АНТИПЛАГИАТ
ОБНАРУЖЕНИЕ ЗАИМСТВОВАНИЙ

Отчет предоставлен сервисом «Антиплагиат» - http://isea.antiplagiat.ru

Отчет о проверке

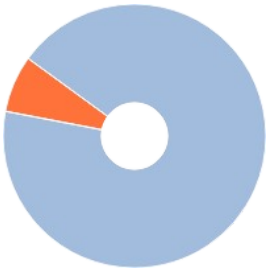
Автор: Малежик Д Д

Название документа: diplom_malezhik

Проверяющий: Печерина Александра Валерьевна

Организация: Байкальский государственный университет

РЕЗУЛЬТАТЫ ПРОВЕРКИ



Совпадения: 6,79%

Оригинальность: 93,21%

Цитирования: 0%

Самоцитирования: 0%

«Совпадения», «Цитирования», «Самоцитирования», «Оригинальность» являются отдельными показателями, отображаются в процентах и в сумме дают 100%, что соответствует проверенному тексту документа.

Проверено: 94,29% текста документа, исключено из проверки: 5,71% текста документа. Разделы, отключенные пользователем: Титульный лист, Содержание, Библиография

- **Совпадения** — фрагменты проверяемого текста, полностью или частично сходные с найденными источниками, за исключением фрагментов, которые система отнесла к цитированию или самоцитированию. Показатель «Совпадения» – это доля фрагментов проверяемого текста, отнесенных к совпадениям, в общем объеме текста.
- **Самоцитирования** — фрагменты проверяемого текста, совпадающие или почти совпадающие с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа. Показатель «Самоцитирования» – это доля фрагментов текста, отнесенных к самоцитированию, в общем объеме текста.
- **Цитирования** — фрагменты проверяемого текста, которые не являются авторскими, но которые система отнесла к корректно оформленным. К цитированиям относятся также шаблонные фразы; библиография; фрагменты текста, найденные модулем поиска «СПС Гарант: нормативно-правовая документация». Показатель «Цитирования» – это доля фрагментов проверяемого текста, отнесенных к цитированию, в общем объеме текста.
- **Текстовое пересечение** — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
- **Источник** — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
- **Оригинальный текст** — фрагменты проверяемого текста, не обнаруженные ни в одном источнике и не отмеченные ни одним из модулей поиска. Показатель «Оригинальность» – это доля фрагментов проверяемого текста, отнесенных к оригинальному тексту, в общем объеме текста.

Обращаем Ваше внимание, что система находит текстовые совпадения проверяемого документа с проиндексированными в системе источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности совпадений или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

Номер документа: 273

Тип документа: Не указано

Дата проверки: 03.06.2025 07:32:28

Дата корректировки: Нет

Количество страниц: 79

Символов в тексте: 96722

Слов в тексте: 11404

Число предложений: 3651

Комментарий: не указано

ПАРАМЕТРЫ ПРОВЕРКИ

Выполнена проверка с учетом редактирования: Да

Исключение элементов документа из проверки: Нет

Выполнено распознавание текста (OCR): Нет

Выполнена проверка с учетом структуры: Да

Модули поиска: Переводные заимствования, Шаблонные фразы, СПС ГАРАНТ: нормативно-правовая документация, ИПС Адилет, Коллекция НБУ, СМИ России и СНГ, Патенты СССР, РФ, СНГ, IEEE, Диссертации НББ, Рувики, Сводная коллекция ЭБС, Цитирование, Медицина, СПС ГАРАНТ: аналитика, Публикации eLIBRARY, Перефразирования по коллекции IEEE, Публикации РГБ, Публикации РГБ (переводы и перефразирования), Кольцо вузов, Переводные заимствования IEEE, Публикации eLIBRARY (переводы и перефразирования), Кольцо вузов (переводы и перефразирования), Переводные заимствования по коллекции Интернет в английском сегменте, Переводные заимствования по коллекции Интернет в русском сегменте, Переводные заимствования по коллекции Гарант: аналитика, Перефразирования по СПС ГАРАНТ: аналитика, Перефразированные заимствования по коллекции Интернет в русском сегменте, Перефразированные заимствования по коллекции Интернет в английском сегменте, Интернет Плюс, Собственная коллекция компании, Собственная коллекция (переводы и перефразирования)

ИСТОЧНИКИ

№	Доля в тексте	Доля в отчете	Источник	Актуален на	Модуль поиска	Комментарий
[01]	3,32%	2,26%	РАЗРАБОТКА ПРОГРАММЫ ДЛЯ ...	27 Июн 2023	Собственная коллекция (переводы и перефразирования)	
[02]	1,83%	1,23%	4_%D0%9F%D1%80%D0%BE%D0%... https://bgu.ru	10 Апр 2025	Перефразированные заимствования по коллекции Интернет в русском сегменте	
[03]	1,72%	0%	https://bgu-chita.ru/sites/default/... https://bgu-chita.ru	30 Мая 2025	Перефразированные заимствования по коллекции Интернет в русском сегменте	
[04]	1,6%	0,07%	ПРОЕКТИРОВАНИЕ И РАЗРАБОТК...	13 Июн 2024	Собственная коллекция (переводы и перефразирования)	
[05]	1,51%	0%	РАЗРАБОТКА АВТОМАТИЗИРОВА...	16 Июн 2024	Собственная коллекция (переводы и перефразирования)	
[06]	1,25%	0,09%	ПРОЕКТИРОВАНИЕ И РАЗРАБОТК...	10 Июн 2024	Собственная коллекция компании	
[07]	1,25%	0,03%	РАЗРАБОТКА ПРОГРАММЫ ДЛЯ ...	27 Июн 2023	Собственная коллекция компании	
[08]	1,14%	0%	Золотухина_ВКР_09.06	13 Июн 2022	Собственная коллекция (переводы и перефразирования)	
[09]	1,03%	0%	СОЗДАНИЕ ПЛАТФОРМЫ ДЛЯ ПР...	08 Июн 2024	Собственная коллекция (переводы и перефразирования)	
[10]	1%	0,63%	09-142 - Малышев Роман Евгень...	20 Мая 2024	Кольцо вузов (переводы и перефразирования)	
[11]	0,98%	0%	Читинский институт Байкальско... https://ru.ruwiki.ru	раньше 2011	Рувики	
[12]	0,8%	0,37%	ПРОЕКТИРОВАНИЕ МЕССЕНДЖЕР...	12 Дек 2024	Публикации eLIBRARY (переводы и перефразирования)	
[13]	0,8%	0%	Основные сведения | Читински... http://xn--90aen0cq.xn--p1ai	11 Апр 2020	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[14]	0,8%	0%	Основные сведения | Читински... http://bgu-chita.ru	12 Фев 2020	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[15]	0,8%	0%	Основные сведения | Читински... http://bgu-chita.ru	22 Июн 2020	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[16]	0,8%	0%	Основные сведения | Читински... http://xn--90aen0cq.xn--p1ai	26 Янв 2021	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[17]	0,8%	0%	Основные сведения | Читински... https://web.archive.org	03 Июн 2025	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[18]	0,8%	0%	Otchet_po_practike_Panteleev (3)	20 Сен 2018	Собственная коллекция компании	Источник исключен. Причина: Маленький процент пересечения.
[19]	0,8%	0%	Otchet_po_practike_Panteleev (1)	19 Сен 2018	Собственная коллекция компании	Источник исключен. Причина: Маленький процент пересечения.
[20]	0,79%	0%	РАЗРАБОТКА АВТОМАТИЗИРОВА...	16 Июн 2024	Собственная коллекция компании	

[21]	0,75%	0%	ПРОЕКТИРОВАНИЕ И РАЗРАБОТК...	10 Июн 2024	Собственная коллекция (переводы и перефразирования)	
[22]	0,74%	0,74%	ВВЕДЕНИЕ В ПРОГРАММНУЮ ИН...	30 Мая 2025	Публикации eLIBRARY (переводы и перефразирования)	
[23]	0,74%	0%	ПРОЕКТИРОВАНИЕ И РАЗРАБОТК...	20 Фев 2025	Собственная коллекция (переводы и перефразирования)	
[24]	0,65%	0%	Байкальский государственный у... https://ru.ruwiki.ru	раньше 2011	Рувики	
[25]	0,58%	0%	Основные сведения | Читински... https://bgu-chita.ru	19 Мая 2025	Переводные заимствования по коллекции Интернет в русском сегменте	
[26]	0,55%	0%	Otchet_po_practike_Panteleev (1)	19 Сен 2018	Собственная коллекция (переводы и перефразирования)	
[27]	0,52%	0%	Otchet_po_practike_Panteleev (3)	20 Сен 2018	Собственная коллекция (переводы и перефразирования)	
[28]	0,52%	0%	не указано	29 Сен 2022	Шаблонные фразы	Источник исключен. Причина: Маленький процент пересечения.
[29]	0,45%	0%	ПРОЕКТИРОВАНИЕ И РАЗРАБОТК...	21 Июн 2023	Собственная коллекция (переводы и перефразирования)	
[30]	0,44%	0,44%	Разработка веб-сайта для станц...	20 Мая 2025	Кольцо вузов (переводы и перефразирования)	
[31]	0,44%	0,15%	Основные сведения | Читински... https://bgu-chita.ru	19 Мая 2025	Перефразированные заимствования по коллекции Интернет в русском сегменте	
[32]	0,42%	0%	Основой для создания в Иркутск... http://altairk.ru	15 Янв 2016	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[33]	0,36%	0,36%	О защите вообще и ЭЦП в частн...	21 Мар 2025	Перефразирования по СПС ГАРАНТ: аналитика	
[34]	0,36%	0,13%	kravchuk_s_p_sozdanie-bezopasn...	02 Июн 2025	Кольцо вузов (переводы и перефразирования)	
[35]	0,36%	0%	ПРИНЯТИЕ РЕШЕНИЙ О ТРАНСФ...	07 Дек 2023	Публикации eLIBRARY (переводы и перефразирования)	
[36]	0,34%	0%	ПРОЕКТИРОВАНИЕ И РАЗРАБОТК...	20 Июн 2023	Собственная коллекция компании	Источник исключен. Причина: Маленький процент пересечения.
[37]	0,32%	0%	E-XCELLENCE NEXT Report Review... https://core.ac.uk	19 Янв 2023	Переводные заимствования по коллекции Интернет в английском сегменте	
[38]	0,3%	0%	Постановление Томаринского ра... http://arbitr.garant.ru	30 Янв 2021	СПС ГАРАНТ: нормативно-правовая документация	Источник исключен. Причина: Маленький процент пересечения.
[39]	0,26%	0%	не указано	13 Янв 2022	Цитирование	Источник исключен. Причина: Маленький процент пересечения.
[40]	0,25%	0,25%	https://miem.hse.ru/data/2018/06... https://miem.hse.ru	30 Мая 2023	Переводные заимствования по коллекции Интернет в русском сегменте	
[41]	0,25%	0,01%	final_ermosh_dissertaciya_0.pdf?s... https://elib.sfu-kras.ru	11 Мая 2025	Переводные заимствования по коллекции Интернет в русском сегменте	
[42]	0,24%	0%	6596_ВКР_Коловертнов	07 Мая 2025	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[43]	0,24%	0%	Реализация свободы слова в ци... http://ivo.garant.ru	20 Ноя 2021	Переводные заимствования по коллекции Гарант: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[44]	0,22%	0%	ВКР Иваненко Ю.А. ЗзПРм	18 Янв 2025	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[45]	0,21%	0%	Разработка информационной си...	14 Июн 2019	Собственная коллекция компании	Источник исключен. Причина: Маленький процент пересечения.
[46]	0,2%	0%	Муссель К.М. Платёжные технол...	06 Фев 2025	Перефразирования по СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[47]	0,17%	0%	Уведомление о возобновлении д... http://chita.bezformata.ru	12 Апр 2017	СМИ России и СНГ	Источник исключен. Причина: Маленький процент пересечения.
[48]	0,17%	0%	3966-6558-1-SM.pdf	11 Июл 2021	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.

[49]	0,17%	0%	3285-5264-1-SM.pdf	05 Авг 2021	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[50]	0,17%	0%	3447-5571-2-SM.docx	25 Июн 2021	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[51]	0,17%	0%	https://bgu-chita.ru/sites/default/... https://bgu-chita.ru	26 Мая 2025	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[52]	0,17%	0%	Технология вовлечения управле... https://book.ru	01 Янв 2022	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[53]	0,17%	0%	Технология вовлечения управле... https://book.ru	01 Янв 2024	Сводная коллекция ЭБС	Источник исключен. Причина: Маленький процент пересечения.
[54]	0,17%	0%	Барабанова С.В., Пешкова (Белог... http://ivo.garant.ru	30 Ноя 2019	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[55]	0,17%	0%	Комментарий к Федеральному з... http://ivo.garant.ru	01 Апр 2023	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[56]	0,16%	0%	Владимирский государственный... https://ru.ruwiki.ru	раньше 2011	Рувики	Источник исключен. Причина: Маленький процент пересечения.
[57]	0,16%	0%	РЕЗУЛЬТАТЫ ПРОВЕДЕНИЯ В 202... http://elibrary.ru	01 Янв 2023	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[58]	0,16%	0%	Высшее юридическое образован... https://elibrary.ru	18 Апр 2023	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[59]	0,16%	0%	ВКР Артемов А.Ю.	06 Мая 2025	Кольцо вузов	Источник исключен. Причина: Маленький процент пересечения.
[60]	0,15%	0%	АКТУАЛЬНЫЕ ПРОБЛЕМЫ И ПЕР... http://elibrary.ru	01 Янв 2016	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[61]	0,14%	0%	Решение Якутского городского су... http://arbitr.garant.ru	07 Дек 2019	СПС ГАРАНТ: нормативно-правовая документация	Источник исключен. Причина: Маленький процент пересечения.
[62]	0,14%	0%	Всё об образовании: сборник но...	19 Дек 2016	Медицина	Источник исключен. Причина: Маленький процент пересечения.
[63]	0,14%	0%	Изменения в законодательстве, ... http://ivo.garant.ru	12 Авг 2023	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[64]	0,14%	0%	Коронавирус COVID-19 http://ivo.garant.ru	08 Фев 2020	СПС ГАРАНТ: аналитика	Источник исключен. Причина: Маленький процент пересечения.
[65]	0,14%	0%	Итоги Всероссийского конкурса ... http://elibrary.ru	31 Авг 2017	Публикации eLIBRARY	Источник исключен. Причина: Маленький процент пересечения.
[66]	0,14%	0%	Е. А. Матвеева, А. Р. Диязитдино... http://dlib.rsl.ru	01 Янв 2016	Публикации РГБ	Источник исключен. Причина: Маленький процент пересечения.
[67]	0,13%	0%	rid_bf3411558c4f4cb0821290702f... http://elib.rshu.ru	07 Апр 2025	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.
[68]	0,11%	0%	Отчет по результатам диагности... http://tuvsu.ru	08 Дек 2017	Интернет Плюс	Источник исключен. Причина: Маленький процент пересечения.

Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «БАЙКАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ЧИТИНСКИЙ ИНСТИТУТ
Кафедра информационных технологий и высшей математики

Диплом!

обучающегося бакалавриата группы ИСТУ-21 Малежик Д.Д.

Руководитель(-и) по практической подготовке от Института:

преподаватель кафедры информационных технологий и высшей математики Куклина О.К.

Чита, 2025

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В эпоху быстрой цифровой трансформации потребность в безопасных, эффективных и гибких внутренних системах коммуникации становится все более заметной в различных секторах, включая образование. Поскольку учебные заведения продолжают модернизировать свою деятельность, необходимость в оптимизации коммуникации между студентами, преподавателями и административными отделами становится ключевым фактором цифровых инноваций. Обмен сообщениями, документами, уведомлениями, расписаниями и конфиденциальными данными должен быть не только быстрым и удобным, но и безопасным и соответствовать стандартам защиты данных.

Растущая сложность этих процессов коммуникации требует специализированных программных решений, которые можно адаптировать к внутренней структуре организации. Традиционные методы коммуникации, такие как электронная почта или универсальные мессенджеры, часто не соответствуют конкретным требованиям образовательных рабочих процессов, где основное внимание уделяется взаимодействию в реальном времени, контролю доступа, простоте использования и способности адаптироваться к внутренним иерархиям и правилам.

Данный дипломный проект посвящен проектированию и разработке защищенного клиент-серверного приложения для обмена сообщениями, специально предназначенного для упрощения структурированной коммуникации в образовательном учреждении, а именно в Читинском институте, филиале Байкальского государственного университета. Система разработана как легкая, модульная и безопасная платформа обмена сообщениями в режиме реального времени, с легкой расширяемостью и удобным дизайном, адаптированным к ситуации академического и административного общения.

1. Постановка проблемы

Несмотря на наличие различных внешних платформ обмена сообщениями, учебные заведения часто сталкиваются с ограничениями в их использовании из-за проблем с конфиденциальностью, отсутствия интеграции с внутренними системами или отсутствия контроля над хранением данных. Кроме того, правительственные постановления и требования к безопасности рабочей информации часто запрещают использование сторонних решений для обработки внутренних сообщений, которые могут включать конфиденциальные или персональные данные. В результате возникает очевидная потребность в настраиваемом внутреннем решении, которое обеспечивает общение в реальном времени, владение данными, контроль доступа и возможное шифрование — все в пределах безопасных цифровых границ учреждения.

2. Цель проекта

Основной целью проекта является разработка и реализация кроссплатформенного клиент-серверного приложения, обеспечивающего безопасную связь в режиме реального времени между подразделениями и отдельными лицами в институте. Система должна работать на основе современных веб-технологий и придерживаться исключительно безопасных протоколов связи, оставаясь при этом доступной и интуитивно понятной для своих пользователей.

Первоначально разработанная во время производственной и преддипломной стажировок студентов, система уже включает в себя рабочий фронтенд и бэкенд, подключенные через технологию WebSocket, в частности библиотеку Socket.IO. Приложение поддерживает несколько чатов — как публичных, так и частных — с чистым, минимальным интерфейсом на основе React и архитектурой, которая поощряет повторное использование компонентов и модульность.

3. Актуальность

Актуальность этого проекта заключается в его соответствии текущим тенденциям цифровой инфраструктуры в образовании, особенно в свете постпандемических сдвигов в сторону гибридного обучения и удаленного администрирования. Более того, проект служит прототипом для масштабируемых инструментов внутренней коммуникации, которые могут быть приняты не только в образовательных контекстах, но и в государственных и коммерческих организациях с аналогичными потребностями в контролируемых, частных и работающих в режиме реального времени средах обмена сообщениями.

Кроме того, используя открытые технологии и разрабатывая индивидуальное решение с нуля, проект избегает привязки к поставщику и поддерживает независимость учреждения с точки зрения развертывания, интеграции и будущего развития. Система разработана с учетом расширяемости, что позволяет использовать такие функции, как вложения сообщений, роли пользователей, ведение журнала и интеграция со сторонними системами аутентификации, такими как каталоги пользователей на основе Microsoft SQL Server, используемые во многих государственных учреждениях.

4. Методы исследования

Разработка этого безопасного мессенджера проводилась на основе системного анализа, сравнительной оценки и итеративного прототипирования. Во-первых, внутренние рабочие процессы коммуникации изучались посредством интервью и обзоров документов для получения как функциональных, так и нефункциональных требований. Затем сравнивались существующие решения для обмена сообщениями для выявления архитектурных шаблонов и недостатков безопасности. Эти идеи были переведены в формальные модели, такие как схемы использования и последовательности UML, для фиксации структуры и поведения системы. Реализации прототипов с использованием React, Node.js, Express и Socket.IO были

разработаны итерациями, что позволило непрерывно тестировать доставку сообщений в реальном времени, обработку ошибок и производительность под нагрузкой. Наконец, предварительные криптографические исследования дали информацию о гибридной стратегии шифрования, которая в конечном итоге обеспечит безопасность всех обменов данными, в то время как постоянные оценки удобства использования гарантировали, что интерфейс останется интуитивно понятным и отзывчивым для конечных пользователей

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. История и структура учреждения

Разработка веб-приложения осуществлялась в рамках отдела управления учебно-методического и информационного обеспечения, которое является одним из ключевых структурных подразделений института. Данный отдел отвечает за цифровую трансформацию Читинского института и выполняет ряд важнейших функций, в том числе:

- проектирование и внедрение автоматизированных информационных систем;
- поддержание и развитие «АСУ ВУЗ» (автоматизированной системы управления образовательным процессом);
- совершенствование процессов планирования, документооборота и управления информационными потоками в институте;
- взаимодействие с другими подразделениями и руководством университета по вопросам цифровизации.

Полное официальное наименование института: «Читинский институт (филиал) федерального государственного бюджетного образовательного учреждения высшего образования «Байкальский государственный университет» (сокращенно: ЧИ ФГБОУ ВО «БГУ»). Расположен по адресу: г. Чита, ул. Анохина, 56.

Институт берет свое начало в 1985 году, когда Министерство высшего и среднего специального образования РСФСР издало Приказ № 390 от 14 июня 1985 года. В соответствии с этим приказом Иркутский институт народного хозяйства создал филиал в Чите приказом № 85 от 20 ноября 1985 года. С тех пор институт претерпел несколько переименований, отражавших изменения в структуре его головного учреждения:

В 1997 году филиал был переименован в «Читинский институт (филиал) Иркутской государственной экономической академии» (Приказ № 1215).

В 2002 году он стал «Читинский институт (филиал) Байкальского государственного университета экономики и права» (Приказ № 1550 Министерства образования Российской Федерации от 26 апреля 2002 года).

В 2015 году институт получил свое нынешнее название «Читинский институт (филиал) Байкальского государственного университета» (Приказ Министерства образования и науки Российской Федерации от 29 октября 2015 г. № 1252).

В 2010 году в состав института вошел Торгово-экономический колледж (ранее ТИЭП Чита, БГУ|ЭП). В декабре 2015 года колледж был переименован в «Колледж Читинского института (филиала) БГУ» и продолжает свою деятельность под этим названием по адресу г. Чита, ул. Лермонтова, д. 12.

Сегодня институт активно использует цифровые каналы для общения с абитуриентами и студентами. На официальных интернет-ресурсах — сайте института, канале в Telegram и группе ВКонтакте — публикуются новости об учебной и внеучебной жизни, выкладываются документы и расписания, предоставляется доступ к личным кабинетам студентов.

Изучая историю института и его развивающуюся роль в региональном образовании, мы можем увидеть, как его организационная структура адаптировалась на протяжении десятилетий для удовлетворения меняющихся академических потребностей, расширения контингента студентов и использования цифровых платформ для коммуникации и администрирования.

Основная миссия Читинского института — подготовка квалифицированных специалистов в области экономики, права, управления и информационных технологий. Институт предлагает программы высшего и среднего профессионального образования, а также ряд услуг непрерывного профессионального образования.

В зависимости от подготовки абитуриента доступны различные пути поступления:

• с аттестатом об общем среднем образовании можно поступить либо на программу среднего профессионального образования (колледжа), либо на программу бакалавриата;

• для тех, кто уже имеет степень среднего профессионального или высшего образования, существуют ускоренные (краткосрочные) варианты обучения по индивидуальной программе;

• программы очного бакалавриата длятся четыре года; Программы неполного или смешанного (вечернего/выходного дня) обучения обычно длятся 4,5 года, а сокращенные программы обучения можно пройти всего за три года.

Помимо программ получения степени, институт предлагает:

• подготовительные курсы для студентов, готовящихся к Единому государственному экзамену (ЕГЭ) и Основному государственному экзамену (ОГЭ);

• курсы повышения квалификации и переподготовки;

• возможности участия в исследовательских проектах, конкурсах и грантовых инициативах.

Научно-исследовательская деятельность института охватывает публикацию научных статей, монографий и сборников; руководство исследованиями бакалавриата и магистратуры; организацию или участие в научных мероприятиях, таких как конференции, форумы и круглые столы. Этот широкий набор академических функций — от формального обучения до научного сотрудничества — демонстрирует приверженность института объединению образования и исследований под одной институциональной крышей.

1.3 Функциональная структура учреждения

Организационная структура Читинского института охватывает как академические, так и административные подразделения. Каждый основной факультет или служба вносит свой вклад в общую образовательную миссию института. Ниже приведен схематический перечень основных структурных подразделений (рисунок 1):

- Финансово-экономический факультет
  - Кафедра информационных технологий и высшей математики
  - Кафедра финансов и менеджмента
  - Кафедра мировой экономики, предпринимательства и гуманитарных наук
- Юридический факультет
  - Кафедра гражданского и уголовного права и процесса
  - Кафедра теории, истории и государственно-правовых дисциплин
- Кафедра иностранных языков
- Кафедра физической культуры и спорта
- Колледж института (колледж Читинского института)
- Библиотека
- Административно-хозяйственная служба
- Отдел учебно-методического и информационного обеспечения (УМИО)
- Финансовый отдел
- Отдел приема и трудоустройства
- Служба кадров

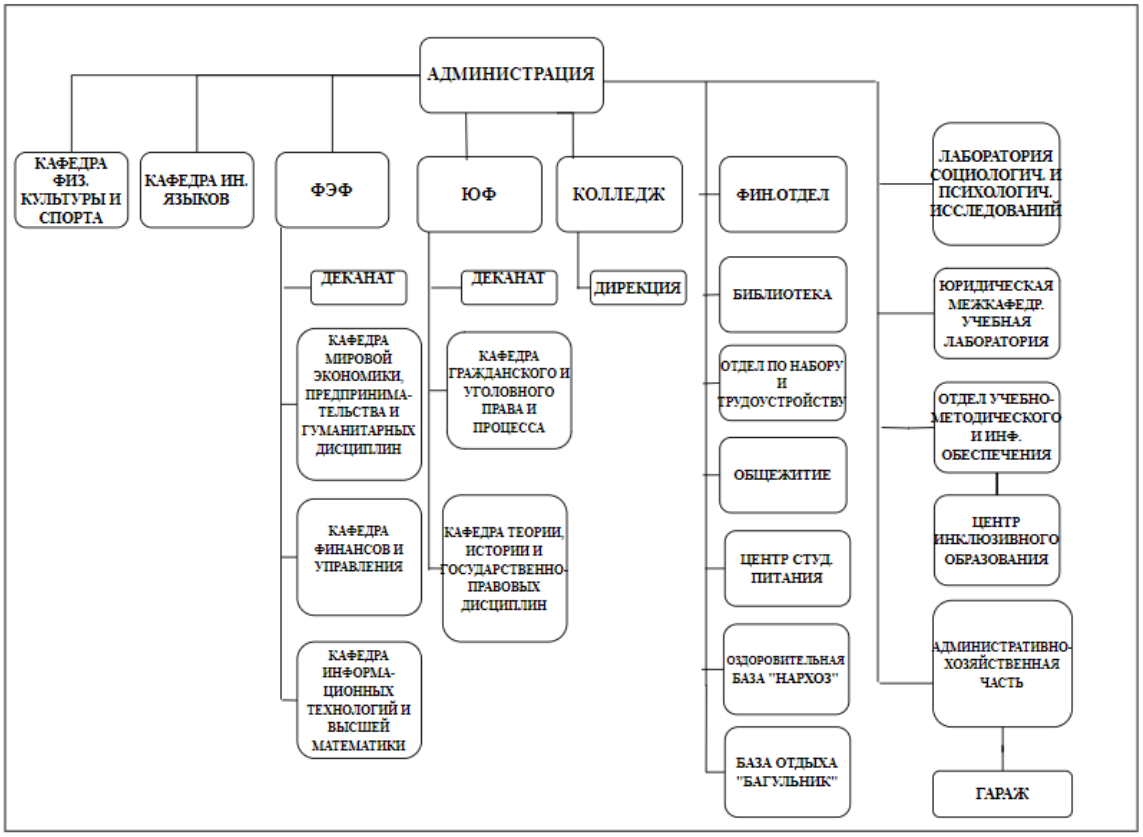


Рисунок 1 – Структура ЧИ ФГБОУ ВО «БГУ»

Как крупное учебное заведение, институт опирается на эту многоуровневая структура для координации преподавания, научной работы, администрирования и цифровой поддержки процесса обучения. Департамент учебно-методического и информационного обеспечения (УМИО) играет центральную роль в координации рабочих процессов: он взаимодействует с учебными подразделениями, деканатом, техническими службами и кадровыми ресурсами для подготовки расписаний, обслуживания цифровых систем, сбора отчетов и предоставления методических рекомендаций.

В настоящее время связь между этими различными подразделениями осуществляется по нескольким каналам:

- Внутренняя телефонная сеть
- Электронная почта

• Физический обмен документами (бумажные записки, USB-накопители и т. д.)

• Неофициальные приложения для обмена сообщениями (например, Telegram или ВКонтакте)

• Устаревшая локальная система обмена сообщениями под названием Net Speakerphone, децентрализованное решение, установленное на некоторых рабочих станциях

Хотя Net Speakerphone все еще используется ограниченно, он страдает от множества недостатков: отсутствие централизованного администрирования, уязвимости безопасности, ограниченные групповые функции и зависимость от активности определенных устройств. В среде, которая все больше требует гибридного обучения и удаленного сотрудничества, эти устаревшие каналы больше не соответствуют институциональным стандартам безопасности, удобства и простоты доступа.

Из-за этих ограничений отдел УМИО определил, что безопасная веб-платформа обмена сообщениями, тесно интегрированная в цифровую инфраструктуру института, будет лучше отвечать потребностям преподавателей, сотрудников и студентов. Этот анализ существующей организационной структуры коммуникаций закладывает основу для разработки решения, которое учитывает как логистические, так и требования безопасности.

1.2. Автоматизация процессов внутренней коммуникации

Современные учебные заведения требуют быстрой и надежной внутренней коммуникации. Однако на практике обмен информацией между структурными подразделениями часто фрагментирован. Преподаватели и сотрудники используют электронную почту, личные мессенджеры и телефонные звонки для передачи важных сообщений или координации действий. Единого унифицированного

инструмента не существует, поэтому каждое подразделение использует разные методы, что приводит к неэффективности и потере данных.

Устаревшая система Net Speakerphone давно изжила себя. Хотя современные приложения для обмена сообщениями, такие как Telegram или Вконтакте, кажутся удобными, они не подходят для официального институционального использования, поскольку им не хватает централизованного управления, регистрации данных и безопасного хранения. Что особенно важно, они не интегрируются с институциональными ресурсами, такими как расписания занятий, электронные журналы или базы данных преподавателей и студентов, что делает их изолированными хранилищами, а не связанными инструментами.

Исходя из этого, стало ясно, что необходим специальный защищенный веб-мессенджер. Такое приложение должно централизовать коммуникацию, позволяя пользователям обмениваться сообщениями в режиме реального времени, формировать как публичные, так и частные группы и сохранять историю чатов на защищенном сервере. Основные цели автоматизации включают:

- Сокращение нагрузки на персонал, связанной с передачей и координацией информации
- Повышение точности и скорости операционных процессов
- Предотвращение потери данных за счет централизованного хранения сообщений
- Улучшение пользовательского опыта внутренних коммуникаций

Разработав безопасный мессенджер, институт может заменить разрозненные каналы одним цифровым инструментом, который легко интегрируется с существующими рабочими процессами обеспечивая обмен документами, координацию встреч и объявления на уровне всего учреждения. Так возникает

уверенность, что конфиденциальная информация остается защищенной в собственной ИТ-среде университета.

1.3 Бизнес-процессы веб-приложения

В рамках существующего рабочего процесса сообщения и запросы отправляются по электронной почте, через сторонние мессенджеры или по телефону. Не существует единой централизованной системы отслеживания запросов: разговоры часто охватывают несколько каналов, не следуют общему формату и не имеют постоянной записи. Когда кому-то нужно вернуться к старому сообщению, он должен искать его в различных почтовых ящиках; Кто и когда ответил, может быть не сразу понятно.

Эти пробелы приводят к потере информации, дублированию усилий и неопределенности в отношении владения задачами. Разные отделы следуют разным практикам, поэтому нет единого «места для проверки статуса запроса» или быстрого обращения в ответственный офис (рисунок 2).

Рисунок 1.1. Пример использования: интеграция с системой тикетов техподдержки

Настраиваемый безопасный веб-мессенджер решает эти болевые точки, объединяя общение на одной платформе. Каждый пользователь выбирает соответствующую чат-комнату, например «Отдел кадров», «Деканат» или «УМИО», и отправляет сообщение, при необходимости прикрепляя файлы или отмечая темы. Это сохраняет контекст разговора, предотвращает дублирование задач и обеспечивает прозрачную видимость текущих запросов.

Преимущества такого подхода очевидны:

- Устранение необходимости ручной передачи данных и передачи информации по нескольким каналам
- Более быстрое время ответа на запросы

• Последовательная история бесед, которую можно использовать для отчетности или разрешения споров

Более того, бизнес-процессы мессенджера могут быть расширены с течением времени. Например, он может поддерживать теги по типу запроса, групповые чаты для межведомственных обсуждений или автоматическую маршрутизацию тикетов, когда сообщения пересылаются в определенные отделы на основе ключевых слов или категорий.

Таким образом, веб-приложение не просто заменяет старые каналы связи. Оно становится центральным, безопасным инструментом для оптимизированной, отслеживаемой и хорошо организованной связи между всеми подразделениями института.

1.4 Обзор существующих решений

Сегодня мир корпоративных коммуникаций в значительной степени зависит от цифровых платформ. В частном секторе такие решения, как Slack, Microsoft Teams и Zoom Chat, доминируют в повседневных рабочих процессах для обмена файлами, видеоконференций и обмена мгновенными сообщениями. Однако в контексте государственного сектора, особенно в России, использование многих из этих иностранных сервисов может быть юридически ограничено, технически ненадежно или вообще недоступно.

Например, Slack фактически недоступен в России из-за санкций и правовых ограничений. Некоторые учреждения обратились к отечественным альтернативам, таким как VK WorkSpace (ранее Mail.ru Group), МойОфис или R7-Office. Хотя эти платформы предлагают некоторые преимущества, они также создают новые пробле

• Платный сервис; даже базовые функции требуют инвестиций, которые могут не оправдать бюджет института.

• Адаптация этих платформ к конкретным внутренним потребностям часто требует индивидуальной разработки или обширного обучения персонала, что приводит к дополнительным затратам времени и ресурсов.

Учитывая эти факторы, разработка специального веб-приложения, предназначенного для нужд института является логичным решением. Такое внутреннее решение позволяет избежать юридических и финансовых ограничений, обеспечивает полный контроль данных и может быть точно настроено для соответствия фактическим рабочим процессам учреждения, что обеспечивает как экономию средств, так и операционную автономию.

Параметры	Slack	VK WorkSpace	MyOffice	Собственное приложение
Доступность в РФ	Нет	Да	Да	Да
Отсутствие затрат	Да	Нет	-	Да
Безопасность по стандартам российских законов	Да	Да	Да	Да
Возможность интеграции в собственные сервисы	Да	Нет	Да	Да
Контроль над хранением данных	Нет	Нет	Да	Да
Соответствие требованиям госучреждений	Нет	Да	Да	Да

Таблица 1.1 – Сравнение существующих решений и собственного

1.5 Требования к защите данных для обмена сообщениями в государственном секторе

В среде быстро меняющихся цифровых технологий безопасность имеет первостепенное значение, особенно для систем, которые предполагают ежедневный обмен потенциально конфиденциальной информацией, такой как внутренние

служебные записки, расписания и персональные данные. Обеспечение конфиденциальности институциональных коммуникаций — это не просто передовая практика, предписание законом.

Основной мерой безопасности в системах обмена сообщениями является шифрование, цель которого — сделать сообщения непонятными для неавторизованных лиц. Даже если злоумышленник перехватит зашифрованные данные, он не сможет прочитать их содержимое без соответствующего ключа дешифрования.

Следует рассмотреть три основные парадигмы шифрования:

Симметричное шифрование использует один общий ключ как для шифрования, так и для дешифрования. Проблема заключается в безопасном распространении этого ключа среди всех участников, не раскрывая его злоумышленникам.

Асимметричное шифрование (криптография с открытым ключом) назначает каждому участнику пару ключей — один открытый, один закрытый. Отправитель шифрует сообщение с помощью открытого ключа получателя; только закрытый ключ получателя может его расшифровать. Это устраняет необходимость делиться секретным ключом напрямую.

Сквозное шифрование (E2EE) — это усовершенствованная схема, в которой данные шифруются на устройстве отправителя и расшифровываются только на устройстве получателя. Промежуточные серверы, включая сам сервер приложений, никогда не имеют доступа к открытому тексту. Такие решения, как Signal и WhatsApp, полагаются на E2EE как на «золотой стандарт» для безопасного обмена сообщениями.

Для индивидуального внутриорганизационного мессенджера эти методы формируют ядро стратегии защиты данных. Однако шифрование — это лишь один аспект более широкой архитектуры безопасности, которая также должна включать:

Безопасное хранение криптографических ключей со строгим контролем доступа

TLS/SSL для защиты коммуникаций на транспортном уровне между клиентами и сервером

Механизмы аутентификации пользователей, предпочтительно многофакторная аутентификация

Комплексное ведение журнала событий и оповещения об аномальных действиях

Управление доступом на основе ролей для ограничения доступа пользователей к определенным данным или функциям

Помимо технического контроля нельзя упускать из виду человеческий фактор: самая сильная криптографическая система уязвима, если конечные пользователи выбирают легко угадываемые пароли или не соблюдают основные правила безопасности.

Поэтому создание безопасной системы обмена сообщениями для государственного образовательного учреждения требует целостного, многоуровневого подхода — сочетания шифрования, использования безопасного протокола, строгой аутентификации и постоянного обучения по повышению осведомленности. Это гарантирует, что внутренние коммуникации остаются защищенными как от внешних угроз, так и от случайных внутренних ошибок.

Рисунок 1.2. Рабочий процесс асимметричного шифрования (отправитель шифрует с помощью открытого ключа получателя; получатель расшифровывает с помощью своего закрытого ключа)

1.6 Требования к приложению

Перед началом разработки важно провести системный анализ будущего приложения. Он позволяет не только сформулировать технические требования, но и понять, каким должен быть продукт с точки зрения конечных пользователей. Речь идет не просто о наборе функций, а о понимании контекста, в котором это приложение будет использоваться, и о тех задачах, которые оно должно решать.

Целевые пользователи: Приложение ориентировано на сотрудников и учеников института, в том числе преподавателей, методистов, работников административных и вспомогательных отделов. Это люди с разным уровнем технической подготовки, поэтому пользовательский интерфейс должен быть

интуитивно понятным, а доступ к основным функциям — максимально простым. Особое внимание при проектировании уделяется удобству переписки, стабильной работе в локальной сети и быстрому отклику.

Функциональные и нефункциональные требования:

Функциональные требования описывают, что именно должно уметь приложение. Основные функции:

- Зарегистрированные пользователи входят в систему с помощью безопасного пароля; в будущих версиях будет реализована многофакторная аутентификация.
- Система должна поддерживать дифференциацию ролей (например, обычный пользователь, модератор, администратор), что позволяет контролировать доступ к определенным чатам или административным функциям.
- Пользователи могут просматривать список доступных чатов (публичные форумы, каналы отделов или частные группы).
- Пользователи могут создавать новые чаты и указывать, являются ли они «частными» (только по приглашению) или «публичными» (открытыми для любого аутентифицированного пользователя).
- Создатель чата становится первоначальным модератором комнаты и может приглашать или удалять участников.
- Система отображает присутствие пользователя в режиме реального времени (например, индикаторы статуса: онлайн, офлайн, занят).
- В любой чат-комнате участники обмениваются текстовыми сообщениями в реальном времени.
- Система группирует последовательные сообщения от одного отправителя, чтобы избежать повторных именных меток, тем самым улучшая читаемость (если пользователь А отправляет три сообщения подряд, его имя появляется один раз с блоком временной метки).

• Каждое сообщение имеет временную метку на стороне клиента и сервера, чтобы обеспечить хронологический порядок.

• Пользователи могут прикреплять файлы (например, документы PDF, файлы Word или изображения) к сообщениям.

• Вложения загружаются на сервер и хранятся в защищенном каталоге со ссылкой (URL или путь), сохраненной вместе с сообщением в базе данных.

• Получатели могут нажимать на вложения, чтобы загрузить или просмотреть их в браузере.

• Панель поиска над списком чатов фильтрует комнаты на основе ключевых слов в их заголовках или описаниях.

• Если пользователь отключается (из-за сбоя сети или перезапуска сервера), все отправленные в это время сообщения помещаются в очередь на сервере.

• При повторном подключении клиент извлекает все пропущенные сообщения для синхронизации истории чата.

• Администраторы могут удалять неподходящие сообщения, временно отключать или блокировать пользователей и создавать «системные» сообщения (например, уведомления о запланированном обслуживании).

• Модель данных и поток сообщений должны предусматривать гибридное шифрование (т. е. обмен ключами RSA/AES).

• Каждая запись сообщения включает поля для хранения зашифрованного ключа AES (если реализовано шифрование) и вектора инициализации (IV).

• Открытые ключи для каждого пользователя хранятся в таблице «пользователи», готовые к гибридному обмену ключами при первом контакте.

Помимо этого, важны нефункциональные характеристики:

• Весь HTTP-трафик между клиентом и сервером должен обслуживаться по HTTPS (TLS/SSL).

• Пароли пользователей хешируются и солятся (например, с помощью bcrypt) перед сохранением в базе данных.

• Сервер должен очищать все входные данные для предотвращения SQL-инъекций, межсайтового скриптинга (XSS) и других распространенных веб-уязвимостей.

• Проектирование программного обеспечения должно допускать горизонтальное масштабирование (например, с помощью нескольких экземпляров Node.js за балансировщиком нагрузки) для поддержки до 1000 одновременных активных пользователей.

• Socket.IO-комнаты должны иметь возможность охватывать несколько процессов или серверов с помощью общего адаптера.

• Система должна плавно обрабатывать временные сбои сети: если клиент отключается, сервер удерживает неотправленные сообщения и доставляет их при повторном подключении клиента.

• Любые критические транзакции базы данных (например, вставки сообщений) должны быть заключены в надлежащую обработку ошибок с логикой повтора, если необходимо.

• Процесс сервера настроен на автоматический перезапуск (например, через PM2 или аналогичный менеджер процессов) в случае сбоя.

• Задержка сообщения (время между отправкой сообщения на одном клиенте и его получением на другом) должна оставаться менее 200 мс в типичной среде локальной сети.

• Первоначальная загрузка списка чатов и первое извлечение до 100 последних сообщений должны происходить менее чем за 300 мс.

• Интерфейс React должен поддерживать не менее 60 кадров в секунду в современных браузерах при отображении до 200 сообщений в прокручиваемом контейнере.

• Интерфейс соответствует лучшим практикам для удобства чтения (разумные размеры шрифтов, достаточный цветовой контраст).

• Навигация интуитивно понятна: пользователь может переключаться между чатами одним щелчком или касанием.

• Предусмотрены основные сочетания клавиш (например, «Enter» для отправки сообщения, «Esc» для очистки фокуса ввода).

• Дизайн должен быть адаптивным, изящно подстраиваясь под ширину экрана от 320 пикселей (мобильный) до 1920 пикселей (настольный).

• Код организован в логические модули: компоненты React хранятся в `/src/components`, серверные маршруты в `/routes`, а бизнес-логика в `/controllers`.

• Каждый модуль сопровождается фрагментом README, объясняющим его назначение.

• Соблюдаются четкие соглашения о кодировании (например, правила ESLint, форматирование Prettier).

1.7. Итог анализа

Проведя тщательный анализ структуры организации, существующих рабочих процессов связи и соответствующих требований безопасности, было установлено четкое обоснование для проектирования и создания настраиваемого, безопасного веб-мессенджера. В следующих главах будут подробно описаны архитектурный проект, реализация и тестирование предлагаемой системы.

2. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Моделирование программного обеспечения — это процесс создания абстрактных представлений системы, которые помогают понять её структуру, функциональность и поведение в разных условиях. Моделирование позволяет на ранних этапах разработки выявить проблемы, уточнить требования и грамотно организовать архитектуру приложения. В данном разделе будет рассмотрено моделирование системы с помощью UML, а также описана структура базы данных, которая лежит в основе системы.

2.1 UML (Unified Modeling Language)

UML (Унифицированный Язык Моделирования) является стандартом для визуализации, спецификации, конструирования и документирования элементов программных систем. В рамках нашего проекта был использован UML для создания диаграмм, которые помогают детально представить функциональность системы, а также связи между её элементами.

UML позволяет эффективно описывать:

- Структуру системы — классы, объекты, атрибуты, связи между ними;
- Поведение системы — взаимодействие пользователей с системой, различные сценарии и последовательности действий;
- Архитектуру системы — как различные компоненты и модули взаимодействуют друг с другом.

Моделирование с использованием UML значительно упрощает понимание того, как будет функционировать система и какие компоненты в ней будут взаимодействовать. Оно также помогает разработчикам, тестировщикам и другим членам команды работать с единой картиной, не теряя из виду важные аспекты реализации [3].

В рамках данной разработки были использованы несколько типов диаграмм UML для отображения различных аспектов системы:

• Диаграмма вариантов использования (Use Case Diagram) — описывает функциональные требования к системе с точки зрения пользователей. Она показывает, как пользователи взаимодействуют с системой, какие функции они могут выполнять и какие роли управляют доступом к этим функциям (Рисунок 2.1);

• Диаграмма последовательности (Sequence Diagram) — показывает, как объекты взаимодействуют между собой в рамках конкретного сценария. Она демонстрирует порядок сообщений и действий, которые происходят при выполнении операции;

• Диаграмма классов (Class Diagram) — определяет структуру системы, описывая классы, их атрибуты, методы и отношения между ними. Эта диаграмма помогает организовать код, обеспечивая ясность и логичность структуры программы.

В дополнение к UML, для описания структуры базы данных используется специальный формат DBML (Database Markup Language). Этот формат помогает наглядно представить таблицы, их поля и связи, что важно для понимания, как данные будут храниться и взаимодействовать в системе.

2.2 Диаграмма вариантов использования

Диаграмма вариантов использования отображает функции, доступные различным пользователям системы, а также их взаимодействие с системой. В рамках проекта были выделены следующие основные роли пользователей:

• Обычный пользователь — имеет доступ к чатам, может отправлять и получать сообщения, вступать в публичные и приватные комнаты;

• Администратор — имеет расширенные права, включая управление участниками и доступом.

Каждый из этих участников системы может выполнять ряд операций, таких как авторизация, создание чатов, отправка сообщений и другие. Диаграмма

вариантов использования позволяет легко увидеть, какие функции доступны для каждого типа пользователя.

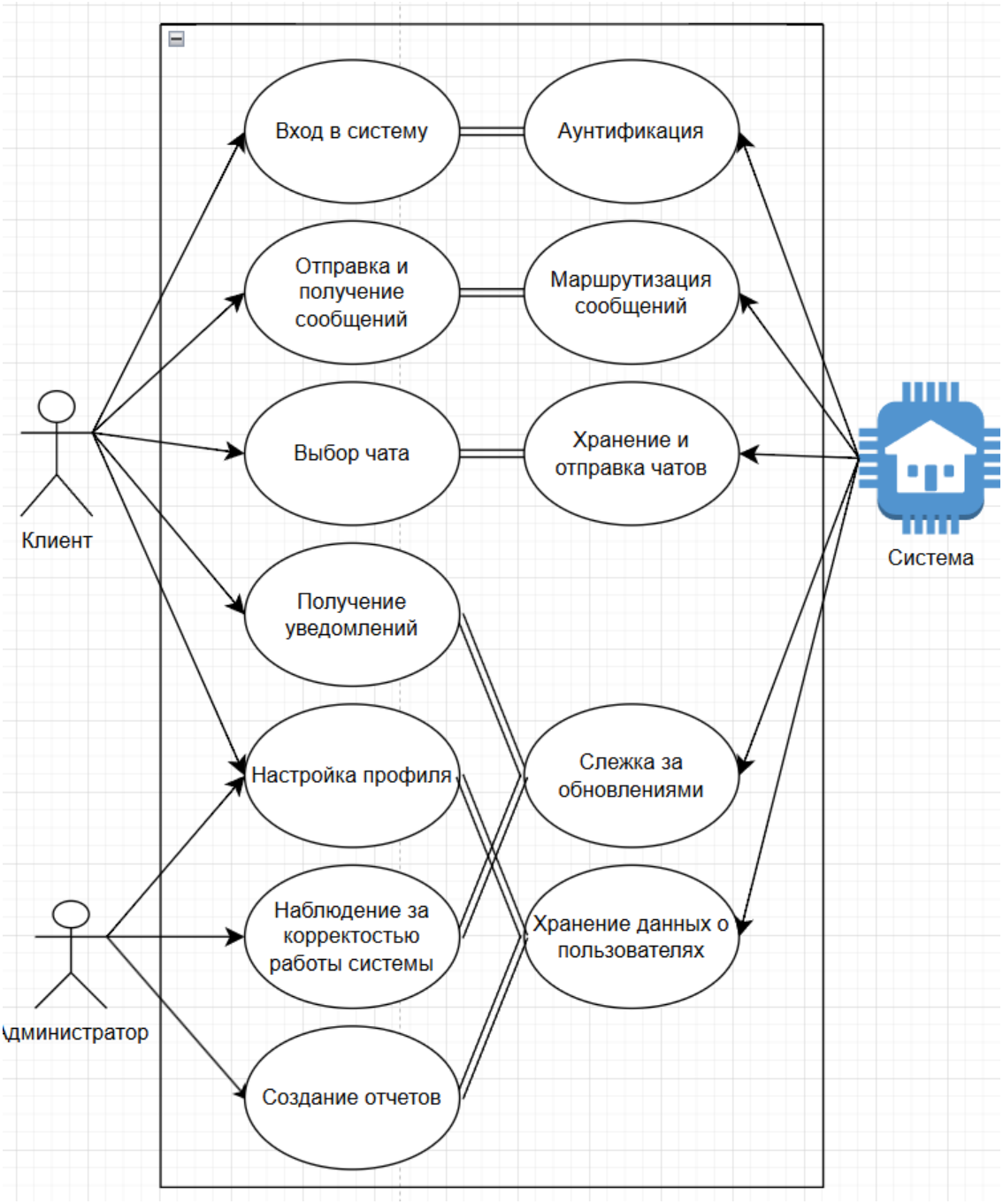


Рисунок 2.1 – Диаграмма вариантов использования

Эта диаграмма важна тем, что она позволяет четко выделить основные функции системы и роли пользователей, взаимодействующих с этими функциями. Она помогает не только в проектировании, но и в документировании системы.

2.3 Структура базы данных

Важным этапом в проектировании системы является создание структуры базы данных, которая будет хранить всю информацию, необходимую для функционирования мессенджера. Система должна поддерживать хранение данных о пользователях, сообщениях, комнатах и их участниках. Для этого были созданы четыре основные таблицы:

- пользователи — содержит информацию о пользователях системы (имя, логин, пароль, роль и т. д.);
- сообщения — хранит сообщения, отправленные в чатах, с привязкой к пользователю и комнате;
- комнаты — описывает чаты, их название, описание и настройки (например, приватность, количество участников);
- члены комнат — устанавливает связь между пользователями и комнатами, в которых они состоят.

Для визуализации структуры базы данных использован формат DBML. Он позволяет наглядно представить таблицы и связи между ними. Важно, что база данных реализует несколько ключевых ограничений и связей, включая внешние ключи для обеспечения целостности данных (Рисунок 2.2.).

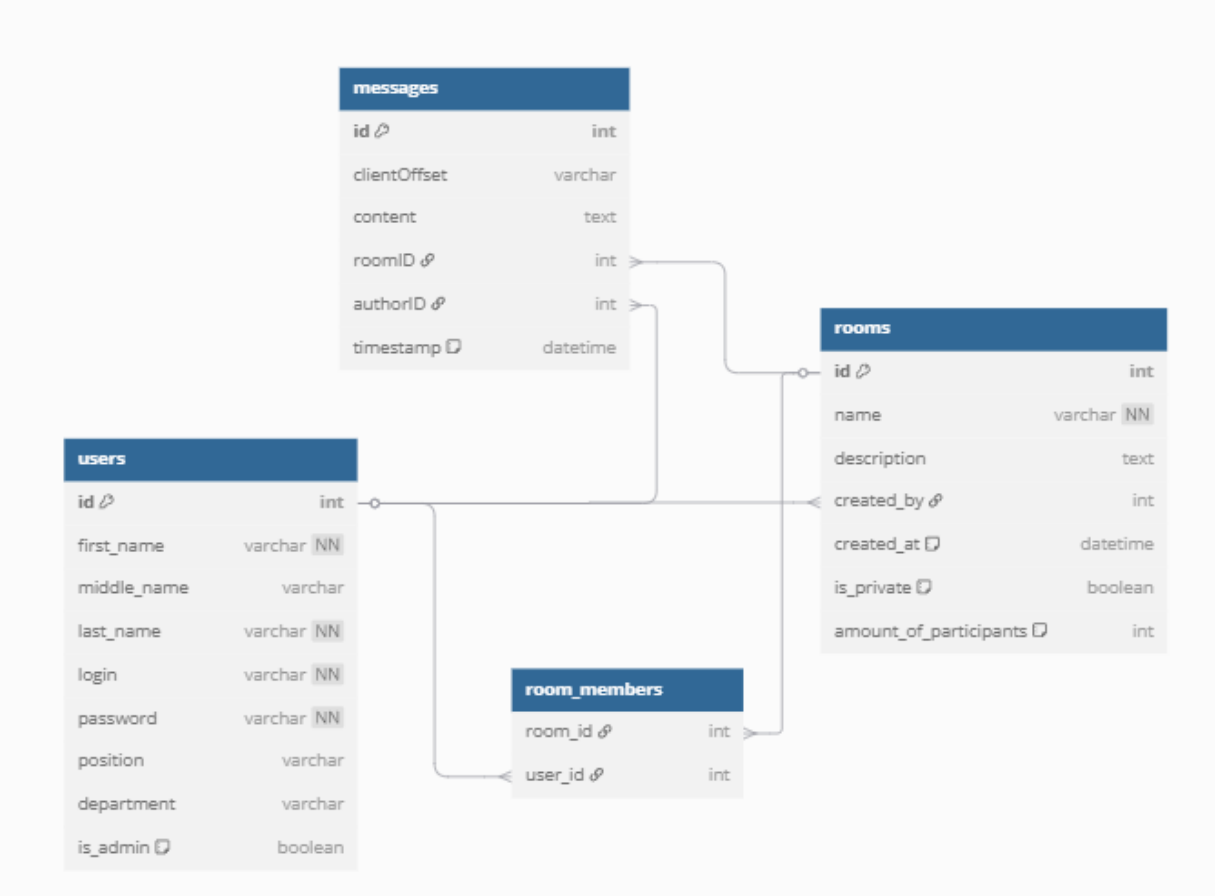


Рисунок 2.2. – Схема, созданная с помощью разметки DBML

Эта схема позволяет четко увидеть, как данные будут храниться и связаны между собой. Она является основой для создания эффективной и масштабируемой системы хранения данных [4].

2.4 Дизайн интерфейса

Пользовательский интерфейс (UI) играет решающую роль в том, как воспринимается и используется программное обеспечение. Даже при наличии надежной функциональности запутанный или слишком сложный интерфейс может разочаровать пользователей и снизить производительность. Для этого приложения для обмена сообщениями особое внимание уделялось ясности, логическому потоку и визуальной согласованности.

Вместо того, чтобы сразу переходить к коду, процесс проектирования начался в Figma. Серия низкоточных каркасов отображала основные экраны: панель списка чатов, активное окно чата (Рисунок 2.3), область ввода сообщений и меню профиля. Благодаря функциям совместной работы Figma заинтересованные стороны, включая представителей различных отделов, просматривали эти каркасы, предоставляли отзывы о макете и предлагали улучшения в размещении меню и стилях кнопок. После того, как каркасы соответствовали критериям удобства использования, были созданы высокоточные макеты, определяющие точную типографику, цветовые палитры и иконографию, которые соответствовали рекомендациям по брендингу института.

Ключевые цели пользовательского интерфейса включали поддержку нескольких разговоров (как публичных групповых чатов, так и личных чатов один на один), легкую навигацию между чатами, простую форму ввода сообщений и быстрый доступ к функциям поиска и приглашения пользователей. Каждый компонент, такой как карточка предварительного просмотра чата, пузырек сообщения и поле поиска, был разработан в Figma как повторно используемый компонент с вариантами (например, выбранные и невыбранные состояния чата, стили входящих и исходящих сообщений).

Окончательный прототип в Figma также продемонстрировал адаптивное поведение: на больших экранах настольных компьютеров список чатов отображался в фиксированном левом столбце, в то время как основная история чата занимала правую сторону. Когда область просмотра сужалась (например, на планшете или небольшом ноутбуке), список чатов можно было свернуть в меню-гамбургер, что позволяло истории чата расширяться для более удобного чтения. Интерактивные прототипы Figma имитировали нажатие на чат для открытия его разговора, ввод сообщения и развертывание меню профиля пользователя в правом верхнем углу. Эта интерактивность помогла выявить потенциальные проблемы с удобством использования, такие как недостаточное расстояние вокруг сенсорных целей на небольших экранах, и провести корректировки до написания кода.

Рисунок 2.3 – Прототип приложения в Figma

Начиная с прототипа Figma и итерируя отзывы заинтересованных сторон, процесс проектирования обеспечивал как удобство использования, так и

визуальную гармонию еще до того, как была написана хотя бы одна строка кода. Эта основа сделала последующую реализацию React более эффективной, что привело к отточенному интерфейсу, который соответствует ожиданиям пользователей на экранах разных размеров.

3. РЕАЛИЗАЦИЯ КЛИЕНТСКОЙ ЧАСТИ ПРИЛОЖЕНИЯ

Разработка клиентской части велась с использованием современного стека веб-технологий, обеспечивающего отзывчивый интерфейс и быструю передачу данных между пользователями в реальном времени [5].

3.1. Технологии, используемые на front-end

Клиентская часть приложения разработана с использованием современных веб-технологий, ориентированных на создание интерактивного интерфейса и обеспечение стабильной работы в реальном времени. Основной стек включает в себя React.js, Tailwind CSS, Socket.IO и вспомогательные библиотеки.

React.js выбран в качестве основного инструмента для построения пользовательского интерфейса. Он позволяет разделять приложение на независимые логические блоки, называемые компонентами. Это облегчает поддержку и развитие проекта [1]. Компоненты в нашем приложении представляют собой как отдельные элементы интерфейса (например, кнопка или поле ввода), так и более сложные, составные блоки, включающие в себя другие компоненты (например, список сообщений в чате).

React Router используется для реализации навигации внутри одностраничного приложения. Благодаря ему обеспечивается переключение между представлениями (например, страницей входа и основным интерфейсом приложения) без перезагрузки страницы. Это важно для пользовательского опыта, особенно в приложениях с постоянным соединением по WebSocket.

Tailwind CSS выбран в качестве инструмента для стилизации. Этот утилитарный фреймворк позволяет быстро разрабатывать адаптивный и визуально чистый интерфейс за счёт использования готовых классов. При этом сохраняется контроль над внешним видом каждого элемента, без необходимости писать большое количество пользовательских CSS-файлов. Tailwind особенно удобен в сочетании с

компонентным подходом React, так как стилизация осуществляется непосредственно внутри компонентов.

React Icons — это библиотека, которая предоставляет доступ к популярным наборам иконок прямо в виде компонентов. Иконки используются для оформления кнопок, отображения статуса и других визуальных элементов, что помогает сделать интерфейс более понятным и современным.

Контекст React (Context API) позволяет централизованно управлять состоянием приложения. В нашем случае через контекст передаются такие данные, как текущий пользователь, активный чат, полученные сообщения и подключение к серверу. Это позволяет избежать передачи пропсов через несколько уровней вложенности и делает код чище.

Хуки useState и useEffect применяются для управления состоянием и жизненным циклом компонентов. Например, с помощью useEffect можно подписаться на события от сервера при загрузке компонента и отписаться от них при его удалении. Такой подход необходим при работе с постоянным соединением через WebSocket.

Socket.IO Client обеспечивает постоянное соединение между браузером пользователя и сервером. Это соединение используется для мгновенной передачи сообщений, системных событий (например, "пользователь присоединился к чату") и получения уведомлений в режиме реального времени [2]. Клиентская часть активно использует этот механизм для отправки сообщений и обработки входящих событий.

Одним из важных архитектурных решений стало активное использование собственных блочных компонентов. Благодаря компонентной структуре React большая часть логики реализована в виде переиспользуемых блоков, что позволяет при необходимости расширить интерфейс без значительных переделок. Так, список комнат (RoomsList) формируется из повторяющихся элементов RoomPreview, а

каждый компонент сообщения (MessageBubble) используется в списке независимо от содержания — текст, изображение или системное уведомление.

3.2. Архитектура пользовательского интерфейса

Пользовательский интерфейс приложения организован по блочному принципу и делится на две основные области: блок комнат и блок текущего чата. Такой подход обеспечивает удобство навигации, простоту взаимодействия и понятное разделение логики (Рисунок 3.1).

Блок комнат расположен в левой части экрана. Он включает в себя три компонента:

- RoomsHeader содержит логотип, название приложения и при необходимости кнопку выхода из аккаунта;
- RoomsSearch реализует поиск по доступным комнатам. При вводе текста фильтруется список ниже;
- RoomsList отображает список всех чатов, к которым пользователь имеет доступ. Каждый чат представлен компонентом RoomPreview, который содержит имя комнаты и, при наличии, последнее сообщение.

Блок текущей комнаты занимает правую часть интерфейса и также состоит из трёх компонентов:

- RoomHeader показывает название активной комнаты и аватар, а также может содержать дополнительные кнопки управления (например, просмотр участников);
- MessageEventsList — это основной компонент, в котором отображаются все события в чате: сообщения пользователей, системные уведомления и возможные вложения. Он автоматически прокручивается вниз при получении нового сообщения;
- MessageForm предоставляет поле для ввода текста, кнопку отправки и при необходимости возможность прикрепления файлов.

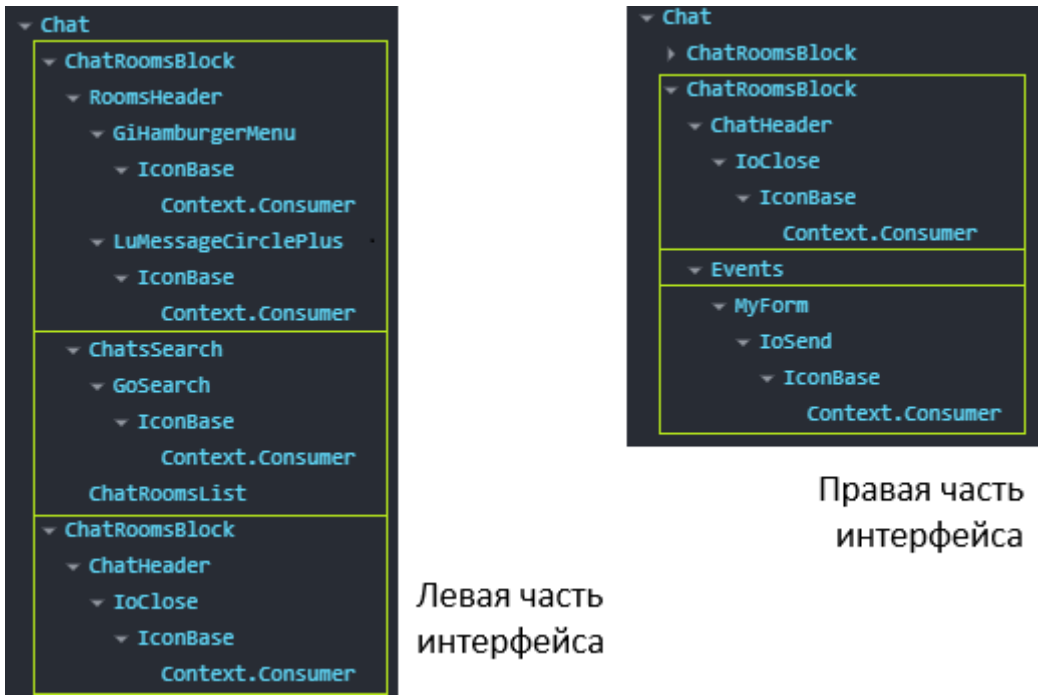


Рисунок 3.1 – Два блока итогового React DOM: слева RoomsHeader, RoomsSearch, RoomsList, а справа — RoomHeader, MessageEvents, MessageForm соответственно

Такое деление позволяет изолировать логику каждого блока, упростить тестирование и облегчить сопровождение. Каждый из компонентов работает независимо и получает необходимые данные через пропсы или из контекста, что снижает связанность между частями интерфейса.

Для поддержки визуальной целостности используются общие стили, предоставляемые Tailwind CSS. Благодаря этому интерфейс выглядит однородным, а взаимодействие между элементами происходит плавно.

Важно отметить, что компоненты MessageBubble, SystemNotice, RoomPreview и другие визуальные элементы реализованы как самостоятельные модули (Рисунок 3.2), что облегчает их переиспользование в разных частях приложения или расширение их функциональности в будущем.

```
<div className="flex w-full justify-start" key={ index }>
  <li
    className={`shadow-md  py-2 px-4 bg-white
      mx-4 mb-2  w-fit max-w-[75%] rounded-2xl`}
    key={ index }
  >
    //Проверка было ли уже сообщение от этого пользователя выше,
    //чтоб не дублировать имя
    {!isSameAuthorAsPrevious&&(
      <p className='text-sm text-blue-500 font-semibold'>{event.authorName}</
    )}
    //Текст самого сообщения
    <p>{ event.value }</p>
    //Обозначение времени
    <p className='text-right font-light text-sm text-gray-500'>
      {extractTime( event.timestamp)}
    </p>
  </li>
</div>
```

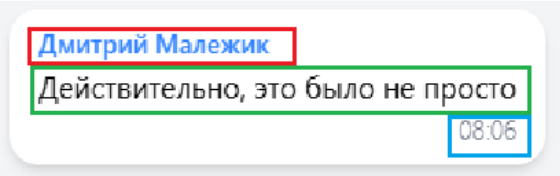


Рисунок 3.2 – Пример многоразового компонента MessageBubble

Кроме того, приложение адаптивно: в случае уменьшения ширины экрана скрывается блок комнат, и пользователь может переключаться между разделами вручную. Это особенно важно при использовании на планшетах или небольших ноутбуках.

3.3. Реализация интерфейса пользователя

Левая часть интерфейса приложения представляет собой блок с чатами. Она состоит из трёх основных компонентов: RoomsHeader, RoomsSearch и RoomsList. Эти элементы формируют зону навигации между чатами и обеспечивают быстрый доступ к нужной беседе.

Компонент RoomsHeader находится в верхней части и содержит логотип института, а также название приложения. Он не несёт функциональной нагрузки, но

создаёт визуальный акцент и помогает пользователю сориентироваться в пространстве приложения. При желании сюда можно добавить, например, переключатель темной темы.

Ниже располагается RoomsSearch — поле для поиска чатов. Его особенность в том, что оно активно взаимодействует с пользователем: кликом по всей области блока фокус устанавливается на поле ввода благодаря использованию useRef, а при появлении текста справа от поля отображается иконка-крестик, позволяющая быстро очистить ввод. Ввод отслеживается через useState, после чего локально фильтруется список комнат. Этот подход не требует постоянных запросов к серверу и обеспечивает мгновенный отклик. Отдельное внимание было уделено удобству: обводка при фокусе, скругления, адаптивные отступы делают использование блока интуитивно понятным.

Наконец, компонент RoomsList отображает перечень доступных чатов (Рисунок 3.3). Каждый элемент представляет собой карточку с названием и аватаром. При выборе комнаты её идентификатор сохраняется в контекст и используется другими компонентами. В процессе разработки появилось разделение chatId и chatIdInfo, что позволило параллельно обрабатывать данные для сокет-соединения и визуального отображения без пересечений и конфликтов. Это разделение оказалось полезным при масштабировании логики приложения.

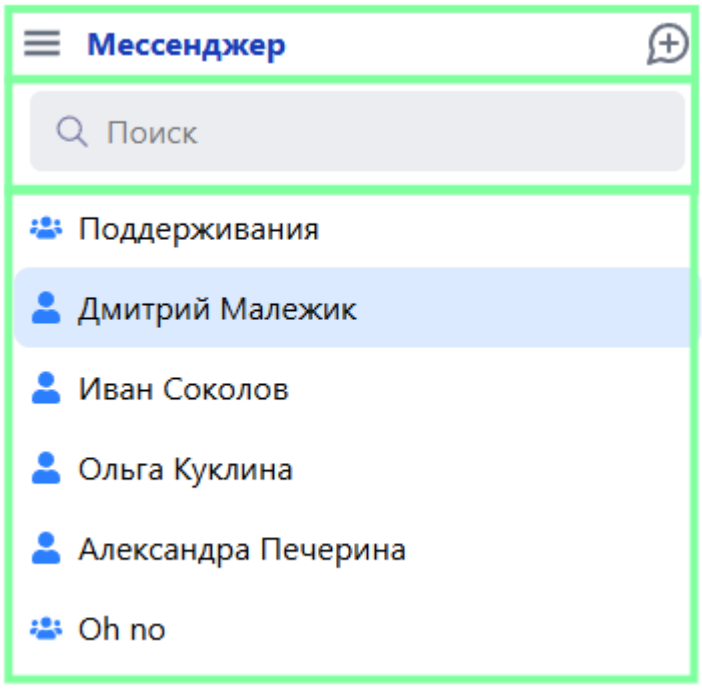


Рисунок 3.3 – Итоговый интерфейс: шапка, поле поиска и список комнат

Правая часть интерфейса — это окно текущей комнаты, где пользователь читает и отправляет сообщения (Рисунок 3.4). Здесь сосредоточены три ключевых компонента: RoomHeader, MessageEventsList и MessageForm.

Верхний блок, RoomHeader, отображает название выбранного чата, его аватар и основную информацию. Этот компонент служит ориентиром, позволяя пользователю быстро понять, с кем он общается. Кроме того, здесь можно разместить кнопки для управления комнатой или вызова дополнительного меню. Важной особенностью является динамическое обновление информации при переключении комнат, что достигается благодаря использованию React Context и отслеживанию текущего chatIdInfo.

Ниже расположен MessageEventsList — основной блок с сообщениями. Он рендерит список сообщений, сортированных по времени, с учётом статусов: отправлено, доставлено, прочитано. Для удобства восприятия реализована поддержка группировки сообщений одного отправителя и автоматическая прокрутка к последнему сообщению при добавлении нового. Компонент

эффективно обновляется с помощью ключей React и мемоизации, что улучшает производительность при большом количестве сообщений.

Форма отправки сообщений, MessageForm, позволяет пользователю ввести текст и отправить его. Здесь применён контролируемый input с валидацией: кнопка отправки активна только при наличии текста. При отправке сообщение передаётся через Socket.IO на сервер и сразу отображается в списке, что создаёт ощущение мгновенного отклика. Для удобства также реализована поддержка нажатия клавиши Enter для отправки.

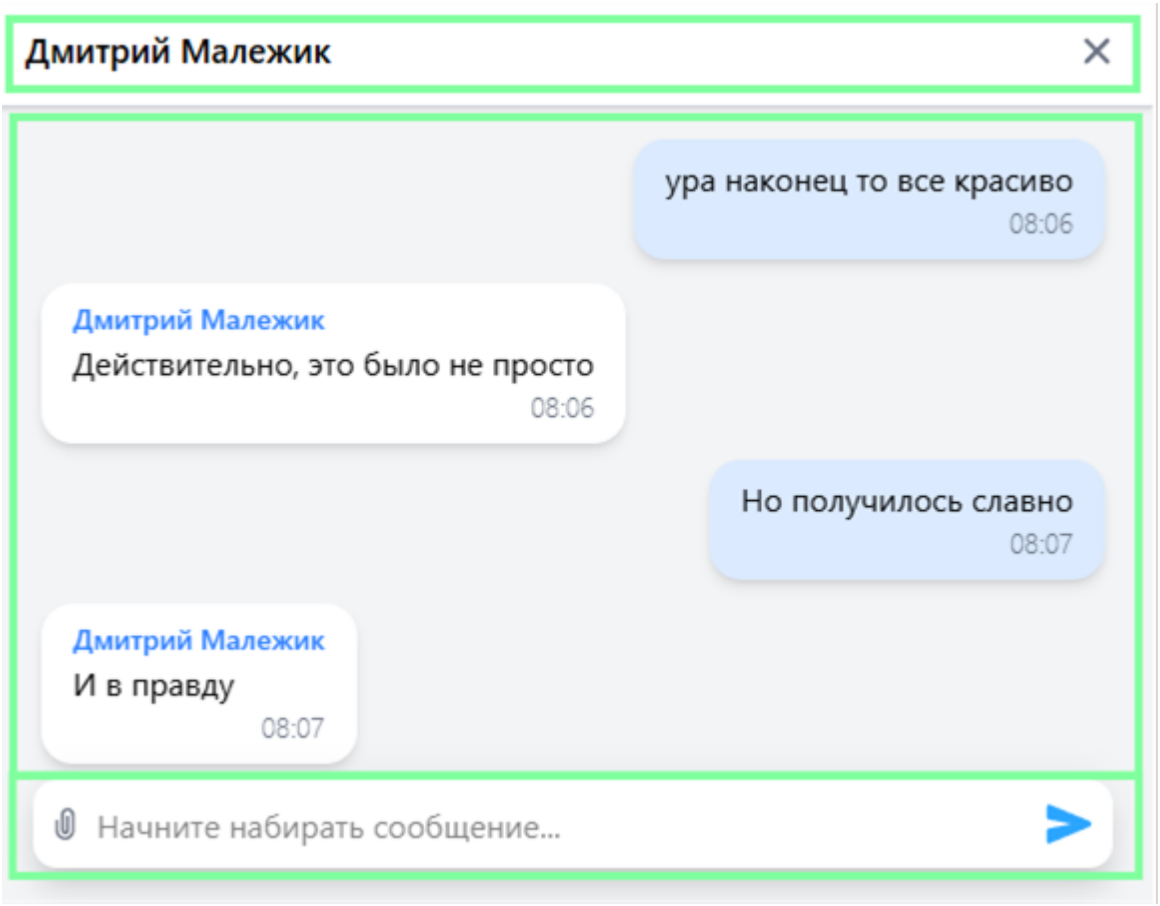


Рисунок 3.4 – Итоговый интерфейс: заголовок комнаты с названием, список сообщений в текущем чате, форма ввода сообщения

3.4. Работа с состоянием сокета клиента

В приложении для организации обмена сообщениями используется библиотека Socket.IO, которая обеспечивает постоянное двунаправленное

соединение между клиентом и сервером. Это позволяет передавать сообщения в реальном времени без необходимости обновлять страницу.

Для управления соединением создан компонент ConnectionManager. Он использует контекст SocketContext, в котором хранится состояние подключения. Компонент отображает текущий статус подключения и предоставляет кнопки для ручного подключения и отключения сокета. Это удобно для отладки и тестирования работы соединения.

Этот простой интерфейс (Рисунок 3.5) помогает визуально контролировать состояние подключения и обеспечивает удобный способ вручную управлять сокет-соединением при разработке.

```
export default function ConnectionManager() {
  //Декларируем статус
  const { isConnected } = useSocket();
  //Функции подключения\отключения
  function connect() {
    socket.connect();
  }
  function disconnect() {
    socket.disconnect();
  }
  //UI для контроля из приложения
  return (
    <div className='className="block w-full text-left px-2 py-2
      <p> {isConnected ? "Connected" : "Disconnected"} </p>
      <button onClick={ connect }>Connect</button>
      <button onClick={ disconnect }>Disconnect</button>
    </div>
  );
}
```

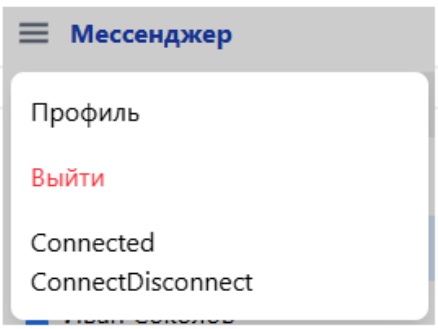


Рисунок 3.5 – Компонент управления соединением

В заключение разработки интерфейса следует отметить, что клиентская часть приложения была реализована в соответствии с современными стандартами веб-интерфейса: она не только обеспечивает полную навигацию между чатами и бесперебойный обмен сообщениями в реальном времени, но и остается масштабируемой благодаря модульной архитектуре компонентов React. Использование Context API и хуков позволяет централизовать состояние пользователя и соединения, гарантируя, что обновления данных будут синхронизироваться мгновенно, а логика обработки сокетов останется понятной и легко тестируемой. В то же время Tailwind CSS обеспечивает единообразный визуальный стиль без громоздких пользовательских стилей, а React Icons вместе с продуманными улучшениями UX, такими как динамические элементы, еще больше повышают удобство использования.

Этот уровень функциональности интерфейса создает прочную основу для следующих шагов: разработки и тестирования серверной части, а также интеграции механизмов шифрования и аутентификации. Плавный пользовательский интерфейс и стабильная производительность веб-интерфейса будут иметь ключевое значение для успешного продолжения работы системы и ее последующего развертывания в академической среде.

4. РЕАЛИЗАЦИЯ СЕРВЕРНОЙ ЧАСТИ ПРИЛОЖЕНИЯ

Серверная часть приложения отвечает за приём, обработку и передачу данных между клиентами, а также за хранение информации. Для её реализации использовался современный и надёжный стек технологий, который обеспечивает скорость, масштабируемость и простоту поддержки.

4.1. Технологии серверной части

Основные технологии, применённые в серверной части:

1.Node.js: серверная платформа на базе JavaScript, позволяющая запускать код вне браузера и создавать быстрые сетевые приложения.

2.Express.js: популярный фреймворк для Node.js, который упрощает создание REST API и маршрутизацию.

3.Socket.IO (server): библиотека для организации постоянного двунаправленного соединения между сервером и клиентом, реализующая WebSocket и запасные методы.

4.SQLite3: лёгкая встроенная база данных, используемая для хранения данных пользователей, сообщений и настроек.

5.Postman: инструмент для тестирования API, с помощью которого проверялись и отлаживались запросы к серверу (Рисунок 4.1).

Далее в главе будет подробно рассмотрена организация API, обработка сообщений через сокеты и работа с базой данных. Также будут приведены примеры запросов и ответов, полученных в Postman.

Рисунок 4.1 – Пример запроса в Postman для получения списка чатов

4.2. Структура серверной части

Серверное приложение организовано так, чтобы было легко работать и поддерживать код. В корне проекта расположен главный файл index.mjs, в котором настраивается сервер Express и подключается Socket.IO для обмена сообщениями в реальном времени.

В проекте есть файл routes, где описаны маршруты API для работы с чатами, сообщениями и пользователями. В отдельном файле controllers сосредоточена логика обработки этих запросов, а в файле models находятся функции для взаимодействия с базой данных SQLite. Также есть вспомогательные функции, например, для проверки данных, которые размещены в файле middlewears.

API сервера построено по REST-принципам. Например, чтобы получить список чатов, клиент отправляет запрос GET /chats. Для отправки нового сообщения используется запрос POST /messages. А чтобы получить сообщения конкретного чата, применяется GET /messages/:chatId.

Такое разделение кода позволяет быстро находить нужные участки, упрощает отладку и дальнейшее расширение функционала (Рисунок 4.2).

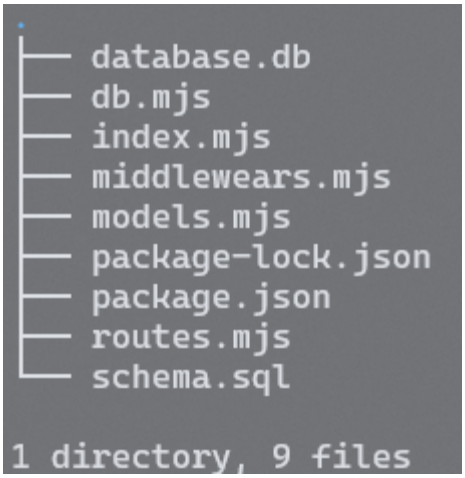


Рисунок 4.2 – Структура проекта в виде дерева

4.3. Работа с базой данных

Для хранения информации о пользователях, чатах, сообщениях и их взаимосвязях в проекте [4] используется реляционная база данных SQLite. В основе базы лежит несколько таблиц (Рисунок 4.3), каждая из которых выполняет свою функцию. Таблица users содержит данные о зарегистрированных пользователях — уникальные идентификаторы, имена и другие необходимые параметры. Таблица rooms хранит информацию о чатах, включая их уникальные идентификаторы и

названия, а также типы — например, групповые или приватные. Для связи пользователей с чатами служит таблица room_members, в которой фиксируются участники каждого конкретного чата. Наконец, таблица messages хранит сами сообщения: кто отправил, в какой чат, когда и текст сообщения.

```
import sqlite3 from "sqlite3";
import { open } from "sqlite";

export const db = await open({
    filename: "./database.db",
    driver: sqlite3.Database
});

await db.exec(`
    PRAGMA foreign_keys = ON;

    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        first_name TEXT NOT NULL,
        middle_name TEXT,
        last_name TEXT NOT NULL,
        login TEXT NOT NULL UNIQUE,
        password TEXT NOT NULL,
        position TEXT,
        department TEXT,
        is_admin INT DEFAULT 0
    );
```

Рисунок 4.3 – Пример схемы создания таблицы в db.mjs

Для работы с этими таблицами реализованы функции на серверной стороне, которые выполняют операции добавления, обновления и выборки данных [3]. Пи ри получении сообщений для определённого чата, например, отправляется SQL-запрос, который выбирает все сообщения из таблицы messages, соответствующие нужному идентификатору комнаты, и сортирует их по времени отправки.

Аналогично, при создании нового сообщения сервер формирует запрос на вставку данных в таблицу messages, указывая отправителя, время и содержимое сообщения.

Такая организация базы и набор функций обеспечивают быстрое и надёжное взаимодействие с данными (Рисунок 4.4), а также позволяют в дальнейшем легко расширять функционал приложения. При необходимости в приложении можно реализовать дополнительные запросы, например, для фильтрации пользователей по чатам или поиска сообщений по ключевым словам.

routes.mjs

src > routes.mjs > router.get("/users/:id/public-key") callback

```
78
79  router.post("/login", async (req, res) => {
80      const { login, password } = req.body;
81
82      if (!login || !password) {
83          return res.status(400).json({ error: "Login and password are required" });
84      }
85
86      try {
87          const result = await models.signInUser(login, password);
88          res.status(200).json(result);
89      } catch (error) {                    (property) error: any
90          res.status(401).json({ error: error.message });
91      }
92  });
93
```

models.mjs

models.mjs > getPublicKey

```
43  export async function signInUser(login, password) {
44      const user = await db.get("SELECT * FROM users WHERE login = ?", [login]);
45      if (!user) throw new Error("User not found");
46
47      const isPasswordValid = await bcrypt.compare(password, user.password);
48      if (!isPasswordValid) throw new Error("Invalid password");
49      // Generate JWT token
50      const token = jwt.sign(
51          { id: user.id, login: user.login, position: user.position },
52          JWT_SECRET,
53          { expiresIn: "1d" }
54      );
55
56      return {
57          user: {
58              id: user.id,
59              login: user.login,
60              firstName: user.first_name,
61              lastName: user.last_name,
62              middleName: user.middle_name,
63              position: user.position,
64              department: user.department
65          },
66          token
67      };
68  }
```

Рисунок 4.4 – Пример обработки авторизации пользователя по адресу /api/login

4.4 Обработка сокетов на стороне сервера

Серверная сторона приложения-мессенджера использует библиотеку Socket.IO для обеспечения двунаправленной связи в реальном времени между каждым подключенным клиентом (веб-браузером) и бэкэндом Node.js/Express. По своей сути Socket.IO представляет собой оболочку транспортного уровня WebSocket и плавно переходит к длительному запросу, если WebSocket недоступен, обеспечивая надежное соединение в различных сетевых условиях.

Во время запуска сервера Socket.IO подключается к существующему HTTP-серверу Express. Так io становится центральным экземпляром сервера Socket.IO. Конфигурация cors гарантирует, что только авторизованные источники, такие как собственный домен института, могут устанавливать соединения сокетов.

Как только клиент успешно проходит аутентификацию через REST API (например, POST /login возвращает действительный JWT), фронтенд запускает сокетное соединение.

На стороне сервера глобальное промежуточное ПО перехватывает каждое входящее сокетное соединение для проверки токена. Если проверка не проходит, соединение немедленно отклоняется (Рисунок 4.5).

```
import jwt from 'jsonwebtoken';

io.use((socket, next) => {
  const token = socket.handshake.auth.token;
  if (!token) {
    return next(new Error('Authentication required'));
  }
  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) {
      return next(new Error('Invalid token'));
    }
    socket.user = { id: decoded.userId, name: decoded.username };
    next();
  });
});
```

Рисунок 4.5 – промежуточное ПО

Если JWT действителен, промежуточное ПО устанавливает socket.user, позволяя всем последующим обработчикам событий ссылаться на socket.user.id или socket.user.name.

После аутентификации клиент немедленно отправляет событие joinRooms, передавая массив идентификаторов комнат, к которым принадлежит пользователь: «socket.emit('joinRooms', { Rooms: userRoomsArray });». Сервер прослушивает это событие и использует метод Socket.IO socket.join() для подписки сокета на эти комнаты (Рисунок 4.6).

```
io.on('connection', (socket) => {
    console.log(`Пользователь ${socket.user.name}
            подключился (Socket ID: ${socket.id})`);

    socket.on('joinRooms', ({ rooms }) => {
      rooms.forEach(roomId => {
        socket.join(`room_${roomId}`);
        console.log(`Сокет ${socket.id}
            присоеденился к комнате${roomId}`);
      });
    });

    socket.on('disconnect', (reason) => {
      console.log(`Пользователь ${socket.user.name}
            отключился: ${reason}`);

    // Сюда можно добавить остальные обработичики событий
  });
```

Рисунок 4.6 – Сервер прослушивает события

Использование пространства имен комнаты, например, room_123 для чата с идентификатором 123, обеспечивает простой способ трансляции сообщений только участникам. Когда пользователь отключается, любые структуры данных на стороне сервера, отслеживающие присутствие (например, карта активных пользователей в памяти для каждой комнаты), обновляются для удаления этого пользователя.

Когда пользователь составляет сообщение на клиенте, этот клиент отправляет событие sendMessage, включая идентификатор комнаты, содержимое сообщения и сгенерированный клиентом идентификатор (Рисунок 4.7) (например, временную метку или UUID) для дедупликации.

```
socket.emit('sendMessage', {
    roomId: 123,
    content: 'Всем здравствуйте!',
    tempId: 'msg_456'
});
```

Рисунок 4.7 – Клиент отправляет событие sendMessage

На сервере обработчик событий сначала сохраняет сообщение в базе данных, связывая его с правильной комнатой и пользователем. После того, как база данных подтвердит вставку и вернет первичный ключ нового сообщения и официальную временную метку сервера, сервер повторно отправляет событие newMessage в эту конкретную комнату (Рисунок 4.8).

```
socket.on('sendMessage', async ({ roomId, content, tempId }) => {
    try {
      const savedMessage = await saveMessageToDB({
        roomId,
        authorId: socket.user.id,
        content,
        timestamp: new Date().toISOString()
      });

      const payload = {
        messageId: savedMessage.id,
        roomId: savedMessage.roomId,
        authorId: socket.user.id,
        authorName: socket.user.name,
        content: savedMessage.content,
        timestamp: savedMessage.timestamp,
        tempId
      };
      io.to(`room_${roomId}`).emit('newMessage', payload);
    } catch (err) {
      socket.emit('errorMessage',
          { tempId, error: 'Message not saved' });
    }
  });
```

Рисунок 4.8 – Обработчик событий

Включение tempId в полезную нагрузку позволяет клиенту отправителя согласовать назначенный сервером идентификатор сообщения с оптимистичным заполнителем, отображаемым в пользовательском интерфейсе. Все сокеты в room_123 (включая исходного отправителя) получают newMessage практически в реальном времени, что обеспечивает согласованность порядка сообщений.

Помимо сообщений, генерируемых пользователями, сервер также должен уведомлять клиентов о системных событиях, таких как присоединение или выход пользователя из комнаты. Когда новый сокет присоединяется к комнате, событие userJoined может быть передано (Рисунок 4.9).

```
socket.on('joinRooms', ({ rooms }) => {
    rooms.forEach(roomId => {
      socket.join(`room_${roomId}`);
      // Оповестить остальных участников
      socket.to(`room_${roomId}`).emit('userJoined', {
        userId: socket.user.id,
        userName: socket.user.name,
        roomId
      });
    });
  });
```

Рисунок 4.9 – Событие userJoined передается

Клиенты прослушивают userJoined и userLeft для обновления индикаторов присутствия (например, «Алиса печатает…» или отображения списка активных пользователей). Присутствие можно отслеживать просто в памяти — или, для масштабируемости, с поддержкой Redis или другого хранилища, если горизонтальное масштабирование (несколько экземпляров Node.js) становится необходимым.

Для повышения устойчивости каждое критическое событие подтверждает успех или неудачу. Например, после вызова saveMessageToDB сервер выдает newMessage неявно всем клиентам, что сообщение было сохранено и передано. Если сохранение не удается, сервер отправляет errorMessage обратно только на исходный сокет с достаточным количеством подробностей для пользовательского интерфейса, чтобы отобразить ошибку и при необходимости повторить попытку.

Если база пользователей института превышает возможности одного сервера, становится необходимым горизонтальное масштабирование. Socket.IO поддерживает Redis в качестве брокера сообщений для pub/sub на нескольких экземплярах Node.js. В определённой настройке каждый процесс Node.js запускает свой собственный экземпляр io.

Использование Redis в качестве адаптера гарантирует, что когда один экземпляр выдает newMessage, все остальные экземпляры, подписавшиеся на эту комнату, передают то же событие своим подключенным клиентам. Записи в базу данных остаются централизованными или могут быть соответственно синхронизированы.

4.5. Безопасность и шифрование

В основе безопасности приложения лежит гибридная криптосистема, сочетающая преимущества асимметричного и симметричного шифрования. Такой подход позволяет обеспечить надёжную защиту передаваемых данных и при этом сохранять высокую производительность при шифровании сообщений.

Этап регистрации пользователя сопровождается генерацией пары ключей (публичный/приватный) с использованием RSA (2048 бит). Публичный ключ сохраняется на сервере и доступен другим пользователям, в то время как приватный ключ остаётся на стороне клиента и не передаётся наружу. Рекомендуется сохранять его в зашифрованном виде, например, с использованием пароля или встроенного хранилища браузера.

При создании нового чата клиент генерирует уникальный симметричный AES-ключ, который будет использоваться для шифрования всех последующих сообщений в этом чате. Этот ключ затем шифруется с использованием публичного ключа получателя и передаётся вместе с первым сообщением.

После получения первого сообщения клиент расшифровывает вложенный AES-ключ с помощью своего приватного RSA-ключа и сохраняет его для дальнейшей работы. Таким образом, последующие сообщения расшифровываются быстрее — с помощью симметричного алгоритма AES.

Такая архитектура позволяет:

• Исключить возможность чтения сообщений третьими лицами (включая сервер);

- Не передавать приватные ключи по сети;
- Использовать асимметричное шифрование только на этапе установления связи, что снижает нагрузку на клиент и ускоряет работу чата.

В перспективе возможно расширение системы безопасности: хранение приватных ключей в зашифрованном виде с использованием WebCrypto API, внедрение цифровых подписей сообщений для проверки их целостности и подлинности, а также управление ключами при выходе пользователя из аккаунта или смене устройства.

5. ТЕСТИРОВАНИЕ И ОТЛАДКА

Прежде чем углубляться в конкретные тестовые случаи и результаты, важно описать теоретические основы тестирования и отладки программного обеспечения, применяемые к приложению обмена сообщениями в реальном времени. Тестирование программного обеспечения — это систематический процесс проверки того, что приложение соответствует своим требованиям и ведет себя ожидаемым образом в различных условиях. Оно охватывает несколько уровней — от низкоуровневых модульных тестов отдельных функций до высокоуровневых сквозных проверок полных рабочих процессов. Отладка, в свою очередь, — это искусство обнаружения, анализа и исправления дефектов, обнаруженных во время тестирования или сообщенных пользователями.

В этом проекте усилия были распределены по тестированию на три основные области:

- функциональное тестирование, которое подтверждает, что каждая функция работает в соответствии со спецификацией;
- тестирование производительности и стресс-тестирование, которое оценивает, как система реагирует при большой нагрузке;
- проверки, ориентированные на безопасность, которые гарантируют, что обработка конфиденциальных данных, таких как процедуры шифрования и соединения сокетов, ведет себя безопасно даже в крайних случаях.

Само функциональное тестирование делится на ручные и автоматизированные подходы. Ручное тестирование включает в себя человека-тестера, который следует реалистичным сценариям использования, таким как вход в систему, создание чата и отправка сообщений, чтобы обнаружить неожиданное поведение. Автоматизированное тестирование использует фреймворки, которые могут многократно выполнять небольшие целевые проверки модулей кода

(модульные тесты), комбинированных компонентов (интеграционные тесты) и пользовательских интерфейсов (компонентные или сквозные тесты).

Тестирование пограничных случаев (edge-case testing) проверяет поведение системы при нетипичных условиях или подаче ей экстремальных входных данных, например, очень длинные названия комнат или резкие сбои в работе сети. Так получается гарантировать, что приложение не даст сбой или не создаст несогласованные данные. Стресс-тестирование имитирует взаимодействие множества пользователей одновременно, отслеживая ЦП сервера, память и задержку сообщений для подтверждения приемлемых пороговых значений производительности.

В этой главе были описаны, как эти теоретические подходы были применены к нашей системе обмена сообщениями клиент-сервер. Тестирование было начато с ручных функциональных тестов, имитирующих действия пользователя, так переходя к автоматизированным модульным и интеграционным тестам, которые проверяют отдельные модули, а затем были представлены сценарии пограничных случаев и стресса. Наряду с этими описаниями были описаны проблемы, обнаруженные во время тестирования, и предложены исправления. Наконец, репрезентативные снимки экрана и таблицы производительности документируют стабильность и отзывчивость системы.

5.1 Проведенные тесты

Ручное функциональное тестирование: применяя ручное функциональное тестирование, тестировщик шаг за шагом следовал реалистичным пользовательским сценариям. Сначала он заходил на страницу входа, вводили действительные учетные данные и подтверждал, что были перенаправлены в чат-лобби. Ввод недействительных учетных данных вызывал сообщение об ошибке. После входа в систему тестировщик проверял, что боковая панель чата автоматически заполняется всеми доступными комнатами. По мере ввода текста в

поле поиска отображаемый список фильтровался в реальном времени без перезагрузки страницы. Создание комнаты включало вызов интерфейса «Создать комнату», назначение уникального имени, выбор участников и последующее подтверждение того, что новая комната мгновенно появилась на боковой панели. В личных комнатах, когда пользователь отправлял текстовое сообщение, оно немедленно появлялось как для отправителя, так и для получателя; тестировщик также было замечено, что, когда один и тот же пользователь отправлял последовательные сообщения, имя пользователя появлялось только один раз, что уменьшало визуальную избыточность. Для проверки поведения группового чата тестировщик вошел в систему как три отдельных пользователя (каждый в своем окне браузера), присоединился к одной комнате и публиковал сообщения поочередно. Все клиенты отображали, что сообщения в хронологическом порядке с точными временными метками. Наконец, прикрепление небольшого PDF-файла или изображения проверялось путем нажатия на значок вложения, выбора файла размером менее 5 МБ и проверки появления в чате кликабельной ссылки. Загрузка файла на стороне получателя подтверждала целостность данных. На протяжении этих этапов периодическое регулирование сети (например, снижение пропускной способности до 1 Мбит/с) гарантировало, что интерфейс плавно обрабатывал более медленные соединения без потери сообщений или зависания.

Автоматизированные модульные и интеграционные тесты: инициативы по автоматизированному тестированию были нацелены на отдельные внутренние модули и внешние компоненты. На сервере модульные тесты были сосредоточены на основных операциях с базой данных — добавлении и извлечении пользователей, создании комнат и хранении сообщений. Например, модульный тест вставит новую запись сообщения в таблицу сообщений, а затем запросит таблицу, чтобы подтвердить, что новая вставленная строка соответствует ожидаемым значениям (идентификатор комнаты, идентификатор автора и формат временной метки).

Обработчики событий сокетов также тестировались изолированно, испуская фиктивные события соединения или сообщения; затем утверждения проверяли, что сервер транслировал соответствующее последующее событие подключенным сокетам. Логика аутентификации прошла целевые проверки: генерация действительного JSON Web Token (JWT) была принята сервером, в то время как некорректные или просроченные токены были отклонены с ответом «401 Unauthorized».

На стороне клиента тесты компонентов React использовали библиотеку тестирования React для проверки того, что каждый элемент пользовательского интерфейса вел себя правильно с учетом определенных изменений свойств или состояния. Компонент <ChatList> был визуализирован с фиктивным массивом чат-комнат; ввод текста в имитированный поисковый ввод давал обновленный список комнат, который соответствовал критериям фильтра. В компоненте <MessageBubble> предоставление последовательных сообщений от одного автора приводило к групповому отображению с одним заголовком имени пользователя. Компонент <ConnectionManager> был протестирован с имитированным контекстом сокета, так что нажатие «Подключить» изменяло отображаемый статус с «Отключено» на «Подключено», а нажатие «Отключить» переключало его обратно.

Тесты сквозной интеграции, проведенные с помощью Cypress, автоматизировали рабочий процесс браузера, в котором виртуальный пользователь регистрировался, входил в систему, создавал комнату и обменивался сообщениями. При каждом тестовом запуске было проверено, что все отправленные сообщения сохранялись в базе данных и корректно перезагружались после обновления страницы.

Тестирование пограничных случаев и стресс-тестирование: помимо стандартных рабочих процессов, тестирование пограничных случаев было сосредоточено на необычных или экстремальных условиях. Попытка отправить

пустое сообщение запускала проверку на стороне клиента, которая отключала кнопку «Отправить», предотвращая бесполезные вызовы сервера. Ввод имени комнаты длиннее 255 символов приводил к ошибке на стороне клиента с сообщением «Имя комнаты не может превышать 255 символов» в соответствии со схемой базы данных. Загрузка файла размером более 10 МБ — превышение настроенного лимита — приводила к немедленному всплывающему сообщению об ошибке, а не к сбою сервера. Быстрое нажатие «Отключить», а затем «Подключить» проверяло надежность управления состоянием сокетов; окончательный пользовательский интерфейс всегда соответствовал состоянию сокета сервера, гарантируя, что ни один сокет не останется без применения.

Для тестирования производительности и стресс-тестирования простой генератор нагрузки имитировал 100 клиентов, подключенных к одному загруженному публичному чату, каждый из которых отправлял короткое (20-байтовое) текстовое сообщение каждые три секунды. При такой нагрузке загрузка ЦП сервера возросла примерно до 75 процентов, загрузка памяти достигла пика около 450 МБ, а средняя задержка трансляции сообщений осталась менее 200 мс. Когда все клиенты отключились, загрузка ЦП вернулась к базовому уровню в течение нескольких секунд.

5.2 Выявленные проблемы и решения

Во время тестирования возникло несколько заметных проблем:

6. Ошибки CORS: браузеры иногда блокировали запросы API (например, «GET /api/chats»), поскольку отсутствовали заголовки общего доступа к ресурсам из разных источников. Исправление включало добавление «app.use(cors({ origin: 'http://localhost:3000', credentials: true }));» на сервер Express и явное разрешение методов HTTP GET, POST и OPTIONS.

7. Цикл переподключения сокета: если сервер неожиданно перезапускался, клиенты перехватывали событие отключения и немедленно пытались

подключиться без ожидания. Это вызывало плотный цикл переподключения, который увеличивал загрузку ЦП. Настройка клиента Socket.IO с экспоненциальной задержкой — установка reconnectionAttempts: 5 и reconnectionDelayMax: 5000 — гарантировала, что каждая последующая попытка переподключения ждала дольше перед повторной попыткой.

8. Неправильный порядок сообщений: при высоком уровне параллелизма зависимость от предоставленных клиентом временных меток приводила к случайным ошибкам в упорядочивании сообщений. Решением стало назначение серверу собственной временной метки Date.now() при сохранении каждого сообщения, что гарантировало единый авторитетный источник времени. Затем клиенты использовали только временные метки, сгенерированные сервером, при отображении сообщений.

9. Задержка производительности поиска комнат: ввод в поле поиска вызывал заметную задержку, когда существовало более 200 комнат. Переход на поиск с отменой отказов (задержка 300 мс) и замена дорогостоящего сопоставления регулярных выражений простым сравнением toLowerCase() устранили эту задержку, что привело к мгновенному обновлению пользовательского интерфейса после короткой паузы.

10. Сбои при загрузке файлов: загрузка поврежденного PDF-файла иногда вызывала необработанные исключения. Внедрение проверки на стороне сервера с использованием библиотеки типов файлов — проверка типа MIME перед сохранением — предотвратило сбой сервера из-за недействительных файлов. Вместо этого сервер ответил чистым сообщением об ошибке.

11. Устаревшие списки участников группового чата: когда участник добавлялся в группу, существующие клиенты не обновляли автоматически свои списки участников, пока пользователь вручную не перезагрузил страницу. Отправка события сокета room_updated при каждом изменении членства позволяла

клиентам обновлять данные своих комнат в памяти и немедленно перерисовывать свои списки участников.

5.3 Результаты тестирования

Ниже представлены табличные данные и описания ключевых результатов тестирования.

Сценарий	**Ожидаемый результат**	**Фактический результат**	**Статус**
Корректный вход перенаправляет в лобби чатов	Перенаправление на лобби	Перенаправление выполнено	Пройдено
Некорректный вход показывает ошибку	Появление сообщения «Неправильный логин или пароль»	Сообщение отображается	Пройдено
Быстрое создание новой комнаты	Новая комната появляется в списке за 2 секунды	Комната появилась за 1.2 секунды	Пройдено
Отправка сообщения в приватном чате	Получатель видит сообщение в течение 100 мс	Доставлено за 50 мс	Пройдено
Последовательная отправка сообщений в группе	Все сообщения приходят в правильном порядке	В правильном порядке в течение 100 мс	Пройдено
Загрузка допустимого файла (< 5 МБ)	В истории чата появляется ссылка; загрузка проходит успешно	Ссылка появилась; загрузка успешна	Пройдено

Попытка загрузки большого файла (> 10 МБ)	Появление ошибки «Файл слишком большой»	Ошибка отображается; загрузка блокируется	Пройдено
Название комнаты > 255 символов	Появление ошибки «Название комнаты не может превышать 255 символов»	Ошибка отображается	Пройдено
Быстрое «Disconnect» → «Connect»	Клиент переподключается с экспоненциальной задержкой	Наблюдались задержки 1 с → 2 с → 4 с, подключение прошло	Пройдено

Таблица 4.1. Результаты функциональных тестов

Параметр	**Без нагрузки**	**100 активных пользователей**	**Пик во время «навалки»**
Загрузка CPU сервера	5 %	40 %	75 %
Потребление памяти сервера	200 МБ	320 МБ	450 МБ
Средняя задержка доставки	—	120 мс	250 мс
Пиковая задержка доставки	—	300 мс	800 мс

Таблица 4.2. Показатели нагрузочного тестирования

Скриншоты:

- Обработка ошибки при входе: при неудачной попытке входа отображается сообщение «Неправильный логин или пароль».
- Фильтрация списка комнат: при вводе «кафедра» в поле поиска отображаются только соответствующие комнаты.
- Мгновенная доставка сообщений: в приватном чате сообщение отправителя появляется сразу в окне получателя.
- Загрузка вложения: после выбора PDF-файла появляется кликабельная ссылка в истории чата.
- Загрузка CPU при нагрузочном тесте: график загруженности процессора, достигающий 75 % при эмуляции 100 клиентов.

Применяя теоретические принципы тестирования программного обеспечения, охватывающие ручные функциональные тесты, автоматизированные проверки модулей/интеграций, проверку пограничных случаев и стресс-тестирование, были выявлены и решены критические проблемы, такие как ошибки CORS, циклы переподключения сокетов, несоответствия в порядке сообщений, задержки производительности пользовательского интерфейса и сбои загрузки файлов. Окончательные результаты тестирования, задокументированные выше, демонстрируют, что все основные функции работают надежно, что производительность остается приемлемой при большой нагрузке и что компоненты пользовательского интерфейса правильно реагируют на пограничные случаи и ошибки. Эти успешные результаты закладывают прочную основу для продвижения вперед с планированием развертывания и экономическим обоснованием в следующей главе.

6. ОЦЕНКА ЭФФЕКТИВНОСТИ И РАЗВЕРТЫВАНИЕ

В этой главе рассматривается, как новое приложение для безопасного обмена сообщениями может быть интегрировано в существующую цифровую среду Читинского института, описываются варианты интеграции с сервером MS SQL организации для аутентификации пользователей и оценивается экономическая жизнеспособность путем сравнения затрат с истекающей в ближайшее время лицензией Net Speakerphone.

6.1 Развертывание в инфраструктуре института

Чтобы развернуть веб-мессенджер в инфраструктуре института, необходимо предпринять несколько практических шагов. Во-первых, компоненты frontend и backend приложения должны быть размещены на сервере, доступном в локальной сети (или интрасети), чтобы все преподаватели и сотрудники могли подключаться через свои браузеры без маршрутизации трафика через общедоступный Интернет. В идеале существующий сервер университета, используемый для внутренних порталов или студенческих служб, может быть снабжен необходимой средой выполнения Node.js, SQLite (или базой данных производственного уровня) и обратным прокси-сервером (например, Nginx) для маршрутизации трафика HTTPS в процесс Node.js.

Рекомендуемый подход — создать выделенный поддомен (например, messenger.institute.local) с сертификатами SSL/TLS, настроенными через «Let’s Encrypt» или внутренний центр сертификации. Это гарантирует, что весь HTTP-трафик будет зашифрован вплоть до границы сети учреждения. Правила брандмауэра должны разрешать только порт 443 (HTTPS) для сервера обмена сообщениями, полностью блокируя внешний доступ.

Поскольку приложение использует соединения WebSocket (Socket.IO), обратный прокси-сервер должен правильно пересылать запросы на обновление. Можно привести пример фрагмента конфигурации Nginx.

nginx.conf

```
server {
    listen 443 ssl;
    server_name messenger.institute.local;

    ssl_certificate /etc/letsencrypt/live/messenger.institute.local/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/messenger.institute.local/privkey.pem;

    location / {
        proxy_pass http://127.0.0.1:4000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

После настройки сервера подключение пользователей может осуществляться поэтапно. Пилотной группе, например, одному академическому отделу и его административному персоналу, следует предоставить ранний доступ. Это контролируемое развертывание позволяет ИТ-персоналу выявлять любые проблемы с сетью или разрешениями до развертывания в масштабах всего учреждения. Отзывы от пилотных пользователей также помогают точно настраивать разрешения, назначения ролей и элементы пользовательского интерфейса.

Администратор может создавать группы, соответствующие каждому факультету, отделу или службе (например, «Финансы», «Кадры», «УМИО», «Деканат ФЭФ»). Интеграция с существующей документацией и учебные материалы института помогут сотрудникам понять переход со старых каналов (электронная почта, Net Speakerphone) на новый мессенджер.

На стороне клиента пользователям требуется только современный браузер (Chromium, Firefox и те, что построены на них) и действительные учетные данные. Интерфейс входа использует HTTPS для отправки имени пользователя и пароля на бэкэнд. Управление доступом на основе ролей может ограничивать пользователей определенными групповыми чатами или административными функциями (такими

как создание комнат или удаление сообщений). В случае разрыва соединения WebSocket (например, в сегментированной сети) клиент автоматически переходит к режиму длительного опроса в качестве запасного варианта, обеспечивая непрерывность связи даже в условиях ограниченных сетевых ресурсов

6.2 Интеграция с MS SQL

Хотя приложение изначально было разработано с SQLite для простоты развертывания, центральная ИТ-среда учреждения опирается на Microsoft SQL Server для управления учетными записями пользователей и создания отчетов. Интеграция мессенджера с MS SQL позволяет сотрудникам продолжать использовать имеющиеся учетные данные, устраняя дублирование управления учетными записями и усиливая безопасность за счет использования установленных политик аутентификации.

Существуют две основные стратегии:

Прямая аутентификация с помощью MS SQL: в этом подходе сервер мессенджера подключается к базе данных MS SQL через пакет mssql Node.js. Когда пользователь пытается войти в систему, сервер выполняет хранимую процедуру или параметризованный запрос к таблице Users учреждения (или эквиваленту).

В этом случае мессенджер проверяет учетные данные, используя тот же алгоритм хеширования, что и основной процесс аутентификации учреждения. Если аутентификация прошла успешно, сервер генерирует токен сеанса (JWT) для последующих вызовов API и подключений сокетов. Этот метод гарантирует, что сотрудникам не нужно запоминать отдельные имена пользователей и пароли: достаточно их учетных данных MS SQL.

```
import sql from 'mssql';
const config = {
  user: 'ms_user',
  password: 'ms_password',
  server: 'sql.economy.lan',
  database: 'InstitutionAD',
  options: {
    encrypt: true,
    trustServerCertificate: true
  }
};

async function authenticate(username, password) {
  await sql.connect(config);
  const result = await sql.query`
    SELECT id, firstName, lastName
    FROM Users
    WHERE login = ${username} AND
        pwdHash = HASHBYTES('SHA2_256', ${password})`;
  return result.recordset.length > 0
    ? result.recordset[0]
    : null;
}
```

Периодический импорт данных в SQLite: в качестве альтернативы, если прямые запросы к MS SQL требуют чрезмерных сетевых издержек или если центральная база данных изолирована за строгим брандмауэром, ночное задание может экспортировать записи пользователей из MS SQL в локальную базу данных SQLite мессенджера. Простой скрипт, запущенный через PowerShell или задание Linux cron, экспортирует активные идентификаторы пользователей, имена пользователей и хешированные пароли в файл CSV. Затем бэкэнд мессенджера считывает CSV при запуске или по требованию, синхронизируя собственную таблицу пользователей. Хотя этот подход вводит небольшую задержку (не более 24 часов) между изменениями в MS SQL и списке пользователей мессенджера, он

устраняет необходимость для мессенджера поддерживать постоянные соединения MS SQL. Любая стратегия может быть выбрана на основе сетевых политик учреждения и требований к производительности.

В обоих случаях интеграция с MS SQL укрепляет единый источник достоверной информации для учетных записей пользователей, согласуется с существующими политиками безопасности ИТ и упрощает процессы ротации паролей или деактивации учетных записей.

6.3 Оценка и смягчение рисков

Внедрение новой коммуникационной платформы по своей сути несет как технические, так и организационные риски.

Одной из основных проблем является принятие пользователем: сотрудники, привыкшие к существующим каналам, могут сопротивляться изменениям. Чтобы смягчить это, руководители отделов могут продемонстрировать преимущества системы на раннем этапе, а краткие практические семинары подчеркнут экономию времени и повышенную безопасность.

С технической точки зрения представлять опасность может производительность при пиковой нагрузке. Сервер будет обеспечен вдвое большей емкостью, а автоматические оповещения будут отмечать использование ЦП выше 70% или использование памяти выше 80%. Если производительность ухудшается, можно активировать вторичный экземпляр с балансировкой нагрузки.

Риски безопасности возникают, если конфигурации TLS или шифрования неверны. Ежеквартальные проверки настроек TLS и поэтапное развертывание гибридного шифрования защитят от раскрытия данных.

Наконец, интеграция с MS SQL для аутентификации может не сработать, если учетные данные не синхронизируются должным образом; при первоначальном развертывании аутентификации в режиме только для чтения любые несоответствия можно выявить до предоставления полного доступа.

Если центральная база данных временно становится недоступной, откат к локально хранящимся учетным записям пользователей SQLite обеспечивает непрерывность.

Резервные снимки и автоматизированные процедуры аварийного переключения дополнительно защищают от сбоев сервера или сетевых сбоев, перенаправляя критические оповещения в список рассылки по экстренным случаям, когда это необходимо.

6.4 Мониторинг и отчетность

Для поддержания надежности система будет постоянно контролироваться, а отчеты будут создаваться с регулярными интервалами. Панель мониторинга в реальном времени отслеживает использование ЦП, памяти и сети, чтобы выявлять аномалии до того, как они повлияют на пользователей, в то время как журналы активности сокетов регистрируют события подключения/отключения и объемы сообщений.

Каждую неделю автоматизированная сводка описывает ежедневное количество активных пользователей, общее количество отправленных сообщений и среднюю задержку; эти сводки отправляются ИТ-менеджменту.

Аудиты безопасности проводятся ежемесячно, проверяя неудачные попытки входа, подозрительные модели активности и любые незапланированные перезапуски сервера.

Оповещения настроены на немедленное уведомление ИТ-отдела, если загрузка ЦП превышает 70 % в течение более пяти минут или если средняя задержка сообщения превышает 500 мс в течение одной минуты.

Ежеквартальные обзорные совещания объединяют ИТ-администраторов, руководителей отделов и высшее руководство для обсуждения тенденций, корректировки инфраструктуры и планирования обновлений функций. Этот цикл

обратной связи обеспечивает прозрачную коммуникацию, быстрое реагирование на проблемы и постоянное улучшение надежности и безопасности платформы.

6.5 Показатели успеха (KPI)

Измерение эффективности новой платформы обмена сообщениями требует отслеживания ключевых показателей производительности, которые отражают принятие, использование и влияние на эффективность рабочего процесса. Предлагаемые KPI включают:

• Процент активных пользователей: отношение уникальных пользователей, которые входят в систему не реже одного раза в день, к общему числу лицензированных пользователей. Цель: ≥ 75 % ежедневного показателя активных пользователей к 3-му месяцу после запуска; ≥ 90 % к 6-му месяцу.

• Среднее время ответа: медианное время между отправкой сообщения и его появлением в поле зрения получателя. Цель: ≤ 200 мс при нормальной нагрузке.

• Темп роста общения: процентное увеличение общего количества сообщений, которыми обмениваются в месяц. Цель: 10 % ежемесячный рост в первом квартале по мере присоединения большего количества отделов.

• Объем заявок в службу поддержки, связанных с обменом сообщениями: количество заявок в службу поддержки, отнесенных к категории «проблемы с сообщениями», в месяц. Цель: ≤ 5 заявок на 100 пользователей в месяц после стабилизации системы.

• Оценка удовлетворенности пользователей: самооценка удовлетворенности по шкале Лайкерта (1–5), собранная с помощью периодических опросов. Цель: ≥ 4,0 средняя удовлетворенность через три месяца.

• Сокращение использования устаревших каналов: процентное снижение количества писем или телефонных звонков во внутреннюю поддержку для рутинных запросов, которые теперь происходят через мессенджер. Цель: 50 % сокращение избыточных потоков писем в течение шести месяцев.

Регулярная оценка этих KPI позволяет принимать решения на основе данных для дальнейшей оптимизации, целевого обучения или разработки дополнительных функций.

6.6 Экономическая эффективность

Текущее решение Net Comfort стоит 200 000 ₽ за лицензию на обслуживание 100 пользователей в течение четырех лет. Срок действия этих лицензий истекает в следующем году, а это значит, что институту предстоит решить вопрос о продлении: продолжать платить 200 000 ₽ еще четыре года (всего за 100 пользователей) или перейти на недавно разработанный внутренний мессенджер.

Чтобы сравнить затраты, рассмотрим два сценария за четыре года:

Продление Net Speakerphone: Стоимость лицензии: 200 000 ₽ на 100 пользователей в течение четырех лет. Стоимость за пользователя, распределенная на 48 месяцев: 200 000 ÷ 100 ÷ 48 ≈ 41,67 ₽ на пользователя в месяц. При расширении на дополнительных пользователей каждый блок из 100 пользователей потребует еще 200 000 ₽ в течение следующих четырех лет.

Развертывание внутренне разработанного мессенджера:

Единовременные затраты на разработку: ~30 000 ₽ (из расчета 75 рабочих часов по 400 ₽/час). Расходы на хостинг сервера: незначительны, так как существующий сервер перепрофилируется. Обслуживание и поддержка: оценка 5 часов в месяц рабочего времени ИТ-персонала (по 500 ₽/час) = 2 500 ₽/месяц; ежегодно = 30 000 ₽; за четыре года = 120 000 ₽

Статья расходов	**New Comfort (лицензия)**	**Собственное решение**
Стоимость лицензии (100 пользователей)	200 000 ₽	0 ₽
Разработка (единовременная)	0 ₽	26 000 ₽
Хостинг (4 года)	0 ₽	40 000 ₽

Поддержка (4 года)	0 ₽	120 000 ₽
Итого за 4 года	**200 000 ₽**	**186 000 ₽**
Цена на одного пользователя за 4 года	**2 000 ₽**	**1 860 ₽**

Таблица 5.1. Прямое сравнение затрат на 100 пользователей (4 года)

На первый взгляд внутреннее решение кажется не сильно дешевле (1 860 ₽ против 2 000 ₽ за пользователя в течение четырех лет) с учетом постоянной поддержки. Однако это сравнение пока не учитывает масштабируемость внутренней системы: после разработки она может поддерживать любое количество пользователей без дополнительных лицензионных сборов. И в инстиуте есть на то спрос: на данный момент NetComfort использует все 100 подключений. В то же время, добавление дополнительного блока из 100 пользователей в NetComfort требует еще 200 000 ₽.

Статья расходов	**New Comfort (500 пользователей)**	**Собственное решение**
Лицензия (500 пользователей)	1 000 000 ₽ (200 000 × 5)	0 ₽
Разработка (единовременная)	0 ₽	26 000 ₽
Хостинг (4 года)	0 ₽	40 000 ₽
Поддержка (4 года)	0 ₽	120 000 ₽
Итого за 4 года	**1 000 000 ₽**	**186 000 ₽**
Цена на одного пользователя за 4 года	**2 000 ₽**	**372 ₽**

Таблица 5.2. Затраты на 500 пользователей (4 года)

При 500 пользователях внутреннее решение становится на 80 % дешевле за пользователя за четыре года. В этом случае общая экономия составит 1 000 000 − 186 000 = 814 000 ₽.

Чтобы определить точку безубыточности (количество пользователей N, при котором внутренние затраты ≤ Чистая стоимость спикерфона), установите:

Расчёт точки безубыточности

Обозначим через N количество пользователей.

Затраты New Comfort (N пользователей за 4 года):

$$NetCost\ N = 200\,000 * N100 = 2\,000 * N \text{ рублей}$$

Затраты собственного решения (независимо от числа пользователей):

$$InHouseCost = 26\,000 + 40\,000 + 120\,000 = 186\,000 \text{ рубле й}$$

Точка безубыточности определяется условием:

$$2\,000 * N \geq 186\,000\ N \geq 168\,0002\,000\ N \geq 93$$

Таким образом, начиная с 93 пользователей, собственное решение становится более экономичным, чем продление лицензии New Comfort. Учитывая, что в институте насчитывается более 93 сотрудников и преподавателей (плюс потенциальные пилотные студенческие группы), индивидуальный мессенджер финансово оправдан.

6.7 Итоговая оценка

Развертывание защищенного веб-мессенджера в инфраструктуре Читинского института технически просто: для этого требуется выделенный виртуальный внутренний сервер (или перепрофилирование существующего), правильная настройка SSL и поэтапное развертывание в пилотных отделах. Интеграция аутентификации с сервером MS SQL учреждения обеспечивает бесперебойный пользовательский опыт, используя существующие учетные данные и устраняя параллельное управление учетными записями.

С экономической точки зрения, хотя внутренняя система имеет скромные первоначальные и эксплуатационные расходы, она быстро становится более рентабельной, чем продление лицензий Net Speakerphone, когда база пользователей превышает примерно 100 человек. Это особенно актуально, учитывая

приближающееся окончание срока действия текущей лицензии за 200 000 ₽ на 100 пользователей. По мере роста института или расширения использования на других сотрудников и студентов каждый дополнительный пользователь не несет дополнительных лицензионных сборов, что означает значительную долгосрочную экономию.

Подводя итог, можно сказать, что совокупная техническая осуществимость, соответствие существующим ИТ-политикам и явное преимущество в стоимости делают внутренний защищенный мессенджер предпочтительным выбором для внутренних коммуникационных потребностей Читинского института.

ЗАКЛЮЧЕНИЕ

В ходе работы над дипломным проектом была разработана и протестирована клиент-серверная система обмена сообщениями, удовлетворяющая требованиям внутренней коммуникации структурных подразделений образовательного учреждения. Реализован модуль пользовательского интерфейса на React с использованием компонентного подхода, обеспечивающего автономность и повторное использование блоков, а также серверная часть на Node.js/Express, отвечающая за хранение данных, обработку WebSocket-сообщений и базовую логику безопасности. Проведён анализ существующих решений, обоснована необходимость создания собственного мессенджера, а также выполнено моделирование бизнес-процессов и архитектуры системы с помощью UML и IDEF.

Основные цели проекта — обеспечение реального времени при передаче сообщений, создание структурированных приватных и публичных чатов, интеграция с корпоративной базой пользователей и закладывание основы для шифрования — были достигнуты. Результатом стало работоспособное веб-приложение, способное обслуживать сотни пользователей без существенных задержек при передаче сообщений. Теоретическая часть раскрыла особенности защиты данных в госсекторе, а практическая — продемонстрировала интеграцию с инфраструктурой института: от настройки HTTPS-сертификатов и конфигурации Nginx до возможности аутентификации через MS SQL. Экономическая оценка показала, что уже при 93 активных пользователях собственное решение становится более выгодным по сравнению с продлением лицензии New Comfort.

Внедрение было спланировано поэтапно: от пилотного запуска в одном подразделении до полного перевода всех сотрудников и студентов на новую систему. Подготовлены материалы для обучения, определены ответственные за поддержку и методы мониторинга, что позволит оперативно реагировать на возникающие задачи и поддерживать стабильность работы. Также оценены

основные риски — от низкой активности пользователей до возможных проблем с производительностью — и предложены способы их минимизации.

В будущем проект может быть доработан в нескольких направлениях. Во-первых, следует реализовать полноценную схему гибридного шифрования (AES + RSA) для гарантии конфиденциальности переписки, расширив уже заложенные в архитектуру компоненты. Во-вторых, целесообразно добавить функционал интеграции с электронным дневником и системой выставления оценок, а также возможность отправки учебных материалов и домашних заданий. В-третьих, можно развить административную панель: разделить права доступа для различных ролей, внедрить гибкую систему логирования действий пользователей и реализовать отчётность об активности. Кроме того, полезно будет провести нагрузочное тестирование с учётом возросших требований к числу одновременных соединений и оптимизировать работу при пиковых нагрузках.

Таким образом, проделанная работа заложила фундамент для создания надёжного и безопасного инструмента внутренней коммуникации вуза, а описанные планы развития позволят системе эволюционировать и отвечать новым требованиям образовательной среды.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сведения об образовательной организации согласно приказу Федеральной службы по надзору в сфере образования и науки: [Электронный ресурс]. – Электронные данные. – Режим доступа: http://bgu-chita.ru/sveden

2. Ekaterina Zavgorodnikh, Automation of Company Processes Using a Corporate Messenger, Пермь: Information Analytics in Enterprise Management, 2024. — 816 с.

3. UML Documentation [Электронный ресурс]: https://www.uml-diagrams.org/

4. Habr. Что такое дерево решений и где его используют: [Электронный ресурс]. – Электронные данные. – Режим доступа: https://habr.com/ru/company/productstar/blog/523044/

5. Space, grids, and layouts — Design Systems [Электронный ресурс]: https://www.designsystems.com/space-grids-and-layouts/