

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Веб-приложение корпоративного мессенджера
для конечного пользователя

Студент

Сахно Никита Сергеевич
(подпись, дата) (ФИО полностью)

Руководитель

Гурьянов Л.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

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

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

Заведующий кафедрой

Козлов А.Ю.
(подпись) (фамилия, инициалы)

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

Секретарь ГЭК

Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2026

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

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

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Сахно Никита Сергеевич гр. 22ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Веб-приложение корпоративного мессенджера для конечного пользователя

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

3. Руководитель работы к.т.н. доцент Гурьянов Л.В.

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

1) *Назначение: веб приложение корпоративного мессенджера*

2) *Среда выполнения: Windows 10.*

Языковые средства разработки: Go, TypeScript, UML.

Инструментальные средства: Visual Studio Code, PostgreSQL,

Visual Paradigm 17.2.

*Технологии разработки: клиент – серверная архитектура,
микросервисная архитектура.*

3) *Основные функции приложения:*

а) регистрация и авторизация;

б) вход в рабочее пространство;

в) создание и управление задачами в рабочем пространстве;

г) общение в рабочем пространстве.

5. Перечень подлежащих разработке вопросов:

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

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 99 страниц, 15 рисунков, 11 таблиц, 22 использованных источника и 8 приложений.

КОРПОРАТИВНЫЙ МЕССЕНДЖЕР, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, ВЕБ-ПРИЛОЖЕНИЕ, GO, TYPESCRIPT, REACT, POSTGRESQL, WEBSOCKET, REST API

Объект исследования – корпоративный мессенджер с поддержкой управления задачами и рабочими пространствами.

Цель выпускной квалификационной работы – спроектировать и разработать программное обеспечение корпоративного мессенджера, обеспечивающего обмен сообщениями в режиме реального времени, управление рабочими пространствами и задачами в рамках единой платформы.

Технологии разработки – язык программирования Go для реализации серверной части, язык программирования TypeScript и библиотека React для клиентского приложения, СУБД PostgreSQL для хранения данных, среда разработки Visual Studio Code.

Среда исполнения – ОС Windows 10, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы разработан корпоративный мессенджер на основе клиент – серверной архитектуры с поддержкой обмена сообщениями в режиме реального времени, управления рабочими пространствами, задачами и пользователями.

					ПГУ 09.03.04 – 08ВР221.18 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Веб-приложение корпоративного мессенджера для конечного пользователя Пояснительная записка	Лит.	Лист	Листов
Разраб.		Сахно Н.С.						
Пров.		Гурьянов Л.В.					4	99
Реценз.						Группа 22ВП1		
Н. Контр.		Такташкин Д.В.						
Утв.		Козлов А.Ю.						

Содержание

Введение	7
1 Анализ предметной области и постановка задачи на разработку программного обеспечения корпоративного мессенджера	9
1.1 Понятие корпоративного мессенджера и управления рабочими процессами в организации	9
1.2 Анализ информационных систем аналогичного назначения	10
1.2.1 Мессенджер Slack	11
1.2.2 Платформа Битрикс24	12
1.2.3 Мессенджер Пачка.....	13
1.2.4 Сравнительный анализ аналогов	14
1.3 Описание предметной области и ее участников.....	15
1.4 Анализ требований	16
1.4.1 Анализ функциональных требований.....	16
1.4.2 Анализ нефункциональных требований.....	19
1.5 Планирование разработки и оценка бюджета.....	21
2 Проектирование корпоративного мессенджера.....	23
2.1 Архитектура приложения	23
2.2 Разработка концептуальной схемы базы данных	26
2.3 Логическое моделирование базы данных.....	29
2.4 Обобщенная структура разрабатываемого сервиса.....	31
2.5 Проектирование клиентской части	34
2.6 Требования к аппаратному обеспечению серверной части	36
3 Разработка корпоративного мессенджера	38
3.1 Реализация базы данных	38
3.2 Реализация клиентской части приложения	39
3.3 Реализация серверной части приложения	41
4 Контроль качества программного обеспечения	45
4.1 Обоснование выбора модели тестирования и метрик кода	45

	6
4.2 Тестирование корпоративного мессенджера	46
4.2.1 Функциональное тестирование создания задачи.....	46
4.2.2 Интеграционное тестирование модуля управления задачами	49
4.2.3 Анализ метрик качества кода	50
Заключение	53
Список использованных источников	55
Приложение А (справочное) Глоссарий.....	57
Приложение Б (рекомендуемое) Спецификации концептуальной схемы базы данных.....	59
Приложение В (рекомендуемое) Графическая часть	66
Приложение Г (обязательное) Листинг серверной части приложения	70
Приложение Д (обязательное) Листинг клиентской части приложения.....	82
Приложение Е (обязательное) Листинг инструмента для подсчета метрик качества кода.....	89
Приложение Ж (рекомендуемое) Результаты функционального тестирования	93
Приложение З (рекомендуемое) Акт внедрения программного средства в эксплуатацию	98

Введение

В современных условиях развития корпоративной среды особую актуальность приобретают вопросы организации внутреннего взаимодействия сотрудников, управления рабочими пространствами и контроля выполнения задач. Для эффективной работы компании недостаточно использовать только разрозненные каналы связи, так как это приводит к потере информации, дублированию сообщений, отсутствию единой истории обсуждений и сложностям при распределении ответственности между участниками рабочих процессов.

Существующие способы коммуникации, основанные на использовании отдельных мессенджеров, электронной почты и ручного контроля задач, не всегда отвечают требованиям оперативности, безопасности и управляемости. Отсутствие единой системы для общения, назначения задач, управления ролями пользователей и обработки обращений снижает прозрачность рабочих процессов и усложняет администрирование корпоративной среды.

Разрабатываемая информационная система корпоративного мессенджера ориентирована на объединение основных процессов внутреннего взаимодействия в единой цифровой среде. В рамках такой системы пользователи получают возможность обмениваться сообщениями, работать в отдельных рабочих пространствах, участвовать в обсуждениях, получать уведомления, создавать и выполнять задачи, а также взаимодействовать с администраторами при возникновении организационных или технических вопросов. Это позволяет снизить зависимость от сторонних сервисов коммуникации и сформировать контролируемую корпоративную среду, соответствующую внутренним требованиям организации.

Актуальность разработки также обусловлена необходимостью повышения прозрачности рабочих процессов внутри организации. Наличие единой платформы позволяет фиксировать историю коммуникации, отслеживать назначенные задачи, определять ответственных исполнителей и контролировать выполнение действий в рамках конкретных рабочих пространств. В результате

уменьшается вероятность потери важной информации, повышается скорость обмена данными между сотрудниками и упрощается администрирование пользователей, ролей и обращений. Таким образом, создание серверного приложения корпоративного мессенджера является практически значимой задачей, направленной на повышение эффективности внутреннего взаимодействия и автоматизацию организационных процессов.

В связи с этим возникает необходимость разработки специализированного программного средства, обеспечивающего централизованную коммуникацию пользователей, управление рабочими пространствами, ведение задач и администрирование системы в рамках единой информационной платформы.

Целью выпускной квалификационной работы является разработка серверного приложения информационной системы корпоративного мессенджера, обеспечивающей автоматизацию ключевых процессов внутреннего взаимодействия пользователей организации.

1 Анализ предметной области и постановка задачи на разработку программного обеспечения корпоративного мессенджера

1.1 Понятие корпоративного мессенджера и управления рабочими процессами в организации

Корпоративный мессенджер представляет собой программную систему для обмена сообщениями и сопутствующей информацией между сотрудниками внутри одной организации или связанных с ней контуров доступа. В отличие от публичных мессенджеров он ориентирован на рабочий контекст: идентификацию пользователей, разграничение прав, привязку общения к структуре компании и к конкретным рабочим единицам, таким как проекты, отделы или рабочие пространства. В состав таких систем обычно входят текстовые чаты, иногда голосовые и видеокommunikации, уведомления, поиск по переписке и интеграция с другими корпоративными сервисами.

Управление рабочими процессами в организации включает постановку целей, распределение задач между исполнителями, контроль сроков и статусов, фиксацию договоренностей и обратную связь между участниками. Эффективность процесса зависит от того, насколько быстро и прозрачно передается информация, кто имеет право на какие действия и как связаны обсуждения с конкретными поручениями. В корпоративной среде часто требуется совмещение коммуникации и учета работ: чтобы переписка не расходилась с фактическим планом, а задачи не «жили» отдельно от контекста обсуждения.

Рабочее пространство в данном контексте — логическая или организационная область, в рамках которой пользователи объединены общими правилами доступа, чатами и задачами. Оно позволяет изолировать данные разных подразделений или проектов, назначать роли участникам и задавать политику взаимодействия внутри границ этого пространства. Для руководителя рабочего пространства важны функции администрирования состава участников,

настройки параметров и контроля соответствия деятельности участников целям пространства.

Контроль выполнения задач – комплекс действий по созданию задач, назначению исполнителей, изменению статусов, фиксации сроков и истории изменений. Он обеспечивает прослеживаемость: кто что сделал, когда задача была создана, завершена или отклонена, какие изменения вносились. Связь задач с чатами позволяет привязать обсуждение к конкретному поручению и снизить потери контекста при передаче работы между людьми.

Автоматизированная система корпоративного мессенджера с функциями управления задачами и рабочими пространствами – это система, которая объединяет каналы коммуникации, модель участия пользователей в организационных единицах, управление задачами и при необходимости административные функции в единой среде доступа. Она используется для ускорения согласований, снижения фрагментации инструментов, повышения управляемости прав доступа и для того, чтобы операционная деятельность сотрудников опиралась на согласованные данные о чатах, участниках и задачах, а не на разрозненные каналы.

1.2 Анализ информационных систем аналогичного назначения

Проведем обзор и анализ существующих программных продуктов со схожим функционалом.

На рынке представлены как отечественные, так и зарубежные решения для организации корпоративной коммуникации и управления рабочими процессами внутри компаний. По результатам анализа будут выявлены преимущества и недостатки каждого рассмотренного решения применительно к разрабатываемой системе корпоративного мессенджера.

Всего были рассмотрены 3 системы: мессенджер «Slack», CRM платформа «Битрикс24» и мессенджер «Пачка».

1.2.1 Мессенджер Slack

Мессенджер «Slack» представляет собой популярную зарубежную платформу для корпоративного общения и совместной работы сотрудников [1].

Графический интерфейс сервиса представлен на рисунке 1.

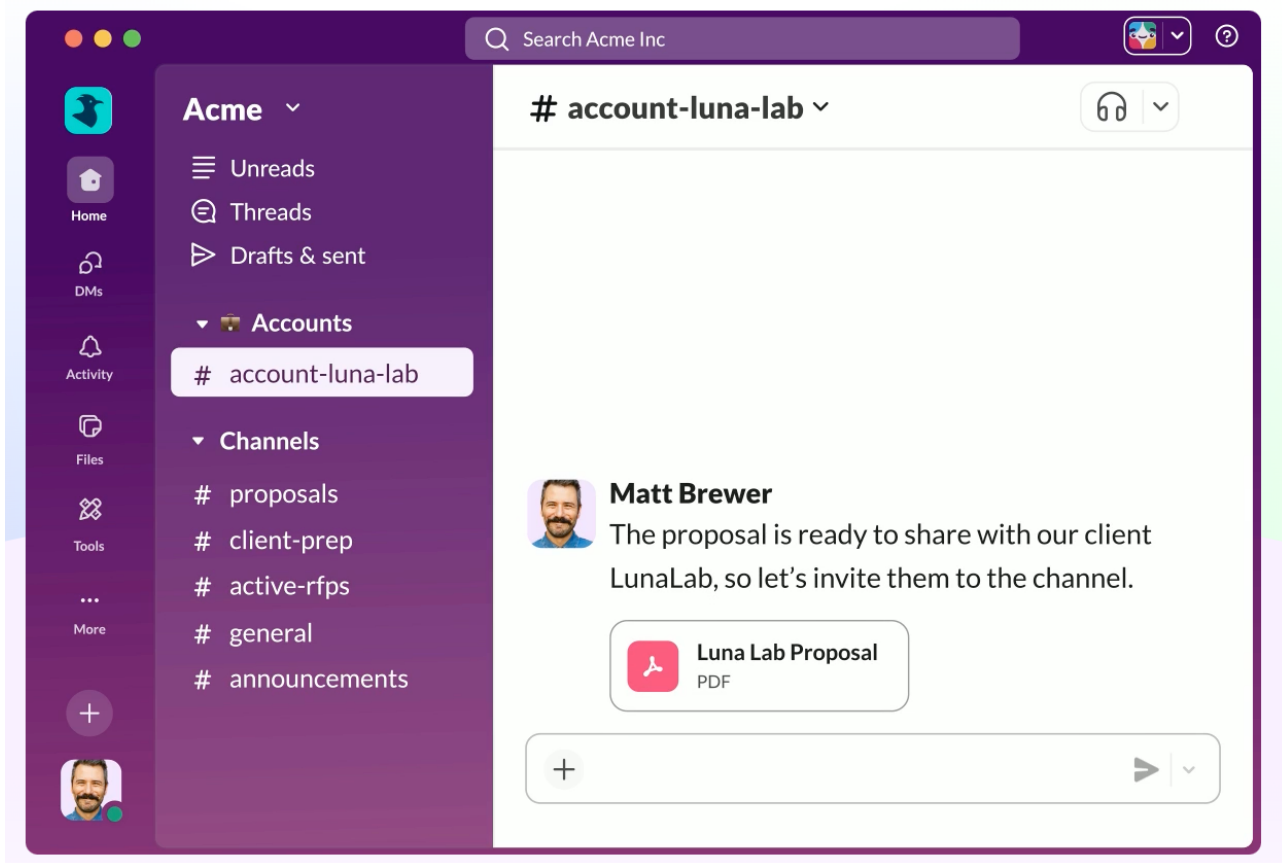


Рисунок 1 – Интерфейс сервиса «Slack»

Сервис предоставляет возможность обмена сообщениями, создания тематических каналов, организации видеоконференций и интеграции со сторонними сервисами.

Основными преимуществами платформы являются развитая система коммуникации между сотрудниками, поддержка интеграций с внешними сервисами и удобный пользовательский интерфейс. Также платформа поддерживает создание рабочих пространств, распределение ролей пользователей и обмен файлами между участниками рабочих групп.

Однако у системы имеются и недостатки. Платформа ориентирована на зарубежный рынок и официально не поддерживает российских пользователей. В

системе отсутствует встроенный механизм контроля выполнения задач и централизованного управления рабочими пространствами в том виде, который требуется для разрабатываемой системы.

1.2.2 Платформа Битрикс24

CRM платформа «Битрикс24» является отечественной платформой для организации корпоративной коммуникации, управления проектами и автоматизации бизнес-процессов. Система включает в себя корпоративный чат, видеозвонки, CRM, инструменты управления задачами и документооборот [2].

Графический интерфейс сервиса представлен на рисунке 2.

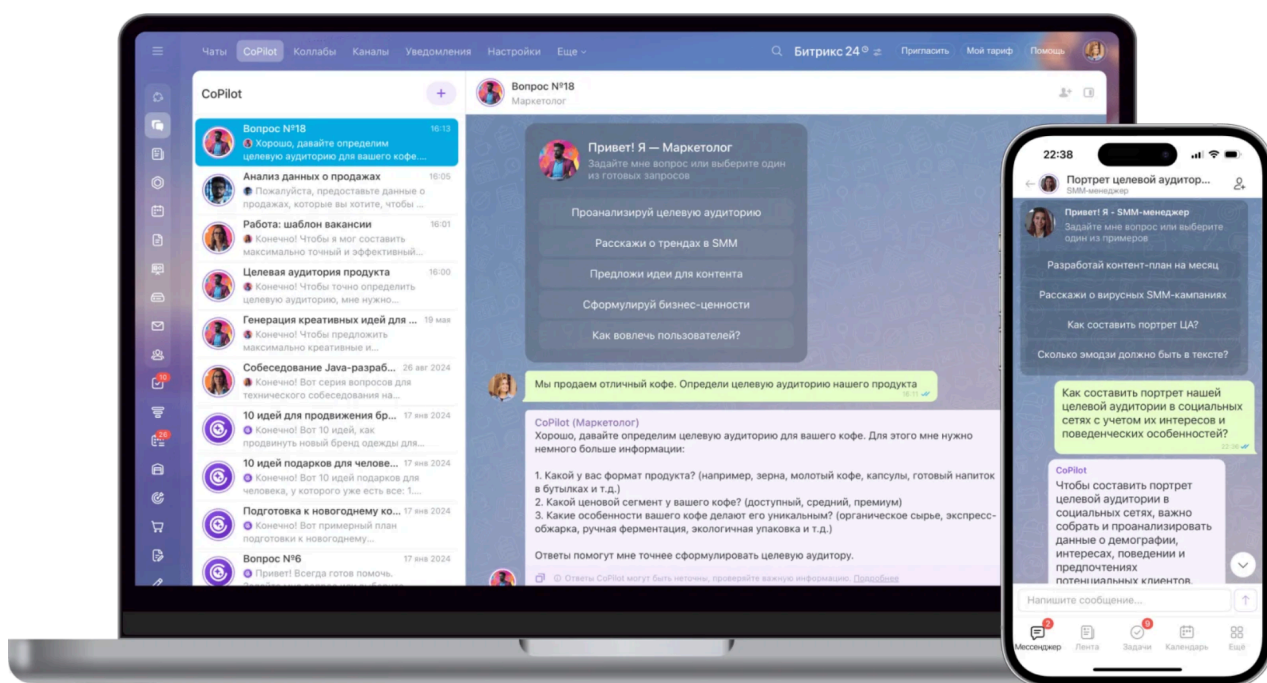


Рисунок 2 – Интерфейс сервиса «Битрикс24»

К преимуществам платформы можно отнести широкий набор инструментов для совместной работы сотрудников, наличие встроенной системы управления задачами и развитые средства администрирования пользователей. Платформа поддерживает создание рабочих групп и организацию внутреннего взаимодействия сотрудников компании.

Недостатком решения является высокая сложность интерфейса и избыточность функционала для небольших организаций.

1.2.3 Мессенджер Пачка

«Пачка» представляет собой российский корпоративный мессенджер, предназначенный для коммуникации сотрудников внутри компании. Платформа поддерживает создание чатов, видеозвонков, обмен файлами и управление командами [3]. Графический интерфейс сервиса представлен на рисунке 3.

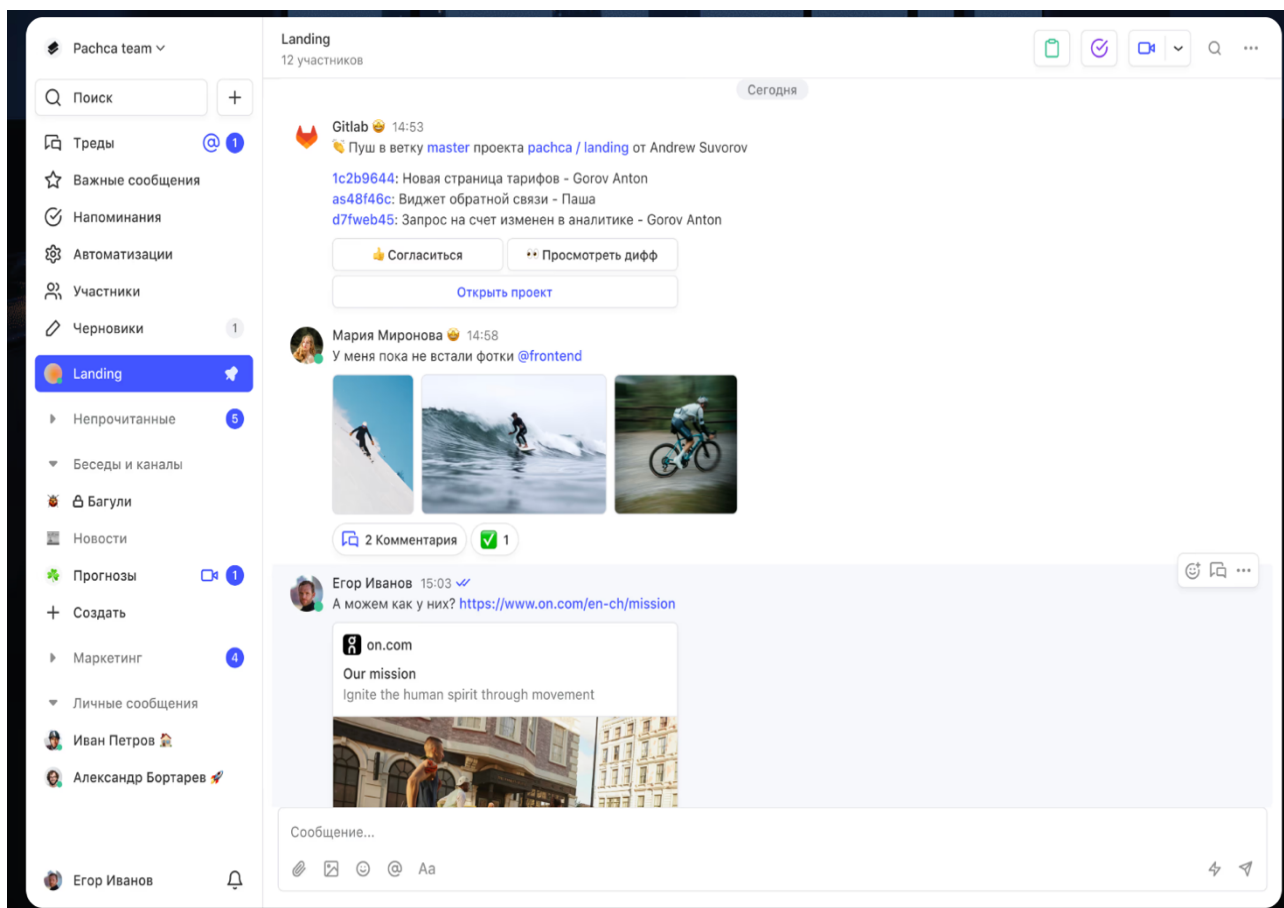


Рисунок 3 – Интерфейс сервиса Пачка

Среди преимуществ платформы можно выделить простоту использования, наличие русскоязычного интерфейса и поддержку базовых функций корпоративного общения. Сервис ориентирован на российский рынок и предоставляет удобные инструменты для организации рабочих коммуникаций.

К недостаткам платформы относится ограниченный набор инструментов для аналитики и контроля задач. Также отсутствуют расширенные механизмы управления рабочими пространствами и централизованного хранения информации о действиях пользователей.

1.2.4 Сравнительный анализ аналогов

В рамках сравнительного анализа был сформирован перечень критериев, по которым оцениваются существующие аналоги и разрабатываемое приложение. Для количественной оценки каждого критерия применяется следующая шкала:

- 0 баллов – критерий не выполняется;
- 1 балл – критерий выполняется частично;
- 2 балла – критерий выполняется в полной мере.

Итоги сравнительного анализа информационных систем аналогичного назначения представлены в таблице 1.

Таблица 1 – Сравнительный анализ информационных систем аналогичного назначения

Критерий	Slack	Битрикс24	Пачка
Поддержка рабочих нескольких пространств	2	2	2
Управление ролями пользователей	2	2	2
Система управления задачами	1	2	0
Контроль выполнения задач	0	2	0
API для интеграции	2	1	2
Формирование аналитики и отчетности	1	2	0
Итого	8	11	6

Таким образом, по результатам сравнительного анализа можно сделать вывод, что существующие программные решения лишь частично удовлетворяют требованиям, предъявляемым к системе корпоративного мессенджера с функциями управления рабочими пространствами и задачами. Большинство рассмотренных платформ ориентированы либо исключительно на коммуникацию, либо обладают избыточным функционалом и сложностью внедрения. Следовательно, разработка собственной информационной системы является актуальной и целесообразной.

1.3 Описание предметной области и ее участников

В качестве предметной области ИС рассматривается коммуникация между сотрудниками внутри компании. Среди участников данного процесса можно выделить:

- сотрудников;
- руководителей структурных подразделений (отделы, команды, филиалы и т.д.).

Сотрудники, непосредственно выполняющие работу внутри предприятия, работают внутри подразделений, взаимодействуют как внутри своей команды, так и обращаются в другие подразделения организации.

Руководители – лица, ответственные за выполнение работы их подразделения. Ведут коммуникацию внутри команды для координации работы подчиненных и участвуют в совещаниях с вышестоящим руководством.

Сотрудники представляют собой базовую категорию пользователей, чья деятельность сосредоточена на исполнении конкретных поручений. В рамках информационной системы они обмениваются текстовой информацией и документами, участвуют в групповых обсуждениях и отслеживают статус выполнения своих задач. Их взаимодействие не ограничивается своим отделом, так как производственные процессы часто требуют привлечения специалистов из смежных структурных подразделений для решения кросс-функциональных вопросов.

Руководители структурных подразделений выполняют функции модераторов и координаторов рабочих процессов. Они формируют информационные рабочие пространства для своих команд, определяют состав участников и контролируют ход обсуждений. В их обязанности входит трансляция распоряжений высшего руководства подчиненным и предоставление обратной связи наверх.

Для данной категории пользователей критически важны инструменты, позволяющие четко разграничивать потоки информации на оперативные

вопросы команды и стратегические совещания с директором. Взаимодействие участников строится на базе иерархической структуры организации.

Система должна поддерживать разграничение прав доступа к информации, чтобы обсуждения внутри одного подразделения могли оставаться конфиденциальными, при этом сохраняя возможность общекорпоративных коммуникаций через глобальный поиск сотрудников и открытые каналы связи [4].

Важным элементом предметной области является интеграция коммуникации с управлением задачами, где сообщения в чате могут трансформироваться в поручения с контролем сроков исполнения.

1.4 Анализ требований

1.4.1 Анализ функциональных требований

В процессе анализа предметной области были выявлены основные требования к разрабатываемой системе. На основе этих требований была составлена диаграмма вариантов использования [4, 5].

Диаграмма вариантов использования (Use Case Diagram) [5] является одним из ключевых инструментов языка моделирования UML и предназначена для наглядного представления функциональных возможностей системы с точки зрения конечных пользователей. Диаграмма вариантов использования фиксирует, что именно система должна делать, не затрагивая вопросы её внутренней реализации. Совокупность таких сценариев формирует полное представление о функциональных границах системы и позволяет на раннем этапе согласовать ожидания всех заинтересованных сторон: заказчиков, аналитиков и разработчиков. Помимо этого, диаграмма служит отправной точкой для разработки тестовых сценариев и детализации требований в рамках последующих этапов проектирования.

Диаграмма вариантов использования для приложения корпоративного мессенджера представлена на рисунке 4

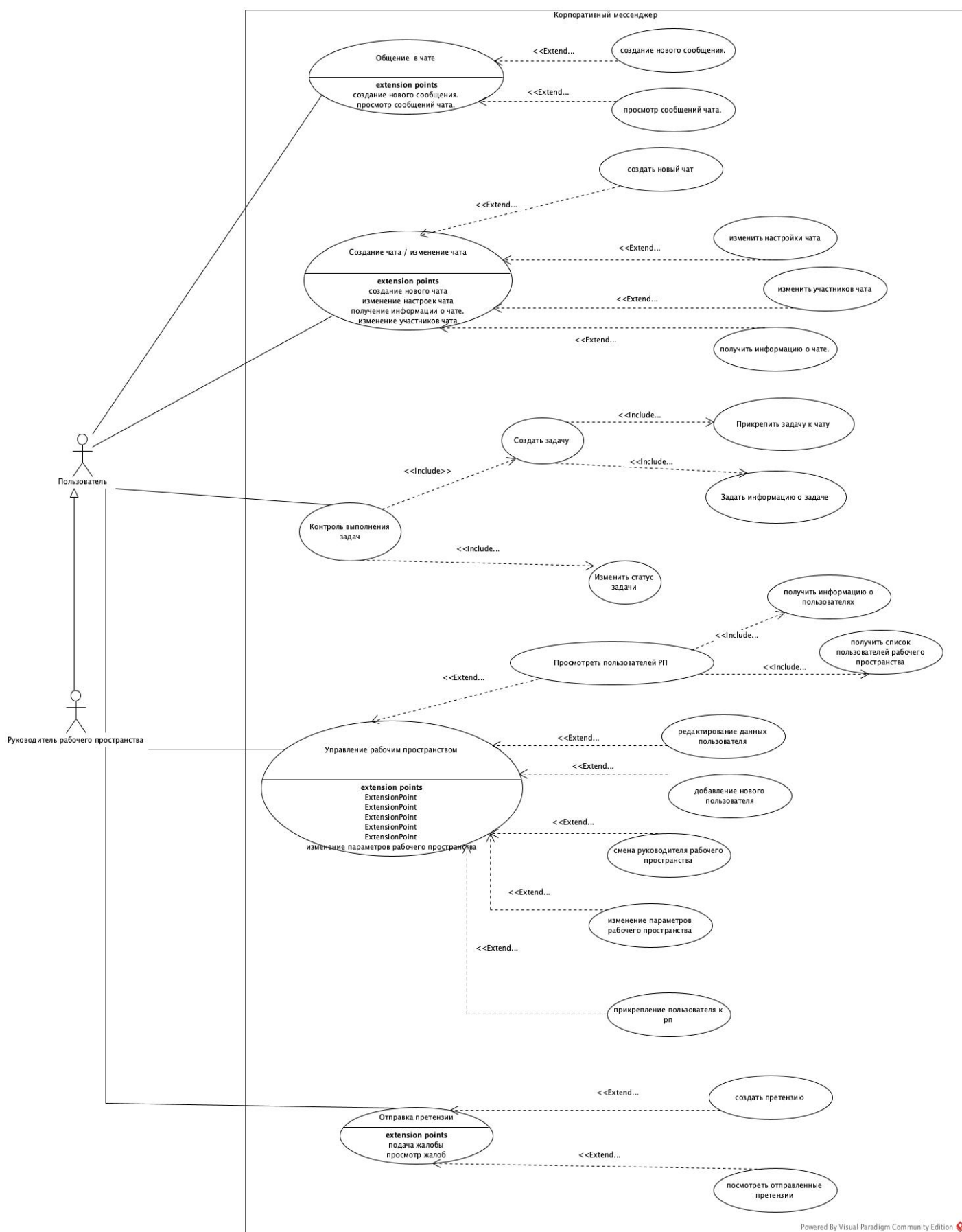


Рисунок 4 – Диаграмма вариантов использования приложения

В таблицах 2 – 3 описаны спецификации некоторых вариантов использования:

Таблица 2 – Спецификация варианта использования «Общение в чате»

ID: 1
Название: Общение в чате
Цель: Обеспечение обмена сообщениями и совместного взаимодействия сотрудников внутри рабочей среды.
<p>Основной поток:</p> <ul style="list-style-type: none"> – Пользователь открывает приложение и авторизуется в системе. – Пользователь выбирает доступное рабочее пространство (РП). – Система отображает список доступных чатов. – Пользователь выбирает существующий чат или создаёт новый. – Пользователь вводит текст сообщения. – Система проверяет корректность данных (непустое сообщение, права доступа). – Система сохраняет сообщение. – Руководитель РП или другой пользователь получает сообщение. – Пользователь может просмотреть историю сообщений. – Система обновляет отображаемую информацию. – Пользователь закрывает чат или выходит из приложения

Представленная спецификация отражает полный цикл взаимодействия пользователя с чатом – от авторизации до завершения сеанса. Ключевым аспектом данного сценария является проверка системой корректности входных данных и прав доступа перед сохранением сообщения, что обеспечивает целостность передаваемой информации и разграничение доступа между участниками рабочего пространства. Поддержка истории сообщений, в свою очередь, предоставляет пользователям возможность обращаться к ранее созданному контексту общения, что особенно актуально в условиях командной работы над совместными задачами.

Таблица 3 – Спецификация варианта использования «Изменение параметров рабочего пространства»

ID: 2
Название: Изменение конфигурации рабочего пространства
Цель: изменить параметры существующего рабочего пространства.
<p>Основной поток:</p> <ul style="list-style-type: none"> – руководитель пространства выбирает рабочее пространство из списка – система отображает текущие параметры – руководитель пространства вносит изменения – система сохраняет изменения через Сервис рабочих пространств – система уведомляет об успешном обновлении.

Таким образом, разработанная диаграмма вариантов использования и представленные спецификации позволяют определить основные сценарии взаимодействия пользователей с корпоративным мессенджером. Описанные варианты использования отражают ключевые функции системы, связанные с обменом сообщениями и управлением рабочими пространствами. Полученные результаты могут быть использованы на следующих этапах проектирования для уточнения структуры приложения, распределения прав пользователей и дальнейшей реализации функциональных возможностей системы.

1.4.2 Анализ нефункциональных требований

При проведении анализа нефункциональных требований к разработке веб-приложения корпоративного мессенджера для конечного пользователя одной из ключевых задач было выбрать правильный архитектурный подход, который обеспечит выполнение всех необходимых требований. Был выделен следующий список требований, которому должна следовать разрабатываемая система:

- гибкая;
- масштабируемая;

- безопасная;
- производительная;
- кроссплатформенная.

Перечисленный набор характеристик обусловлен предметной областью разрабатываемого решения: корпоративная коммуникационная платформа обязана поддерживать мгновенный обмен данными, организацию совместной работы посредством чатов, постановку и контроль задач, управление рабочими пространствами, а также обеспечивать интуитивно понятный доступ ко всем ключевым возможностям системы.

Совокупность сформулированных требований определила выбор микросервисной архитектуры как архитектурного решения, наиболее полно удовлетворяющего поставленным условиям – прежде всего за счёт декомпозиции системы на независимые, слабосвязанные модули. Взаимодействие между клиентом и сервером реализовано через REST API; передача данных в режиме реального времени осуществляется по протоколу WebSocket [6].

Далее представлено обоснование соответствия выбранной архитектуры каждому из нефункциональных требований.

Гибкость – Слабая связанность сервисов в микросервисной архитектуре создаёт условия для независимого развития отдельных модулей: изменения в логике работы с чатами, задачами, рабочими пространствами или системой жалоб не затрагивают остальные компоненты. Это существенно снижает сложность сопровождения и ускоряет внедрение новой функциональности.

Масштабируемость – По мере роста пользовательской базы и увеличения объёма данных архитектура позволяет наращивать вычислительные ресурсы точно – исключительно для тех сервисов, на которые приходится наибольшая нагрузка, например сервиса чатов и сообщений, без необходимости масштабировать всё приложение целиком.

Безопасность – корпоративный мессенджер предназначен для внутренней коммуникации сотрудников организации, поэтому система должна обеспечивать

защиту пользовательских данных и ограничение доступа к функционалу. Для этого предусматривается авторизация пользователей, использование JWT-токенов и разграничение прав доступа в зависимости от роли пользователя в рабочем пространстве.

Производительность – приложение должно обеспечивать быструю загрузку интерфейса, оперативную отправку и получение сообщений, а также своевременное обновление данных о задачах и рабочих пространствах. Использование WebSocket позволяет передавать сообщения в режиме реального времени без постоянной перезагрузки страницы, что повышает скорость взаимодействия пользователя с системой.

Кроссплатформенность – разрабатываемое приложение должно быть доступно пользователю вне зависимости от операционной системы и устройства. Данное требование обеспечивается за счет реализации приложения в формате web-приложения, которое работает через браузер и не требует установки дополнительного программного обеспечения.

Таким образом, выбранная архитектура соответствует основным нефункциональным требованиям разрабатываемого веб-приложения корпоративного мессенджера. Использование микросервисной архитектуры, REST API и WebSocket позволяет обеспечить гибкость, масштабируемость, безопасность и производительность системы [7].

1.5 Планирование разработки и оценка бюджета

Планирование сроков и формирование бюджетной оценки представляют собой фундаментальные составляющие процесса создания программного обеспечения, во многом определяющие итоговый результат. В качестве инструмента управления проектом была выбрана среда ProjectLibre – система, предназначенная для составления календарных планов, мониторинга хода выполнения работ и учёта ресурсов, задействованных в процессе разработки [8]. После того как в систему были введены сведения обо всех стадиях создания веб-

приложения корпоративного мессенджера, была построена модель плана разработки и выполнен расчёт финансовых затрат.

Для визуализации временной структуры проекта применяется диаграмма Ганта. Она представляет собой табличный план, в котором перечислены этапы работы, указаны ответственные исполнители, а длительность каждого этапа отображается в виде горизонтальных отрезков на временной оси [8]. Полученная диаграмма приведена на рисунке В.1 в приложении В.

В рамках планирования проект был декомпозирован на 27 задач. Дата начала работ – 18 марта 2026 года; плановая дата завершения, согласно построенной диаграмме, – 12 мая 2026 года. Суммарная продолжительность разработки составила 56 дней. Перечень задач отражён на рисунке В.2, а сведения о расчете бюджета – на рисунке В.3 в приложении В.

На основании назначенных ресурсов была рассчитана итоговая стоимость реализации проекта. Общие затраты на разработку веб-приложения корпоративного мессенджера составили 228 800 рублей.

2 Проектирование корпоративного мессенджера

2.1 Архитектура приложения

После проведенного анализа функциональных и нефункциональных требований, поставленных для разрабатываемого программного средства, была выбрана архитектура на основе микросервисов.

Такой архитектурный подход позволяет разделить приложение корпоративного мессенджера на независимые компоненты, каждый из которых отвечает за отдельную предметную область. Это упрощает сопровождение системы, повышает гибкость разработки и позволяет масштабировать только те части приложения, на которые приходится наибольшая нагрузка [7].

Разрабатываемое приложение представляет собой web-приложение корпоративного мессенджера для конечного пользователя. Основной задачей системы является обеспечение удобного и безопасного взаимодействия сотрудников организации в рамках рабочих пространств, чатов и задач. Пользователь получает возможность авторизоваться в системе, выбрать доступное рабочее пространство, обмениваться сообщениями в режиме реального времени, создавать и просматривать задачи, а также взаимодействовать с участниками рабочих чатов. Также предусмотрена роль руководителя рабочего пространства, пользователь с этой ролью может просматривать все задачи, управлять настройками рабочего пространства и регистрировать новых пользователей.

Для лучшего понимания структуры системы и взаимосвязей между ее компонентами была разработана диаграмма компонентов приложения. Диаграмма компонентов позволяет представить архитектуру программного средства на верхнем уровне, определить основные сервисы системы и показать зависимости между клиентской частью, серверными микросервисами, базой данных и средствами асинхронного обмена сообщениями [9].

Диаграмма компонентов web-приложения корпоративного мессенджера представлена на рисунке 5.

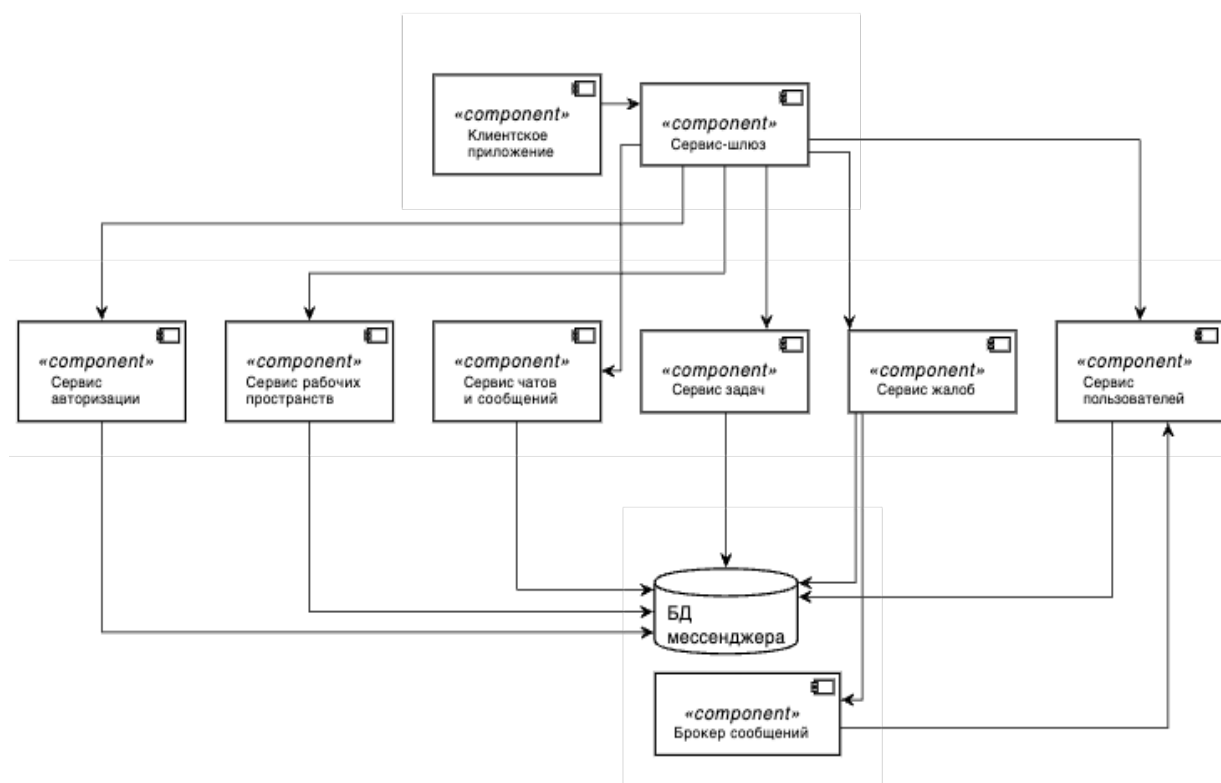


Рисунок 5 – Диаграмма компонентов приложения

На диаграмме выделены следующие основные компоненты системы:

- клиентское приложение – пользовательский интерфейс приложения, через который конечный пользователь взаимодействует с системой: проходит авторизацию, выбирает рабочее пространство, открывает чаты, отправляет сообщения, просматривает задачи и получает уведомления;
- сервис шлюз – единая точка входа в серверную часть приложения;
- сервис аутентификации – отвечает за регистрацию, авторизацию пользователей, обновление и проверку JWT-токенов, а также за ограничение доступа к защищенным функциям приложения;
- сервис пользователей – обеспечивает хранение и обработку информации о пользователях, их профилях, статусах и ролях в системе;
- сервис рабочих пространств – управляет рабочими пространствами организации, составом участников, ролями пользователей и параметрами пространства;

- сервис чатов и сообщений – является одним из ключевых компонентов мессенджера;
- сервис задач – обеспечивает создание задач, назначение исполнителей, изменение статусов, хранение истории изменений и связь задач с чатами рабочего пространства;
- сервис жалоб – предназначен для обработки обращений пользователей о проблемах в работе приложения или некорректных действиях других участников.

Такое разделение системы на отдельные компоненты позволяет изолировать бизнес-логику разных частей приложения. Например, изменение логики работы задач не требует изменения сервиса чатов, а доработка механизма авторизации не затрагивает работу сервиса сообщений. Это особенно важно для корпоративного мессенджера, поскольку приложение должно оставаться расширяемым и удобным для сопровождения.

Для организации взаимодействия между клиентской и серверной частями используются два основных подхода: REST API и WebSocket.

REST API применяется для выполнения синхронных операций, при которых клиент отправляет запрос и получает ответ от сервера. Через REST API выполняются такие действия, как авторизация пользователя, получение списка рабочих пространств, создание чатов, управление задачами, просмотр участников рабочего пространства и изменение пользовательских данных. Для разграничения выполняемых операций используются стандартные HTTP-методы:

- GET – получение данных;
- POST – создание новых объектов;
- PUT – обновление существующих данных;
- DELETE – удаление объектов.

WebSocket используется для функций, требующих обмена данными в реальном времени. В рамках корпоративного мессенджера данный механизм необходим прежде всего для отправки и получения сообщений без перезагрузки

страницы. После установки соединения между web-клиентом и сервером пользователь может получать новые сообщения, уведомления и изменения состояния чата практически мгновенно.

Для асинхронной обработки отдельных событий в системе используется Apache Kafka. Данный брокер сообщений позволяет передавать события между сервисами без жесткой зависимости между ними. Например, сервис жалоб может отправить событие об изменении статуса обращения, а другой сервис обработает это событие и сформирует уведомление для пользователя. Такой подход снижает нагрузку на основные пользовательские сценарии и повышает устойчивость приложения.

Хранение данных реализуется с использованием PostgreSQL. Данная СУБД применяется для надежного хранения структурированной информации корпоративного мессенджера: учетных записей пользователей, рабочих пространств, чатов, сообщений, задач, жалоб и связей между ними. Использование реляционной модели данных позволяет обеспечить целостность информации и корректное отображение связей между сущностями системы [10].

Таким образом, выбранная архитектура web-приложения корпоративного мессенджера обеспечивает модульность, масштабируемость, безопасность и возможность дальнейшего развития системы. Разделение приложения на микросервисы, использование REST API для стандартных операций и WebSocket для обмена сообщениями в реальном времени позволяют реализовать удобный и современный инструмент корпоративной коммуникации для конечного пользователя.

2.2 Разработка концептуальной схемы базы данных

На основе полученных бизнес-процессов, была составлена диаграмма концептуальной схемы базы данных (рисунок 3)

Целью разработки концептуальной схемы являлось выявление ключевых сущностей (объектов), значимых для бизнеса, их атрибутов (характеристик) и смысловых связей между ними. Данная модель служит важнейшим

инструментом коммуникации между заказчиками, экспертами предметной области и разработчиками, обеспечивая единое и непротиворечивое понимание структуры хранимой информации [11].

При построении концептуальной схемы особое внимание уделялось корректному отражению бизнес-правил и ограничений предметной области, включая кардинальности связей, обязательность атрибутов и условия их использования. Это позволило обеспечить целостность данных уже на раннем этапе проектирования и минимизировать риски логических ошибок при дальнейшей реализации базы данных.

Кроме того, разработанная концептуальная модель обладает достаточной степенью абстракции, что делает её независимой от конкретных технологий хранения и управления данными. Такой подход обеспечивает гибкость проектного решения и возможность последующей адаптации схемы при изменении требований бизнеса или выборе иной системы управления базами данных.

Для обеспечения детальной проработки каждой компоненты модели были составлены спецификации первоначального набора сущностей. Составление спецификаций позволило формализовать и уточнить требования к данным, исключить неоднозначности и заложить основу для последующей стадии логического проектирования, на которой концептуальная модель будет преобразована в схему, ориентированную на реляционную или иную модель данных [12].

Итоговая концептуальная схема охватывает восемь сущностей: пользователя, администратора, рабочее пространство, чат, сообщение, задачу, жалобу и тариф. Такая структура обеспечивает логическую полноту модели и создаёт предпосылки для её однозначного преобразования в реляционную схему на следующем этапе проектирования.

Модель представлена на рисунке 6.

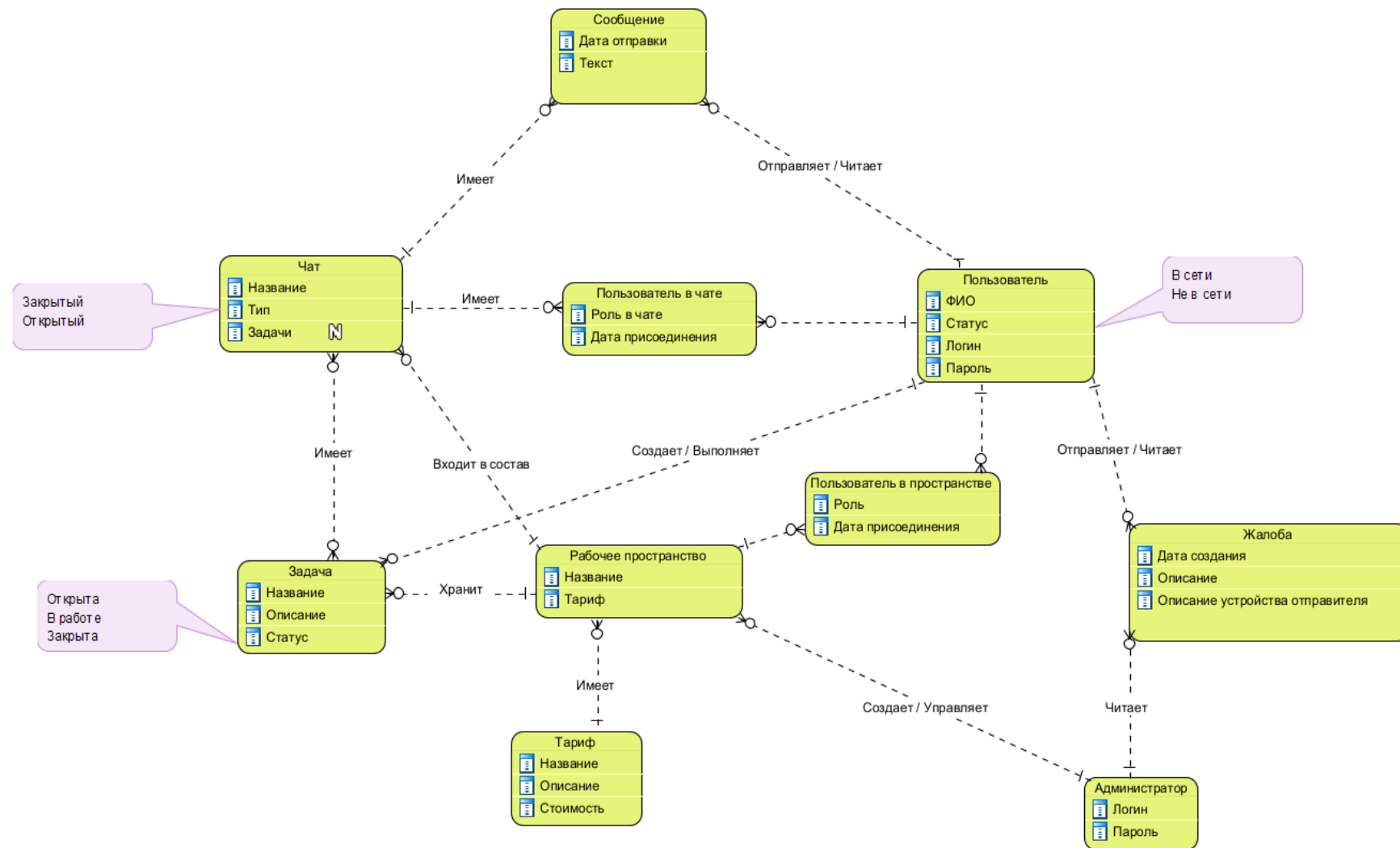


Рисунок 6 – Концептуальная модель БД

Разработанная концептуальная схема и ее детализация в таблице 4 формируют согласованную и полную документальную базу для перехода к следующему этапу создания информационной системы.

Таблица 4 – Спецификации первоначального набора сущностей

№ п/п	Наименование	Описание
1	Пользователь	Пользователь мессенджера
2	Администратор	Человек, контролирующий работу приложения
3	Рабочее пространство	Область коммуникации внутри одной компании
4	Чат	Отдельная комната для общения в рабочем пространстве
5	Задача	Задача, выделенная в рамках работы одной команды (т.е. в одном чате)
6	Сообщение	Текстовое сообщение внутри чата
7	Жалоба	Описание проблемы в работе приложения, отправляемое администратору
8	Тариф	Условия предоставления организации доступа к мессенджеру

В приложении Б приведены спецификации сущностей концептуальной схемы данных и связей между сущностями концептуальной схемы данных.

Таким образом, в результате анализа предметной области и концептуального проектирования были предварительно выделены основные сущности и связи между ними.

2.3 Логическое моделирование базы данных

Логическое проектирование является следующим ключевым этапом после создания концептуальной модели. Его основной задачей выступает преобразование абстрактной, предметно-ориентированной схемы в формализованную структуру, которая уже напрямую соответствует принципам конкретной модели данных в данном случае, реляционной. Исходя из полученных данных и на основе разработанной ранее концептуальной схемы, была составлена детализированная логическая модель базы данных [13].

Данная диаграмма отображает не только состав таблиц и их атрибутов, но и все ключевые связи между ними, что позволяет наглядно оценить структуру будущей базы данных и является основной технической документацией для этапа физического проектирования и непосредственного написания SQL-скриптов создания базы.

Полученная модель представлена на рисунке 7.

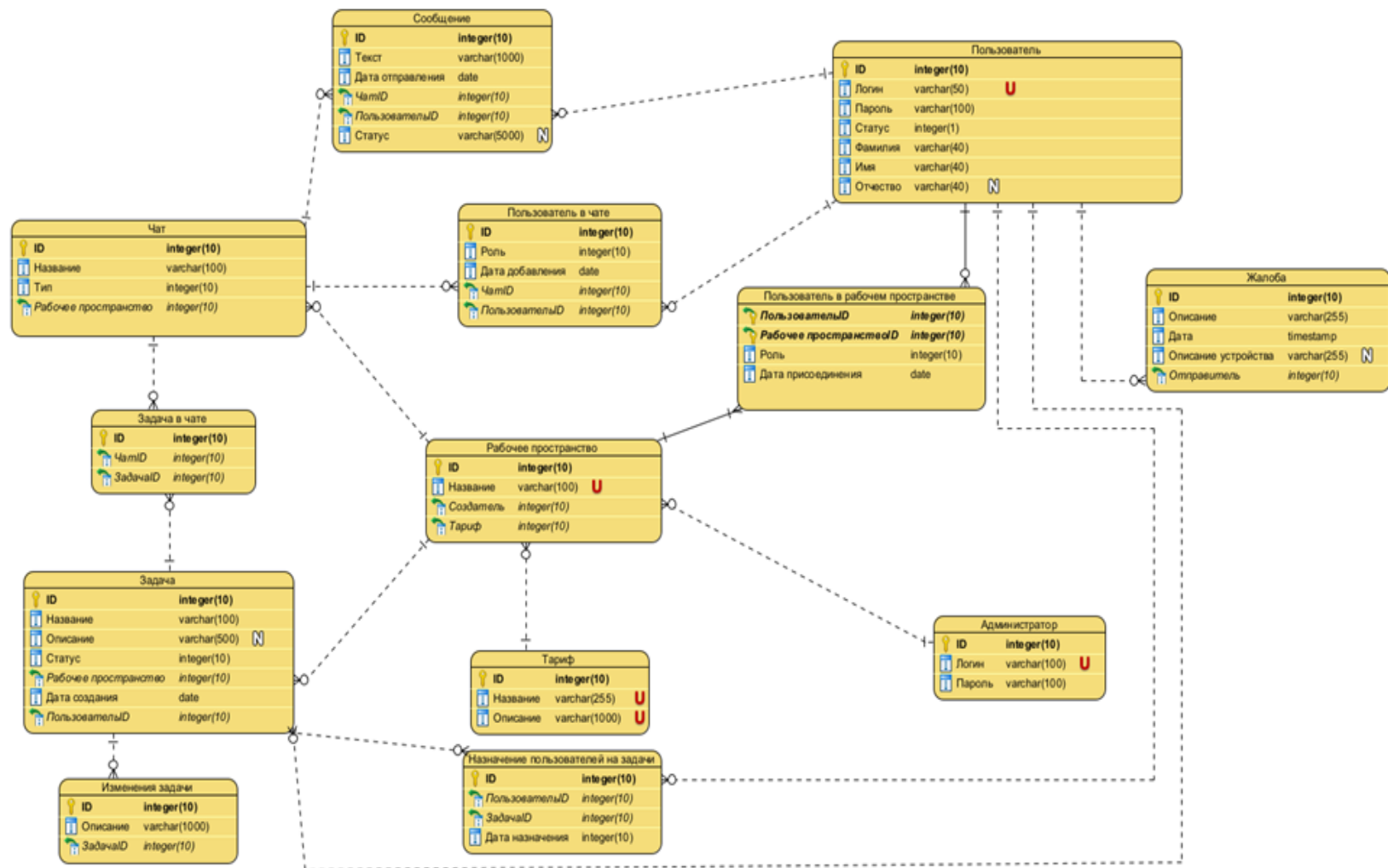


Рисунок 7 – Логическая схема базы данных

Таким образом, в ходе логического проектирования была получена структурированная модель базы данных корпоративного мессенджера, отражающая основные информационные объекты системы: пользователей, рабочие пространства, чаты, сообщения, задачи, жалобы и связанные с ними вспомогательные таблицы. Разработанная схема обеспечивает целостность данных, исключает избыточное дублирование информации и задает основу для дальнейшего физического проектирования базы данных. Наличие первичных и внешних ключей, ограничений и связей между таблицами позволяет корректно хранить данные приложения и поддерживать основные бизнес-процессы корпоративного мессенджера, включая общение пользователей, управление рабочими пространствами и контроль выполнения задач.

2.4 Обобщенная структура разрабатываемого сервиса

В своей основе разрабатываемое решение представляет собой клиент-серверное веб-приложение, ориентированное на обеспечение внутренней коммуникации между сотрудниками организации. Система охватывает управление рабочими пространствами, чатами, задачами и обращениями пользователей. Клиентская часть служит точкой входа для конечного пользователя и предоставляет доступ ко всему функциональному набору системы. Роль клиента выполняет браузерное веб-приложение, запускаемое на персональном компьютере или любом другом устройстве пользователя.

Серверная часть приложения представляет собой набор микросервисов, размещенных на удаленном сервере. Каждый микросервис отвечает за отдельную область бизнес-логики корпоративного мессенджера: авторизацию пользователей, управление профилями, рабочими пространствами, чатами, сообщениями, задачами и жалобами. Для централизованной маршрутизации запросов используется API Gateway, который выступает единой точкой входа в систему и перенаправляет входящие HTTP- и WebSocket-запросы к соответствующим сервисам.

Для хранения данных используется сервер базы данных PostgreSQL. В базе данных хранятся сведения о пользователях, рабочих пространствах, чатах, сообщениях, задачах, ролях участников, жалобах и тарифах. Такое разделение позволяет отделить логику обработки запросов от уровня хранения данных и обеспечить централизованное управление информацией корпоративного мессенджера.

Основные преимущества клиент-серверной структуры для разрабатываемого приложения:

- централизованное хранение данных: вся информация о пользователях, сообщениях, задачах и рабочих пространствах хранится в единой базе данных;
- минимальные требования к пользователю: для работы с приложением требуется только web-браузер и доступ к сети;
- удобство сопровождения: обновление серверной части и клиентского web-приложения может выполняться централизованно без установки отдельного программного обеспечения на устройства пользователей;
- безопасность: доступ к функциям системы контролируется механизмами аутентификации и авторизации, а данные пользователей не хранятся локально на клиентском устройстве;
- масштабируемость: отдельные микросервисы могут разворачиваться и масштабироваться независимо в зависимости от нагрузки.

Основные недостатки подобной структуры:

- зависимость от доступности сервера: при отказе серверной части пользователи не смогут работать с приложением;
- зависимость от базы данных: сбой сервера базы данных может привести к невозможности получения и сохранения сообщений, задач и другой информации;
- риск перегрузки серверных компонентов: при большом количестве одновременных пользователей может потребоваться дополнительное масштабирование сервисов;

– необходимость обеспечения сетевой безопасности, так как взаимодействие клиента и сервера выполняется по сети.

Архитектура приложения в общем виде представлена на диаграмме развертывания на рисунке 8.

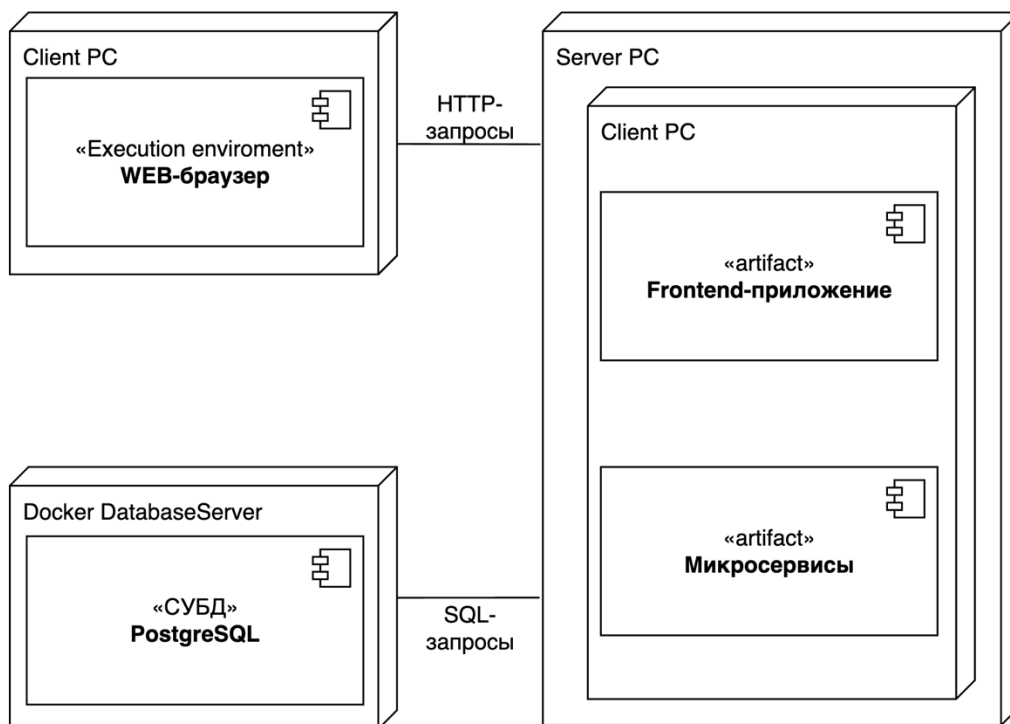


Рисунок 8 – Диаграмма развертывания

Диаграмма иллюстрирует три основных инфраструктурных узла, обеспечивающих работоспособность корпоративного мессенджера: пользовательское устройство, сервер приложения и сервер базы данных. На стороне пользователя функционирует веб-браузер, через который осуществляется взаимодействие с интерфейсом системы. Сервер приложения включает клиентское веб-приложение, шлюз API Gateway и совокупность микросервисов, несущих ответственность за реализацию бизнес-логики. Сервер базы данных предназначен для долговременного хранения всех данных, необходимых для корректного функционирования системы.

Пользователь через веб-браузер направляет HTTP-запросы к серверу приложения в процессе выполнения следующих операций, как авторизация, просмотр рабочих пространств, управление задачами, создание чатов и отправка

жалоб. Для обмена сообщениями в режиме реального времени используется WebSocket-соединение, позволяющее передавать новые сообщения без постоянного обновления страницы.

API Gateway принимает входящие запросы от клиентской части и перенаправляет их к соответствующим микросервисам. Например, запросы на вход в систему направляются в сервис авторизации, операции с чатами и сообщениями – в сервис чатов, операции с задачами – в сервис задач, а обращения пользователей – в сервис жалоб. При необходимости микросервисы обращаются к серверу базы данных PostgreSQL с SQL-запросами для получения, добавления, изменения или удаления данных.

Также для обеспечения развертывания приложения на сервере используется Docker. Контейнеризация позволяет запускать каждый микросервис, API Gateway, базу данных и вспомогательные компоненты в изолированной среде. Для описания порядка запуска и взаимодействия контейнеров применяется Docker Compose. Такой подход упрощает развертывание корпоративного мессенджера, обеспечивает воспроизводимость среды выполнения и позволяет быстрее переносить приложение между различными серверами [14].

2.5 Проектирование клиентской части

В качестве клиентской части приложения было решено разрабатывать web-приложение на JavaScript. Для удобства разработки были использованы следующие инструменты:

- библиотека React – для создания продвинутых пользовательских интерфейсов [15];
- библиотека Tailwind – для удобной стилизации компонентов [16].

Взаимодействие с серверной частью приложения происходит с помощью REST API через API-Gateway. Для получения сообщений используется веб-сокеты. Среди основных страниц приложения можно выделить следующие:

Главная страница приложения – страница со списком рабочих пространств пользователя. Интерфейс главной страницы приведен на рисунке 9.

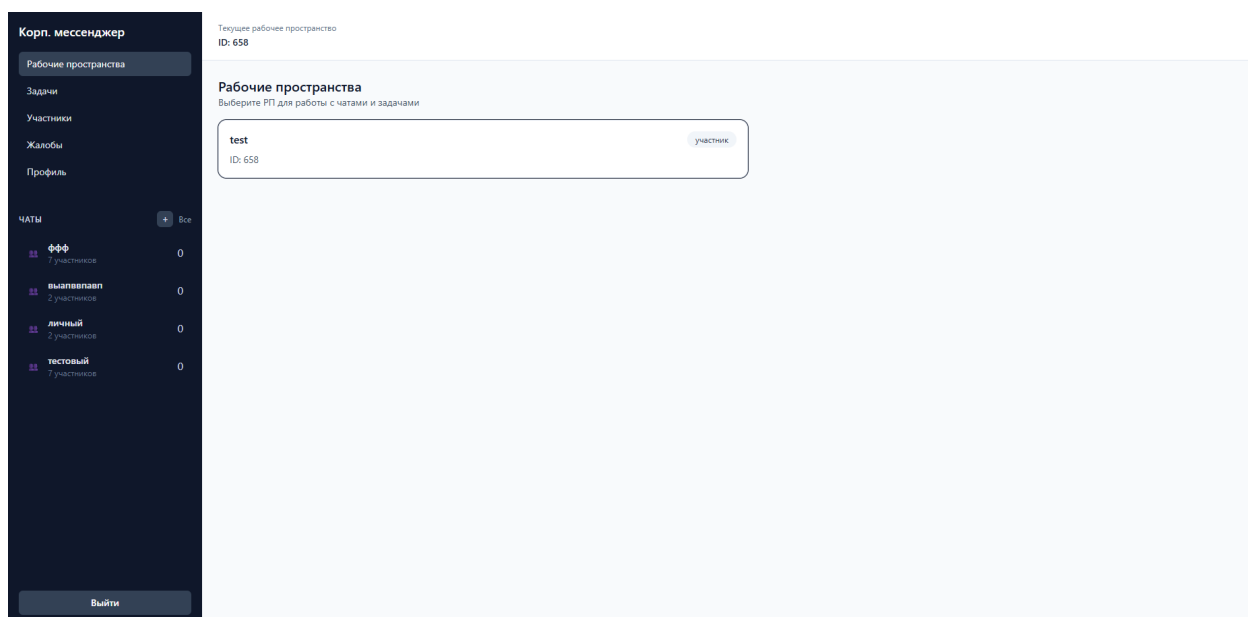


Рисунок 9 – Интерфейс главной страницы

Страница задач – страница, на которой перечислены задачи к которым пользователь имеет доступ. Интерфейс страницы приведен на рисунке 10.

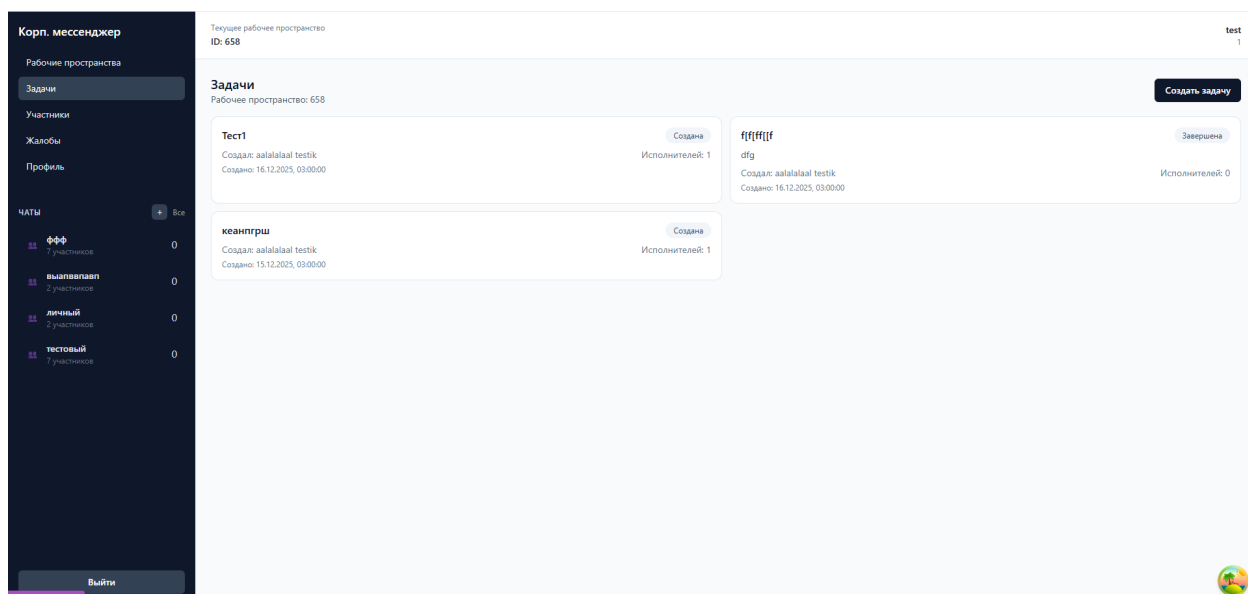


Рисунок 10 – Интерфейс страницы рабочих пространств

Страница чата – на данной странице пользователь может читать и писать сообщения другим пользователям в пространстве. Интерфейс страница представлен на рисунке 11.

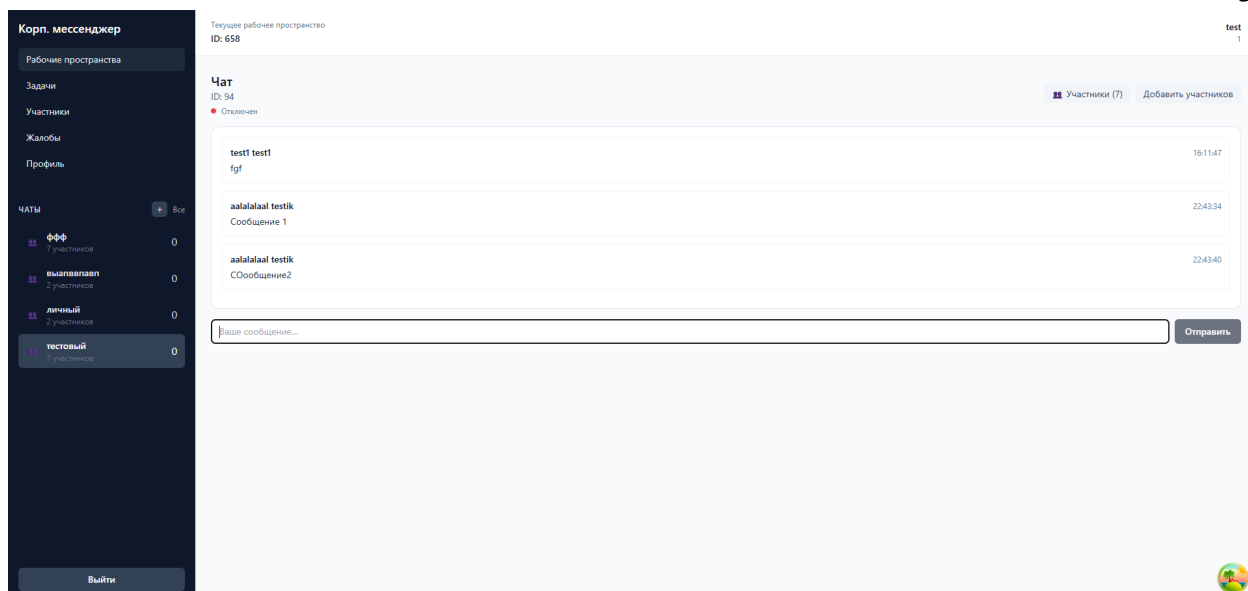


Рисунок 11 – Интерфейс страницы жалоб

Страница авторизации – точка входа в приложение. Интерфейс страницы представлен на рисунке 12.

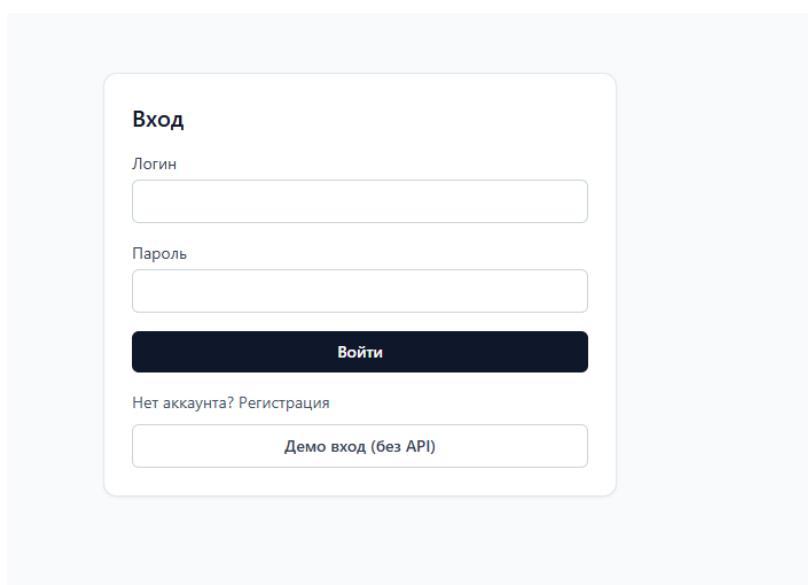


Рисунок 12 – Интерфейс страницы авторизации

Интерфейс отвечает всем поставленным требованиям.

2.6 Требования к аппаратному обеспечению серверной части

В целях обеспечения стабильного и бесперебойного функционирования серверной части веб-приложения были определены пороговые и оптимальные характеристики аппаратной среды развёртывания. Соответствие указанным

параметрам является необходимым условием для корректной обработки пользовательских запросов, поддержания обмена сообщениями в режиме реального времени и надёжного хранения данных. Сформированные требования приведены в таблице 5.

Таблица 5 – Требования к программному обеспечению

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

По результатам проектирования инфраструктуры серверной части были установлены аппаратные конфигурации двух уровней. Минимальная конфигурация рассчитана на выполнение базового набора функций системы: аутентификации пользователей, работы с чатами, задачами, рабочими пространствами и обращениями. Рекомендуемая конфигурация ориентирована на эксплуатацию в условиях повышенной нагрузки – при росте числа активных пользователей и увеличении объёма хранимых данных. Соблюдение приведённых требований при развёртывании гарантирует устойчивую работу всех компонентов системы, своевременную обработку входящих запросов и надёжное взаимодействие сервисов между собой.

3 Разработка корпоративного мессенджера

3.1 Реализация базы данных

Физическая модель базы данных является наиболее детальным уровнем её описания и определяет конкретную реализацию хранилища данных: типы данных каждого атрибута, ограничения целостности, индексы и физическое размещение файлов на носителе. В отличие от концептуальной и логической моделей, физическая модель строится с учётом особенностей конкретной СУБД.

На основе ранее разработанных концептуальной и логической моделей была построена физическая модель базы данных в среде Visual Paradigm 17.0 в нотации IDEF1X. Данная нотация позволяет явно отображать первичные и внешние ключи, типы данных атрибутов и характер связей между сущностями. Физическая модель представлена на рисунке В.4 в приложении В.

База данных содержит 13 таблиц. Основные сущности – users, administrators, tariffs, workspaces, chats, tasks, messages и complaints – хранят данные предметной области. Связующие таблицы userinworkspace, userinchat, userintask, taskinchat и taskchanges реализуют отношения многие-ко-многим между основными сущностями. Для обеспечения аутентификации предусмотрена таблица refresh_tokens, а история изменения статусов обращений фиксируется в таблице complaint_status_history.

С целью повышения производительности на всех внешних ключах и часто используемых в условиях выборки полях созданы индексы. Всего определено 23 индекса.

База данных, соответствующая разработанной физической модели, была развёрнута с помощью механизма миграций на основе инструмента golang-migrate. Механизм миграций обеспечивает версионирование и синхронизацию структуры базы данных: каждое изменение схемы оформляется в виде пронумерованного SQL-файла и применяется последовательно, что гарантирует воспроизводимость развёртывания во всех средах – разработки, тестирования и эксплуатации – а также возможность отката к предыдущей версии схемы при

необходимости. В ходе разработки было применено три миграции: первая формирует исходную схему со всеми таблицами, ограничениями и индексами; вторая добавляет в таблицу `tasks` поля `created_at` и `completed_at` для разграничения момента постановки задачи и её фактического завершения; третья уточняет семантику поля `date`, ограничивая его использование только для завершённых задач.

3.2 Реализация клиентской части приложения

Клиентская часть реализована как SPA веб-приложение, написанное на TypeScript с использованием библиотек React, Tailwind и react-router-dom. В архитектуре приложения были задействованы некоторые элементы из Feature-Sliced Design, что позволило эффективно разделить функционал приложения в соответствующих компонентах [16, 18].

Согласно принципам Feature-Sliced Design, исходный код разделён на слои: `app` – глобальная конфигурация и провайдеры, `pages` – страницы приложения, `features` – составные и атомарные блоки функциональности. Такое разграничение исключает хаотичные зависимости между модулями: слой более высокого уровня может импортировать из нижележащего, но не наоборот. Взаимодействие с сервером централизовано в `api`.

В общем виде структура приложения выглядит следующим образом (рисунок 13):

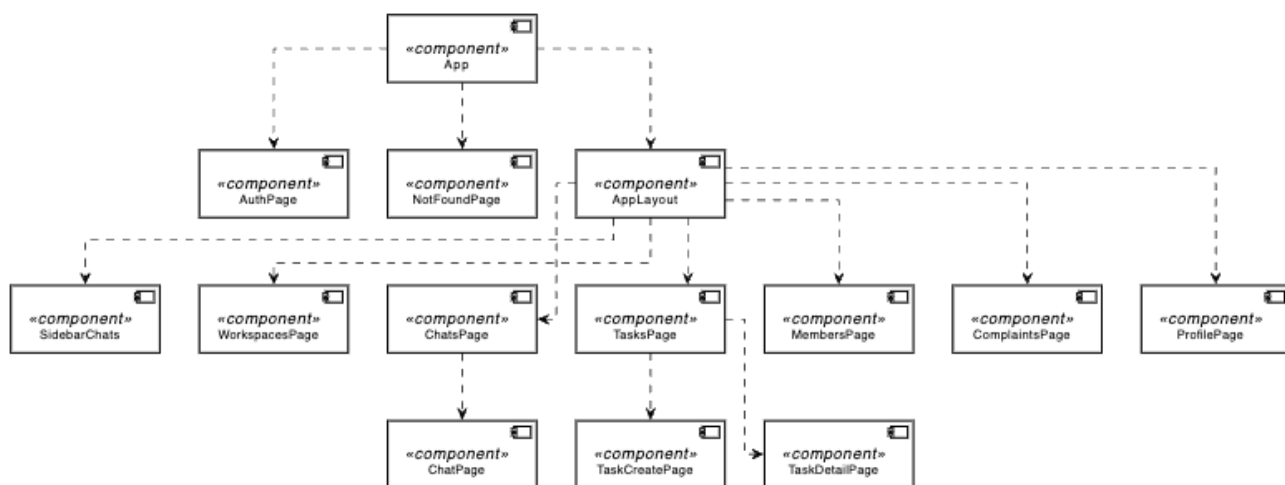


Рисунок 13 – Общая структура приложения

На представленном рисунке можно выделить следующие компоненты:

- App – центральный компонент приложения, отвечающий за маршрутизацию и навигацию между страницами;
- main – стартовая точка приложения, здесь инициализируется корневой контекст выполнения (маршрутизатор, провайдер кеширования данных и подключение к DOM);
- AuthPage – страница входа и регистрации пользователя;
- WorkspacesPage – страница выбора и управления рабочими пространствами;
- ChatsPage – страница списка чатов в выбранном рабочем пространстве;
- ChatPage – страница просмотра чата и обмена сообщениями в режиме реального времени;
- TasksPage – страница списка задач;
- TaskCreatePage – страница создания новой задачи;
- TaskDetailPage – страница просмотра и редактирования задачи;
- MembersPage – страница управления участниками рабочего пространства;
- ComplaintsPage – страница отправки и просмотра жалоб;
- ProfilePage – страница профиля пользователя;
- NotFoundPage – служебная страница при обращении к несуществующему маршруту;
- AppLayout – общий макет авторизованной части приложения: боковое меню, шапка с данными пользователя и область отображения текущей страницы;
- SidebarChats – компонент боковой панели со списком чатов и быстрым переходом к диалогу.

Листинг клиентской части программного обеспечения представлен в приложении Д.

Корневой компонент App разделяет приложение на публичную зону и защищённую область, отображаемую через AppLayout, который выступает общей оболочкой для авторизованных пользователей: в нём размещены боковая панель SidebarChats и все рабочие страницы.

3.3 Реализация серверной части приложения

Серверная часть системы построена на основе микросервисной архитектуры, что обеспечивает масштабируемость, отказоустойчивость и возможность независимого развертывания компонентов [17].

В качестве основного языка программирования выбран язык Go, который обеспечивает высокую производительность и эффективное управление параллельными процессами, что критически важно для систем реального времени. База данных PostgreSQL используется для хранения структурированных данных благодаря своей надежности и поддержке сложных запросов. Вся инфраструктура контейнеризирована с использованием Docker и оркестрирована через Docker Compose, что упрощает развертывание и управление окружениями [10, 14, 19].

Архитектурно система состоит из нескольких взаимосвязанных микросервисов, каждый из которых отвечает за определенную предметную область. API Gateway служит единой точкой входа для всех клиентов, осуществляя маршрутизацию запросов к соответствующим сервисам и предоставляя функцию health check для мониторинга состояния системы. Сервис аутентификации управляет процессами регистрации и входа пользователей, а также генерацией и валидацией JWT токенов, обеспечивая безопасный доступ к ресурсам.

Сервис пользователей занимается управлением профилями, включая обновление личной информации и отслеживание статусов активности. Сервис рабочих пространств реализует логику создания и управления организационными единицами – рабочими пространствами, включая управление участниками, ролями и тарифными планами. Ключевым компонентом системы

является сервис чатов и сообщений, который обеспечивает реальное время общение через WebSocket протокол, а также хранение истории переписки через REST API интерфейс.

Сервис задач предоставляет функционал для создания, назначения и отслеживания выполнения задач, включая ведение истории изменений и привязку задач к соответствующим чатам. Сервис жалоб обрабатывает обращения пользователей, позволяя создавать, просматривать и администрировать претензии.

База данных построена по реляционной модели с четкой структурой сущностей и связей. Основные таблицы включают пользователей, сообщения, чаты, рабочие пространства, задачи, администраторов, жалобы и тарифы. Связующие таблицы определяют отношения между сущностями: участие пользователей в рабочих пространствах и чатах, назначение задач на исполнителей, привязка задач к чатам. Рабочее пространство выступает центральной организационной единицей, к которой привязаны чаты и задачи, а пользователи могут участвовать в нескольких пространствах с разными ролями.

Система предоставляет два основных интерфейса взаимодействия: REST API для синхронных операций и WebSocket для коммуникации в реальном времени. REST API включает пятьдесят восемь эндпоинтов, распределенных по сервисам, с полной документацией по каждому методу, включая форматы запросов и ответов, коды ошибок и примеры использования. WebSocket обеспечивает двустороннюю связь для мгновенного обмена сообщениями, уведомлений и отслеживания онлайн-статусов.

При реализации каждого из микросервисов в качестве единого структурного шаблона была выбрана многослойная архитектура. Её суть состоит в разделении приложения на несколько горизонтальных уровней, каждый из которых несёт ответственность за строго определённый функциональный аспект системы и не зависит от внутренней реализации соседних слоёв. Подобный подход обеспечивает простоту сопровождения и внесения изменений, повышает степень модульности кода и способствует повторному использованию уже

реализованных компонентов [7]. Типовая структура многослойной архитектуры, применённая в каждом микросервисе разрабатываемой системы, представлена на рисунке 14.

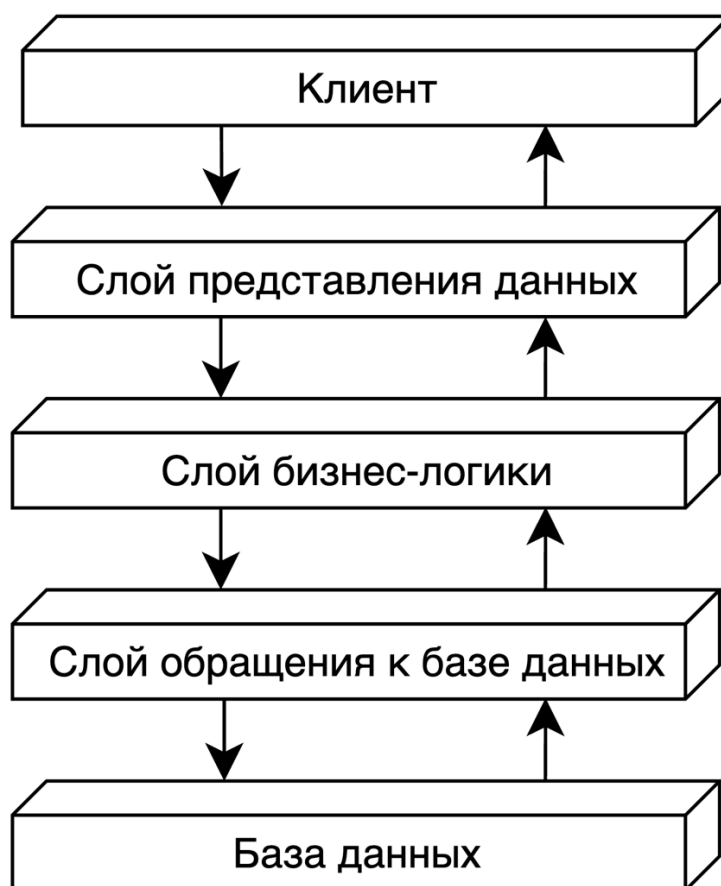


Рисунок 14 – Типовая архитектура микросервисов

Каждый слой архитектуры разрабатываемого приложения выполняет отдельную функцию и обеспечивает последовательную обработку действий пользователя в корпоративном мессенджере.

Пользователь взаимодействует с системой через веб-интерфейс клиентской части. Через него выполняются основные действия: авторизация, выбор рабочего пространства, просмотр чатов, отправка сообщений, создание задач, изменение их статусов и управление участниками. После ввода данных запрос передаётся api gateway а после на соответствующие микросервисы.

Внутри каждого микросервиса слой обработки запросов принимает данные от клиента, проверяет обязательные поля, формирует ответы и передаёт

информацию в слой бизнес-логики. На этом уровне определяется, какой именно обработчик должен выполнить действие пользователя в рамках данного сервиса.

Слой бизнес-логики реализует основные сценарии работы своего функционального модуля. Он отвечает за выполнение операций, связанных с управлением сущностями сервиса: проверяет роли участников, принадлежность пользователя к рабочему пространству, возможность выполнения конкретного действия, а также соблюдение правил предметной области. Например, при создании задачи система проверяет корректность названия, наличие исполнителей и их принадлежность к выбранному рабочему пространству.

Слой доступа к данным обеспечивает взаимодействие микросервиса с базой данных. Он содержит методы для получения, добавления, изменения и удаления информации, относящейся к данному сервису. Этот слой изолирует работу с базой данных от бизнес-логики, что делает структуру каждого микросервиса понятной и удобной для сопровождения.

На уровне базы данных хранится вся основная информация, необходимая для работы корпоративного мессенджера. Такая структура обеспечивает целостность информации и корректную работу всех пользовательских сценариев.

Листинг программы представлен в приложении Г.

Таким образом, сочетание микросервисной архитектуры на уровне системы и многослойной архитектуры внутри каждого сервиса упрощает развитие системы, повышает удобство сопровождения и позволяет независимо дорабатывать отдельные части корпоративного мессенджера без нарушения общей логики его работы.

4 Контроль качества программного обеспечения

4.1 Обоснование выбора модели тестирования и метрик кода

В процессе разработки web-приложения корпоративного мессенджера для конечного пользователя особое внимание уделялось контролю качества программного обеспечения. Для такого приложения важно обеспечить корректную работу основных пользовательских сценариев: авторизацию, работу с рабочими пространствами, обмен сообщениями, создание задач, назначение исполнителей и обработку пользовательских действий в интерфейсе.

Использование многоуровневого подхода к тестированию даёт возможность удостовериться в том, что разработанное приложение соответствует заявленным функциональным требованиям, обнаружить дефекты в логике работы системы и оценить качество программного кода. В рамках настоящей выпускной квалификационной работы применялись следующие виды тестирования [20]:

- функциональное тестирование – верификация реализованного функционала на предмет соответствия установленным требованиям;
- интеграционное тестирование – оценка корректности взаимодействия между различными частями и сервисами приложения.

В качестве основного метода проверки был выбран метод функционального тестирования, так как он позволяет оценить работу приложения с точки зрения пользователя и убедиться, что система корректно реагирует на допустимые и ошибочные входные данные. Для построения набора тестов использовались функциональная диаграмма, таблица решений и тест-кейсы [20].

Для оценки качества исходного кода были выбраны следующие метрики:

- SLOC;
- метрика Холстеда;
- метрика Чепина.

SLOC (Source Lines of Code) – метрика, определяющая количество строк исходного кода приложения. Она позволяет оценить общий объем разработанного программного продукта и приблизительную трудоемкость его сопровождения.

Метрика Холстеда используется для оценки сложности программы на основе количества операторов и операндов. Данная метрика позволяет определить объем программы, ее словарь, длину, сложность и трудоемкость анализа кода.

Метрика Чепина применяется для оценки сложности программы через роли переменных. В отличие от простого подсчета количества переменных, данный подход учитывает, как именно переменные используются в логике приложения: являются ли они входными, изменяемыми, управляющими или вспомогательными.

Выбор данных метрик обусловлен тем, что они позволяют получить количественную оценку объема и сложности программного кода, а также выявить потенциально сложные участки приложения, требующие дополнительного контроля при сопровождении и развитии системы.

4.2 Тестирование корпоративного мессенджера

4.2.1 Функциональное тестирование создания задачи

После завершения разработки основных функций корпоративного мессенджера было проведено функциональное тестирование сценария создания задачи в рабочем пространстве. Данный сценарий является одним из ключевых для приложения, так как система корпоративного мессенджера предназначена не только для обмена сообщениями, но и для организации совместной работы пользователей.

Пользователь с соответствующими правами должен иметь возможность создать задачу внутри рабочего пространства. Для этого необходимо указать название задачи, описание, дату выполнения, рабочее пространство и назначить хотя бы одного исполнителя.

Для функции создания задачи были определены следующие требования:

- название задачи является обязательным и не должно превышать 255 символов;
- описание задачи является необязательным, но при наличии не должно превышать 5000 символов;
- дата выполнения задачи не может быть в прошлом;
- у задачи должен быть назначен хотя бы один исполнитель;
- все исполнители должны являться участниками выбранного рабочего пространства;
- пользователь, создающий задачу, должен состоять в данном рабочем пространстве;
- при корректном вводе задача должна быть успешно создана.

Для проведения тестирования были выделены причины и следствия, представленные в таблице 6.

Таблица 6 – Причины и следствия

Обозначение	Причины
П1	Название задачи корректное: не пустое и не превышает 255 символов
П2	Описание задачи корректное: не указано или не превышает 5000 символов
П3	Дата выполнения корректная и не находится в прошлом
П4	Назначен хотя бы один исполнитель
П5	Все исполнители являются участниками рабочего пространства
П6	Пользователь является участником рабочего пространства
С1	Сообщение: «Название задачи не может быть пустым или превышать 255 символов»
С2	Сообщение: «Описание задачи не может превышать 5000 символов»
С3	Сообщение: «Дата выполнения не может быть в прошлом»
С4	Сообщение: «Необходимо назначить хотя бы одного исполнителя»
С5	Сообщение: «Исполнитель не является участником рабочего пространства»
С6	Сообщение: «Вы не являетесь участником этого рабочего пространства»
С7	Сообщение: «Задача успешно создана»

На основе выделенных причин и следствий была построена функциональная диаграмма, описывающая поведение приложения при создании задачи, изображенная на рисунке 15. Диаграмма показывает, какие ошибки могут возникнуть при нарушении каждого из условий, а также при какой комбинации данных задача создается успешно.

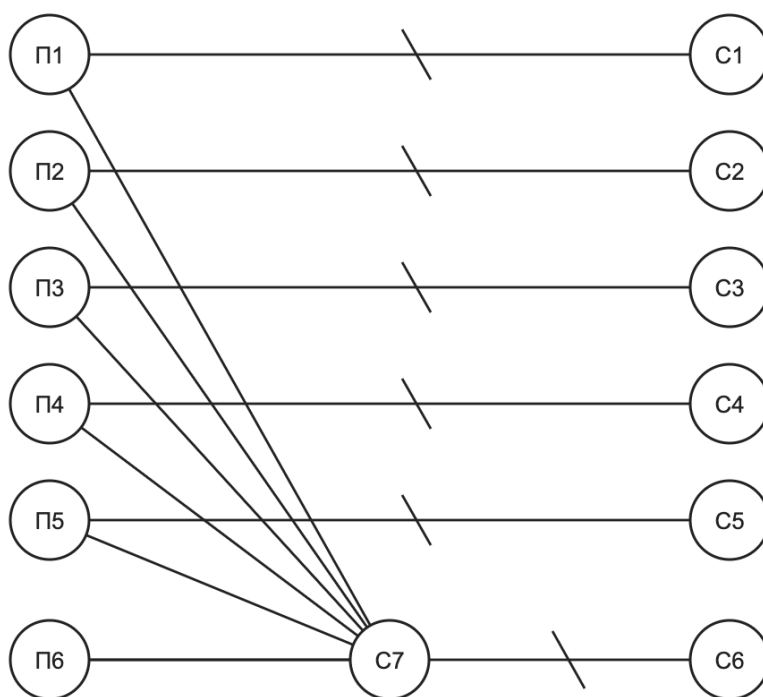


Рисунок 15 – Функциональная диаграмма создания задачи

На основе функциональной диаграммы была составлена таблица 7. В верхней части таблицы представлены условия, влияющие на результат выполнения операции, а в нижней части – соответствующие реакции системы.

Таблица 7 – Таблица решений

Правило	1	2	3	4	5	6	7
П1 – Название корректное	0	1	1	1	1	1	1
П2 – Описание корректное	1	0	1	1	1	1	1
П3 – Дата корректная	1	1	0	1	1	1	1
П4 – Есть исполнители	1	1	1	0	1	1	1
П5 – Исполнители в РП	1	1	1	1	0	1	1
П6 – Пользователь в РП	1	1	1	1		0	1
С1 – Ошибка названия	1	0	0	0	0	0	0
С2 – Ошибка описания	0	1	0	0	0	0	0
С3 – Ошибка даты	0	0	1	0	0	0	0
С4 – Нет исполнителей	0	0	0	1	0	0	0
С5 – Исполнитель не в РП	0	0	0	0	1	0	0
С6 – Пользователь не в РП	0	0	0	0	0	1	0
С7 – Успех	0	0	0	0	0	0	1

На основе таблицы решений были составлены тест-кейсы, позволяющие проверить реакцию системы на основные корректные и некорректные варианты

входных данных. Тест-кейсы представлены в таблице 8. Результаты приведены в приложении Ж.

Таблица 8 – Тест-кейсы создания задачи

№	Входные данные	Ожидаемый результат
1	Название: пустое. Описание: любое. Дата: любая. Исполнители: любые. РП: любое	Ошибка: название задачи не может быть пустым или превышать 255 символов (рисунок Ж.1)
2	Название: «Разработать новый модуль». Описание: более 5000 символов. Дата: будущая дата. Исполнители: user1. РП: Workspace1	Ошибка: описание задачи не может превышать 5000 символов (рисунок Ж.2)
3	Название: «Разработать новый модуль». Описание: «Разработать модуль авторизации». Дата: 2020.01.01. Исполнители: user1, user2. РП: Workspace1	Ошибка: дата выполнения не может быть в прошлом (рисунок Ж.3)
4	Название: «Разработать новый модуль». Описание: «Разработать модуль авторизации». Дата: будущая дата. Исполнители: не назначены. РП: Workspace1	Ошибка: необходимо назначить хотя бы одного исполнителя (рисунок Ж.4)
5	Название: «Разработать новый модуль». Описание: «Разработать модуль авторизации». Дата: будущая дата. Исполнители: outsider, не состоящий в РП. РП: Workspace1	Ошибка: исполнитель не является участником рабочего пространства (рисунок Ж.5)
6	Название: «Разработать новый модуль». Описание: «Разработать модуль авторизации». Дата: будущая дата. Исполнители: user1. РП: Workspace1, пользователь не состоит в РП	Ошибка: вы не являетесь участником этого рабочего пространства (рисунок Ж.6)
7	Название: «Разработать новый модуль». Описание: «Разработать модуль авторизации». Дата: будущая дата. Исполнители: user1, не состоящий в РП. РП: Workspace1, пользователь состоит в РП	Ошибка: исполнитель не является участником рабочего пространства (рисунок Ж.7)
8	Название: «Разработать новый модуль». Описание: «Разработать модуль авторизации». Дата: будущая дата. Исполнители: user1, user2, состоящие в РП. РП: Workspace1, пользователь состоит в РП	Успех: задача успешно создана (рисунок Ж.8)

В результате функционального тестирования было проверено поведение приложения при создании задачи в рабочем пространстве. Программа корректно обрабатывает ошибочные ситуации, связанные с некорректным названием, слишком длинным описанием, датой в прошлом, отсутствием исполнителей и нарушением правил принадлежности пользователей к рабочему пространству. При корректных входных данных задача создается успешно.

4.2.2 Интеграционное тестирование модуля управления задачами

Для проверки корректности взаимодействия компонентов разработанного программного средства было проведено интеграционное тестирование.

Тестирование модуля управления задачами проводилось с использованием языка программирования Python и библиотеки requests, реализующей HTTP-запросы к API Gateway. Тесты охватывают полный жизненный цикл задачи: создание, обновление, управление исполнителями, получение истории изменений и привязку к чатам рабочего пространства. Результаты проведения тестов представлены в таблице 9.

Таблица 9 – Результаты интеграционного тестирования модуля задач

№	Входные данные	Ожидаемый результат	Фактический результат	Статус теста
1	POST /tasks с корректными данными: заголовок, описание, идентификатор рабочего пространства, статус 1	Задача создана, HTTP 201, в теле ответа возвращён идентификатор задачи	Задача создана, HTTP 201, идентификатор задачи получен	Пройден
2	PUT /tasks/{id} с обновлёнными заголовком и описанием	Данные задачи обновлены, HTTP 200	Данные задачи обновлены, HTTP 200	Пройден
3	PUT /tasks/{id}/status со значением статуса 2	Статус задачи изменён, HTTP 200	Статус задачи изменён, HTTP 200	Пройден
4	POST /tasks/{id}/assignees с идентификатором исполнителя	Исполнитель назначен на задачу, HTTP 201	Исполнитель назначен на задачу, HTTP 201	Пройден
5	DELETE /tasks/{id}/assignees/{user_id}	Исполнитель снят с задачи, HTTP 204	Исполнитель снят с задачи, HTTP 204	Пройден
6	GET /tasks/{id}/history	История изменений задачи возвращена, HTTP 200	История изменений задачи возвращена, HTTP 200	Пройден
7	POST /tasks/{id}/chats с идентификатором чата	Задача привязана к чату, HTTP 201	Задача привязана к чату, HTTP 201	Пройден
8	DELETE /tasks/{id}/chats/{chat_id}	Привязка задачи к чату удалена, HTTP 204	Привязка задачи к чату удалена, HTTP 204	Пройден
9	GET /tasks с параметром workspace_id	Возвращён список задач рабочего пространства, созданная задача присутствует в списке	Возвращён список задач рабочего пространства, созданная задача присутствует в списке	Пройден

Все этапы интеграционного тестирования прошли успешно. Программное средство корректно обрабатывает полный жизненный цикл задачи: от создания до удаления привязок, обеспечивая взаимодействие между сервисами задач, чатов и рабочих пространств через единую точку входа.

4.2.3 Анализ метрик качества кода

Для дополнительной оценки качества разработанного программного обеспечения был выполнен расчет метрик исходного кода. Анализ проводился для файлов клиентской части и серверной части приложения. Из анализа

исключались директории `node_modules`, `build`, `dist`, `docs`, а также тестовые, резервные и служебные файлы.

Расчет метрик был выполнен с помощью разработанного инструмента «Code Metrics», исходный код которого представлен в приложении Е. Результаты работы инструмента были проверены вручную,

В результате выполнения анализа было обработано 100 файлов исходного кода. Итоговые значения метрик представлены в таблице 10.

Таблица 10 – Итоговые значения метрик качества кода

Метрика	Итоговое значение по проекту
Количество проанализированных файлов	100
SLOC	13763
Метрика Холстеда, объем	622511.65
Метрика Холстеда, сложность	274.87
Метрика Холстеда, трудоемкость	171112007.95
Метрика Чепина	4603
Количество функций	388

Дополнительно был выполнен анализ метрик по отдельным частям проекта. Результаты представлены в таблице 11.

Таблица 11 – Метрики качества кода по частям проекта

Часть проекта	Файлы	SLOC	Объем Холстеда	Метрика Чепина
Клиентская часть	27	3216	120590.40	577
Микросервис chat	10	2197	89505.87	978
Микросервис workspace	8	1460	55622.40	655
Микросервис task	7	1302	48351.82	583
Микросервис auth	8	947	32390.41	353
Микросервис user	8	931	33632.69	420
Микросервис complaint	7	877	29350.88	412
Микросервис gateway	5	390	15633.90	208

По результатам анализа можно сделать вывод, что наибольший объем кода приходится на клиентскую часть приложения и сервис чатов. Это связано с тем, что клиентская часть содержит всю логику пользовательского интерфейса, а

сервис чатов отвечает за одну из ключевых функций корпоративного мессенджера – обмен сообщениями и работу с участниками чатов.

Высокие значения метрик у сервисов chat, workspace и task объясняются тем, что именно эти компоненты содержат основную бизнес-логику приложения. При этом разделение проекта на отдельные части позволяет локализовать сложность и упростить дальнейшее сопровождение системы.

Таким образом, проведенный расчет метрик подтвердил, что разработанный проект имеет значительный, но структурированный объем исходного кода. Использование метрик SLOC, Холстеда и Чепина позволило количественно оценить сложность приложения и определить наиболее нагруженные части системы. В совокупности с функциональным тестированием это подтверждает корректность работы основных функций корпоративного мессенджера и возможность дальнейшего сопровождения разработанного программного обеспечения [21].

Заключение

В рамках выпускной квалификационной работы было спроектировано и разработано веб-приложение корпоративного мессенджера для конечного пользователя.

Разрабатываемое приложение предназначено для организации внутренней коммуникации сотрудников компании. Система позволяет пользователям авторизоваться, выбирать рабочее пространство, обмениваться сообщениями, создавать чаты, просматривать и контролировать задачи, а также взаимодействовать с другими участниками корпоративной среды в едином информационном пространстве.

В ходе выполнения работы был проведен анализ предметной области, связанной с корпоративными коммуникациями. Также были рассмотрены программные решения аналогичного назначения, выявлены их преимущества и недостатки, на основании чего были сформулированы требования к разрабатываемому приложению.

На этапе проектирования была разработана общая архитектура приложения, определены основные компоненты системы и их назначение. Отдельное внимание было уделено проектированию пользовательского интерфейса, так как разрабатываемое приложение ориентировано на конечного пользователя и должно обеспечивать удобное взаимодействие с основными функциями корпоративного мессенджера.

В процессе разработки были реализованы основные функции веб-приложения: авторизация пользователя, работа с рабочими пространствами, создание и просмотр чатов, обмен сообщениями, создание и изменение задач, а также отображение необходимой пользователю информации в интерфейсе. Приложение построено с учетом требований к безопасности, кроссплатформенности, масштабируемости и удобству дальнейшего сопровождения.

Серверная часть корпоративного мессенджера была реализована в виде шести микросервисов на языке Go с применением реляционной базы данных под управлением PostgreSQL, включая её проектирование и последующее сопровождение.

Разработка клиентской части включала создание веб-приложения на React и TypeScript с использованием Vite, Tailwind CSS.

Исходный код программного обеспечения выложен на сервисе «GitHub» [22].

Для подтверждения корректности работы разработанного программного средства был проведен контроль качества. По результатам тестирования было установлено, что реализованные функции работают корректно и соответствуют поставленным требованиям.

Также был выстроен календарный план разработки и проведена оценка финансовых затрат на реализацию проекта. В рамках планирования проект был декомпозирован на 27 задач; суммарные расходы на разработку составили 228 800 рублей.

Разработанное программной обеспечение было успешно внедрено в производственную деятельность предприятия ООО «Смартвей». Акт о внедрении представлен в приложении 3.

Таким образом, в результате выполнения выпускной квалификационной работы было разработано веб-приложение корпоративного мессенджера для конечного пользователя. Созданное программное средство обеспечивает удобную коммуникацию внутри организации, поддерживает работу с рабочими пространствами и задачами, а также может использоваться как основа для дальнейшего развития корпоративной информационной системы.

Список использованных источников

1. Корпоративная платформа Slack [Электронный ресурс]. – URL: <https://slack.com/features> (дата обращения: 01.06.2026).
2. Битрикс24: инструменты для бизнеса [Электронный ресурс]. – URL: <https://www.bitrix24.ru/features/> (дата обращения: 01.06.2026).
3. Пачка: корпоративный мессенджер [Электронный ресурс]. – URL: <https://pachsa.com/> (дата обращения: 01.06.2026).
4. Балашова, И. Ю. Современные информационные технологии в проектировании программных систем и комплексов : учеб. пособие / И. Ю. Балашова, Д. В. Такташкин ; под ред. П. П. Макарычева. – Пенза : Изд-во ПГУ, 2019. – 106 с.
5. UML – диаграмма вариантов использования (Use Case Diagram) // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/47940/> (дата обращения: 01.06.2026).
6. Ньюмен С. От монолита к микросервисам: Пер. с англ. / С. Ньюмен – СПб. : БХВ-Петербург, 2021. – 272 с.
7. Чикилев Д. Д. Виды архитектуры приложений // Научный лидер. – 2024. – С. 172–176 [Электронный ресурс]. URL: <https://scilead.ru/article/6673-vidi-arkhitekturi-prilozhenij> (дата обращения: 01.06.2026).
8. Создаем диаграмму Ганта // Unisender [Электронный ресурс]. – URL: <https://www.unisender.com/ru/blog/cto-takoe-diagramma-ganta-i-kak-ee-sdelat/> (дата обращения: 01.06.2026).
9. UML: обзор основных типов диаграмм, диаграмма компонентов. Часть 2 // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/756552/> (дата обращения: 01.06.2026).
10. PostgreSQL 16 Documentation [Электронный ресурс]. – URL: <https://www.postgresql.org/docs/16/> (дата обращения: 01.06.2026).

11. Концептуальная модель // Материалы по базам данных [Электронный ресурс]. URL: <https://mlkv.ru/sample-page/материалы-по-базам-данных/концептуальная-модель/> (дата обращения: 01.06.2026).
12. Моделирование данных: обзор // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/articles/556790/> (дата обращения: 01.06.2026).
13. Клиент-серверная архитектура // Hexlet [Электронный ресурс]. URL: https://ru.hexlet.io/courses/internet-fundamentals/lessons/client-server/theory_unit (дата обращения: 01.06.2026).
14. Docker Documentation [Электронный ресурс]. URL: <https://docs.docker.com/> (дата обращения: 01.06.2026).
15. Документация React [Электронный ресурс]. URL: <https://react.dev/learn> (дата обращения: 01.06.2026).
16. Документация TypeScript [Электронный ресурс]. URL: <https://www.typescriptlang.org/docs/> (дата обращения: 01.06.2026).
17. Документация Vite [Электронный ресурс]. – URL: <https://vite.dev/guide/> (дата обращения: 01.06.2026).
18. Документация Tailwind CSS [Электронный ресурс]. URL: <https://tailwindcss.com/docs/installation> (дата обращения: 01.06.2026).
19. Документация языка программирования Go [Электронный ресурс]. URL: <https://go.dev/doc/> (дата обращения: 01.06.2026).
20. Куликов, С. С. Тестирование программного обеспечения. Базовый курс. / С. С. Куликов – Минск : EPAM Systems, 2020. – 299 с.
21. Программный код и его метрики // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/companies/intel/articles/106082/> (дата обращения: 01.06.2026).
22. Исходный код программного обеспечения корпоративного мессенджера // GitHub [Электронный ресурс]. URL : <https://github.com/niksahn/diploma> (дата обращения: 01.06.2026).

Приложение А
(справочное)
Глоссарий

Серверное приложение – программный комплекс, функционирующий на стороне сервера и предоставляющий услуги клиентским компонентам системы.

API (Application Programming Interface) – совокупность соглашений и протоколов, регламентирующих взаимодействие программных компонентов между собой.

REST (Representational State Transfer) – архитектурный подход к проектированию распределённых веб-служб, опирающийся на стандартный набор методов протокола HTTP.

API Gateway – архитектурный элемент, консолидирующий все входящие клиентские запросы через единую точку доступа.

Микросервис – автономный программный модуль с ограниченной зоной ответственности, взаимодействующий с остальными компонентами системы посредством API.

HTTP-запрос – обращение клиента к серверу с целью получения или передачи данных в рамках протокола HTTP.

JWT (JSON Web Token) – открытый стандарт, обеспечивающий безопасную передачу сведений между участниками обмена в форме криптографически подписанного токена.

Бизнес-логика – система правил и алгоритмов, задающих поведение приложения в соответствии с требованиями конкретной предметной области.

Приложение Б
(рекомендуемое)
Спецификации концептуальной схемы базы данных

Таблица Б.1 – Спецификации связей между отношениями концептуальной схемы данных

№ п/п	Наименование	Сущность 2	Семантика связи	Мощность	Обязательность	Ограничения
1	Пользователь	Рабочее пространство	Рабочее пространство состоит из пользователей	M : M	Обязательная для рабочего пространства, необязательная для пользователя	В бригаде должен быть как минимум один пользователь
2	Пользователь	Сообщение	Пользователь может отправлять сообщения	1:M	Обязательная для сообщения, необязательная для сообщения	У сообщения обязательно должен быть один автор
3	Пользователь	Чат	Пользователь может состоять в чатах	M:M	Обязательная для чата, необязательная для пользователя	В чате обязательно должен быть хотя бы один пользователь
4	Пользователь	Задача	Пользователь может создавать задачи	1:M	Обязательная для задачи, необязательная для пользователя	У задачи обязательно должен быть автор
5	Пользователь	Задача	Пользователь может быть назначен на задачу	M:M	Необязательная для пользователя, необязательная для задачи	
6	Задача	Чат	Задача может быть прикреплена к чатам	M:M	Необязательная для задачи, необязательная для чата	
7	Чат	Рабочее пространство	В рабочем пространстве находятся чаты	M:1	Необязательная для рабочего пространства, обязательная для чата	Чат может находиться только в одном рабочем пространстве
8	Пользователь	Жалоба	Пользователь может составлять жалобы	1:M	Необязательная для пользователя, обязательная для жалобы	У жалобы обязательно должен быть автор
9	Рабочее пространство	Администратор	Рабочее пространство создается администратором	M:1	Необязательная для администратора, обязательная для рабочего пространство	У рабочего пространства должен быть указан администратор, создавший его
10	Задача	Изменения задачи	Каждое изменение задачи содержит номер измененной задачи	1:M	Обязательная для задачи, обязательная для изменения задачи	У любой задачи есть хотя бы одна запись изменения – запись о создании, у любого изменения есть номер задачи
11	Рабочее пространство	Тариф	Рабочее пространство привязано к определенному тарифу	M:1	Обязательная для рабочего пространства, необязательная для тарифа	У каждого рабочего пространства должен быть указан тариф

Таблица Б.2 – Спецификация атрибутов отношения Пользователь

№ п/п	Наименование	Описание	Тип данных	Размер (в символах)	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	ID	идентификатор пользователя	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Логин	Логин пользователя, использующийся при входе в систему	VARCHAR	50	SK	+	Уникальное значение	
3	Пароль	Пароль, использующийся пользователем для входа в систему	VARCHAR	100		+		
4	Статус	Статус пользователя в системе	INT	1		+	0 - 1	0
5	Фамилия	Фамилия пользователя	VARCHAR	40		+		
6	Имя	Имя пользователя	VARCHAR	40		+		
7	Отчество	Отчество пользователя	VARCHAR	40		-		

Таблица Б.3 – Спецификация атрибутов отношения Рабочее пространство

№ п/п	Наименование	Описание	Тип данных	Размер (в символах)	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	ID	идентификатор рабочего пространство	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Название	Название рабочего пространства	VARCHAR	100	SK	+		
3	Создатель	Администратор, создавший рабочее пространство	INT	10	FK	+		
4	Тариф	Тариф, по которому работает рабочее пространство	INT	10	FK	+		0

Таблица Б.4 – Спецификация атрибутов отношения Чат

№ п/п	Наименование	Описание	Тип данных	Размер (в символах)	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	ID	идентификатор чата	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Название	Название чата	VARCHAR	100	SK	+		
3	Тип	Тип чата (открытый/закрытый)	INT	10		+	0 - 1	0
4	Рабочее пространство	ID рабочего пространства, в котором находится чат	INT	10	FK	+		0

Таблица Б.5 – Спецификация атрибутов отношения Сообщение

№ п/п	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	ID	Идентификатор сообщения	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Текст	Содержание сообщения	VARCHAR	1000		+		
3	Дата отправления	Дата отправки сообщения	DATE			+		
4	ЧатID	ID чата в который было отправлено сообщение	INT	10		+		
5	ПользовательID	ID отправителя	INT	10		+		
6	Статус	Список прочитавших пользователей	VARCHAR	5000		-		

Таблица Б.6 – Спецификация атрибутов отношения Задача

№	Наименование	Описание	Тип данных	Размер	Роль	Обязательны й	Ограничения значений	Значение по умолчанию
1	ID	идентификатор задачи	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Название	Название задачи	VARCHAR	100		+		
3	Описание	Описание задачи	VARCHAR	500		-		
4	Статус	Статус выполнения задачи	INT	10		+	0 - 4	0
5	Рабочее пространство	ID рабочего пространства в котором создана задача	INT	10	FK	+		
6	Дата создания	Дата создания задачи	DATE			+		
7	ПользовательID	ID создателя задачи	INT	10	FK	+		

Таблица Б.7 – Спецификация атрибутов отношения Тариф

№ п/п	Наименование	Описание	Тип данных	Размер (в символах)	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	ID	Идентификатор тарифа	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Название	Наименование тарифа	VARCHAR	255	SK	+	Уникальное значение	
3	Описание	Условия тарифа	VARCHAR	1000	SK	+	Уникальное значение	

Таблица Б.8 – Спецификация отношения Администратор

№	Наименование	Описание	Тип данных	Размер	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	ID	идентификатор администратора	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Логин	Логин администратора	VARCHAR	100	SK	+	Уникальное значение	
3	Пароль	Пароль администратора	VARCHAR	100		+	Уникальное значение	

Таблица Б.9 – Спецификация отношения Жалоба

№ п/п	Наименование	Описание	Тип данных	Размер (в символах)	Роль	Обязательный	Ограничения значений	Значение по умолчанию
1	ID	Идентификатор администратора	INT	10	PK	+	Уникальное значение	автоматическая генерация
2	Описание	Сообщение от пользователя, описывающее проблему	VARCHAR	255		+		
3	Дата	Время отправки жалобы	timestamp	100		+		
4	Описание устройства	Характеристика устройства отправителя	VARCHAR	255		-		
5	Отправитель	ID пользователя, отправившего жалобу	INT	10		+		

Таблица Б.10 – Спецификации сущностей концептуальной модели

№	Наименование	Описание	Категория	Количество экземпляров	Режим обновления
1	Пользователь	Пользователь мессенджера	Стержневая	1000	Постоянно
2	Администратор	Человек, контролирующий работу приложения	Независимая	1	Нет
3	Рабочее пространство	Область коммуникации внутри одной компании	Стержневая	10	10% в год
4	Чат	Отдельная комната для общения в рабочем пространстве	Стержневая	5000	40% в год
5	Задача	Задача, выделенная в рамках работы одной команды	Стержневая	100	Постоянно
6	Сообщение	Текстовое сообщение внутри чата	Стержневая	100	15% в год
7	Жалоба	Описание проблемы в работе приложения, отправляемое администратору	Стержневая	400	Нет
8	Тариф	Условия, на которых организации предоставлен доступ к мессенджеру	Независимая	5	Нет

Таблица Б.11 – Спецификации связей между сущностями концептуальной схемы данных

№	Сущность 1	Сущность 2	Семантика связи	Мощность	Обязательность	Ограничения
1	Пользователь	Рабочее пространство	В рабочем пространстве находятся пользователи	М : 1	Обязательно для каждой сущности:	Минимум 1 пользователь в РП
2	Пользователь	Чат	Пользователь создает и управляет своими чатами	М : М	- Чате обязательно должен быть хотя бы 1 пользователь	-
3	Пользователь	Сообщения	Пользователь отправляет сообщения	1 : М	- У сообщения обязательно должен быть автор	-
4	Пользователь	Задача	Пользователь создает, управляет задачами и выполняет назначенные на него задачи	1 : М	- У задачи обязательно должен быть автор	-
5	Пользователь	Жалоба	Пользователь отправляет жалобу	1 : М	- У жалобы всегда должен быть автор	-
6	Администратор	Рабочее пространство	Администратор создает и управляет рабочими пространствами	1 : М	-	-
7	Администратор	Жалобы	Администратор читает жалобы пользователей	1 : М	-	-
8	Чаты	Сообщения	Сообщения хранятся в чатах	1 : М	- Сообщение обязательно хранится в конкретном чате	-

Приложение В
(рекомендуемое)
Графическая часть

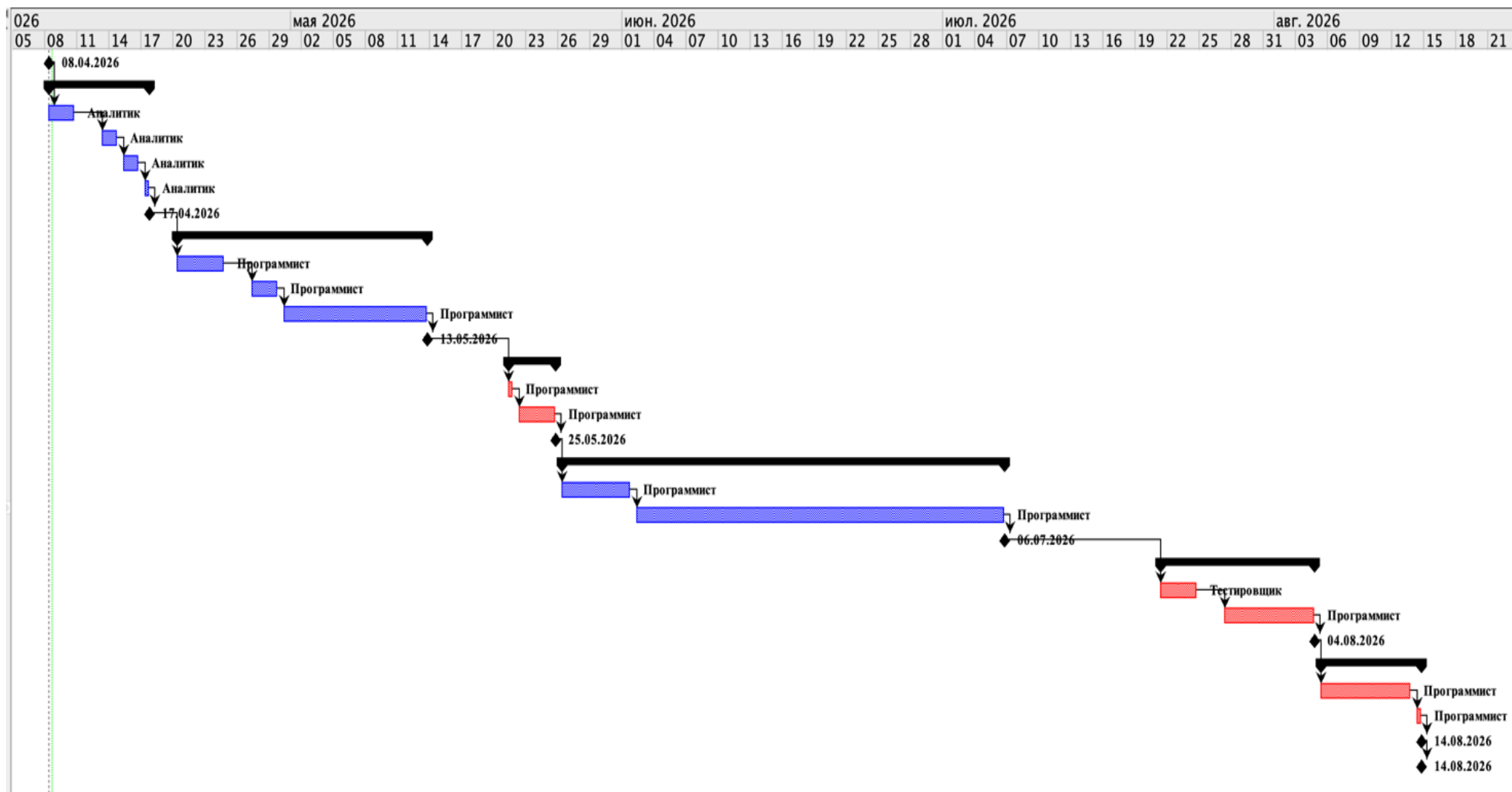


Рисунок В.1 – Диаграмма Ганта

	⑩	Название	Продолжи...	Начало	Окончание	Предшествующие	Название ресурса
1		Начало проекта	0 дней	08.04.2026, 09:00	08.04.2026, 09:00		
2		Анализ предметной области и постановка задачи	8 дней	08.04.2026, 09:00	17.04.2026, 18:00		
3		Анализ систем-аналогов	3 дней	08.04.2026, 09:00	10.04.2026, 18:00	1	Аналитик
4		Анализ предметной области и постановка задачи	2 дней	13.04.2026, 09:00	14.04.2026, 18:00	3	Аналитик
5		Анализ функциональных требований	2 дней	15.04.2026, 09:00	16.04.2026, 18:00	4	Аналитик
6		Анализ нефункциональных требований	1 день	17.04.2026, 09:00	17.04.2026, 18:00	5	Аналитик
7		Анализ предметной области и постановка задачи завершены	0 дней	17.04.2026, 18:00	17.04.2026, 18:00	6	
8		Проектирование программных средств	18 дней	20.04.2026, 09:00	13.05.2026, 18:00		
9		Выбор технологий и инструментальных средств	5 дней	20.04.2026, 09:00	24.04.2026, 18:00	7	Программист
10		Проектирование базы данных	3 дней	27.04.2026, 09:00	29.04.2026, 18:00	9	Программист
11		Проектирование серверной части корпоративного мессенджера	10 дней	30.04.2026, 09:00	13.05.2026, 18:00	10	Программист
12		Проектирование программных средств завершено	0 дней	13.05.2026, 18:00	13.05.2026, 18:00	11	
13		Подготовка механизма развертывания приложения	3 дней	21.05.2026, 09:00	25.05.2026, 18:00		
14		Выбор метода развертывания	1 день	21.05.2026, 09:00	21.05.2026, 18:00	12	Программист
15		Реализация механизма развертывания	2 дней	22.05.2026, 09:00	25.05.2026, 18:00	14	Программист
16		Подготовка механизма развертывания приложения завершена	0 дней	25.05.2026, 18:00	25.05.2026, 18:00	15	
17		Реализация программных средств	30 дней	26.05.2026, 09:00	06.07.2026, 18:00		
18		Реализация базы данных и миграций	5 дней	26.05.2026, 09:00	01.06.2026, 18:00	16	Программист
19		Реализация серверной части приложения	25 дней	02.06.2026, 09:00	06.07.2026, 18:00	18	Программист
20		Реализация программных средств завершена	0 дней	06.07.2026, 18:00	06.07.2026, 18:00	19	
21		Тестирование программных средств	11 дней	21.07.2026, 09:00	04.08.2026, 18:00		
22		Функциональное тестирование	4 дней	21.07.2026, 09:00	24.07.2026, 18:00	20	Тестировщик
23		Модульное тестирование	7 дней	27.07.2026, 09:00	04.08.2026, 18:00	22	Программист
24		Тестирование завершено	0 дней	04.08.2026, 18:00	04.08.2026, 18:00	23	
25		Завершение работы над проектом	8 дней	05.08.2026, 09:00	14.08.2026, 18:00		
26		Составление документации	7 дней	05.08.2026, 09:00	13.08.2026, 18:00	24	Программист
27		Сдача проекта	1 день	14.08.2026, 09:00	14.08.2026, 18:00	26	Программист
28		Работа над проектом завершена	0 дней	14.08.2026, 18:00	14.08.2026, 18:00	27	
29		Конец проекта	0 дней	14.08.2026, 18:00	14.08.2026, 18:00	28	

Рисунок В.2 – Список задач проекта

Название	Оставшаяся стоимость
Начало проекта	0,00 ₽
Анализ предметной области и поста	19200,00 ₽
Анализ систем-аналогов	7200,00 ₽
Анализ предметной области и поста	4800,00 ₽
Анализ функциональных требовани	4800,00 ₽
Анализ нефункциональных требован	2400,00 ₽
Анализ предметной области и поста	0,00 ₽
Проектирование программных сред	54720,00 ₽
Выбор технологий и инструментальн	15200,00 ₽
Проектирование базы данных	9120,00 ₽
Проектирование серверной части к	30400,00 ₽
Проектирование программных сред	0,00 ₽
Подготовка механизма развертыван	9120,00 ₽
Выбор метода развертывания	3040,00 ₽
Реализация механизма развертыван	6080,00 ₽
Подготовка механизма развертыван	0,00 ₽
Реализация программных средств	91200,00 ₽
Реализация базы данных и миграци	15200,00 ₽
Реализация серверной части прилож	76000,00 ₽
Реализация программных средств з	0,00 ₽
Тестирование программных средств	30240,00 ₽
Функциональное тестирование	8960,00 ₽
Модульное тестирование	21280,00 ₽
Тестирование завершено	0,00 ₽
Завершение работы над проектом	24320,00 ₽

Рисунок В.3 – Оценка бюджета

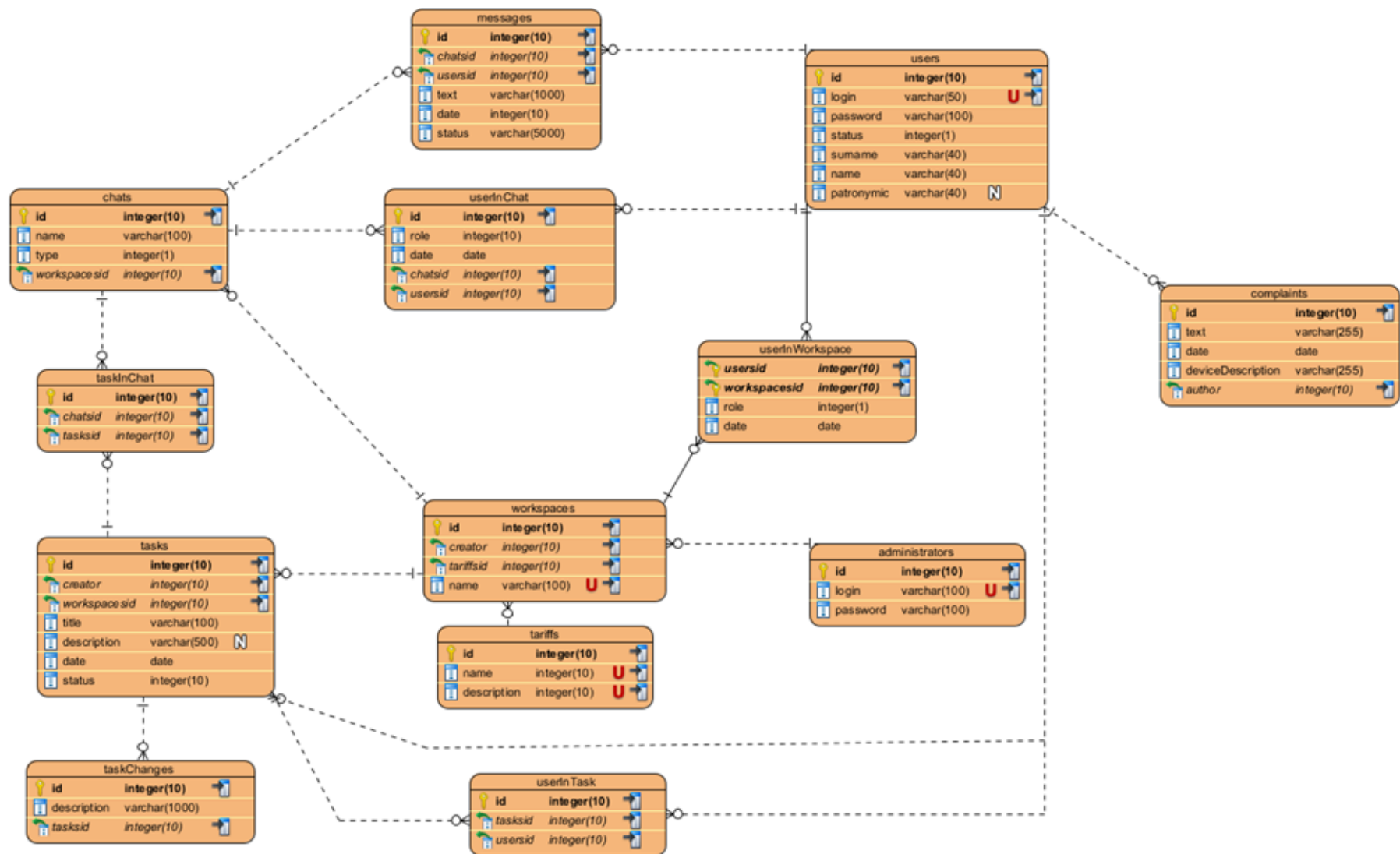


Рисунок В.4 – Физическая модель БД

Приложение Г
(обязательное)

Листинг серверной части приложения

Листинг Г.1 – Файл models

```
package models

import (
    "database/sql"
    "time"
)

// Task представляет задачу
type Task struct {
    ID            int            `db:"id"`
    Creator       int            `db:"creator"`
    WorkspaceID   int            `db:"workspaceid"`
    Title         string         `db:"title"`
    Description    *string        `db:"description"`
    Date          sql.NullTime   `db:"date"`
    Status        int            `db:"status"`
    CreatedAt     time.Time      `db:"created_at,omitempty"`
}

// TaskWithDetails представляет задачу с дополнительной
информацией
type TaskWithDetails struct {
    ID            int            `db:"id"`
    Creator       int            `db:"creator"`
    CreatorName   string         `db:"creator_name"`
    CreatorSurname string        `db:"creator_surname"`
    WorkspaceID   int            `db:"workspace_id"`
    WorkspaceName string        `db:"workspace_name"`
    Title         string         `db:"title"`
    Description    *string        `db:"description"`
    Date          sql.NullTime   `db:"date"`
    Status        int            `db:"status"`
    AssigneeCount int            `db:"assignee_count"`
    ChatCount     int            `db:"chat_count"`
    CreatedAt     time.Time      `db:"created_at"`
    CompletedAt   sql.NullTime   `db:"completed_at"`
}

// TaskAssignee представляет исполнителя задачи
type TaskAssignee struct {
    UserID       int            `db:"user_id"`
    Login        string         `db:"login"`
    Name         string         `db:"name"`
    Surname      string         `db:"surname"`
    AssignedAt   string         `db:"assigned_at"`
}

// TaskChat представляет чат, связанный с задачей
type TaskChat struct {
```

```

    ChatID      int      `db:"chat_id"`
    ChatName     string   `db:"chat_name"`
    ChatType     int      `db:"chat_type"`
    WorkspaceID int      `db:"workspace_id"`
    AttachedAt   string   `db:"attached_at"`
}

// TaskChange представляет запись истории изменений задачи
type TaskChange struct {
    ID          int      `db:"id"`
    Description string   `db:"description"`
    TaskID      int      `db:"tasksid"`
    ChangedAt   time.Time `db:"changed_at,omitempty"`
}

const (
    TaskStatusCreated   = 1
    TaskStatusInWork    = 2
    TaskStatusReview    = 3
    TaskStatusCompleted = 4
    TaskStatusCancelled = 5
)

func IsValidTaskStatus(status int) bool {
    return status >= TaskStatusCreated && status <=
TaskStatusCancelled
}

func GetTaskStatusName(status int) string {
    switch status {
    case TaskStatusCreated:
        return "Создана"
    case TaskStatusInWork:
        return "В работе"
    case TaskStatusReview:
        return "На проверке"
    case TaskStatusCompleted:
        return "Завершена"
    case TaskStatusCancelled:
        return "Отменена"
    default:
        return "Неизвестный статус"
    }
}

```

Листинг Г.2 – Файл models

```

package models

import "time"

type CreateTaskRequest struct {

```

```

        WorkspaceID    int        `json:"workspace_id" binding:"required"`
        Title           string     `json:"title"
binding:"required,min=3,max=100"`
        Description    *string    `json:"description,omitempty"`
        Status         int        `json:"status,omitempty"`
        AssignedUsers  []int      `json:"assigned_users,omitempty"`
        ChatID         *int        `json:"chat_id,omitempty"`
    }

    type UpdateTaskRequest struct {
        Title           *string    `json:"title,omitempty"`
        Description *string    `json:"description,omitempty"`
    }

    type UpdateTaskStatusRequest struct {
        Status int `json:"status" binding:"required,min=1,max=5"`
    }

    type AddTaskAssigneesRequest struct {
        UserIDs []int `json:"user_ids" binding:"required,min=1"`
    }

    type AttachTaskToChatRequest struct {
        ChatID int `json:"chat_id" binding:"required"`
    }

    type TaskResponse struct {
        ID            int        `json:"id"`
        Creator       int        `json:"creator"`
        CreatorName   string     `json:"creator_name"`
        WorkspaceID   int        `json:"workspace_id"`
        WorkspaceName string     `json:"workspace_name"`
        Title         string     `json:"title"`
        Description   *string    `json:"description,omitempty"`
        Date          *time.Time `json:"date,omitempty"`
        Status        int        `json:"status"`
        StatusName    string     `json:"status_name"`
        AssigneeCount int        `json:"assignee_count"`
        ChatCount     int        `json:"chat_count"`
        CreatedAt     time.Time  `json:"created_at"`
        CompletedAt   *time.Time `json:"completed_at,omitempty"`
    }

    type TaskListResponse struct {
        Tasks []TaskResponse `json:"tasks"`
        Total int          `json:"total"`
    }

    type TaskAssigneeResponse struct {
        UserID    int    `json:"user_id"`
        Login     string `json:"login"`
        Name      string `json:"name"`
        Surname   string `json:"surname"`
    }

```

```

    Patronymic *string `json:"patronymic,omitempty"`
    AssignedAt string `json:"assigned_at"`
}

type TaskAssigneesResponse struct {
    Assignees []TaskAssigneeResponse `json:"assignees"`
    Total     int                       `json:"total"`
}

type TaskChatResponse struct {
    ChatID      int    `json:"chat_id"`
    ChatName    string `json:"chat_name"`
    ChatType    int    `json:"chat_type"`
    WorkspaceID int    `json:"workspace_id"`
    AttachedAt  string `json:"attached_at"`
}

type TaskChatsResponse struct {
    Chats []TaskChatResponse `json:"chats"`
    Total int             `json:"total"`
}

type TaskChangeResponse struct {
    ID          int    `json:"id"`
    Description string `json:"description"`
    TaskID      int    `json:"task_id"`
    ChangedAt   time.Time `json:"changed_at"`
}

type TaskHistoryResponse struct {
    Changes []TaskChangeResponse `json:"changes"`
    Total   int             `json:"total"`
}

type ErrorResponse struct {
    Error string `json:"error"`
}

type SuccessResponse struct {
    Message string `json:"message"`
    Data    interface{} `json:"data,omitempty"`
}

```

Листинг Г.3 – Файл repository

```

package repository

import (
    "context"
    "fmt"

    "github.com/diploma/task-service/data/database"

```

```

        "github.com/diploma/task-service/data/models"
        "github.com/jackc/pgx/v5"
    )

    type Repository struct {
        db *database.DB
    }

    func NewRepository(db *database.DB) *Repository {
        return &Repository{db: db}
    }

    func (r *Repository) CreateTask(ctx context.Context, task
    *models.Task) (*models.Task, error) {
        query := `
            INSERT INTO tasks (creator, workspacesid, title,
description, date, status)
            VALUES ($1, $2, $3, $4, $5, $6)
            RETURNING id, creator, workspacesid, title, description,
date, status
        `

        err := r.db.Pool.QueryRow(ctx, query,
            task.Creator, task.WorkspaceID, task.Title,
            task.Description, task.Date, task.Status,
        ).Scan(
            &task.ID, &task.Creator, &task.WorkspaceID,
            &task.Title, &task.Description, &task.Date, &task.Status,
        )
        if err != nil {
            return nil, fmt.Errorf("failed to create task: %w", err)
        }
        changeDesc := fmt.Sprintf("Задача создана: %s", task.Title)
        r.addTaskChange(ctx, task.ID, changeDesc)
        return task, nil
    }

    func (r *Repository) GetTasksByWorkspace(
        ctx context.Context, workspaceID, userID int,
    ) ([]models.TaskWithDetails, error) {
        query := `
            SELECT t.id, t.creator,
                u.surname || ' ' || u.name as creator_name,
                u.surname as creator_surname,
                t.workspacesid as workspace_id,
                w.name as workspace_name,
                t.title, t.description, t.date, t.status,
                COALESCE((SELECT COUNT(*) FROM "userintask"
                    WHERE tasksid = t.id), 0) as
assignee_count,
                COALESCE((SELECT COUNT(*) FROM "taskinchat"
                    WHERE tasksid = t.id), 0) as chat_count,
                t.created_at, t.completed_at
            FROM tasks t
        `
    }

```

```

        INNER JOIN users u ON t.creator = u.id
        INNER JOIN workspaces w ON t.workspacesid = w.id
        INNER JOIN "userinworkspace" uiw
            ON w.id = uiw.workspacesid AND uiw.usersid = $2
        WHERE t.workspacesid = $1
        ORDER BY t.created_at DESC, t.id DESC
    `

    rows, err := r.db.Pool.Query(ctx, query, workspaceID, userID)
    if err != nil {
        return nil, fmt.Errorf("failed to get tasks: %w", err)
    }
    defer rows.Close()

    var tasks []models.TaskWithDetails
    for rows.Next() {
        var task models.TaskWithDetails
        if err := rows.Scan(
            &task.ID, &task.Creator, &task.CreatorName,
            &task.CreatorSurname,
            &task.WorkspaceID, &task.WorkspaceName, &task.Title,
            &task.Description, &task.Date, &task.Status,
            &task.AssigneeCount, &task.ChatCount,
            &task.CreatedAt, &task.CompletedAt,
        ); err != nil {
            return nil, fmt.Errorf("failed to scan task: %w", err)
        }
        tasks = append(tasks, task)
    }
    return tasks, nil
}

func (r *Repository) addTaskChange(
    ctx context.Context, taskID int, description string,
) error {
    _, err := r.db.Pool.Exec(ctx,
        `INSERT INTO "taskchanges" (description, tasksid) VALUES
        ($1, $2)`,
        description, taskID,
    )
    return err
}

```

Листинг Г.4 – Файл task_handler

```
package handlers
```

```
import (
    "database/sql"
    "net/http"
    "strconv"
    "time"

```

```

    dm "github.com/diploma/task-service/data/models"
    "github.com/diploma/task-service/data/repository"
    "github.com/diploma/task-service/presentation/models"
    "github.com/gin-gonic/gin"
)

type TaskHandler struct {
    repo *repository.Repository
}

func NewTaskHandler(repo *repository.Repository) *TaskHandler {
    return &TaskHandler{repo: repo}
}

func getUserID(c *gin.Context) (int, error) {
    userIDStr := c.GetHeader("X-User-ID")
    if userIDStr == "" {
        return 0, nil
    }
    return strconv.Atoi(userIDStr)
}

func (h *TaskHandler) GetTasks(c *gin.Context) {
    userID, err := getUserID(c)
    if err != nil || userID == 0 {
        c.JSON(http.StatusUnauthorized, models.ErrorResponse{Error: "unauthorized"})
        return
    }
    workspaceIDStr := c.Query("workspace_id")
    if workspaceIDStr == "" {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error: "workspace_id is required"})
        return
    }
    workspaceID, err := strconv.Atoi(workspaceIDStr)
    if err != nil {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error: "invalid workspace_id"})
        return
    }
    ctx := c.Request.Context()
    if err := h.repo.ValidateUserInWorkspace(ctx, userID, workspaceID); err != nil {
        c.JSON(http.StatusForbidden, models.ErrorResponse{Error: "user is not a member of workspace"})
        return
    }
    tasks, err := h.repo.GetTasksByWorkspace(ctx, workspaceID, userID)
    if err != nil {

```

```

        c.JSON(http.StatusInternalServerError,
models.ErrorResponse{Error: "failed to get tasks"})
        return
    }
    var taskResponses []models.TaskResponse
    for _, task := range tasks {
        taskResponses = append(taskResponses,
h.convertToTaskResponse(&task))
    }
    c.JSON(http.StatusOK, models.TaskListResponse{
        Tasks: taskResponses,
        Total: len(taskResponses),
    })
}

func (h *TaskHandler) UpdateTask(c *gin.Context) {
    userID, err := getUserID(c)
    if err != nil || userID == 0 {
        c.JSON(http.StatusUnauthorized,
models.ErrorResponse{Error: "unauthorized"})
        return
    }
    taskID, err := strconv.Atoi(c.Param("id"))
    if err != nil {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error:
"invalid task id"})
        return
    }
    var req models.UpdateTaskRequest
    if err := c.ShouldBindJSON(&req); err != nil {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error:
"invalid request data"})
        return
    }
    ctx := c.Request.Context()
    task, err := h.repo.GetTaskByID(ctx, taskID, userID)
    if err != nil {
        if err.Error() == "task not found" {
            c.JSON(http.StatusNotFound,
models.ErrorResponse{Error: "task not found"})
            return
        }
    }
    c.JSON(http.StatusInternalServerError,
models.ErrorResponse{Error: "failed to get task"})
    return
}
    if task.Creator != userID {
        c.JSON(http.StatusForbidden, models.ErrorResponse{Error:
"only task creator can update it"})
        return
    }
    if err := h.repo.UpdateTask(ctx, taskID, req.Title,
req.Description); err != nil {

```

```

        c.JSON(http.StatusInternalServerError,
models.ErrorResponse{Error: "failed to update task"})
        return
    }
    c.JSON(http.StatusOK, models.SuccessResponse{Message: "task
updated successfully"})
}

func (h *TaskHandler) UpdateTaskStatus(c *gin.Context) {
    userID, err := getUserID(c)
    if err != nil || userID == 0 {
        c.JSON(http.StatusUnauthorized,
models.ErrorResponse{Error: "unauthorized"})
        return
    }
    taskID, err := strconv.Atoi(c.Param("id"))
    if err != nil {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error:
"invalid task id"})
        return
    }
    var req models.UpdateTaskStatusRequest
    if err := c.ShouldBindJSON(&req); err != nil {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error:
"invalid request data"})
        return
    }
    if !dm.IsValidTaskStatus(req.Status) {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error:
"invalid task status"})
        return
    }
    ctx := c.Request.Context()
    if _, err = h.repo.GetTaskByID(ctx, taskID, userID); err !=
nil {
        if err.Error() == "task not found" {
            c.JSON(http.StatusNotFound,
models.ErrorResponse{Error: "task not found"})
            return
        }
        c.JSON(http.StatusInternalServerError,
models.ErrorResponse{Error: "failed to get task"})
        return
    }
    if err := h.repo.UpdateTaskStatus(ctx, taskID, req.Status);
err != nil {
        c.JSON(http.StatusInternalServerError,
models.ErrorResponse{Error: "failed to update task
status"})
        return
    }
    c.JSON(http.StatusOK, models.SuccessResponse{Message: "task
status updated successfully"})
}

```

```

}

func (h *TaskHandler) DeleteTask(c *gin.Context) {
    userID, err := getUserID(c)
    if err != nil || userID == 0 {
        c.JSON(http.StatusUnauthorized,
models.ErrorResponse{Error: "unauthorized"})
        return
    }
    taskID, err := strconv.Atoi(c.Param("id"))
    if err != nil {
        c.JSON(http.StatusBadRequest, models.ErrorResponse{Error:
"invalid task id"})
        return
    }
    ctx := c.Request.Context()
    task, err := h.repo.GetTaskByID(ctx, taskID, userID)
    if err != nil {
        if err.Error() == "task not found" {
            c.JSON(http.StatusNotFound,
models.ErrorResponse{Error: "task not found"})
            return
        }
        c.JSON(http.StatusInternalServerError,
models.ErrorResponse{Error: "failed to get task"})
        return
    }
    if task.Creator != userID {
        c.JSON(http.StatusForbidden, models.ErrorResponse{Error:
"only task creator can delete it"})
        return
    }
    if err := h.repo.DeleteTask(ctx, taskID); err != nil {
        c.JSON(http.StatusInternalServerError,
models.ErrorResponse{Error: "failed to delete task"})
        return
    }
    c.Status(http.StatusNoContent)
}

func (h *TaskHandler) convertToTaskResponse(task
*dm.TaskWithDetails) models.TaskResponse {
    var completed *time.Time
    if task.CompletedAt.Valid {
        t := task.CompletedAt.Time
        completed = &t
    }
    var datePtr *time.Time
    if task.Date.Valid {
        t := task.Date.Time
        datePtr = &t
    }
    return models.TaskResponse{

```

```
ID:                task.ID,
Creator:           task.Creator,
CreatorName:       task.CreatorName,
WorkspaceID:       task.WorkspaceID,
WorkspaceName:     task.WorkspaceName,
Title:             task.Title,
Description:        task.Description,
Date:              datePtr,
Status:            task.Status,
StatusName:        dm.GetTaskStatusName(task.Status),
AssigneeCount:     task.AssigneeCount,
ChatCount:         task.ChatCount,
CreatedAt:         task.CreatedAt,
CompletedAt:       completed,
```

Приложение Д
(обязательное)

Листинг клиентской части приложения

Листинг Д.1 – Файл App

```

import { Navigate, Route, Routes } from 'react-router-dom'
import AuthPage      from './pages/AuthPage'
import WorkspacesPage from './pages/WorkspacesPage'
import ChatsPage      from './pages/ChatsPage'
import ChatPage       from './pages/ChatPage'
import TasksPage      from './pages/TasksPage'
import TaskDetailPage from './pages/TaskDetailPage'
import TaskCreatePage from './pages/TaskCreatePage'
import MembersPage    from './pages/MembersPage'
import ComplaintsPage from './pages/ComplaintsPage'
import ProfilePage     from './pages/ProfilePage'
import NotFoundPage    from './pages/NotFoundPage'
import AppLayout       from './shared/layouts/AppLayout'
import ProtectedRoute  from './shared/components/ProtectedRoute'

function App() {
  return (
    <Routes>
      <Route path="/auth" element={<AuthPage />} />
      <Route element={<ProtectedRoute />>
        <Route element={<AppLayout />>
          <Route index element={<Navigate to="/workspaces"
replace />} />
            <Route path="/workspaces"
element={<WorkspacesPage />} />
              <Route path="/chats"                element={<ChatsPage
/>} />
                <Route path="/chats/:chatId"        element={<ChatPage />}
/>
                  <Route path="/tasks"                element={<TasksPage
/>} />
                    <Route path="/tasks/new"
element={<TaskCreatePage />} />
                      <Route path="/tasks/:taskId"
element={<TaskDetailPage />} />
                        <Route path="/members"        element={<MembersPage
/>} />
                          <Route path="/complaints"
element={<ComplaintsPage />} />
                            <Route path="/profile"    element={<ProfilePage
/>} />
                              </Route>
                                </Route>
                                  <Route path="*" element={<NotFoundPage />} />
                                </Routes>
                              )
        }
      )
    </Routes>
  )
}

export default App

```

Листинг Д.2 – Файл auth

```
import { create } from 'zustand'
import { persist } from 'zustand/middleware'

export type UserProfile = {
  id: string
  login: string
  name?: string
  surname?: string
  patronymic?: string
  status?: string
}

type AuthState = {
  token: string | null
  refreshToken: string | null
  user: UserProfile | null
  setAuth: (token: string, refreshToken?: string | null,
    user?: UserProfile | null) => void
  setToken: (token: string) => void
  setUser: (user: UserProfile | null) => void
  logout: () => void
}

export const useAuthStore = create<AuthState>() (
  persist(
    (set) => ({
      token: null,
      refreshToken: null,
      user: null,
      setAuth: (token, refreshToken = null, user = null) =>
        set({ token, refreshToken, user }),
      setToken: (token) => set({ token }),
      setUser: (user) => set({ user }),
      logout: () => set({ token: null, refreshToken:
null, user: null }),
    }),
    { name: 'auth-store' },
  ),
)
```

Листинг Д.3 – Файл workspace

```
import { create } from 'zustand'
import { persist } from 'zustand/middleware'

type WorkspaceState = {
  selectedWorkspaceId: number | null
  selectedWorkspaceName: string | null
  selectedWorkspaceRole: number | null
  setSelectedWorkspace: (id: number | null,
    role?: number | null,
```

```

                                name?: string | null) => void
  reset: () => void
}

export const useWorkspaceStore = create<WorkspaceState>()(
  persist(
    (set) => ({
      selectedWorkspaceId: null,
      selectedWorkspaceName: null,
      selectedWorkspaceRole: null,
      setSelectedWorkspace: (id, role = null, name = null) =>
        set({
          selectedWorkspaceId: id,
          selectedWorkspaceRole: role,
          selectedWorkspaceName: name ?? null,
        }),
      reset: () =>
        set({
          selectedWorkspaceId: null,
          selectedWorkspaceName: null,
          selectedWorkspaceRole: null,
        }),
    }),
    { name: 'workspace-store' },
  ),
)

```

Листинг Д.4 – Файл client

```

import { useAuthStore } from '../state/auth'
import { authApi } from './auth'

const API_BASE_URL = import.meta.env.VITE_API_BASE_URL ||
'http://localhost:8080'
const API_PREFIX = '/api/v1'

type RequestOptions = RequestInit & { skipAuthHeader?: boolean }

let isRefreshing = false
let refreshPromise: Promise<string> | null = null

async function refreshToken(): Promise<string> {
  const refreshToken = useAuthStore.getState().refreshToken
  if (!refreshToken) throw new Error('No refresh token
available')

  try {
    const response = await authApi.refresh({ refresh_token:
refreshToken })
    const newToken = response.access_token
    useAuthStore.getState().setToken(newToken)
    return newToken
  } catch (error) {

```

```

    useAuthStore.getState().logout()
    if (typeof window !== 'undefined') {
      window.location.href = '/auth'
    }
    throw error
  }
}

export async function request<T>(
  path: string,
  options: RequestOptions = {},
): Promise<T> {
  const { skipAuthHeader, ...init } = options
  const headers = new Headers(init.headers || {})

  let token = useAuthStore.getState().token
  const shouldAttachAuth = token && !skipAuthHeader

  if (!headers.has('Content-Type') && !(init.body instanceof
FormData)) {
    headers.set('Content-Type', 'application/json')
  }
  if (shouldAttachAuth) {
    headers.set('Authorization', `Bearer ${token}`)
  }

  const url = path.startsWith('http')
    ? path
    : `${API_BASE_URL}${API_PREFIX}${path.startsWith('/') ? path
: `/${path}`}`

  let response = await fetch(url, { ...init, headers })

  if (response.status === 401
    && !skipAuthHeader
    && useAuthStore.getState().refreshToken) {
    if (!isRefreshing) {
      isRefreshing = true
      refreshPromise = refreshToken().finally(() => {
        isRefreshing = false
        refreshPromise = null
      })
    }
    try {
      const newToken = await refreshPromise!
      headers.set('Authorization', `Bearer ${newToken}`)
      response = await fetch(url, { ...init, headers })
    } catch (error) {
      throw error
    }
  }

  const text = await response.text()

```

```

const data = text ? safeJson(text) : null

if (!response.ok) {
  const message =
    (data as { message?: string })?.message ??
    `Request failed: ${response.status}`
  throw new Error(message)
}

return data as T
}

function safeJson(payload: string) {
  try { return JSON.parse(payload) }
  catch { return payload }
}

this.ws.onerror = (e) => this.onError?.(e)
}

disconnect() {
  this.ws?.close()
  this.ws = null
  this.chatId = null
}

send(data: unknown) {
  if (this.ws?.readyState === WebSocket.OPEN) {
    this.ws.send(JSON.stringify(data))
  }
}

onMessageReceived(cb: (msg: Message) => void) {
this.onMessage = cb }
onConnected      (cb: () => void) {
this.connectedCallback = cb }
onErrorReceived  (cb: (e: Event) => void) { this.onError
= cb }
onConnectionClosed(cb: () => void) { this.onClose
= cb }

private attemptReconnect() {
  if (this.reconnectAttempts >= this.maxReconnectAttempts ||
!this.chatId) return
  this.reconnectAttempts++
  setTimeout(() => this.connect(this.chatId!),
    this.reconnectDelay * this.reconnectAttempts)
}

get isConnected() { return this.ws?.readyState ===
WebSocket.OPEN }
}

```

```

export const chatApi = {
  list: (workspaceId: number) =>
    request<Chat[]>(`/chats?workspace_id=${workspaceId}`),
  create: (p: { name: string; workspace_id: number; type:
number; members: number[] }) =>
    request<Chat>(`/chats`, { method: 'POST', body:
JSON.stringify(p) }),
  getDetails: (chatId: number) =>
    request<ChatDetails>(`/chats/${chatId}`),
  members: (chatId: number) =>
    request<{ members: ChatMember[]; total: number
}>(`/chats/${chatId}/members`),
  addMembers: (chatId: number, p: { user_ids: number[]; role:
number }) =>
    request<{ added: number[]; chat_id: number
}>(`/chats/${chatId}/members`,
    { method: 'POST', body: JSON.stringify(p) }),
  messages: (chatId: number) =>
    request<{ messages: Message[]; has_more: boolean; total:
number }>(
    `/chats/${chatId}/messages`).then(r => r.messages),
  sendMessage: (chatId: number, text: string) =>
    request<Message>(`/chats/${chatId}/messages`,
    { method: 'POST', body: JSON.stringify({ text }) }),
  tasks: (chatId: number) =>
    request<{ tasks: unknown[]; total: number
}>(`/chats/${chatId}/tasks`),
}

```

Приложение Е
(обязательное)

Листинг инструмента для подсчета метрик качества кода

```
#!/usr/bin/env node

const fs = require('fs')
const path = require('path')
const Parser = require('tree-sitter')
const Go = require('tree-sitter-go')
const TsGrammars = require('tree-sitter-typescript')
const fg = require('fast-glob')

const ROOT = process.cwd()

const GO_KEYWORDS = new Set([
  'break', 'case', 'chan', 'const', 'continue', 'default',
  'defer', 'else', 'fallthrough',
  'for', 'func', 'go', 'goto', 'if', 'import', 'interface', 'map',
  'package', 'range',
  'return', 'select', 'struct', 'switch', 'type', 'var',
])

const TS_KEYWORDS = new Set([
  'as', 'async', 'await', 'break', 'case', 'catch', 'class',
  'const', 'continue', 'debugger',
  'default', 'delete', 'do', 'else', 'enum', 'export', 'extends',
  'finally', 'for', 'from',
  'function', 'if', 'implements', 'import', 'in', 'instanceof',
  'interface', 'let', 'new',
  'of', 'return', 'static', 'super', 'switch', 'throw', 'try',
  'type', 'typeof', 'var',
  'void', 'while', 'yield',
])

const SYMBOL_OPERATORS = new Set([
  '=', ':=', '+=', '-=', '*=', '/=', '%=', '&=', '|=', '^=',
  '<<=', '>>=', '&&=', '||=',
  '??=', '==', '!=', '===', '!==', '<', '<=', '>', '>=', '+', '-',
  '*', '/', '%', '&&',
  '||', '!', '&', '|', '^', '~', '<<', '>>', '++', '--', '?', ':',
  '=>', '.', '...', '<-',
])

const OPERAND_NODE_TYPES = new Set([
  'identifier', 'type_identifier', 'field_identifier',
  'property_identifier', 'shorthand_property_identifier',
  'shorthand_property_identifier_pattern', 'nested_identifier',
  'this', 'super', 'true', 'false',
  'null', 'undefined', 'nil', 'int_literal', 'float_literal',
  'imaginary_literal', 'rune_literal',
  'raw_string_literal', 'interpreted_string_literal', 'string',
  'string_fragment', 'template_string',
  'template_substitution', 'number', 'regex',
  'private_property_identifier',
])
```

```

const LITERAL_NODE_TYPES = new Set([
  'int_literal', 'float_literal', 'imaginary_literal',
  'rune_literal', 'raw_string_literal',
  'interpreted_string_literal', 'number', 'string',
  'template_string', 'regex', 'true',
  'false', 'null', 'undefined', 'nil',
])

const FUNCTION_NODE_TYPES = new Set([
  'function_declaration', 'method_declaration', 'func_literal',
  'function_expression', 'arrow_function',
  'generator_function_declaration',
  'generator_function', 'method_definition',
])

const CONTROL_NODE_TYPES = new Set([
  'if_statement', 'for_statement', 'for_in_statement',
  'while_statement', 'do_statement',
  'switch_statement', 'switch_expression', 'case_clause',
  'case_statement', 'ternary_expression',
])

(const role of roles.values()) {
  const klass = classifyChepinRole(role)
  totals[klass] += 1
}

return false
})

totals.Q = round2(totals.P + (2 * totals.M) + (3 * totals.C) +
(0.5 * totals.T))
return totals
}

function registerParameters(functionNode, roles, source) {
  walk(functionNode, (node) => {
    if (node === functionNode) {
      return true
    }
    if (FUNCTION_NODE_TYPES.has(node.type)) {
      return false
    }
    if (node.type !== 'parameter_list' && node.type !==
'formal_parameters') {
      return true
    }

    for (const identifier of findIdentifiers(node, source, {
includeFields: false })) {
      ensureRole(roles, identifier).input = true
    }
  })
}

```

```

        return false
    })
}

function round2(value) {
    return Math.round(value * 100) / 100
}

function printReport(payload) {
    console.log('=== Metrics Report ===')
    console.log(`Generated at: ${payload.generatedAt}`)
    console.log(`Files analyzed: ${payload.scope.files}`)
    console.log(`Included: ${payload.scope.included.join(', ')}`)
    console.log(`Excluded: ${payload.scope.excluded.join(', ')}`)
    console.log('')
    console.log('Project totals:')
    printGroup(payload.totals)
    console.log('')
    console.log('By group:')
    for (const group of payload.groups) {
        printGroup(group)
        console.log('')
    }
}

function printGroup(group) {
    console.log(`- ${group.name}`)
    console.log(`  Files: ${group.files}`)
    console.log(`  SLOC: ${group.sloc}`)
    console.log(
        `    Halstead: n1=${group.halstead.n1}, n2=${group.halstead.n2},`
        `N1=${group.halstead.N1}, N2=${group.halstead.N2}, ` +
        `    vocabulary=${group.halstead.vocabulary},`
        `length=${group.halstead.length}, volume=${group.halstead.volume},`
        ` +`
        `    difficulty=${group.halstead.difficulty},`
        `effort=${group.halstead.effort}`
    )
    console.log(
        `    Chepin: P=${group.chepin.P}, M=${group.chepin.M},`
        `C=${group.chepin.C}, T=${group.chepin.T}, ` +
        `    Q=${group.chepin.Q}, functions=${group.chepin.functions}`
    )
}

main()

```

Приложение Ж
(рекомендуемое)

Результаты функционального тестирования

[illegible]

Рисунок Ж.2 – Результаты тестирования класса эквивалентности «Описание задачи превышает 5000 символов»

Корпоративный мессенджер

Рабочие пространства

Задачи

Участники

Жалобы

Профиль

ЧАТЫ

Нет чатов

Выйти

Текущее рабочее пространство
Workspace1

testadmin

Создание задачи

Заполните основные поля — название и срок обязательны. Исполнители и чат можно выбрать по желанию.

Основная информация

Название задачи *

Разработать новый модуль

Описание

Разработать модуль авторизации

Дата выполнения *

01.01.2020

Статус

Начальный статус

К выполнению

Дату завершения система проставит автоматически при переводе задачи в статус «Выполнена».

Исполнители и чат (необязательно)

Исполнители

☒ Иванов Иван (user1)

☒ Петров Петр (user2)

☐ Сахно Никита (testadmin)

Привязать к чату

Без чата

Ошибка: дата выполнения не может быть в прошлом

Создать задачу

Отмена

Рисунок Ж.3 – Результаты тестирования класса эквивалентности «Дата выполнения указана в прошлом»

Корпоративный мессенджер

Рабочие пространства

Задачи

Участники

Жалобы

Профиль

ЧАТЫ

Нет чатов

Выйти

Текущее рабочее пространство
Workspace1

testadmin

Создание задачи

Заполните основные поля — название и срок обязательны. Исполнители и чат можно выбрать по желанию.

Основная информация

Название задачи *

Разработать новый модуль

Описание

Разработать модуль авторизации

Дата выполнения *

03.07.2026

Статус

Начальный статус

К выполнению

Дату завершения система проставит автоматически при переводе задачи в статус «Выполнена».

Исполнители и чат (необязательно)

Исполнители

☐ Иванов Иван (user1)

☐ Петров Петр (user2)

☐ Сахно Никита (testadmin)

Привязать к чату

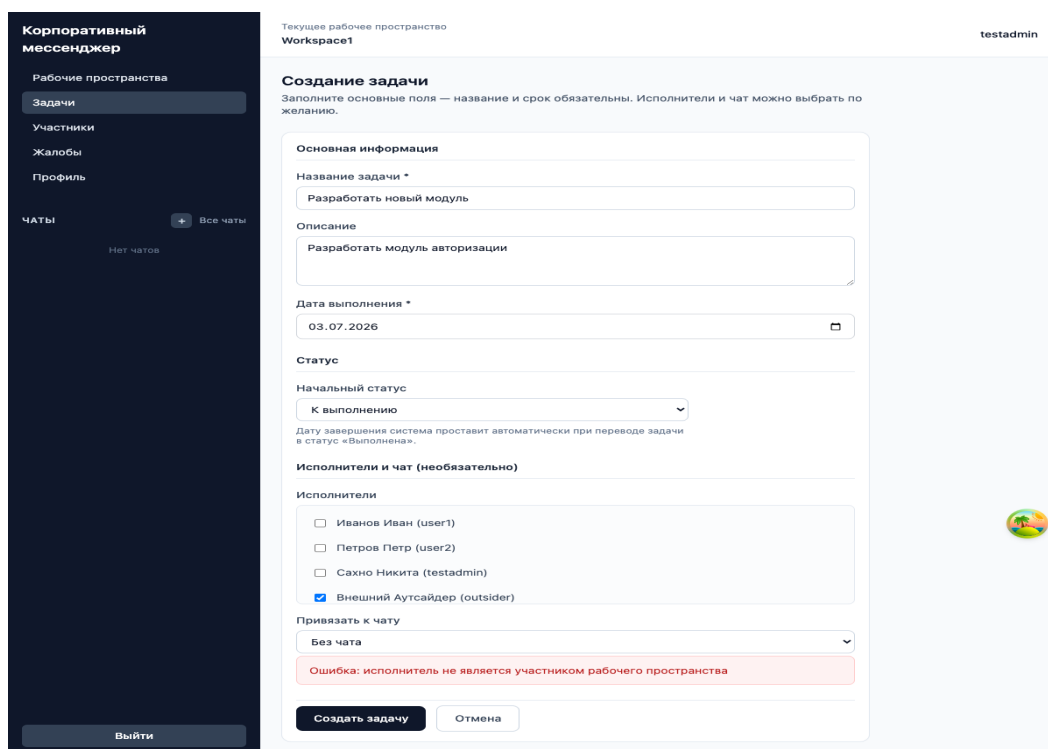
Без чата

Ошибка: необходимо назначить хотя бы одного исполнителя

Создать задачу

Отмена

Рисунок Ж.4 – Результаты тестирования класса эквивалентности «Исполнители не назначены»



Корпоративный мессенджер

Рабочие пространства

Задачи

Участники

Жалобы

Профиль

ЧАТЫ

Нет чатов

Выйти

Текущее рабочее пространство
Workspace1

testadmin

Создание задачи

Заполните основные поля — название и срок обязательны. Исполнители и чат можно выбрать по желанию.

Основная информация

Название задачи *

Разработать новый модуль

Описание

Разработать модуль авторизации

Дата выполнения *

03.07.2026

Статус

Начальный статус

К выполнению

Дату завершения система проставит автоматически при переводе задачи в статус «Выполнена».

Исполнители и чат (необязательно)

Исполнители

☐ Иванов Иван (user1)

☐ Петров Петр (user2)

☐ Сахно Никита (testadmin)

☒ Внешний Аутсайдер (outsider)

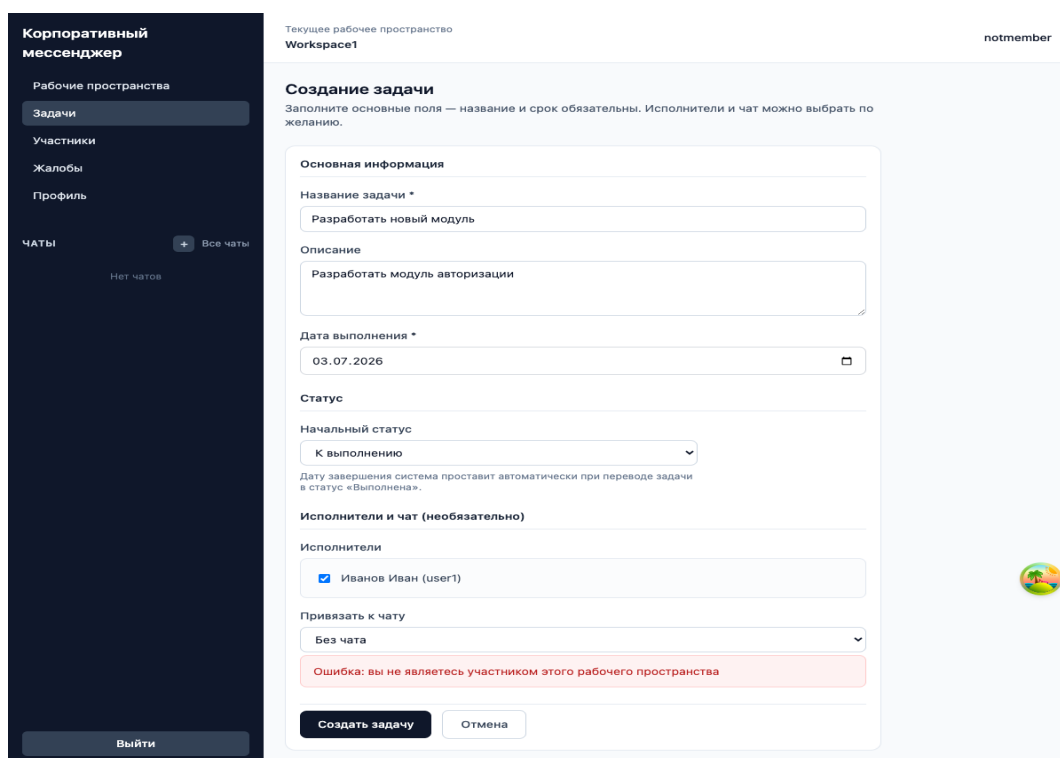
Привязать к чату

Без чата

Ошибка: исполнитель не является участником рабочего пространства

Создать задачу Отмена

Рисунок Ж.5 – Результаты тестирования класса эквивалентности «Исполнитель не является участником рабочего пространства»



Корпоративный мессенджер

Рабочие пространства

Задачи

Участники

Жалобы

Профиль

ЧАТЫ

Нет чатов

Выйти

Текущее рабочее пространство
Workspace1

notmember

Создание задачи

Заполните основные поля — название и срок обязательны. Исполнители и чат можно выбрать по желанию.

Основная информация

Название задачи *

Разработать новый модуль

Описание

Разработать модуль авторизации

Дата выполнения *

03.07.2026

Статус

Начальный статус

К выполнению

Дату завершения система проставит автоматически при переводе задачи в статус «Выполнена».

Исполнители и чат (необязательно)

Исполнители

☒ Иванов Иван (user1)

Привязать к чату

Без чата

Ошибка: вы не являетесь участником этого рабочего пространства

Создать задачу Отмена

Рисунок Ж.6 – Результаты тестирования класса эквивалентности «Текущий пользователь не состоит в рабочем пространстве»

Корпоративный мессенджер

Рабочие пространства

Задачи

Участники

Жалобы

Профиль

ЧАТЫ

Все чаты

Нет чатов

Выйти

Текущее рабочее пространство
Workspace1

testadmin

Создание задачи

Заполните основные поля — название и срок обязательны. Исполнители и чат можно выбрать по желанию.

Основная информация

Название задачи *

Разработать новый модуль

Описание

Разработать модуль авторизации

Дата выполнения *

03.07.2026

Статус

Начальный статус

К выполнению

Дату завершения система проставит автоматически при переводе задачи в статус «Выполнена».

Исполнители и чат (необязательно)

Исполнители

☒ Иванов Иван (user1)

Привязать к чату

Без чата

Ошибка: исполнитель не является участником рабочего пространства

Создать задачу Отмена

Рисунок Ж.7 – Результаты тестирования класса эквивалентности «Исполнитель не является участником рабочего пространства»

Корпоративный мессенджер

Рабочие пространства

Задачи

Участники

Жалобы

Профиль

ЧАТЫ

Все чаты

Нет чатов

Выйти

Текущее рабочее пространство
Workspace1

testadmin

Разработать новый модуль

Задача #7

Редактировать

Описание

Разработать модуль авторизации

Дата завершения

— Проставляется при статусе «Выполнена»

Статус

Создана

Изменить статус

Исполнители

+ Добавить

Иванов Иван
@user1
Назначен: 01.08.2001

Петров Петр
@user2
Назначен: 01.09.2001

Прикрепленные чаты

+ Прикрепить

Задача не прикреплена ни к одному чату

Создатель: Сахно Никита
Рабочее пространство: Workspace1
Создана: 03.06.2026, 21:47:40

Рисунок Ж.8 – Результаты тестирования класса эквивалентности «Задача успешно создана»

Приложение 3
(рекомендуемое)

Акт внедрения программного средства в эксплуатацию

АКТ ВНЕДРЕНИЯ программного продукта в эксплуатацию

г. Пенза

«05» июня 2026 г.

Настоящий акт составлен о том, что программный продукт «Корпоративный мессенджер "КМ-1"», разработанный Сахно Никитой Сергеевичем и Новосельцев Александром Андреевичем в рамках выполнения дипломного проекта, был внедрён в эксплуатацию в ООО «Смартвей» по адресу: г. Пенза, ул. Московская, д. 39.

Программный продукт внедрён в деятельность Отдела разработки клиентских продуктов, а именно в работу Команды мобильной разработки. В ходе внедрения разработанное веб-приложение корпоративного мессенджера было успешно интегрировано в рабочие процессы команды и используется для организации внутренних коммуникаций, управления задачами и рабочими пространствами, а также повышения эффективности взаимодействия между участниками проектных групп.

По результатам эксплуатации установлено, что программный продукт функционирует корректно, соответствует поставленным задачам и может быть рекомендован для дальнейшего использования в производственной деятельности подразделения.

Ответственным за внедрение программного продукта является:
Сахно Никита Сергеевич

Руководитель
Моисеев

Максим

подразделения:
Геннадьевич.

Разработчик	Сахно Н.С.
Разработчик	Новосельцев А.А.
Руководитель подразделения	Моисеев М.Г.
Подпись / дата	

