Код Тип скидки Значение Макс. исп. Начало Окончание

Процент (%) 10 100 ДД.ММ.ГГГГ --: ДД.ММ.ГГГГ --:

3.

Код	Скидка	Использования	Действует	Активен
WELCOME	100 %	0 / 100	С: 16.06.2026, 21:40 По: 16.09.2026, 21:40	ДА

Транзакции

Важно: При возврате средств пользователю необходимо перезайти в систему для обновления уровня подписки.

User ID Статус С (Дата) По (Дата)

Все ДД.ММ.ГГГГ ДД.ММ.ГГГГ

Дата	User ID	План/Тип	Статус	Сумма	Действия
15.06.2026, 16:16	41eafbf...	TRIAL_30 (TRIAL)	SUCCESS	0 RUB	
15.06.2026, 14:41	4a9da9b5...	TRIAL_30 (TRIAL)	SUCCESS	0 RUB	
15.06.2026, 00:18	cb1cd882...	MONTH_1 (PURCHASE)	SUCCESS	499 RUB	Возврат
14.06.2026, 23:27	428e9582...	TRIAL_30 (TRIAL)	SUCCESS	0 RUB	
14.06.2026, 22:57	ada3f794...	MONTH_1 (PURCHASE)	SUCCESS	499 RUB	Возврат
13.06.2026, 17:28	d99eb259...	TRIAL_30 (TRIAL)	SUCCESS	0 RUB	

4.

Выдать подписку

Поиск пользователя (опционально)

Искать

Форма выдачи (BASIC)

User ID

Дней

Примечание (опц.)

5.

Рисунок 22 – Страница управления биллингом

На рисунке цифрами отмечены следующие компоненты:

- 1 – вкладки раздела: «Промокоды», «Транзакции», «Выдать подписку»;
- 2 – форма создания промокода: код, тип скидки (процентная или фиксированная), значение, максимальное число использований, период действия;
- 3 – список существующих промокодов с признаком активности;
- 4 – список транзакций системы;
- 5 – форма ручной выдачи подписки пользователю с указанием причины.

Спроектированные страницы покрывают все маршруты клиентского приложения и все автоматизируемые бизнес-процессы, определенные в

таблице 1: от регистрации и работы с контентом до проведения ИИ-терапевтических сессий и администрирования системы. Единый каркас с боковой навигацией и ролевым отображением разделов обеспечивает выполнение требований к интерфейсу, сформулированных в подразделе 1.4.2.2.

2.3 Проектирование серверной части

В серверной части информационной системы выделены пять микросервисов, рассмотренных в подразделах 2.3.1–2.3.5: сервис шлюза (API Gateway), сервис пользователей (User Service), сервис контента (Content Service), сервис ИИ-сессий (Chat Service) и сервис биллинга (Billing Service). Каждый из них спроектирован в соответствии с принципами многослойной архитектуры с четким разделением слоев данных, бизнес-логики, безопасности, обмена сообщениями и представления, что обеспечивает требуемый уровень модульности и упрощает дальнейшее сопровождение системы.

2.3.1 Проектирование сервиса шлюза

Сервис API-шлюза выступает единой точкой входа для всего внешнего HTTP-трафика, поступающего от клиентского приложения к серверной части системы. В отличие от прочих микросервисов, шлюз реализован как тонкий реверс-прокси на базе Spring Cloud Gateway и не содержит ни бизнес-логики, ни собственного слоя хранения данных: его единственной задачей является прием входящих запросов, их сопоставление с predetermined правилами маршрутизации и перенаправление к соответствующему нижележащему микросервису. Валидация JWT-токенов и проверка прав доступа выполняются непосредственно на целевых сервисах, что позволяет избежать дублирования логики и сохранить шлюз максимально нейтральным.

С учетом узкой специализации модуля для разработки была выбрана однослойная архитектура с единственным слоем маршрутизации и конфигурации, отвечающим за декларативное описание маршрутов, политики кросс-доменных запросов и параметров подключения к нижележащим сервисам.

Вся конфигурация маршрутизации вынесена в файл `application.yml`, благодаря чему правила перенаправления могут изменяться без перекомпиляции исходного кода приложения.

Структура сервиса представлена на диаграмме классов на рисунке 23.

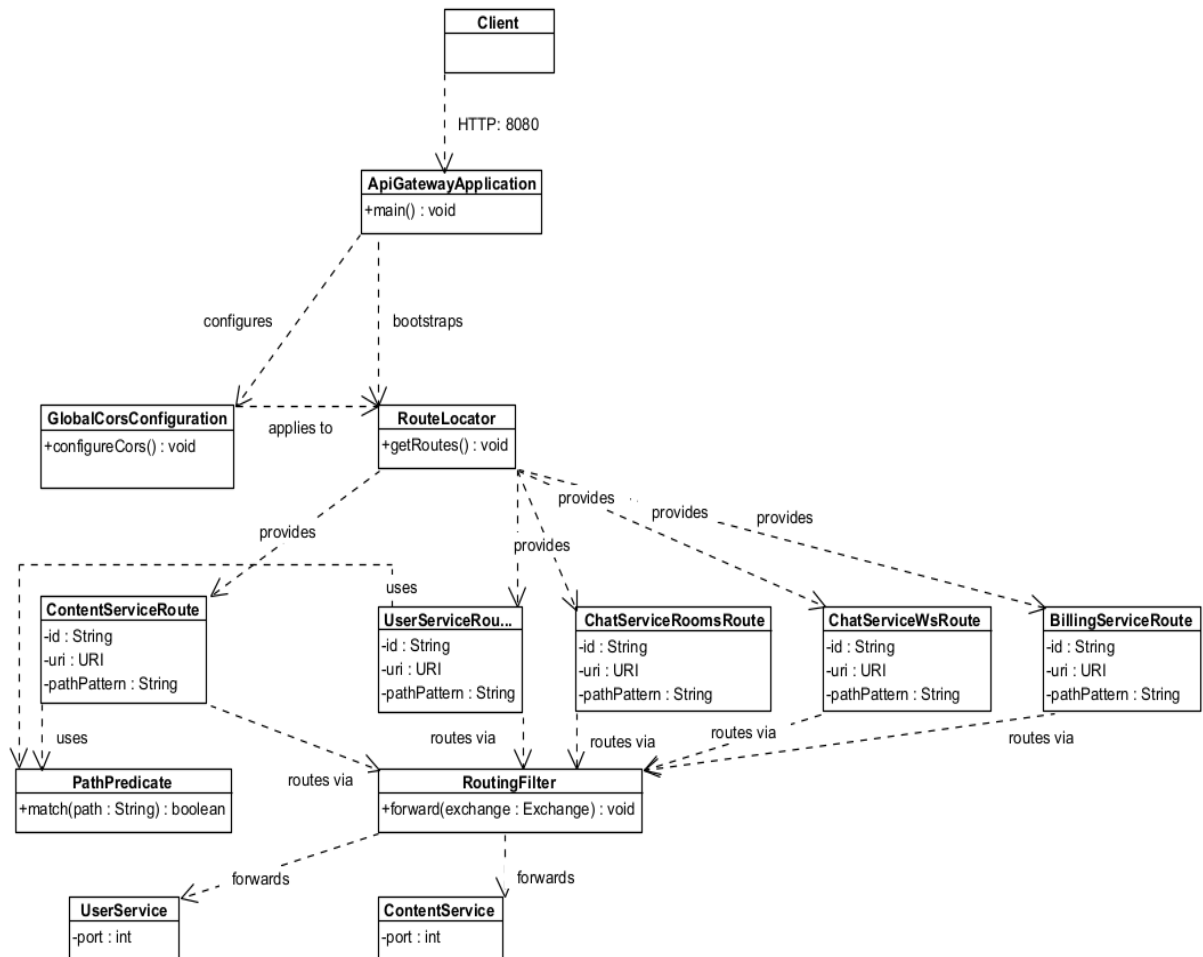


Рисунок 23 – Диаграмма классов сервиса шлюза

Согласно данной схеме можно выделить следующие составные элементы:

- `ApiGatewayApplication` – точка входа микросервиса; при запуске инициализирует встроенный реактивный сервер Netty и активирует механизм автоконфигурации Spring Cloud Gateway, считывающий определения маршрутов из конфигурационного файла;
- `GlobalCorsConfiguration` – глобальная политика кросс-доменных запросов, применяемая ко всем маршрутам и обеспечивающая

беспрепятственное взаимодействие клиентского приложения с серверной частью;

– набор предикатов маршрутов на основе шаблона пути (Path Predicate), каждый из которых идентифицируется уникальным значением id, целевым URI сервиса и шаблоном пути.

Перечень маршрутов, обрабатываемых сервисом шлюза, представлен в таблице 7.

Таблица 7 – Маршруты сервиса API Gateway

Шаблон пути	Целевой сервис	Описание
/api/v1/users/	User Service	Регистрация, аутентификация, профили, поиск пользователей, подписки
/api/v1/social/	User Service	Социальные функции: подписки на авторов, заявки в друзья
/api/v1/admin/users/	User Service	Административная модерация учетных записей
/api/v1/posts/	Content Service	Жизненный цикл публикаций, оценки, комментарии к постам, поиск
/api/v1/feed	Content Service	Персонализированная лента публикаций
/api/v1/tags/	Content Service	Просмотр справочника тегов
/api/v1/comments/	Content Service	Операции с отдельными комментариями
/api/v1/authors/*/posts	Content Service	Список публикаций конкретного автора
/internal/v1/admin/posts/, /internal/v1/admin/tags/, /internal/v1/admin/comments/	Content Service	Административная модерация контента и управление тегами
/api/v1/rooms/	Chat Service	Управление ИИ-терапевтическими сессиями
/ws/	Chat Service	WebSocket-соединения терапевтических сессий
/internal/v1/admin/rooms/, /internal/v1/admin/llm/	Chat Service	Административный просмотр сессий и управление ИИ-провайдером
/api/v1/billing/	Billing Service	Тарифы, оформление подписки, промокоды, транзакции пользователя
/api/v1/admin/billing/	Billing Service	Административное управление биллингом

Декларативная маршрутизация покрывает все публичные и административные точки доступа системы, а добавление нового сервиса сводится к описанию нового маршрута в конфигурационном файле.

2.3.2 Проектирование сервиса пользователей

Сервис пользователей является единым источником истины по идентификации пользователей, их ролям и социальному графу. Помимо операций с учетной записью, сервис обеспечивает хранение психологического профиля пользователя и предоставляет другим сервисам пакетный API для обогащения ответов публичными данными профилей.

При разработке сервиса была выбрана многослойная архитектура для четкой структуры компонентов и поддержки высокой модульности программы. Были выделены следующие слои:

- слой данных (Data Layer) – JPA-сущности, соответствующие таблицам схемы `user_schema`, Spring Data JPA репозитории для CRUD-операций, перечисления справочных таблиц (роли, статусы) и составные ключи для связей «многие-ко-многим»;

- слой бизнес-логики (Business Logic Layer) – сервисные классы, реализующие сценарии регистрации пользователей, управления профилями, модерации учетных записей и социальные функции (подписки на авторов, заявки в друзья); слой отвечает за валидацию бизнес-правил и генерацию доменных исключений;

- слой безопасности (Security Layer) – аутентификация и авторизация на основе JWT-токенов: сервис генерации и валидации токенов, фильтр аутентификации запросов и обработчики ошибок безопасности;

- слой обмена сообщениями (Messaging Layer) – реализация паттерна Transactional Outbox для надежной асинхронной доставки событий через RabbitMQ: фабрика создания событий, планировщик их публикации и конфигурация брокера;

– слой представления (Presentation Layer) – REST-контроллеры для обработки HTTP-запросов, модели запросов и ответов (DTO) и централизованный обработчик исключений.

Структура сервиса представлена на диаграмме классов на рисунке 24.

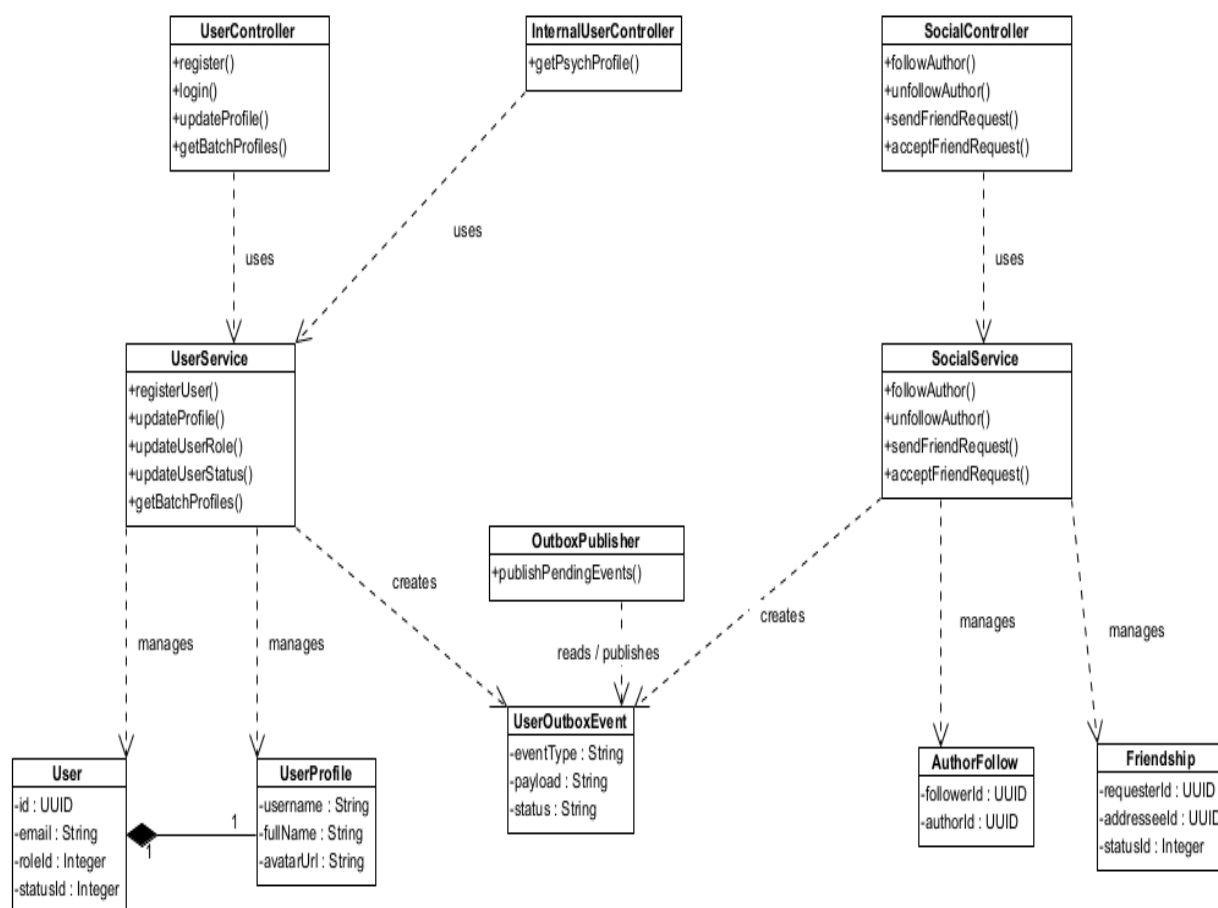


Рисунок 24 – Диаграмма классов сервиса пользователей

Согласно данной схеме можно выделить следующие ключевые элементы. Контроллеры **UserController**, **SocialController** обрабатывают соответственно запросы учетных записей, социальных операций; **InternalUserController** предоставляет внутренний интерфейс выдачи психологической анкеты сервису ИИ-сессий. Сервисные классы **UserService** и **SocialService** инкапсулируют бизнес-логику: регистрацию с хешированием пароля, частичное обновление профиля с выборочной генерацией outbox-событий, подписку на автора с проверкой роли AUTHOR, обработку заявок в друзья с контролем состояния.

OutboxPublisher по расписанию публикует накопленные события в обменник user.events.exchange. Слой данных представлен сущностями User, UserProfile, Friendship, AuthorFollow и UserOutboxEvent с соответствующими репозиториями; учетные данные авторизации (User) физически отделены от публичного профиля (UserProfile), что соответствует требованиям безопасности из подраздела 1.4.2.1.

Для взаимодействия с сервисом пользователей были разработаны методы API, описание которых приведено в таблице 8.

Таблица 8 – Методы API сервиса User Service

Метод	Точка доступа	Описание
POST	/api/v1/users/register	Регистрация нового пользователя
POST	/api/v1/users/login	Аутентификация пользователя и получение JWT-токена
GET	/api/v1/users/me/profile	Получение профиля текущего пользователя
PUT	/api/v1/users/me/profile	Частичное обновление профиля и психологической анкеты
GET	/api/v1/users/search	Поиск пользователей по имени
POST	/api/v1/users/batch	Пакетное получение кратких профилей по списку UUID
GET	/api/v1/users/{userId}/follows	Получение списка подписок пользователя на авторов
POST	/api/v1/social/me/follow/{authorId}	Подписка на автора
DELETE	/api/v1/social/me/follow/{authorId}	Отписка от автора
GET	/api/v1/social/me/friends	Получение списка друзей и заявок
POST	/api/v1/social/me/friends/request/{addresseeId}	Отправка заявки в друзья
POST	/api/v1/social/me/friends/{requesterId}/accept	Принятие заявки в друзья
POST	/api/v1/social/me/friends/{requesterId}/decline	Отклонение заявки в друзья
DELETE	/api/v1/social/me/friends/{addresseeId}	Отмена исходящей заявки в друзья
PUT	/api/v1/admin/users/{userId}/role	Изменение роли пользователя (только ADMIN)
PUT	/api/v1/admin/users/{userId}/status	Изменение статуса учетной записи (только ADMIN)
GET	/internal/v1/users/{userId}/psych-profile	Внутренняя выдача психологической анкеты сервису ИИ-сессий

Методы API сервиса предоставляют возможности для полного управления учетными записями, профилями, социальным графом и административной модерацией.

2.3.3 Проектирование сервиса контента

Сервис контента отвечает за полный жизненный цикл публикаций авторского контента: создание черновика, частичное редактирование, публикацию, архивирование, а также за древовидные комментарии, оценки публикаций, персонализированную ленту, полнотекстовый поиск и справочник тегов. Сервис является единственным источником истины для схемы `content_schema` и не дублирует справочник пользователей: при необходимости отображения имени или аватара автора применяется пакетный запрос к сервису пользователей с последующим кэшированием результата в Redis.

При разработке сервиса были выделены следующие слои:

- слой данных (Data Layer) – JPA-сущности таблиц схемы `content_schema`, Spring Data репозитории и перечисления справочных значений (статусы публикаций);
- слой бизнес-логики (Business Logic Layer) – сервисные классы жизненного цикла публикаций (черновик → публикация → архивирование), управления комментариями, оценками, лентой, поиском и тегами, включая валидацию перед публикацией и маппинг сущностей в DTO;
- слой безопасности (Security Layer) – JWT-фильтр и сервис валидации токенов: токен выпускается сервисом пользователей, а сервис контента валидирует его самостоятельно;
- слой обмена сообщениями (Messaging Layer) – реализация паттерна Transactional Outbox: бизнес-мутации и событийные записи сохраняются в одной транзакции, а планировщик публикует события в RabbitMQ;
- слой обработки исключений (Exception Layer) – централизованное преобразование доменных исключений в стандартизированные JSON-ответы об ошибках.

Структура сервиса представлена на диаграмме классов на рисунке 25.

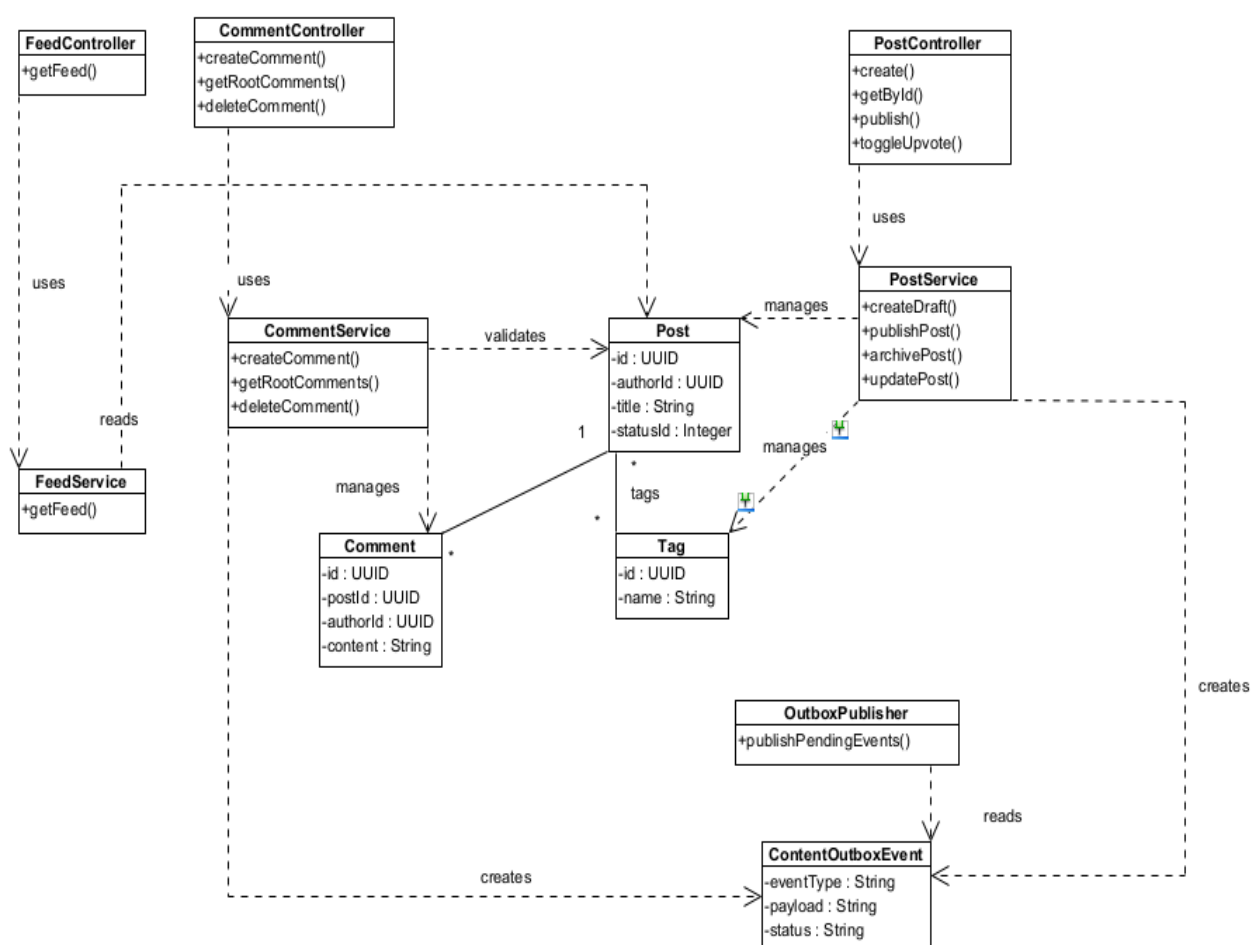


Рисунок 25 – Диаграмма классов сервиса контента

Согласно данной схеме можно выделить следующие ключевые элементы. Слой представления образуют контроллеры: PostController (жизненный цикл публикаций и оценки), CommentController (древовидные комментарии и ответы), FeedController (персонализированная лента), SearchController (полнотекстовый поиск), TagController и AuthorPageController (страница автора), а также административные контроллеры AdminPostController, AdminCommentController и AdminTagController, защищенные ролью ADMIN. Центральный сервис PostService инкапсулирует бизнес-логику публикаций с контролем доступа по статусу и роли, а при публикации и архивировании формирует outbox-события. Сущность Post хранит контент в формате JSONB, массив ключевых слов, денормализованные счетчики просмотров, оценок и комментариев и номер версии для оптимистичной блокировки; сущности Comment, Tag и PostUpvote

дополняют модель данных, а связь публикаций с тегами реализована отношением «многие-ко-многим».

Для взаимодействия с сервисом контента были разработаны методы API, описание которых приведено в таблице 9.

Таблица 9 – Методы API сервиса Content Service

Метод	Точка доступа	Описание
POST	/api/v1/posts	Создание черновика публикации (роль AUTHOR)
GET	/api/v1/posts/mine	Получение списка публикаций текущего автора
GET	/api/v1/posts/{postId}	Получение публикации по идентификатору
PATCH	/api/v1/posts/{postId}	Частичное обновление публикации
DELETE	/api/v1/posts/{postId}	Удаление черновика
POST	/api/v1/posts/{postId}/publish	Публикация черновика
POST	/api/v1/posts/{postId}/archive	Архивирование публикации
POST	/api/v1/posts/{postId}/upvote	Оценка публикации
GET	/api/v1/feed	Получение персонализированной ленты с пагинацией
GET	/api/v1/posts/search	Полнотекстовый поиск публикаций
GET	/api/v1/authors/{authorId}/posts	Список публикаций автора
POST	/api/v1/posts/{postId}/comments	Добавление комментария к публикации
GET	/api/v1/posts/{postId}/comments	Получение комментариев публикации
GET	/api/v1/comments/{commentId}/replies	Получение ответов на комментарий
PATCH	/api/v1/comments/{commentId}	Редактирование комментария
DELETE	/api/v1/comments/{commentId}	Удаление комментария
GET	/api/v1/tags	Просмотр справочника тегов
PATCH, DELETE	/internal/v1/admin/posts/{postId}	Административная модерация публикации
PATCH, DELETE	/internal/v1/admin/comments/{commentId}	Административная модерация комментария
POST, DELETE	/internal/v1/admin/tags	Административное управление тегами

Методы API сервиса покрывают контентные и социальные сценарии платформы: от авторского цикла создания публикаций до читательских операций с лентой, поиском, оценками и многоуровневыми комментариями.

2.3.4 Проектирование сервиса чата

Сервис чата реализует ключевую функциональную особенность системы – индивидуальные и парные ИИ-модерируемые терапевтические сессии. Сервис отвечает за жизненный цикл комнат, очередность ходов участников, хранение сообщений, приглашения по ссылке, а также за взаимодействие с внешней языковой моделью.

Центральным элементом проектирования сервиса является модель комнаты с явной машиной состояний. Комната создается в состоянии «Создана» (CREATED), после присоединения второго участника переходит в состояние «Ожидание согласия» (WAITING_CONSENT), а после взаимного подтверждения готовности обоих участников – в состояние «Активна» (ACTIVE). Предложение завершить диалог переводит комнату в состояние «Завершение» (ENDING), из которого она либо возвращается в активное состояние при отказе второго участника, либо переходит в терминальное состояние «Архивирована» (ARCHIVED). Для индивидуальных сессий состояния согласия исключаются: комната активируется сразу и архивируется по единственному подтверждению владельца. Диаграмма состояний комнаты представлена на рисунке 26.

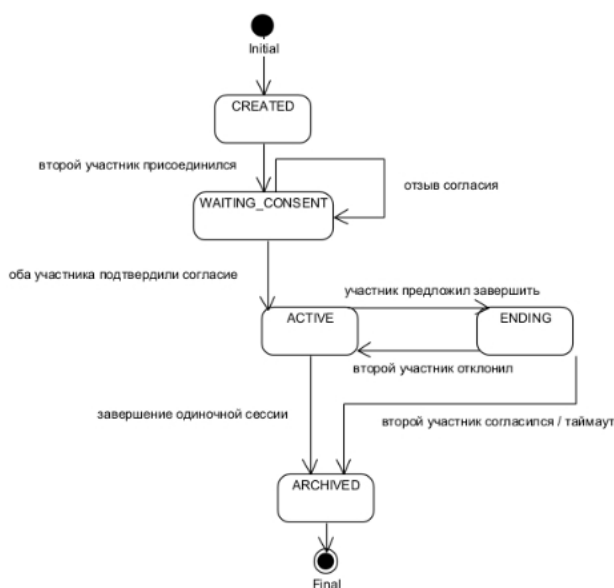


Рисунок 26 – Диаграмма состояний терапевтической комнаты

В активном состоянии комнатой управляет механизм очередности ходов: правом голоса владеет один участник, определяемый полем текущего владельца хода. Сообщения владельца хода накапливаются как эфемерные черновики в Redis и транслируются собеседнику в реальном времени; команда завершения мысли упаковывает черновики в неизменяемый ход, передает диалог на анализ внешней языковой модели, а после получения ответа ИИ право голоса переходит ко второму участнику. Идемпотентность команды обеспечивается монотонным порядковым номером хода, а конкурентные переходы состояний сериализуются оптимистичной блокировкой записи комнаты.

При разработке сервиса были выделены следующие слои:

- слой данных (Data Layer) – JPA-сущности схемы chat_schema: Room (комната), RoomParticipant (участник), Turn (ход диалога), Invite (приглашение), FriendLink (локальная проекция дружеских связей) и ChatOutboxEvent, с соответствующими репозиториями;

- слой бизнес-логики (Business Logic Layer) – сервисы жизненного цикла комнат и состояний, черновиков, приглашений, присутствия участников, лимитирования частоты обращений к ИИ и обогащения ответов данными профилей;

- слой интеграции с ИИ (LLM Layer) – абстракция клиента языковой модели: интерфейс LlmClient с реализацией, совместимой с OpenAI-протоколом, и сборщик контекста ConversationAssembler, формирующий запрос из системного промпта, психологических анкет участников и истории диалога;

- слой безопасности (Security Layer) – JWT-фильтр для REST-запросов и перехватчик аутентификации WebSocket-канала; помимо зарегистрированных пользователей поддерживается гостевой принципал с токеном, ограниченным одной комнатой;

- слой обмена сообщениями (Messaging Layer) – Transactional Outbox для публикации событий (отправка приглашения, архивация комнаты) и потребитель событий сервиса пользователей для поддержания локальной проекции дружеских связей и ролей;

– слой представления (Presentation Layer) – REST-контроллеры управления комнатами и приглашениями, WebSocket-контроллеры команд реального времени и административный контроллер инспекции комнат.

Отдельного внимания заслуживает слой интеграции с ИИ. Доступ к внешней языковой модели абстрагирован интерфейсом провайдера, что позволяет заменить поставщика без изменения бизнес-логики. Обращение к модели выполняется асинхронно: на время обработки комната переходит в фазу ожидания, а участникам транслируется индикатор «ИИ печатает». Психологические анкеты участников запрашиваются у сервиса пользователей по внутреннему интерфейсу и включаются в контекст запроса, что обеспечивает персонализацию модерации без раскрытия анкет собеседнику.

Структура сервиса представлена на диаграмме классов на рисунке 27.

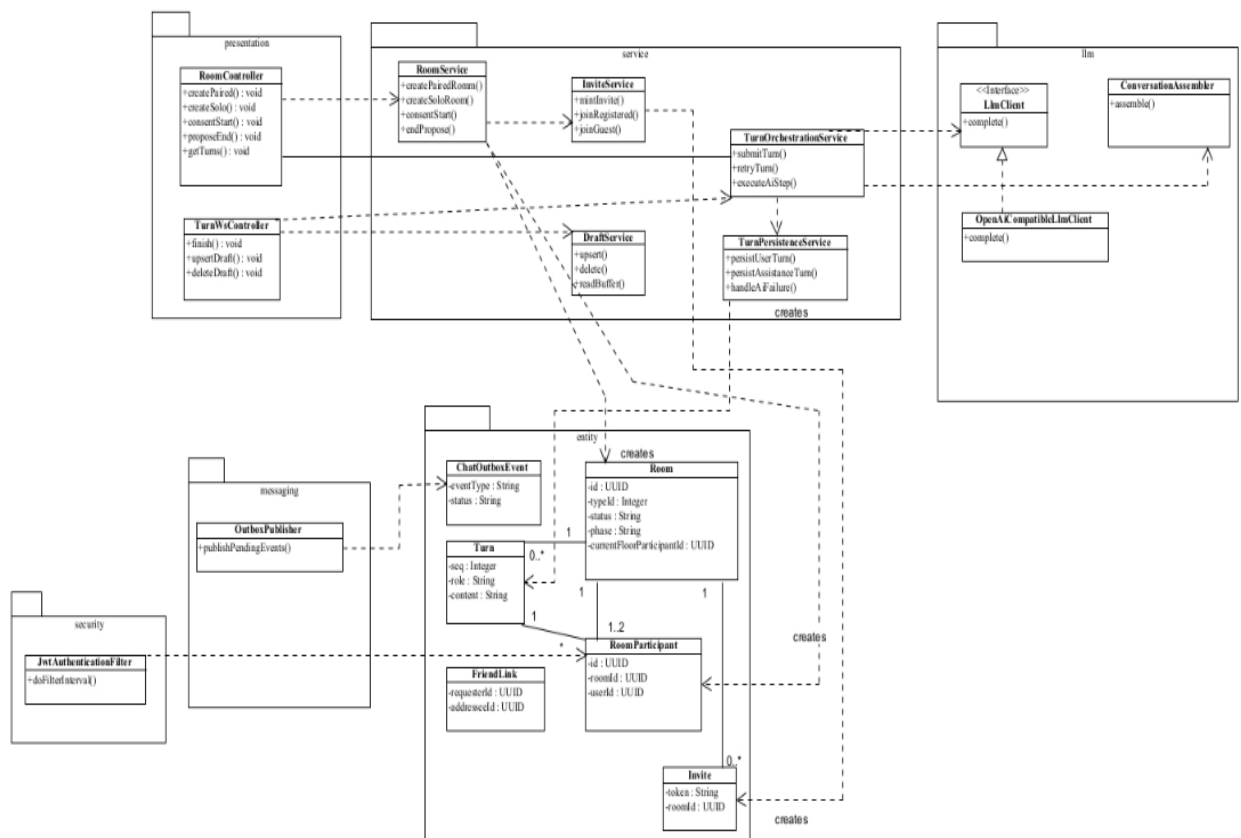


Рисунок 27 – Диаграмма классов сервиса ИИ-сессий

Для взаимодействия с сервисом были разработаны методы REST API, описание которых приведено в таблице 10.

Таблица 10 – Методы REST API сервиса Chat Service

Метод	Точка доступа	Описание
POST	/api/v1/rooms/solo	Создание индивидуальной сессии
POST	/api/v1/rooms/paired	Создание парной сессии (требуется активная подписка)
POST	/api/v1/rooms/{roomId}/join	Присоединение приглашенного друга к комнате
POST	/api/v1/rooms/{roomId}/consent/start	Подтверждение согласия на начало диалога
POST	/api/v1/rooms/{roomId}/consent/revoke	Отзыв согласия до начала диалога
POST	/api/v1/rooms/{roomId}/turns	Отправка хода (завершение мысли)
POST	/api/v1/rooms/{roomId}/turns/retry	Повтор обращения к ИИ после сбоя
GET	/api/v1/rooms/{roomId}/turns	Получение истории ходов с пагинацией
POST	/api/v1/rooms/{roomId}/end/propose	Предложение завершить диалог
POST	/api/v1/rooms/{roomId}/end/respond	Ответ второго участника на предложение
POST	/api/v1/rooms/{roomId}/end	Завершение индивидуальной сессии
GET	/api/v1/rooms, /api/v1/rooms/archived	Списки активных и архивных сессий пользователя
GET	/api/v1/rooms/{roomId}	Получение состояния комнаты
POST	/api/v1/invites/mint	Выпуск пригласительной ссылки
POST	/api/v1/invites/{token}/join, .../join/guest	Присоединение по ссылке (пользователь или гость)
GET, POST	/internal/v1/admin/rooms/{roomId}, .../terminate	Административная инспекция и принудительное завершение

Команды реального времени передаются по WebSocket-каналу через брокер STOMP; их перечень приведен в таблице 11.

Таблица 11 – WebSocket-команды сервиса Chat Service

Команда	Назначение
/rooms/{roomId}/draft/upsert	Создание или изменение черновика сообщения
/rooms/{roomId}/draft/delete	Удаление черновика
/rooms/{roomId}/finish	Завершение мысли и передача диалога на анализ ИИ
/rooms/{roomId}/consent/start, .../consent/revoke	Подтверждение и отзыв согласия на начало
/rooms/{roomId}/end/propose, .../end/agree, .../end/decline	Предложение, принятие и отклонение завершения диалога

Сочетание REST API для управляющих операций и WebSocket-команд для взаимодействия в реальном времени обеспечивает живой характер диалога при сохранении строгого контроля очередности ходов на стороне сервера.

2.3.5 Проектирование сервиса биллинга

Сервис биллинга является финансовым ядром системы: он владеет жизненным циклом подписок, журналом транзакций, промокодами и пробным периодом, а также служит источником сигнала о платном статусе пользователя, который сервис пользователей преобразует в изменение роли. Сервис является единственным источником истины для схемы `billing_schema`.

Для обеспечения финансовой прозрачности и возможности последующего аудита все изменения баланса и статусов сохраняются в неизменяемом журнале истории, что гарантирует защиту от потери данных при системных сбоях.

Подписки оформляются как разовые покупки фиксированной длительности без автопродления; повторная покупка при действующей подписке продлевает срок ее действия. Пробный период предоставляет доступ уровня BASIC однократно для каждого пользователя и недоступен при активной подписке. Промокоды поддерживают процентную и фиксированную скидки, ограничены по числу использований глобально и однократны для каждого пользователя; промокод, снижающий стоимость до нуля, активирует подписку немедленно без обращения к платежному провайдеру. По истечении срока действия подписки фоновое задание возвращает пользователя к роли FREE.

Ключевым проектным решением является пессимистичная сага выдачи роли: повышение роли пользователя выполняется только после подтверждения оплаты. Сценарий покупки выглядит следующим образом: клиент инициирует оформление, сервис создает транзакцию в состоянии ожидания и платежное намерение у провайдера; провайдер уведомляет систему вебхуком с проверкой подписи; при успешном статусе сервис активирует подписку и в той же транзакции базы данных формирует `outbox`-событие об изменении подписки; событие доставляется через RabbitMQ сервису пользователей, который повышает роль до BASIC. Такой порядок исключает необходимость компенсирующего понижения роли при неуспешной оплате. Взаимодействие с платежным провайдером вынесено за интерфейс `PaymentProvider`; в качестве реализации по умолчанию используется заглушка, эмулирующая полный цикл

«намерение – оплата – вебхук», а подключение реального российского провайдера сводится к конфигурационной задаче.

При разработке сервиса были выделены следующие слои:

- слой данных (Data Layer), включающий JPA-сущности схемы billing_schema: Plan (тарифный план), Subscription (подписка), Transaction (транзакция), PromoCode и PromoRedemption (промокоды и их погашения), BillingAccount (биллинговый счет с признаком права на пробный период), BillingOutboxEvent и справочные перечисления статусов, с соответствующими репозиториями;

- слой бизнес-логики (Business Logic Layer), объединяющий сервисы оформления покупки (CheckoutService), активации подписки (ActivationService), пробного периода (TrialService), промокодов (PromoService), обработки вебхуков (WebhookService), возвратов (RefundService), ручной выдачи подписки (GrantService) и фоновых заданий жизненного цикла (BillingScheduledJobs);

- слой платежной интеграции (Payment Layer), содержащий интерфейс PaymentProvider с моделями платежного намерения и вебхук-события и реализация-заглушка StubPaymentProvider;

- слой безопасности (Security Layer), обеспечивающий локальную валидацию JWT-токенов, аналогичная остальным сервисам; эндпоинты покупки доступны любому аутентифицированному пользователю, поскольку подписку оформляет пользователь с ролью FREE;

- слой обмена сообщениями (Messaging Layer), использующий Transactional Outbox для публикации событий изменения подписки и потребитель события регистрации пользователя, создающий биллинговый счет;

- слой представления (Presentation Layer), состоящий из REST-контроллеров пользовательских, административных и вебхук-операций.

Разделение ответственности между описанными слоями обеспечивает высокую тестопригодность кодовой базы и позволяет независимо модифицировать логику обработки платежей без затрагивания смежных компонентов.

Структура сервиса представлена на диаграмме классов на рисунке 28.

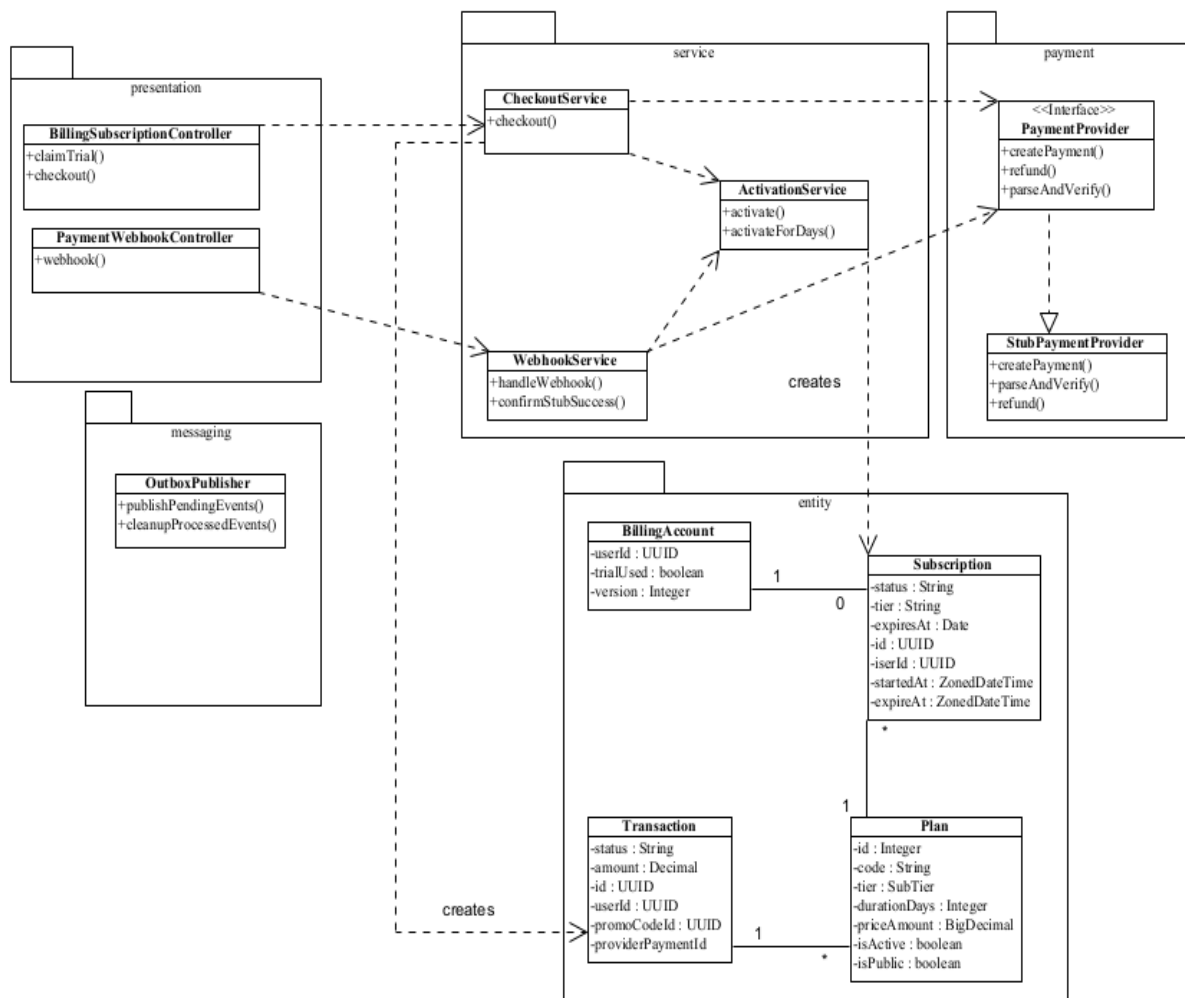


Рисунок 28 – Диаграмма классов сервиса биллинга

Для взаимодействия с сервисом были разработаны методы API, описание которых приведено в таблице 12.

Таблица 12 – Методы API сервиса Billing Service

Метод	Точка доступа	Описание
GET	/api/v1/billing/plans	Получение списка тарифных планов
GET	/api/v1/billing/me/subscription	Статус подписки текущего пользователя
GET	/api/v1/billing/me/transactions	История транзакций пользователя
POST	/api/v1/billing/subscriptions/trial	Активация пробного периода
POST	/api/v1/billing/subscriptions/checkout	Оформление покупки подписки

Продолжение таблицы 12

Метод	Точка доступа	Описание
POST	/api/v1/billing/promo/validate	Проверка применимости промокода
POST	/api/v1/billing/payments/webhook	Прием уведомления платежного провайдера
POST	/api/v1/billing/payments/stub/confirm/{transactionId}	Подтверждение оплаты в режиме заглушки
GET, POST	/api/v1/admin/billing/promo, .../promo/{id}	Административное управление промокодами
GET	/api/v1/admin/billing/transactions	Просмотр транзакций системы
POST	/api/v1/admin/billing/transactions/{id}/refund	Возврат средств по транзакции
POST	/api/v1/admin/billing/subscriptions/{userId}/grant	Ручная выдача подписки пользователю

Методы API сервиса покрывают полный финансовый контур платформы: от витрины тарифов и оформления покупки до административного контроля транзакций, предусмотренного для роли администратора в таблице 1.

Таким образом, в результате проектирования серверной части были определены зоны ответственности, слои, структуры классов и интерфейсы всех пяти микросервисов системы. Полученные результаты являются основой для проектирования базы данных и последующей реализации.

2.4 Проектирование базы данных

2.4.1 Разработка концептуальной модели базы данных

Концептуальная модель – это модель, отображающая данные и связи предметной области с учетом их семантического содержания, независимо от их представления в компьютере [12]. Несмотря на то, что физически данные системы разделены на четыре изолированные схемы по числу микросервисов, на концептуальном уровне целесообразно рассматривать единую модель предметной области: сущности разных схем связаны между собой семантически, а их разнесение по схемам является решением логического и физического уровней.

Для разработанной концептуальной модели базы данных было создано табличное описание сущностей, представленное в таблице 13.

Таблица 13 – Табличное описание сущностей для концептуальной модели базы данных

№ п/п	Наименование	Описание	Категория
1	Пользователь	Учетная запись: данные аутентификации, роль, статус	Стержневая
2	Профиль	Публичные данные пользователя и психологическая анкета	Характеристическая
3	Дружба	Заявка и состояние дружеской связи двух пользователей	Ассоциативная
4	Подписка на автора	Связь читателя с автором, на которого он подписан	Ассоциативная
5	Публикация	Авторская статья с обложкой, содержимым и счетчиками	Характеристическая
6	Тег	Элемент справочника тем для категоризации публикаций	Стержневая
7	Тег публикации	Связь публикации с присвоенным ей тегом	Ассоциативная
8	Комментарий	Древовидный комментарий пользователя к публикации	Характеристическая
9	Оценка публикации	Факт оценки публикации пользователем	Ассоциативная
10	Запись модерации	Журнальная запись административного действия над контентом	Характеристическая
11	Комната	Терапевтическая сессия: тип, состояние, владелец, фаза хода	Характеристическая
12	Участник комнаты	Участие пользователя или гостя в комнате, согласия	Ассоциативная
13	Ход	Зафиксированное сообщение участника или ответ ИИ в диалоге	Характеристическая
14	Приглашение	Пригласительный токен для присоединения к комнате	Характеристическая
15	Тарифный план	Покупаемый план: уровень, длительность, стоимость	Стержневая
16	Подписка	Оплаченный период доступа пользователя к уровню BASIC	Характеристическая
17	Транзакция	Финансовая операция покупки, возврата или выдачи подписки	Характеристическая
18	Промокод	Код скидки с типом, значением и лимитами использования	Стержневая
19	Погашение промокода	Факт применения промокода пользователем в транзакции	Ассоциативная

На основе анализа предметной области и результатов проектирования серверной части была разработана концептуальная модель базы данных, представленная на рисунке 29.

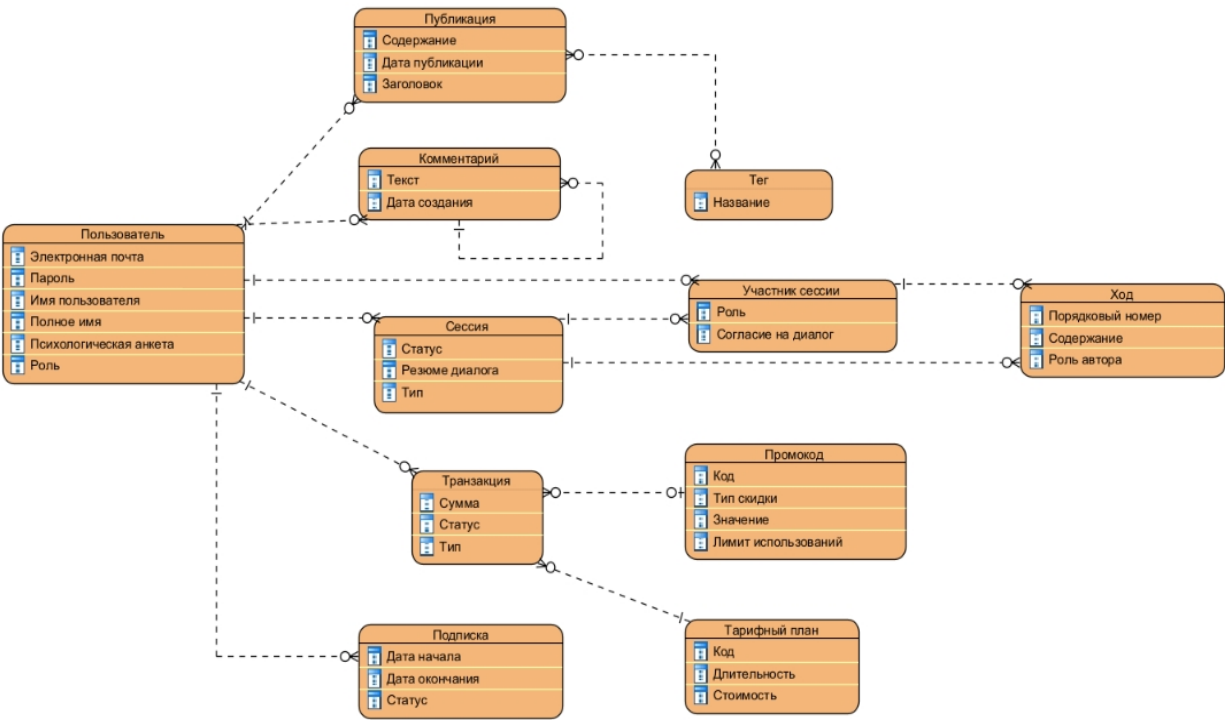


Рисунок 29 – Графическое представление концептуальной модели

Для разработанной концептуальной модели базы данных также было составлено табличное описание связей между сущностями, представленное в таблице 14.

Таблица 14 – Спецификации связей между сущностями для концептуальной модели базы данных

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность
1	Пользователь	Профиль	Пользователь имеет профиль / Профиль принадлежит пользователю	1:1	Обязательная
2	Пользователь	Дружба	Пользователь отправляет и получает заявки в друзья	M:N (через ассоциативную сущность)	Необязательная
3	Пользователь	Подписка на автора	Читатель подписывается на автора	M:N (через ассоциативную сущность)	Необязательная

Продолжение таблицы 14

№	Тип сущности 1	Тип сущности 2	Семантика связи	Мощность	Обязательность
4	Пользователь	Публикация	Автор создает публикации / Публикация принадлежит автору	1:M	Необязательная (публикации создают только авторы)
5	Публикация	Тег	Публикации присваиваются теги	M:N (через сущность «Тег публикации»)	Необязательная
6	Публикация	Комментарий	Публикация содержит комментарий	1:M	Необязательная
7	Комментарий	Комментарий	Комментарий может быть ответом на родительский комментарий	1:M (рекурсивная)	Необязательная
8	Пользователь	Оценка публикации	Пользователь оценивает публикации	M:N (через ассоциативную сущность)	Необязательная
9	Пользователь	Комната	Пользователь создает комнаты / Комната принадлежит владельцу	1:M	Необязательная
10	Комната	Участник комнаты	Комната содержит одного или двух участников	1:M	Обязательная
11	Комната	Ход	Комната содержит последовательность ходов диалога	1:M	Необязательная (до начала диалога ходов нет)
12	Комната	Приглашение	Для комнаты может выпускаться приглашение	1:M	Необязательная
13	Комната	Комната	Архивная комната может служить контекстом новой	1:M (рекурсивная)	Необязательная
14	Пользователь	Подписка	Пользователь оформляет подписки	1:M	Необязательная
15	Тарифный план	Транзакция	Транзакция оформляется по тарифному плану	1:M	Обязательная
16	Промокод	Погашение промокода	Промокод погашается пользователями в транзакциях	1:M	Необязательная

Заданные связи полноценно отражают взаимодействие объектов предметной области: социальный контур (дружба, подписки на авторов), контентный контур (публикации, теги, комментарии, оценки), терапевтический

контур (комнаты, участники, ходы) и финансовый контур (планы, подписки, транзакции, промокоды).

2.4.2 Разработка логической модели базы данных

Логическая модель строится на основе концептуальной путем отображения сущностей и связей в отношения реляционной модели данных. При построении логической модели были применены следующие проектные решения.

Все отношения приведены к третьей нормальной форме: каждый неключевой атрибут зависит только от первичного ключа. Перечислимые значения (роли, статусы пользователей и публикаций, типы и состояния комнат, уровни подписок, статусы транзакций, типы скидок) вынесены в отдельные справочные отношения, на которые основные отношения ссылаются внешними ключами.

Связи «многие-ко-многим» реализованы через ассоциативные отношения с составными первичными ключами: «Тег публикации» (post_tags), «Оценка публикации» (post_upvotes), «Подписка на автора» (author_follows), «Дружба» (friendships, ключ из пары идентификаторов с состоянием заявки), «Погашение промокода» (promo_redemptions).

Границы микросервисов отражены на логическом уровне разделением отношений на четыре схемы: user_schema, content_schema, chat_schema и billing_schema. Жесткие внешние ключи существуют только внутри схем; ссылки между схемами (автор публикации, владелец комнаты, пользователь подписки) реализованы как мягкие ссылки – атрибуты типа UUID без ограничения ссылочной целостности на уровне СУБД. Согласованность таких ссылок обеспечивается на уровне приложений через асинхронные события. Дополнительно сервис ИИ-сессий поддерживает локальную проекцию дружеских связей (friend_links), реплицируемую по событиям сервиса пользователей, что исключает синхронные обращения при проверке права создания парной комнаты.

В каждой схеме предусмотрено отношение исходящих событий (user_outbox_events, content_outbox_events, chat_outbox_events, billing_outbox_events) для реализации паттерна Transactional Outbox, спроектированного в разделе 2.1.

Логическая модель базы данных представлена на рисунке 30.

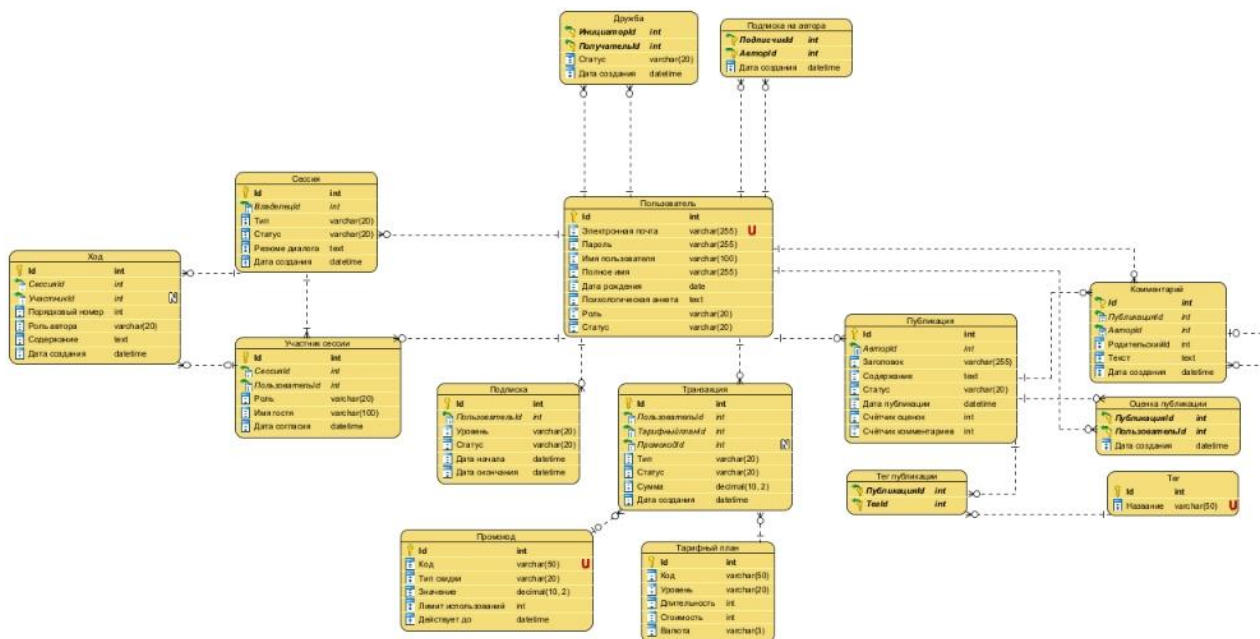


Рисунок 30 – Логическая модель базы данных

Таким образом, спроектированная логическая модель данных обеспечивает оптимальный баланс между строгой нормализацией внутри отдельных микросервисов и слабой связностью между ними.

2.4.3 Разработка физической модели базы данных

Физическая модель определяет реализацию логической модели средствами СУБД PostgreSQL 16 с учетом типов данных, индексов и ограничений целостности. Выбор конкретных физических параметров направлен на минимизацию задержек при выполнении транзакций и обеспечение отказоустойчивости хранилища при пиковых нагрузках.

В качестве первичных ключей основных отношений используются идентификаторы типа UUID, генерируемые приложением: это исключает конфликты идентификаторов между сервисами и упрощает мягкие межсхемные

ссылки. Справочные отношения используют целочисленные ключи. Все временные атрибуты хранятся в типе `TIMESTAMP WITH TIME ZONE`.

Для хранения слабоструктурированных данных применяется тип `JSONB`: содержимое публикаций (блочная структура статьи), а также полезная нагрузка событий во всех `outbox`-отношениях. Использование двоичного формата `JSONB` обеспечивает оптимальный баланс между гибкостью схемы и скоростью извлечения данных за счет предварительного синтаксического анализа документов при записи. Массив ключевых слов публикации хранится в типе `TEXT[]`.

Для обеспечения производительности контентной ленты и поиска предусмотрены специализированные индексы: по полю `search_vector` типа `TSVECTOR`, автоматически наполняемому из заголовка и содержимого публикации, построен `GIN`-индекс полнотекстового поиска; выборки ленты поддерживаны составными индексами по статусу и дате публикации; во всех `outbox`-отношениях создан частичный индекс по записям в состоянии ожидания, ускоряющий опрос планировщиком.

Целостность данных обеспечивается ограничениями уровня СУБД: уникальностью электронной почты и имени пользователя, уникальностью кода промокода и идемпотентного ключа транзакции, `CHECK`-ограничениями на положительность сумм и корректность периодов действия, а также составными первичными ключами ассоциативных отношений, исключающими дублирование связей. Для защиты от потерянных обновлений при конкурентном доступе основные отношения содержат атрибут `version`, используемый механизмом оптимистичной блокировки.

Счетчики просмотров, оценок и комментариев публикации хранятся денормализованно в отношении `posts` и обновляются транзакционно вместе с породившей операцией, что соответствует требованию производительности чтения из подраздела 1.4.2.1. Журнал модерации (`moderation_log`) фиксирует административные действия над контентом с сохранением состояния записи до

и после изменения в формате JSONB, обеспечивая возможность проведения сквозного аудита системы.

Физическая модель базы данных представлена на рисунке 31.

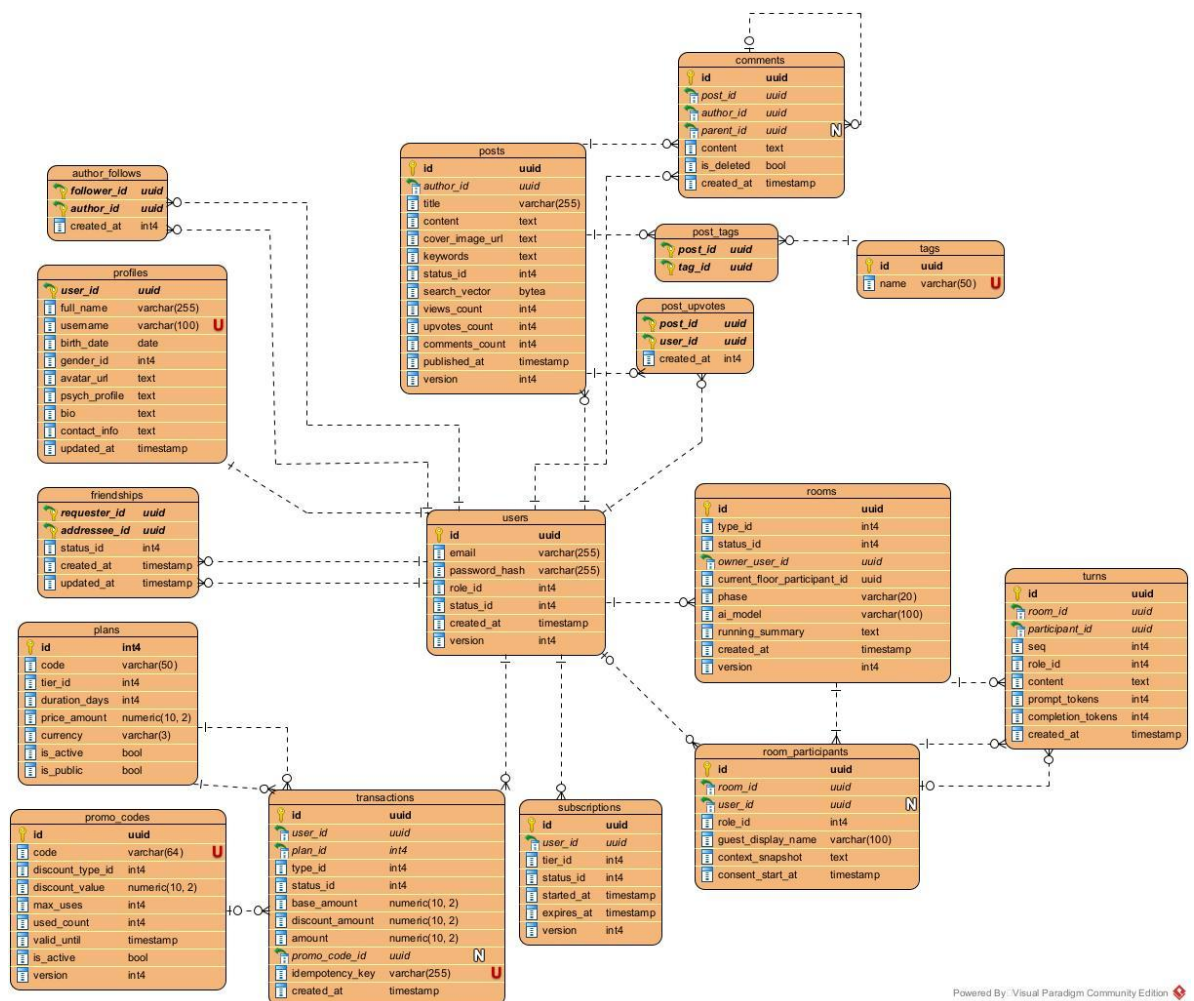


Рисунок 31 – Физическая модель базы данных

Подробное описание атрибутов сущностей базы данных с указанием типов данных, размерности, области допустимых значений и роли в схеме приведено в приложении Г.

Таким образом, в результате проектирования базы данных была построена единая концептуальная модель предметной области, отображенная в логическую модель с разделением на четыре схемы по границам микросервисов и реализованная физически средствами PostgreSQL с применением специализированных типов данных и индексов. Полученные модели являются основой для реализации информационной системы.

2.4.4 Развертывание базы данных в среде pgAdmin 4

Для создания базы данных, используемой в разработке, была выбрана PostgreSQL – реляционная СУБД, обеспечивающая надежность, согласованность и поддержку сложных связей между сущностями; она работает под управлением любой распространенной операционной системы. PostgreSQL гарантирует целостность данных и высокую производительность при одновременной работе нескольких пользователей [8].

Структура базы данных создается автоматически при запуске сервисов: каждый микросервис применяет собственный набор версионированных миграций к своей схеме, в результате чего в едином экземпляре PostgreSQL разворачиваются четыре изолированные схемы – `user_schema`, `content_schema`, `chat_schema` и `billing_schema`. Управление развернутой базой данных и визуальный контроль ее структуры выполняются в среде pgAdmin 4 – штатном графическом инструменте администрирования PostgreSQL [19].

Для наглядного представления развернутой структуры средствами pgAdmin 4 была сгенерирована диаграмма данных, представленная на рисунке 32.

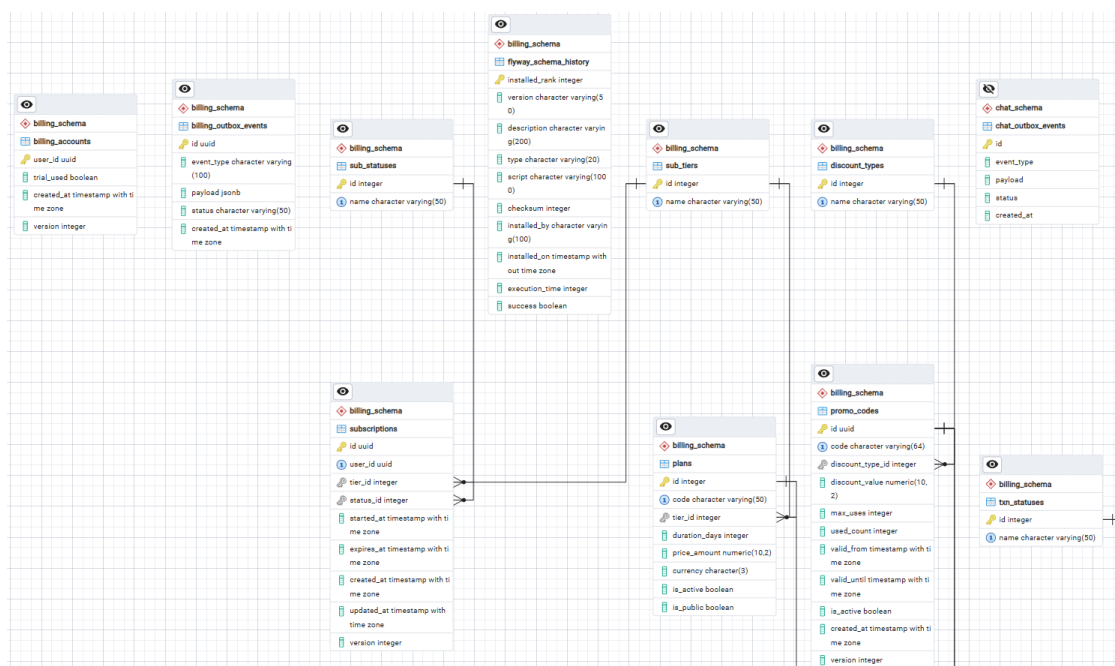


Рисунок 32 – Диаграмма данных, сгенерированная в pgAdmin 4

Полученные результаты используются в ходе наполнения базы данных и в процессе разработки приложения.

2.5 Проектирование взаимодействия со средствами контейнеризации и обмена сообщениями

Для обеспечения изолированного, воспроизводимого и масштабируемого исполнения микросервисов в разрабатываемой информационной системе применяется архитектура, основанная на контейнеризации и асинхронном обмене сообщениями. Такой подход позволяет независимо развертывать и масштабировать компоненты системы, а также гарантировать согласованность данных между сервисами при распределенных операциях.

В качестве основы для изоляции и управления средой исполнения применяется технология контейнеризации Docker [20]. Каждый микросервис системы – сервис-шлюз, сервис пользователей, сервис контента, сервис ИИ-сессий и сервис биллинга, – а также инфраструктурные компоненты – СУБД PostgreSQL, резидентное хранилище Redis и брокер сообщений RabbitMQ – упаковываются в отдельные контейнеры. Оркестрация контейнеров выполняется средствами Docker Compose, что обеспечивает воспроизводимость окружения, изоляцию процессов и упрощает перенос системы между серверами.

Для асинхронного взаимодействия между сервисами используется брокер сообщений RabbitMQ [21]. Брокер выступает в роли шины событий: при изменении состояния в одном сервисе соответствующее событие публикуется в брокер и доставляется заинтересованным сервисам, что исключает прямую синхронную зависимость между ними. Гарантированная доставка обеспечивается реализацией паттерна Transactional Outbox, при котором бизнес-изменение и факт публикации события сохраняются в одной транзакции базы данных, а согласованность данных при распределенных операциях достигается применением паттерна Saga.

Кроме того, два сервиса взаимодействуют с внешними системами через абстрагированные интерфейсы: сервис ИИ-сессий обращается к внешней

языковой модели, а сервис биллинга – к платежному провайдеру. Вынесение внешних систем за интерфейсы позволяет заменять поставщиков без изменения бизнес-логики сервисов.

Интеграция контейнеризации и асинхронного обмена сообщениями в архитектуре системы обеспечивает:

- изоляцию и воспроизводимость окружения каждого сервиса;
- возможность независимого масштабирования компонентов системы;
- гарантированную доставку событий между сервисами;
- слабую связанность сервисов и устойчивость к временной недоступности отдельных компонентов.

Для наглядной иллюстрации работы проектируемой архитектуры на рисунке 33 представлена диаграмма последовательности процесса оформления подписки пользователем.

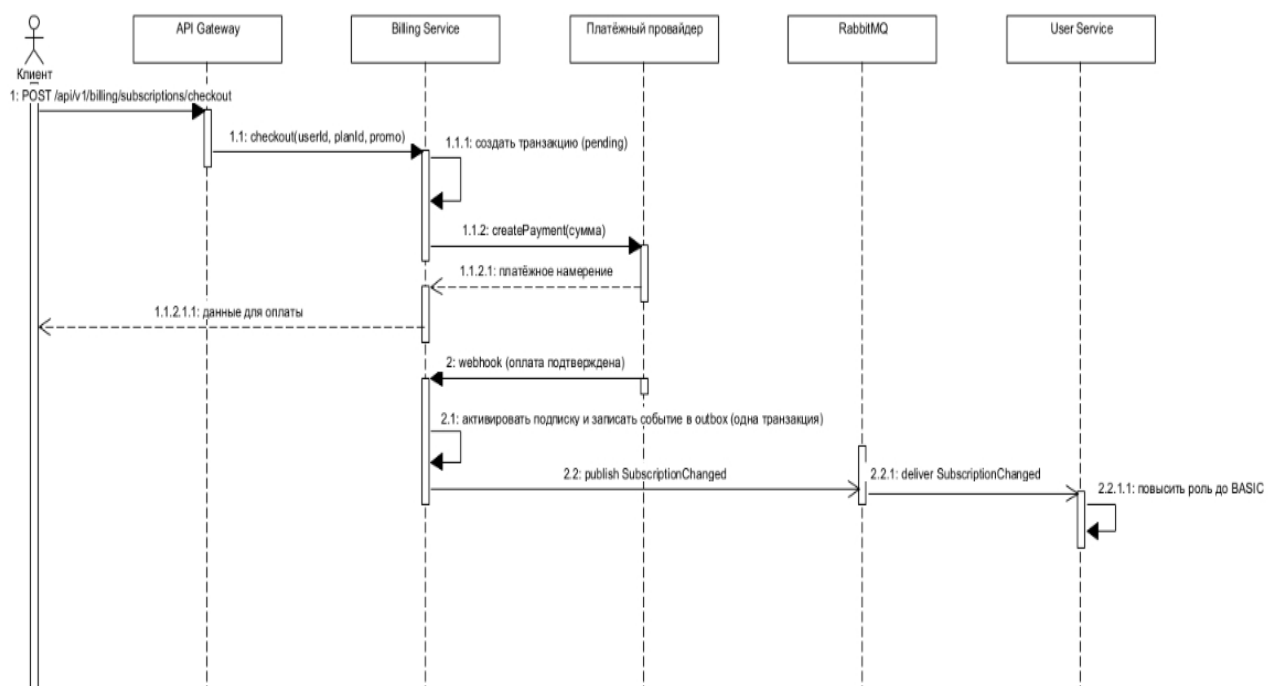


Рисунок 33 – Диаграмма последовательности процесса оформления подписки

Данная диаграмма последовательности иллюстрирует взаимодействие компонентов системы при покупке подписки. Запрос клиента проходит через сервис-шлюз и поступает в сервис биллинга, который создает транзакцию в состоянии ожидания и формирует платежное намерение у внешнего платежного

провайдера. После подтверждения оплаты провайдер уведомляет систему вебхуком, сервис биллинга активирует подписку и в той же транзакции базы данных формирует событие об изменении подписки. Событие публикуется в брокер сообщений RabbitMQ и доставляется сервису пользователей, который повышает роль пользователя. Такой порядок реализует пессимистичную сагу выдачи роли: права пользователя повышаются только после подтвержденной оплаты, а атомарная запись события гарантирует его доставку даже при кратковременном сбое. Полученная архитектура взаимодействия является основой для реализации информационной системы.

3 Реализация информационной системы оказания психологической помощи

3.1 Реализация взаимодействия с базой данных

3.1.1 Реализация миграций схем и генерируемых столбцов

Управление структурой базы данных в системе реализовано через версионированные миграции с использованием инструмента Flyway [13]. Каждый микросервис содержит собственный набор SQL-скриптов миграций, применяемых автоматически при запуске приложения: Flyway сравнивает версию схемы в служебной таблице с набором скриптов в каталоге db/migration и последовательно выполняет недостающие. Такой подход гарантирует, что структура базы данных в любом окружении – от локальной разработки до продуктивного сервера – однозначно соответствует исходному коду приложения. Фрагмент конфигурации Flyway сервиса пользователей приведен в листинге 1.

Полный скрипт создания объектов базы данных приведён в приложении Д.

Листинг 1 – Конфигурация Flyway в файле application.yml сервиса пользователей

```
flyway:
  enabled: true
  baseline-on-migrate: true
  schemas: user_schema
  default-schema: user_schema
  create-schemas: true
```

Каждый сервис мигрирует исключительно собственную схему, что закрепляет спроектированную в подразделе 2.4.2 изоляцию данных на уровне инструмента развертывания. Фрагмент первой миграции сервиса пользователей, создающей таблицы учетных записей и профилей, приведен в листинге 2.

Листинг 2 – Фрагмент миграции V1 создания таблиц схемы user_schema

```
CREATE SCHEMA IF NOT EXISTS user_schema;

CREATE TABLE user_schema.users (
  id UUID PRIMARY KEY,
```

```

email VARCHAR(255) UNIQUE NOT NULL,
password_hash VARCHAR(255) NOT NULL,
role_id INT REFERENCES user_schema.user_roles(id),
status_id INT REFERENCES user_schema.user_statuses(id),
created_at TIMESTAMP WITH TIME ZONE NOT NULL,
version INT DEFAULT 0 NOT NULL
);

CREATE TABLE user_schema.profiles (
    user_id UUID PRIMARY KEY REFERENCES user_schema.users(id),
    full_name VARCHAR(255) NOT NULL,
    birth_date DATE,
    gender_id INT REFERENCES user_schema.genders(id),
    username VARCHAR(100) UNIQUE NOT NULL,
    avatar_url TEXT,
    psych_profile JSONB,
    updated_at TIMESTAMP WITH TIME ZONE NOT NULL
);

```

Приведенный фрагмент демонстрирует реализацию ключевого решения по безопасности: учетные данные авторизации (таблица `users`) физически отделены от публичного профиля (таблица `profiles`), а психологическая анкета хранится в столбце `psych_profile` типа `JSONB` и не участвует ни в каких публичных выборках.

Вместо триггеров для автоматического сопровождения производных данных в системе применяются генерируемые столбцы PostgreSQL [8]. Поисковый вектор публикации объявлен с модификатором `GENERATED ALWAYS AS` и вычисляется системой управления базами данных автоматически при каждом изменении заголовка или содержания статьи, что исключает рассинхронизацию поискового индекса с данными. Скрипт добавления генерируемого столбца и индекса полнотекстового поиска приведен в листинге 3.

Листинг 3 – Создание генерируемого столбца поискового вектора и GIN-индекса

```

ALTER TABLE content_schema.posts ADD COLUMN search_vector tsvector
    GENERATED ALWAYS AS (
        to_tsvector('russian',
            coalesce(title, '') || ' ' || coalesce(content::text,
''))
    ) STORED;

CREATE INDEX idx_posts_search_vector

```

```
ON content_schema.posts USING GIN (search_vector);
```

Таким образом, версионированные миграции обеспечивают воспроизводимость структуры базы данных во всех окружениях, а генерируемый столбец с GIN-индексом автоматизирует сопровождение полнотекстового поискового индекса без написания триггерных функций.

3.1.2 Реализация серверных запросов к базе данных

Для обращения к базе данных в микросервисах используются технологии Spring Data JPA поверх Hibernate ORM [14]. Репозитории объявляются интерфейсами, а реализация типовых операций выборки и сохранения генерируется автоматически. Использование данного стека дает следующие преимущества:

- защита от SQL-инъекций за счет параметризованных запросов;
- автоматический маппинг результатов запросов в Java-объекты;
- декларативное управление транзакциями;
- встроенная поддержка оптимистичной блокировки и постраничной выборки.

Отображение таблиц на объекты выполняется JPA-сущностями. Фрагмент сущности Post, демонстрирующий работу со специализированными типами PostgreSQL и механизмом оптимистичной блокировки, приведен в листинге 4.

Листинг 4 – Фрагмент JPA-сущности Post

```
@Column(columnDefinition = "jsonb")
@JdbcTypeCode(SqlTypes.JSON)
private String content;

@Column(columnDefinition = "text[]")
@JdbcTypeCode(SqlTypes.ARRAY)
private List<String> keywords;

@Version
@Column(nullable = false)
private Integer version;
```

Аннотация `@JdbcTypeCode` связывает поля сущности со столбцами типов JSONB и TEXT[], а аннотация `@Version` включает механизм оптимистичной блокировки: при попытке сохранить устаревшую версию записи Hibernate

выбрасывает исключение, которое транслируется клиенту в ответ с кодом 409, что защищает счетчики и статусы от потерянных обновлений при конкурентном доступе.

Для сложных выборок, не выразимых средствами JPA, применяются нативные SQL-запросы. Наиболее показательным примером является запрос полнотекстового поиска публикаций, использующий поисковый вектор из подраздела 3.1.1, ранжирование результатов функцией `ts_rank_cd` и совпадение по массиву ключевых слов. Фрагмент запроса приведен в листинге 5.

Листинг 5 – Фрагмент нативного запроса полнотекстового поиска публикаций

```
@Query(value = ""
    SELECT p.id AS postId,
           (ts_rank_cd(p.search_vector, query.q)
            + CASE WHEN :keywordCount > 0
                   AND p.keywords && CAST(:keywordsArr AS text[])
                   THEN 0.3 ELSE 0 END) AS score
    FROM content_schema.posts p,
         websearch_to_tsquery('russian', :q) AS query(q)
    WHERE p.status_id = 2
          AND ( p.search_vector @@ query.q
                OR (:keywordCount > 0
                    AND p.keywords && CAST(:keywordsArr AS text[])) )
    ORDER BY score DESC, p.published_at DESC, p.id DESC
    LIMIT :limit
    "", nativeQuery = true)
List<PostSearchHit> searchPostIds(@Param("q") String q, ...);
```

Функция `websearch_to_tsquery` преобразует пользовательский запрос в поисковое выражение с поддержкой естественного синтаксиса, оператор `@@` выполняет сопоставление с генерируемым вектором по GIN-индексу, а совпадение по массиву ключевых слов повышает итоговый ранг результата. Такое решение обеспечивает релевантный поиск на русском языке штатными средствами PostgreSQL без подключения внешней поисковой системы.

3.1.3 Наполнение базы данных начальными данными

Наполнение базы данных начальными данными реализовано через те же версионированные миграции, что и создание структуры: справочные значения добавляются командами `INSERT` в отдельных скриптах. Такой подход гарантирует наличие справочников в любом окружении сразу после первого

запуска сервиса и фиксирует их состав в системе контроля версий. Фрагмент скрипта наполнения справочников сервиса пользователей приведен в листинге 6.

Листинг 6 – Наполнение справочников схемы user_schema

```
INSERT INTO user_schema.user_roles (id, name) VALUES
(1, 'GUEST'),
(2, 'FREE'),
(3, 'BASIC'),
(4, 'AUTHOR');

INSERT INTO user_schema.user_statuses (id, name) VALUES
(1, 'ACTIVE'),
(2, 'BANNED'),
(3, 'DELETED');
```

Эволюция справочников также выполняется миграциями: роль ADMIN была добавлена отдельным скриптом версии V4 командой INSERT INTO user_schema.user_roles (id, name) VALUES (5, 'ADMIN'), что иллюстрирует управляемое изменение данных без ручного вмешательства в базу.

Помимо справочников, миграциями наполняется витрина тарифных планов сервиса биллинга, включающая публичные планы подписки и скрытый план пробного периода с нулевой стоимостью. Скрипт наполнения приведен в листинге 7.

Листинг 7 – Наполнение справочников и тарифных планов схемы billing_schema

```
INSERT INTO sub_tiers (id, name) VALUES (1, 'BASIC'), (2, 'PREMIUM');
INSERT INTO txn_statuses (id, name) VALUES
(1, 'PENDING'), (2, 'SUCCESS'), (3, 'FAILED'), (4, 'REFUNDED');

INSERT INTO plans (id, code, tier_id, duration_days,
                  price_amount, currency, is_active, is_public)
VALUES
(1, 'MONTH_1', 1, 30, 499.00, 'RUB', TRUE, TRUE),
(2, 'MONTH_3', 1, 90, 1299.00, 'RUB', TRUE, TRUE),
(3, 'TRIAL_30', 1, 30, 0.00, 'RUB', TRUE, FALSE);
```

Таким образом, реализованный механизм взаимодействия с базой данных объединяет версионированные миграции структуры и данных, объектно-реляционное отображение со специализированными типами PostgreSQL и нативные запросы для полнотекстового поиска, обеспечивая воспроизводимость, целостность и производительность слоя хранения данных.

3.2 Реализация серверной части

3.2.1 Реализация аутентификации и авторизации

Аутентификация в системе реализована по схеме без сохранения состояния (stateless) на основе токенов JSON Web Token. Выпуск токенов сосредоточен в сервисе пользователей: при успешной проверке учетных данных формируется подписанный токен, содержащий идентификатор пользователя и его роль. Каждый микросервис проверяет токен самостоятельно, что устраняет необходимость в общей серверной сессии и соответствует выбранной микросервисной архитектуре. Метод генерации токена приведен в листинге 8.

Полный исходный код ключевых модулей серверной части приведён в приложении Б.

Листинг 8 – Генерация JWT-токена в сервисе пользователей

```
public String generateToken(UUID userId, String email, Integer
roleId) {
    Map<String, Object> claims = new HashMap<>();
    claims.put("userId", userId.toString());
    claims.put("role", UserRole.fromId(roleId).getName());
    return createToken(claims, email);
}

private String createToken(Map<String, Object> claims, String
subject) {
    return Jwts.builder()
        .setClaims(claims)
        .setSubject(subject)
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis()
            + (expirationHours * 3600000)))
        .signWith(getSigninKey(), SignatureAlgorithm.HS256)
        .compact();
}
```

Токен подписывается алгоритмом HS256 на общем секретном ключе и включает идентификатор и роль пользователя в составе полезной нагрузки, что позволяет любому сервису определить права обращающегося без запроса к сервису пользователей. Пароли при этом никогда не хранятся в открытом виде: при регистрации они хешируются алгоритмом BCrypt.

Проверка токена на стороне каждого сервиса выполняется фильтром, перехватывающим входящие запросы, извлекающим токен из заголовка `Authorization` и устанавливающим контекст безопасности. Особенностью сервиса ИИ-сессий является поддержка двух типов субъектов: зарегистрированного пользователя и гостя, присоединившегося по пригласительной ссылке. Гостевой токен ограничен областью действия одной комнаты, что отражено в его полезной нагрузке. Фрагмент фильтра аутентификации сервиса ИИ-сессий приведен в листинге 9.

Листинг 9 – Фрагмент фильтра аутентификации с поддержкой гостевого доступа

```
final String token = authHeader.substring(7);

if (jwtService.isTokenValid(token)) {
    Claims claims = jwtService.extractAllClaims(token);

    String scope = claims.get("scope", String.class);
    if ("guest".equals(scope)) {
        UUID participantId = UUID.fromString(claims.getSubject());
        UUID roomId = UUID.fromString(
            claims.getAudience().replace("room:", ""));
        GuestPrincipal guestPrincipal =
            new GuestPrincipal(participantId, roomId);
        var authToken = new UsernamePasswordAuthenticationToken(
            guestPrincipal, null,
            Collections.singleton(
                new
SimpleGrantedAuthority("ROLE_GUEST"))));

SecurityContextHolder.getContext().setAuthentication(authToken);
    } else {
        String email = claims.getSubject();
        UUID userId = UUID.fromString(claims.get("userId",
String.class));
        // ... установка контекста для зарегистрированного
пользователя
    }
}
```

Разделение субъектов на зарегистрированного пользователя и гостя с областью действия токена, ограниченной одной комнатой, позволяет приглашать второго участника парной сессии по ссылке без регистрации, не

открывая ему доступ к остальным функциям системы. Так реализуется требование разграничения доступа из подраздела 1.4.2.1.

3.2.2 Реализация очередности ходов и взаимодействия с ИИ

Центральным механизмом сервиса ИИ-сессий является управление очередностью ходов в парной комнате и взаимодействие с внешней языковой моделью. После того как владеющий ходом участник завершает мысль, комната переводится в фазу обработки, диалог передается языковой модели, а после получения ответа право голоса передается второму участнику. Эта последовательность реализована в сервисе оркестрации ходов, фрагмент которого приведен в листинге 10.

Листинг 10 – Фрагмент сервиса оркестрации обращения к языковой модели

```
public Turn executeAiStep(UUID roomId) {
    Room updatedRoom =
roomRepository.findById(roomId).orElseThrow();
    List<RoomParticipant> participants =
        participantRepository.findByRoomId(roomId);
    List<Turn> allTurns = loadHistory(roomId);

    LlmRequest llmRequest =
        conversationAssembler.assemble(updatedRoom,
participants, allTurns);

    LlmResponse llmResponse;
    try {
        llmResponse = llmClient.complete(llmRequest);
    } catch (LlmUnavailableException e) {
        turnPersistenceService.handleAiFailure(roomId);
        throw e;
    }

    return turnPersistenceService.persistAssistantTurn(roomId,
        llmResponse.content(),
        llmResponse.promptTokens(),
        llmResponse.completionTokens());
}
```

Обращение к модели абстрагировано интерфейсом `LlmClient`, что позволяет заменять поставщика без изменения логики оркестрации. Сборщик контекста `ConversationAssembler` формирует запрос из системного промпта, психологических анкет участников и истории диалога. При недоступности

модели вызывается обработчик сбоя, возвращающий комнату в исходную фазу, что не приводит к потере уже отправленных сообщений.

Передача права голоса второму участнику выполняется при сохранении ответа модели в одной транзакции с фиксацией хода. Фрагмент логики смены владельца хода приведен в листинге 11.

Листинг 11 – Фрагмент логики передачи права голоса

```
// сохранение хода ассистента с очередным порядковым номером
turn = turnRepository.save(turn);

// передача права голоса второму участнику в парной комнате
if (room.getTypeId() == 1) { // 1 = PAIRED
    UUID currentHolderId = room.getCurrentFloorParticipantId();
    UUID otherHolderId =
participantRepository.findByRoomId(roomId).stream()
    .map(RoomParticipant::getId)
    .filter(id -> !id.equals(currentHolderId))
    .findFirst()
    .orElse(currentHolderId);
    room.setCurrentFloorParticipantId(otherHolderId);
}

room.setPhase("A_COMPOSING");
roomRepository.save(room);
```

Каждый ход получает монотонно возрастающий порядковый номер, по которому определяется очередность и обеспечивается идемпотентность операции, а проверка фазы комнаты перед отправкой исключает одновременную обработку двух ходов. Совместно эти механизмы гарантируют строгую поочередность диалога даже при нестабильном сетевом соединении участников.

3.2.3 Реализация гарантированной доставки событий

Согласованность данных между микросервисами обеспечивается асинхронным обменом событиями через брокер сообщений RabbitMQ с применением паттерна Transactional Outbox. Наиболее показательным примером является сценарий повышения роли пользователя после оплаты подписки: сервис биллинга не обращается к сервису пользователей напрямую, а публикует событие, которое тот обрабатывает самостоятельно.

Ключевой принцип паттерна состоит в том, что бизнес-изменение и факт публикации события сохраняются в одной транзакции базы данных. При активации подписки в той же транзакции, в которой обновляется запись подписки, формируется запись исходящего события. Фрагмент сервиса активации приведен в листинге 12.

Листинг 12 – Запись события в таблицу исходящих сообщений при активации подписки

```
@Transactional
public Subscription activateForDays(UUID userId, Transaction
transaction,
                                int durationDays) {
    // ... обновление записи подписки и продление срока действия

    // запись события SUBSCRIPTION_CHANGED в outbox в той же
транзакции
    SubscriptionChangedEvent payload = new
SubscriptionChangedEvent(
        userId, "BASIC",
        newExpiresAt.toOffsetDateTime(),
        OffsetDateTime.now());
    BillingOutboxEvent outboxEvent =

outboxEventFactory.create(EventType.SUBSCRIPTION_CHANGED,
payload);
    billingOutboxEventRepository.save(outboxEvent);

    return sub;
}
```

Поскольку и подписка, и событие сохраняются атомарно, отказ сервиса между этими операциями невозможен: либо фиксируются оба изменения, либо ни одного. Накопленные события публикуются в брокер отдельным фоновым обработчиком по расписанию, после чего помечаются как обработанные. Фрагмент обработчика приведен в листинге 13.

Листинг 13 – Фоновая публикация накопленных событий в брокер

```
@Scheduled(fixedDelay = 1000)
@Transactional
public void publishPendingEvents() {
    List<BillingOutboxEvent> pending =
        billingOutboxEventRepository
            .findTop100ByStatusOrderByCreatedAtAsc("PENDING");
```

```

        for (BillingOutboxEvent event : pending) {
            try {
                String routingKey =
getRoutingKey(event.getEventType());
                rabbitTemplate.convertAndSend(
                    RabbitMQConfig.EXCHANGE_NAME, routingKey,
                    event.getPayload());
                event.setStatus("PROCESSED");
                billingOutboxEventRepository.save(event);
            } catch (Exception ex) {
                // событие остается в статусе PENDING и будет
переотправлено
            }
        }
    }
}

```

На принимающей стороне сервис пользователей подписан на очередь событий биллинга: при поступлении события об изменении подписки соответствующему пользователю повышается роль. Подписка обработчика приведена в листинге 14.

Листинг 14 – Потребитель событий биллинга в сервисе пользователей

```

@RabbitListener(queues = RabbitMQConfig.BILLING_EVENTS_QUEUE)
public void onBillingEvent(@Payload String json,
    @Header("amqp_receivedRoutingKey") String routingKey) {
    try {
        if ("billing.subscription-changed".equals(routingKey)) {
            // разбор события и повышение роли пользователя до
BASIC
        }
    } catch (Exception e) {
        // ошибка обработки приводит к повторной доставке события
    }
}

```

Таким образом реализуется пессимистичная сага выдачи роли: повышение прав происходит только после подтвержденной оплаты, а атомарная запись события в одной транзакции с бизнес-изменением в сочетании с переотправкой необработанных событий гарантирует, что изменение подписки рано или поздно будет доставлено получателю даже при кратковременном сбое брокера или сервиса. Так выполняется требование согласованности данных при распределенных транзакциях из подраздела 1.4.2.1.

3.3 Реализация клиентской части

3.3.1 Реализация взаимодействия в реальном времени

Терапевтические сессии требуют двусторонней связи в реальном времени: участники видят онлайн-статус друг друга, печатаемые черновики и смену хода без перезагрузки страницы. Для этого на клиенте реализован WebSocket-клиент поверх протокола STOMP с транспортом SockJS. Подключение устанавливается единожды и переиспользуется всеми сессиями, аутентификация выполняется передачей JWT-токена в заголовке кадра подключения, а при разрыве соединения восстанавливается автоматически. Фрагмент инициализации клиента приведен в листинге 15.

Полный исходный код ключевых модулей клиентской части приведён в приложении В.

Листинг 15 – Инициализация STOMP-клиента с JWT-аутентификацией

```
this.client = new Client({
  websocketFactory: () => new SockJS(`${getWsBaseUrl()}/ws`),
  connectHeaders: {
    Authorization: `Bearer ${token}`,
  },
  reconnectDelay: 3000,
  onConnect: () => {
    if (!resolved) {
      resolved = true
      resolve()
    }
    this.onConnectCallbacks.forEach((cb) => cb())
  },
  onStompError: (frame) => {
    console.error('Broker reported error: ' +
      frame.headers['message'])
  },
})
```

Параметр `reconnectDelay` включает автоматическое переподключение при обрыве связи, а зарегистрированные обратные вызовы повторно оформляют подписки на каналы комнаты после восстановления соединения, что обеспечивает устойчивость сессии к нестабильной сети.

Взаимодействие с комнатой организовано через публикацию команд в адреса приложения и подписку на широковещательные каналы. Клиентские команды – управление черновиками, завершение мысли, согласие на начало и завершение диалога – реализованы тонкими обертками над публикацией в

STOMP, что соответствует перечню WebSocket-команд из таблицы 11. Фрагмент команд комнаты приведен в листинге 16.

Листинг 16 – Команды управления терапевтической сессией

```
const upsertDraft = (bubbleId: string, text: string) =>
  stompClient.publish(`/app/rooms/${roomId}/draft/upsert`, {
    bubbleId, text })

const finishThought = (turnSeq: number) =>
  stompClient.publish(`/app/rooms/${roomId}/finish`, { turnSeq })

const proposeEnd = () =>
  stompClient.publish(`/app/rooms/${roomId}/end/propose`, {})

const agreeEnd = () =>
  stompClient.publish(`/app/rooms/${roomId}/end/agree`, {})
```

Прием обновлений организован подписками на каналы комнаты: широковещательный канал событий комнаты, персональный канал присутствия участников и персональный канал ошибок. При подписке учтен порядок: канал присутствия оформляется до канала комнаты, поскольку подписка на комнату инициирует рассылку снимка присутствия. Так клиент в реальном времени отображает онлайн-статус собеседника, смену владельца хода и ответы ИИ-модератора. Внешний вид реализованного окна парной терапевтической сессии показан на рисунке 34.

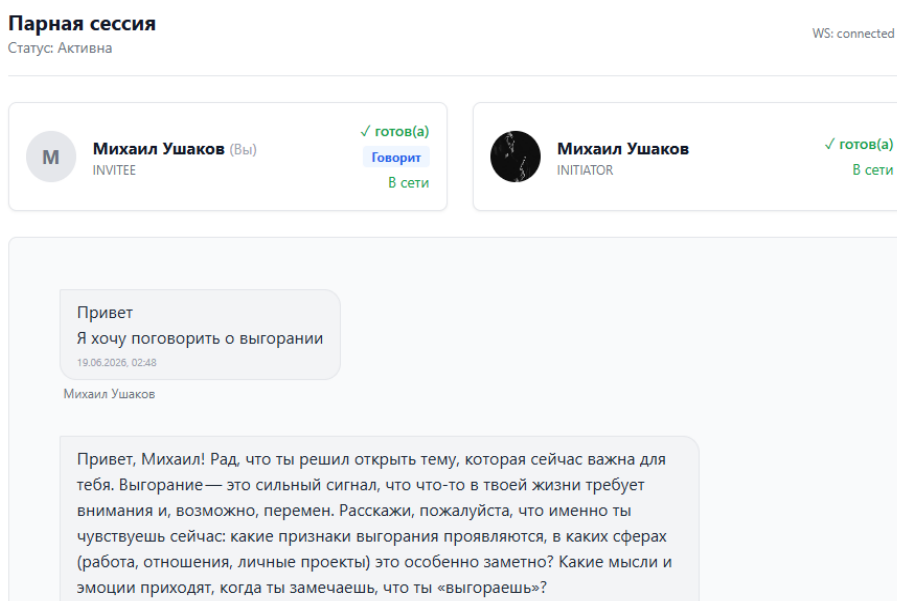


Рисунок 34 – Окно парной ИИ-терапевтической сессии

На изображении видно область переписки с сообщениями участников и ответами ИИ-модератора, индикаторы присутствия собеседника и индикатор текущего владельца хода, а поле ввода сообщения доступно только тому участнику, которому принадлежит ход. Такой интерфейс наглядно реализует спроектированный в подразделе 2.2 сценарий поочередного диалога: пользователь в реальном времени видит, чья сейчас очередь, и не может прервать собеседника, что обеспечивает упорядоченное и спокойное взаимодействие.

3.3.2 Реализация управления серверным состоянием

Управление серверным состоянием на клиенте реализовано с использованием библиотеки TanStack Query [15]. Библиотека берет на себя кэширование ответов сервера, отслеживание актуальности данных, повторные запросы при ошибках и согласованное обновление кэша после изменяющих операций, что избавляет от ручного управления состоянием загрузки. Применение данной глобальной конфигурации позволяет оптимизировать сетевой трафик за счет отключения избыточных автоматических запросов при смене фокуса окна браузера пользователем. Глобальная конфигурация клиента запросов приведена в листинге 17.

Листинг 17 – Конфигурация клиента запросов TanStack Query

```
export const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      retry: 1,
      refetchOnWindowFocus: false,
    },
  },
})
```

Операции чтения объявляются запросами с уникальным ключом кэша. Контентная лента, обладающая потенциально неограниченным объемом, реализована запросом с подгрузкой страниц по курсору: ключ запроса включает режим сортировки и набор выбранных тегов, а следующая страница запрашивается по курсору из предыдущего ответа. Такое решение обеспечивает

плавное динамическое обновление графического интерфейса без необходимости полной перезагрузки страницы. Соответствующий хук приведен в листинге 18.

Листинг 18 – Хук постраничной загрузки контентной ленты

```
export function useFeed(params: { sort: SortMode; tags: string[]
}) {
  const tagsKey = params.tags.slice().sort().join(',')

  return useInfiniteQuery({
    queryKey: ['feed', params.sort, tagsKey],
    queryFn: ({ pageParam }) =>
      getFeed({ sort: params.sort, cursor: pageParam, tags:
params.tags }),
    initialPageParam: undefined as string | undefined,
    getNextPageParam: (lastPage) => lastPage.nextCursor ??
undefined,
  })
}
```

Изменяющие операции объявляются мутациями, которые после успешного выполнения инвалидируют затронутые ключи кэша, вызывая автоматическое обновление зависящих от них данных. Например, при добавлении комментария инвалидируются кэш самой публикации, список ее комментариев и, для вложенного ответа, ветка родительского комментария. Декларативное описание связей между сущностями в локальном хранилище гарантирует отсутствие устаревших данных на экране и исключает возникновение гонок за превосходство ответов. Хук создания комментария приведен в листинге 19.

Листинг 19 – Хук добавления комментария с инвалидацией кэша

```
export function useCreateComment(postId: string) {
  const queryClient = useQueryClient()

  return useMutation<Comment, unknown, CreateCommentBody>({
    mutationFn: (body) => createComment({ postId, body }),
    onSuccess: (newComment) => {
      queryClient.invalidateQueries({ queryKey: ['post', postId],
exact: true })
      queryClient.invalidateQueries({ queryKey: ['post', postId,
'comments'] })
      if (newComment.parentId) {
        queryClient.invalidateQueries({
          queryKey: ['comment', newComment.parentId, 'replies'],
        })
      }
    }
  })
}
```

```

    }
  },
})
}

```

Такой подход обеспечивает согласованность отображаемых данных с сервером: после любой изменяющей операции связанные представления обновляются автоматически, а пользователь видит индикаторы загрузки на время выполнения запросов. Внедрение данного слоя абстракции позволяет полностью изолировать логику синхронизации данных от компонентов представления, существенно повышая масштабируемость и тестируемость клиентского приложения. Результат работы механизма управления серверным состоянием на примере контентной ленты показан на рисунке 35.

Добро пожаловать в Клавдиум

Автор: [mixanewstyle](#) [Подписаться](#) · 13.06.2026

Философское

Я невероятно рад, что этот проект был рожден на свет. Он очень многое для меня значит, здесь всегда будет находиться частичка моей души.

Надеюсь, что это место станет безопасным перевалом среди сложных горных путей ответственности и жизненных решений для многих.

Мне хочется построить комьюнити, объединяющее активных, любознательных и очень живущих людей.

Мечтать, любить и творить.

👁 19 💬 0 ❤ 4

Комментарии

Напишите комментарий...

0/2000

[Отправить](#)

В

Владос · 15.06.2026

[comment deleted]

Рисунок 35 – Область просмотра публикации

На изображении видно список карточек публикаций с обложкой, заголовком, именем автора и счетчиками оценок и комментариев; при прокрутке к концу списка автоматически подгружается следующая страница ленты. Благодаря кэшированию данных повторное открытие ранее загруженных публикаций происходит мгновенно, а постраничная подгрузка по курсору

обеспечивает плавную работу с лентой неограниченного объема, что соответствует требованиям к интерфейсу из подраздела 1.4.2.2.

3.3.3 Реализация разграничения доступа на стороне клиента

Разграничение доступа на клиенте реализовано на двух уровнях: защита маршрутов от неаутентифицированных пользователей и ролевое отображение элементов навигации. Защита маршрутов выполнена компонентом-оберткой, который проверяет наличие и срок действия JWT-токена и перенаправляет на форму входа, если токен отсутствует или истек. Компонент приведен в листинге 20.

Листинг 20 – Компонент защиты маршрутов

```
export function ProtectedRoute({ children }: ProtectedRouteProps)
{
  const token = useAuthStore((state) => state.token)
  const clearAuth = useAuthStore((state) => state.clearAuth)

  const isExpired = token ? isTokenExpired(token) : true

  useEffect(() => {
    if (isExpired) {
      clearAuth()
    }
  }, [isExpired, clearAuth])

  if (isExpired) {
    return <Navigate to="/login" replace />
  }

  return <>{children}</>
}
```

Состав отображаемых пунктов навигации зависит от роли пользователя, извлекаемой из полезной нагрузки токена: раздел студии автора доступен пользователям с ролью AUTHOR или ADMIN, а раздел администрирования – только пользователям с ролью ADMIN. Фрагмент условного отображения пунктов меню приведен в листинге 21.

Листинг 21 – Ролевое отображение пунктов навигации

```
{user && (user.role === 'AUTHOR' || user.role === 'ADMIN')} && (
  <NavLink to="/authoring">Мои публикации</NavLink>
)
```

```
{user && user.role === 'ADMIN' && (
  <NavLink to="/admin">Администрирование</NavLink>
)}
```

Необходимо подчеркнуть, что разграничение доступа на стороне клиента носит исключительно интерфейсный характер и служит для удобства пользователя, скрывая недоступные ему разделы. Окончательная проверка прав доступа во всех случаях выполняется на стороне сервера, как описано в подразделе 3.2.1, что исключает несанкционированный доступ при попытке обращения к защищенным эндпоинтам в обход интерфейса. Доступный только пользователю с ролью администратора раздел администрирования показан на рисунке 36.

Панель администратора

Пользователи Теги **Биллинг**

Промокоды

Код	Тип скидки	Значение	Макс. исп.	Начало	Окончание
<input type="text"/>	Процент (%)	<input type="text" value="10"/>	<input type="text" value="100"/>	<input type="text" value="дд.мм.гггг --:--"/>	<input type="text" value="дд.мм.гггг --:--"/>

[Создать промокод](#)

Код	Скидка	Использования	Действует	Активен
WELCOME	100 %	0 / 100	С: 16.06.2026, 21:40 По: 16.09.2026, 21:40	ДА

Рисунок 36 – Область раздела администрирования

На изображении видно панель администрирования с вкладками управления пользователями, тегами и биллингом; данный раздел отображается в навигации и доступен только пользователю с ролью администратора, тогда как для остальных ролей соответствующий пункт меню скрыт. Это демонстрирует работу ролевого разграничения доступа на стороне клиента, дополняющего серверную проверку прав.

Таким образом, в результате реализации клиентской части было создано одностраничное приложение, обеспечивающее взаимодействие в реальном времени в терапевтических сессиях, согласованное с сервером управление

состоянием и ролевое разграничение доступа. Совместно с реализованной серверной частью и слоем хранения данных это составляет полнофункциональную информационную систему оказания психологической помощи, удовлетворяющую требованиям, сформулированным в первой главе.

3.3.4 Реализация сборки компонентов

Сборка клиентской части выполняется с помощью современного сборщика Vite. Он поднимает dev-сервер с возможностью моментальной горячей перезагрузки модулей, автоматически управляет зависимостями и разбивает итоговый бандл на части для ускорения первоначальной загрузки приложения. Применение языка TypeScript обеспечивает статическую типизацию на этапе сборки, а компонентная организация по функциональным модулям позволяет масштабировать интерфейс без необходимости модифицировать остальные части программы. На рисунке 37 показана диаграмма компонентов клиентской части приложения.

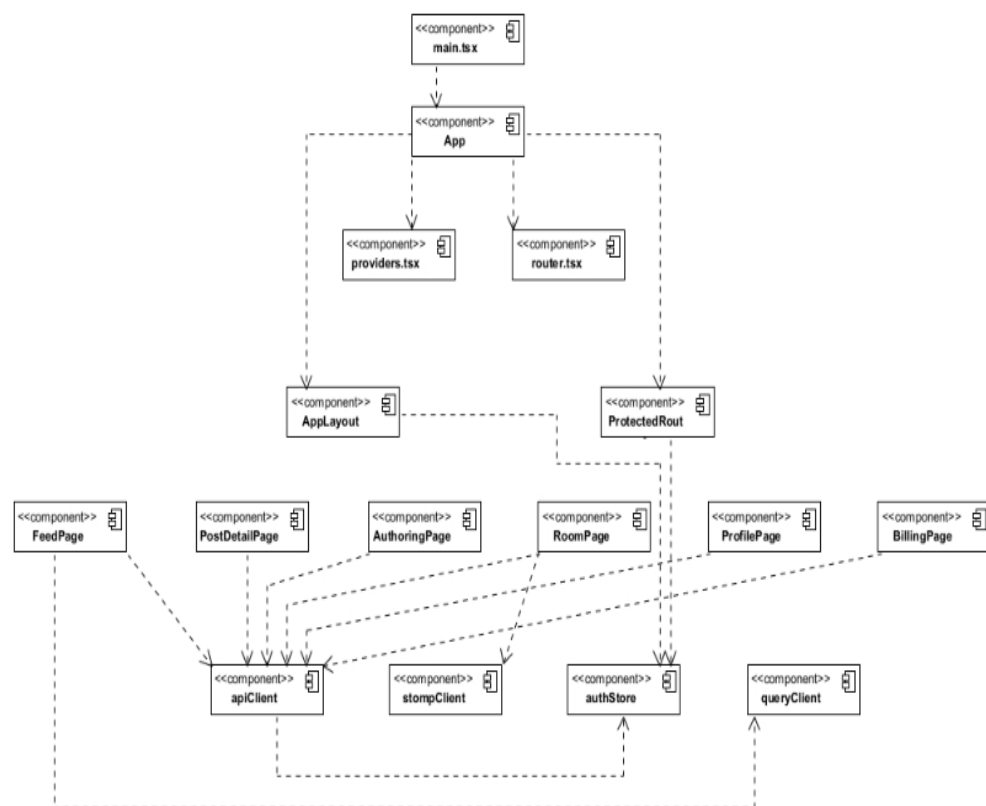


Рисунок 37 – Диаграмма компонентов клиентской части приложения

Описание компонентов клиентской части приложения представлено в таблице 15.

Таблица 15 – Компоненты клиентской части приложения

Компонент	Назначение
main.tsx	Входная точка React-приложения, монтирование корневого компонента и подключение провайдеров
App.tsx	Корневой компонент приложения
router.tsx	Конфигурация маршрутизации приложения
providers.tsx	Поставщики контекста: клиент запросов и маршрутизатор
AppLayout.tsx	Каркас приложения с боковой панелью навигации и областью контента; состав пунктов зависит от роли
ProtectedRoute.tsx	Защита маршрутов: проверка JWT-токена и перенаправление неаутентифицированных пользователей
FeedPage.tsx	Страница контентной ленты с постраничной загрузкой
PostDetailPage.tsx	Страница просмотра публикации с древовидными комментариями
AuthoringPage.tsx	Студия автора с редактором публикаций
ChatPage.tsx	Страница списка ИИ-терапевтических сессий
RoomPage.tsx	Окно терапевтической сессии с взаимодействием в реальном времени
ProfilePage.tsx	Личный кабинет пользователя с психологической анкетой
BillingPage.tsx	Страница управления подпиской и просмотра транзакций
FriendsPage.tsx	Страница управления кругом общения
Административные страницы	Страницы панели администрирования: пользователи, теги, биллинг
apiClient	HTTP-клиент на основе axios для запросов к серверной части
stompClient	Клиент WebSocket поверх протокола STOMP для взаимодействия в реальном времени
authStore	Хранилище состояния аутентификации на основе Zustand: токен и роль пользователя
queryClient	Клиент TanStack Query для управления серверным состоянием и кэширования

Для передачи данных между клиентом и сервером используется библиотека axios, реализующая обмен по протоколу HTTP, а для взаимодействия в реальном времени в терапевтических сессиях – соединение по протоколу WebSocket. Для визуализации полной архитектуры и иллюстрации взаимодействия пользовательского устройства с сервером была разработана диаграмма развертывания, представленная на рисунке 38.

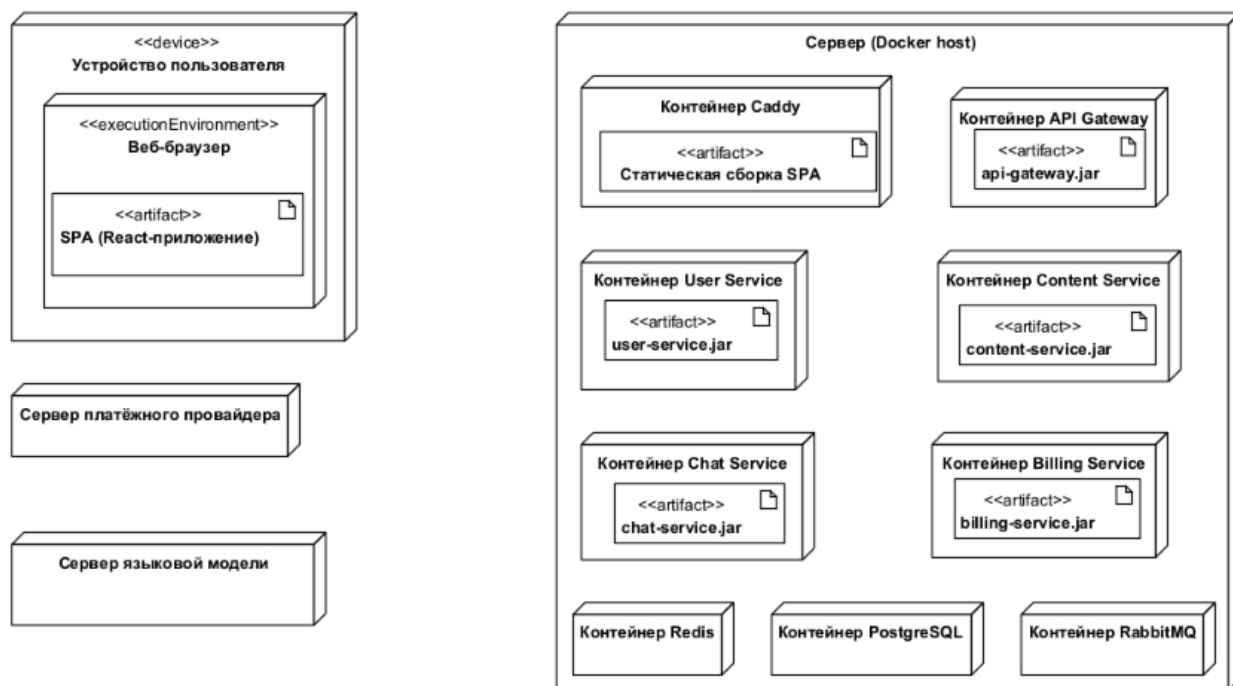


Рисунок 38 – Диаграмма развертывания приложения

Диаграмма развертывания моделирует физическое размещение артефактов на узлах. Серверная часть развернута на удаленной вычислительной машине средствами контейнеризации Docker: каждый микросервис, веб-сервер, СУБД, кэш и брокер сообщений исполняются в отдельных контейнерах. Веб-сервер Caddy отдает статическую сборку клиентского приложения и проксирует запросы к серверной части: обращения к REST API направляются к сервису-шлюзу, а соединения по WebSocket – напрямую к сервису ИИ-сессий. В роли клиентов выступают веб-браузеры на устройствах пользователей. Сервис ИИ-сессий и сервис биллинга дополнительно взаимодействуют с внешними системами – сервером языковой модели и сервером платежного провайдера.

По результатам кодирования была разработана информационная система оказания психологической помощи, позволяющая пользователям получать психологическую поддержку без необходимости установки стороннего программного обеспечения. Реализованное приложение было запущено согласно диаграмме развертывания.

4 Тестирование информационной системы оказания психологической помощи

4.1 Функциональное тестирование

Для информационной системы оказания психологической помощи функциональное тестирование гарантирует, что основные операции, предоставляемые приложением, выполняются корректно и безопасно для пользователей.

Функциональное тестирование – вид тестирования, который проверяет, насколько точно реализованные в приложении функции соответствуют предъявленным требованиям. Обычно данный подход называется тестированием «черным ящиком», то есть оценивается только внешнее поведение системы, а ее внутреннее устройство при этом не учитывается [16].

В рамках функционального тестирования системы было проведено тестирование «черным ящиком» формы регистрации пользователя следующими методиками:

- классы эквивалентности;
- функциональная диаграмма и таблица решений;
- диаграмма состояний и переходов.

Далее рассмотрим каждый из перечисленных способов проектирования тестовых сценариев более подробно

4.1.1 Классы эквивалентности

Классы эквивалентности – группы входных значений, при обработке которых приложение должно демонстрировать одинаковое поведение и выдавать одинаковый результат. Такое разбиение упорядочивает процесс тестирования: на его основе формируется компактный набор проверок, подтверждающий корректность и полноту реализации функции. Классы эквивалентности поля «Имя пользователя» формы регистрации приведены в таблице 16.

Таблица 16 – Классы эквивалентности поля «Имя пользователя»

Параметр	Правильные классы	Неправильные классы
Длина	3–30 символов (1)	0–2 символа (2); 31 символ и более (3)
Первый символ	начинается с буквы (4)	начинается не с буквы (5)
Состав символов	латинские буквы (6); цифры (7); дефис (8); подчеркивание (9)	кириллица (10); пробелы (11); прочие спецсимволы (12)

Для данных классов были составлены соответствующие тест-кейсы, представленные в таблице 17.

Таблица 17 – Тест-кейсы для поля «Имя пользователя»

№	Покрываемые КЭ	Входные данные	Ожидаемый результат
1	1, 4, 6, 7, 8, 9	Ivan_2024-01	Успешно
2	2, 4, 6	iv	Ошибка: некорректная длина имени пользователя
3	3, 4, 6	aaaaaaaaaaaaaaaaaaaaaaaaaaaaa	Ошибка: некорректная длина имени пользователя
4	1, 5, 7	2024user	Ошибка: имя пользователя должно начинаться с буквы
5	1, 4, 10	Иван	Ошибка: имя пользователя содержит недопустимые символы
6	1, 4, 11	Ivan Petrov	Ошибка: имя пользователя содержит недопустимые символы
7	1, 4, 12	Ivan!	Ошибка: имя пользователя содержит недопустимые символы

Каждый тест-кейс включает в себя классы эквивалентности, которые будут выполняться, входные и выходные данные. Выполнение всех тест-кейсов подтвердило корректную обработку формой как допустимых, так и недопустимых значений.

4.1.2 Функциональная диаграмма и таблица решений

Функциональная диаграмма – визуальная схема, отражающая функции системы и связи между ними; при помощи нее можно увидеть, как отдельные компоненты взаимодействуют друг с другом для достижения целевых результатов.

При применении этой методики требовалось перечислить все возможные причины некорректной работы системы, зафиксировать сообщения об ошибках, на основе чего строится упрощенная схема работы приложения. В результате были проанализированы поля формы регистрации и выделены причины и следствия событий, приведенные в таблице 18.

Таблица 18 – Причины и следствия событий

Обозначение	Описание
П1	Корректная длина имени пользователя
П2	Корректный состав символов имени пользователя
П3	Заполнено поле «Полное имя»
П4	Корректная длина пароля
С1	Сообщение: «Некорректная длина имени пользователя»
С2	Сообщение: «Имя пользователя содержит недопустимые символы»
С3	Сообщение: «Не указано полное имя»
С4	Сообщение: «Пароль слишком короткий»
С5	Сообщение: «Регистрация выполнена успешно»

По данной таблице была составлена функциональная диаграмма, представленная на рисунке 39.

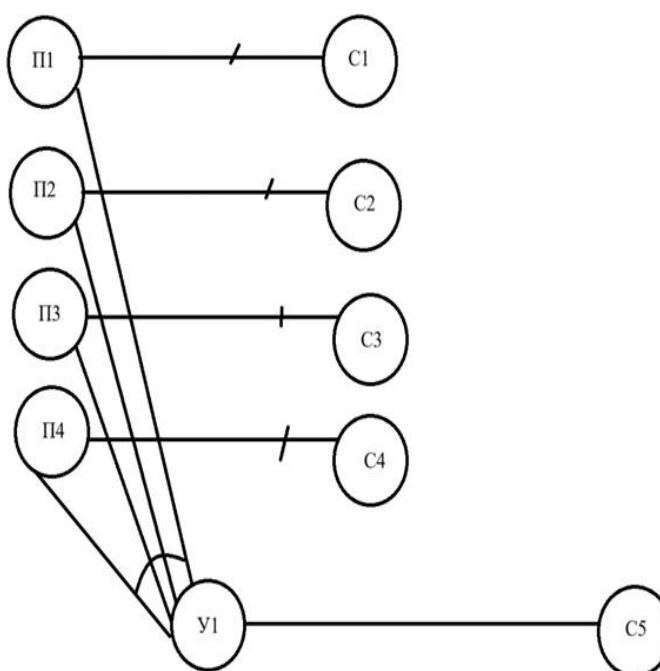


Рисунок 39 – Функциональная диаграмма

Таблица с представлением полного перебора всех существующих ограничений составляет 16 вариантов. Чтобы уменьшить число проверок без потери охвата, был применен комбинаторный подход – попарное тестирование. Метод гарантирует, что для каждой возможной пары входных параметров рассматриваются все сочетания значений, тем самым ощутимо сокращая объем сценариев. Из исходных 16 вариантов осталось 8 тест-сценариев. Окончательный перечень приведен в таблице 19.

Таблица 19 – Таблица решений

Условие	1	2	3	4	5	6	7	8
П1	0	0	0	0	1	1	1	1
П2	0	0	1	1	0	0	1	1
П3	0	1	0	1	0	1	0	1
П4	0	1	1	0	1	0	0	1
С1	1	1	1	1	0	0	0	0
С2	1	1	0	0	1	1	0	0
С3	1	0	1	0	1	0	1	0
С4	1	0	0	1	0	1	1	0
С5	0	0	0	0	0	0	0	1

Результат выполнения каждого тестового сценария, определенного в таблице 19, привел к успешному выполнению тестирования. Таким образом, можно сделать вывод о том, что разработанная форма регистрации работает корректно.

4.1.3 Диаграмма состояний и переходов

Диаграмма состояний и переходов отображает все возможные состояния системы и пути, по которым она может перемещаться между ними. Узлы на схеме обозначают конкретные состояния, а стрелки – события или условия, вызывающие переходы. Применение данной методики позволяет эффективно выявлять дефекты в бизнес-логике и гарантировать покрытие всех маршрутов компонента. В качестве объекта тестирования была взята терапевтическая комната, жизненный цикл которой спроектирован в подразделе 2.3.4 и представлен диаграммой состояний на рисунке 26.

На основе диаграммы состояний были составлены тестовые сценарии, каждый из которых соответствует пути по графу состояний и покрывает определенный набор переходов. Перечень сценариев приведен в таблице 20.

Таблица 20 – Тестовые сценарии на основе диаграммы состояний

№	Тестовый сценарий (путь по состояниям)	Покрываемые переходы	Результат
1	Создание соло-сессии и ее завершение: CREATED → ACTIVE → ARCHIVED	активация соло-комнаты, архивация	Успешно
2	Парная сессия, штатный путь: CREATED → WAITING_CONSENT → ACTIVE → ENDING → ARCHIVED	присоединение, взаимное согласие, предложение и подтверждение завершения	Успешно
3	Отзыв согласия до начала: CREATED → WAITING_CONSENT → WAITING_CONSENT	присоединение, отзыв согласия	Успешно
4	Отклонение завершения и продолжение диалога: ACTIVE → ENDING → ACTIVE	предложение завершения, отклонение вторым участником	Успешно

Использование данной диаграммы в функциональном тестировании дало несколько результатов:

- сокращение времени на составление сценариев;
- каждая ветка жизненного цикла была проверена минимум одним тестом;
- подтверждение корректности переходов между состояниями комнаты.

Таким образом, визуальное моделирование состояний не только упростило разработку тестов, но и повысило логическую уверенность в корректности ключевых операций над терапевтической сессией.

4.2 Тестирование API средствами Postman

Поскольку серверная часть системы построена по микросервисной архитектуре и предоставляет интерфейсы в виде REST API, важной частью тестирования является проверка корректности работы программных интерфейсов. Для этого был использован инструмент Postman, представляющий собой платформу для построения и использования API. Postman позволяет отправлять запросы, анализировать ответы и автоматизировать тестирование

программных интерфейсов, упрощая взаимодействие с серверной частью приложения [17].

Тестирование выполнялось по сценарию, воспроизводящему типовой путь пользователя: регистрация, авторизация с получением JWT-токена и последующее обращение к защищенному эндпоинту с этим токеном. Рассмотрим запрос авторизации на рисунке 40.

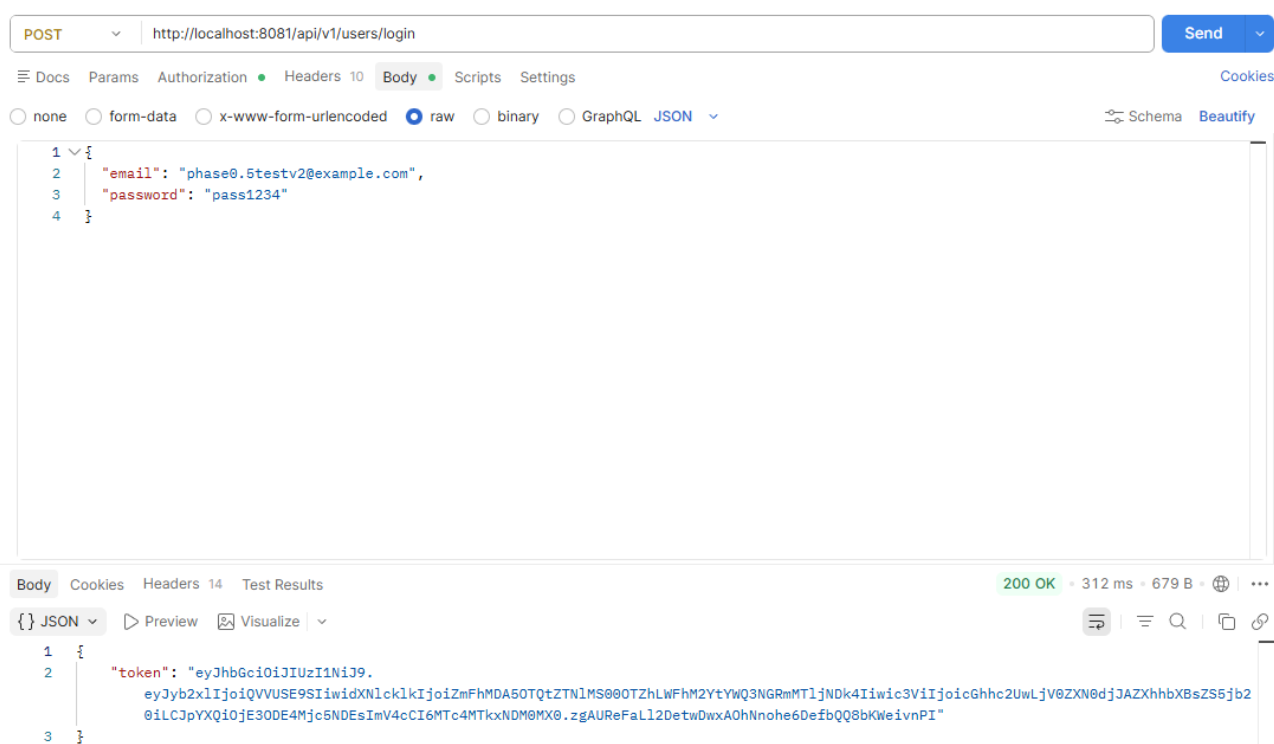


Рисунок 40 – Запрос авторизации пользователя

На представленном рисунке видна рабочая область Postman, включающая следующие элементы:

- строку ввода метода и адреса запроса с кнопкой отправки;
- вкладку тела запроса с передаваемыми учетными данными в формате JSON;
- вкладку ответа, где отображается возвращенный сервером JWT-токен и код состояния ответа.

Полученный токен подставляется в заголовок Authorization последующих запросов. Рассмотрим обращение к защищенному эндпоинту создания публикации на рисунке 41.

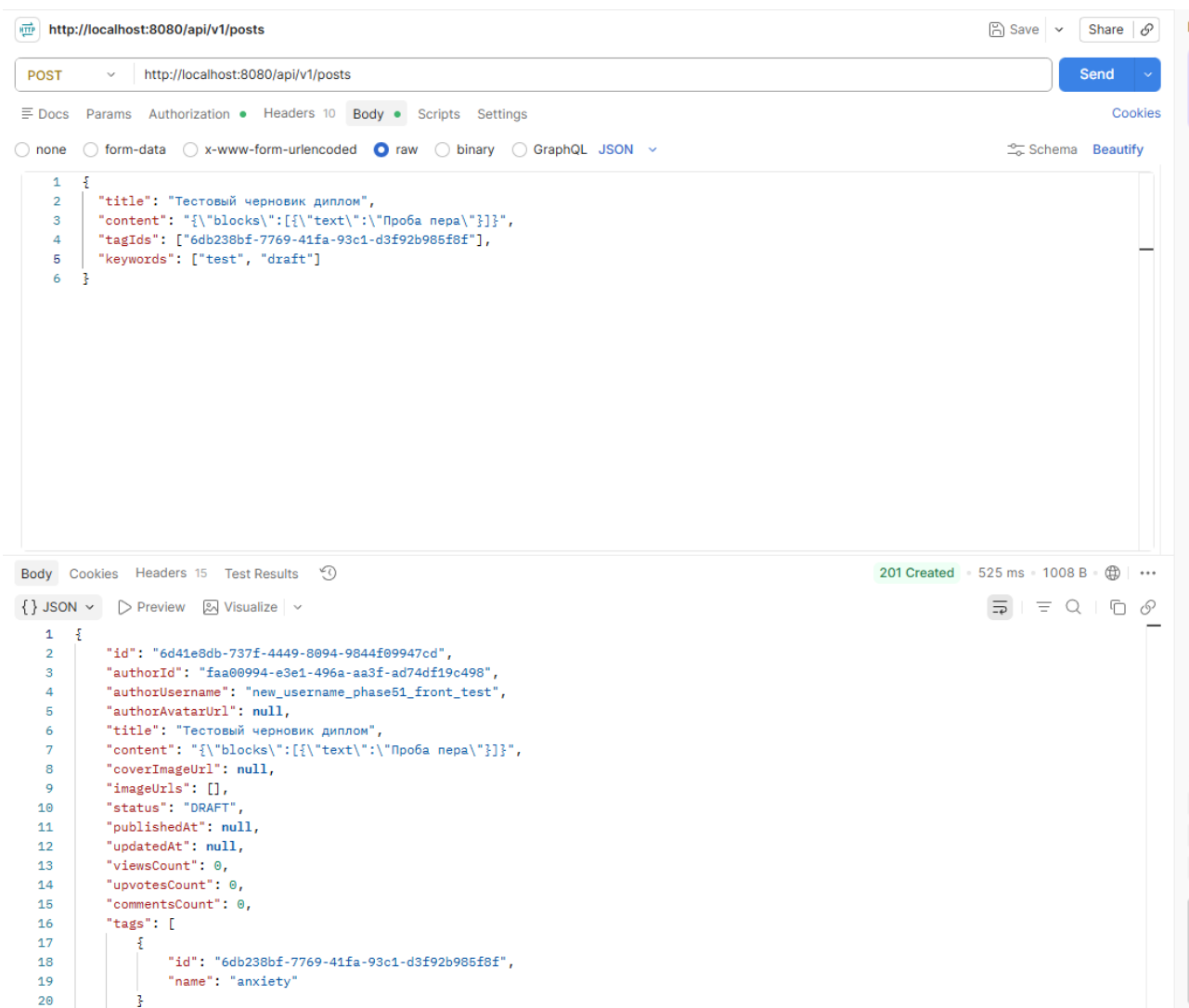


Рисунок 41 – Авторизованный запрос создания публикации

На рисунке видно, что запрос содержит заголовок `Authorization` со значением токена, а в теле передаются данные создаваемой публикации. В ответ сервер вернул код состояния об успешном создании ресурса и тело созданной публикации, что подтверждает корректную работу как механизма авторизации, так и эндпоинта создания контента.

Подобным образом средствами Postman были протестированы основные эндпоинты всех микросервисов системы: управление профилем и социальным графом, операции с публикациями, комментариями и лентой, а также сценарии биллинга. Благодаря встроенной функции запуска коллекций проверки были автоматизированы, а их результаты задокументированы.

Таким образом, тестирование API средствами Postman подтвердило корректность работы программных интерфейсов серверной части и механизма разграничения доступа на основе JWT-токенов.

4.3 Модульное тестирование клиентской части

Модульный тест – это автоматизированный тест, предназначенный для проверки корректности работы отдельного модуля программы в изоляции от других компонентов. Он позволяет убедиться, что конкретная часть кода функционирует согласно заданным требованиям, независимо от внешних зависимостей [16].

Модульное тестирование клиентской части выполнялось с помощью библиотеки Vitest в сочетании с React Testing Library. Vitest обеспечивает высокую скорость выполнения тестов и интегрирован со сборщиком Vite, а также предоставляет поддержку мокирования, измерения покрытия кода и снапшотов, что делает его удобным инструментом для современных веб-приложений [22]. React Testing Library позволяет рендерить компоненты в тестовом окружении и эмулировать действия пользователя.

Рассмотрим работу модульных тестов на примере проверки компонента формы авторизации LoginForm. Тест убеждается, что при заполнении полей и отправке формы вызывается функция входа с корректными данными. Данный тест представлен в листинге 22.

Листинг 22 – Тест отправки формы авторизации

```
it('отправляет введённые данные при входе', async () => {
  const loginMock = vi.fn().mockResolvedValue({ token: 'jwt-token' })
  vi.spyOn(api, 'loginUser').mockImplementation(loginMock)

  renderWithProviders(<LoginForm />)
  const user = userEvent.setup()

  await user.type(screen.getByLabelText('Email'), 'user@mail.ru')
  await user.type(screen.getByLabelText('Пароль'), 'secret123')
  await user.click(screen.getByRole('button', { name: 'Войти' }))

  expect(loginMock).toHaveBeenCalledWith({
    email: 'user@mail.ru',
```

```

    password: 'secret123',
  })
})

```

По данному тесту можно заметить, что компонент LoginForm рендерится в тестовом окружении, а пользовательское взаимодействие эмулируется с помощью `userEvent` – модуля, позволяющего выполнять клики и ввод текста так, как это делает настоящий пользователь. Сетевой вызов функции авторизации подменяется заглушкой, что обеспечивает изоляцию модуля от серверной части. Тест имитирует ввод электронной почты и пароля, отправку формы, а затем проверяет, что функция входа была вызвана с корректными данными. Если данные переданы верно, тест считается пройденным. На рисунке 42 представлен результат выполнения этого теста в терминале.

```

✓ src/features/auth/components/LoginForm.test.tsx (1 test) 590ms
  ✓ LoginForm (1)
    ✓ отправляет введённые данные при входе 589ms

Test Files  1 passed (1)
Tests       1 passed (1)
Start at    03:24:54
Duration    1.72s (transform 81ms, setup 72ms, import 308ms, tests 590ms, environment 619ms)

PS E:\Unik\Diplom Development\frontend>

```

Рисунок 42 – Выполнение теста формы авторизации

Аналогичным образом были написаны и успешно выполнены модульные тесты для других компонентов клиентской части, в том числе для формы регистрации, управления кругом общения, оформления подписки и оценки публикаций. Общий результат выполнения всех модульных тестов представлен на рисунке 43.

```

✓ src/features/billing/components/CheckoutPanel.test.tsx (1 test) 148ms
✓ src/features/feed/components/PostCard.test.tsx (1 test) 175ms
✓ src/pages/FriendsPage.test.tsx (2 tests) 342ms
✓ src/features/auth/components/LoginForm.test.tsx (1 test) 490ms
  ✓ отправляет введённые данные при входе 489ms
✓ src/features/auth/components/RegisterForm.test.tsx (1 test) 1080ms
  ✓ отправляет введённые данные при регистрации 1079ms

Test Files  5 passed (5)
Tests       6 passed (6)
Start at    06:08:19
Duration    2.29s (transform 422ms, setup 706ms, import 1.21s, tests 2.23s, environment 3.21s)

PS E:\Unik\Diplom Development\frontend>

```

Рисунок 43 – Результат выполнения всех модульных тестов

Использование Vitest для модульного тестирования клиентской части позволило обеспечить высокое качество кода, оперативно обнаруживать ошибки и гарантировать стабильную работу интерфейса. Полный код модульных тестов приведён в приложении Е.

4.4 Оценка качества программного кода

Метрики программного кода являются важнейшими инструментами для оценки и анализа характеристик качества разрабатываемого программного обеспечения. Используя метрики, разработчики могут изучить технические характеристики продукта, прогнозировать трудоемкость работы и оценивать качество принятых решений [18].

Для оценки качества кода разработанной информационной системы были выбраны широко известные и хорошо зарекомендовавшие себя метрики – метрика Холстеда и метрика Чепина.

Метрика Холстеда основана на количественном анализе используемых в коде операторов и операндов. Она позволяет определить сложность реализации программы, ее объем, а также прогнозировать трудозатраты на дальнейшую поддержку и расширение функционала.

Метрика Чепина предназначена для оценки внутренней логики программных модулей. Она классифицирует и подсчитывает переменные, используемые внутри программного модуля, позволяя таким образом определить его сложность и улучшить понятность кода.

4.4.1 Метрика Холстеда

Для количественной оценки сложности программного кода была проанализирована функция `generateToken` класса `JwtService`, формирующая JWT-токен пользователя.

Исходный код для метрики Холстеда рассматривается как набор двух типов лексем – операторов и операндов. Метрика требует зафиксировать:

- количество уникальных операторов (n_1);

- общее число всех операторов (N1);
- количество уникальных операндов (n2);
- общее число всех операндов (N2).

Эти четыре базовые величины служат входными параметрами формул Холстеда. Перечень операторов, встречающихся в функции generateToken, а также частота их использования приведены в таблице 21.

Таблица 21 – Операторы функции generateToken

№	Оператор	Число вхождений
1	=	1
2	new	1
3	put()	2
4	toString()	1
5	fromId()	1
6	getName()	1
7	return	1
8	createToken()	1
	n1 = 8	N1 = 9

Перечень операндов, встречающихся в функции generateToken, а также частота их использования приведены в таблице 22.

Таблица 22 – Операнды функции generateToken

№	Операнд	Число вхождений
1	generateToken	1
2	userId	2
3	email	2
4	roleId	2
5	claims	4
6	HashMap	1
7	«serId»	1
8	«role»	1
9	UserRole	1
	n2 = 9	N2 = 15

Результаты лексического анализа функции послужили исходными данными для вычисления всех показателей по методике Холстеда. Итоговые значения сведены в таблице 23.

Таблица 23 – Результаты вычислений по метрике Холстеда

Показатель	Значение
Словарь программы (n)	17
Длина программы (N)	24
Теоретическая длина ($N^{\wedge}$)	52,53
Объем программы (V)	98,10
Потенциальный объем (V^*)	15,51
Уровень программы (L)	0,16
Сложность программы (D)	6,33
Трудоемкость (E)	620,63

Таким образом, вычисленные метрики Холстеда позволили количественно оценить сложность реализации функции, ее объем и уровень программирования, подтвердив достаточный уровень компактности и понятности рассматриваемого кода.

4.4.2 Метрика Чепина

Для характеристики структурной сложности была произведена оценка по метрике Чепина для функции `persistAssistantTurn` класса `TurnPersistenceService`, отвечающей за сохранение ответа ИИ и передачу права голоса. Все идентификаторы функции проанализированы и распределены на четыре категории:

- параметры, поступающие на вход и участвующие в формировании результата (P);
- переменные, чье значение изменяется в ходе работы (M);
- переменные, влияющие на выбор ветви исполнения (C);
- неиспользуемые («паразитные») идентификаторы (T).

Так как один и тот же идентификатор способен выполнять несколько ролей, при классификации он учитывался во всех соответствующих категориях. Для каждой группы методика Чепина приписывает собственный коэффициент, отражающий ее вклад в общую сложность программного блока: для входных и выходных переменных коэффициент равен единице, для изменяемых – удвоенный, для управляющих – тройной, так как они наиболее воздействуют на

поток управления, для «паразитных» переменных – половинный. Результат классификации приведен в таблице 24.

Таблица 24 – Классификация переменных функции persistAssistantTurn

Категория	Переменные	Количество	Коэффициент
Входные (P)	roomId, content, promptTokens, completionTokens	4	1
Изменяемые (M)	room, maxSeq, turn, currentHolderId, otherHolderId	5	2
Управляющие (C)	room, currentHolderId, id	3	3
Паразитные (T)	–	0	0,5

Итоговый показатель вычисляется как сумма произведений количества переменных каждой категории, умноженный на ее коэффициент, затем полученные произведения складываются.

Полученное значение метрики Чепина $Q = 23$ говорит о том, что функция persistAssistantTurn имеет умеренную структурную сложность. Это облегчает понимание и поддержку класса, улучшая надежность программы.

Таким образом, проведенное тестирование информационной системы – функциональное тестирование «черным ящиком», тестирование программных интерфейсов средствами Postman, модульное тестирование клиентской части и оценка качества кода по метрикам Холстеда и Чепина – подтвердило корректность реализованных функций, надежность взаимодействия клиентской и серверной частей и достаточное качество программного кода.

Заключение

В рамках выполнения выпускной квалификационной работы была разработана информационная система оказания психологической помощи, позволяющая повысить доступность психологической поддержки и объединить инструменты саморефлексии, профессиональный контент и социальное взаимодействие в единой безопасной экосистеме.

В начале был выполнен анализ предметной области и существующих программных решений, включая контентные психологические порталы, анкетные агрегаторы специалистов и сервисы онлайн-психотерапии с алгоритмическим подбором. На основе данного анализа были сформированы задачи на разработку с подробным описанием функциональных и нефункциональных требований. Также был проанализирован и выбран набор современных инструментов и технологий, а также выполнена оценка проекта с приближенным планированием бюджета.

На этапе проектирования была разработана архитектура приложения, состоящая из клиентской и серверной части. Клиентская часть была спроектирована с учетом современных тенденций и стандартов веб-разработки и специфики предметной области, требующей создания спокойного и безопасного пространства. Серверная часть включила в себя проектирование каждого из микросервисов по отдельности: сервиса-шлюза, сервиса пользователей, сервиса контента, сервиса ИИ-сессий и сервиса биллинга. Была выполнена разработка концептуальной, логической и физической модели базы данных с разделением на изолированные схемы по границам микросервисов.

Разработка системы включила в себя реализацию взаимодействия с базой данных, в том числе версионированных миграций, генерируемых столбцов и нативных запросов полнотекстового поиска на языке SQL. Было произведено наполнение базы данных начальными данными. Серверная часть включила в себя реализацию на языке Java механизма аутентификации на основе JWT-токенов, оркестрации очередности ходов и взаимодействия с языковой моделью в терапевтических сессиях, а также гарантированной доставки событий между

сервисами по паттерну Transactional Outbox. Клиентский модуль был реализован с приоритетом функциональной полноты и лаконичности; в его состав вошли взаимодействие в реальном времени по протоколу WebSocket, согласованное с сервером управление состоянием и ролевое разграничение доступа.

Также было проведено тщательное тестирование разработанной системы. С помощью функционального тестирования методами классов эквивалентности, функциональной диаграммы и диаграммы состояний и переходов был протестирован основной функционал приложения. Для проверки работы программных интерфейсов серверной части применялся инструмент Postman, результаты подтвердили корректность работы API и механизма разграничения доступа. Надежность и эффективность программного кода дополнительно была проверена с помощью метрик Холстеда и Чепина, их значения оказались в рамках умеренной сложности.

Проведенные работы по созданию приложения показали, что разработанная информационная система полностью удовлетворяет поставленным на разработку задачам.

Дальнейшее развитие данной системы подразумевает расширение возможностей ИИ-модерации, в том числе внедрение групповых терапевтических сессий, разработку мобильного приложения, а также интеграцию с реальным платежным провайдером и сервисами подбора специалистов, что позволит увеличить охват аудитории и расширить спектр предоставляемой психологической поддержки.

Список использованных источников

1. Психическое здоровье // Всемирная организация здравоохранения [Электронный ресурс]. URL: <https://www.who.int/ru/news/item/17-06-2022-who-highlights-urgent-need-to-transform-mental-health-and-mental-health-care> (дата обращения: 10.06.2026).
2. Б17 – сайт психологов // Б17 [Электронный ресурс]. URL: <https://www.b17.ru> (дата обращения: 10.06.2026).
3. Поиск психолога // ПроДокторов [Электронный ресурс]. URL: <https://prodoctorov.ru/psiholog/> (дата обращения: 10.06.2026).
4. Ясно – сервис подбора психолога онлайн // Ясно [Электронный ресурс]. URL: <https://yasno.live> (дата обращения: 10.06.2026).
5. Unified Modeling Language (UML) Specification. Version 2.5.1 // Object Management Group [Электронный ресурс]. URL: <https://www.omg.org/spec/UML/2.5.1/About-UML> (дата обращения: 10.06.2026).
6. Spring Boot Reference Documentation // Spring [Электронный ресурс]. URL: <https://docs.spring.io/spring-boot/index.html> (дата обращения: 10.06.2026).
7. Ньюмен, С. Создание микросервисов / С. Ньюмен. – 2-е изд. – СПб. : Питер, 2023. – 624 с.
8. PostgreSQL 16 Documentation // PostgreSQL Global Development Group [Электронный ресурс]. URL: <https://www.postgresql.org/docs/16/> (дата обращения: 10.06.2026).
9. React – The library for web and native user interfaces // Meta Open Source [Электронный ресурс]. URL: <https://react.dev> (дата обращения: 10.06.2026).
10. Управление проектами в MS Project: планирование и контроль // KoderLine [Электронный ресурс]. URL: <https://koderline-projects.ru/tekhnologii/stati-ekspertov/upravlenie-proektami-v-ms-project-planirovanie-i-kontrol/> (дата обращения: 10.06.2026).
11. RFC 6455. The WebSocket Protocol // Internet Engineering Task Force [Электронный ресурс]. URL: <https://datatracker.ietf.org/doc/html/rfc6455> (дата обращения: 10.06.2026).

12. Дейт, К. Дж. Введение в системы баз данных / К. Дж. Дейт. – 8-е изд. – М. : Вильямс, 2018. – 1328 с.
13. Flyway Documentation // Redgate Software [Электронный ресурс]. URL: <https://documentation.red-gate.com/flyway> (дата обращения: 10.06.2026).
14. Hibernate ORM Documentation // Red Hat [Электронный ресурс]. URL: <https://hibernate.org/orm/documentation/6.4/> (дата обращения: 10.06.2026).
15. TanStack Query Documentation // TanStack [Электронный ресурс]. URL: <https://tanstack.com/query/latest/docs/framework/react/overview> (дата обращения: 10.06.2026).
16. Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – 3-е изд. – Минск : Четыре четверти, 2022. – 312 с.
17. Postman Documentation // Postman, Inc. [Электронный ресурс]. URL: <https://learning.postman.com/docs/introduction/overview/> (дата обращения: 10.06.2026).
18. Холстед, М. Х. Начала науки о программах / М. Х. Холстед. – М. : Финансы и статистика, 1981. – 128 с.
19. pgAdmin 4 Documentation // The pgAdmin Development Team [Электронный ресурс]. URL: <https://www.pgadmin.org/docs/> (дата обращения: 10.06.2026).
20. Docker Documentation // Docker Inc. [Электронный ресурс]. URL: <https://docs.docker.com/> (дата обращения: 10.06.2026).
21. RabbitMQ Documentation // Broadcom [Электронный ресурс]. URL: <https://www.rabbitmq.com/docs> (дата обращения: 10.06.2026).
22. Vitest – Next Generation Testing Framework // VoidZero [Электронный ресурс]. URL: <https://vitest.dev> (дата обращения: 10.06.2026).

Приложение А
(справочное)
Глоссарий

БД (База данных) – упорядоченный набор взаимосвязанных данных, хранимых по заранее определённой структуре и обеспечивающих систематическое накопление, поиск и совместное использование информации.

СУБД (Система управления базами данных) – программный комплекс для создания, модификации и обслуживания баз данных, обеспечивающий многопользовательский доступ, целостность и безопасность данных.

PostgreSQL – объектно-реляционная СУБД с открытым исходным кодом, поддерживающая ACID-транзакции, расширяемые типы данных и полнотекстовый поиск.

UUID (Universally Unique Identifier) – 128-битный уникальный идентификатор, генерируемый без центрального координатора и используемый в качестве первичного ключа сущностей.

JSONB – бинарный тип данных PostgreSQL для хранения документов в формате JSON с возможностью индексирования и эффективного доступа к полям.

JWT (JSON Web Token) – компактный самодостаточный токен в формате JSON, используемый для передачи проверяемых утверждений о пользователе при аутентификации.

API (Application Programming Interface) – задокументированный набор правил, эндпоинтов и форматов сообщений, предоставляющий внешним программам доступ к функциям системы.

REST (Representational State Transfer) – архитектурный стиль построения программных интерфейсов поверх протокола HTTP, основанный на представлении ресурсов и стандартных методах доступа к ним.

WebSocket – протокол полнодуплексного постоянного обмена сообщениями поверх TCP, применяемый для двустороннего взаимодействия клиента и сервера в реальном времени.

STOMP (Simple Text Oriented Messaging Protocol) – простой текстовый протокол обмена сообщениями, используемый поверх WebSocket для организации подписок на каналы и публикации команд.

Микросервис – автономный программный сервис, отвечающий за ограниченную область бизнес-логики, развёртываемый и масштабируемый независимо от других сервисов.

Docker – платформа контейнеризации, позволяющая упаковывать приложение и его зависимости в изолированные образы, обеспечивая воспроизводимость и переносимость среды.

RabbitMQ – брокер сообщений, реализующий асинхронный обмен событиями между сервисами и выступающий в роли шины событий.

Redis – резидентное хранилище данных по схеме «ключ–значение», применяемое для кэширования и хранения эфемерных данных.

Transactional Outbox – шаблон проектирования, при котором бизнес-изменение и факт публикации события сохраняются в одной транзакции базы данных, что гарантирует доставку события.

Saga (Сага) – шаблон управления распределённой транзакцией как последовательностью локальных транзакций сервисов, согласуемых через обмен событиями.

ORM (Object-Relational Mapping) – технология сопоставления объектов программы строкам реляционной базы данных, избавляющая от написания части SQL-запросов вручную.

Flyway – инструмент управления версионированными миграциями базы данных, применяющий изменения схемы в заданном порядке.

SPA (Single Page Application) – одностраничное веб-приложение, которое динамически обновляет содержимое страницы без её полной перезагрузки.

Языковая модель – модель искусственного интеллекта, генерирующая связный текст на основе переданного контекста; в системе используется для модерации терапевтического диалога.

Фреймворк – программная платформа, выступающая каркасом с архитектурными принципами и едиными нормативами, направляющими разработку приложений.

Приложение Б
(обязательное)
Листинг серверной части приложения

Листинг Б.1 – Файл user-service/security/JwtService.java

```

package com.diplom.userservice.security;

import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import io.jsonwebtoken.security.Keys;
import org.springframework.stereotype.Service;

import java.security.Key;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.UUID;
import com.diplom.userservice.entity.UserRole;
import io.jsonwebtoken.Claims;

import org.springframework.beans.factory.annotation.Value;

@Service
public class JwtService {

    @Value("${jwt.secret}")
    private String secretKey;

    @Value("${jwt.expiration-hours}")
    private long expirationHours;

    public String generateToken(UUID userId, String email, Integer
roleId) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("userId", userId.toString());
        claims.put("role", UserRole.fromId(roleId).getName());

        return createToken(claims, email);
    }

    private String createToken(Map<String, Object> claims, String
subject) {
        return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis()
+ (expirationHours * 3600000)))
            .signWith(getSigninKey(),
SignatureAlgorithm.HS256)
            .compact();
    }
}

```

```

        private Key getSigninKey() {
            byte[] keyBytes =
io.jsonwebtoken.io.Decoders.BASE64.decode(secretKey);
            return Keys.hmacShaKeyFor(keyBytes);
        }

        public Claims extractAllClaims(String token) {
            return Jwts.parserBuilder()
                .setSigningKey(getSigninKey())
                .build()
                .parseClaimsJws(token)
                .getBody();
        }

        public boolean isTokenValid(String token) {
            try {
                return
!extractAllClaims(token).getExpiration().before(new Date());
            } catch (Exception e) {
                return false;
            }
        }
    }
}

```

Листинг Б.2 – Файл user-service/security/JwtAuthenticationFilter.java

```

package com.diplom.userservice.security;

import io.jsonwebtoken.Claims;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.UsernamePasswordAuthen
ticationToken;
import org.springframework.security.core.authority.SimpleGrantedAuthority
;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;
import java.util.Collections;
import java.util.UUID;

@Component
@RequiredArgsConstructor

```

```

public class JwtAuthenticationFilter extends OncePerRequestFilter
{
    private final JwtService jwtService;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {

        final String authHeader =
request.getHeader("Authorization");

        if (authHeader == null || !authHeader.startsWith("Bearer
")) {
            filterChain.doFilter(request, response);
            return;
        }

        final String token = authHeader.substring(7);

        if (jwtService.isTokenValid(token)) {
            Claims claims = jwtService.extractAllClaims(token);
            String email = claims.getSubject();
            UUID userId = UUID.fromString(claims.get("userId",
String.class));
            String roleName = claims.get("role", String.class);

            CustomUserDetails userDetails = new
CustomUserDetails(userId, email, roleName);

            UsernamePasswordAuthenticationToken authToken = new
UsernamePasswordAuthenticationToken(
                userDetails,
                null,
                Collections.singleton(new
SimpleGrantedAuthority("ROLE_" + roleName))
            );

```

```
SecurityContextHolder.getContext().setAuthentication(authToken);  
    }  
  
    filterChain.doFilter(request, response);  
}  
}
```

Приложение В
(обязательное)
Листинг клиентской части приложения

Листинг В.1 – Файл src/shared/ws/stompClient.ts

```
import { Client } from '@stomp/stompjs'
import type { StompSubscription } from '@stomp/stompjs'
import SockJS from 'sockjs-client'
import { useAuthStore } from '../stores/authStore'

const getWsBaseUrl = () => {
  return (
    import.meta.env.VITE_WS_BASE_URL ??
    import.meta.env.VITE_API_BASE_URL ??
    'http://localhost:8080'
  )
}

export class StompClientWrapper {
  private client: Client | null = null
  private connectionPromise: Promise<void> | null = null
  private onConnectCallbacks: Array<() => void> = []

  /** Register a callback that runs on every connect/reconnect.
  Returns an unsubscribe function. */
  onReconnect(callback: () => void): () => void {
    this.onConnectCallbacks.push(callback)
    if (this.client?.connected) {
      callback()
    }
    return () => {
      this.onConnectCallbacks =
this.onConnectCallbacks.filter((cb) => cb !== callback)
    }
  }

  connect(): Promise<void> {
    if (this.client?.connected) {
      return Promise.resolve()
    }
    if (this.connectionPromise) return this.connectionPromise

    this.connectionPromise = new Promise((resolve, reject) => {
      const token = useAuthStore.getState().token

      if (!token) {
        this.connectionPromise = null
        reject(new Error('No auth token available for STOMP
connection'))
        return
      }

      let resolved = false

      this.client = new Client({
```

```

        websocketFactory: () => new
SockJS(`${getWsBaseUrl()}/ws`),
        connectHeaders: {
            Authorization: `Bearer ${token}`,
        },
        reconnectDelay: 3000,
        // debug: (msg) => console.log('[STOMP]', msg),
        onConnect: () => {
            if (!resolved) {
                resolved = true
                resolve()
            }
            this.onConnectCallbacks.forEach((cb) => cb())
        },
        onStompError: (frame) => {
            console.error('Broker reported error: ' +
frame.headers['message'])
            console.error('Additional details: ' + frame.body)
            if (!resolved) {
                this.connectionPromise = null
                reject(new Error(frame.headers['message']))
            }
        },
        onWebSocketClose: () => {
            this.connectionPromise = null
        },
        onDisconnect: () => {
            this.connectionPromise = null
        },
        onWebSocketError: (event) => {
            console.error('WebSocket error:', event)
        },
    })

    this.client.activate()
})

return this.connectionPromise
}

disconnect() {
    if (this.client && this.client.active) {
        this.client.deactivate()
    }
    this.client = null
    this.connectionPromise = null
}

subscribe(destination: string, callback: (body: any) => void):
StompSubscription | null {
    if (!this.client || !this.client.connected) {

```

```

        console.warn('Cannot subscribe: STOMP client is not
connected')
        return null
    }

    return this.client.subscribe(destination, (message) => {
        try {
            const body = message.body ? JSON.parse(message.body) :
null
            callback(body)
        } catch (e) {
            console.error('Failed to parse STOMP message body:', e)
            callback(message.body)
        }
    })
}

publish(destination: string, body: any) {
    if (!this.client || !this.client.connected) {
        console.warn('Cannot publish: STOMP client is not
connected')
        return
    }

    this.client.publish({
        destination,
        body: JSON.stringify(body),
    })
}
}

export const stompClient = new StompClientWrapper()

```

Листинг В.2 – Файл src/features/feed/hooks/useFeed.ts

```

import { useInfiniteQuery } from '@tanstack/react-query'
import { getFeed } from '../api'
import type { SortMode } from '../types'

export function useFeed(params: { sort: SortMode; tags: string[]
}) {

```

```
const tagsKey = params.tags.slice().sort().join(',')

return useInfiniteQuery({
  queryKey: ['feed', params.sort, tagsKey],
  queryFn: ({ pageParam }) =>
    getFeed({ sort: params.sort, cursor: pageParam, tags:
params.tags }),
  initialPageParam: undefined as string | undefined,
  getNextPageParam: (lastPage) => lastPage.nextCursor ??
undefined,
})
}
```

Приложение Г
(рекомендуемое)
Атрибуты сущностей базы данных

Таблица Г.1 – Атрибуты отношения «Пользователь» (users)

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение null	Роль	Пример
id	Уникальный идентификатор пользователя	UUID	16	UUID версии 4	нет	PK	f47ac10b-...
email	Адрес электронной почты	varchar	255	корректный e-mail	нет	U	user@mail.ru
password_hash	Хеш пароля	varchar	255	строка хеша	нет	–	\$2a\$10\$...
role_id	Идентификатор роли	integer	4	1 – 5	нет	FK	2
status_id	Идентификатор статуса	integer	4	1 – 3	нет	FK	1
created_at	Дата и время регистрации	timestampz	8	дата и время	нет	–	2026-05-01 12:30+03
version	Версия записи (оптим. блокировка)	integer	4	целое ≥ 0	нет	–	0

Таблица Г.2 – Атрибуты отношения «Профиль» (profiles)

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение null	Роль	Пример
user_id	Идентификатор пользователя	UUID	16	UUID версии 4	нет	PK, FK	f47ac10b-...

Продолжение таблицы Г.2

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение null	Роль	Пример
full_name	Полное имя	varchar	255	строка	да	–	Иванова Арина
username	Отображаемое имя	varchar	100	латиница, цифры, дефис	да	U	arina_i
birth_date	Дата рождения	date	4	дата	да	–	2000-04-15
gender_id	Идентификатор пола	integer	4	1 – 3	да	FK	2
avatar_url	Ссылка на аватар	text	–	URL	да	–	https://.../a.png
psych_profile	Психологическая анкета	jsonb	–	JSON-документ	да	–	{«goals»:[...]}
bio	Текст «О себе»	text	–	строка	да	–	Психолог
contact_info	Контактные данные	text	–	строка	да	–	tg: @arina
updated_at	Дата и время изменения	timestampz	8	дата и время	нет	–	2026-05-02 09:00+03

Таблица Г.3 – Атрибуты отношения «Публикация» (posts)

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение null	Роль	Пример
id	Уникальный идентификатор	UUID	16	UUID версии 4	нет	PK	a1b2c3d4-...

Продолжение таблицы Г.3

Идентификатор столбца	Содержательное описание	Тип данных	Размерность	Область допустимых значений	Значение null	Роль	Пример
author_id	Идентификатор автора	UUID	16	UUID версии 4	нет	FK	f47ac10b-...
title	Заголовок публикации	varchar	255	строка	нет	—	Как справиться с тревогой
content	Содержимое	jsonb	—	JSON-документ	нет	—	{«blocks»:[...]}
cover_image_url	Ссылка на обложку	text	—	URL	да	—	https://.../c.png
keywords	Ключевые слова	text[]	—	массив строк	да	—	{тревога,сон}
status_id	Идентификатор статуса	integer	4	1 – 4	нет	FK	2
search_vector	Поисковый вектор (генерируемый)	tsvector	—	лексеммы текста	да	—	'тревог':2
upvotes_count	Счётчик оценок	integer	4	целое ≥ 0	нет	—	17
comments_count	Счётчик комментариев	integer	4	целое ≥ 0	нет	—	4
published_at	Дата публикации	timestampz	8	дата и время	да	—	2026-05-03 10:00+03
version	Версия записи	integer	4	целое ≥ 0	нет	—	0

Приложение Д
(рекомендуемое)
Скрипт для создания объектов базы данных

Листинг Д.1 – Файл V1__init_users_schema.sql

```

CREATE SCHEMA IF NOT EXISTS user_schema;

CREATE TABLE user_schema.user_roles (
    id INT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE user_schema.user_statuses (
    id INT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE user_schema.genders (
    id INT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE user_schema.users (
    id UUID PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role_id INT REFERENCES user_schema.user_roles(id),
    status_id INT REFERENCES user_schema.user_statuses(id),
    created_at TIMESTAMP WITH TIME ZONE NOT NULL,
    version INT DEFAULT 0 NOT NULL
);

-- UNIQUE constraint explicitly maintains a b-tree index
-- CREATE INDEX idx_users_email ON user_schema.users USING btree
(email);

CREATE TABLE user_schema.profiles (
    user_id UUID PRIMARY KEY REFERENCES user_schema.users(id),
    full_name VARCHAR(255) NOT NULL,
    birth_date DATE,
    gender_id INT REFERENCES user_schema.genders(id),
    username VARCHAR(100) UNIQUE NOT NULL,
    avatar_url TEXT,
    psych_profile JSONB,
    updated_at TIMESTAMP WITH TIME ZONE NOT NULL
);

CREATE TABLE user_schema.user_outbox_events (
    id UUID PRIMARY KEY,
    event_type VARCHAR(100) NOT NULL,
    payload JSONB NOT NULL,
    status VARCHAR(50) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL
);

```

Листинг Д.2 – Файл V1__init_content_schema.sql

```

CREATE SCHEMA IF NOT EXISTS content_schema;

CREATE TABLE content_schema.post_statuses (
    id INT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL
);

INSERT INTO content_schema.post_statuses (id, name) VALUES
    (1, 'DRAFT'),
    (2, 'PUBLISHED'),
    (3, 'ARCHIVED'),
    (4, 'MODERATED');

CREATE TABLE content_schema.posts (
    id UUID PRIMARY KEY,
    author_id UUID NOT NULL,
    cover_image_url TEXT NULL,
    title VARCHAR(255) NOT NULL,
    content JSONB NULL,
    keywords TEXT[] NULL,
    status_id INT NOT NULL REFERENCES
content_schema.post_statuses(id),
    published_at TIMESTAMP WITH TIME ZONE NULL,
    views_count INT NOT NULL DEFAULT 0,
    upvotes_count INT NOT NULL DEFAULT 0,
    comments_count INT NOT NULL DEFAULT 0,
    updated_at TIMESTAMP WITH TIME ZONE NOT NULL,
    version INT NOT NULL DEFAULT 0
);

CREATE TABLE content_schema.content_outbox_events (
    id UUID PRIMARY KEY,
    event_type VARCHAR(100) NOT NULL,
    payload JSONB NOT NULL,
    status VARCHAR(50) NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE NOT NULL
);

CREATE INDEX idx_content_outbox_events_status
    ON content_schema.content_outbox_events (status);

```

Приложение Е
(рекомендуемое)
Листинг модульных тестов

Листинг Е.1 – Файл src/test/setup.ts

```
import '@testing-library/jest-dom'
```

Листинг Е.2 – Файл src/test/renderWithProviders.tsx

```
import { render } from '@testing-library/react'
import { QueryClient, QueryClientProvider } from '@tanstack/react-query'
import { MemoryRouter } from 'react-router-dom'
import type { ReactElement } from 'react'

export function renderWithProviders(ui: ReactElement) {
  const queryClient = new QueryClient({
    defaultOptions: { queries: { retry: false } },
  })
  return render(
    <QueryClientProvider client={queryClient}>
      <MemoryRouter>{ui}</MemoryRouter>
    </QueryClientProvider>
  )
}
```

Листинг Е.3 – Файл src/features/auth/components/LoginForm.test.tsx

```
import { describe, it, expect, vi } from 'vitest'
import { screen } from '@testing-library/react'
import userEvent from '@testing-library/user-event'
import { renderWithProviders } from
'../../../../../test/renderWithProviders'
import { LoginForm } from './LoginForm'
import * as api from '../api'

describe('LoginForm', () => {
  it('отправляет введённые данные при входе', async () => {
    const loginMock = vi.fn().mockResolvedValue({ token: 'jwt-token' })
    vi.spyOn(api, 'loginUser').mockImplementation(loginMock)

    renderWithProviders(<LoginForm />)
    const user = userEvent.setup()

    await user.type(screen.getByLabelText('Email'),
'user@mail.ru')
    await user.type(screen.getByLabelText('Пароль'), 'secret123')
    await user.click(screen.getByRole('button', { name: 'Войти'
}))
  })
})
```

```
    expect(loginMock).toHaveBeenCalledWith({  
      email: 'user@mail.ru',  
      password: 'secret123',  
    })  
  })  
})
```