МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт цифровых технологий
(наименование института полностью)

Департамент бакалавриата
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки / специальности)

Разработка программного обеспечения
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

на тему «Разработка географически распределённого корпоративного мессенджера с поддержкой секретных чатов и интеграцией с системами управления проектами»

Обучающийся М.Е. Еремин
(Инициалы Фамилия) (личная подпись)

Руководитель к. п. н., доцент, Е.А.Ерофеева
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Выпускная квалификационная работа посвящена разработке географически распределённого корпоративного мессенджера с поддержкой секретных чатов и интеграцией с системами управления проектами. Целью исследования является создание безопасного и масштабируемого программного решения, обеспечивающего обмен сообщениями и совместную работу в распределённой корпоративной среде.

В ходе работы проведён анализ существующих решений и технологий защиты данных, спроектирована архитектура системы на основе микросервисного подхода, включающая сервисы аутентификации, обмена сообщениями, управления проектами и событийную шину. Для обеспечения отказоустойчивости и доставки сообщений в офлайн-режиме применены механизмы потоковой обработки и брокеры сообщений.

Результатом исследования является прототип мессенджера, обеспечивающий защищённый обмен данными, масштабируемость и интеграцию с корпоративными инструментами управления задачами. Разработанное решение может быть использовано в организациях с распределённой структурой для повышения эффективности взаимодействия между сотрудниками.

Работа изложена на 85 страницах, содержит 35 рисунков, 17 таблицы, 2 приложения и список литературы из 25 источников.

Содержание

Введение

Современные корпоративные информационные системы предъявляют повышенные требования к защищённости, масштабируемости и интеграции с бизнес-процессами. В условиях перехода организаций к удалённым и гибридным форматам работы особенно возрастает потребность в надёжных средствах коммуникации между сотрудниками, подразделениями и руководством. Для компаний с географически распределённой структурой эффективное взаимодействие между удалёнными офисами становится ключевым условием устойчивого функционирования.

На практике многие предприятия используют сторонние мессенджеры, такие как Slack, Microsoft Teams или Telegram. Однако их применение связано с рядом ограничений: отсутствием контроля над корпоративными данными, риском утечек, высокой стоимостью владения и недостаточной интеграцией с внутренними системами управления проектами. Эти факторы формируют потребность в разработке собственного корпоративного мессенджера, обеспечивающего безопасность, управляемость и гибкость при взаимодействии в распределённой среде.

Объектом исследования является корпоративная система обмена сообщениями, функционирующая в распределённой информационной среде.

Предметом исследования выступают архитектурные и технологические решения, обеспечивающие безопасную, масштабируемую и интегрируемую платформу корпоративных коммуникаций.

Целью выпускной квалификационной работы является разработка прототипа географически распределённого корпоративного мессенджера, обеспечивающего защищённый обмен сообщениями, поддержку секретных чатов и интеграцию с системами управления проектами.

Для достижения поставленной цели необходимо решить следующие задачи:

- Провести анализ предметной области и существующих корпоративных коммуникационных систем.
- Определить функциональные и нефункциональные требования к разрабатываемому решению.
- Спроектировать архитектуру корпоративного мессенджера на основе микросервисного подхода.
- Реализовать основные модули системы, включая сервисы аутентификации, обмена сообщениями и маршрутизации запросов.
- Внедрить механизмы защиты данных и сквозного шифрования сообщений.
- Провести тестирование разработанных компонентов и оценить экономическую эффективность решения.

Научная новизна работы заключается в применении комплексного подхода к построению корпоративной коммуникационной платформы, основанного на сочетании микросервисной архитектуры, асимметричного шифрования и интеграции с системами управления проектами. Практическая значимость исследования заключается в возможности использования полученных результатов при создании защищённых корпоративных мессенджеров и оптимизации коммуникаций в организациях с распределённой структурой.

1 Теоретические основы проектирования корпоративного мессенджера

1.1 Описание организации, являющейся объектом исследования ВКР

ООО «ПроектКом» — средняя IT-компания, специализирующаяся на разработке программных решений в сфере проектного управления и корпоративной коммуникации. Головной офис расположен в Москве, филиалы — в Санкт-Петербурге, Казани и Новосибирске. Количество сотрудников: около 300.

Миссия организации - Создавать надёжные, масштабируемые и безопасные программные решения, способствующие повышению эффективности внутренних бизнес-процессов организаций-клиентов.

Стратегическая цель - стать ведущим отечественным поставщиком решений для корпоративной коммуникации и управления проектами с упором на защищённость и адаптацию к специфике крупных распределённых организаций.

Основная задача организации – разработка и поддержка облачного продукта (SAAS), который бы отвечал следующим основным требованиям:

- Обеспечение быстрой и безопасной передачи сообщений, включая защищенные чаты.
- Возможность интегрировать мессенджер с многими системами управления проектами, такими как Jira, Trello, Yandex tracker.
- Обеспечить соответствие требованиям корпоративной безопасности
- Поддерживать работу в условиях географической распределённости.

Организационная структура компании — матричная, что обеспечивает гибкость, быструю переориентацию команд на новые задачи и эффективное использование ресурсов. Основные блоки:

– Функциональные отделы — постоянные подразделения (разработка, безопасность, DevOps и т.д.)
– Проектные команды — временные группы, сформированные под конкретный проект (например, интеграция с Jira)

Ответственность и полномочия распределены в соответствии с таблицей 1.

Таблица 1 – Распределение ответственности и полномочий

Должность / Отдел	Ответственность	Полномочия
Генеральный директор	Стратегическое управление, утверждение бюджета, контроль выполнения целей	Назначение руководителей отделов, утверждение проектов, внешние связи
Технический директор (CTO)	Техническое развитие продуктов, архитектура решений	Утверждение технологий, назначение архитекторов, руководство техническими стратегиями
Отдел разработки ПО	Разработка клиентской и серверной части мессенджера	Выбор технологий, принятие архитектурных решений, оценка трудозатрат
Отдел информационной безопасности	Анализ рисков, проектирование E2EE, аудит	Оценка уязвимостей, блокировка небезопасных решений, утверждение криптографических протоколов
DevOps-отдел	Обеспечение CI/CD, отказоустойчивости, мониторинга	Настройка инфраструктуры, масштабирование, аварийное восстановление
Команда UX/UI	Разработка интерфейса, логики взаимодействия	Проектирование макетов, проведение UX-исследований
Бизнес-аналитики	Сбор требований, постановка задач	Общение с заказчиком, приоритизация функционала
Менеджер проекта	Планирование, контроль сроков и качества	Распределение задач, управление рисками, коммуникация с заказчиком
Тестировщики	Проверка качества ПО, автоматизация тестирования	Разработка тест-кейсов, запуск CI-тестов

На рисунке 1 отражена структура управления компании, в которой четко выделены основные направления деятельности и их подчинённость.

В верхнем уровне иерархии располагается генеральный директор, сосредоточивающий в себе функции стратегического руководства. Под его началом находятся три ключевых должностных лица: технический директор,

финансовый директор и коммерческий директор. Такая модель управления соответствует классической линейно-функциональной структуре, при которой генеральный директор делегирует ответственность по основным направлениям деятельности компании.

Технический директор курирует наиболее разветвлённый блок — технические подразделения. В его подчинении находятся:

Отдел разработки программного обеспечения, включающий специализированные группы (Frontend, Backend и мобильная разработка), что отражает современный подход к разделению труда по технологическим стекам.

Отдел информационной безопасности, выполняющий функцию защиты информационных активов.

Отдел DevOps и инфраструктуры, обеспечивающий непрерывную интеграцию и доставку программных продуктов, а также эксплуатацию инфраструктурных решений.

UX/UI и аналитики, ориентированные на проектирование пользовательского опыта и анализ требований.

Отдел контроля качества (QA), выполняющий задачи тестирования и обеспечения соответствия программного обеспечения установленным стандартам.

Менеджеры проектов (Project Managers, PM), координирующие взаимодействие команд и управление жизненным циклом проектов.

Финансовый директор возглавляет отдел продаж, что свидетельствует о совмещении финансовых и коммерческих функций на данном уровне управления. Коммерческий директор, в свою очередь, контролирует финансовый отдел, что указывает на возможное перераспределение функций, традиционно закреплённых за финансовым блоком. Такое решение может объясняться спецификой бизнеса компании и ориентацией на коммерческую эффективность.

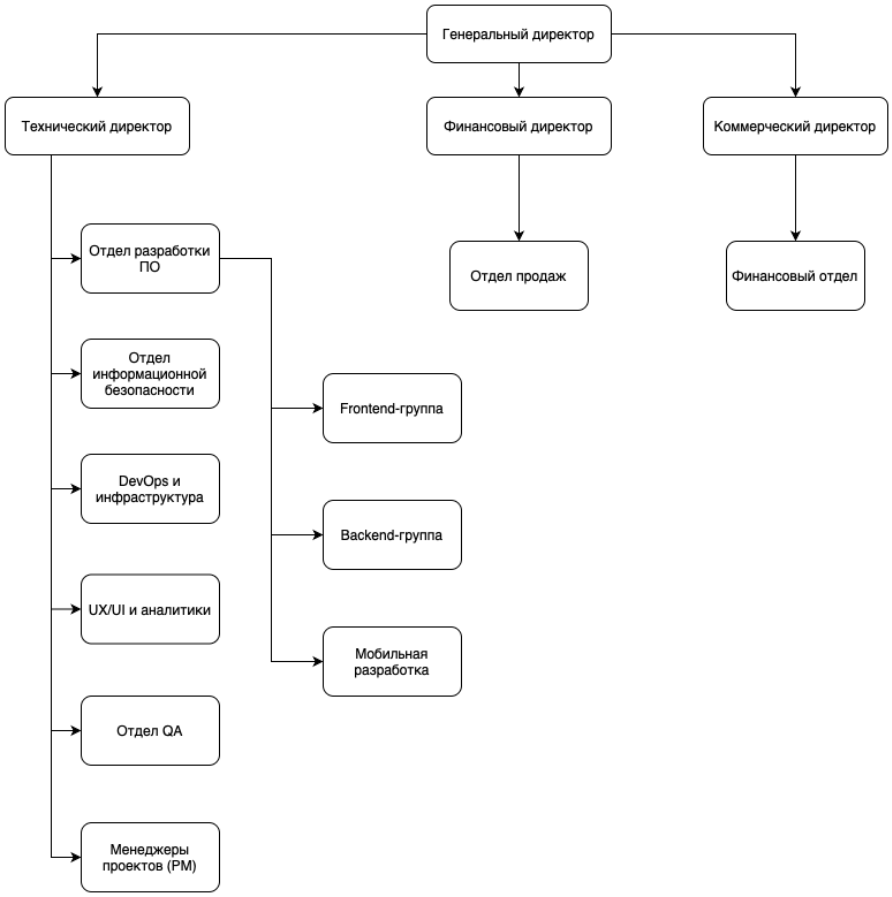

Рисунок 1 – Структурная схема предприятия

1.2 Модель процесса «Постановка, выполнение и контроль задач в проекте» «как есть», «как должно быть» в нотации BPMN.

До внедрения корпоративного мессенджера процесс «Постановка, выполнение и контроль задач в проекте», представленный на рисунке 2, осуществлялся преимущественно посредством стандартных уведомлений по электронной почте, а также с использованием функционала комментариев в системе управления проектами (СУП).

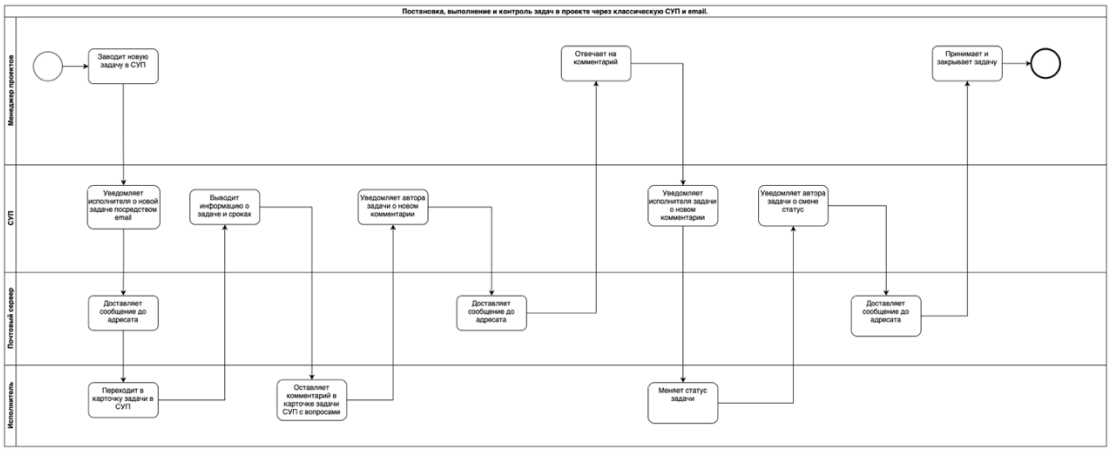

Рисунок 2 – Процесс «Постановка, выполнение и контроль задач в проекте» как есть.

Указанный процесс характеризуется рядом проблем, которые могут быть устранены посредством использования корпоративного мессенджера.

Низкая надежность уведомлений. Электронные письма подвержены риску попадания в спам либо утраты среди большого количества корреспонденции, при этом обновление почтовых клиентов осуществляется с заметной задержкой.

Ограниченность встроенного функционала комментариев в СУП. Данный инструмент не ориентирован на коммуникацию в режиме реального времени, вследствие чего участники проекта зачастую получают информацию о новых комментариях с опозданием — от нескольких часов до суток, особенно при высокой загруженности другими задачами.

Внедрение корпоративного мессенджера в процесс «Постановка, выполнение и контроль задач в проекте» оказывает комплексное позитивное влияние на организацию работы. Прежде всего, оно позволяет значительно сократить среднее время реакции участников: если ранее оно составляло от двух до восьми часов, то с использованием мессенджера этот показатель снижается до диапазона от пяти до тридцати минут. Кроме того, интеграция мессенджера уменьшает количество переключений между различными системами, поскольку исполнителю в большинстве случаев нет

необходимости обращаться напрямую к системе управления проектами — ключевая информация и коммуникации сосредоточены в едином интерфейсе. Дополнительным преимуществом становится рост вовлечённости команды, что достигается за счёт динамичного характера обсуждений и возможности быстрой обратной связи. При этом сохраняется контекст взаимодействия по задачам, так как все сообщения и обсуждения фиксируются в чатах, обеспечивая доступность и прозрачность коммуникаций. Обновленный процесс представлен на рисунке 3.

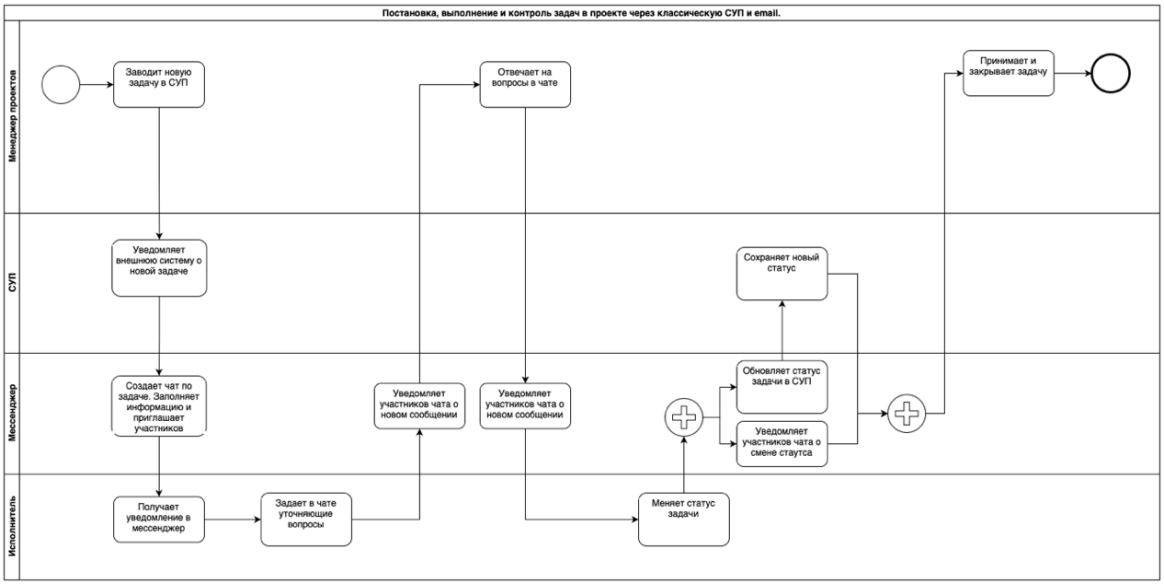

Рисунок 3 – Процесс «Постановка, выполнение и контроль задач в проекте» как будет.

Рисунок 3 иллюстрирует обновлённый процесс «Постановка, выполнение и контроль задач в проекте» после внедрения корпоративного мессенджера с интеграцией в систему управления проектами. В представленной модели отражено перераспределение коммуникационных потоков, переход от асинхронного взаимодействия по электронной почте к работе в режиме реального времени, а также устранение выявленных ранее узких мест.

Ключевым изменением в процессе является появление единого коммуникационного канала — чатов, автоматически создаваемых и обновляемых в соответствии с состоянием задач в СУП. Менеджер проекта формирует новую задачу, после чего корпоративный мессенджер инициирует создание тематического чата, приглашает участников и отправляет первичное уведомление. Исполнители получают мгновенные сообщения о назначении задачи, уточняют детали в чате и при необходимости задают вопросы, что уменьшает время реакции и повышает прозрачность взаимодействия [12].

Система уведомлений работает в обоих направлениях: изменения статуса задачи в СУП автоматически отправляются в чат, а действия участников (например, смена статуса или добавление комментария) синхронизируются с карточкой задачи. Это обеспечивает целостность данных и исключает дублирование действий. Статусы задач обновляются в режиме реального времени, а все ключевые события сопровождаются соответствующими уведомлениями мессенджера.

Таким образом, представленная схема демонстрирует целостную интеграцию коммуникационной платформы и проектного контура: информация концентрируется в одном интерфейсе, минимизируется задержка между действиями участников, повышается контролируемость и предсказуемость хода выполнения задач.

1.3 Анализ лучших практик в предметной области и обоснование выбора решения по оптимизации / реинжинирингу

Анализ современных практик управления проектами в цифровых командах показывает, что их деятельность характеризуется распределённой структурой, активным использованием систем управления задачами и высокой потребностью в оперативной, прозрачной и безопасной коммуникации. В подобных условиях нередко возникают трудности, связанные с фрагментацией каналов взаимодействия, утратой контекста при

обсуждении задач, задержками в реакции на изменения и дублированием действий, а также с отсутствием единого пространства для принятия решений [3].

Наиболее эффективным подходом к решению этих проблем является интеграция коммуникационных инструментов и систем управления проектами. Подобные решения позволяют автоматизировать уведомления, обеспечивать постановку задач непосредственно из чатов и сохранять весь контекст обсуждений внутри задач. Такой подход способствует росту вовлечённости участников и сокращению времени реакции. Существенную роль в оптимизации процессов играют чат-боты и автоматизированные сценарии, которые берут на себя задачи распределения нагрузки, назначения исполнителей и контроля соблюдения сроков. Дополнительным инструментом являются проектные чаты, формируемые для конкретных задач или инициатив, что позволяет концентрировать коммуникацию, оперативно реагировать на инциденты и фиксировать все ключевые решения.

Таким образом, внедрение корпоративного мессенджера, глубоко интегрированного с системой управления проектами, представляет собой не просто модификацию коммуникационного процесса, а полноценную оптимизацию производственного цикла. Это решение повышает скорость выполнения задач, обеспечивает прозрачность статусов и в конечном счёте способствует росту удовлетворённости участников проектной деятельности.

1.4 Бизнес-цели и требования ИТ-проекта

1.4.1 Бизнес-цели проекта

В таблице 2 представлены бизнес-цели проекта по методу SMART, позволяющему сформулировать измеримые и достижимые ориентиры. Каждой цели присвоен код (BC-x), установлен приоритет по MoSCoW**,** а также обозначены ключевые KPI, горизонты достижения и ответственные стейкхолдеры.

Таблица 2 – бизнес-цели проекта

Код	Формулировка бизнес-цели (SMART)	KPI / Метрика	Приоритет	Горизонт	Метрики успеха	Ответственные стейкхолдеры
BC-1	Сократить среднее время реакции на сообщения и задачи с 2–8 часов до ≤30 минут в течение 6 мес.	MTTR ≤ 30 мин	Must	6 мес	MTTR, NPS	Руководитель проекта, тимлиды
BC-2	Обеспечить централизованную цифровую платформу для управления проектами и коммуникаций, так чтобы ≥80% задач и обсуждений фиксировались в мессенджере в течение 4 мес.	≥80% задач и обсуждений в системе	Must	4 мес	Количество задач, активность пользователей	PMO, руководители подразделений
BC-3	Снизить лицензионные затраты на SaaS-сервисы минимум на 30% в течение 12 мес.	Экономия ≥30%	Should	12 мес	ROI	Финансовый директор
BC-4	Повысить прозрачность статусов задач и обсуждений, увеличив информированность команд на ≥50% по результатам опросов за 6 мес.	Увеличение информированности ≥50%	Should	6 мес	Опросы, NPS	HR-департамент, тимлиды

Продолжение таблицы 2

BC-5	Обеспечить 100% соответствие требованиям ИБ при хранении и обработке данных в закрытом контуре, начиная с 3 мес эксплуатации.	100% соответствие политике	Must	3 мес	Аудит, 0 инцидентов	CISO, служба ИБ
BC-6	Обеспечить доступность мессенджера на всех популярных платформах и в офлайн-режиме в течение 4 мес.	Поддержка платформ, офлайн-доступ	Must	4 мес	SLA, % доступности	Технический директор, DevOps
BC-7	Обеспечить устойчивую работу мессенджера в георасределённой инфраструктуре с синхронизацией данных между узлами, с доступностью ≥99% в течение 6 мес.	SLA ≥ 99%, % доступности, время синхронизации ≤5 сек	Must	6 мес	SLA, метрики задержек репликации, NPS	Технический директор, DevOps, архитектор инфраструктуры

Анализ бизнес-целей, представленных в таблице 2, позволяет выделить ключевые ориентиры проекта и определить ожидаемый эффект от внедрения корпоративного мессенджера. Цели сформулированы по методу SMART и охватывают все критически важные аспекты функционирования будущей системы — скорость коммуникаций, безопасность, интеграцию с проектной деятельностью, экономическую эффективность и устойчивость работы в георасределённой инфраструктуре [4].

Приоритеты, установленные по методу MoSCoW, демонстрируют, что основу проекта составляют цели уровня Must, связанные с сокращением

времени реакции (BC-1), централизацией коммуникаций (BC-2), обеспечением требований информационной безопасности (BC-5), кроссплатформенной доступностью (BC-6) и поддержкой распределённой архитектуры (BC-7). Эти цели являются фундаментальными для построения минимально жизнеспособного продукта и определяют необходимые технические и организационные решения.

Цели уровня Should (BC-3, BC-4) направлены на повышение эффективности и улучшение пользовательского опыта: снижение лицензионных затрат и рост информированности команд. Они усиливают ценность проекта, обеспечивая дополнительные преимущества при эксплуатации системы [2].

Таким образом, таблица 2 отражает комплексный характер проекта, где бизнес-цели формируют основу требований, определяют направления оптимизации процессов и служат базой для последующего формирования бизнес-требований и функциональных характеристик системы. Формализованная структура целей обеспечивает прозрачность планирования, контролируемость результатов и возможность объективной оценки успешности внедрения.

1.4.2 Основные бизнес-требования

В таблице 3, на основании целей, определены бизнес-требования (BR-x), которые сформулированы в проверяемом виде («Система должна позволять…»). Каждое требование связано с бизнес-целью, имеет критерий измеримости (SLA, KPI), категорию FURPS+ и приоритет.

Таблица 3 – Основные бизнес-требования

ID	Описание бизнес-требования	Критерий (KPI/SLA)	Связь с целью (BC-x)	Категория FURPS+	Приоритет
BR-1	Система должна обеспечивать защищённую передачу сообщений	100% трафика шифруется (TLS, E2EE)	BC-5	Reliability	Must
BR-2	Система должна обеспечивать интеграцию с системой управления проектами	Интеграция Jira/Redmine, задержка ≤ 1 мин	BC-2	Functionality	Must
BR-3	Система должна позволять создавать задачи из сообщений	Создание задачи ≤ 2 клика	BC-2	Functionality	Should
BR-4	Система должна централизовать коммуникации	≥80% коммуникаций через мессенджер	BC-1	Usability	Must
BR-5	Система должна разграничивать доступ к чатам и проектам	100% политик доступа выполняются	BC-5	Supportability	Must
BR-6	Система должна работать на всех популярных устройствах	Поддержка Android, iOS, Windows, macOS, Linux	BC-6	Usability	Must
BR-7	Система должна поддерживать офлайн-режим	Сообщения синхронизируются при восстановлении	BC-6	Performance	Should
BR-8	Система должна предоставлять API для внешней интеграции	SLA ≥ 99.5%, документированное REST API	BC-2	Supportability	Must
BR-9	Система должна поддерживать локальное развёртывание в закрытом контуре	Self-hosted, без подключения к интернету	BC-5	Constraints	Must
BR-10	Система должна снижать затраты на лицензии	Экономия ≥ 30% от SaaS	BC-3	Cost	Should

Продолжение таблицы 3

BR-11	Система должна позволять интеграцию с CI/CD процессами	Jenkins, GitLab CI, Webhooks, токены	BC-2	Supportability	Could
BR-12	Система должна обеспечивать георасределённое хранение и синхронизацию данных между узлами	SLA ≥ 99%, задержка репликации ≤5 сек	BC-7	Reliability / Performance	Must

1.4.3 Функциональные и нефункциональные требования

Для систематизации требований к разрабатываемой системе использована методология FURPS+, включающая функциональные (Functional) и нефункциональные (Usability, Reliability, Performance, Supportability) характеристики, а также расширение «+» для учёта дополнительных аспектов (например, ограничений и интерфейсов) [1].

Для определения приоритетов требований применена методика MoSCoW, позволяющая разделить их на четыре группы:

Must (обязательно) — критические требования, определяющие минимально жизнеспособный продукт (MVP);

Should (желательно) — значимые, но не критические для первой версии;

Could (может быть) — опциональные, повышающие ценность, но не влияющие на базовую функциональность;

Won’t (не входит в релиз) — требования, которые осознанно откладываются на будущее.

Каждое требование снабжено краткими критериями приёмки, позволяющими однозначно проверить его выполнение и сгруппированы в следующие блоки:

- Коммуникация
- Безопасность
- Интеграция с PM-системами
- UX и доступность

– Поиск

– Автоматизация

Функциональные и нефункциональные требования представлены в таблице 4.

Таблица 4 – Функциональные и нефункциональные требования по FURPS+

ID	Описание требования	Категория (FURPS+)	Приоритет (MoSCoW)	Критерий приёмки (Дано / Когда / Тогда)
Коммуникация				
FR-1	Система должна обеспечивать чаты один-на-один	Functional (F)	Must	Дано: два авторизованных пользователя. Когда: один отправляет сообщение другому. Тогда: получатель видит сообщение.
FR-2	Система должна поддерживать групповые чаты	Functional (F)	Must	Дано: создан групповой чат с тремя участниками. Когда: один отправляет сообщение. Тогда: оно доставляется всем.
FR-3	Система должна обеспечивать обмен файлами	Functional (F)	Should	Дано: пользователь в чате. Когда: он загружает файл. Тогда: файл доступен другим участникам.
FR-4	Система должна поддерживать упоминания пользователей	Functional (F)	Could	Дано: пользователь в групповом чате. Когда: другой упоминает его. Тогда: он получает уведомление.
Безопасность				
FR-5	Система должна использовать TLS для передачи данных	Reliability (R)	Must	Дано: пользователь подключается. Когда: устанавливается соединение. Тогда: трафик идёт по TLS.
FR-6	Система должна поддерживать сквозное шифрование (E2EE)	Reliability (R)	Should	Дано: два пользователя общаются. Когда: сообщение отправлено. Тогда: его расшифровывает только адресат.
FR-7	Система должна предоставлять ролевую модель доступа	Supportability (S)	Should	Дано: администратор назначает роль. Когда: пользователь выполняет действие. Тогда: система разрешает или запрещает его.

Продолжение таблицы 4

Интеграция с PM-системами				
FR-8	Система должна автоматически уведомлять о действиях в PM-системе	Functional (F)	Could	Дано: чат связан с PM. Когда: создаётся задача. Тогда: в чате появляется уведомление.
FR-9	Система должна позволять создавать задачи из сообщений	Functional (F)	Should	Дано: пользователь выделяет сообщение. Когда: выбирает «Создать задачу». Тогда: задача появляется в PM-системе.
UX и доступность				
FR-10	Система должна поддерживать десктопные и мобильные платформы	Usability (U)	Must	Дано: пользователь устанавливает приложение. Когда: он авторизуется. Тогда: функциональность доступна.
FR-11	Система должна работать офлайн с последующей синхронизацией	Usability (U)	Could	Дано: пользователь офлайн. Когда: он отправляет сообщение. Тогда: оно синхронизируется при подключении.
Поиск				
FR-12	Система должна обеспечивать полнотекстовый поиск по сообщениям	Functional (F)	Should	Дано: пользователь вводит ключевое слово. Когда: выполняется поиск. Тогда: система возвращает все совпадения.
Автоматизация				
FR-13	Система должна поддерживать подключение ботов и автоматических ассистентов	Functional (F)	Could	Дано: бот подключён. Когда: пользователь отправляет команду. Тогда: бот отвечает в чате.
FR-14	Система должна предоставлять API для внешней интеграции	Supportability (S)	Should	Дано: приложение обращается к API. Когда: запрос корректен. Тогда: система возвращает данные.
Георазпределённость				
FR-15	Система должна обеспечивать георазпределённость с синхронизацией данных между узлами	Reliability (R), Performance (P)	Must	Дано: развернуто два узла системы в разных дата-центрах. Когда: пользователь отправляет сообщение. Тогда: сообщение сохраняется и становится доступным на обоих узлах независимо от точки подключения.

Анализ требований, представленных в таблице 4, показывает, что сформированная спецификация охватывает все ключевые аспекты функционирования корпоративного мессенджера как интегрированной коммуникационной и проектной платформы. Структурирование требований по модели FURPS+ позволяет последовательно выделить функциональные элементы системы (обмен сообщениями, групповые чаты, работа с задачами, интеграция с PM-системами) и определить нефункциональные характеристики, критически важные для корпоративной среды: безопасность, надёжность, производительность, удобство использования и поддерживаемость [5].

Приоритизация по MoSCoW демонстрирует, что в первую очередь должны быть реализованы функции уровня Must, обеспечивающие базовую работоспособность и соответствие бизнес-целям: защищённая передача данных (TLS), чаты один-на-один и групповые чаты, кроссплатформенность, георасределённая синхронизация и строгая модель разграничения доступа. Требования уровня Should усиливают продуктивность и удобство системы — это E2EE, работа с файлами, создание задач из сообщений, полнотекстовый поиск, поддержка API. Требования уровня Could добавляют гибкость и расширяемость системы, включая поддержку ботов и уведомления от PM-систем.

Таким образом, таблица 4 формирует целостный набор требований, который напрямую отражает стратегические цели проекта и задаёт чёткие критерии для проектирования архитектуры, разработки MVP и последующей валидации системы. Сформированная спецификация обеспечивает прослеживаемость требований, логичное распределение приоритетов и служит надёжной основой для последующих этапов разработки.

1.5 Анализ существующих разработок

Для анализа были отобраны три наиболее релевантных решения: Slack в связке с Jira, Microsoft Teams с Planner, а также Rocket.Chat совместно с Redmine. Данные варианты охватывают как модели SaaS, так и Open Source-подходы, что отражает наиболее распространённые практики, применяемые в корпоративной и проектной среде.

Оценка проводилась по ряду критериев, включающих возможности коммуникации, функционал управления задачами, степень интеграции с внешними системами и API, уровень безопасности, показатели пользовательского опыта, а также стоимость внедрения и поддержки и особенности лицензирования. В соответствии с таблицей 5, сравнительный анализ показал, что решения на основе SaaS (Slack + Jira, Microsoft Teams + Planner) обеспечивают высокий уровень интеграции и удобства использования, однако характеризуются ограниченным контролем над данными и зависимостью от платной подписки. В то же время Open Source-вариант (Rocket.Chat + Redmine) предоставляет полный контроль над системой и возможность локального развёртывания, но требует значительных ресурсов DevOps и уступает в аспектах пользовательского опыта.

Таблица 5 – Сравнительный анализ наиболее релевантных решений

Критерий	Slack + Jira	MS Teams + Planner	Rocket.Chat + Redmine
Коммуникация	5	4	4
Управление задачами	4	5	3
Интеграции	5	5	3
Безопасность	3	4	5
Георазпределённость	5	5	5
UX	5	4	3
Стоимость	2	3	5
Лицензирование	SaaS	SaaS	Open Source
Итоговый балл /40	29	30	28

Таким образом, ни одно из рассмотренных решений не является полностью удовлетворяющим требованиям организации. В этой связи

оправданной представляется разработка собственной системы, что позволит избежать зависимости от конкретного вендора, внедрить кастомизированную модель безопасности, сократить среднее время восстановления работоспособности (MTTR) на 30–40 % благодаря более глубокой интеграции мессенджера и системы управления проектами, а также обеспечить соответствие требованиям критической инфраструктуры [6].

1.7 Минимальный жизнеспособный продукт (MVP)

Минимально жизнеспособный продукт корпоративного мессенджера был сформирован на основе анализа функциональных и нефункциональных бизнес-требований, а также их привязки к бизнес-целям проекта. Отбор требований осуществлялся с учётом метода приоритизации MoSCoW, где в состав MVP вошли все требования уровня Must и часть требований уровня Should, критически необходимых для практической ценности продукта на раннем этапе внедрения [11].

Ключевым критерием включения требования в MVP является его способность обеспечивать достижение бизнес-целей, связанных с сокращением времени реакции на сообщения (BC-1), централизованной фиксацией задач и обсуждений (BC-2), безопасностью и соответствием требованиям ИБ (BC-5), кроссплатформенностью и доступностью в офлайн-режиме (BC-6), а также устойчивостью работы системы в георасределённой инфраструктуре (BC-7).

На основании анализа требований был сформирован целостный набор функций MVP, охватывающий базовые коммуникационные возможности, безопасность, кроссплатформенность и поддержку распределённой архитектуры. В основу минимально жизнеспособного продукта заложены ключевые механизмы обмена сообщениями, включающие личные и групповые чаты, возможность передачи файлов и создание задач

непосредственно из сообщений, что обеспечивает выполнение бизнес-целей по ускорению взаимодействия и централизации проектной коммуникации.

Блок требований по безопасности определяет критическую часть MVP и включает использование TLS и механизмов сквозного шифрования, реализацию ролевой модели доступа и возможность локального развёртывания системы в закрытом контуре. Эти функции обеспечивают соблюдение корпоративных требований информационной безопасности и защиту данных на всех этапах обработки [7].

Функциональность, обеспечивающая доступность и удобство работы на разных устройствах, также входит в состав MVP. Поддержка настольных и мобильных платформ, а также возможность работы в офлайн-режиме с последующей синхронизацией позволяют пользователям взаимодействовать с системой независимо от условий подключения.

Отдельным приоритетным направлением является обеспечение георазпределённости и надёжности. В MVP включены механизмы распределённого хранения и синхронизации данных между узлами, поддержка высокой доступности и интеграция с внешними системами управления проектами. Дополнительная ценность достигается за счёт предоставления API для внешних сервисов и взаимодействия с PM-платформами, что обеспечивает консистентность проектной информации и упрощает интеграцию в существующую инфраструктуру организации.

Таким образом, сформированный набор функций MVP представляет собой сбалансированный минимум, позволяющий проверить ключевые гипотезы проекта, обеспечить практическую ценность системы и заложить основу для её дальнейшего расширения [10].

Включение вышеописанных функций обосновано как балансом между минимальной реализуемостью продукта и его практической ценностью**,** так и соответствием ключевым бизнес-целям проекта. Все функции уровня Must обеспечивают критически необходимый функционал для начальной

эксплуатации продукта, а часть функций уровня Should включена для проверки востребованности дополнительных возможностей у пользователей.

Таким образом, сформированный набор функций MVP позволяет проверить основные гипотезы проекта, получить обратную связь от пользователей и создать базу для дальнейшего развития корпоративного мессенджера с учётом распределённой архитектуры, безопасности и интеграции с бизнес-процессами.

1.8 Трассировочная матрица

Для обеспечения прослеживаемости целей до функций и тест-кейсов составлена трассировочная матрица «BC ↔ BR ↔ FR ↔ Тест-кейс». Она позволяет показать, как стратегические ориентиры бизнеса трансформируются в требования к системе и проверяются на этапе тестирования. Матрица представлена в таблице 6.

Таблица 6 – Трассировочная матрица

Бизнес-цель (BC)	Бизнес-требование (BR)	Функциональное требование (FR)	Тест-кейс
BC-1	BR-4	FR-1, FR-2	TC-1. Отправка сообщения один-на-один; TC-2. Групповой чат ≥3 участников
BC-2	BR-2, BR-3, BR-8, BR-11	FR-8, FR-9, FR-14, FR-3	TC-3. Создание задачи из сообщения; TC-4. Уведомление о событии из PM-системы; TC-5. Отправка сообщения с вложенным файлом
BC-3	BR-10	FR-14	TC-6. Проверка снижения затрат по отчётам ROI
BC-4	BR-4	FR-12	TC-7. Полнотекстовый поиск сообщений; TC-8. Опрос об информированности команды
BC-5	BR-1, BR-5, BR-9	FR-5, FR-6, FR-7	TC-9. Проверка TLS-шифрования; TC-10. Проверка E2EE; TC-11. Локальная установка
BC-6	BR-6, BR-7	FR-10, FR-11	TC-12. Авторизация на разных устройствах; TC-13. Отправка сообщения офлайн

Продолжение таблицы 6

BC-7	BR-12	FR-15	TC-14. Проверка георасределённого хранения и синхронизации данных между узлами; TC-15. Доступность сообщений при подключении к разным узлам

Сравнение криптографических протоколов, представленное в таблице 8, демонстрирует, что различные подходы к реализации сквозного шифрования существенно отличаются по уровню безопасности, поддержке групповых коммуникаций, удобству интеграции и сложности внедрения. Анализ показывает, что наиболее сбалансированным по сочетанию свойств является протокол семейства Signal (X3DH + Double Ratchet + SenderKey), обеспечивающий как прямую, так и обратную криптографическую секретность, безопасное управление ключами при входе и выходе участников и надёжную защиту в групповых чатах. Однако высокая сложность реализации и отсутствие стабильных библиотек на Go делают его менее подходящим для быстрых прототипов.

Альтернативные решения — NaCl/box и Olm/Megolm — предлагают другие компромиссы. NaCl/box отличается простотой и высокой производительностью, но не обеспечивает forward secrecy и безопасного изменения состава участников, что снижает его применимость в корпоративной среде с повышенными требованиями к защите данных. Протоколы Matrix (Olm/Megolm) обеспечивают баланс между безопасностью и поддержкой истории сообщений, однако требуют более сложной инфраструктуры и нативно ориентированы на экосистему Matrix.

Обобщая результаты анализа, можно заключить, что выбор гибридной схемы на основе RSA-OAEP и AES-GCM является оптимальным для разрабатываемого прототипа корпоративного мессенджера.

2 Проектирование и реализация корпоративного мессенджера

2.1 Логическое моделирование автоматизированной системы «Корпоративный мессенджер»

2.1.1 Логическая модель и ее описание

Взаимодействие сущностей автоматизированной системы отображают логические модели. Они строятся исходя из модели «Как должно быть».

На рисунке 4 изображена диаграмма вариантов использования будущей системы, которая демонстрирует взаимодействие основных акторов с системой и отображает ключевые прецеденты использования.

Сотрудник. Базовый пользователь системы, который осуществляет авторизацию, получает доступ к задачам, чатам и списку сотрудников. Он может инициировать создание групповых чатов, управлять участниками, отправлять сообщения, просматривать историю переписки, а также получать информацию о задачах и чатах. Дополнительно сотрудник имеет возможность изменять статус задачи, что расширяет базовый сценарий взаимодействия [9].

Администратор. Надстройка над функциональностью сотрудника. Он обладает правами управления сотрудниками, что включает в себя добавление и удаление пользователей системы. Данный набор прецедентов расширяет возможности стандартного сотрудника, формируя ролевую модель доступа.

Внешняя система управления проектами (СУП). Интеграционный актор, обеспечивающий взаимодействие мессенджера с внешней PM-системой. Основные сценарии включают синхронизацию задач, а также синхронизацию истории чата с комментариями. Таким образом, достигается сквозная интеграция коммуникационной и проектной деятельности.

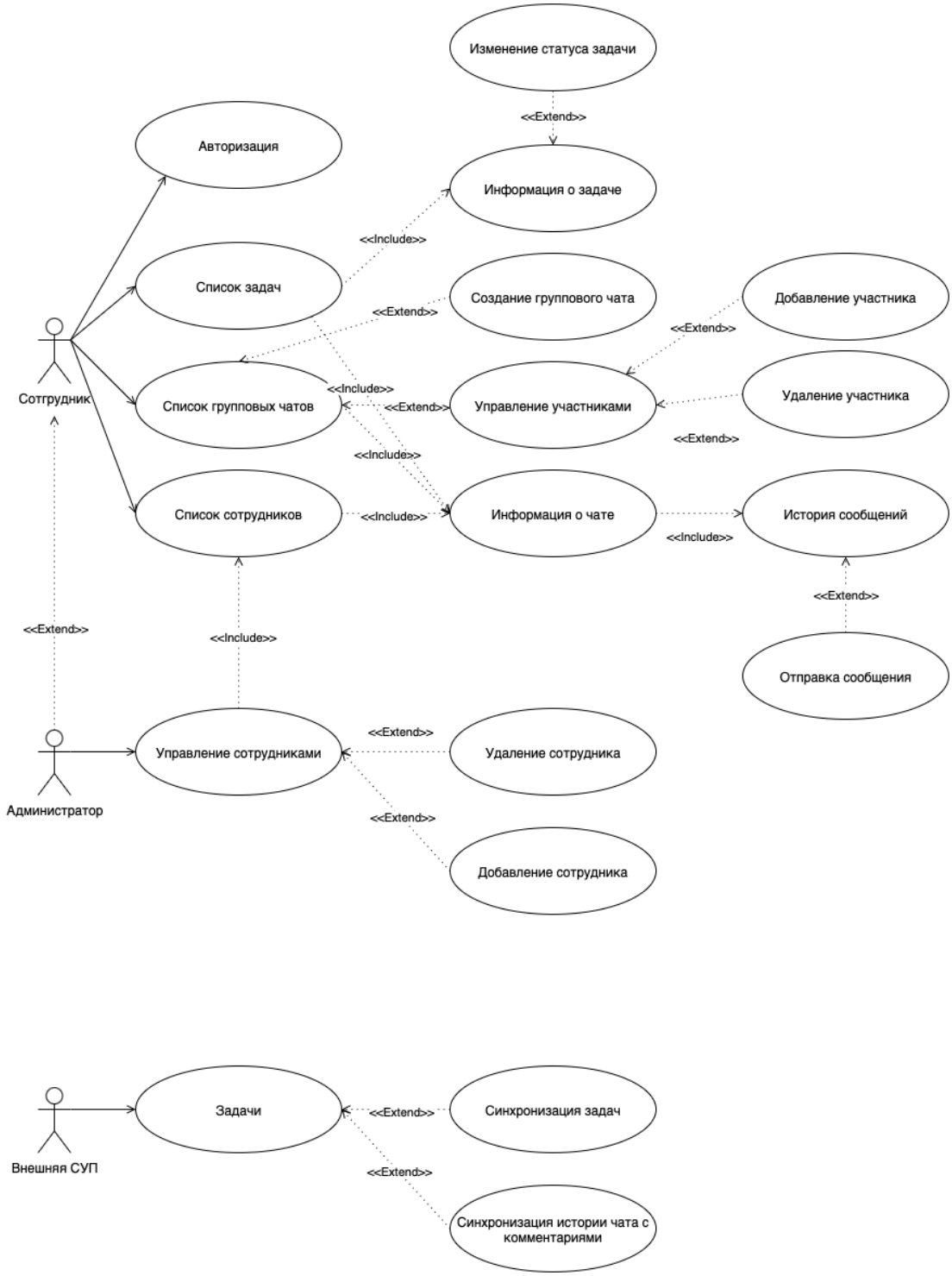

Рисунок 4 – Диаграмма вариантов использования

На диаграмме четко прослеживаются отношения «include» и «extend», которые фиксируют взаимосвязь между базовыми и дополнительными сценариями. Так, например, прецедент «Список задач» включает получение

информации о задаче, а «Создание группового чата» расширяется сценариями добавления и удаления участников. Аналогично, работа с историей сообщений расширяет возможности базового обмена сообщениями.

Таким образом, диаграмма использования отражает структурированное представление функциональных требований к системе. Она показывает распределение ролей, разграничение прав доступа и интеграционные сценарии, что служит основой для последующего формирования логической модели и детального проектирования архитектуры системы.

2.1.2 Концептуальная модель данных

На рисунке 5 изображена концептуальная модель данных ИС. В центре находится сущность «Пользователь», которая является ключевым субъектом информационной системы и определяет все основные процессы взаимодействия. Каждый пользователь может состоять в одной или нескольких компаниях, что задаёт организационный контекст коммуникаций. Компания в данном случае выступает интегрирующей сущностью, группирующей пользователей, чаты и связанные с ними сообщения.

Коммуникационное взаимодействие реализуется через сущность «Чат», которая фиксирует контекст обмена сообщениями. Принадлежность пользователей к чатам выражена посредством сущности «Участник чата», обеспечивающей реализацию связи типа «многие-ко-многим» между пользователями и чатами. Таким образом, достигается гибкость модели, позволяющая одному пользователю участвовать одновременно в нескольких каналах коммуникации. Каждое сообщение принадлежит определённому чату и связывается с конкретным пользователем-отправителем, что отражает принцип прослеживаемости и позволяет реализовать управление авторством сообщений.

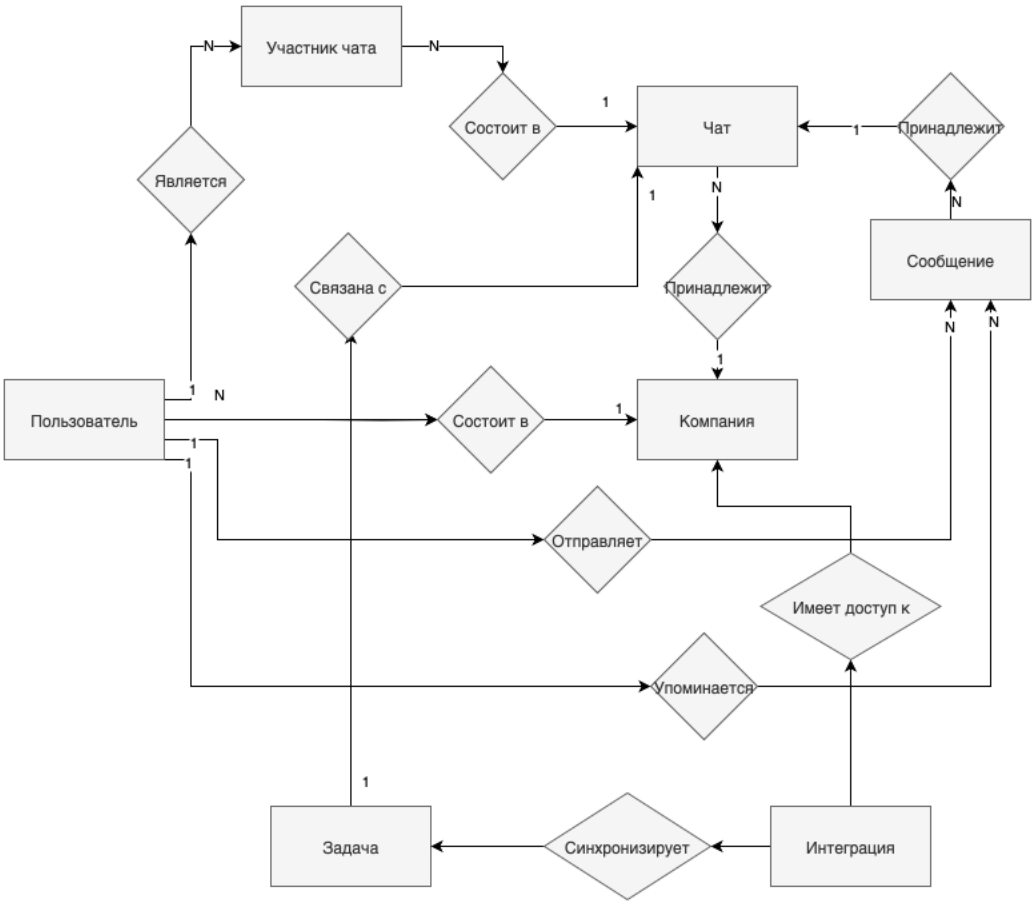

Рисунок 5 – ER диаграмма сущность-связь

Важным аспектом является связь сообщений с задачами. Данное отношение обеспечивает возможность перехода от коммуникационного взаимодействия к формированию проектных артефактов, что является критически значимым для корпоративного мессенджера, интегрированного в среду управления проектами. Сущность «Задача» в модели не существует изолированно: она связана с сообщениями и может синхронизироваться через сущность «Интеграция» с внешними информационными системами, такими как Jira или Redmine. Таким образом, обеспечивается целостность проектной информации и исключается дублирование данных между различными цифровыми платформами [8].

Модель также предусматривает механизм упоминаний, который фиксируется в связях между пользователем и сообщением. Это позволяет

реализовать прямую адресацию в коммуникации и одновременно формализовать процесс уведомлений. Принадлежность чата компании и, в свою очередь, принадлежность сообщений к чатам и задачам формирует многоуровневую структуру, отражающую организационную и функциональную иерархию корпоративного взаимодействия.

В целом, предложенная ER-диаграмма обеспечивает логически выверенное представление предметной области, демонстрирует взаимосвязь коммуникационной и проектной деятельности, а также закладывает основу для проектирования реляционной базы данных. Модель ориентирована на обеспечение целостности информации, минимизацию дублирования данных и поддержку ключевых бизнес-процессов, связанных с коллективным взаимодействием, управлением задачами и интеграцией с внешними сервисами.

2.1.3 Взаимодействие с системой управления проектами.

На рисунках 6-7 представлена диаграмма потоков данных, которая отражает логику функционирования корпоративного мессенджера и его взаимодействие с основными субъектами и внешней системы управления проектами. На контекстном уровне система изображается как единый процесс, внутри которого сосредоточены все ключевые функции. Взаимодействие осуществляется через потоки информации между сотрудником, корпоративным мессенджером и системой управления проектами. Потоки данных фиксируют как исходящие действия пользователя, так и отклики системы, формируя замкнутый цикл информационного обмена.

Сотрудник в рамках этой модели передаёт в систему новые сообщения, и параметры задач, которые поступают в корпоративный мессенджер для дальнейшей обработки. Взаимодействие с системой управления проектами осуществляется через обмен параметрами задач и комментариев. Корпоративный мессенджер передаёт в СУП сведения об изменениях

параметров задач, а также синхронизирует комментарии, обеспечивая консистентность данных в обеих системах.

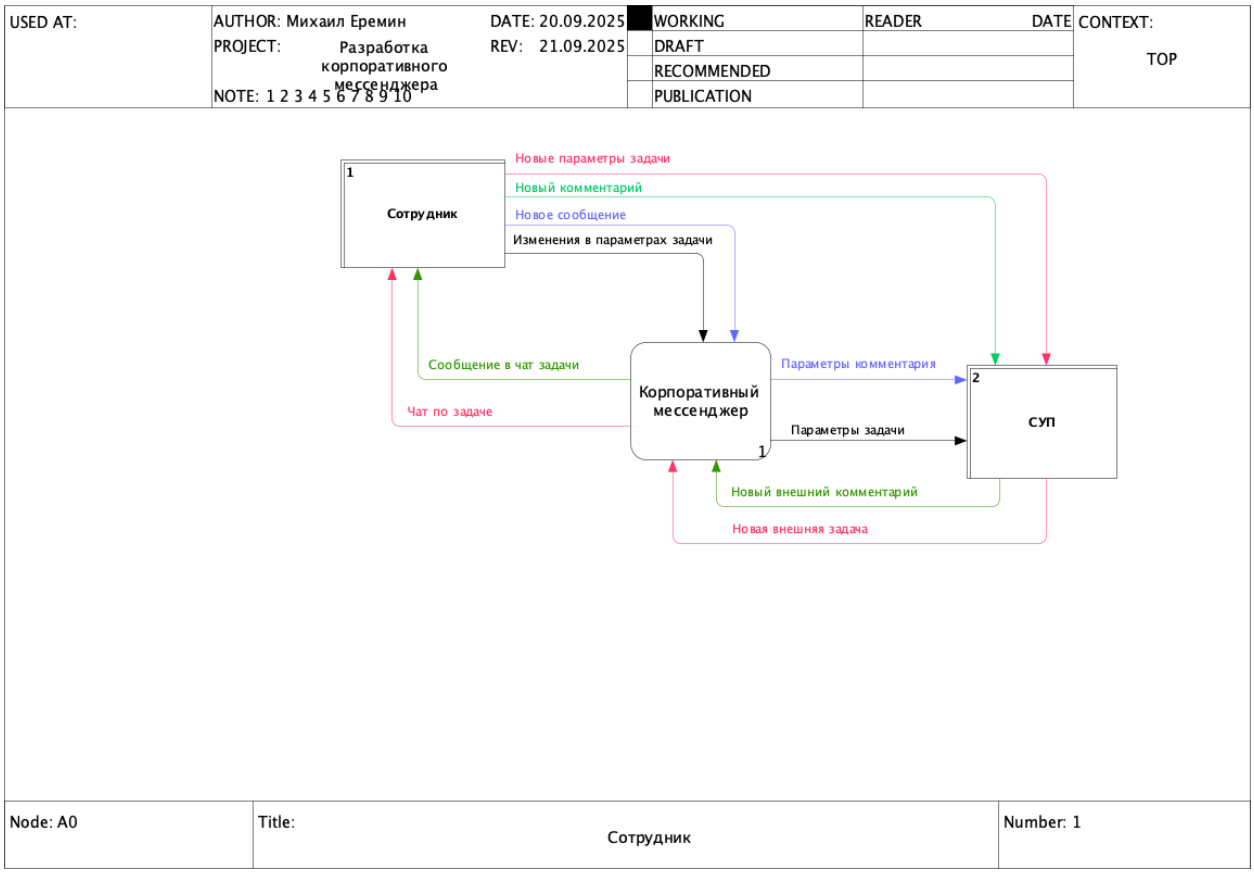

Рисунок 6 – Контекстный уровень DFD диаграммы обмена данными с СУП

На детализированном уровне мессенджер представлен как совокупность взаимосвязанных процессов. Блок обработки и хранения сообщений получает новые данные от пользователя, сохраняет их в специализированном хранилище и передаёт в другие компоненты системы. Обработка и хранение чатов структурирует коммуникацию в рамках задач, взаимодействует с хранилищем чатов и обеспечивает доступ сотрудников к актуальной информации. Управление задачами и проектами отвечает за фиксацию изменений в параметрах задач, координирует обмен информацией с системой управления проектами и направляет обновления обратно в корпоративный мессенджер.

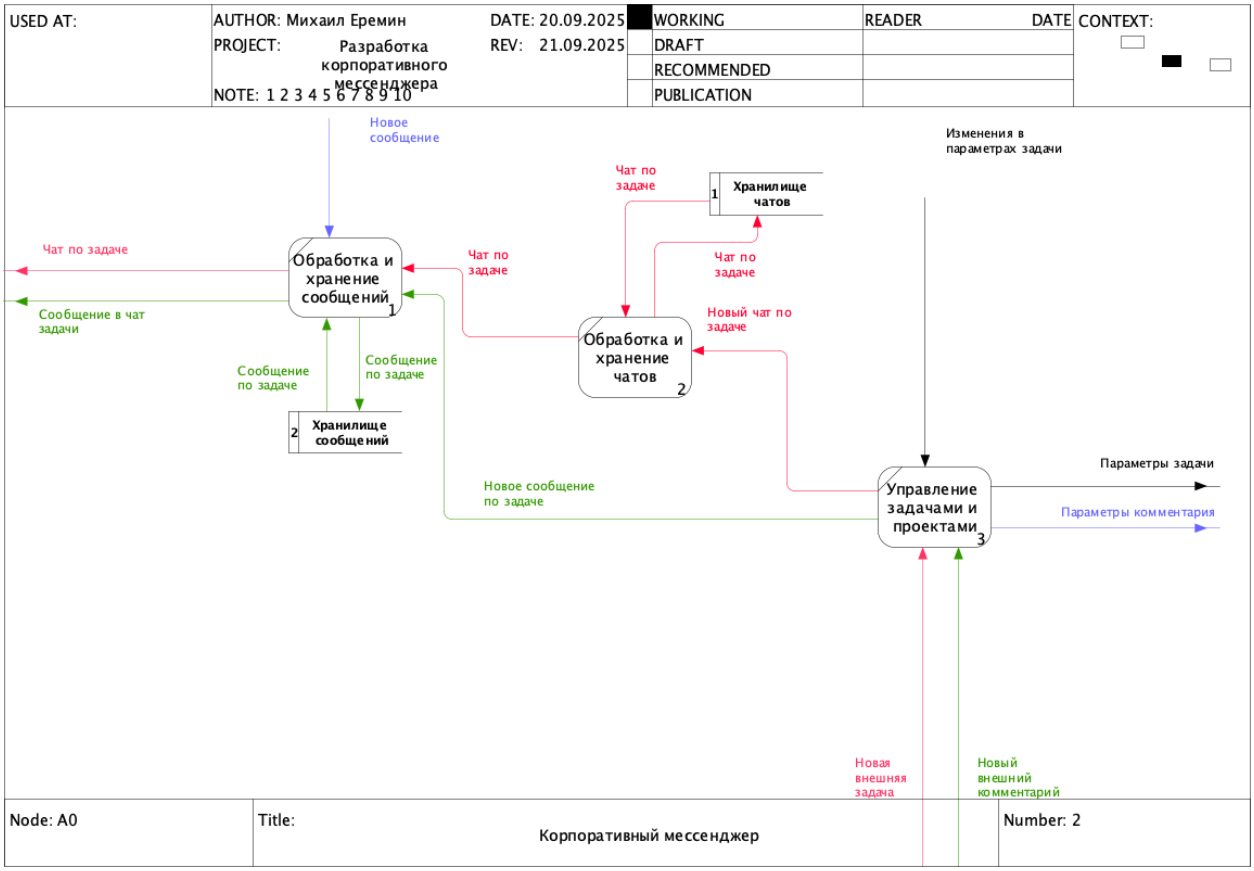

Рисунок 7 – Детализированный уровень DFD диаграммы обмена данными с СУП

Таким образом, диаграмма демонстрирует целостную картину потоков данных в системе, отражая маршрутизацию сообщений и комментариев, хранение информации в специализированных репозиториях и интеграцию с внешними сервисами. Такая модель позволяет выявить логику работы системы на уровне процессов и потоков данных, обеспечивая основу для дальнейшей детализации и перехода к физическому моделированию.

2.1.4 Разработка информационной модели базы данных

Разработанная информационная модель базы данных представлена на рисунке 8.

Таблица *Region* хранит логические географические зоны (регионы), к которым привязываются пользователи и сервисные компоненты. Атрибут id служит первичным ключом и используется как внешний ключ в таблицах User

и Chat. Подобная декомпозиция позволяет реализовать механизмы маршрутизации трафика и географически распределённого хранения данных.

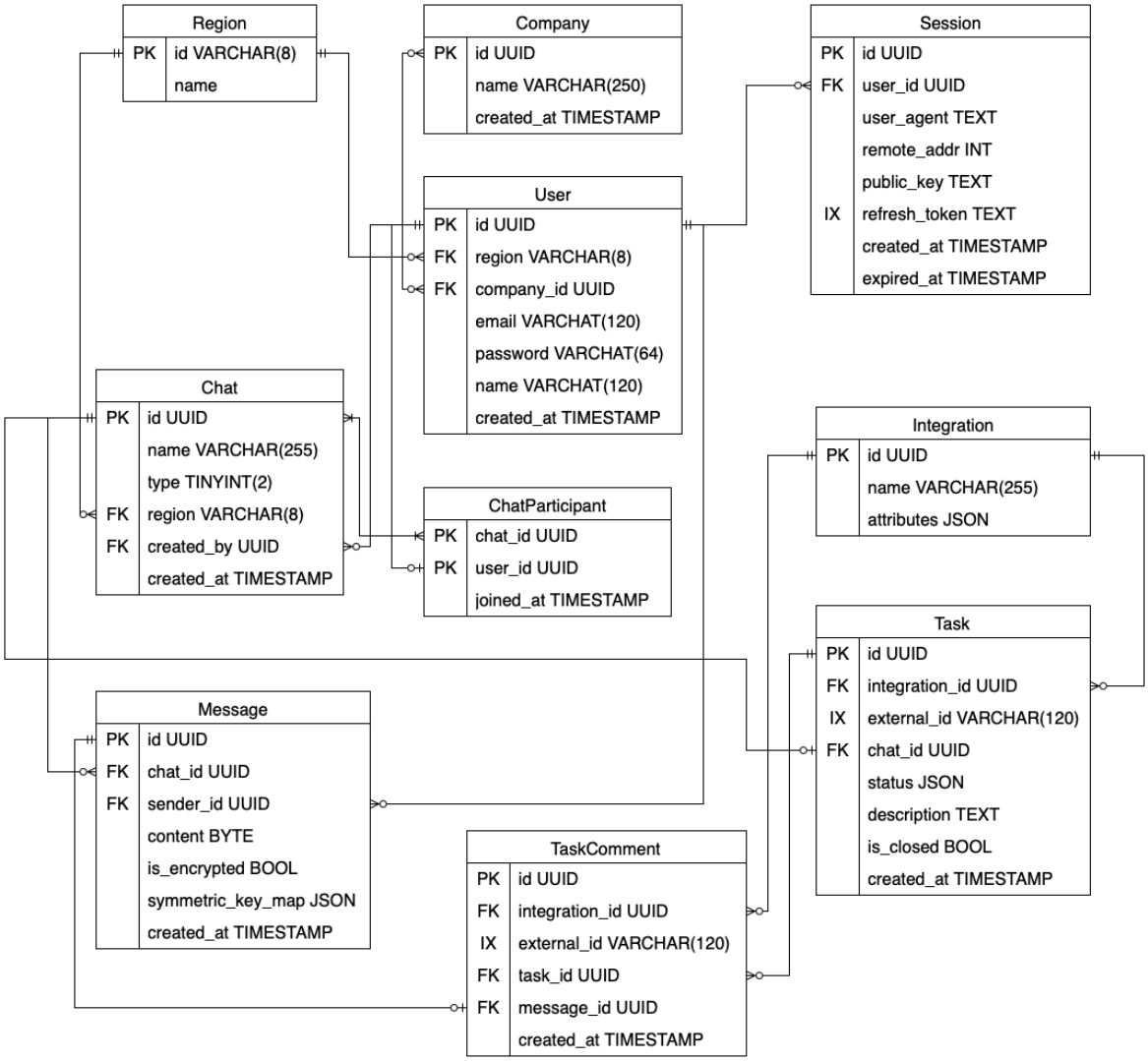

Рисунок 8 - Схема базы данных корпоративного мессенджера

Таблица *Company* хранит организационную структуру заказчика. Каждая компания может иметь множество пользователей, ассоциированных с ней через внешний ключ company_id в таблице User.

Пользователи системы представлены в таблице *User*. Связи с регионами и компаниями реализуются через внешние ключи region и company_id. Атрибуты аутентификации и профиля (e-mail, пароль, имя) хранятся в

соответствующих полях. Каждому пользователю может соответствовать множество сессий и множество участий в чатах.

Таблица *Session* обеспечивает учёт сессий пользователей. Каждая сессия привязана к конкретному пользователю (user_id) и содержит информацию об агенте, IP-адресе, открытом ключе и refresh-токене. Это позволяет реализовать механизмы авторизации, обновления доступа и отслеживания активности.

Чаты, как каналы коммуникации, представлены в таблице *Chat*. Чаты привязываются к региону и создателю (created_by). Атрибут type определяет тип чата: личный, групповой, секретный или связанный с задачей.

Связующая таблица *ChatParticipant* реализует отношение многие-ко-многим между пользователями и чатами. Она хранит информацию о моменте вступления в чат, что может использоваться для контроля доступа к истории сообщений.

Таблица *Message* представляет сообщения, отправленные в чатах. Каждое сообщение содержит ссылку на чат (chat_id), отправителя (sender_id), бинарное содержимое (content) и флаг is_encrypted, указывающий на использование end-to-end шифрования. В случае защищённых чатов применяется поле symmetric_key_map, содержащее симметричные ключи, зашифрованные для каждого получателя.

Таблица *Integration* хранит внешние информационные системы, с которыми осуществляется взаимодействие (например, Jira, Redmine). Поле attributes позволяет гибко хранить параметры подключения или метаданные интеграции.

Таблица *Task* хранит представление задач, синхронизированных из внешних систем. Каждая задача связана с интеграцией и, опционально, с определённым чатом. Состояние задачи, описание и факт завершения хранятся в соответствующих полях. Атрибут external_id обеспечивает возможность двусторонней синхронизации.

Таблица *TaskComment* содержит комментарии к задачам, которые также могут быть ассоциированы с конкретными сообщениями в чатах. Это

позволяет пользователям осуществлять контекстную коммуникацию в рамках задач, поддерживая связь между проектной деятельностью и корпоративной перепиской.

Модель поддерживает расширение — как на уровне интеграции с внешними ИС, так и на уровне расширения бизнес-логики (например, типизация чатов, хранение метаданных сообщений). Включение полей с типами JSON (например, attributes, status) предоставляет гибкость и адаптивность при изменении требований.

2.2 Физическое моделирование автоматизированной системы «Корпоративный мессенджер»

2.2.1 Выбор архитектуры системы

Существуют различные виды архитектуры автоматизированных информационных систем, отличающиеся принципами организации вычислительных ресурсов и распределения функций. Файл-серверная архитектура предполагает использование одного компьютера, на котором сосредоточены операции обработки данных и обеспечения диалога с пользователем. Клиент-серверная модель основана на разделении компонентов приложения: сервер баз данных функционирует отдельно от клиентских рабочих станций, что позволяет повысить производительность и устойчивость системы. Многоуровневая архитектура включает три взаимосвязанных слоя: клиентские приложения, выполняющие функции интерфейса; сервер приложений, содержащий прикладную логику; и удалённый файловый сервер, обеспечивающий хранение данных. Ещё одним типом является архитектура интернет/интранет-систем, применяемая в веб-приложениях и опирающаяся на использование браузера как клиентской среды [13].

На рисунке 9 представлена система, построенная по принципам многоуровневой архитектуры (Three-tier architecture) с

разделением на клиентский уровень, сервер приложений (Backend), базу данных, шину событий, интеграционный уровень и слой безопасности.

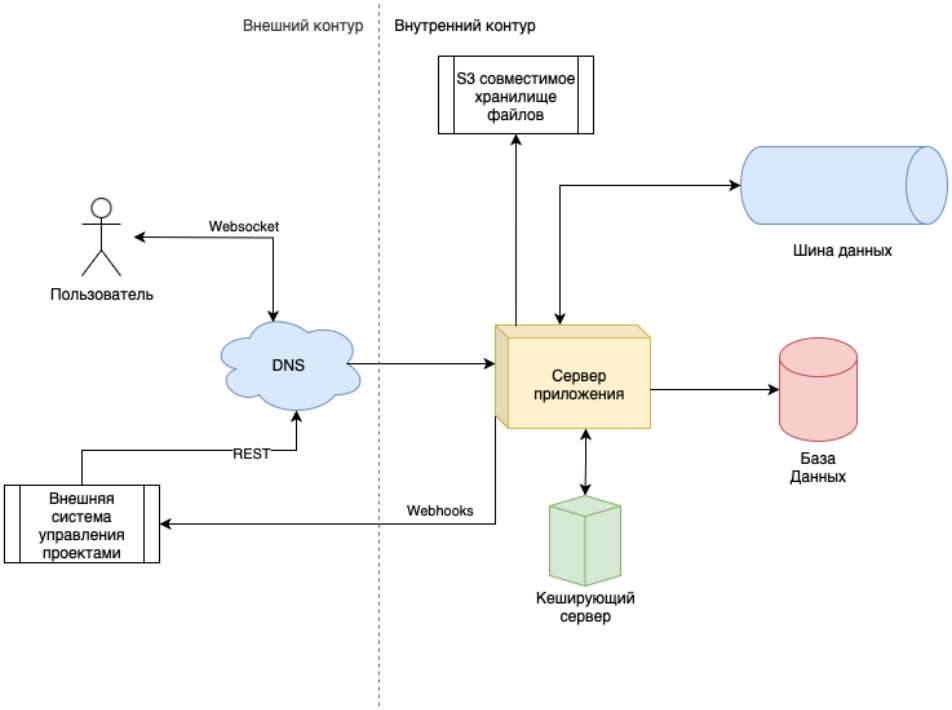

Рисунок 9 – Серверная архитектура корпоративного мессенджера

На клиентском уровне реализуется web-клиент на React и мобильный клиент на React Native. Приложения устанавливаю прямое соединение с сервером приложений используя протокол Websocket. Часть запросов может использовать протокол REST.

Сервер приложений (Backend) отвечает за удержание постоянных соединений с клиентскими приложениями, обработку команд и доставку изменений, происходящих в системе в режиме реального времени.

Сервер приложений написан на языке Golang и имеет микросервисную архитектуру, о которой более подробно будет описано в следующем разделе. Для написания сервера выбран язык Golang, который дает ему высокую производительность, сетевую масштабируемость и простоту деплоя. Для оптимизации трудозатрат и повешении качества приложения используются

такие популярные зависимости, как Echo (лёгкий HTTP-фреймворк), gorilla/websoket (для доставки сообщений в режиме реального времени), gRPC (протокол внутреннего межсервисного взаимодействия).

В качестве основной реляционной системы управления базами данных используется PostgreSQL, обладающая надёжной транзакционной моделью и поддержкой сложных запросов. Для реализации кэширования и хранения пользовательских сессий применяется Redis, обеспечивающий высокую скорость доступа к данным. Хранение вложений и других файлов реализовано посредством распределённого файлового хранилища, такого как MinIO или совместимого с Amazon S3, что позволяет масштабировать объём хранимой информации и интегрироваться с облачной инфраструктурой [14].

Для обеспечения взаимодействия между микросервисами используется система обмена сообщениями NATS JetStream, реализующая модель публикации и подписки (publish-subscribe). Она гарантирует надёжную доставку событий, что критично при обработке сообщений, управлении уведомлениями, интеграции с внешними сервисами (например, чат-ботами) и реализации других реактивных компонентов.

Интеграция с внешними системами управления проектами (Project Management Systems) и другими корпоративными решениями осуществляется через REST API. Для приёма событий от сторонних сервисов и автоматизации обработки задач реализован механизм webhook-интерфейсов с возможностью подключения кастомных обработчиков [26].

Безопасность системы обеспечивается с использованием OAuth 2.0 в сочетании с JWT (JSON Web Tokens), что позволяет реализовать надёжную модель авторизации и разграничения доступа. На всех внешних точках входа применяется шифрование с использованием TLS, что обеспечивает конфиденциальность передаваемых данных. Для защиты содержимого приватных и секретных чатов используется механизм сквозного шифрования (End-to-End Encryption, E2EE). Управление правами доступа осуществляется

на основе ролевой модели (RBAC, Role-Based Access Control), позволяющей гибко определять права пользователей в системе.

В таблице 7 приведены обоснования выбора той или иной технологии.

Таблица 7 – Обоснование выбора архитектурных решений

Компонент	Решение	Почему выбрано	Альтернатива	Причина отказа
Backend	Go	Высокая скорость, минимальный overhead, DevOps-френдли	Node.js, Java	Node.js — ниже производительность; Java — сложнее в деплое
Frontend	React + React Native	Единый стек, повторное использование логики	Angular, Flutter	Angular — громоздкий, Flutter — не Web
БД	PostgreSQL	ACID, масштабируемость, поддержка JSONB	MySQL, MongoDB	MongoDB — не подходит для связанных сущностей
Сообщения	WebSocket + Redis	Быстрое обновление чатов, очереди	Kafka	Сложнее для небольших чатов
События	NATS JetStream	Простота, высокая надёжность, low-latency	RabbitMQ, Kafka	Kafka — избыточен, RabbitMQ — сложен в HA
Безопасность	OAuth2 + JWT + E2EE	Стандарт в корпоративной среде	BasicAuth, session-кейсы	Менее безопасные подходы
Интеграции	REST, Webhooks	Поддержка всеми системами, простота	SOAP	Устаревший подход
Хранилище файлов	MinIO	S3-совместимость, on-premise	AWS S3	Нарушение требований ИБ

Анализ архитектурных решений, представленных в таблице 7, показывает, что выбор технологий для проектируемого корпоративного мессенджера основан на сочетании производительности, масштабируемости, безопасности и соответствия требованиям корпоративной среды. Каждая выбранная компонентная технология оптимально отражает специфику задач распределённой коммуникационной платформы и обеспечивает стабильную работу системы в условиях высокой нагрузки и георасределённости.

Использование Go на стороне backend обосновано высокой производительностью и минимальными накладными расходами, что критически важно для обработки большого количества параллельных соединений. Стек React + React Native обеспечивает единообразие пользовательского интерфейса и ускоряет разработку клиентов для различных платформ. PostgreSQL выбран как надёжная транзакционная СУБД, поддерживающая сложные связи и JSONB-структуры, что делает её оптимальной для хранения сообщений, чатов и данных о пользователях [15].

Связка WebSocket + Redis обеспечивает низкую задержку и мгновенную доставку сообщений, а NATS JetStream формирует отказоустойчивую событийную шину с низким временем реакции. В области безопасности предпочтение отдано современному стеку OAuth2 + JWT и механизмам E2EE, которые удовлетворяют строгим требованиям корпоративной защиты данных. REST-интерфейсы и webhooks выбраны в качестве основного интеграционного механизма благодаря их универсальности и поддержке со стороны внешних сервисов. MinIO обеспечивает хранение файлов в on-premise среде, сохраняя совместимость с S3 и соответствуя требованиям информационной безопасности [28].

Таким образом, набор технологий, отражённый в таблице 7, формирует архитектурный фундамент системы, обеспечивая баланс между производительностью, гибкостью интеграции, безопасностью и соответствием требованиям георасределённой корпоративной инфраструктуры.

2.2.2 Микросервисная структура

На рисунке 10 изображена микросервисная архитектура сервера приложений, где каждый сервис отвечает за строго определённую область функциональности. Такой подход обеспечивает модульность, удобство сопровождения, а также независимое масштабирование компонентов [21].

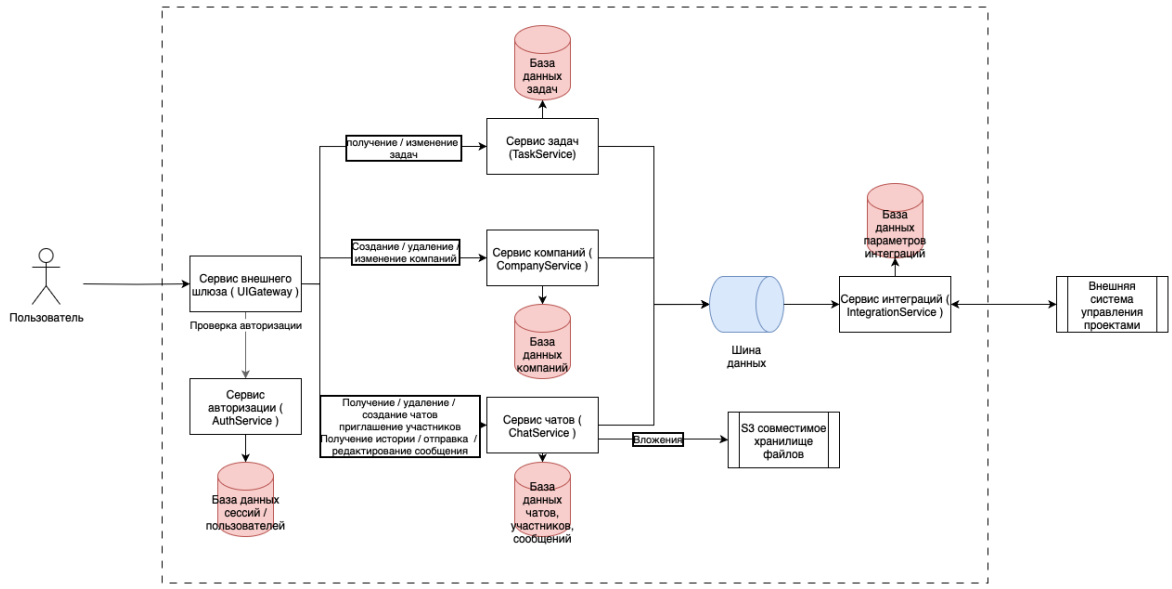

Рисунок 10 – Микросервисная архитектура сервера приложений

Сервис авторизации (AuthService) отвечает за регистрацию и аутентификацию пользователей, обработку OAuth-запросов и управление токенами доступа.

Сервис чатов (ChatService) реализует хранение, отправку и получение сообщений, а также предоставление истории переписки.

Сервис задач (TaskService) выступает в роли прокси-обёртки над внешними API (например, Jira, Redmine) и обеспечивает интеграцию с системами управления задачами.

Шина данных (EventBus) реализует механизм брокера событий, обеспечивая маршрутизацию уведомлений, команд и других сообщений между сервисами;

Внешний шлюз (UIGateway) служит единым входом для всех клиентских приложений, маршрутизируя запросы к соответствующим микросервисам.

Выбор микросервисной архитектуры обусловлен рядом преимуществ:

– Модульность. Позволяет независимо разворачивать, тестировать и обновлять отдельные компоненты;

– Масштабируемость. Ресурсоёмкие сервисы могут быть масштабированы независимо от других;
– Упрощение DevOps - практик. Система легко разворачивается в Kubernetes - кластере, поддерживает A/B-тестирование и непрерывную поставку изменений.

2.2.3 Распределённость и отказоустойчивость

Система спроектирована с приоритетом на отказоустойчивость и устойчивую работу в условиях распределённой инфраструктуры. Все сервисы контейнеризованы с использованием Docker и развёрнуты в среде Kubernetes, что обеспечивает автоматическое масштабирование, восстановление после сбоев и изоляцию компонентов. Непрерывная интеграция и доставка изменений реализованы средствами GitLab CI, где организована автоматическая сборка образов и их деплой в кластер [16].

Для мониторинга состояния системы используется связка Prometheus и Grafana, позволяющая в реальном времени отслеживать ключевые метрики и оперативно реагировать на аномалии. Поддержка health-check механизмов и автоматическое восстановление компонентов при сбоях реализованы средствами самой платформы оркестрации, что сводит к минимуму время простоя и необходимость ручного вмешательства.

2.2.4 Поддержка географически распределённой архитектуры

Система изначально ориентирована на работу в географически распределённой среде, охватывающей несколько территориально удалённых сегментов корпоративной сети. В каждом регионе (например, Москва, Казань, Новосибирск) может функционировать собственный автономный узел, включающий локальные экземпляры микросервисов, собственную базу данных PostgreSQL с поддержкой репликации, а также подсистему кэширования на базе Redis. Как показано на рисунке 11, между регионами осуществляется синхронизация данных и маршрутизация событий посредством кластера NATS JetStream, объединённого в SuperCluster с

поддержкой зеркалирования сообщений по интересам (interest-based mirroring).

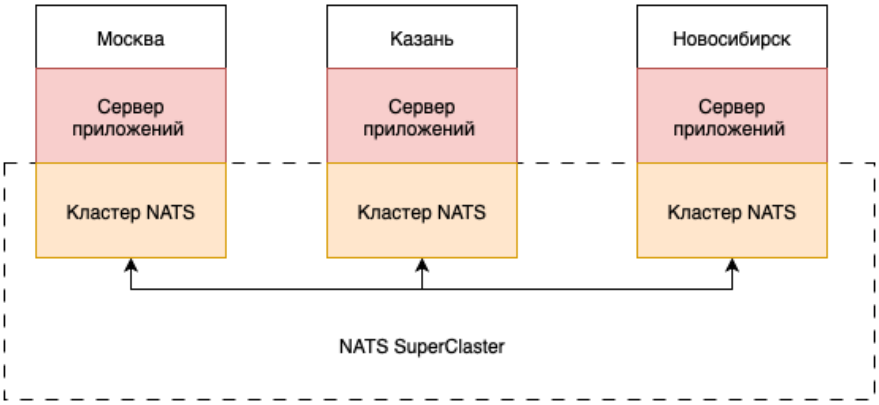

Рисунок 11 – Географически распределенная архитектура на основе NATS SuperCluster

Процесс регистрации пользователей сопровождается географической привязкой: каждый клиент закрепляется за ближайшим региональным узлом, что позволяет минимизировать сетевые задержки. Установление WebSocket-соединений осуществляется с учётом топологии сети и обеспечивает оптимальное быстродействие.

Особое внимание в архитектуре уделяется устойчивости системы к нестабильному соединению. Мобильные клиенты используют локальные хранилища (например, SQLite или AsyncStorage), позволяя сохранять и буферизовать сообщения в автономном режиме. При восстановлении подключения происходит синхронизация данных с сервером, что особенно важно для отдалённых подразделений или пользователей с ограниченным интернет-доступом.

2.2.5 Механизм межрегиональной маршрутизации и хранения данных

Архитектура предусматривает децентрализованное хранение сообщений. Каждый региональный экземпляр приложения осуществляет запись и хранение сообщений в собственной базе данных. В случае, если в чате участвуют пользователи из разных регионов, сообщения сначала сохраняются локально в регионе отправителя, а затем транслируются через шину событий в региональные базы данных других участников. Такая схема позволяет существенно снизить задержки и повысить надёжность доставки данных, сохраняя при этом согласованность истории общения.

2.2.6 Репликация истории при динамическом подключении регионов

В условиях расширяющегося географического охвата системы возникает необходимость обеспечения доступа к истории чатов при подключении участников из новых регионов. Для решения этой задачи реализован механизм отложенной репликации (lazy synchronization). При вступлении пользователя из региона, ранее не участвовавшего в чате, производится проверка наличия локальной истории. В случае её отсутствия региональная нода инициирует запрос к региону-владельцу чата, получает необходимую часть истории (например, последние 1000 сообщений), сохраняет её локально и предоставляет пользователю полный доступ к содержимому [18].

Исторические данные передаются только при наличии соответствующих прав доступа, а в случае секретных чатов с использованием сквозного шифрования (E2EE) доступ к предыдущим сообщениям невозможен — новые участники получают лишь текущие сообщения. Весь процесс синхронизации защищён с использованием TLS, что гарантирует конфиденциальность передаваемой информации.

2.2.7 Политика миграции чатов и управление региональной принадлежностью

С целью обеспечения непрерывности функционирования и отказоустойчивости предусмотрен механизм автоматической смены региона-владельца чата. При создании чата один из регионов назначается в качестве первичного хранилища истории. В процессе эксплуатации, при подключении пользователей из других регионов, активируется межрегиональная репликация. Если впоследствии все пользователи исходного региона покидают чат, система определяет наиболее активный регион среди оставшихся участников и осуществляет перенос статуса владельца. Новый регион получает полномочия на хранение и маршрутизацию сообщений, обеспечивая тем самым устойчивость функционирования чата даже при полной утрате исходной инфраструктуры [17].

Данный подход обеспечивает не только защиту от потери данных, но и балансирует нагрузку между регионами, сохраняя производительность системы на высоком уровне и снижая межрегиональный трафик.

2.2.8 Архитектура секретных чатов

В целях обеспечения конфиденциальности пользовательской переписки в системе реализован механизм сквозного шифрования (End-to-End Encryption, E2EE), при котором только конечные участники общения обладают технической возможностью дешифровать содержимое сообщений. Данный подход исключает возможность доступа к передаваемым данным со стороны промежуточных узлов, включая серверную инфраструктуру [22].

В таблице 8 представлено сравнений популярных E2E алгоритмов.

Таблица 8 – Сравнение популярных E2E алгоритмов

Характеристика	Signal Protocol (SenderKey)	NaCl/box с общим ключом	Matrix Olm/Megolm
Тип шифрования	E2E: X3DH + Double Ratchet + SenderKey	E2E: Symmetric (один ключ) + Box	E2E: X3DH + Megolm (ratcheted group encryption)
Forward Secrecy (FS)	Да	Нет	Частично (внутри Megolm session)
Backward Secrecy (при входе нового участника)	Да (новые ключи)	Нет	Только с новыми Megolm-сессиями
Групповые чаты	Да (SenderKey)	Да	Да
Удаление участника	Безопасен	Нет	Только с перевыпуском session key
Сложность реализации	Очень высокая	Низкая	Высокая
Поддержка истории сообщений	Нет	Да	Да
Производительность (CPU)	Умеренная	Высокая	Средняя
Готовые библиотеки на Go	Нет официальной libsignal-go	Да (x/crypto/nacl/box)	Только через C или обёртки
Использование в реальных системах	WhatsApp, Signal, Messenger	Нет информации	Matrix / Element
Ключевая особенность	Лучшее соотношение FS + безопасность	Простота	

Процесс инициализации защищённого обмена сообщениями начинается с генерации криптографической пары ключей непосредственно на клиентском устройстве пользователя. Приватный ключ остаётся исключительно на устройстве и не передаётся по сети, что исключает риски его компрометации. Публичный ключ, напротив, отправляется на сервер и используется другими пользователями системы для шифрования сообщений, адресованных владельцу соответствующего приватного ключа.

При формировании сообщения отправителем генерируется уникальный симметричный ключ (например, с использованием алгоритмов AES или ChaCha20), применяемый для шифрования содержимого сообщения. Далее, для каждого получателя создаётся персонализированная копия симметричного ключа, зашифрованная его публичным ключом с использованием асимметричного криптографического алгоритма (возможны реализации на основе RSA-OAEP, NaCl Box или ECDH-соглашений). Результатом становится структура сообщения как показано в листинге на рисунке 12.

```
{
  "EncryptedMessage": "...",
  "EncryptedKeys": {
    "user1": "...",
    "user2": "..."
  },
  "Sender": "userX",
  "Timestamp": "...",
  ...
}
```

Рисунок 12 - Структура сообщения с ключами шифрования

Каждый клиент, получив сообщение, извлекает соответствующий ему зашифрованный симметричный ключ, расшифровывает его при помощи собственного приватного ключа, после чего использует полученный симметричный ключ для расшифровки основного содержимого сообщения.

Преимуществом подобного подхода является реализация полноценного сквозного шифрования, при котором ни сервер, ни иные посредники не обладают возможностью получить доступ к расшифрованным данным. Архитектура протокола остаётся сравнительно простой, а его реализация не требует сложной инфраструктурной поддержки. Благодаря тому, что каждый пользователь получает отдельную копию ключа, система естественным образом поддерживает как индивидуальные, так и групповые переписки.

Дополнительным преимуществом выбранного подхода является возможность безопасной репликации зашифрованных сообщений между

географически распределёнными регионами: поскольку все данные передаются исключительно в зашифрованном виде, репликация не нарушает требований конфиденциальности, а история сообщений может быть восстановлена при необходимости без ущерба безопасности. Более того, отсутствует необходимость кеширования полного чата на клиентском устройстве, поскольку каждый получатель способен расшифровать только адресованные ему сообщения, независимо от региона получения.

2.3 Разработка пользовательского интерфейса

В процессе работы над проектом были разработаны прототипы пользовательского интерфейса, выполненные в виде низкоуровневой (wireframe) схемы. Следует подчеркнуть, что целью данной визуализации не является разработка окончательного графического дизайна, а лишь демонстрация функциональной структуры и ключевых элементов интерфейса. Использование условных обозначений и абстрактной стилизации служит для акцентирования внимания на логике взаимодействия, а не на визуальной эстетике [20].

Представленные схемы служит промежуточным артефактом проектирования, используемым на этапе прототипирования пользовательского интерфейса. Подобные абстракции позволяют на ранних стадиях разработки формализовать требования к логике отображения и взаимодействия, не отвлекаясь на визуальное оформление. В дальнейшем данный макет может служить основой для реализации высокоуровневых прототипов или непосредственного фронтенд-кода.

2.3.1 Экран авторизации

На рисунке 12 представлен экран авторизации, предназначенный для идентификации пользователей посредством ввода учётных данных. В данном случае реализована классическая схема входа по электронной почте и паролю. Указанный функционал является частью модуля аутентификации системы,

который в дальнейшем взаимодействует с серверной частью по защищённому протоколу (например, HTTPS с JWT или OAuth 2.0/PKCE).

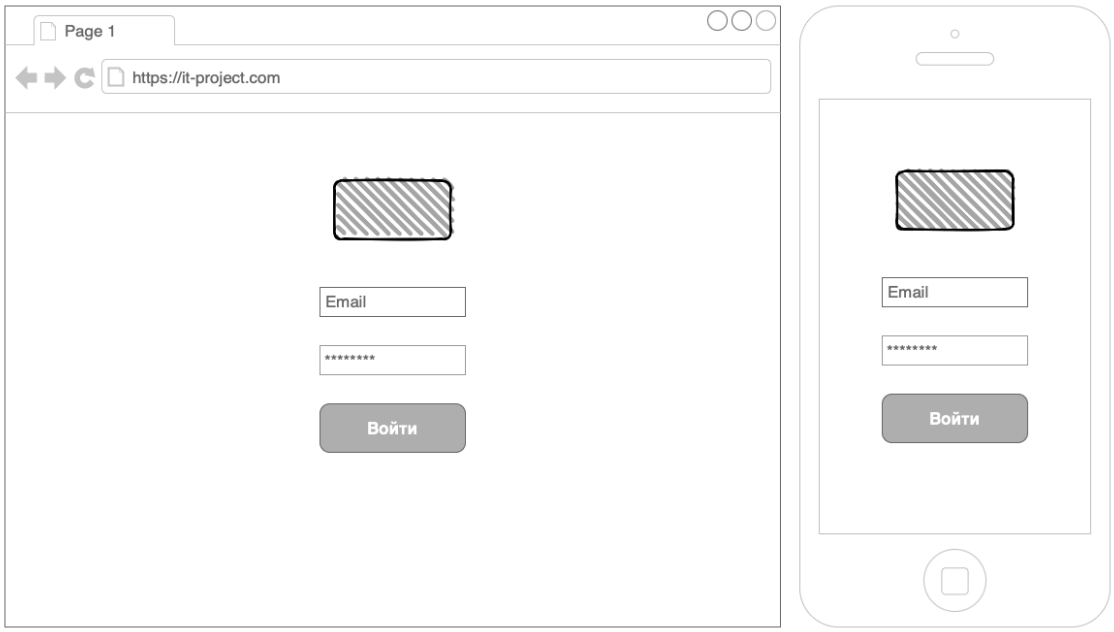

Рисунок 12 – Прототип экрана авторизации

Рисунок 13 демонстрирует прототип экрана авторизации, реализованный в виде низкоуровневой wireframe-схемы. На макете представлены два варианта отображения — веб-интерфейс и мобильная версия, что подчёркивает кроссплатформенность разрабатываемого корпоративного мессенджера. Оба интерфейса содержат минимальный набор элементов, необходимый для идентификации пользователя: поля ввода адреса электронной почты и пароля, а также кнопку входа. Данный прототип отражает базовую логику взаимодействия в модуле аутентификации и служит отправной точкой для дальнейшего проектирования пользовательского интерфейса, обеспечивая единообразие визуальной структуры и удобство использования на различных устройствах.

2.3.2 Основной экран мессенджера

На рисунке 13 представлен стартовый экран мессенджера, выполняющий функцию центральной навигационной точки, с которой пользователь начинает взаимодействие с системой. Визуально он представлен в виде классического интерфейса мессенджера: слева — список диалогов и задач, справа (или в основной зоне) — область для отображения содержимого выбранного чата. В случае, если чат не выбран, отображается информационное сообщение-заглушка.

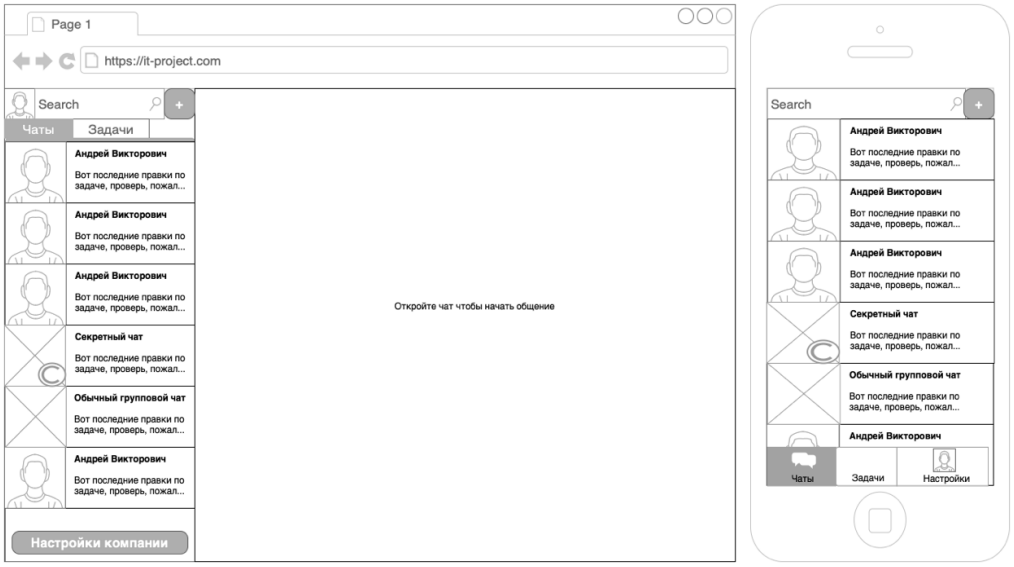

Рисунок 13 – Стартовый экран мессенджера

Рисунок 14 демонстрирует стартовый экран корпоративного мессенджера, выполняющий роль основной навигационной точки, с которой пользователь начинает работу в системе. Интерфейс построен по классическому принципу мессенджеров: слева располагается панель с перечнем доступных чатов и задач, позволяющая быстро переключаться между диалогами и рабочими процессами, а центральная область предназначена для отображения содержимого выбранного чата. Если диалог не активен, в основной зоне выводится информационное сообщение-заглушка, подсказывающее пользователю продолжить работу.

На рисунке представлены как веб-версия, так и мобильная адаптация, что подчёркивает кроссплатформенность разрабатываемого решения. Оба варианта сохраняют единый подход к визуальной структуре: список чатов расположен слева (или в нижней навигационной панели в мобильной версии), а основная часть экрана отведена под содержимое выбранного диалога. Такое решение обеспечивает единообразие пользовательского опыта и удобство работы вне зависимости от устройства.

2.3.3 Экран чата

На рисунке 14 изображен интерфейс системы обмена сообщениями. Он реализован таким образом, что пользователю предоставляются все необходимые элементы для эффективного взаимодействия в корпоративной среде. В настольной версии он разделён на несколько функциональных областей. Слева находится панель навигации, где собраны чаты и задачи. Пользователь видит список доступных диалогов с кратким содержанием последних сообщений, что позволяет быстро сориентироваться и выбрать нужный контакт или группу. В нижней части панели расположен доступ к настройкам компании, которые позволяют управлять параметрами работы системы в организационном контексте.

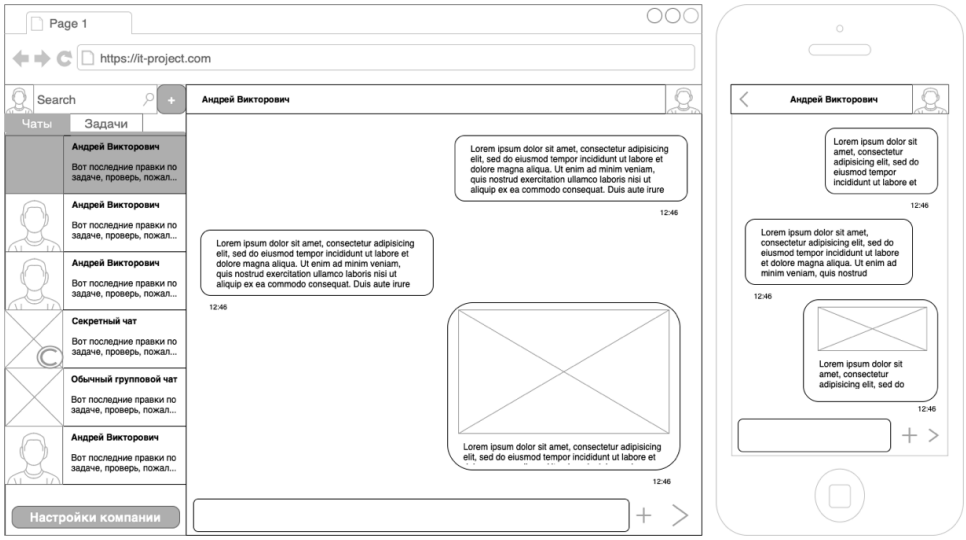

Рисунок 14 – Прототип экрана открытого личного чата с сотрудником

Основная часть окна занимает область активного диалога. Здесь отображаются все сообщения в хронологическом порядке, каждое снабжено временной меткой, что обеспечивает прозрачность коммуникации. Сообщения представлены в удобочитаемой форме и могут содержать не только текст, но и изображения, встроенные в поток беседы. В нижней части интерфейса размещено поле для ввода текста, а также кнопки для отправки и добавления вложений, что обеспечивает удобство в создании мультимедийных сообщений.

Мобильная версия сохраняет все ключевые возможности, адаптируя их к небольшому экрану. Пользователь сразу попадает в выбранный диалог и имеет возможность читать сообщения, просматривать изображения и отвечать на них. Поле ввода и иконки отправки располагаются в нижней части экрана, что соответствует привычной логике мобильных приложений.

Пользователь выбирает имя или аватар собеседника в верхней части окна чата, после чего открывается карточка профиля, как на рисунке 15. Аналогичная логика сохраняется и в мобильной версии: нажатие на область с именем контакта переводит пользователя к подробной информации о нём.

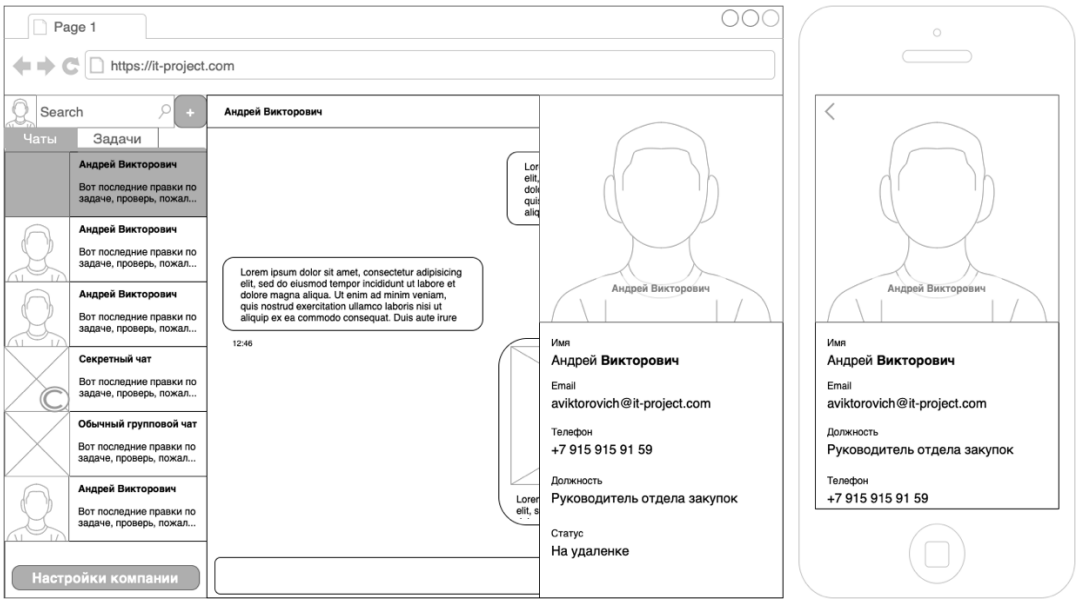

Рисунок 15 – Карточка сотрудника

Назначение экрана заключается в предоставлении расширенных сведений о собеседнике. В карточке отображаются основные идентификационные данные: имя, адрес электронной почты, номер телефона, должность и статус, отражающий текущее положение сотрудника (например, работа в офисе или удалённый режим). Такая информация обеспечивает пользователю возможность быстро установить контакт альтернативными средствами связи, уточнить должностные полномочия или статус доступности собеседника.

Таким образом, данный экран играет вспомогательную роль в структуре интерфейса: он связывает функциональность обмена сообщениями с корпоративным справочником персонала. Это позволяет не только вести коммуникацию в режиме чата, но и получать справочные данные о сотрудниках, что повышает интеграцию системы в деловые процессы организации.

На рисунке 16 представлен список задач, который является неотъемлемой частью интерфейса и обеспечивает пользователю доступ к функциональности управления рабочими процессами в рамках корпоративной коммуникационной системы. Она реализована в виде отдельной панели, доступной через переключатель между чатами и задачами, что подчёркивает равнозначность коммуникационного и организационного компонентов системы.

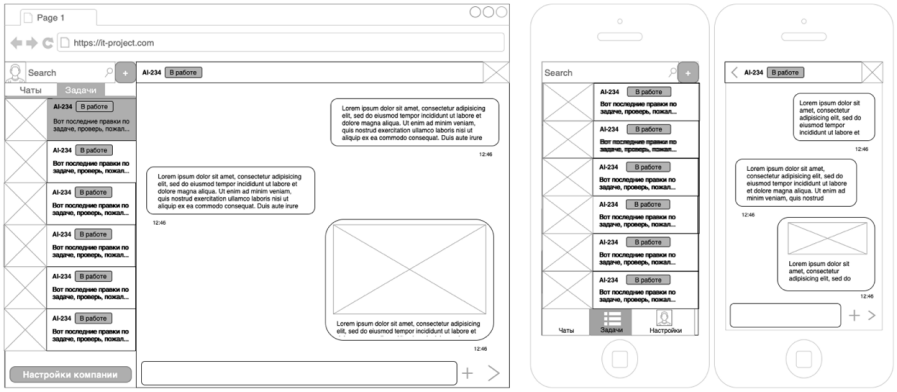

Рисунок 16 – Прототип экрана открытого чата по задаче

В настольной версии вкладка отображает список актуальных задач, каждая из которых представлена в виде карточки с уникальным идентификатором (например, «A1234»), названием или кратким описанием, а также статусом, отражённым с помощью текстового маркера («в работе», «выполнено» и т.д.). Такая организация позволяет быстро просматривать состояние задач и выбирать необходимую для дальнейшей работы. Выбор конкретной задачи открывает в центральной области окна её подробное содержание, включая текстовое описание, вложенные материалы или обсуждение. Здесь сохраняется визуальная логика диалогового окна, что обеспечивает единообразие пользовательского опыта: задачи представлены в том же формате «сообщений», что делает их восприятие привычным и упрощает взаимодействие.

Мобильная версия воспроизводит аналогичный функционал с адаптацией к компактному экрану. Вкладка задач открывается через соответствующий переключатель в нижней части интерфейса. Список задач представлен в вертикальном скроллинге, где каждая карточка включает идентификатор, описание и статус. При выборе задачи пользователь переходит на экран с её детализированным содержанием, выполненным в формате диалога, где также предусмотрена возможность чтения текста и просмотра вложений.

На рисунке 17 представлен интерфейс детализированного просмотра задачи, доступ к которому осуществляется посредством выбора её идентификатора в общем списке. При активации элемента происходит переход на экран с расширенной информацией, где отображаются ключевые атрибуты задачи: уникальный идентификатор, название, назначенный ответственный исполнитель, текущий статус, а также текстовое описание.

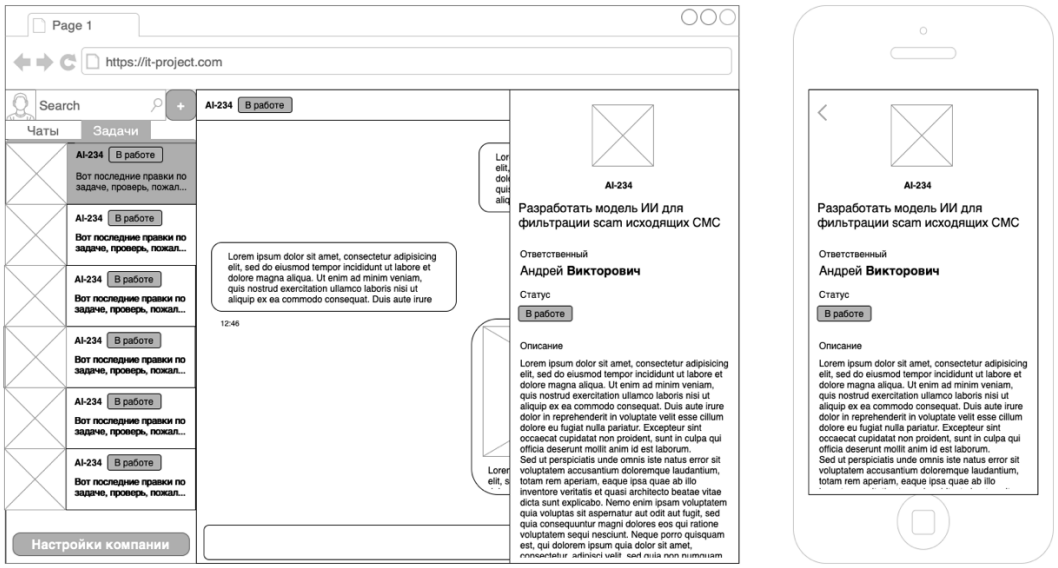

Рисунок 17 – Прототип экрана открытого чата по задаче с детальной информацией.

Особое внимание уделено функциональности изменения статуса, что позволяет пользователю в интерактивном режиме управлять прогрессом выполнения задачи. Данный элемент интерфейса отражает принцип интеграции коммуникационных и организационных инструментов в единую среду, где задачи могут не только просматриваться, но и оперативно редактироваться. Визуальное оформление сохраняет преемственность с чатовым форматом, что снижает когнитивную нагрузку при переключении между типами активности.

2.4 Разработка программных модулей

Архитектура программной системы построена на принципах микросервисной архитектуры, что обеспечивает высокий уровень модульности, масштабируемости и устойчивости к ошибкам. В рамках реализации информационной системы программный комплекс был декомпозирован на шесть специализированных микросервисов, каждый из которых инкапсулирует строго определённую предметную область и

взаимодействует с другими компонентами посредством асинхронной или синхронной межсервисной коммуникации.

2.4.1 Сервис авторизации (AuthService)

Микросервис аутентификации и авторизации обеспечивает функции регистрации и входа пользователей, управления жизненным циклом сессий, выпуска и валидации токенов доступа (JWT/refresh token), поддержки протоколов аутентификации сторонних поставщиков (OAuth 2.0, OpenID Connect).

На рисунке 18 изображена диаграмма трехслойной архитектуры аутентификации: транспорт (GRPCAuthServer), доменная логика (AuthService) и доступ к данным (UserRepository, SessionRepository). GRPCAuthServer лишь проксирует вызовы. AuthService оркестрирует регистрацию, вход и валидацию сессий, опираясь на абстракции репозиториев. UserRepository управляет поиском и сохранением пользователей, SessionRepository — созданием, получением и инвалидированием сессий. Доменные сущности: User (UUID, email, хеш пароля, время создания) и Session (UUID, связь с пользователем, клиентские метаданные, время создания и истечения). Зависимости направлены сверху вниз к абстракциям, что соответствует принципам инверсии зависимостей и упрощает тестирование и замену инфраструктуры.

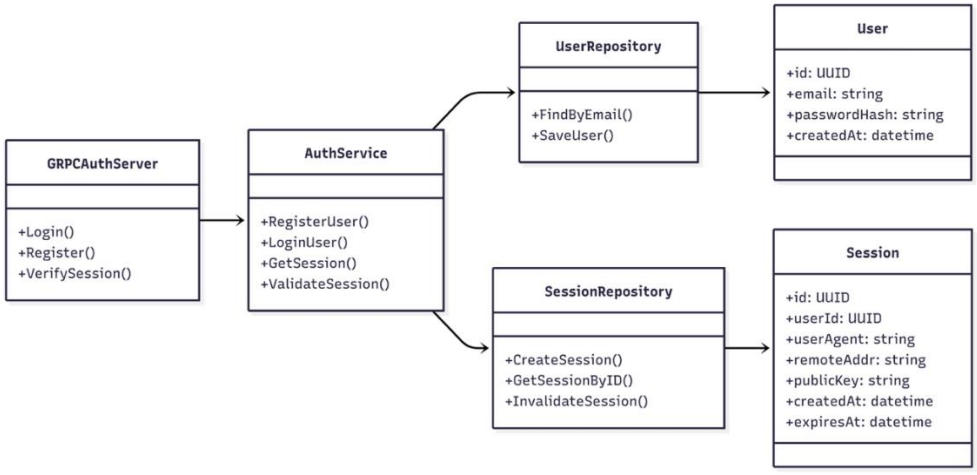

Рисунок 18 – Диаграмма классов сервиса авторизации

2.4.2 Сервис компании (CompanyService)

Микросервис управления корпоративной информацией обеспечивает функции получения и изменения сведений о компании, работы со списком сотрудников, их структурной иерархией.

На рисунке 20 изображена диаграмма классов сервиса с разделением на транспорт, доменную логику и доступ к данным. На входном уровне GRPCCompanyServer предоставляет операции получения списков и отдельных сущностей (сотрудников и подразделений), делегируя всю бизнес-логику CompanyService.

CompanyService инкапсулирует предметные сценарии: агрегирует данные из репозиториев, применяет инварианты (корректность ссылок сотрудник–подразделение), обеспечивает единые контракты выдачи. Доступ к данным реализован репозиториями: EmployeeRepository и DepartmentRepository скрывают детали персистентности, предоставляя поиск всех и выборку по идентификатору.

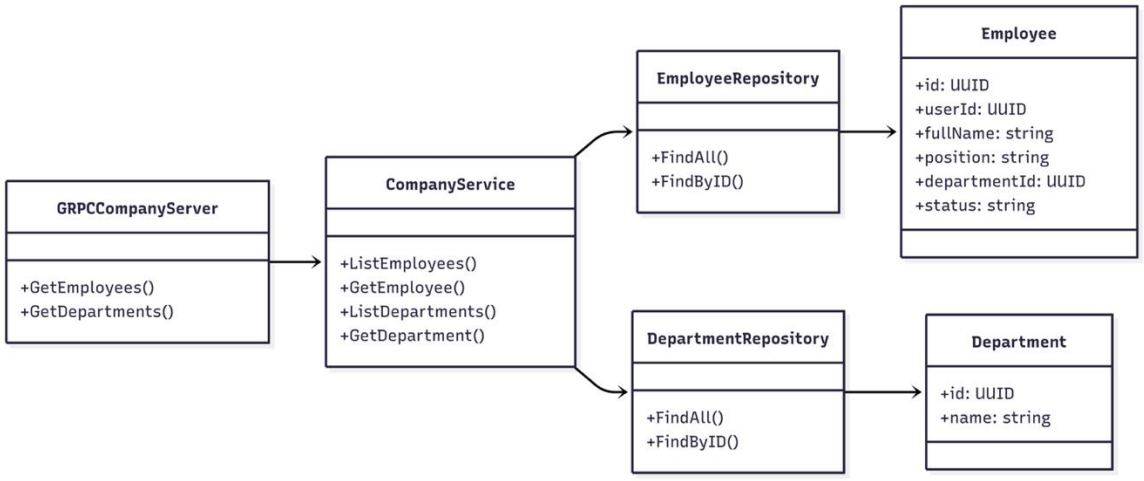

Рисунок 20 – Диаграмма классов сервиса Company

Рисунок 20 иллюстрирует диаграмму классов сервиса Company, отражающую трёхслойную структуру взаимодействия между транспортным уровнем, доменной логикой и слоем доступа к данным. На стороне транспортного уровня представлен компонент GRPCCompanyServer, обеспечивающий внешние точки входа для получения информации о

сотрудниках и подразделениях. Все запросы маршрутизируются в CompanyService, который инкапсулирует бизнес-логику, агрегирует данные и обеспечивает целостность предметных сущностей.

Доступ к хранилищу данных осуществляется через два репозитория — EmployeeRepository и DepartmentRepository. Они скрывают детали персистентности, предоставляют операции поиска всех элементов и выборки по идентификатору, а также служат абстракциями для работы с таблицами сотрудников и подразделений. Сущности Employee и Department представляют структурированные данные, используемые в доменной модели и включающие идентификаторы, основные атрибуты и служебную информацию.

Диаграмма демонстрирует строгую декомпозицию обязанностей между слоями, соблюдение принципа инверсии зависимостей и единый формат взаимодействия, что соответствует требованиям масштабируемой микросервисной архитектуры.

2.4.3 Сервис чатов (ChatService)

Сервис обмена сообщениями обеспечивает функционал управления чатами, участниками и хранения сообщений. На рисунке Рисунок изображена диаграмма классов трехслойной архитектуры: транспорт (GRPCChatServer), доменная логика (ChatService) и доступ к данным (ChatRepository, MessageRepository). GRPCChatServer экспонирует операции листинга чатов, получения чата и сообщений, отправки сообщения и управления участниками, делегируя логику ChatService.

ChatService оркестрирует сценарии: создает/находит чаты, обеспечивает целостность состава участников, контролирует доставку и сохранение сообщений, связывает сообщения с ключами шифрования. Репозитории инкапсулируют персистентность: ChatRepository отвечает за выборку чатов и их участников, создание чатов; MessageRepository — за сохранение, выборку сообщений и ключей шифрования, а также атомарные операции записи.

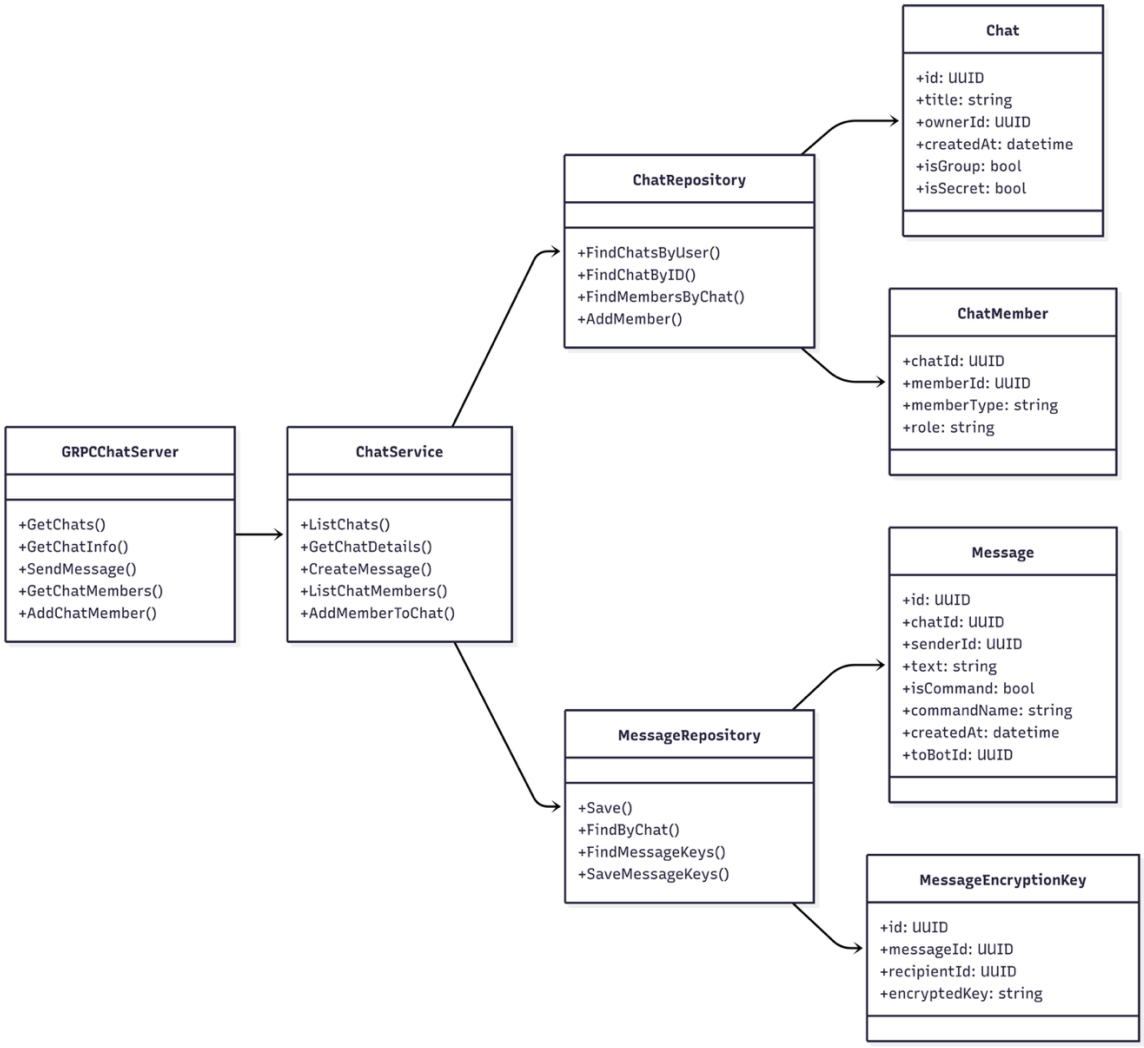

Рисунок 21 – Диаграмма классов сервиса чатов

Рисунок 21 представляет диаграмму классов сервиса чатов, демонстрирующую многослойную организацию компонента, обеспечивающего управление чатами и обработку сообщений в корпоративном мессенджере. На транспортном уровне функционирует GRPCChatServer, предоставляющий внешний интерфейс для получения списка чатов, информации о конкретном чате, отправки сообщений и управления участниками. Все входящие запросы передаются в ChatService, который реализует центральную доменную логику: обработку команд, формирование структуры чата, создание и сохранение сообщений, а также управление составом участников.

Доступ к данным инкапсулирован репозиториями ChatRepository и MessageRepository. Первый отвечает за выборку чатов, поиск участников и добавление новых членов, второй — за сохранение сообщений, поиск истории переписки и работу с ключами шифрования. Над ними располагаются доменные сущности Chat, ChatMember, Message и MessageEncryptionKey, описывающие структуру данных, используемых системой: параметры чата, роли участников, содержание сообщений и зашифрованные симметричные ключи, необходимые для реализации механизма E2EE.

Диаграмма демонстрирует чёткое разделение обязанностей между транспортным, доменным и инфраструктурным уровнями, что обеспечивает гибкость, расширяемость и тестируемость сервиса. Такая структура соответствует принципам Clean Architecture и является необходимой для построения масштабируемой микросервисной системы.

2.4.4 Сервис задач (TaskService)

Сервис управления задачами обеспечивает функции создания, получения и комментирования задач. На рисунке 22 изображена диаграмма классов, структурированная по трем слоям: транспорт (GRPCTaskServer), доменная логика (TaskService) и доступ к данным (TaskRepository, CommentRepository). GRPCTaskServer экспонирует операции получения списков и деталей задач, создания задач и получения комментариев, делегируя бизнес-логику TaskService.

TaskService оркестрирует сценарии: агрегирует задачи по пользователю, извлекает детали, создает новые записи и формирует ленту комментариев. Он обеспечивает инварианты целостности (согласованность статусов, корректность ссылок задача–комментарии) и изолирует доменную логику от механизмов хранения, опираясь на абстракции репозиториев.

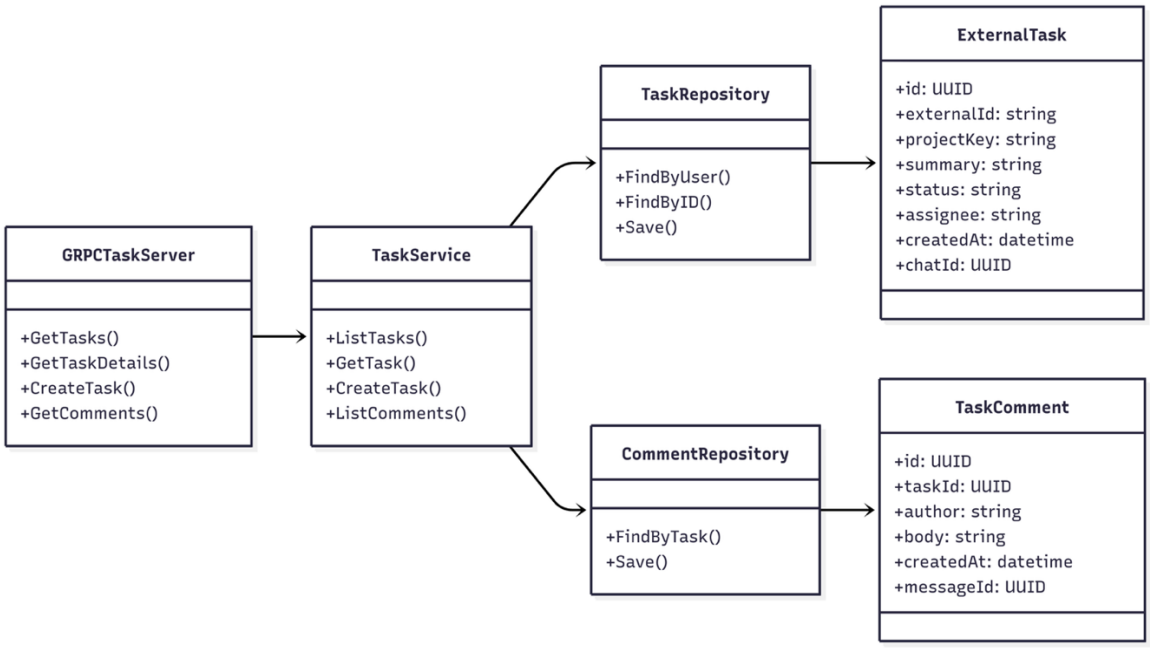

Рисунок 19 – Диаграмма классов сервиса чатов

TaskRepository инкапсулирует доступ к задачам: поиск по пользователю, выборка по идентификатору и сохранение. CommentRepository отвечает за комментарии: выборка по задаче и сохранение. Такое разделение повышает тестируемость и позволяет варьировать персистентность без изменения доменного слоя.

2.4.5 Сервис интеграций (IntegrationService)

Сервис интеграции обеспечивает функции управление интеграциями, ботами и их командами, а также обработку вебхуков и создание задач на основе команд ботов. На рисунке 20 изображена диаграмма классов с трехслойной организацией: транспорт (GRPCIntegrationServer), доменная логика (IntegrationService) и доступ к данным (IntegrationRepository, BotRepository, BotCommandRepository). Транспортный слой лишь маршрутизирует вызовы получения/создания интеграций, ботов и команд, а также постановку и контроль интеграционных задач.

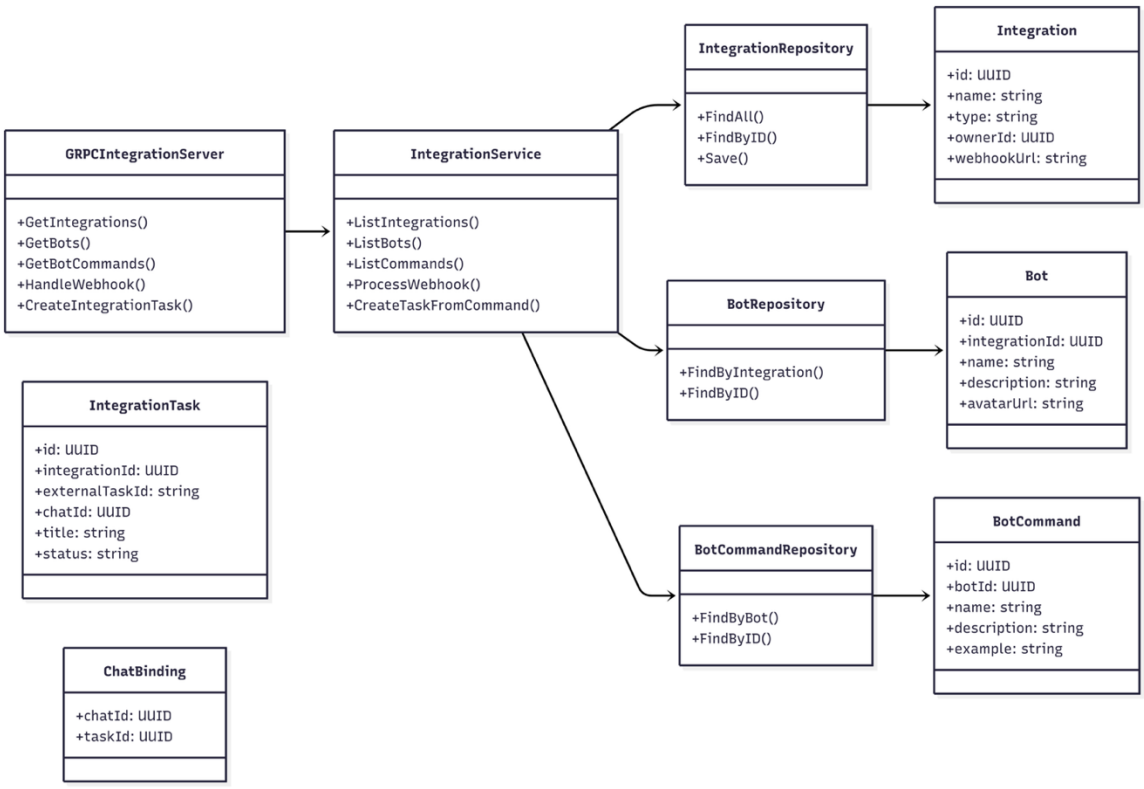

Рисунок 20 – Диаграмма классов сервиса интеграций

IntegrationService координирует сценарии: ведет реестр интеграций, управляет ботами и их командами, инициирует задачи интеграции и формирует связывание с чатами. Он проверяет инварианты (уникальность и корректность webhook/endpoint, соответствие команд ботам, валидность связей чат–интеграция) и изолирует доменную логику от персистентности.

Репозитории инкапсулируют доступ к данным: IntegrationRepository — поиск/сохранение интеграций; BotRepository — выборка ботов в контексте интеграции и по идентификатору; BotCommandRepository — получение команд бота и выборка по ID. Это обеспечивает заменяемость хранилища и тестируемость.

2.4.6 Сервис внешнего шлюза (GatewayService)

Сервис представляет собой специализированный программный компонент, реализующий паттерн фасада (API Gateway) для унифицированного взаимодействия внешних клиентов с микросервисной

системой. Его основное предназначение — консолидация API и маршрутизация запросов к соответствующим внутренним сервисам.

Сервис обеспечивает функции приёма и обработка HTTP- и WebSocket-запросов от пользовательских интерфейсов, верификацию и декодирование токенов авторизации, поддержку двустороннего взаимодействия с клиентом через WebSocket, получение событий из внутренней шины событий (Event Bus) и доставка уведомлений, маршрутизации API-запросов к соответствующим микросервисам: auth-service, chat-service, task-service, company-service.

На рисунке 21 представлена диаграмма классов, которая отражает фасадный сервис внешнего шлюза, объединяющий HTTP и событийное взаимодействие. Центральный компонент UIGateway реализует прием и маршрутизацию HTTP-запросов, управление подписками и пересылку событий клиентам доменных сервисов, а также уведомление подключенных фронтендов.

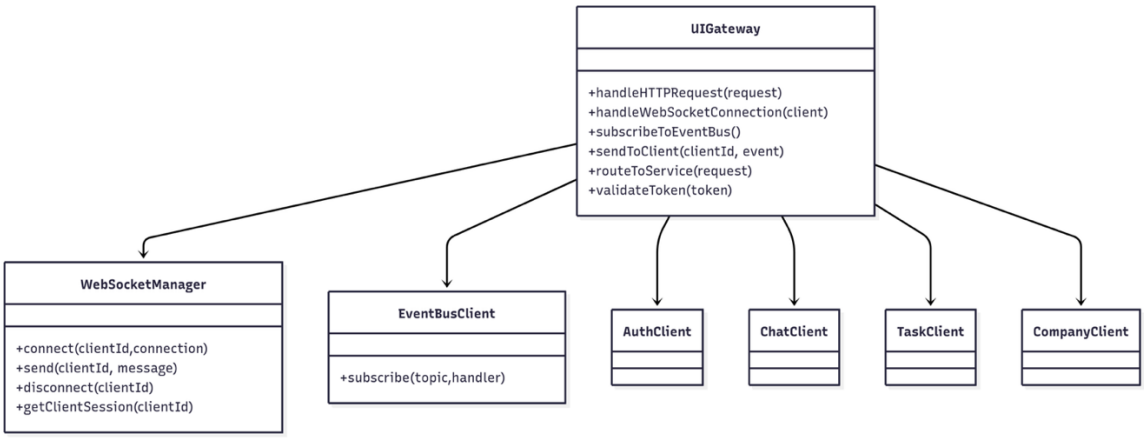

Рисунок 21 – Диаграмма классов сервиса внешнего шлюза

UIGateway опирается на:

– WebSocketManager: управление жизненным циклом веб-сокет-сессий (регистрация/удаление клиентов, отправка сообщений, закрытие соединений).

– EventBusClient: подписка на внутренние топики и доставка событий в UGateway.
– Клиентские адаптеры доменных сервисов (AuthClient, ChatClient, TaskClient, CompanyClient): инкапсуляция RPC-вызовов, унификация контрактов и трансляция ошибок.

Зависимости направлены к абстракциям клиентов и менеджера соединений, что обеспечивает слабую связанность, расширяемость (подключение новых сервисов), и единый контроль кросс-сервисных политик (аутентификация, формат событий).

2.5 Разработка диаграмм межсервисного взаимодействия

2.5.1 Авторизация клиента

На рисунке 22 представлена диаграмма последовательности, которая описывает протокол установления двунаправленного канала связи поверх WebSocket с обязательной аутентификацией по маркеру.

Инициирующее сообщение от клиента поступает на внешний шлюз и содержит токен. Шлюз не интерпретирует токен самостоятельно, а передает его в специализированный сервис аутентификации для криптографической и семантической проверки, тем самым отделяя ответственность за безопасность от транспортного уровня. В случае положительного вердикта выполняется регистрация соединения во внутреннем менеджере веб-сокетов, который фиксирует соответствие идентификатора клиента и сетевого канала, резервирует необходимые ресурсы и переводит состояние сессии в «подключено». Только после подтверждения обоих этапов — валидации и регистрации — шлюз инициирует обратную передачу сигнала подтверждения установления канала клиенту [25]. Такой порядок обеспечивает причинно-следственную согласованность операций, предотвращает создание неаутентифицированных сессий и минимизирует окно уязвимости между

сетевым рукопожатием и включением соединения в контур обработки событий.

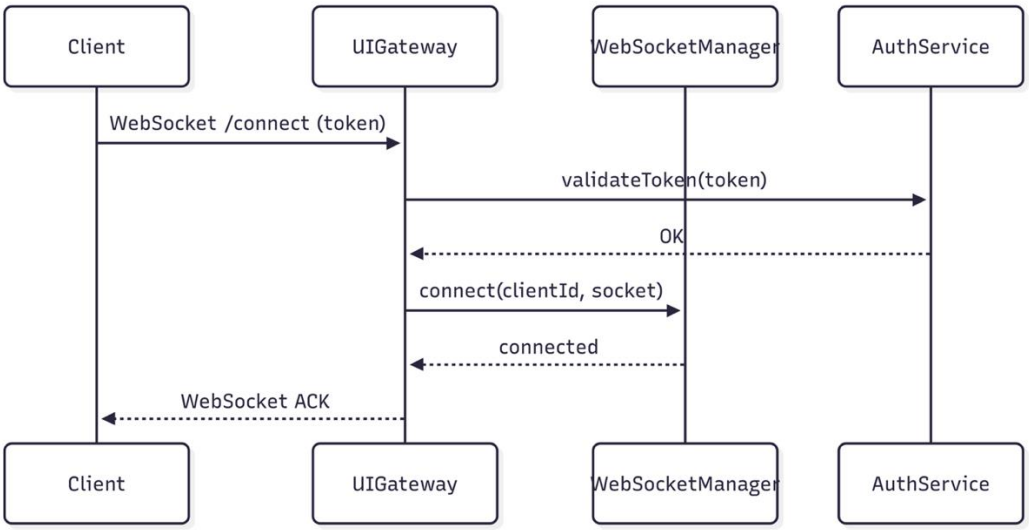

Рисунок 22 – Диаграмма последовательности установления двунаправленного канала связи

Диаграмма на рисунке 23 демонстрирует транзит события о новом сообщении от внутренней шины к конкретному клиенту через внешний шлюз. Публикация события на шине фиксирует факт появления сообщения в заданном чате и инициирует реакцию шлюза. UIGateway извлекает назначение, разрешает пользовательскую сессию через менеджер веб-сокетов и получает дескриптор активного соединения. При наличии валидной сессии шлюз осуществляет адресную доставку полезной нагрузки с сохранением порядка публикации событий. WebSocketManager выполняет операцию push, транслируя сообщение по установленному каналу до клиента [27]. Такой конвейер обеспечивает отделение доменных событий от транспортной логики, минимизирует задержку доставки и сохраняет неизменность семантики события при преобразовании в протокол уведомлений.

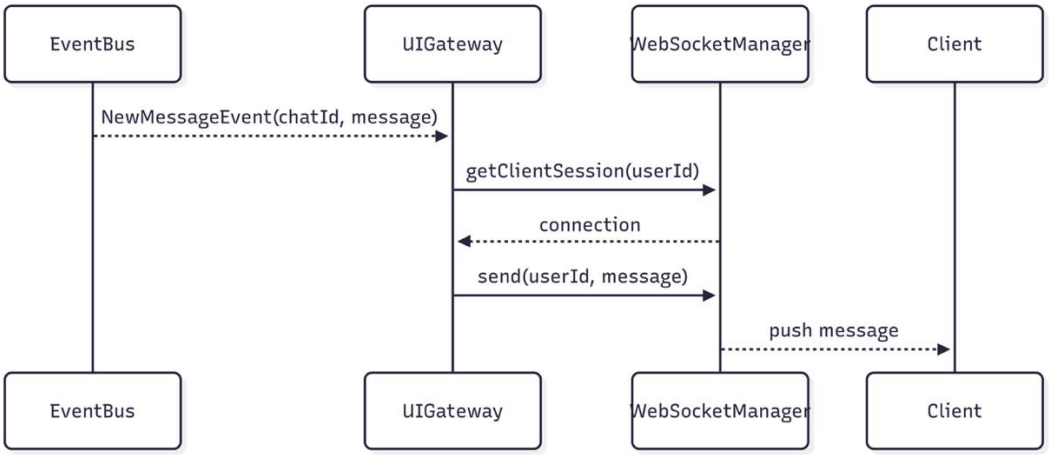

Рисунок 23 – Диаграмма последовательности обработки события о входящем сообщении

Диаграмма на рисунке 24 описывает процесс обработки клиентского запроса на получение списка доступных чатов.

Клиент инициирует WebSocket-запрос на получение перечня чатов. Внешний шлюз извлекает серверный сеансовый контекст через менеджера веб-сокетов, тем самым однозначно устанавливая субъект запроса. Используя этот контекст, шлюз обращается к сервису чатов для выборки разговоров, релевантных пользователю и его правам. Полученный результат нормализуется в типизированный ответ и передается клиенту по исходному каналу. Такая последовательность гарантирует аутентифицированность запроса, изоляцию транспортного уровня от доменной логики и детерминированную доставку данных. в рамках событийно-ориентированной микросервисной архитектуры корпоративного мессенджера.

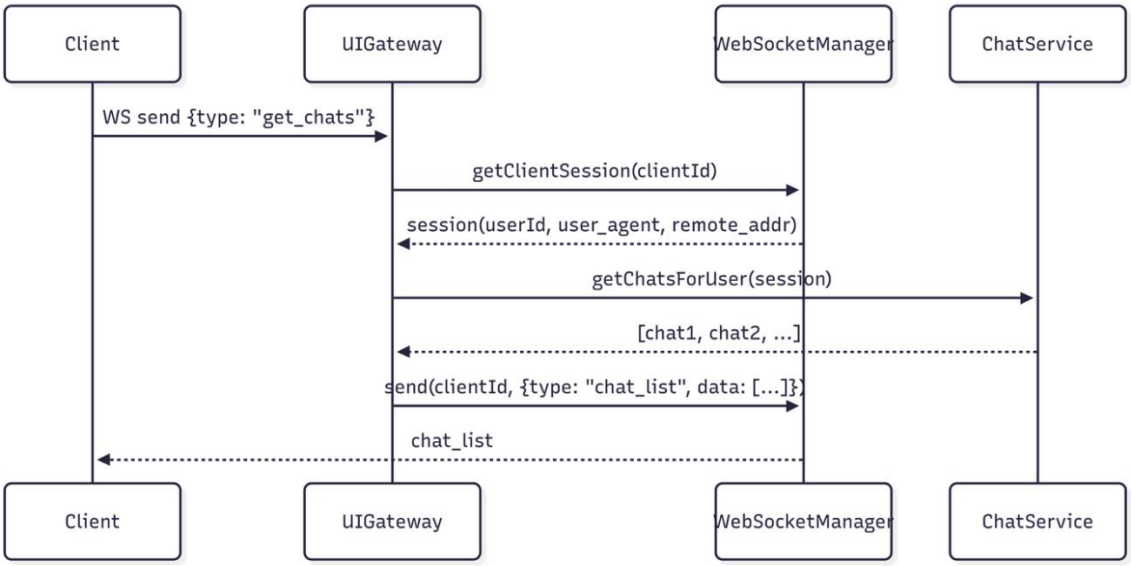

Рисунок 24 – Диаграмма последовательности получение списка чатов

Диаграмма на рисунке 25 моделирует обработку клиентского запроса на получение истории сообщений конкретного чата.

Клиент инициирует WebSocket-запрос на выдачу истории сообщений конкретного чата с параметрами пагинации. Внешний шлюз запрашивает у менеджера веб-сокетов серверный сеансовый контекст, тем самым фиксируя аутентифицированного субъекта и параметры окружения. С использованием данного контекста шлюз обращается к сервису чатов за выборкой сообщений по идентификатору чата и запрошенному окну истории. Возвращенная выборка нормализуется в типизированный ответ и передается клиенту по исходному каналу. Последовательность обеспечивает контекст запроса, изоляцию транспортного уровня от доменной логики и устойчивую детерминированность выдачи при постраничной навигации.

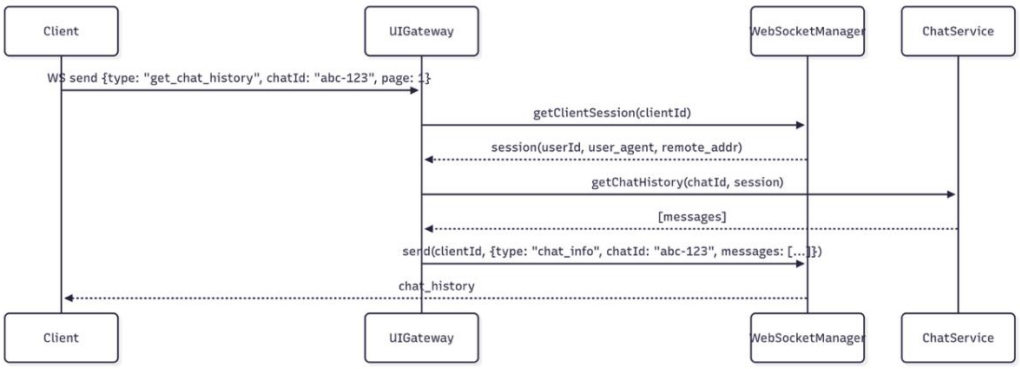

Рисунок 25 – Диаграмма последовательности получения истории чата

Диаграмма на рисунке 26 моделирует жизненный цикл отправки текстового сообщения пользователем. Клиент формирует WebSocket-запрос на отправку сообщения, указывая чат и содержание. Внешний шлюз извлекает сеансовый контекст через менеджера веб-сокетов, идентифицируя субъекта и параметры окружения. С данным контекстом шлюз передает в сервис чатов команду на публикацию сообщения; сервис валидирует права и состояние чата, создает доменную запись и возвращает материализованный объект сообщения. Шлюз нормализует результат и подтверждает отправку клиенту типизированным ответом, после чего инициирует публикацию доменного события в шину. Такая последовательность обеспечивает аутентифицированное действие, атомарную фиксацию факта отправки и последующую доставку уведомлений подписчикам.

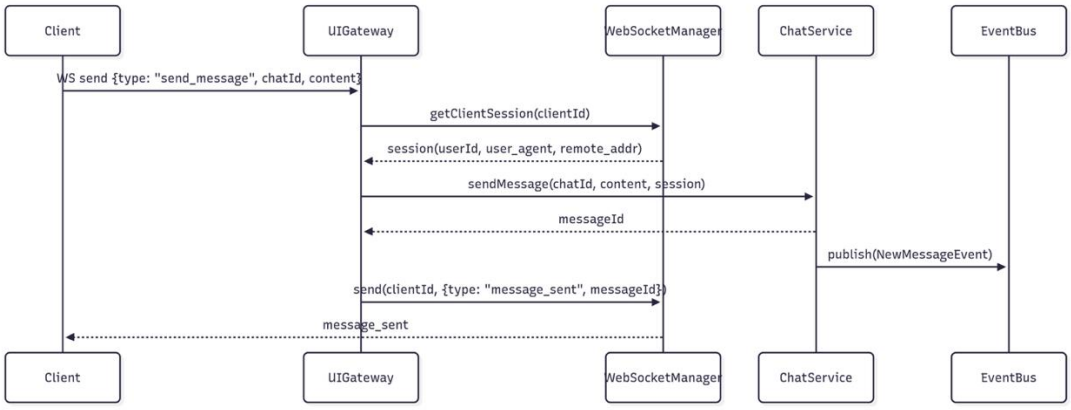

Рисунок 26 – Диаграмма последовательности отправки сообщения

Диаграмма на рисунке 30 иллюстрирует процесс запроса и получения перечня сотрудников компании.

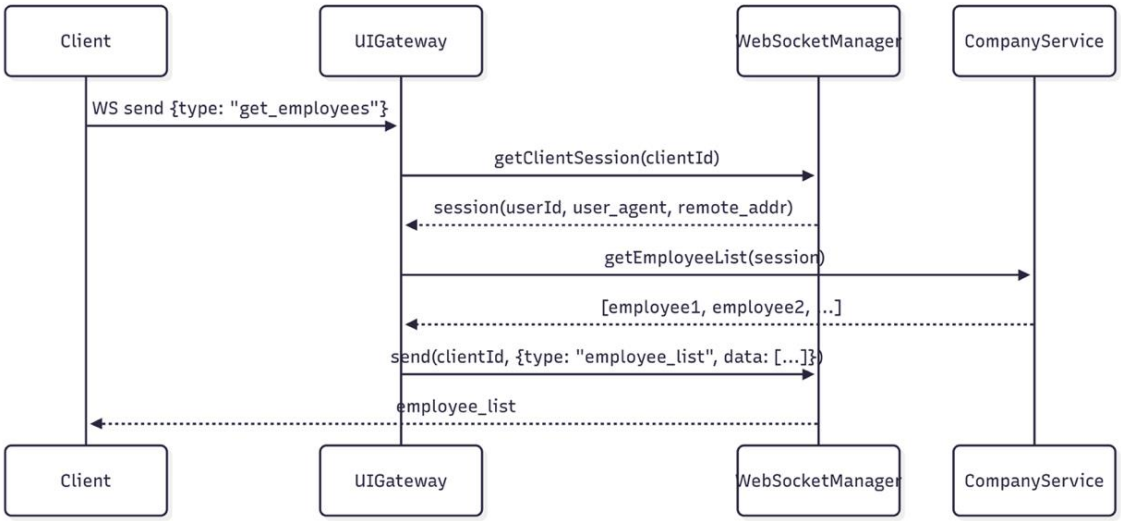

Рисунок 30 – Диаграмма последовательности получения списка сотрудников

Клиент инициирует WebSocket-запрос на выдачу перечня сотрудников. Внешний шлюз извлекает через менеджер веб-сокетов сеансовый контекст, тем самым однозначно устанавливая субъект запроса. Используя данный контекст, шлюз обращается к сервису компании за выборкой сотрудников, релевантной правам пользователя и параметрам сессии. Полученный результат нормализуется в типизированный ответ и по исходному соединению возвращается клиенту, что обеспечивает аутентифицированность запроса, изоляцию транспортного уровня от доменной логики и детерминированную доставку данных.

Диаграмма на рисунке 31 моделирует сценарий запроса клиентским приложением детализированной информации о задаче, идентифицированной по внешнему идентификатору (например, JIRA-101).

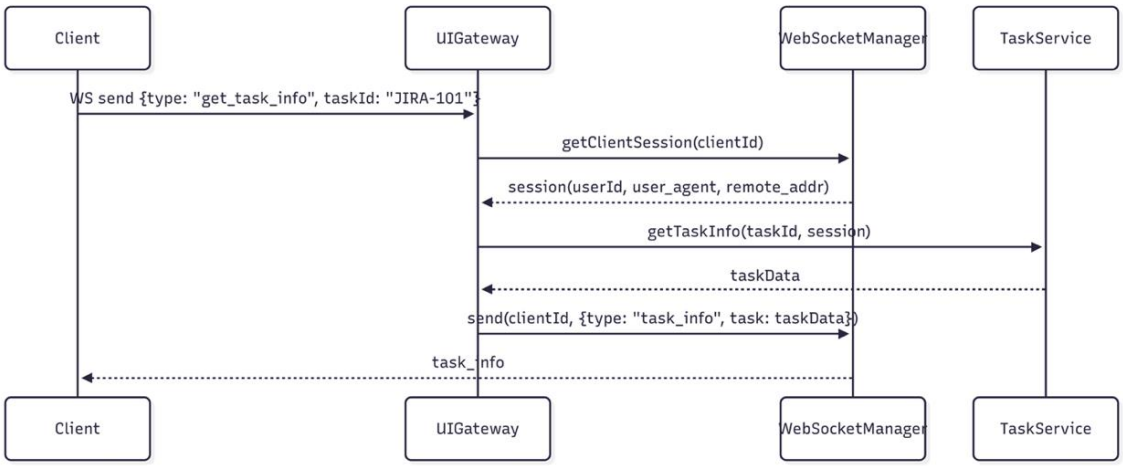

Рисунок 31– Диаграмма последовательности получения информации о задаче

Клиент инициирует WebSocket-запрос с идентификатором задачи. Внешний шлюз запрашивает у менеджера веб-сокетов серверный сеансовый контекст, фиксируя аутентифицированного субъекта. Используя этот контекст, шлюз обращается к сервису задач для выборки сведений о задаче с учетом прав доступа. Полученные данные нормализуются в типизированный ответ и возвращаются по исходному каналу, обеспечивая контекст запроса и изоляцию транспортного уровня от доменной логики.

Диаграмма на рисунке 32 иллюстрирует взаимодействие компонентов распределённой системы при обработке запроса клиента на получение списка чат-ботов и связанных с ними команд в контексте определённого чата.

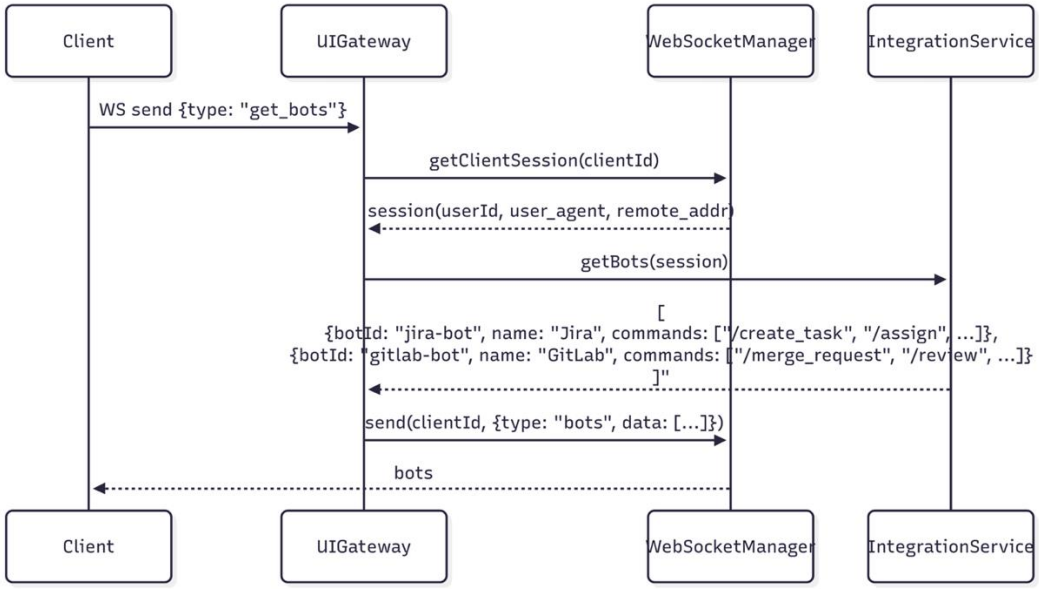

Рисунок 32 – Диаграмма последовательности. Список ботов и команд

Клиент формирует WebSocket-запрос на выдачу перечня ботов и их команд. Внешний шлюз извлекает через менеджера веб-сокетов сеансовый контекст, идентифицируя субъекта и параметры окружения. С использованием этого контекста он обращается к сервису интеграций для выборки доступных ботов и связанных с ними команд в рамках прав пользователя. Результат нормализуется в типизированный ответ и возвращается по исходному соединению, что обеспечивает аутентифицированность запроса, изоляцию транспортного уровня от доменной логики и предсказуемую доставку данных.

Диаграмма на рисунке 33 описывает процесс получения списка участников чата.

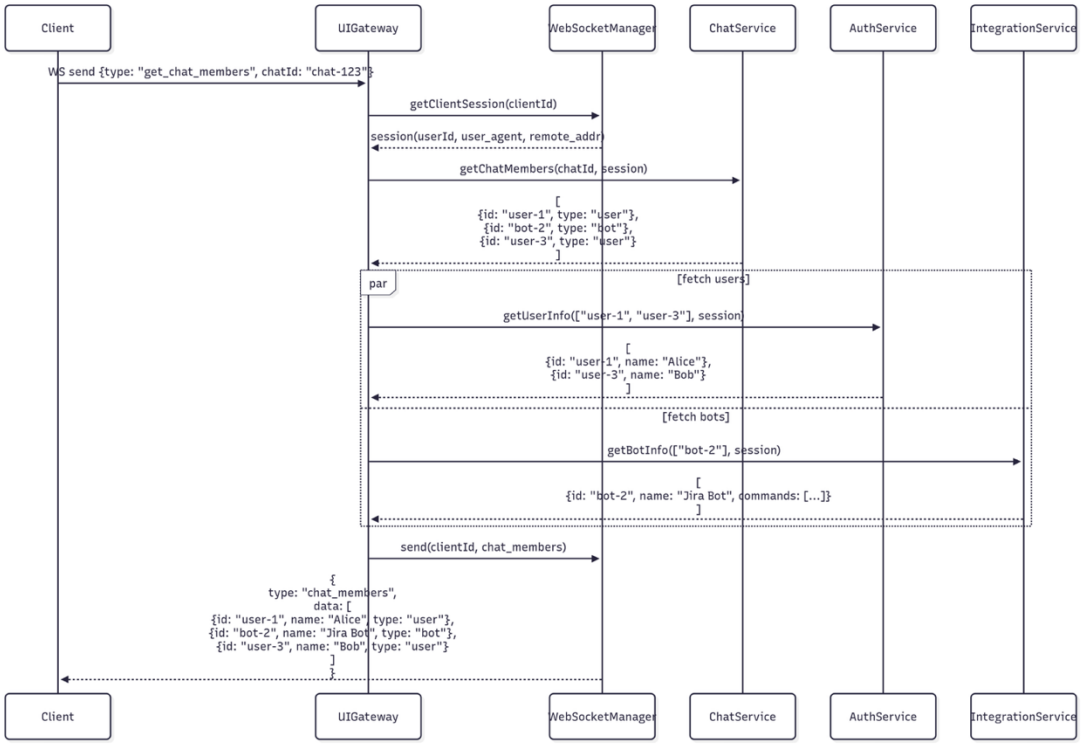

Рисунок 33– Диаграмма последовательности получения списка участников чата

Клиент инициирует WebSocket-запрос на выдачу состава участников выбранного чата. Внешний шлюз восстанавливает сеансовый контекст через менеджера веб-сокетов, тем самым фиксируя аутентифицированного субъекта. Используя контекст и идентификатор чата, шлюз обращается к сервису чатов для получения базового списка членов. Для каждого участника выполняется обогащение атрибутов через профильные источники: сервис аутентификации возвращает пользовательские данные, сервис компаний — должностные и организационные сведения, сервис интеграций — машинные аккаунты и их параметры. Полученная совокупность нормализуется в единый представимый формат и передается клиенту по исходному каналу. Последовательность обеспечивает субъектно ориентированный доступ, консолидацию разнородных данных и строгую изоляцию доменной логики от транспортного уровня.

Диаграмма на рисунке 34 описывает процесс добавления нового участника (пользователя или бота) в чат.

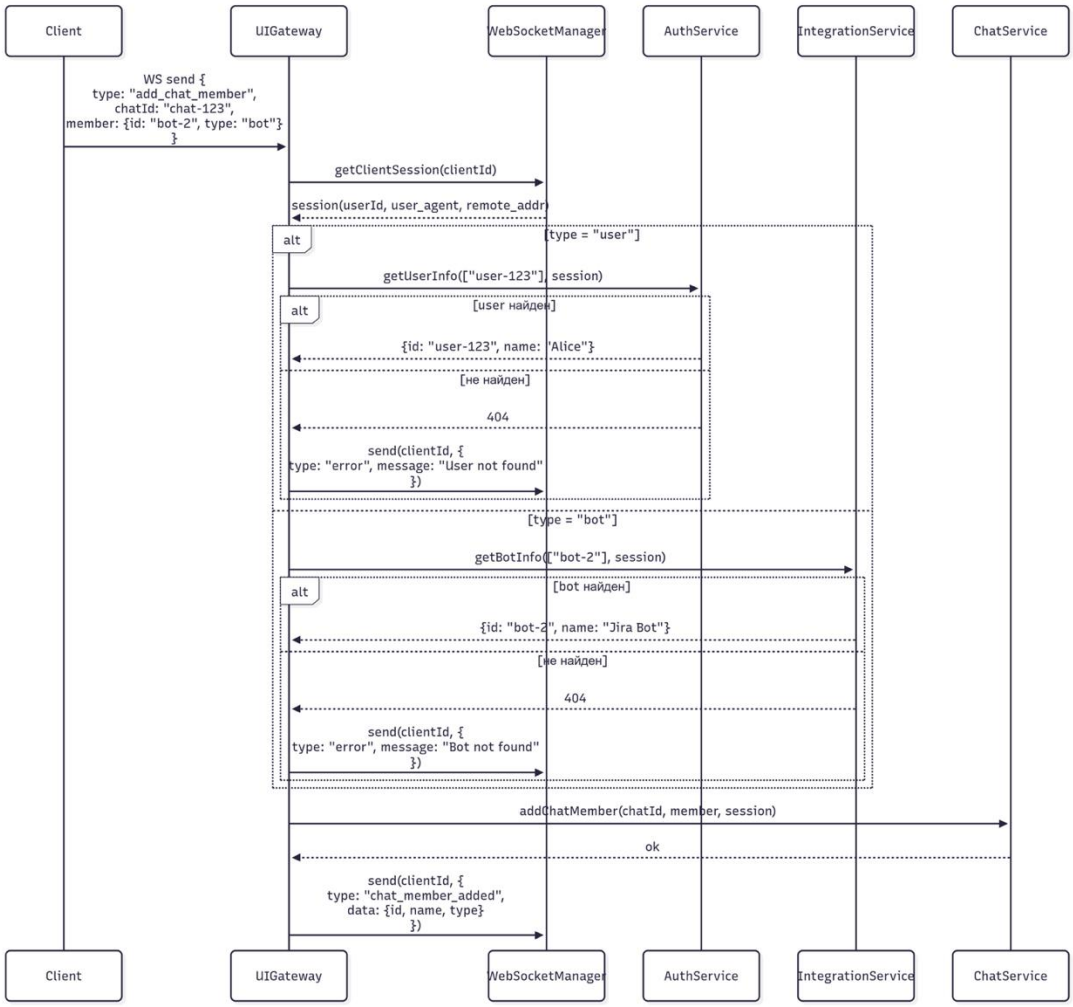

Рисунок 34 – Диаграмма последовательности добавления участника в чат

Клиент инициирует команду добавления участника, передавая идентификатор чата и тип участника (пользователь или бот). Внешний шлюз извлекает сеансовый контекст через менеджера веб-сокетов, тем самым фиксируя аутентифицированного инициатора. Далее выполняется разрешение субъекта: для пользователя — запрос к сервису аутентификации, для бота — к сервису интеграций; при отсутствии целевой сущности формируется детерминированная ошибка «not found». После успешного разрешения шлюз обращается к сервису чатов с командой добавления, где проверяются права инициатора, отсутствие дубликатов и согласованность ролей. Результат

операции нормализуется и возвращается клиенту; при неуспехе передается типизированное сообщение об ошибке с инвариантно заданной семантикой причины. Такая последовательность обеспечивает аутентифицированное и авторизованное изменение состава чата, устраняет гонки за счет централизованной проверки инвариантов и сохраняет изоляцию транспортного уровня от доменной логики.

Диаграмма на рисунке 35 описывает процесс обработки командного сообщения, отправленного пользователем в чат, с последующей интеграцией с внешней системой управления задачами (Jira).

Клиент передает сообщение с типизированной командой; внешний шлюз, опираясь на сеансовый контекст, маршрутизует его в сервис чатов, где сообщение фиксируется как доменное событие и подтверждается отправителю. Обработчик команд извлекает параметризацию (проекта, исполнителя, сроков), разрешает бота и связанный интеграционный профиль, после чего инициирует вызов интеграции. Адаптер интеграции формирует запрос к внешнему трекеру (например, Jira), проводит аутентификацию и создает задачу в удаленной системе, возвращая ее идентификатор и метаданные. На основе ответа формируется запись о внешней задаче и публикуется событие о создании; сервис чатов эмиттирует системное сообщение в ленту с реквизитами новой задачи. Таким образом обеспечиваются: аутентифицированное происхождение команды, атомарная фиксация факта создания, идемпотентная интеграция с внешним API и консистентная обратная связь пользователю внутри исходного чата.

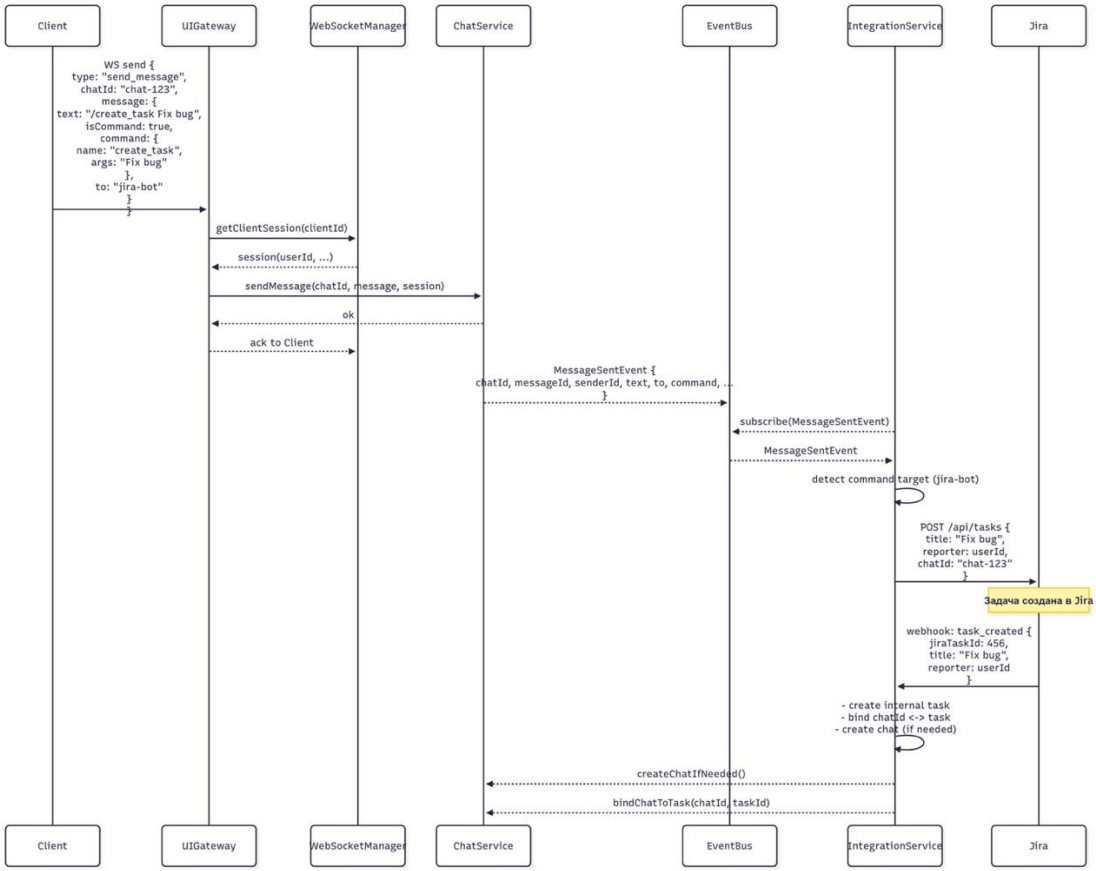

Рисунок 35 – Диаграмма последовательности процесса создания задачи через команду бота

Проведённое моделирование межсервисного взаимодействия позволило сформировать целостное представление о том, как распределённая микросервисная архитектура корпоративного мессенджера обеспечивает обработку запросов, доставку событий и синхронизацию данных между компонентами системы. Последовательные диаграммы, представленные в пункте 2.5, демонстрируют чёткую декомпозицию обязанностей между шлюзом, сервисами доменного уровня, менеджером соединений и шиной событий, а также иллюстрируют принципы асинхронного обмена сообщениями и управления состоянием пользователей.

Анализ взаимодействий подтверждает, что архитектура спроектирована в соответствии с принципами слабой связности и высокой изолированности компонентов: каждый сервис выполняет строго ограниченную роль, а коммуникация осуществляется через протоколы RPC и событийные

сообщения. Такой подход обеспечивает масштабируемость, устойчивость к сбоям и возможность гибкого расширения функциональности системы без нарушения существующих сценариев работы.

Диаграммы наглядно показывают ключевые процессы — установление соединения, доставку новых сообщений, получение истории чата, управление участниками, обработку команд ботов и интеграционные сценарии с внешними системами. Это подтверждает корректность выбранной архитектурной модели и демонстрирует, что система способна стабильно функционировать в условиях высокой нагрузки, распределённой инфраструктуры и необходимости синхронной работы множества сервисов.

Таким образом, разработанные диаграммы межсервисного взаимодействия формируют архитектурный фундамент, необходимый для реализации надёжного, масштабируемого и интегрируемого корпоративного мессенджера, и являются ключевым инструментом для последующей разработки, тестирования и сопровождения системы.

2.6 Реализация и тестирование гибридной схемы сквозного шифрования

В Приложение А представлен код, который реализует протокол сквозного шифрования сообщений в групповом чате на основе гибридной криптосхемы. Данные пользователя моделируются парой ключей RSA, а сообщения защищаются симметричным алгоритмом AES-GCM с уникальным одноразовым ключом на каждое сообщение.

Инициализация включает генерацию индивидуальных RSA-ключей при регистрации субъекта и формирование чата как множества участников, связанных с их открытыми ключами. При отправке сообщения формируется случайный 256-битный симметричный ключ, которым шифруется полезная нагрузка в режиме AEAD (GCM); полученный шифротекст содержит аутентифицирующий тег и префиксируется случайным вектором инициализации (nonce). Далее тот же симметричный ключ распределяется

адресно: для каждого участника выполняется шифрование ключа схемой RSA-OAEP на соответствующем открытом ключе, после чего результат кодируется Base64 и сохраняется в карте адресатов. Таким образом достигается линейная по числу участников стоимость доставки при постоянном размере шифротекста сообщения [19].

Получатель восстанавливает свой симметричный ключ, расшифровывая адресованную ему капсулу RSA-OAEP на личном закрытом ключе, после чего выполняет дешифрование основного шифротекста режимом AES-GCM. Аутентификационное свойство AEAD обеспечивает детектирование любых модификаций как ключевой капсулы, так и полезной нагрузки.

В Приложение представлен листинг кода реализующий тест, который моделирует целостный поток сквозного шифрования в групповом чате на чистом состоянии памяти. На рисунке 36 представлен успешный результат выполнения теста.

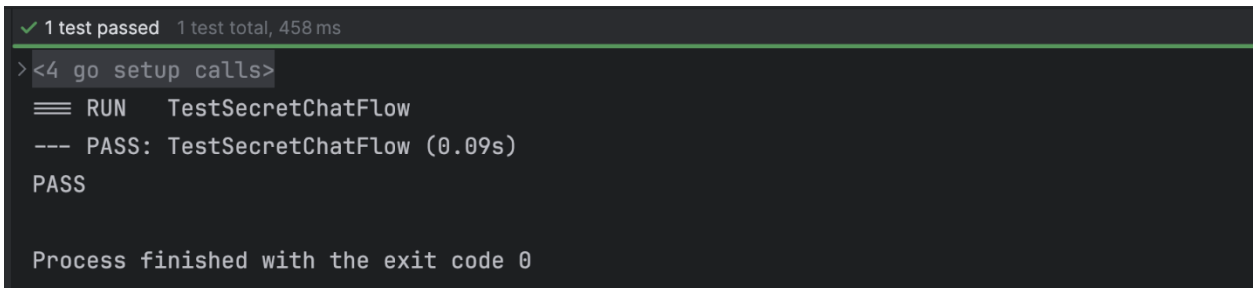

Рисунок 36 – Успешный результат выполнения теста кода программы сквозного шифрования

Сначала регистрируются три пользователя с генерацией пар ключей RSA; наличие открытых ключей служит первичной проверкой корректной инициализации криптографического контекста. Далее формируется секретный чат, и валидируется его состав как необходимое условие для распределения ключей. После этого инициируется отправка сообщения: шифрование полезной нагрузки выполняется режимом AES-GCM, а одноразовый симметричный ключ адресно капсулируется для каждого участника посредством RSA-OAEP. Факт успешной отправки подтверждается

непустым шифротекстом и полнотой набора зашифрованных ключей. Заключительная фаза воспроизводит чтение сообщения каждым участником: индивидуальная капсула ключа расшифровывается личным закрытым ключом пользователя, после чего выполняется дешифрование основного шифротекста; результат сопоставляется с исходной строкой, что подтверждает корректность всей гибридной схемы, включая генерацию ключей, построение чата и доступность расшифровки для всех членов.

2.7 Тестирование

В проекте используется статический анализ исходного кода, позволяющий выявлять потенциальные уязвимости и нарушения стандартов программирования ещё на этапе сборки. Такой подход снижает вероятность попадания критических ошибок в рабочую среду и повышает общий уровень качества программного обеспечения.

Для проектов, реализованных на языке Go, применяются специализированные инструменты статического анализа, среди которых можно выделить golangci-lint и gosec. Первый объединяет набор линтеров, проверяющих стиль кода, использование неинициализированных переменных, избыточные конструкции и ошибки в логике. Второй ориентирован на поиск уязвимостей безопасности: неправильное использование криптографических библиотек, опасные вызовы системных функций, хранение конфиденциальных данных в открытом виде.

Статический анализ интегрирован в CI/CD-процесс, настроенный в GitLab. Для этого в конфигурационный файл .gitlab-ci.yml включён отдельный этап pipeline, автоматически запускающий линтеры и анализаторы при каждом коммите или merge request. В случае выявления ошибок сборка помечается как неуспешная, что предотвращает попадание небезопасного кода в основную ветку разработки. Такой подход обеспечивает непрерывный контроль качества и позволяет оперативно устранять потенциальные уязвимости ещё на стадии разработки.

2.7.1 Системное тестирование

В рамках системного тестирования для проверки функциональной корректности работы приложения разрабатываются отдельные тест-кейсы. Каждый тест-кейс формулируется как самостоятельная единица проверки, включающая цель испытания, предусловия, последовательность шагов, вводимые данные, ожидаемый результат и фиксируемые изменения в базе данных. Такой подход позволяет обеспечить прослеживаемость требований и комплексную оценку качества системы.

В таблице 9 представлен тест-кейс ТС-12 «Авторизация на разных устройствах», отражающий соответствующий сценарий использования.

Таблица 9 – Тест-кейс ТС-12 «Авторизация на разных устройствах»

Название	Авторизация на разных устройствах
Цель	Проверить корректность входа пользователя в систему с различных устройств и синхронизацию данных между ними
Предусловия	Пользователь зарегистрирован; доступны как минимум два устройства (ПК, смартфон)
Входные данные	Email: ivanov@example.com, Пароль: Qwerty123!
Основной поток	Открыть приложение/веб-интерфейс. Ввести корректные логин и пароль.Получить доступ к рабочему интерфейсу. Повторить авторизацию на втором устройстве. Убедиться, что данные синхронизированы.
Изменения в БД	В таблице Sessions создаётся новая запись с уникальным идентификатором сессии, временем начала и привязкой к устройству.В таблице Users обновляется поле last_login с текущим временем.В случае нескольких авторизаций у одного пользователя в БД хранится несколько активных сессий, каждая из которых помечена устройством и временем активности.При смене статуса (например, «онлайн») в таблице Presence обновляется информация для данного пользователя.
Альтернативные потоки	– Введён неверный пароль → запись о новой сессии не создаётся, фиксируется событие неуспешной попытки входа в таблице AuthLogs. – Учётная запись не активирована → отсутствуют изменения в Sessions, создаётся запись в AuthLogs о неудачной авторизации. – Превышен лимит активных устройств → новая запись в Sessions не создаётся, фиксируется отказ в AuthLogs. Нет сети → изменений в БД не происходит.
Ожидаемый результат	Успешная авторизация на разных устройствах; корректная синхронизация данных; в БД отражаются новые активные сессии

Базовым функционалом корпоративного мессенджера является передача сообщений между пользователями в приватном чате. Тест-кейс ТС-1 направлен на проверку корректности доставки сообщений и фиксации их статусов. Подробное содержание представлено в таблице 10.

Таблица 10 – Тест-кейс ТС-5 «Отправка сообщения с вложенным файлом»

Поле	Описание
Цель теста	Проверить корректность отправки и доставки текстового сообщения в приватном диалоге.
Предусловия	Пользователь А и Пользователь В авторизованы в системе, установлен активный чат.
Шаги	1) Пользователь А открывает приватный чат. 2) Вводит текст «Привет». 3) Нажимает «Отправить».
Ожидаемый результат	У пользователя А сообщение появляется в чате; у пользователя В отображается входящее сообщение с отметкой времени.
Альтернативные потоки	– Нет сети → сообщение сохраняется в очередь и отправляется после восстановления. – Получатель оффлайн → сообщение сохраняется на сервере и доставляется при следующем входе.
Изменения в БД	В таблице messages создаётся запись с идентификатором, текстом, отправителем, получателем, временем и статусом sent. После подтверждения обновляется на delivered.

Интеграция мессенджера и PM-системы предполагает автоматическую генерацию чатов для новых задач, что обеспечивает удобный канал обсуждения. Тест-кейс ТС-3 проверяет корректность этого процесса. Подробное описание представлено в таблице 11.

Таблица 11 – Тест-кейс ТС-3. Создание задачи из PM-системы

Поле	Описание
Цель теста	Проверить автоматическое создание чата в мессенджере при создании новой задачи в PM-системе.
Предусловия	Пользователь авторизован в PM-системе и мессенджере; интеграция настроена.
Шаги	Пользователь в PM-системе создаёт новую задачу с названием «Подготовить отчёт». Сохраняет задачу. В мессенджере проверяется наличие нового чата, связанного с задачей.

Продолжение таблицы 11

Ожидаемый результат	В PM-системе создаётся задача; в мессенджере автоматически формируется чат с идентификатором, совпадающим с задачей, и системой отправляется первое служебное сообщение «Чат создан для задачи».
Альтернативные потоки	– Нет связи между системами → чат не создаётся, фиксируется ошибка синхронизации. – У пользователя нет прав на создание задач → операция отклоняется.
Изменения в БД	В PM-системе создаётся запись в tasks с параметрами задачи. В базе данных мессенджера создаётся новая запись в chats, связанная с идентификатором задачи, и в messages добавляется служебное сообщение о создании чата.

Тест-кейс TC-4 проверяет, как сообщение в чате задачи синхронизируется с карточкой задачи. Подробное описание представлено в таблице 12.

Таблица 12 – Тест-кейс TC-4 «Ответ в чат задачи»

Поле	Описание
Цель теста	Проверить корректность отображения ответов в чате задачи и их синхронизации.
Предусловия	Задача создана и доступна; пользователи авторизованы.
Шаги	1) Пользователь открывает чат задачи. 2) Вводит сообщение «Я беру в работу». 3) Отправляет его.
Ожидаемый результат	Сообщение отображается в чате и в комментариях задачи в PM-системе.
Альтернативные потоки	Нет связи с PM-системой → сообщение сохраняется локально и синхронизируется позже.
Изменения в БД	В messages сохраняется сообщение, связанное с задачей. В PM-системе в таблице commentsпоявляется новая запись.

Тест-кейс TC-9 проверяет использование TLS при передаче данных. Подробное описание представлено в таблице 13.

Таблица 13 – Тест-кейс TC-9 «Проверка TLS-шифрования»

Поле	Описание
Цель теста	Проверить, что соединение между клиентом и сервером устанавливается по протоколу TLS.
Предусловия	Сервер настроен с TLS-сертификатом.

Продолжение таблицы 13

Шаги	Пользователь открывает клиент. Подключается к серверу. Отправляет сообщение.
Ожидаемый результат	Трафик шифруется, соединение защищено.
Альтернативные потоки	– Недействительный сертификат → клиент сообщает об ошибке. – Отсутствует поддержка TLS → соединение не устанавливается.
Изменения в БД	Изменения отсутствуют; проверяется лишь зашифрованная передача сообщений.

Тест-кейс ТС-13 фиксирует работу оффлайн-механизма. Подробное описание представлено в таблице 14.

Таблица 14 – Тест-кейс ТС-13 «Отправка сообщения оффлайн»

Поле	Описание
Цель теста	Проверить корректность сохранения и последующей доставки сообщений при отсутствии сети.
Предусловия	Пользователь авторизован; сеть недоступна.
Шаги	1) Пользователь вводит текстовое сообщение «Свяжусь позже». 2) Нажимает «Отправить». 3) Подключение восстанавливается.
Ожидаемый результат	Сообщение отправляется автоматически после восстановления соединения.
Альтернативные потоки	Если сеть не восстанавливается длительное время, сообщение хранится в локальной очереди.
Изменения в БД	Сообщение сохраняется с меткой pending. После доставки обновляется статус на sent и затем delivered.

Для обеспечения отказоустойчивости система поддерживает репликацию данных между дата-центрами. Тест-кейс ТС-14 проверяет корректность синхронизации. Подробности представлены в таблице 15.

Таблица 15 – Тест-кейс ТС-14 «Проверка георасределённого хранения и синхронизации данных между узлами»

Поле	Описание
Цель теста	Проверить корректность синхронизации сообщений и задач при репликации данных между узлами.
Предусловия	Развёрнуты два и более узла системы; между ними настроена репликация.

Продолжение таблицы 15

Шаги	1) Пользователь на узле A отправляет сообщение. 2) На узле B проверяется его отображение. 3) Создаётся задача на узле A. 4) На узле B проверяется её наличие.
Ожидаемый результат	Сообщения и задачи доступны на всех узлах системы без задержек.
Альтернативные потоки	В случае сетевых задержек данные доставляются после восстановления связи.
Изменения в БД	На всех узлах создаются идентичные записи в messages и tasks, синхронизированные через механизм репликации.

2.7.2 Нагрузочное тестирование

Нагрузочное тестирование применяется для оценки производительности системы при интенсивной эксплуатации и выявления предельных значений её устойчивости. В процессе испытаний моделируется одновременная активность сотен и тысяч пользователей, что позволяет проверить корректность доставки сообщений, синхронизации данных и обработки вложенных файлов в условиях повышенной нагрузки.

Для проведения нагрузочного тестирования используются специализированные инструменты. Наиболее распространённым является Apache JMeter, позволяющий эмулировать большое количество одновременных подключений и фиксировать показатели времени отклика, пропускной способности и процента ошибок. В случаях, когда требуется моделирование сетевых задержек и ограничений пропускной способности каналов, применяются такие решения, как Gatling или встроенные механизмы эмуляции сетей (например, tc в Linux). Для тестирования масштабируемости в распределённой среде возможно использование облачных сервисов нагрузочного тестирования, таких как Locust или k6, поддерживающих сценарии с постепенным наращиванием нагрузки.

Результаты фиксируются в отчётах, включающих метрики времени отклика, количество успешных и неуспешных транзакций, использование ресурсов (CPU, память, сетевой трафик) и устойчивость системы к деградации производительности. Такой подход позволяет выявить «узкие места» и

сформировать рекомендации по оптимизации балансировщиков, очередей сообщений и механизмов репликации данных.

2.7.3 Тестирование безопасности

Тестирование безопасности направлено на выявление уязвимостей и подтверждение надёжности реализованных механизмов защиты корпоративного мессенджера. Основной акцент делается на проверке шифрования трафика, корректности разграничения прав доступа, устойчивости к атакам типа «человек посередине» (MITM), SQL-инъекциям и другим видам эксплойтов.

Для анализа сетевого взаимодействия используются инструменты вроде Wireshark, позволяющие перехватывать и анализировать трафик для подтверждения его шифрования по протоколу TLS. Для автоматизированного поиска уязвимостей в веб-интерфейсе применяются сканеры, такие как OWASP ZAP или Burp Suite, которые проверяют корректность обработки запросов, устойчивость к внедрению вредоносных данных и надёжность аутентификации. В случае проверки серверной инфраструктуры могут использоваться фреймворки типа Metasploit, позволяющие моделировать атаки с целью оценки устойчивости к взлому.

Дополнительно проводится проверка хранения данных: используются утилиты анализа состояния баз данных (например, SQLMap для тестирования на инъекции) и механизмы верификации шифрования на уровне файловой системы. Для тестирования многопользовательских сценариев авторизации и управления доступом применяются эмуляторы параллельных подключений и скрипты автоматизации, которые моделируют попытки входа с различных устройств и аккаунтов.

Результаты тестирования безопасности оформляются в виде отчётов с классификацией найденных уязвимостей по критичности (например, по шкале CVSS), а также с рекомендациями по устранению обнаруженных проблем и повышению уровня защищённости.

2.8 Расчет экономической эффективности проекта

Для количественной оценки эффективности проекта использовались стандартные показатели: капитальные вложения, годовой экономический эффект, срок окупаемости и коэффициент экономической эффективности.

Капитальные вложения проекта (K) формировались исходя из фактических затрат на реализацию, включая заработную плату разработчиков, расходы на тестирование, управление проектом и облачную инфраструктуру, а также налоги и накладные расходы. Итоговая сумма капитальных вложений представлена в таблице 16.

Таблица 16 – Капитальные вложения

Статья расходов	Кол-во	Ставка (руб)	Всего (руб)
Заработная плата (разработчики, 2 чел × 4 мес)	2 × 4	150 000	1 200 000
Тестирование (1 чел × 1 мес)	1	100 000	100 000
Менеджмент проекта	1	120 000	120 000
Облачная инфраструктура (серверы, трафик)	4 мес	30 000	120 000
Налоги и накладные расходы (30%)	—	—	474 000
Итого капитальные вложения K	—	—	2 014 000

Годовой экономический эффект (E) рассчитывался как разница между затратами до и после внедрения проекта:

$$E = E_{\text{баз}} - E_{\text{нов}} \quad (1)$$

Подробный расчет представлен в таблице 17.

Таблица 17 – Затратами до и после внедрения проекта

Показатель	До внедрения (руб/год)	После внедрения (руб/год)	Экономия (руб/год)
Затраты на Slack	400 × 50 × 12 = 240 000	0	240 000
Затраты на защищённые каналы связи (VPN, email)	60 000	0	60 000
Трудозатраты на согласование и комментирование задач (0.5 FTE)	400 000	0	400 000
Итого годовой экономический эффект E	700 000	0	700 000

Срок окупаемости проекта (T_0) определяется как отношение капитальных вложений к годовому экономическому эффекту:

$$\mathrm{T}_0 = \frac{K}{E} = \frac{2\,014\,000}{700\,000} \approx 2.88 \text{ года} \tag{2}$$

Коэффициент экономической эффективности ($\mathrm{K}_э$) определяется как обратная величина срока окупаемости:

$$\mathrm{K}_э = \frac{1}{\mathrm{T}_0} = \frac{1}{2.88} \approx 0.347 \tag{3}$$

В результате можно сделать вывод что внедрение корпоративного мессенджера с поддержкой секретных чатов и интеграцией с Jira позволяет существенно снизить операционные расходы на внешние платные мессенджеры, оптимизировать временные и финансовые издержки сотрудников при работе с задачами, а также обеспечивает полное возмещение капитальных вложений менее чем за три года. Коэффициент экономической эффективности существенно превышает минимальное нормативное значение $\mathrm{E}_н = 0.15$, что свидетельствует о целесообразности и экономической оправданности проекта.

Заключение

В ходе выполнения выпускной квалификационной работы была достигнута поставленная цель — разработан прототип географически распределённого корпоративного мессенджера, обеспечивающего безопасное взаимодействие сотрудников и интеграцию с системами управления проектами. Реализация данного решения стала результатом комплексного исследования, включавшего анализ предметной области, проектирование архитектуры, разработку микросервисных компонентов и их тестирование в распределённой инфраструктуре.

В процессе работы проведено исследование существующих корпоративных коммуникационных решений и выявлены их ограничения, связанные с безопасностью, зависимостью от внешних поставщиков и недостаточной интеграцией с внутренними информационными системами. На основе анализа разработана архитектура микросервисного типа, обеспечивающая масштабируемость, отказоустойчивость и гибкость при внедрении новых функций. Реализованы ключевые модули системы, включая сервисы аутентификации, обмена сообщениями и маршрутизации запросов, а также внедрены механизмы сквозного шифрования и секретных чатов, основанные на принципах асимметричной криптографии.

Проведённое тестирование подтвердило надёжность и устойчивость прототипа при работе в распределённой среде, корректность реализации криптографических протоколов и надёжное разграничение прав доступа. Полученные результаты показали, что разработанная система соответствует современным требованиям к корпоративным коммуникационным платформам и обеспечивает необходимый уровень защищённости данных.

Практическая значимость работы заключается в возможности применения созданного решения для организации корпоративной коммуникации с контролем над данными и минимизацией зависимости от сторонних сервисов. Экономическая эффективность подтверждена расчётами,

демонстрирующими снижение затрат на лицензирование и повышение производительности проектных команд.

В ходе разработки были выявлены определённые сложности, связанные с реализацией распределённого хранения и синхронизации данных между географически удалёнными узлами, а также с интеграцией с внешними системами, обладающими различными API и форматами данных. Тем не менее предложенные архитектурные решения позволили минимизировать данные ограничения и обеспечить стабильную работу прототипа.

Перспективы дальнейшего развития связаны с расширением функциональности системы, внедрением дополнительных интеграций с корпоративными сервисами, совершенствованием инструментов администрирования и аналитики, а также оптимизацией мобильных клиентов для работы в условиях ограниченного доступа к сети. Перспективным направлением может стать применение технологий искусственного интеллекта для автоматизации обработки сообщений и поддержки управления задачами.

Таким образом, проведённая работа подтвердила практическую значимость разработанного решения и его соответствие поставленным целям, а также создала основу для дальнейшего развития корпоративного мессенджера как ключевого элемента цифровой инфраструктуры современных распределённых организаций.

Список используемой литературы и используемых источников

1. Буч Г., Рамбо Дж., Джекобсон И. UML. Унифицированный язык моделирования: руководство пользователя. 2-е изд. М.: Вильямс, 2020. 496 с.
2. Гребенников С. Г., Соловьёв А. С. Безопасность информационных систем. М.: Академия, 2021. 288 с.
3. Дейт К. Дж. Введение в системы баз данных. 8-е изд. М.: Вильямс, 2021. 864 с.
4. Камалов Р. Микросервисная архитектура на практике. СПб.: Питер, 2022. 320 с.
5. Кормен Т. Х., Лейзерсон Ч. Е., Ривест Р. Л., Штайн К. Алгоритмы: построение и анализ. 3-е изд. М.: Вильямс, 2020. 1328 с.
6. Куликов А. А. Основы построения распределённых вычислительных систем. М.: ИНФРА-М, 2021. 296 с.
7. Лисовский В. А. Технологии проектирования распределённых систем. М.: ИНФРА-М, 2023. 284 с.
8. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. М.: Питер, 2021. 352 с.
9. Мельников А. В. Безопасные распределённые приложения и сервисы. СПб.: БХВ-Петербург, 2020. 304 с.
10. Поликарпов А. Распределённые вычисления и системы реального времени. М.: Наука, 2022. 312 с.
11. Постел Дж. RFC 6455 – The WebSocket Protocol. IETF, 2011. URL: https://datatracker.ietf.org/doc/html/rfc6455 (дата обращения: 01.06.2025).
12. Романенко А. В. Методы и средства защиты информации в распределённых системах. М.: Горячая линия — Телеком, 2020. 224 с.
13. Соммервилл И. Инженерия программного обеспечения. 10-е изд. СПб.: Питер, 2021. 816 с.
14. Столлингс У. Криптография и защита сетей. 8-е изд. М.: Вильямс, 2022. 784 с.

15. Таненбаум Э., Везеролл Д. Сетевые технологии. 5-е изд. СПб.: Питер, 2020. 960 с.

16. Хант Э., Томас Д. Программист-прагматик. Путь от подмастерья к мастеру. 2-е изд. СПб.: Питер, 2020. 352 с.

17. Шлейфер Д. Архитектура микросервисов. Проектирование и эволюция распределённых систем. М.: ДМК Пресс, 2022. 384 с.

18. RFC 8446 – The Transport Layer Security (TLS) Protocol Version 1.3. IETF, 2018. URL: https://datatracker.ietf.org/doc/html/rfc8446 (дата обращения: 01.06.2025).

19. Signal Protocol Specification. Open Whisper Systems. URL: https://signal.org/docs/ (дата обращения: 01.06.2025).

20. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston: Addison-Wesley, 2021. 544 p.

21. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley, 2004. 560 p.

22. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2003. 560 p.

23. Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Boston: Addison-Wesley, 2019. 448 p. 24. Hohpe G., Woolf B. Enterprise Integration Patterns. Boston: Addison Wesley, 2012. 736 p.

25. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O’Reilly Media, 2021. 412 p.

26. Richards M., Ford N. Fundamentals of Software Architecture. Sebastopol: O’Reilly Media, 2020. 376 p.

27. Richards M. Software Architecture: The Hard Parts. Sebastopol: O’Reilly Media, 2022. 312 p.

28. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. 2nd ed. New York: Wiley, 2020. 784 p.

Приложение А

Листинг кода программы, реализующей пример использования алгоритма сквозного шифрования

```
package main
import (
          "bytes"
          "crypto/aes"
          "crypto/cipher"
          "crypto/rand"
          "crypto/rsa"
          "crypto/sha256"
      "encoding/base64"
          "errors"
          "fmt"
          "io"
          "sync"
)
type User struct {
          ID        string
          PrivKey   *rsa.PrivateKey
          PubKey    *rsa.PublicKey
}
type Chat struct {
    ID          string
    Participants map[string]*User // userID -> User
}
type Message struct {
          ChatID           string
          SenderID         string
          EncryptedText    string
          EncryptedKeysFor map[string]string // userID -> encrypted AES key (base64)
}
var (
          users    = map[string]*User{}
          chats    = map[string]*Chat{}
          messages = []Message{}
          mu       sync.Mutex
)

// --- Регистрация пользователя --- //
func registerUser(id string) (*User, error) {
          priv, err := rsa.GenerateKey(rand.Reader, 2048)
          if err != nil {
        return nil, err
    }

    user := &User{
        ID:      id,
              PrivKey: priv,
        PubKey:  &priv.PublicKey,
          }

    mu.Lock()
          defer mu.Unlock()

    users[id] = user
```

```
 return user, nil
}

// --- Создание чата с участниками --- //
func createSecretChat(chatID string, participantIDs []string) (*Chat, error) {
    chat := &Chat{
        ID:          chatID,
        Participants: map[string]*User{},
    }
    mu.Lock()
    defer mu.Unlock()

    for _, uid := range participantIDs {
        user, ok := users[uid]
               if !ok {
            return nil, fmt.Errorf("user %s not found", uid)
        }
        chat.Participants[uid] = user
    }
           chats[chatID] = chat
    return chat, nil
}

// --- Шифрование сообщения --- //

// Генерация случайного AES-256 ключа
func generateAESKey() ([]byte, error) {
          key := make([]byte, 32) // 256 бит
          _, err := rand.Read(key)
          return key, err
}

// AES-GCM шифрование
func encryptAESGCM(key, plaintext []byte) (string, error) {
          block, err := aes.NewCipher(key)
          if err != nil {
        return "", err
          }

          aesGCM, err := cipher.NewGCM(block)
          if err != nil {
        return "", err
          }

          nonce := make([]byte, aesGCM.NonceSize())
          if _, err := io.ReadFull(rand.Reader, nonce); err != nil {
        return "", err
          }

          ciphertext := aesGCM.Seal(nonce, nonce, plaintext, nil)
          return base64.StdEncoding.EncodeToString(ciphertext), nil
}

// RSA OAEP шифрование (ключа AES)
func encryptRSAOAEP(pub *rsa.PublicKey, data []byte) (string, error) {
       label := []byte("") // can be empty
           hash := sha256.New()
       encrypted, err := rsa.EncryptOAEP(hash, rand.Reader, pub, data, label)
```

```
  if err != nil {
        return "", err
          }
          return base64.StdEncoding.EncodeToString(encrypted), nil
}

// Отправка сообщения
func sendMessage(chatID, senderID string, plaintext string) (*Message, error) {
          mu.Lock()
          chat, ok := chats[chatID]
          mu.Unlock()
    if !ok {
              return nil, errors.New("chat not found")
          }

          // Генерируем AES ключ для сообщения
          aesKey, err := generateAESKey()
          if err != nil {
        return nil, err
          }

          // Шифруем текст сообщения AES
          encryptedText, err := encryptAESGCM(aesKey, []byte(plaintext))
          if err != nil {
        return nil, err
    }

          // Для каждого участника шифруем AES ключ публичным ключом
          encryptedKeys := map[string]string{}
          for uid, user := range chat.Participants {
        encKey, err := encryptRSAOAEP(user.PubKey, aesKey)
        if err != nil {
            return nil, err
        }
        encryptedKeys[uid] = encKey
          }

          msg := Message{
                ChatID:           chatID,
                SenderID:         senderID,
                EncryptedText:    encryptedText,
                EncryptedKeysFor: encryptedKeys,
          }

          mu.Lock()
          messages = append(messages, msg)
          mu.Unlock()

          return &msg, nil
}

// --- Расшифровка сообщения --- //
func decryptRSAOAEP(priv *rsa.PrivateKey, base64data string) ([]byte, error) {
          data, err := base64.StdEncoding.DecodeString(base64data)
          if err != nil {
                return nil, err
          }
          label := []byte("")
```

```
 hash := sha256.New()
	return rsa.DecryptOAEP(hash, rand.Reader, priv, data, label)
}

func decryptAESGCM(key []byte, base64ciphertext string) (string, error) {
	data, err := base64.StdEncoding.DecodeString(base64ciphertext)
	if err != nil {
		return "", err
	}

	block, err := aes.NewCipher(key)
	if err != nil {
		return "", err
	}

	aesGCM, err := cipher.NewGCM(block)
	if err != nil {
		return "", err
	}

	nonceSize := aesGCM.NonceSize()
	if len(data) < nonceSize {
		return "", errors.New("ciphertext too short")
	}

	nonce, ciphertext := data[:nonceSize], data[nonceSize:]
	plaintext, err := aesGCM.Open(nil, nonce, ciphertext, nil)
	return string(plaintext), err
}

// Получение и расшифровка сообщения для пользователя
func readMessage(userID string, msg *Message) (string, error) {
	mu.Lock()
	user, ok := users[userID]
	mu.Unlock()
	if !ok {
		return "", errors.New("user not found")
	}

	encAESKey, ok := msg.EncryptedKeysFor[userID]
	if !ok {
		return "", errors.New("no encrypted key for user")
	}

	// Расшифровка AES ключа
	aesKey, err := decryptRSAOAEP(user.PrivKey, encAESKey)
	if err != nil {
		return "", err
	}

	// Расшифровка текста
	return decryptAESGCM(aesKey, msg.EncryptedText)
}

func main() {
	// Регистрация пользователей
	alice, _ := registerUser("alice")
	bob, _ := registerUser("bob")
```

```
carol, _ := registerUser("carol")

	fmt.Println("Пользователи зарегистрированы.")

	// Создание секретного чата с 3 участниками
	chat, err := createSecretChat("chat1", []string{"alice", "bob", "carol"})
	if err != nil {
		panic(err)
	}
	fmt.Println("Секретный чат создан с участниками:",
chat.Participants)

	// Отправка зашифрованного сообщения
	msg, err := sendMessage("chat1", "alice", "Привет, секретный чат!")
	if err != nil {
		panic(err)
	}
	fmt.Println("Сообщение отправлено и зашифровано.")

	// Чтение сообщения участниками
	for _, user := range []*User{alice, bob, carol} {
		text, err := readMessage(user.ID, msg)
		if err != nil {
			fmt.Printf("Ошибка при чтении сообщения %s: %v\n", user.ID,
err)
			continue
		}
		fmt.Printf("Пользователь %s прочитал сообщение: %s\n", user.ID,
text)
	}
}
```

Приложение F

Листинг кода программы, реализующей тестирование примера использования алгоритма сквозного шифрования

```
package main
import (	"testing")
func TestSecretChatFlow(t *testing.T) {
	// Очистка глобального состояния
	users = map[string]*User{}
	chats = map[string]*Chat{}
	messages = []Message{}
	// Шаг 1: Регистрация пользователей
	alice, err := registerUser("alice")
	if err != nil {
		t.Fatalf("failed to register alice: %v", err)
	}
	bob, err := registerUser("bob")
	if err != nil {
		t.Fatalf("failed to register bob: %v", err)
	}
	carol, err := registerUser("carol")
	if err != nil {
		t.Fatalf("failed to register carol: %v", err)
	}
	// Проверка публичных ключей
	if alice.PubKey == nil || bob.PubKey == nil || carol.PubKey == nil {
		t.Fatal("public key is nil after registration")
	}
	// Шаг 2: Создание секретного чата
	chat, err := createSecretChat("chat-test", []string{"alice", "bob",
"carol"})
	if err != nil {
		t.Fatalf("failed to create chat: %v", err)
	}
	if len(chat.Participants) != 3 {
		t.Fatalf("chat should have 3 participants, got %d",
len(chat.Participants))
	}
	// Шаг 3: Отправка сообщения
	plaintext := "Test secret message"
	msg, err := sendMessage("chat-test", "alice", plaintext)
	if err != nil {
		t.Fatalf("failed to send message: %v", err)
	}
	if msg.EncryptedText == "" {
		t.Fatal("encrypted message text is empty")
	}
	if len(msg.EncryptedKeysFor) != 3 {
		t.Fatalf("expected 3 encrypted keys, got %d",
len(msg.EncryptedKeysFor))
	}
	// Шаг 4: Проверка расшифровки каждым участником
	for _, user := range []*User{alice, bob, carol} {
		text, e rr := readMessage(user.ID, msg)
		if err != nil {
			t.Errorf("failed to read message for user %s: %v",
user.ID, err)
		}		if text != plaintext {
			t.Errorf("wrong plaintext for user %s: got '%s', want
'%s'", user.ID, text, plaintext)
		}
```