МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

КОРПОРАТИВНЫЙ МЕССЕНДЖЕР ЗАЩИЩЁННОГО ВЗАИМОДЕЙСТВИЯ СОТРУДНИКОВ В ЛОКАЛЬНОЙ СЕТИ

Студент _____ 25.05.2026 Коннов А.Д.
(подпись, дата)

Руководитель _____ 25.05.2026 Слепцов Н.В.
(подпись, дата)

Нормоконтролёр _____ 25.05.2026 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 22.05.2026 г. №14)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 08.06.26 № 14)

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2026

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

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
_____М.А. Митрохин
«07» 04 2026г.

ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ

1. Студент Коннов Антон Дмитриевич гр.22ВВП1 факультета вычислительной техники.

2.Тема работы: «Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети»

Тема утверждена приказом ПГУ №347/о от "07" апреля 2026 г.

3. Руководитель бакалаврской работы к.т.н., доцент, каф. ВТ ПГУ Слепцов Н.В.

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

Разработать приложение «Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети», позволяющее безопасно обмениваться сообщениями и данными в удобном формате между пользователями. Приложение должно обладать графическим и интуитивно понятным интерфейсом, для взаимодействия с пользователем. Приложение должно предоставить возможность обмениваться текстовыми сообщениями, мультимедиа или аудио в реальном времени с высокой степенью безопасности данных.

5. Перечень подлежащих разработке вопросов

Требуется произвести анализ предметных областей: изучить существующие корпоративные мессенджеры, выделить их достоинства и недостатки. Выбрать язык программирования, который подойдет для создания графического интерфейса. Определить архитектуру приложения и вспомогательные средства разработки. Изучить функционал выбранных средств для разработки.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	32	07.04.2026 – 18.05.2026	
2	Анализ требований	32	10.04.2026 – 20.05.2026	
3	Создание макета графического интерфейса	30	13.04.2026 – 17.04.2026	
4	Выбор технологий, применяемых при проектировании и разработке	40	24.04.2026 – 28.04.2026	
5	Реализация графического интерфейса	66	1.05.2026 – 6.05.2026	
6	Разработка основной части программы	70	7.05.2026 – 13.05.2026	
7	Оформление пояснительной записки	54	14.05.2026 – 21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы _____ Слепцов Н.В..
(подпись, дата)

Задание к исполнению принял студент _____ Коннов А.Д.
(подпись, дата)

Работу к защите допустить
Декан факультета 03.06.2026 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 124 листа, 58 рисунков 1 таблицу, 6 источников, 7 приложений.

КОРПОРАТИВНЫЙ МЕССЕНДЖЕР ЗАЩИЩЁННОГО ВЗАИМОДЕЙСТВИЯ СОТРУДНИКОВ В ЛОКАЛЬНОЙ СЕТИ.

Цель работы – разработать корпоративный мессенджер, позволяющий сотрудникам взаимодействовать между собой в условиях локальной сети.

Объектом разработки является приложение, работающее исключительно в локальной сети, имеющее собственную базу данных и удобный графический интерфейс.

Разработка проводилась в среде программирования Visual Studio Code. В качестве языка программирования был выбран TypeScript.

В результате проведенной работы разработано приложение, удовлетворяющее поставленной задаче, а именно: удобный пользовательский интерфейс, наличие вариантов кастомизации и работа исключительно в локальной сети.

Созданное приложение может быть внедрено в Пензенский государственный университет для взаимодействия сотрудников между собой в локальной сети.

Содержание

Перечень сокращений и обозначений 7
Введение 6
1 Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети 7
1.1 Постановка задачи 8
1.2 Анализ существующих аналогов 10
1.2.1 Корпоративный мессенджер Bitrix24 10
1.2.2 Корпоративный Яндекс.Мессенджер 12
1.2.3 Мессенджер Telegram 13
1.2.4 Мессенджер MAX 14
1.2.5 Выводы 15
1.3 Выбор языка программирования 16
1.3.1 Язык программирования C++ 16
1.3.2 Язык программирования JavaScript 16
1.3.3 Язык программирования Python 16
1.3.4 Язык программирования TypeScript 16
1.4 Выводы 17
1.5 Выбор инструментов разработки 17
1.5.1 Выбор среды разработки 17
1.5.2 Выбор платформы для разработки серверной части 17
1.5.3 Выбор базы данных 18
1.5.4 Выбор технологии для взаимодействия базы данных и сервера 18
1.5.5 Выбор инструмента для разработки клиентской части 18
1.5.6 Выбор инструмента для сборки 19
1.6 Настройка инструментов разработки 19
1.6.1 Настройка среды разработки 19
1.6.2 Настройка платформы Node.js 21

1.6.3 Настройка React 22
1.6.4 Настройка ORM Prisma 22
1.6.5 Настройка Vite 23
1.7 Архитектура приложения 23
1.7.1 Архитектура фронтенд-части приложения 23
1.7.2 Архитектура бэкенд-части приложения 24
1.8 Разработка приложения 25
1.8.1 Разработка графического интерфейса 25
1.8.2 Разработка интерфейса администратора 32
1.8.3 Разработка модулей 33
1.8.3.1 Класс Chat 33
1.8.3.2 Класс Message 35
1.8.3.3 Класс ChatDetails 35
1.8.3.4 Класс Login 36
1.8.3.5 Класс Form 36
1.8.3.6 Класс Navigation 37
1.8.3.7 Класс Profile 37
1.8.3.8 Класс ProfileSettings 37
1.8.3.9 Класс LocalStorage 38
1.8.4 Разработка базы данных 39
1.9 Тестирование приложения 40
1.10 Описание приложения 42
1.10.1 Руководство пользователя 42
2 Охрана труда 55
2.1 Анализ негативных факторов, воздействующих на оператора ПЭВМ и способы их устранения 55
2.1.1 Оборудование рабочего места 56
2.1.2 Освещённость рабочего пространства 58
2.1.3 Электробезопасность 58

2.1.4 Пожарная безопасность 59
2.1.5 Режим труда и отдыха 60
Заключение 61
Список использованных источников 62
Приложение А – UML-диаграммы 63
Приложение Б – Листинг программы 69
Приложение В – Презентация 106

Перечень сокращений и обозначений

В данной работе используются следующие сокращения и обозначения

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств

ИИ – искусственный интеллект

ОЗУ – оперативное запоминающее устройство

ПЗУ – постоянное запоминающее устройство

ПК – персональный компьютер

ПЭВМ – персональная электронно-вычислительная машина

СУБД – система управления базами данных

API (Application Programming Interface) – это программный интерфейс, который позволяет разным приложениям «общаться» и обмениваться данными друг с другом

DOM (Document Object Module) – это объектная модель документа, которую браузер создает в памяти компьютера на основании HTML-кода, полученного им от сервера

URL (Uniform Resource Locator) – это точный адрес любого ресурса в интернете

UML (Unified Modeling Language) – унифицированный язык моделирования

Введение

С каждым годом увеличивается процент населения, который пользуется мобильными телефонами или другими гаджетами. Огромный рост происходит и сфере оперативной и удобной коммуникации между пользователями. Это происходит как за счёт развития информационных технологий, так и за счёт общего ускорения темпа жизни. При этом пользователи предпочитают, чтобы обмен информацией происходил практически мгновенно, а также надеются на безопасность своих данных. По этим причинам сейчас высокий спрос идёт на такие платформы, как Bitrix24, Яндекс Мессенджер, Telegram, MAX и прочие.

Эти мессенджеры предоставляют пользователям огромный функционал разных возможностей, в том числе:

– обмен текстовыми и голосовыми сообщениями;

– передача изображений, аудио- и видеозаписей;

– отправка файлов;

– создание аудио- и видеозвонков

– создание личных и общих чатов;

– поддержка каналов и сообществ.

Помимо этих функций каждая платформа может иметь уникальный в своём роде функционал. В пример можно привести интеграцию с ботами, автоматизацию процессов, синхронизацию между устройствами и облачное хранение данных.

Целью выпускной работы является разработка корпоративного мессенджера, который будет работать исключительно в локальной сети и удовлетворять критериям безопасности.

1 Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети

Актуальность данной темы обусловлена отсутствием собственного защищённого мессенджера, позволяющего безопасно коммуницировать преподавателям и прочим сотрудникам Факультета Вычислительной Техники Пензенского Государственного Университета. Безопасность данных очень важна при обмене служебной информацией, персональными данными студентов, внутренних документов и прочего.

Существующие аналоги не соответствуют требованиям безопасности из-за нескольких проблем:

– отсутствие полного контроля над хранением и обработкой данных;

– риски утечки конфиденциальных данных;

– невозможность адаптации системы под специфику учебного заведения.

По данным причинам использование сторонних мессенджеров не обеспечит достаточного уровня защищённости данных, что создаёт риски для образовательного учреждения.

1.1 Постановка задачи

Требуется разработать приложение для обеспечения быстрого и безопасного взаимодействия между пользователями в локальной сети. Оно должно обеспечивать удобство использования на уровне с современными мессенджерами, но при этом иметь высокую степень безопасности.

Функциональные требования приложения:

– администрирование. В системе должен быть администратор, который имеет возможность зарегистрировать пользователя в системе, выдать ему роль или создать групповой чат;

– авторизация пользователей. Система должна предоставить возможность для авторизации пользователя, которому ранее был создан аккаунт администратором;

– просмотр списка чатов. После успешной авторизации пользователь должен иметь доступ к списку доступных для него диалогов с возможностью выбора одного из них;

– просмотр истории сообщений. В каждом чате должна храниться и отображаться полная история переписки с момента его создания;

– обмен текстовыми сообщениями. Пользователь должен иметь возможность отправлять и получать текстовые сообщения от других пользователей;

– обмен голосовыми сообщениями. Пользователь должен иметь возможность отправлять или получать голосовые сообщения от других пользователь с возможностью их воспроизведения;

– совершение аудио- и видеозвонков. Приложение должно обеспечивать соединение между пользователями для осуществления голосовых или видеозвонков;

– передача файлов. Пользователь должен иметь возможность отправки и получения файлов различных форматов с возможностью их сохранить на своё устройство;

– отправка мультимедийного контента. Пользователь должен иметь возможность отправлять эмодзи или GIF-изображения среди доступных;

– кастомизация профиля. Пользователь должен иметь возможность изменять данные своего профиля, которые должны быть видны другим пользователям.

Технические требования приложения:

– наличие клиент-серверной архитектуры. Должно быть обеспечено разделение на frontend и backend разработку;

– обмен данными должен быть осуществлен по протоколу TCP/IP;

– система должна поддерживать одновременную работу нескольких пользователей, все взаимодействия с интерфейсом должны выполняться без заметных задержек;

– при потере соединения система должна оперативно его восстанавливать, а также правильно обрабатывать ошибки. Потеря данных при сбоях не допустима;

– передача данных должна быть осуществлена с применением технологий шифрования, данные пользователей должны храниться в хешированном виде;

– должно быть обеспечено хранение учётных записей пользователей, сообщений, файлов и мультимедиа;

– интерфейс приложения должен быть интуитивно понятным.

Системные требования:

– операционная система Windows;

– минимум 512 Мб ОЗУ и ПЗУ;

– наличие любого браузера.

1.2 Анализ существующих аналогов

1.2.1 Корпоративный мессенджер Bitrix24

Bitrix24 [bitrix24.ru] – это не просто мессенджер, это полноценный российский сервис для управления бизнесом, в который интегрированы различные функции для этого: календарь, облачное хранилище и онлайн документы, бизнес-процессы, мобильное приложение, ИИ для бизнеса, онлайн подпись и сам корпоративный мессенджер. Это всё является одним большим преимуществом этой платформы, что привлекает огромную аудиторию и всё больше организаций выбирают её для организации деятельности.

Такое большое количество функций влечёт за собой и очевидные минусы. Самым большим можно выделить избыточность функционала,

которая будет очень сложной для освоения рядовому пользователю. Также платформа имеет облачное хранилище, что может повлечь за собой возможные утечки данных, а также он не предоставляет полный функционал для работы в локальной сети. Графический интерфейс мессенджера Bitrix24 представлен на рисунках 1.1, 1.2.

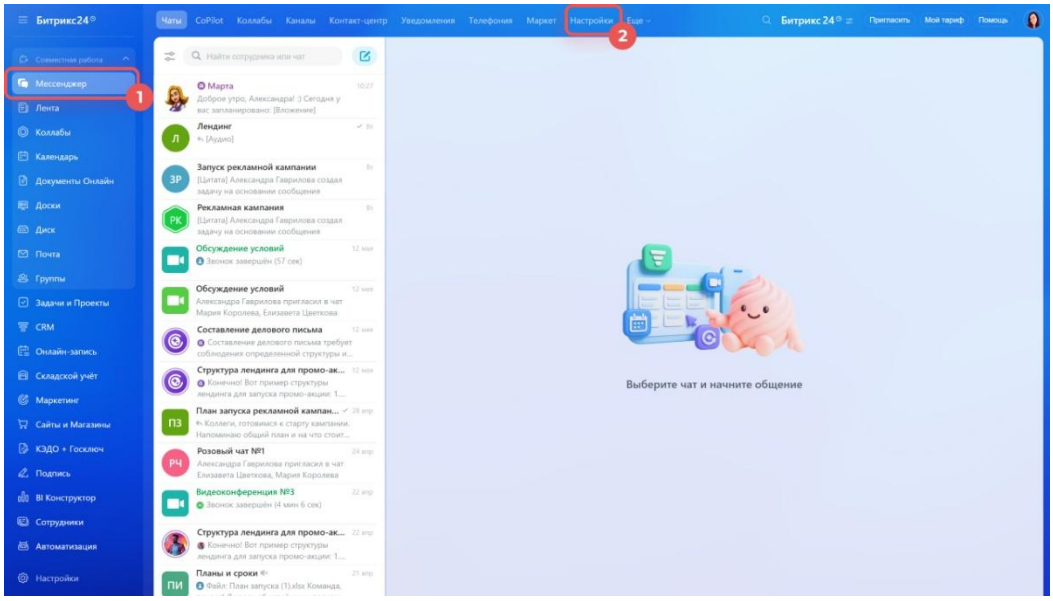


Рисунок 1.1 – Окно выбора чатов

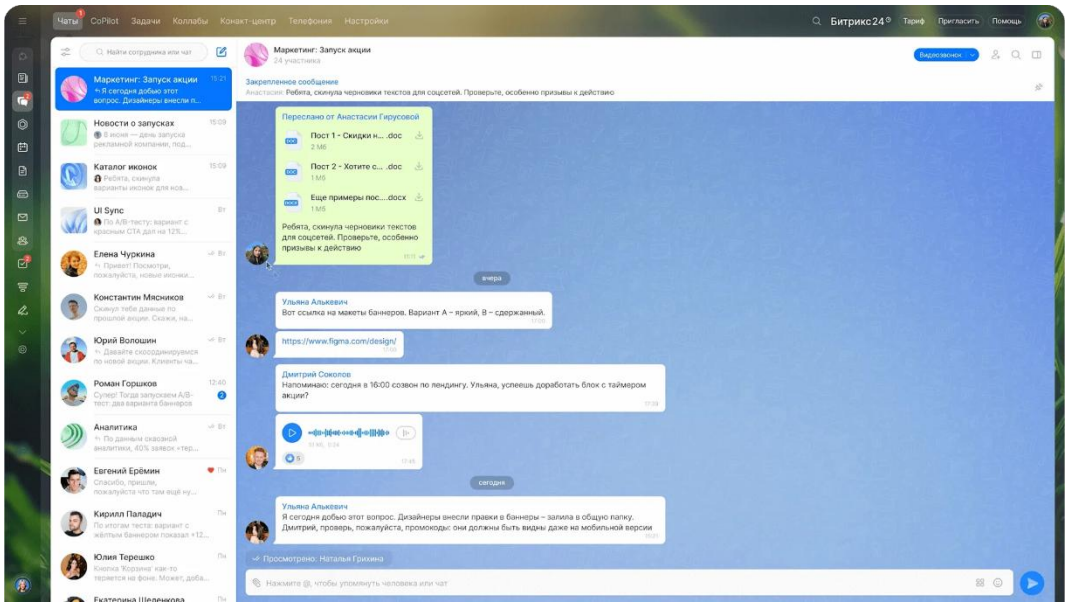


Рисунок 1.2 – Окно чата

Исходя из вводных данных для разрабатываемого приложения можно сказать, что данный мессенджер не подошёл бы для поставленной цели за счёт неполноценной работы в локальной сети, что ставит под угрозу безопасность конфиденциальных данных и важных документов.

1.2.2 Корпоративный Яндекс.Мессенджер

Яндекс.Мессенджер [360.yandex.ru] – система мгновенного обмена сообщениями, входящая в экосистему Яндекса. Она позволяет переписываться, отправлять голосовые сообщения, которые автоматически превращаются в текст, разговаривать голосом и по видеосвязи, читать и обсуждать информацию в каналах. Также в систему встроен виртуальный ИИ ассистент Алиса, который помогает пользователю в написании текстов или исправлении ошибок.

За счёт своего нахождения в экосистеме Яндекса мессенджер полностью интегрирует с другими сервисами Яндекса, например диском, почтой или календарём. Он имеет простой и понятный интерфейс, что привлекает многих пользователей. Графический интерфейс приложения представлен на рисунках 1.3, 1.4.

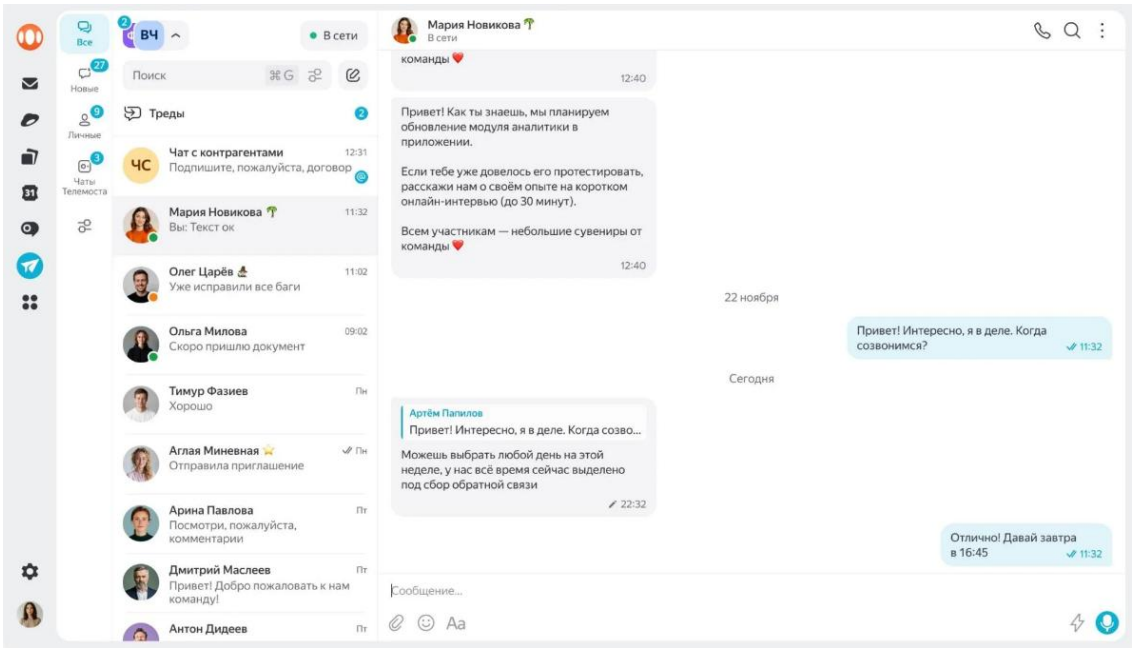


Рисунок 1.3 – Окно чата

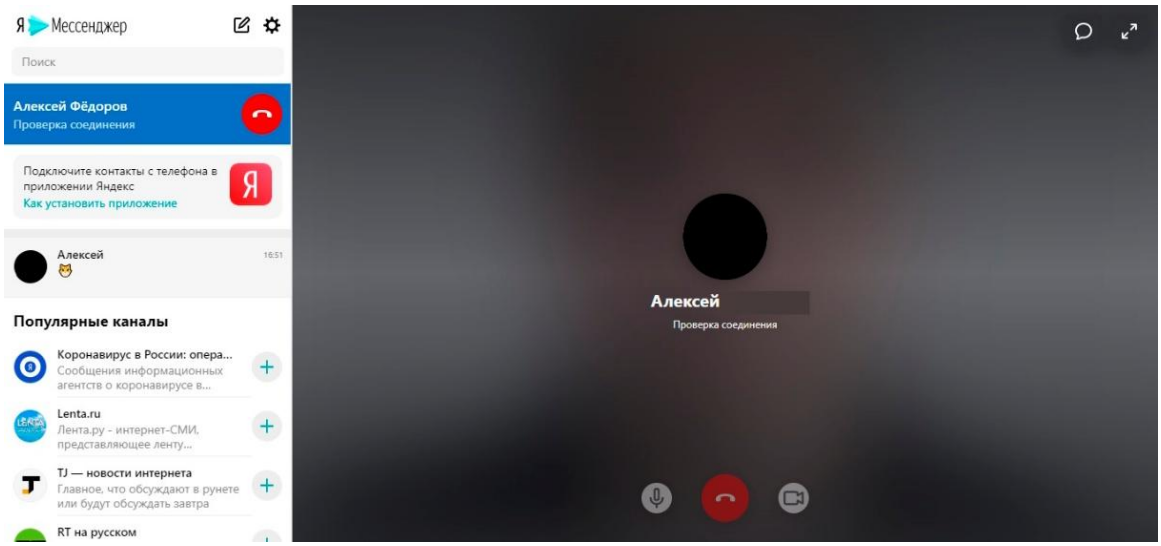


Рисунок 1.4 – Окно аудиозвонка

Если Bitrix24 может частично работать в локальной сети, то Яндекс.Мессенджер полностью лишён этой возможности из-за только облачного хранения данных, что ставит под сомнение вопросы безопасности и контроля над данными. Также платформа имеет большие ограничения в кастомизации под конкретного пользователя.

1.2.3 Мессенджер Telegram

Самым востребованным и активно развивающимся мессенджером является Telegram [web.telegram.org], который был разработан российским бизнесменом Павлом Дуровым с 2013 году. Можно сказать, что он не является корпоративным, но это не мешает огромному количеству организаций использовать его для своих целей на постоянной основе. На данный момент этот мессенджер является лидером по узнаваемости и популярности среди современных пользователей по всему миру. К его преимуществам можно отнести высокую скорость работы, облачную синхронизацию, очень развитую экосистему ботов, возможность создания каналов и сообществ. Недостатками являются несоответствия требованиям безопасности, например отсутствие сквозного шифрования для данных во всех чатах по умолчанию, а только в «приватных». Графический интерфейс приложения представлен на рисунках 1.5, 1.6.

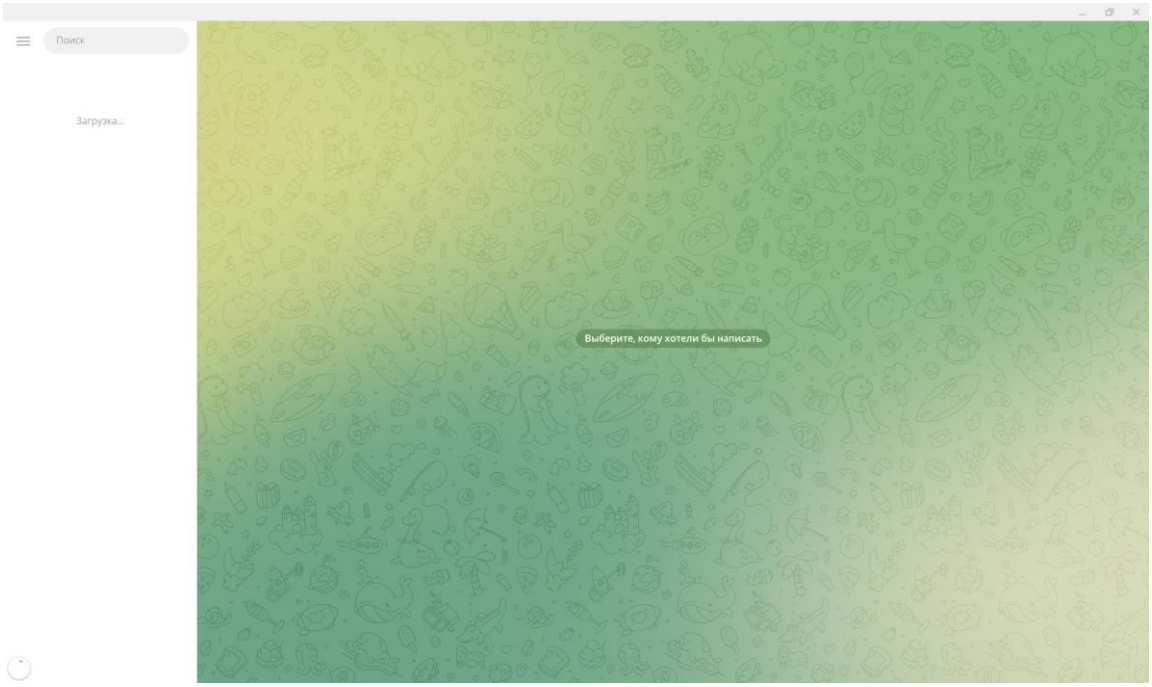


Рисунок 1.5 – Окно выбора чатов

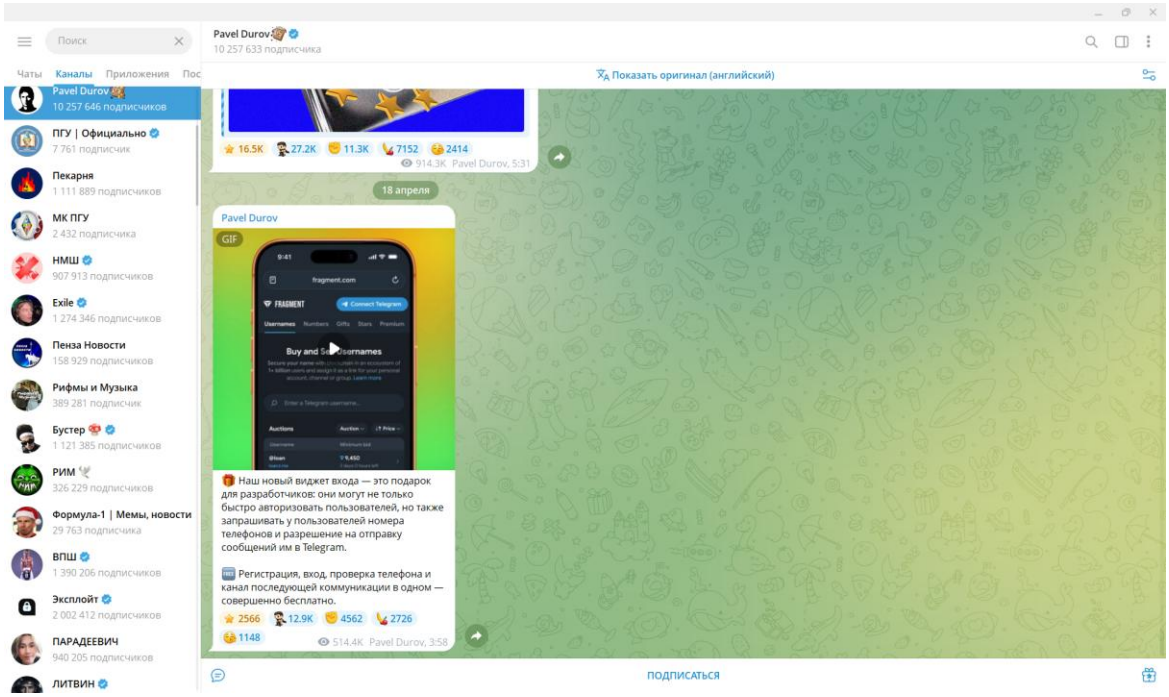


Рисунок 1.6 – Окно общедоступного канала

1.2.4 Мессенджер MAX

Не таким популярным, но не менее обсуждаемым в нашей стране сейчас является мессенджер MAX [max.ru], который является «новичком» на

данном рынке. Он был разработан в 2025 году с целью выхода на лидирующие позиции среди прочих аналогов. С каждым днём всё больше организаций переходят на данную платформу и используют её для рабочих целей. К преимуществам мессенджера можно отнести современный интерфейс и высокую скорость работы в возможных условиях ограничения сети. Однако, среди пользователей платформа не пользуется высокой популярность за счёт своей новизны и отсутствии проверенной времени репутацией в области безопасности. Графический интерфейс приложения представлен на рисунке 1.7.

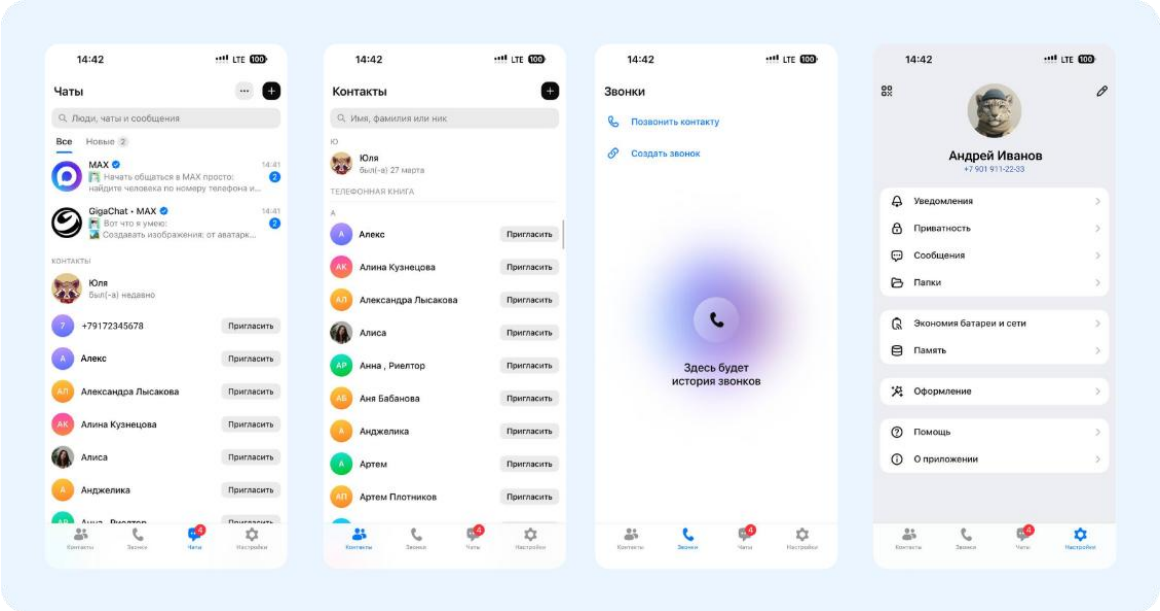


Рисунок 1.7 – Графический интерфейс мобильного приложения

1.2.5 Выводы

По итогам анализа можно сделать вывод, что ни один существующий мессенджер не подходит под задачи, поставленные требованиям разрабатываемого приложения по причине отсутствия возможности работы исключительно в локальной сети.

1.3 Выбор языка программирования

1.3.1 Язык программирования C++

C++ – компилируемый, статически типизированный язык программирования общего назначения. Он поддерживает разные способы программирования, например объектно-ориентированный, процедурный и т. д. Это делает данный язык достаточно гибким, но недостаточно для создания графического интерфейса.

1.3.2 Язык программирования JavaScript

JavaScript – это мультипарадигменный, легкий интерпретируемый язык программирования, являющийся основным инструментом для создания интерактивных веб-страниц, динамического контента, анимации и управления мультимедиа. Его преимуществом является работа и в браузере (frontend), и на сервере (backend). Это позволяет делать сайты функциональными с поддержкой объектно-ориентированного и императивного стиля.

1.3.3 Язык программирования Python

Python – мультипарадигменный высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью. Его преимуществом является очень простой синтаксис, что позволяет быстрее и проще изучить язык и ускорить разработку.

Стандартные библиотеки и интерпретатор можно установить бесплатно за счёт наличия их исходного кода в открытом доступе на официальном сайте.

При всей простоте языка у него есть существенные недостатки, а именно более медленная работа относительно других аналогов.

1.3.4 Язык программирования TypeScript

TypeScript – язык программирования, позиционируемый как инструмент, расширяющий возможности JavaScript. Он является обратно

совместимым с JavaScript и компилируется в него. Это означает, что после компиляции программу на этом языке можно выполнять в любом современном браузере или использовать совместно с серверной платформой Node.js.

1.4 Выводы

Исходя из вышеперечисленной информации было принято решение выбрать для разработки язык программирования TypeScript, поскольку он является отличным вариантом для разработки веб-приложений за счёт больших возможностей в разработке графического интерфейса и хорошего взаимодействия с серверной платформой.

1.5 Выбор инструментов разработки

1.5.1 Выбор среды разработки

Как среду разработки было решено выбрать Visual Studio Code [visualstudio.com], разработанную корпорацией Microsoft. Преимуществом этой среды является простой пользовательский интерфейс с большими возможностями в кастомизации. Также немаловажным плюсом является наличие встроенного отладчика, а также инструментов для работы с Git. Для упрощения написания кода встроена подсветка синтаксиса. Главным преимуществом можно выделить наличие огромного количества подключаемых расширений, что позволяет пользователю использовать только то, что ему нужно.

1.5.2 Выбор платформы для разработки серверной части

Для разработки серверной части выбрана платформа Node.js [nodejs.org] – кроссплатформенная среда выполнения с открытым исходным кодом. Она способна превращать JavaScript из узкоспециализированного языка в язык общего назначения, а также взаимодействовать ему с устройствами ввода-вывода через свой API. Эта платформа применяется

преимущественно на сервере, но также предоставляет возможность разрабатывать десктопные оконные приложения или программировать микроконтроллеры.

В состав платформы входит собственный менеджер пакетов npm с помощью которого можно установить, опубликовать или найти пакеты из репозитория.

1.5.3 Выбор базы данных

В качестве системы управления базами данных выбрана SQLite – компактную встраиваемую реляционную базу данных, не требующую установки отдельного серверного программного обеспечения. Данная СУБД представляет собой библиотека, с которой компонуется программа, что делает движок, который не является отдельным процессом, её составной частью. В качестве протокола обмена используются вызовы функции (API). SQLite хранит в одном файле всю базу данных.

1.5.4 Выбор технологии для взаимодействия базы данных и сервера

Для организации взаимодействия было принято решение использовать ORM Prisma [prisma.io] – технология, которая позволяет работать с базой данных через объекты языка программирования. Это позволяет избежать написать большого количества SQL-запросов вручную.

С помощью Prisma можно централизовано управлять моделями данных, что повышает читаемость кода и упрощает разработку. Также происходит автоматическая генерация клиента для работы с базой данных, который полностью интегрируется с языком программирования TypeScript.

1.5.5 Выбор инструмента для разработки клиентской части

Для реализации клиентской части выбран фреймворк React [react.dev], который используется для разработки пользовательского интерфейса. Он отлично подходит для разработки веб-приложений за счёт своей простоты, масштабируемости и высокой скорости работы.

React использует виртуальный DOM, что позволяет вычислять разницу между текущим и предыдущим состоянием интерфейса для оптимального обновления браузера. Это позволяет пользователю работать со страницей, думая, что она вся обновляется, но на самом деле библиотека сама решает, какие компоненты необходимо обновить.

1.5.6 Выбор инструмента для сборки

Для разработки и сборки frontend-приложения используется инструмент Vite [vite.dev] – локальный сервер разработки, который поддерживает язык программирования TypeScript и поддерживает фреймворк React.

Vite отслеживает файлы во время редактирования, что позволяет во время сохранения файла веб-браузеру перезагрузить редактируемый код с помощью процесса под названием Hot Module Replacement, который работает за счёт перезагрузки конкретного файла вместо повторной компиляции всего приложения. Это заметно сокращает время ожидания во время разработки приложения.

1.6 Настройка инструментов разработки

1.6.1 Настройка среды разработки

Для установки Visual Studio Code необходимо перейти на официальный сайт и скачать установщик. После успешной установки запускаем среду, начальное окно которой представлено на рисунке 1.8.

Для более комфортной разработки можно установить необходимые расширения, но для использования разрабатываемого приложения таковые не требуются. Окно с выбором расширений представлено на рисунке 1.9.

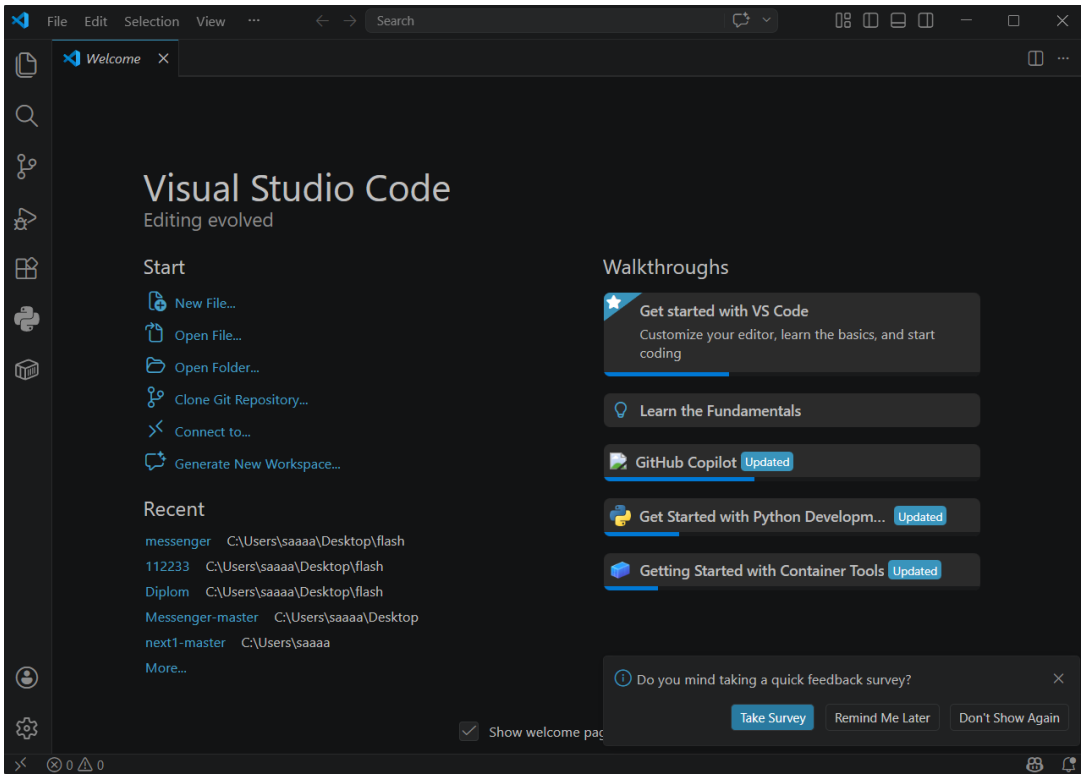


Рисунок 1.8 – Начальное окно

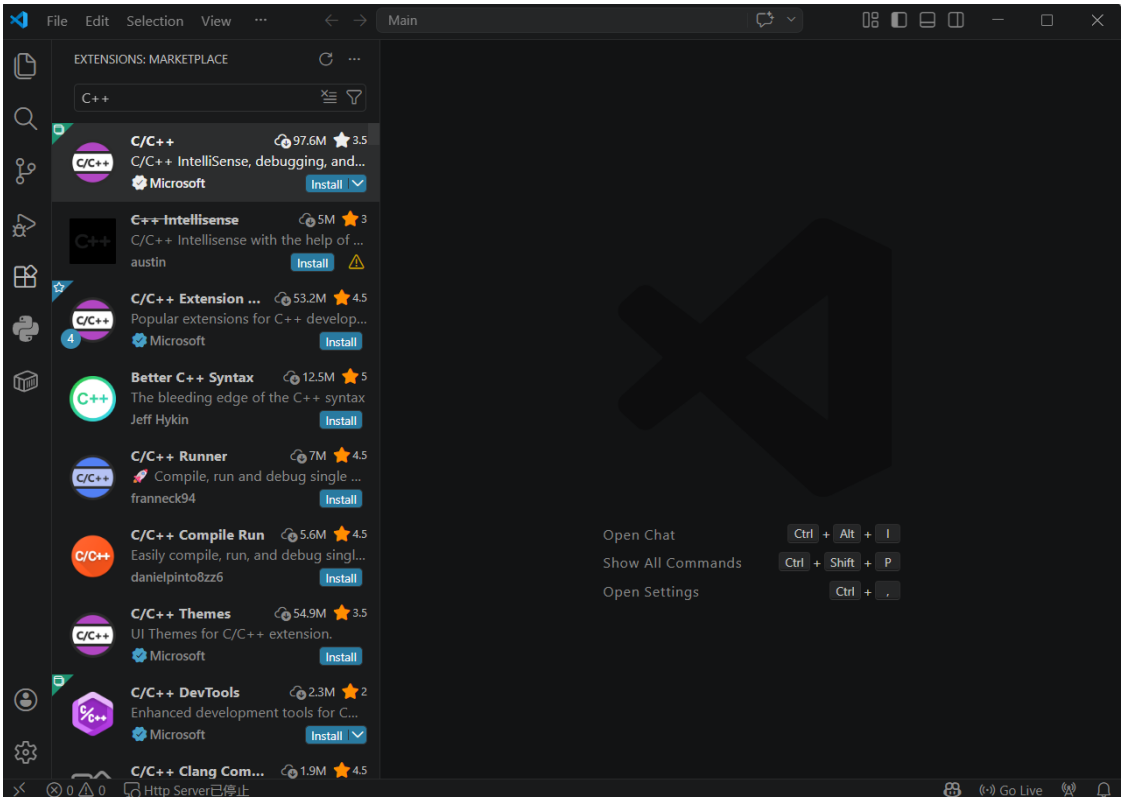


Рисунок 1.9 – Окно выбора расширений

Чтобы начать разработку необходимо выбрать папку, в которой будут помещаться все сопутствующие файлы. Для этого нужно создать её в любом месте на компьютере, а потом выбрать, либо выбрать уже существующую. В обоих вариантах переходим во вкладку File – Open Folder. После этого можно приступать к разработке. Окно папки представлено на рисунке 1.10.

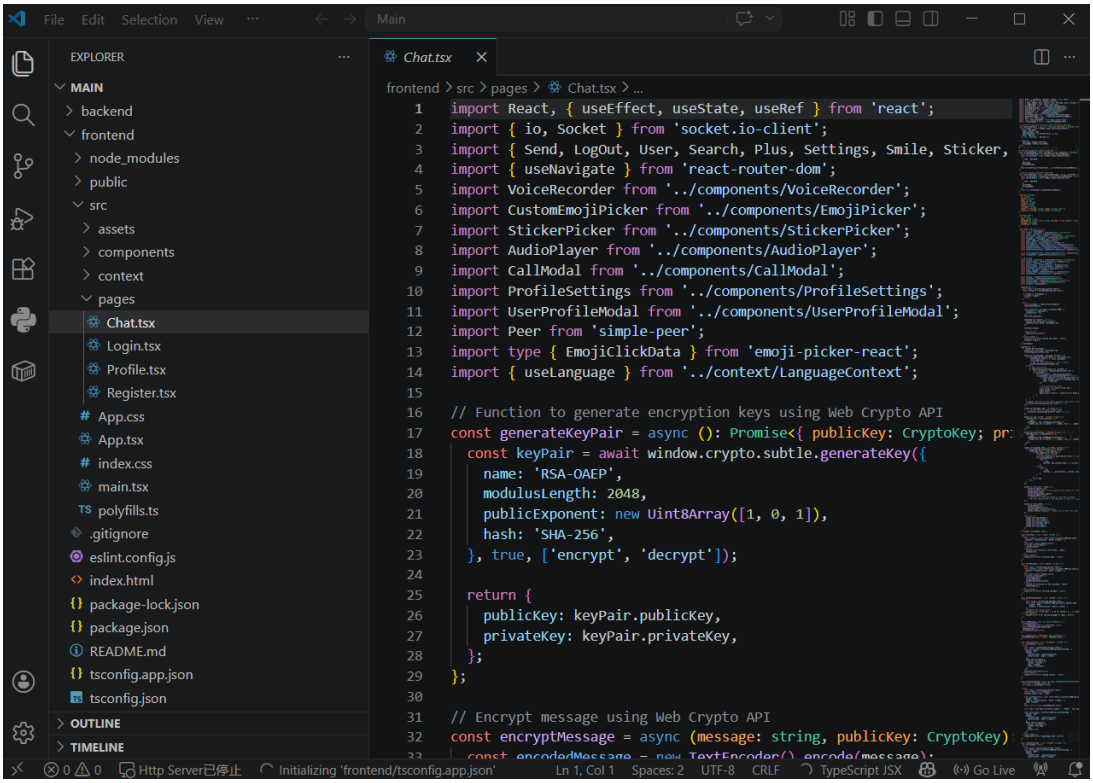


Рисунок 1.10 – Окно выбранной папки

1.6.2 Настройка платформы Node.js

Для установки платформы Node.js на операционную систему Windows необходимо перейти на официальный сайт и скачать MSI-установщик, запустить его с настройками по умолчанию. Он автоматически добавит на компьютер Node.js и пакетный менеджер npm. Для проверки работоспособности можно перейти в командную строку и ввести npm -v. Если всё прошло успешно, то в ответ вылезает установленная версия. Результат успешной установки показан на рисунке 1.11.

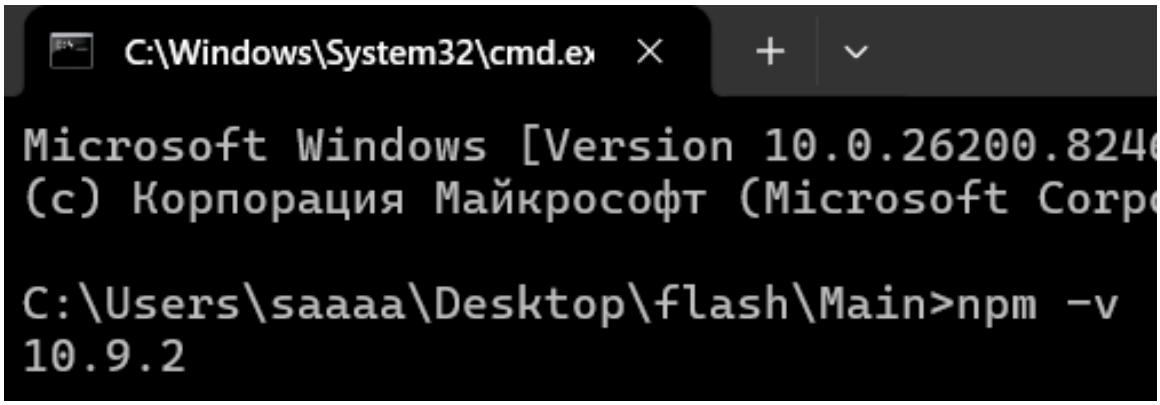


Рисунок 1.11 – Результат успешной установки

1.6.3 Настройка React

Для создания React приложения в указанной ранее папке для разработки нужно перейти в командную строку и ввести команду npm install react react-dom. Если установка прошла успешно – в папке с проектом появится соответствующая папка react. Наличие её показано на рисунке 1.12.

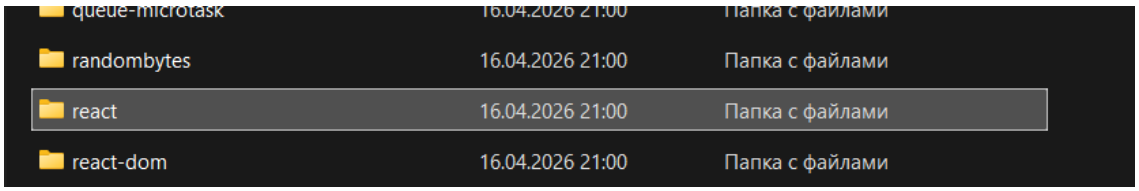


Рисунок 1.12 – Наличие папки react

1.6.4 Настройка ORM Prisma

Для установки ORM Prisma нужно убедиться в наличии Node.js и npm, сделать это можно по ранее описанным действиям. После этого переходим в командную строку и вводим команду npm install -g prisma. После успешной установки её нужно инициализировать командой npx prisma init. При успешной инициализации в папке проекта появится новая папка prisma, её наличие показано на рисунке 1.13.

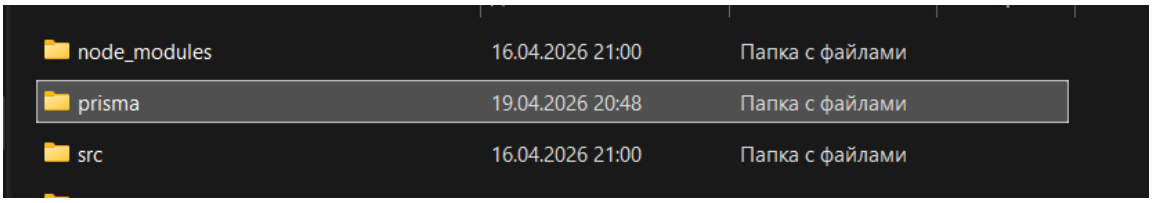


Рисунок 1.13 – Наличие папки prisma

Для завершения настройки нужно сгенерировать клиент для работы с базой данных командой npx prisma generate. Последнее действие, которое нужно совершить – добавить миграции командой npx prisma migrate dev. Они необходимы для управления изменениями структуры.

1.6.5 Настройка Vite

Для установки Vite в созданный проект нужно в командной строке ввести команду npm install vite @vitejs/plugin-react, после чего в папке появится соответствующая папка vite. Её наличие показано на рисунке 1.14.

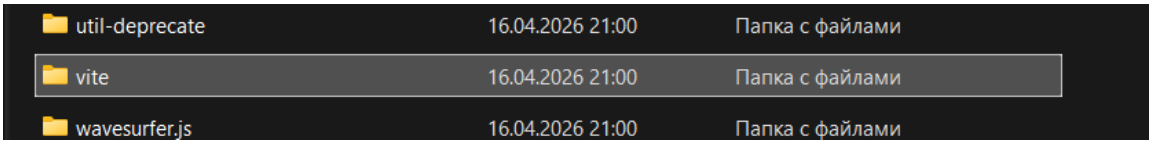


Рисунок 1.14 – Наличие папки vite

1.7 Архитектура приложения

Приложение представляет из себя клиент-серверную систему, включающую в себя фронтенд-часть (интерфейс пользователя) и бэкенд-часть (сервер).

1.7.1 Архитектура фронтенд-части приложения

Фронтенд-часть разработанного приложения является по сути своей клиентом. Она реализована с использованием JavaScript-библиотеки React, которая используется для создания графического интерфейса, и языка программирования TypeScript. Для настройки среды разработки и сборки проекта используется библиотека Vite.

Каждая функциональность этой части реализована отдельными компонентами, которые инкапсулируют логику. Для объединения всех компонентов реализованы страницы Chat, Login и Profile. Они предоставляют пользователю доступ ко всем основным функциям приложения. Подробная архитектура представлена на рисунке 1.15.

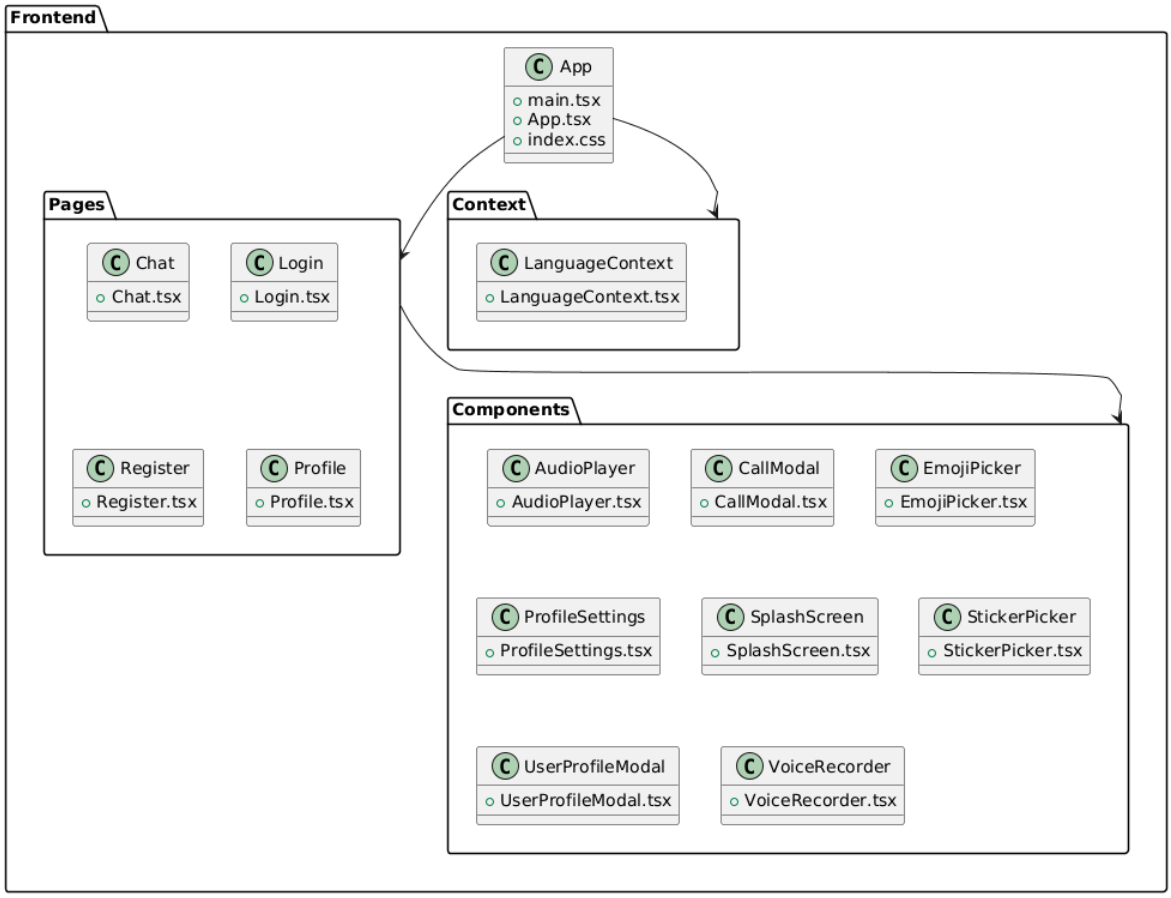


Рисунок 1.15 – Архитектура фронтенд-части приложения

1.7.2 Архитектура бэкенд-части приложения

Бэкенд-часть разработанного приложения является сервером и построена с использованием фреймворка Node.js и языка программирования TypeScript. Здесь реализован основной серверный файл app.ts, отвечающий за настройку приложения, подключение middleware и маршрутов.

Бэкенд реализован с использованием модульного принципа. Здесь имеются контроллеры authController, chatController, messageController и

userController. Они содержат логику обработки запросов, а маршруты уже связывают их с нужными URL. Также реализован Middleware – программное обеспечение, которое используют различные приложения, чтобы взаимодействовать друг с другом. Это файлы authMiddleware и uploadMiddleware, которые нужны для проверки авторизации пользователей или загрузки файлов на сервер, проверяя их типы. Подробная архитектура представлена на рисунке 1.16.

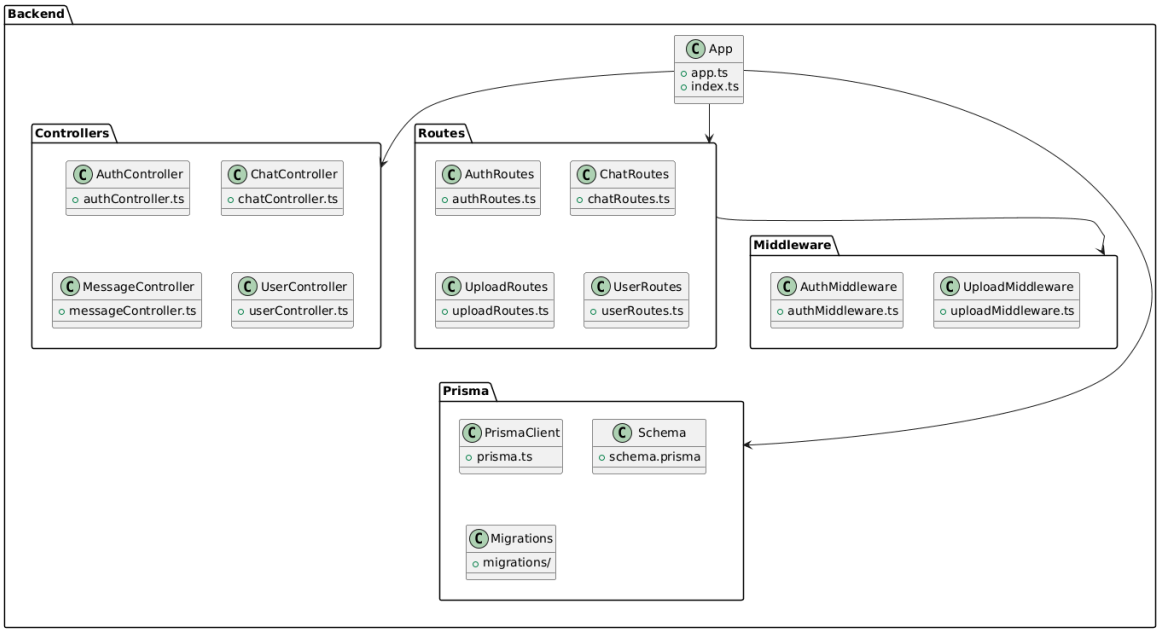


Рисунок 1.16 – Архитектура бэкенд-части приложения

1.8 Разработка приложения

1.8.1 Разработка графического интерфейса

Графический интерфейс должен давать пользователю возможность взаимодействовать с приложением в соответствии с UML-диаграммой вариантов использования, представленной на рисунке 1.17.

Рисунок 1.17 – UML-диаграмма вариантов использования

Графический интерфейс состоит из разных страниц (pages), которые представляют собой основные экраны приложения. Они используются для объединения компонентов и логики для отображения нужных функций пользователю.

Chat – страница, содержащая основной экран чата. Здесь пользователь видит список чатов и сообщений, имеет возможность отправлять текстовые, голосовые сообщения, совершать аудио- и видеозвонки, отправлять мультимедиа и файлы.

Login – страницы, содержащая окно авторизации. Здесь пользователь может ввести логин и пароль для входа в свой аккаунт.

Profile – страница, содержащая окно профиля. Здесь пользователь может увидеть информацию о себе или выбранном пользователе и редактировать данные профиля.

Во всех страницах есть свои компоненты, которые инкапсулируют логику и стилизацию для конкретных частей приложения.

AudioPlayer – компонент для воспроизведения аудиосообщений, поддерживающий управление воспроизведением или отображением прогресса.

CallModal – модальное окно для звонков. Оно отображает информацию о текущем звонке и содержит кнопки для завершения звонка или отключения микрофона.

EmojiPicker – компонент для выбора эмодзи, которые можно использовать в чатах для отправления эмоций в сообщениях. Поддерживает поиск по эмодзи.

ProfileSettings – компонент для изменения настроек профиля пользователя, который позволяет загружать аватар, менять имя пользователя.

SplashScreen – экран загрузки, отображаемый при запуске приложения. Он содержит логотип приложения.

StickerPicker – компонент для выбора стикеров, позволяющий отправлять в чат графические стикеры.

UserProfileModal – модальное окно для отображения профиля пользователя. Он показывает информацию о пользователе, его аватар и имя.

VoiceRecorder – компонент для записи голосовых сообщений. Он позволяет пользователю начать записывать голосовое сообщение с возможностью его дальнейшей отправки в чат или, при желании, отмены записи.

Графический интерфейс представлен на рисунках 1.18 – 1.27.

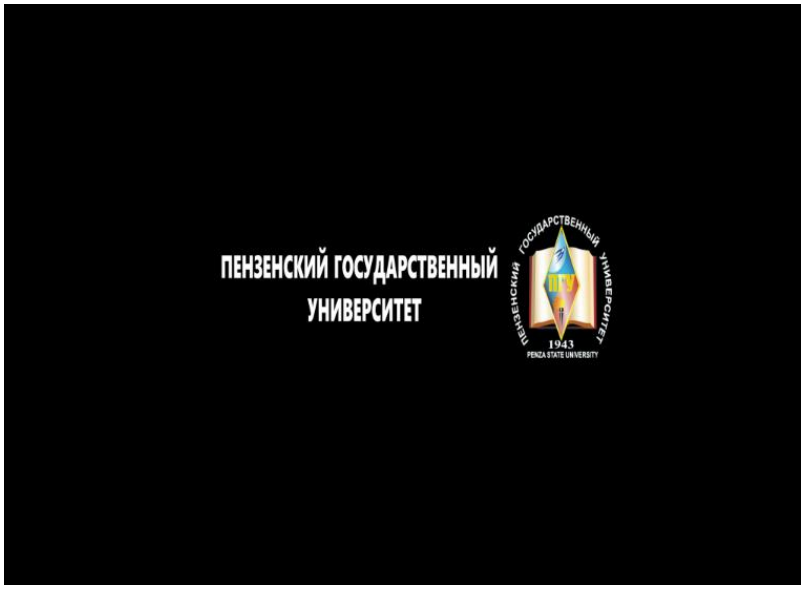


Рисунок 1.18 – Окно загрузки

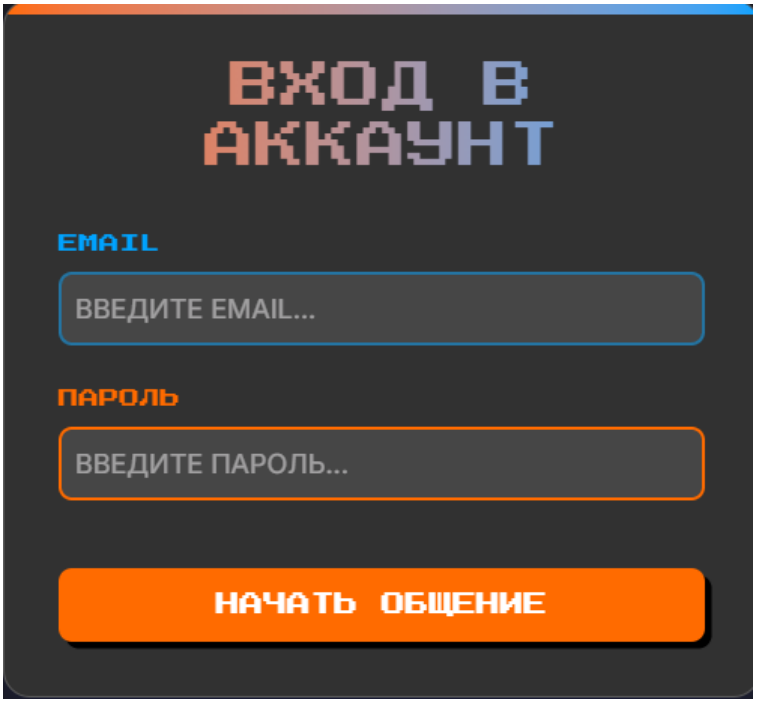


Рисунок 1.19 – Окно авторизации

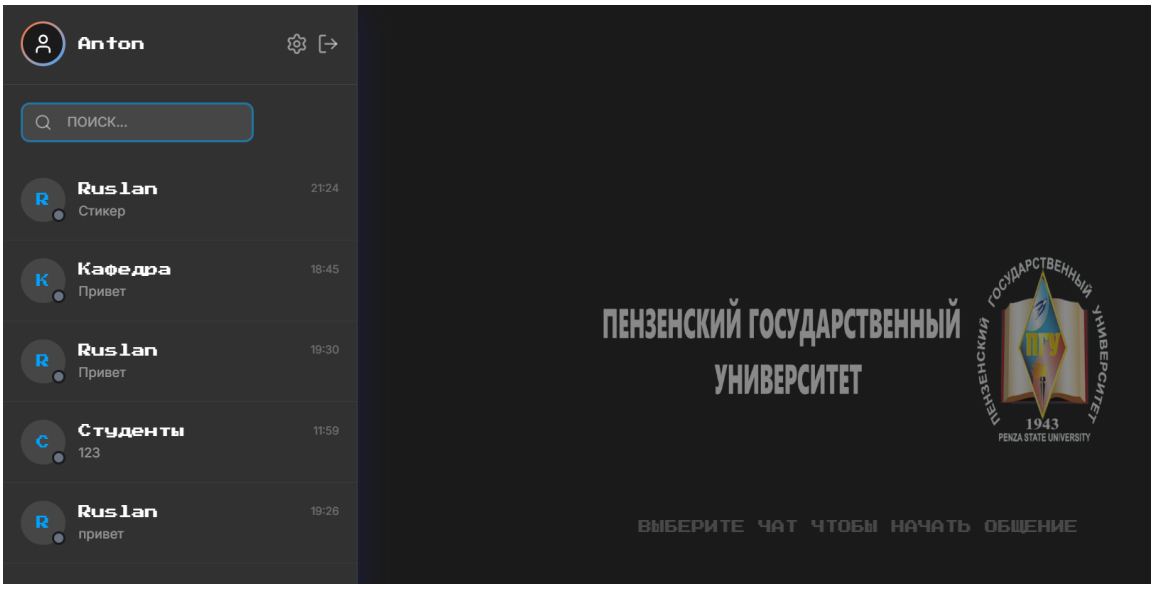


Рисунок 1.20 – Окно начального экрана

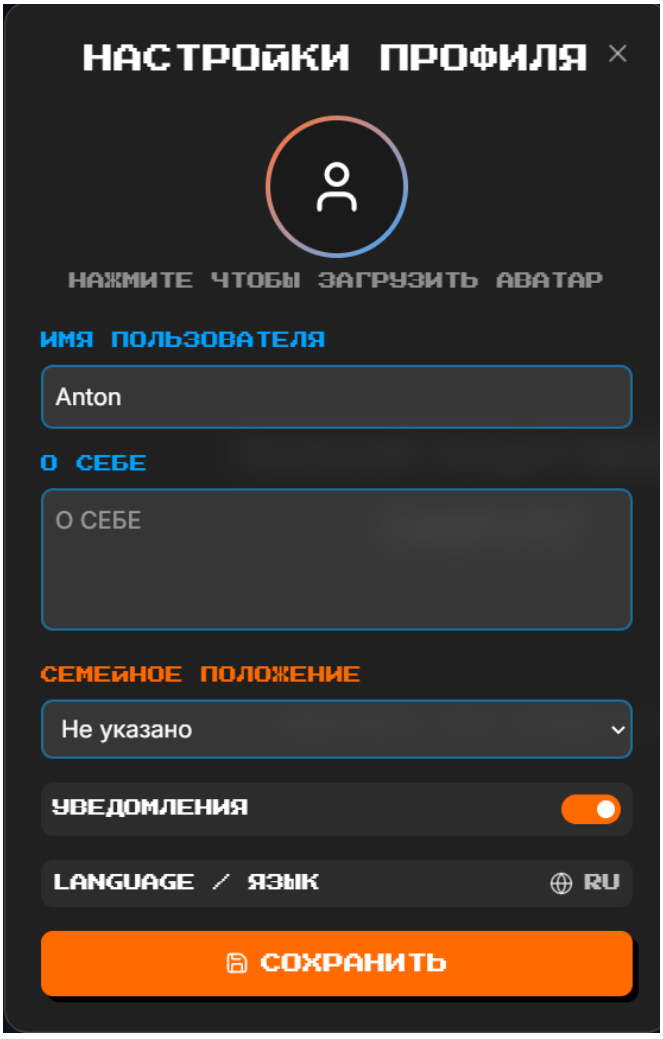


Рисунок 1.21 – Окно профиля пользователя

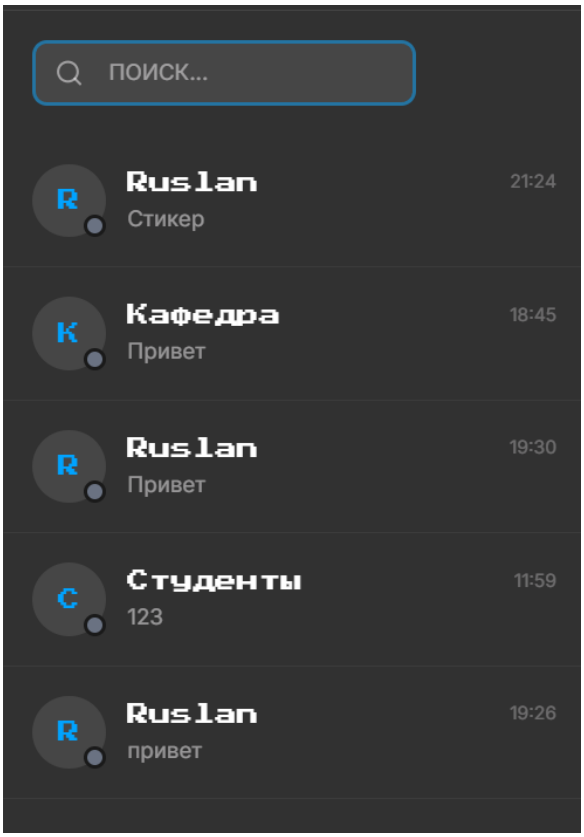


Рисунок 1.22 – Окно списка чатов

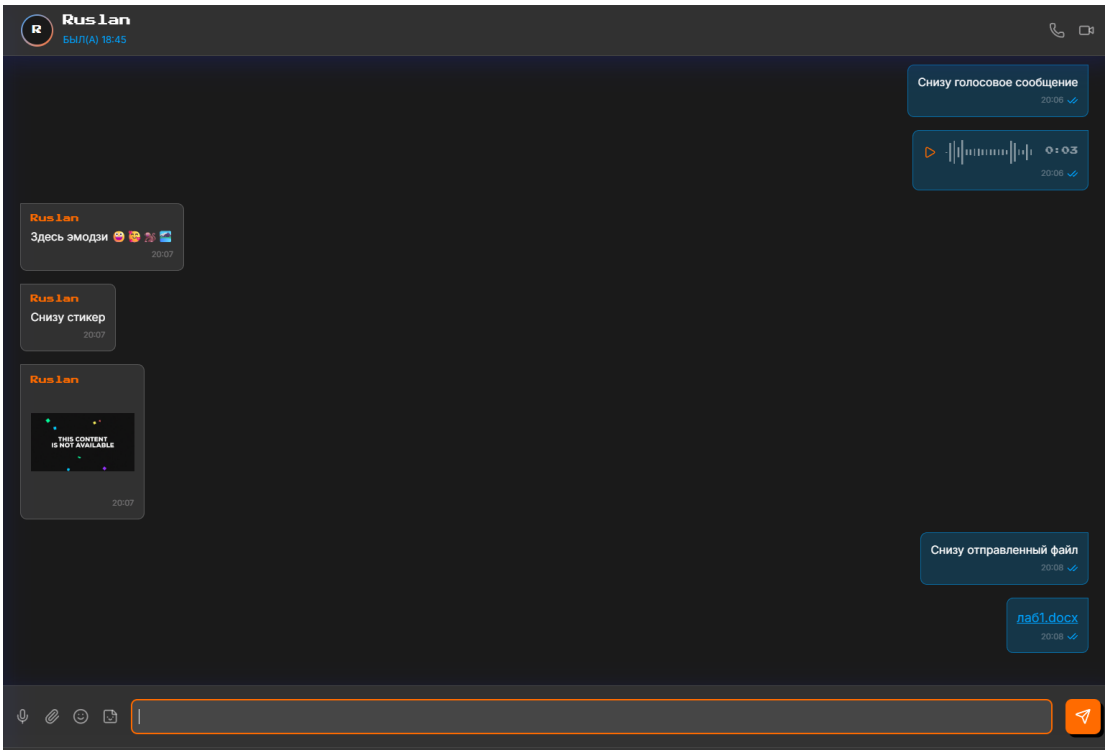


Рисунок 1.23 – Окно чата

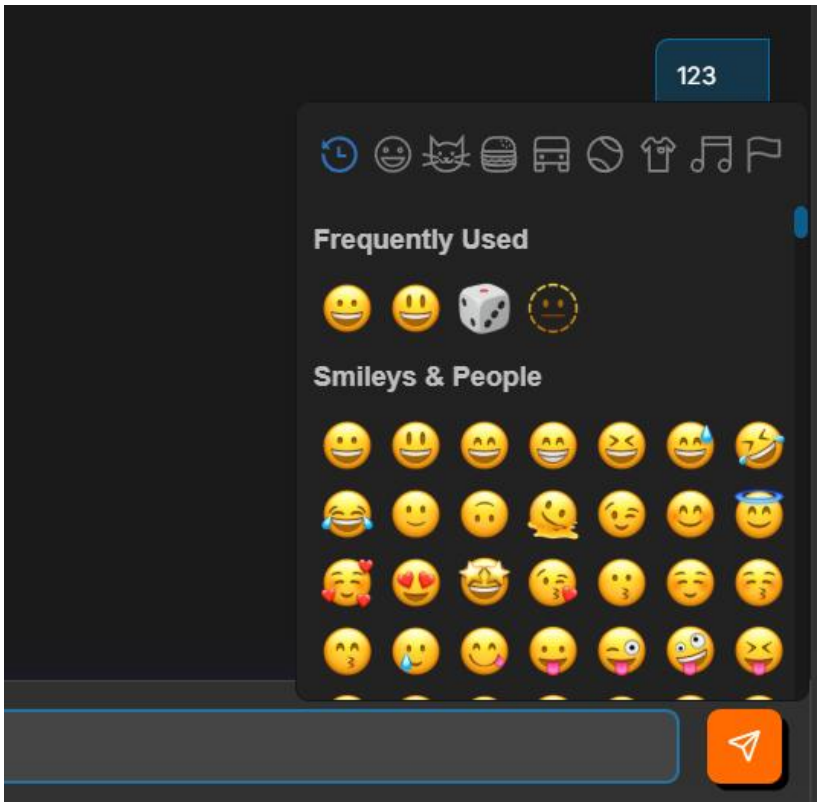


Рисунок 1.24 – Выбор эмодзи

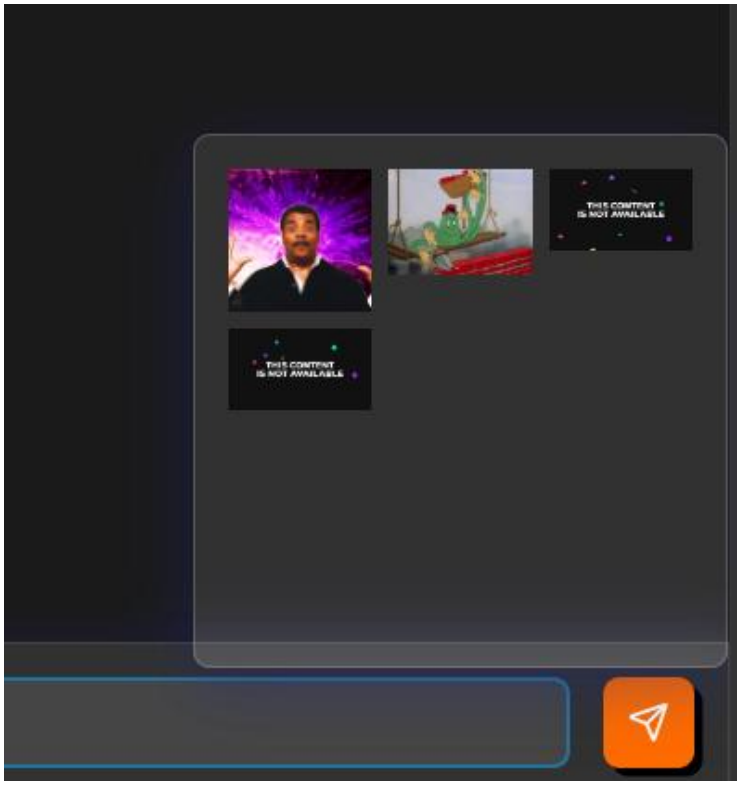


Рисунок 1.25 – Выбор стикера

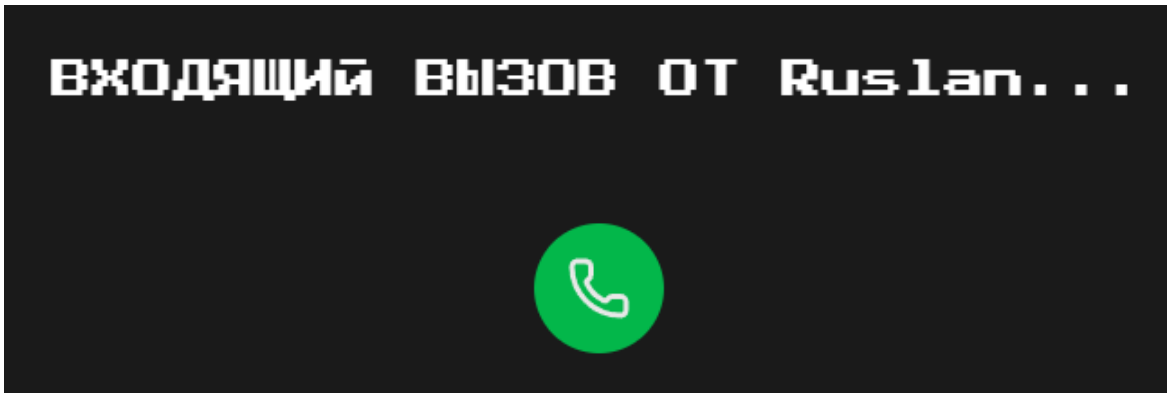


Рисунок 1.26 – Окно входящего вызова

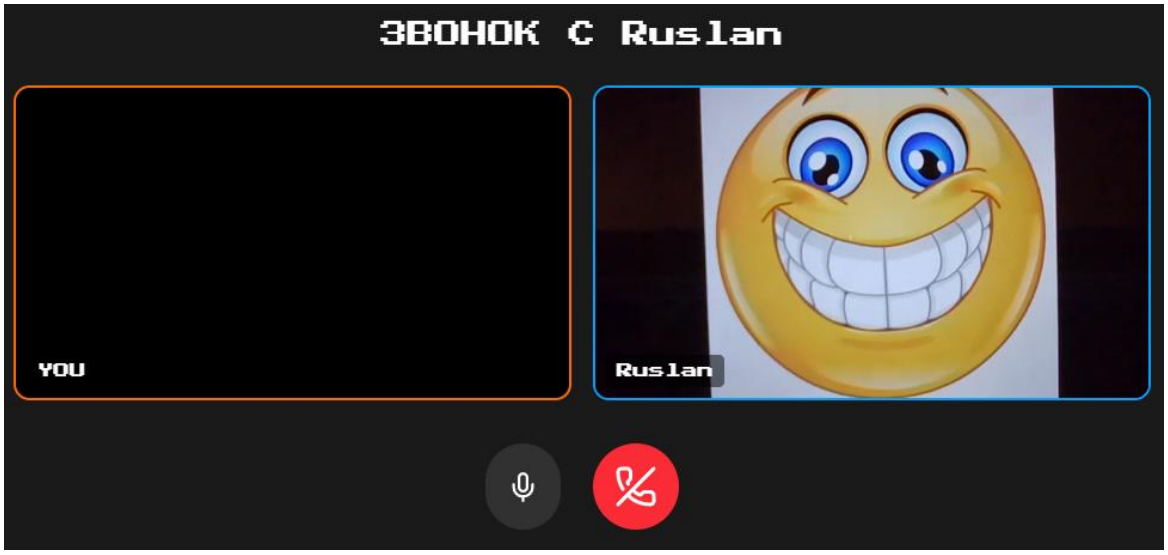


Рисунок 1.27 – Окно звонка

1.8.2 Разработка интерфейса администратора

Возможность администрирования мессенджера реализована через встроенный клиент Prisma Studio. Это позволяет легко и быстро взаимодействовать с данными пользователей, чатов и прочего в приложении. Возможность попасть на этот клиент есть исключительно у обладателя исходников приложения, обычным пользователем он недоступен. Это отличает мессенджер от обычного, делая его корпоративным. Графический интерфейс администратора показан на рисунках 1.28, 1.29.

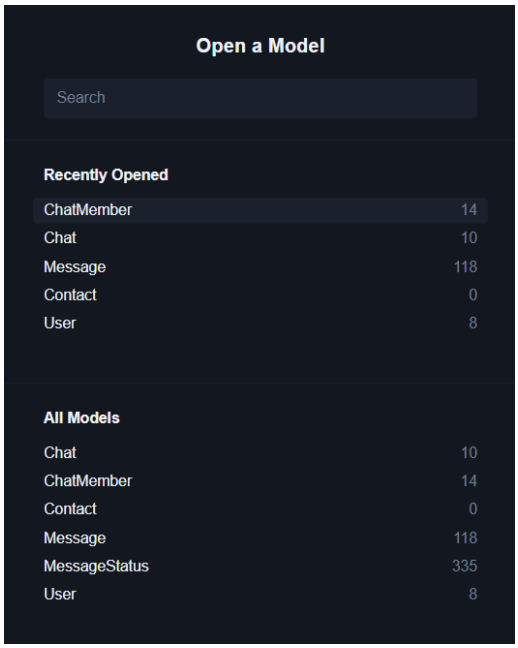


Рисунок 1.28 – Начальное окно

Рисунок 1.29 – Окно взаимодействия с пользователями

1.8.3 Разработка модулей

Для организации взаимодействия пользователя с приложением в процессе работы используются классы.

1.8.3.1 Класс Chat

Класс Chat является основным компонентом, который управляет состоянием чатов, сообщений и прочего взаимодействия с пользователем. Основные поля описаны ниже.

socket – хранит экземпляр WebSocket-соединения для обмена данными в реальном времени.

chats – содержит информацию о списке всех доступных чатов для пользователя.

activeChat – содержит информацию о текущем выбранном чате.

messages – содержит информацию о сообщениях, которые относятся к чату.

newMessage – содержит текст нового сообщения, которое вводит пользователь.

user – содержит данные текущего авторизованного пользователя.

searchQuery – содержит текущий текст, который ввёл пользователь в строку поиска.

searchResults – содержит результаты поиска пользователей или чатов.

showEmojiPicker – содержит информацию об отображении окна выбора эмодзи.

showStickerPicker – содержит информацию об отображении окна выбора стикеров.

showProfileSettings – содержит информацию об отображении окна настроек профиля.

selectedUserProfile – содержит данные профиля выбранного пользователя.

fileInputRef – содержит ссылку на файл для загрузки.

stream – хранит поток мультимедиа для звонков.

receivingCall – содержит информацию о принятии пользователем входящего звонка.

caller – содержит идентификатор пользователя, совершающего звонок.

callerSignal – содержит данные сигнала для установления соединения при звонке.

callAccepted – содержит информацию об успешном принятии текущего звонка.

callEnded – содержит информацию о завершении текущего звонка.

name – показывает имя текущего пользователя.

showCallModal – содержит информацию об отображении окна звонка.

isVideoCall – показывает, является ли звонок видеозвонком.

1.8.3.2 Класс Message

Класс Message представляет сообщения в чате и содержит информацию его содержимом, типе, отправителе и времени создания. Основные поля описаны ниже.

id – уникальный идентификатор сообщения.

content – содержит текст сообщения.

type – содержит информацию о типе сообщения (текст, изображение, файл и прочее).

fileUrl – содержит URL загруженного файла при наличии вложения в сообщении.

senderId – идентификатор пользователя, который отправил сообщение.

chatId – идентификатор чата, к которому относится сообщение.

createdAt – содержит время создания сообщения.

sender – содержит информацию об отправителе сообщения.

statuses – содержит список статусов сообщения (отправлено, доставлено, прочитано).

1.8.3.3 Класс ChatDetails

Этот класс включает в себя данные о чате: члены чаты, сообщения, количество непрочитанных сообщений. Основные поля описаны ниже.

id – уникальный идентификатор чата.

name – содержит в себе название чата.

chatMembers – содержит список участников чата с их данными.

messages – показывает список сообщений, которые связаны с конкретным чатом.

unreadCount – содержит в себе количество непрочитанных сообщений.

UML-диаграмма классов представлена на рисунке 1.30.

Рисунок 1.30 – UML-диаграмма взаимодействия классов

1.8.3.4 Класс Login

Данный класс является основным компонентом страницы авторизации пользователя. Основные поля описаны ниже.

email – содержит введённый пользователем email.

password – хранит в себе введённый пользователем пароль.

handleSubmit – используется для выполнения запроса на сервер для авторизации пользователя.

1.8.3.5 Класс Form

Данный класс является формой для входа, которая содержит элементы ввода для email и пароля, а также кнопку для отправки. Основные поля отображены ниже.

emailInput – является полем для ввода email.

passwordInput – поле для ввода пароля.

submitButton – представляет собой кнопку для отправки формы на сервер.

1.8.3.6 Класс Navigation

Данный класс содержит в себе хук useNavigate, который используется для перенаправления пользователя на страницу чата при успешном входе.

UML-диаграмма классов представлена на рисунке 1.31.

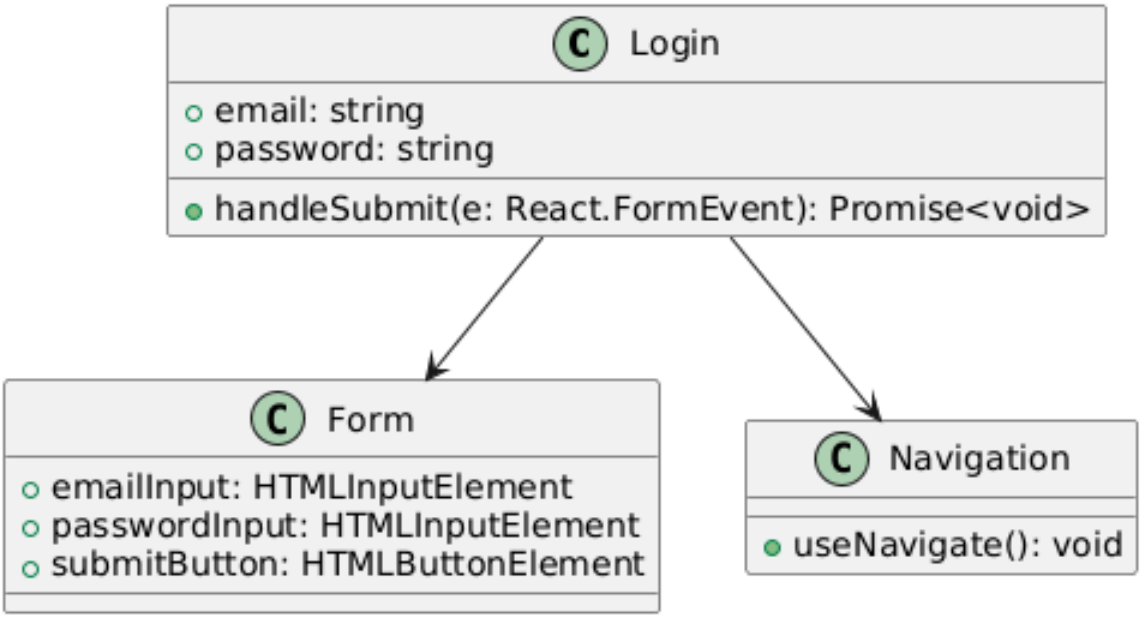


Рисунок 1.31 – UML-диаграмма взаимодействия классов

1.8.3.7 Класс Profile

Класс Profile является основным компонентом страницы профиля. Основные поля описаны ниже.

user – используется для хранения данных текущего пользователя.

handleUpdate – используется для обновления данных пользователя в базе данных.

1.8.3.8 Класс ProfileSettings

Данный класс является компонентом для отображения и редактирования настроек профиля. Основные поля описаны ниже.

user – хранит данные текущего пользователя.

onClose – функция для закрытия страницы настроек профиля.

onUpdate – функция для обновления данных пользователя.

isPage – указывает на использование компонента как отдельную страницу.

1.8.3.9 Класс LocalStorage

Данный класс используется для хранения и извлечения данных пользователя. Основные поля описаны ниже.

gettem – метод, который используется для получения данных из базы данных по ключу.

settem – метод, который используется для сохранения данных в базе данных по ключу.

removeItem – метод, который используется для удаления данных из базы данных по ключу.

UML-диаграмма классов представлена на рисунке 1.32.

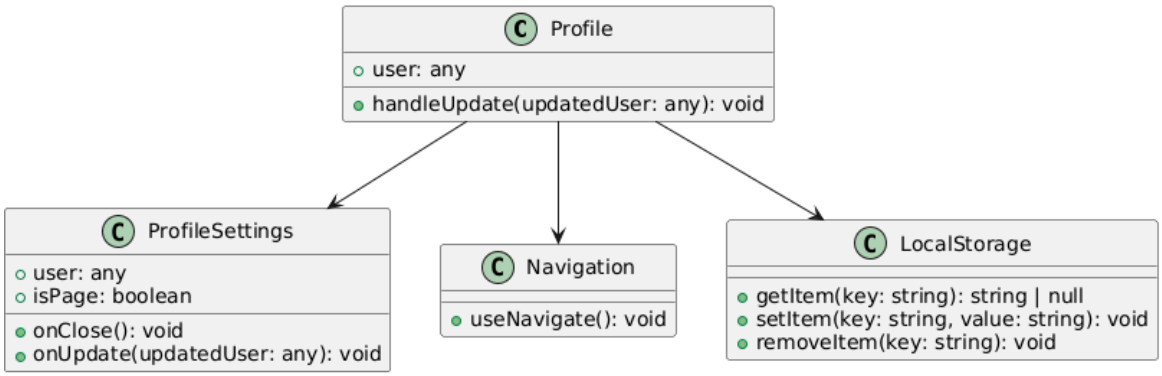


Рисунок 1.32 – UML-диаграмма взаимодействия классов

Чтобы наглядно увидеть последовательность действий была сделана UML-диаграмма последовательности. Это позволит понять, как взаимодействуют с пользователем разработанные окна. UML-диаграмма последовательности представлена на рисунке 1.33.

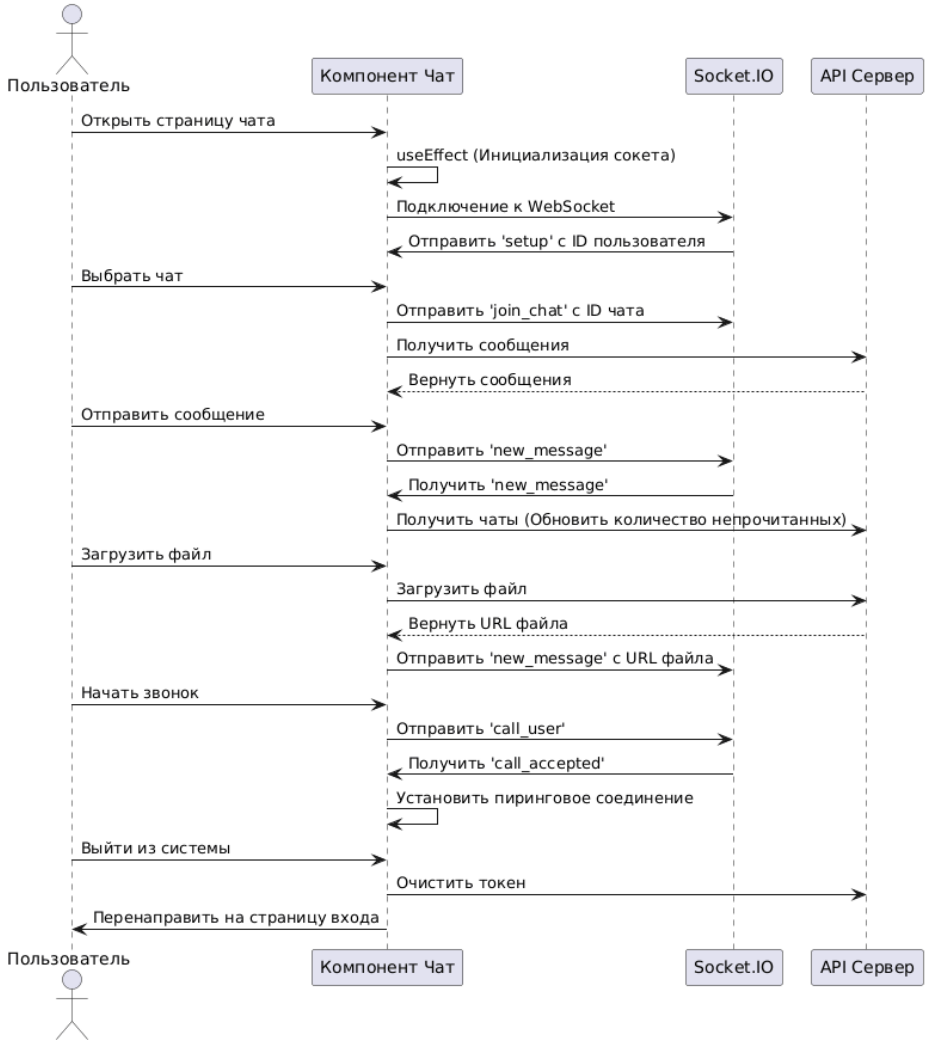


Рисунок 1.33 – UML-диаграмма последовательности

1.8.4 Разработка базы данных

Для разработки базы данных используется Prisma ORM, которая позволяет в удобном варианте взаимодействовать с базой данных через декларативное описание схемы и автоматическую генерацию SQL-запросов.

Для описания моделей используется файл schema.prisma. В нём модели определяют структуру таблиц, их поля, типы данных и связи между таблицами.

Для фиксации изменений используются миграции, которые являются набором SQL-запросов. Это позволяет в удобном варианте создавать или изменять таблицы в имеющейся базе данных.

1.9 Тестирование приложения

Во время использования приложения пользователи могут допускать различные ошибки, из-за которых возможна некорректная работа. Такие ошибки необходимо обрабатывать и уведомлять о них пользователя. Для проверки работоспособности обработчиков был составлены тест-кейсы, показанные в таблице 1.

Таблица 1 – Тест-кейсы

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
Проверка наличия пустых полей при авторизации	Для входа нужно заполнить поля «Логин» и «Пароль»	Запустить приложение. Не заполнять поля для ввода.	На экране отобразилось уведомление о необходимости ввести логин и пароль
Ввод несуществующих данных при авторизации	При авторизации необходимо ввести существующие данные от аккаунта	Запустить приложение и выполнить вход. В поле «Логин» или «Пароль» ввести неверные данные пользователя.	На экране отобразилось уведомление о несуществующем аккаунте
Отправка пустого сообщения	В чат нельзя отправить сообщение без содержания	Запустить приложение и выполнить вход. Перейти в чат. В поле ввода ничего не писать. Нажать на кнопку отправки сообщения.	Сообщение не отправляется
Поиск несуществующего пользователя	Поиск пользователей происходит по совпадению в имени	Запустить приложение и выполнить вход. Нажать на строку поиска. В поле поиска ввести имя несуществующего пользователя.	В списке не должно отобразиться ни одного пользователя

Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
Попытка скачивания несуществующего файла	Файл должен быть загружен на сервере для успешного скачивания	Запустить приложение и выполнить вход. Выбрать любой из чатов. В сообщениях найти файл, удалённый с сервера. Попытаться его установить.	На экране отобразится уведомление «Файл не найден»
Попытка записи голосового без наличия разрешений	Для записи голосового сообщения должно быть системное разрешение на использование микрофона	Запустить приложение и выполнить вход. Выбрать любой из чатов. Нажать на кнопку записи голосового сообщения. Сказать что-то в микрофон.	Голосовое сообщение не содержит никакую информацию
Попытка совершения аудиозвонка без наличия разрешений	Для совершения аудиозвонка должно быть системное разрешение на использование микрофона	Запустить приложение и выполнить вход. Выбрать любой из чатов. Нажать на кнопку аудиозвонка. Дождаться его принятия от пользователя. Начать разговор.	Собеседник не слышит пользователя
Попытка совершения видеозвонка без наличия разрешений	Для совершения видеозвонка должно быть системное разрешение на использование микрофона и камеры	Запустить приложение и выполнить вход. Выбрать любой из чатов. Нажать на кнопку видеозвонка. Дождаться его принятия от другого пользователя.	Веб-камера не показывает ничего, голос не слышно

1.10 Описание приложения

1.10.1 Руководство пользователя

После запуска клиента на странице браузера открывается загрузочное окно приложения. Оно содержит в себе логотип Пензенского Государственного университета. При правильной работе приложения загрузочное окно со временем пропадёт, и пользователь попадёт на следующее окно. Загрузочное окно представлено на рисунке 1.34.

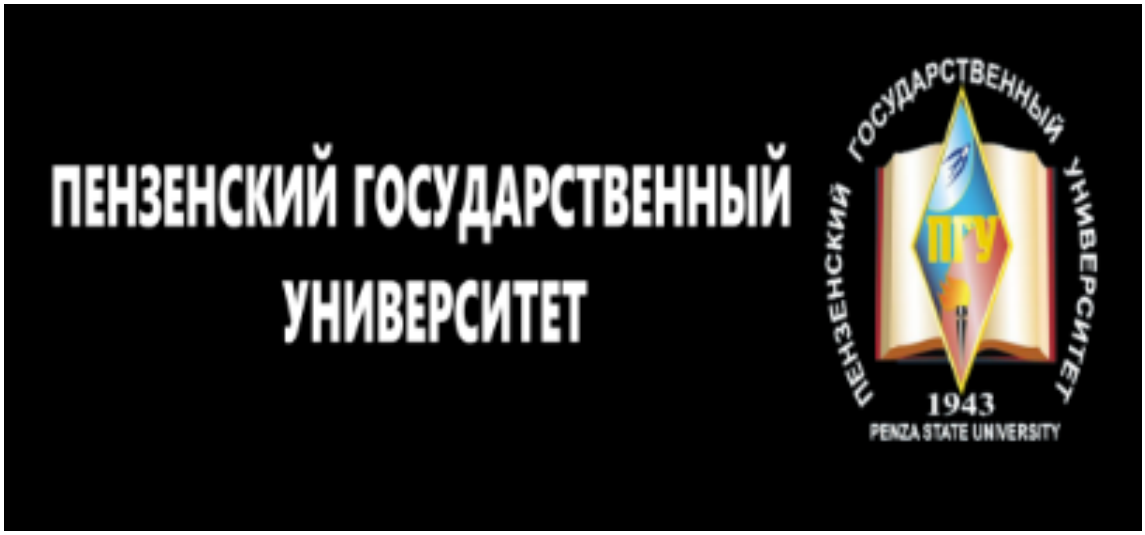


Рисунок 1.34 – Загрузочное окно

При успешной загрузке пользователь попадает на окно авторизации, где может ввести логин и пароль для своего аккаунта, которые ему ранее задал администратор. При повторном входе в клиент повторная авторизация не требуется, и пользователь сразу попадает на главный экран. При попытке ввести неверные или несуществующие данные пользователь получит уведомление о неверном действии. Пока не будут введены существующие данные от аккаунта дальше не получится зайти. Окно авторизации показано на рисунке 1.35.

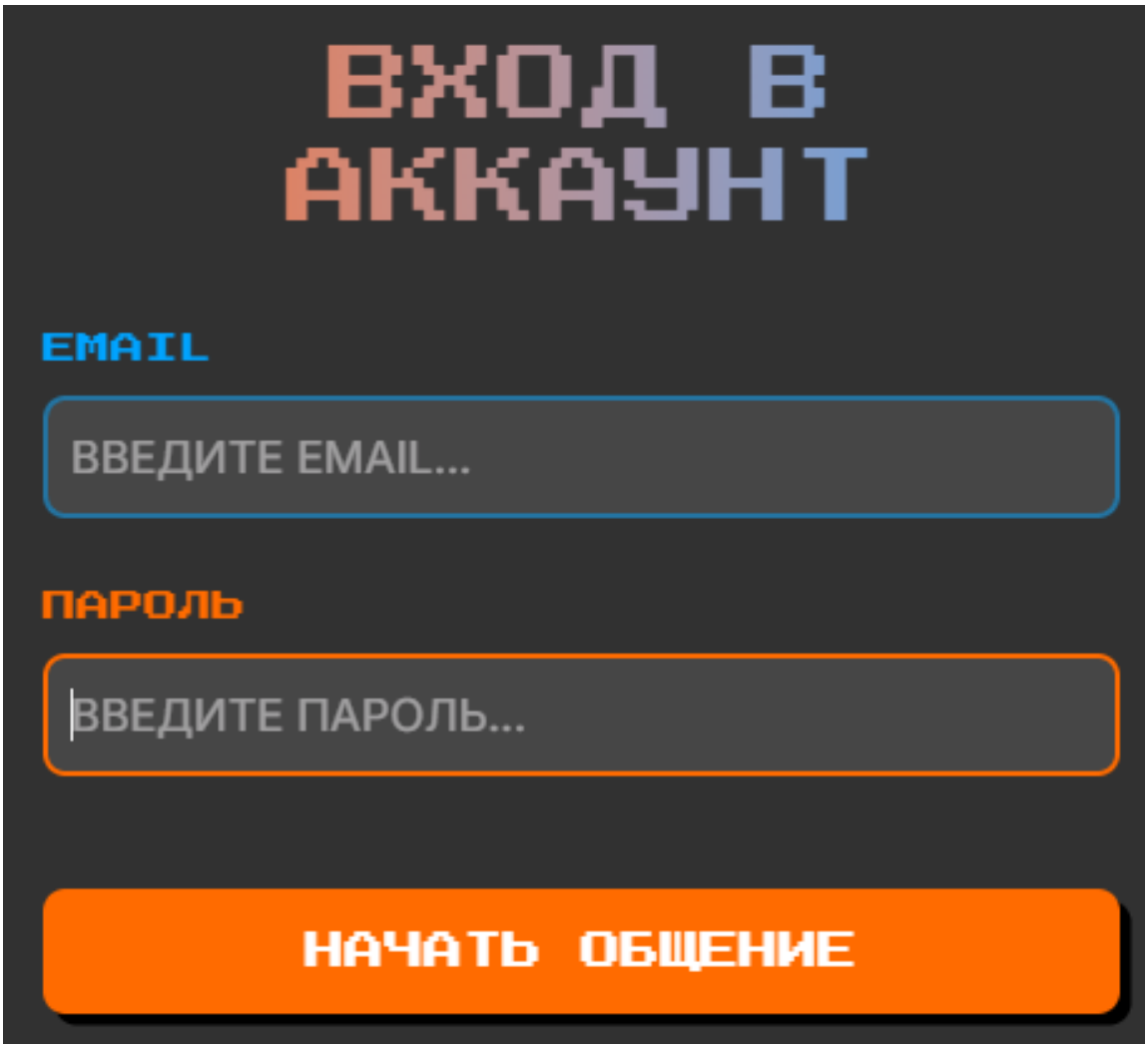


Рисунок 1.35 – Окно авторизации

Если логин и пароль введены верно, то после нажатия на кнопку «Начать общение» пользователь попадает на главное окно приложения, который содержит список чатов, окно чата, строку поиска, имя и аватар профиля. В дальнейшем будет возможность сменить аккаунт при необходимости. Главное окно представлено на рисунке 1.36.

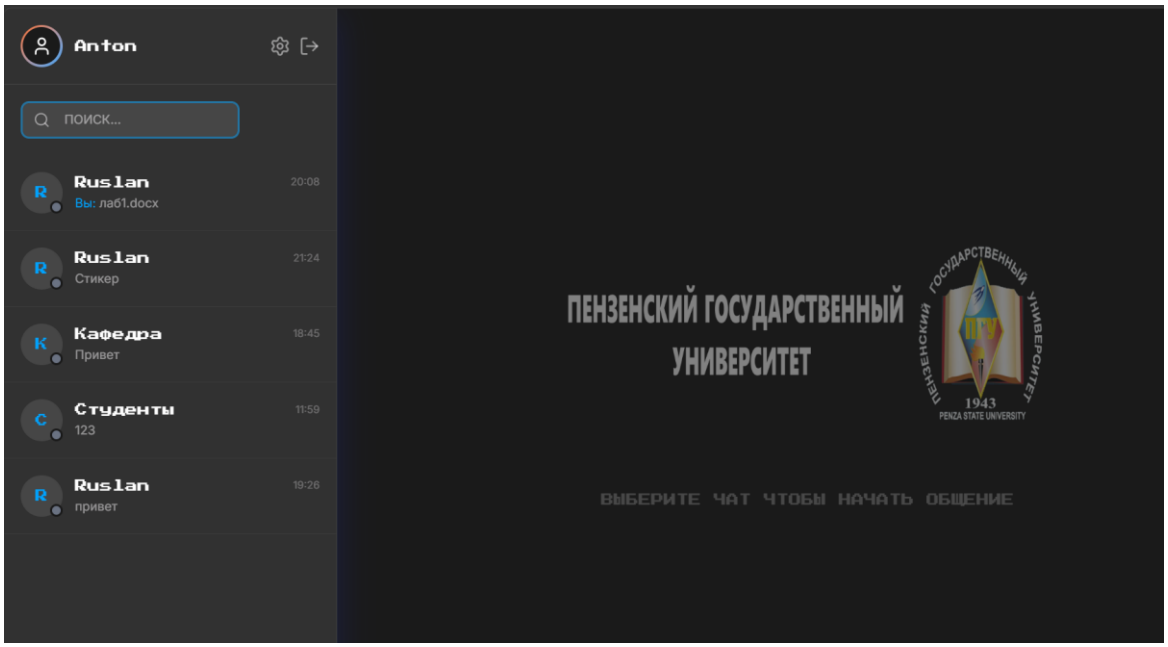


Рисунок 1.36 – Главное окно приложения

Пользователь имеет возможность выбрать любой чат из доступных, либо выполнить поиск при его отсутствии. При вводе вылезают подсказки, которые показывают зарегистрированных пользователей и позволяют выбрать любого из них. Строка поиска показана на рисунке 1.37.

Рисунок 1.37 – Строка поиска

По нажатию на свой аватар пользователь попадает на страницу настройки профиля, где можно кастомизировать его или сделать, например, описание. Это показано на рисунках 1.38, 1.39.

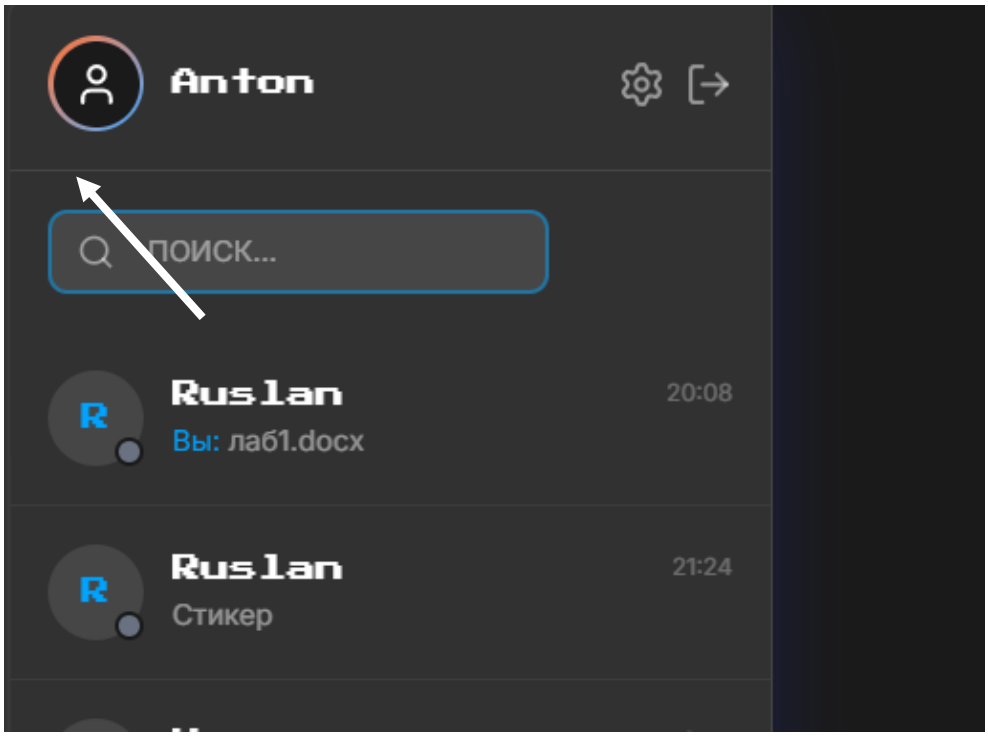


Рисунок 1.38 – Аватар пользователя

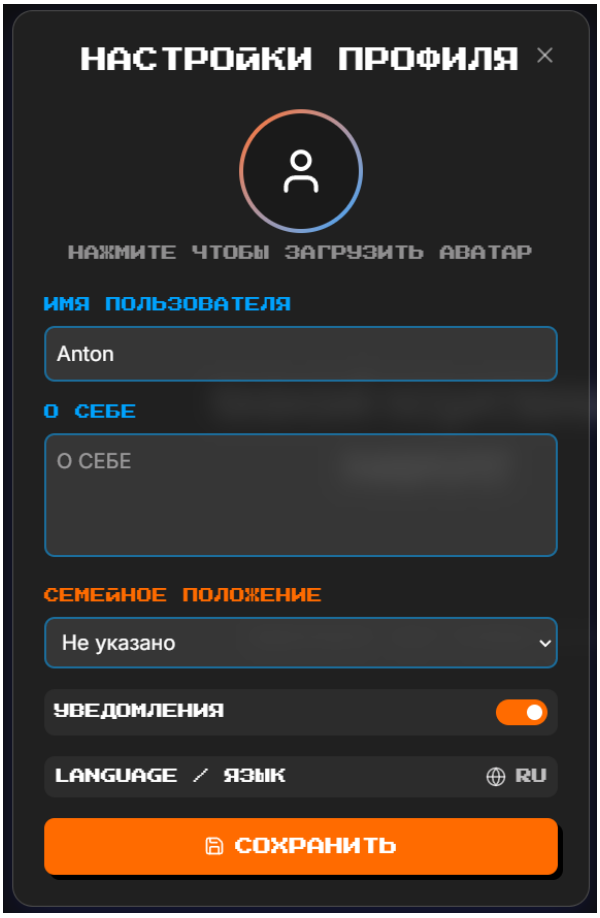


Рисунок 1.39 – Окно настройки профиля

В окне настройки можно нажать на аватар и изменить его, выбрав картинку из файлов. Изменение аватара показано на рисунках 1.40 – 1.42.

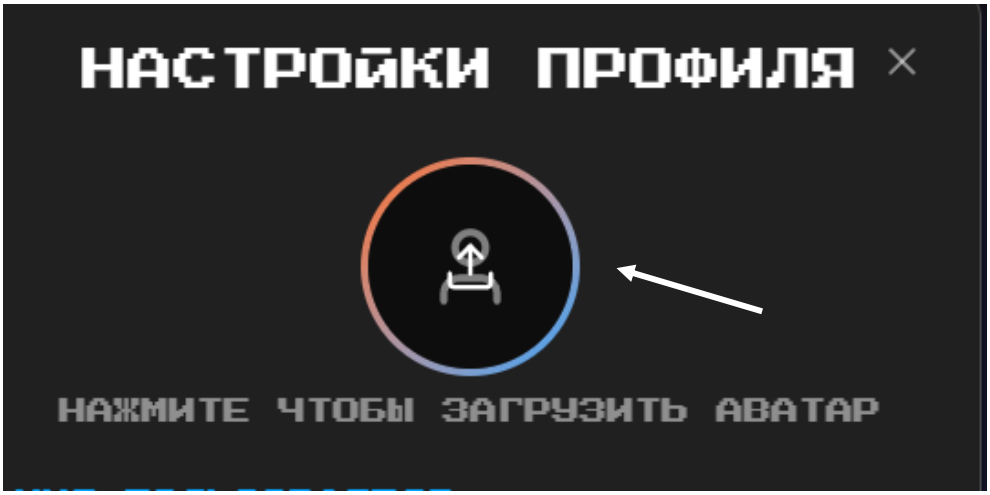


Рисунок 1.40 – Изменение аватара пользователя

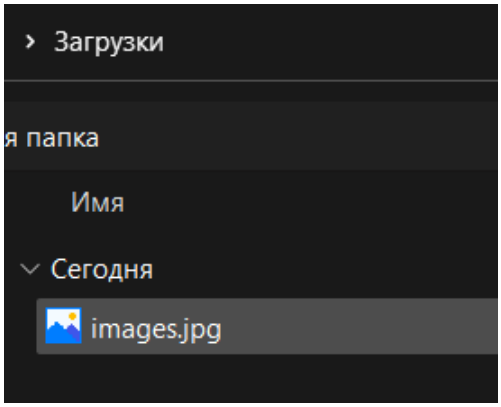


Рисунок 1.41 – Выбор файла для аватара

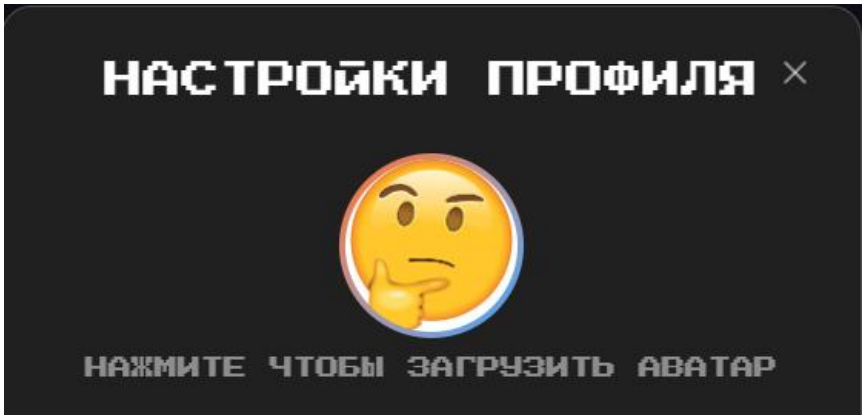


Рисунок 1.42 – Изменённый аватар

Также в окне настройки профиля есть возможность изменить своё имя или добавить информацию о себе. Поля для этого показаны на рисунке 1.43.

Рисунок 1.43 – Поля для изменения данных пользователя

По желанию пользователя можно включить или выключить уведомления, а также сменить язык интерфейса на английский. Это показано на рисунках 1.44, 1.45.

Рисунок 1.44 – Поле включения/выключения и уведомлений и смены языка

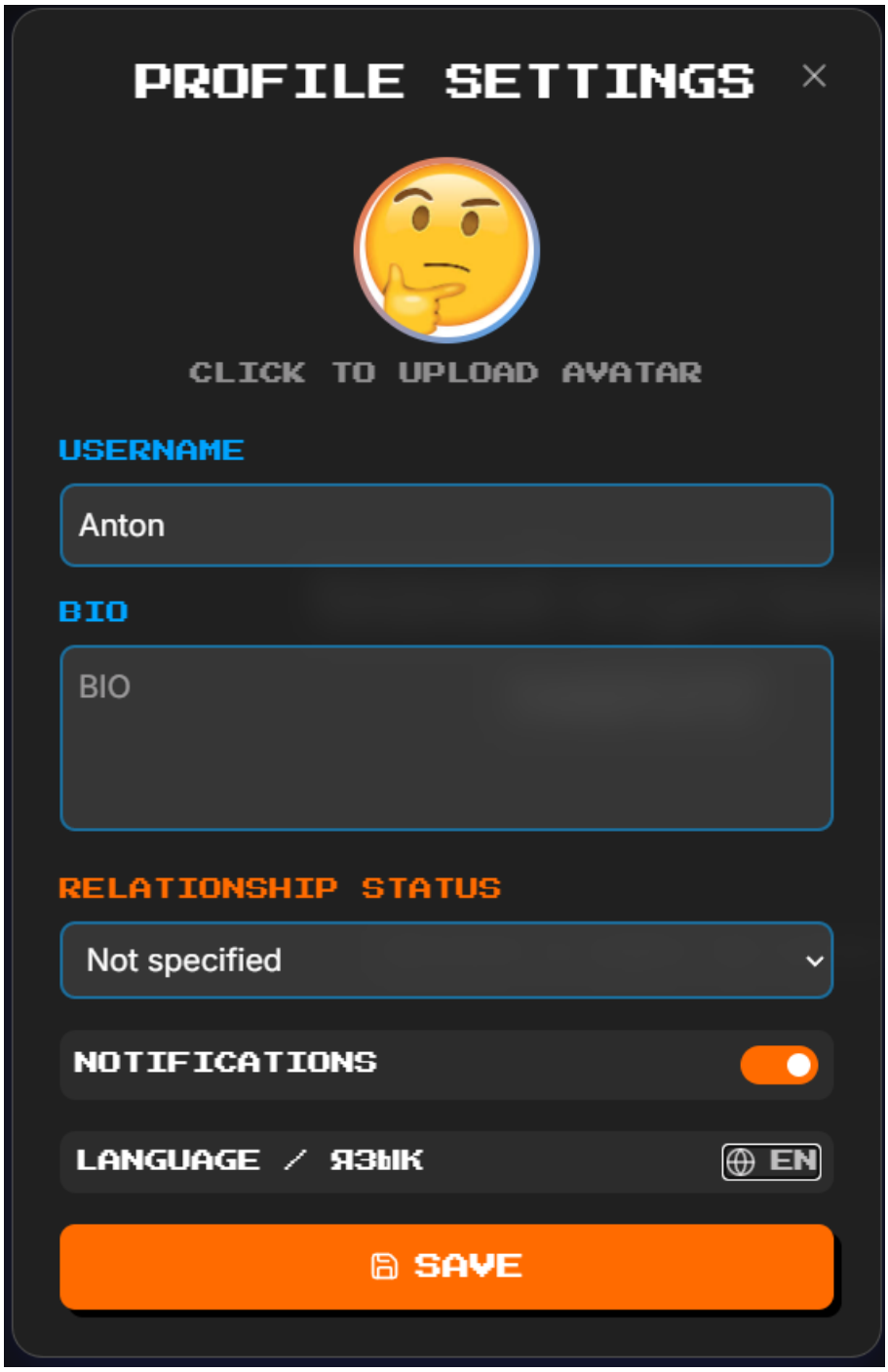


Рисунок 1.45 – Интерфейс после смены языка

Выбрав нужный чат, пользователь может отправить текстовое или голосовое сообщение, прикрепить эмодзи или стикер, а также отправить файл. Это показано на рисунках 1.46 – 1.48.

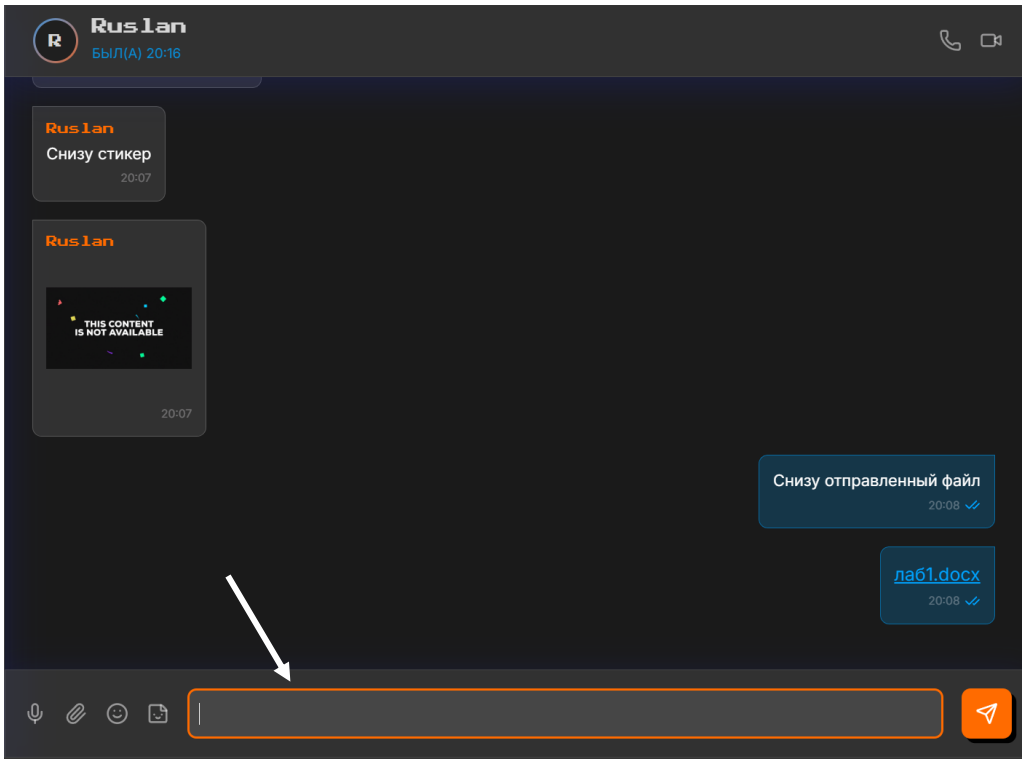


Рисунок 1.46 – Поле для ввода текстового сообщения

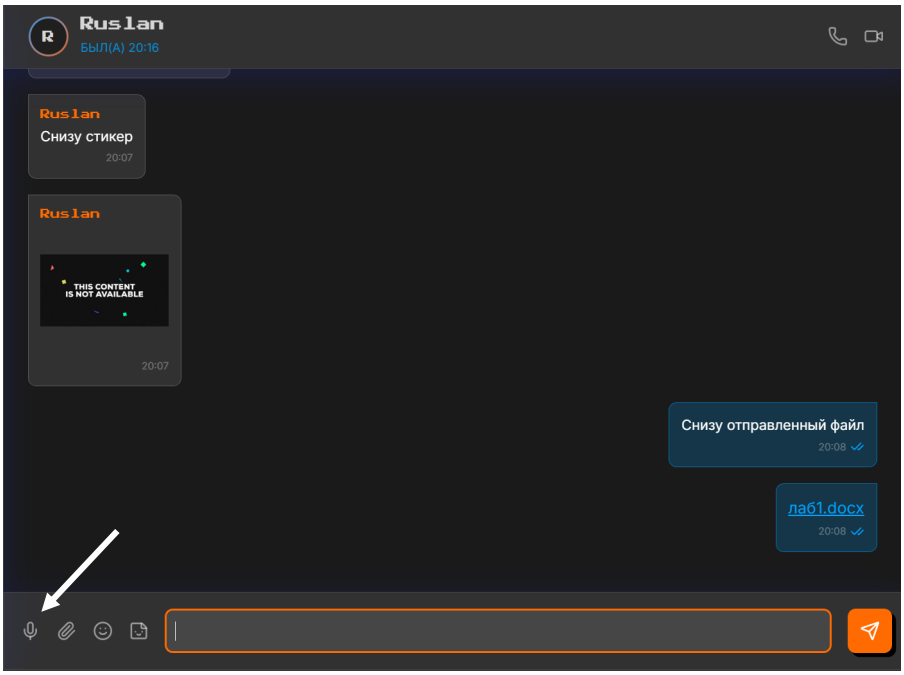


Рисунок 1.47 – Кнопка отправки голосового сообщения

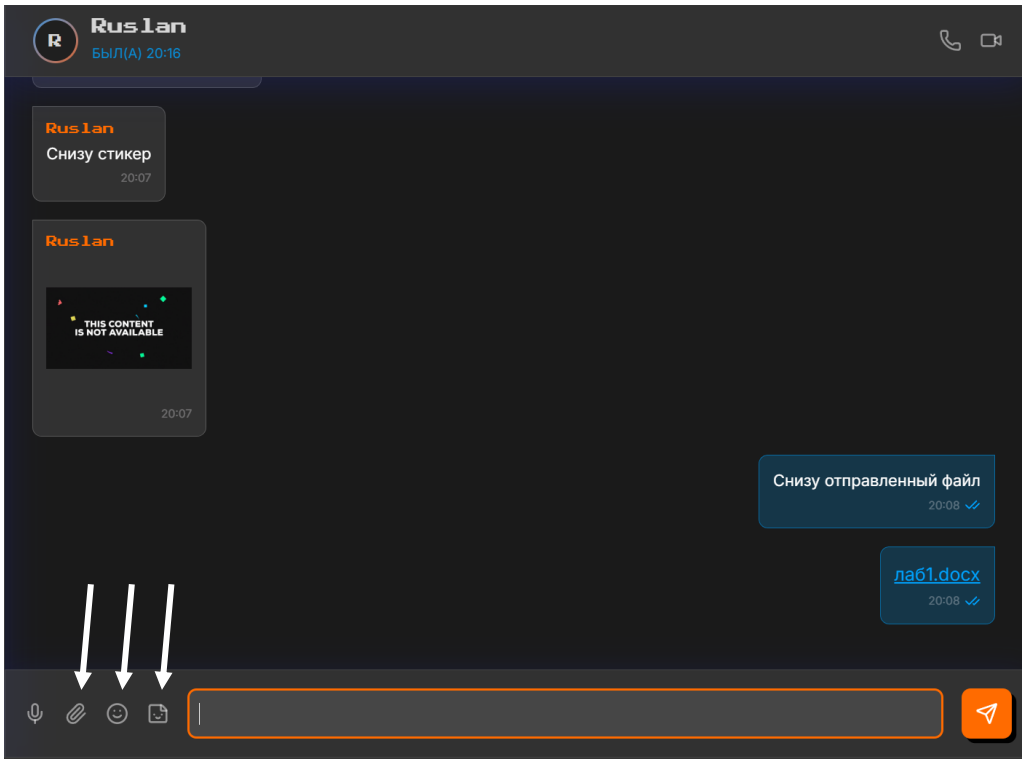


Рисунок 1.48 – Кнопки для отправки файла, эмодзи или стикера

Если пользователь хочет скачать отправленный файл, то ему необходимо нажать на название файла в чате, после чего начнётся загрузка. Это отображено на рисунках 1.49, 1.50.

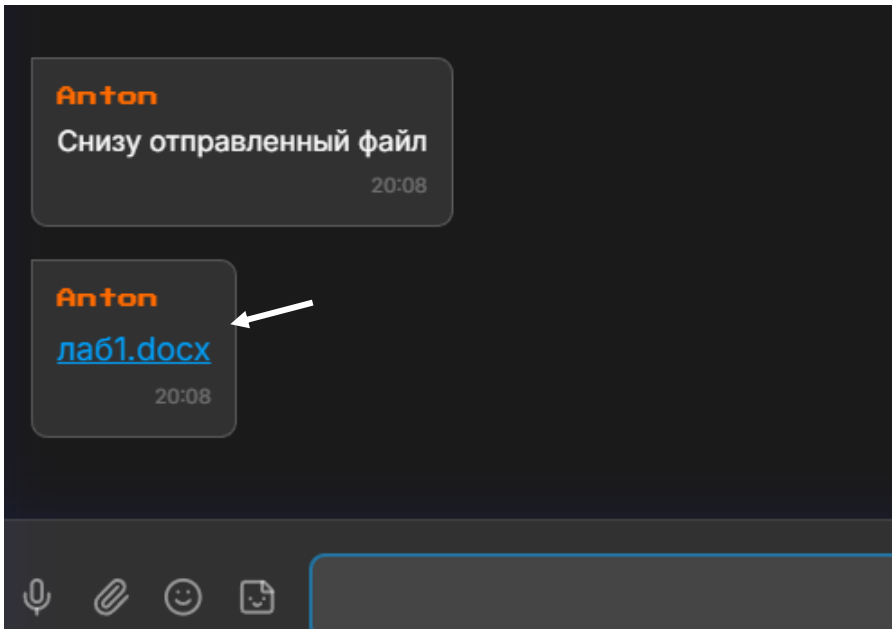


Рисунок 1.49 – Файл в чате

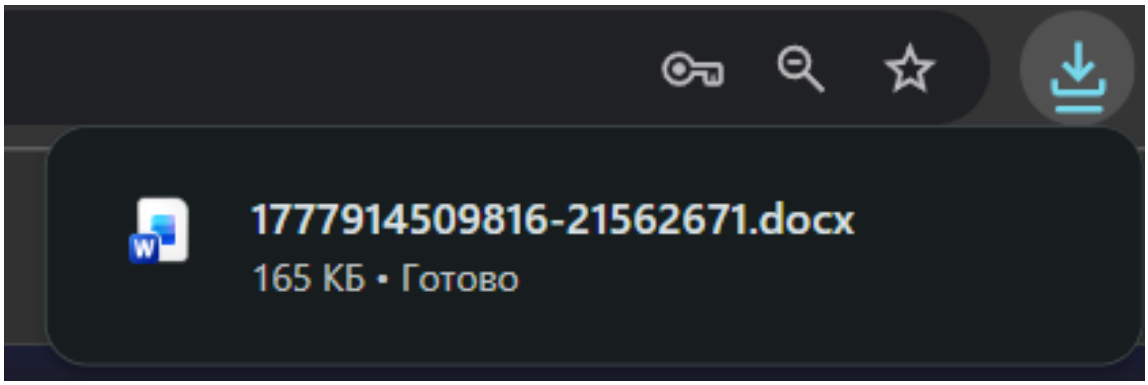


Рисунок 1.50 – Файл в загрузках

Для совершения аудио- и видеозвонков в интерфейсе предусмотрены соответствующие кнопки. Они отображены на рисунке 1.51.

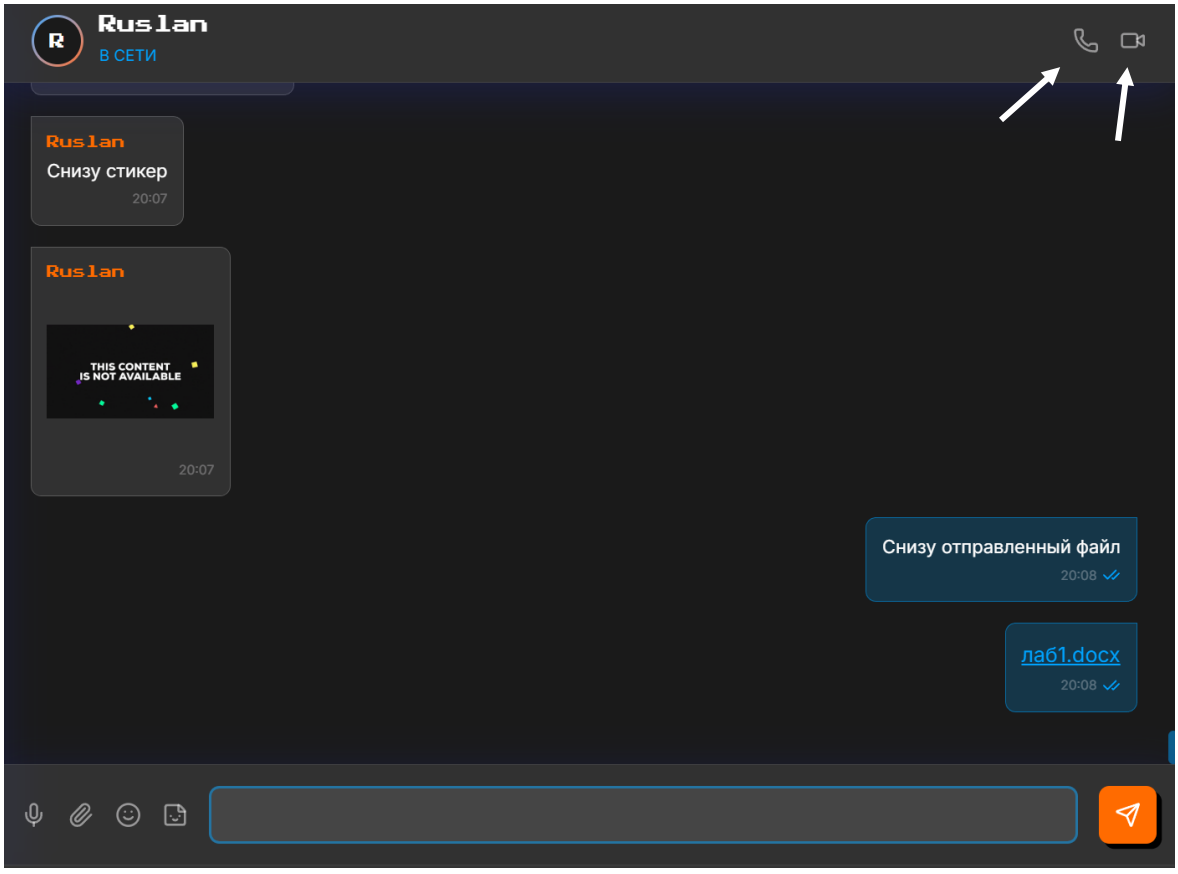


Рисунок 1.51 – Кнопки аудио- и видеозвонка

После нажатия одной из этих кнопок пользователь увидит окно ожидания ответа от другого пользователя, который в свою очередь увидит

окно входящего звонка. Чтобы его принять нужно нажать на зелёную кнопку. Это отображено на рисунках 1.52, 1.53.

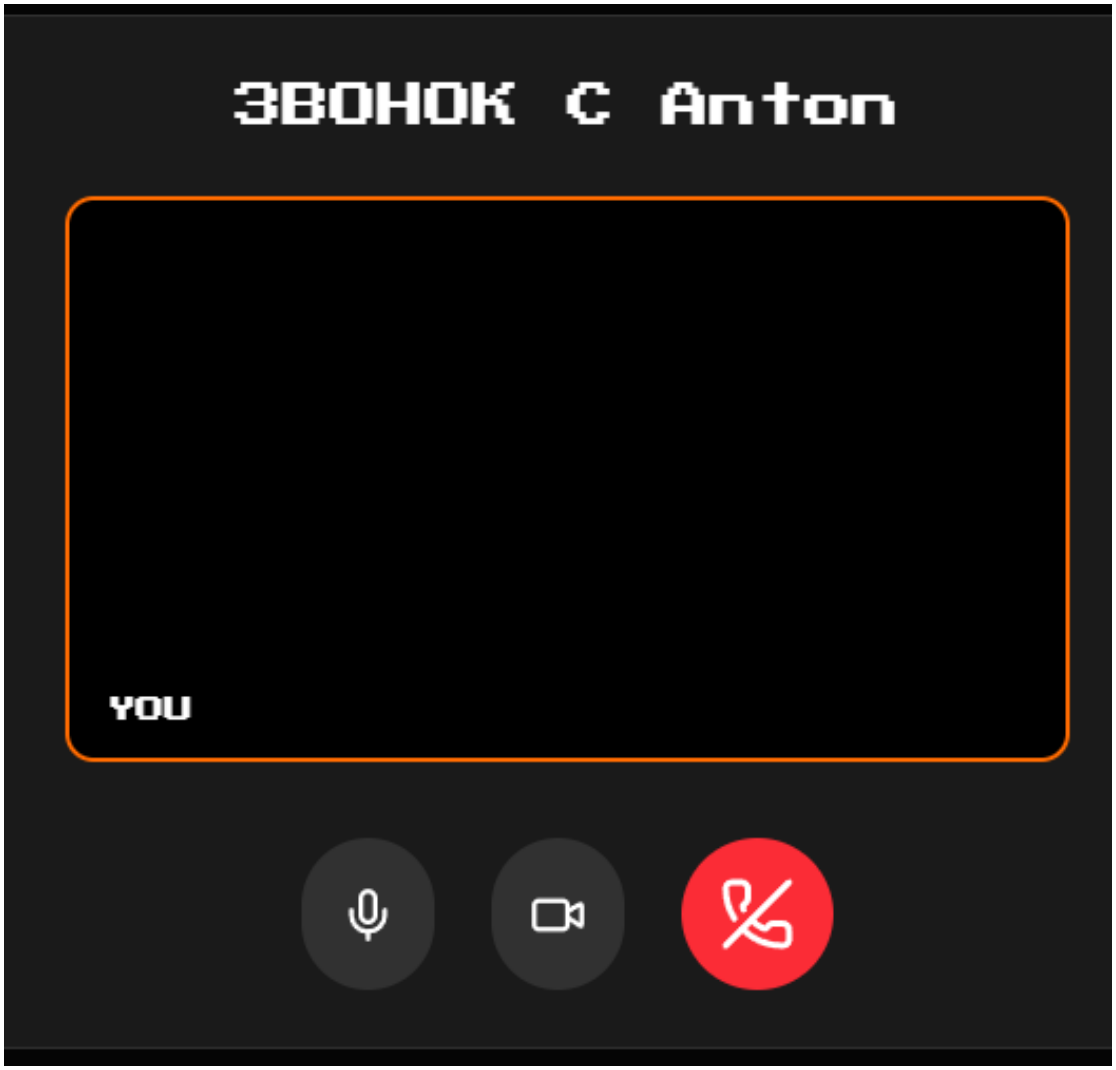


Рисунок 1.52 – Окно ожидания ответа

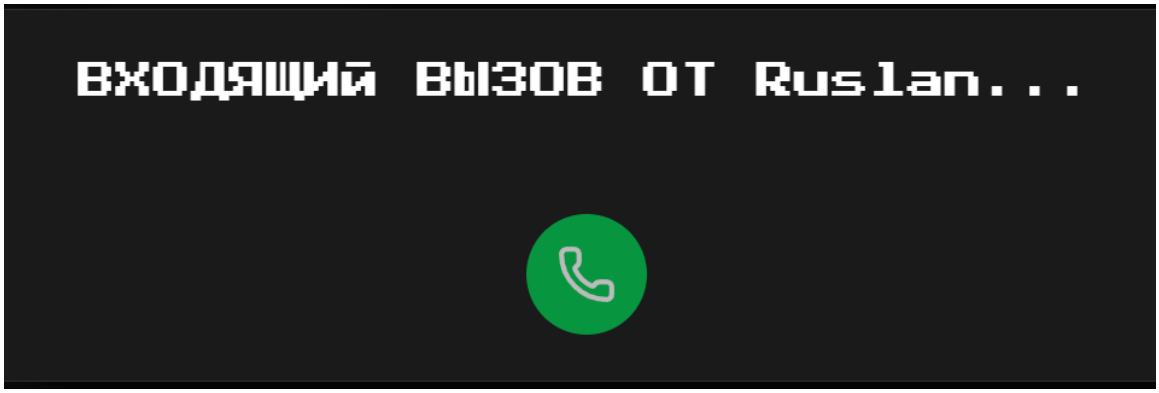


Рисунок 1.53 – Окно входящего звонка

Если звонок был принят, то оба пользователя увидят соответствующее окно, где можно включить/выключить микрофон, камеру или завершить звонок. Эти кнопки показаны на рисунке 1.54.

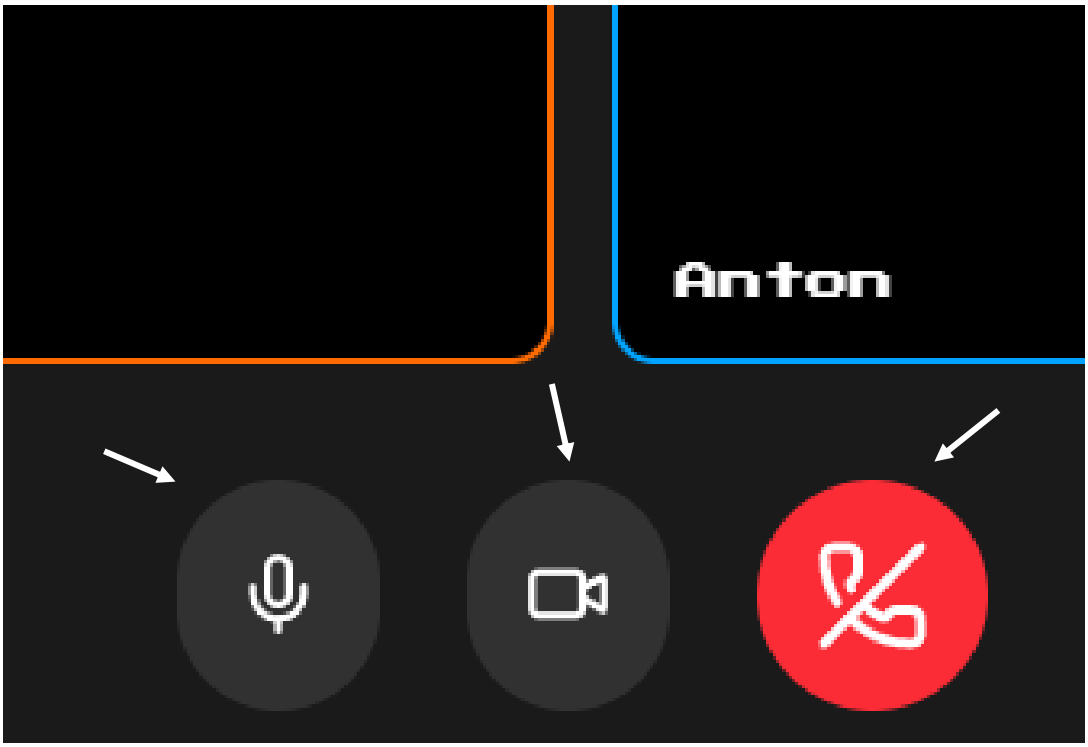


Рисунок 1.54 – Кнопки управления звонком

При наличии соответствующих разрешений будет работать и микрофон, и веб-камера. Это показано на рисунке 1.55.

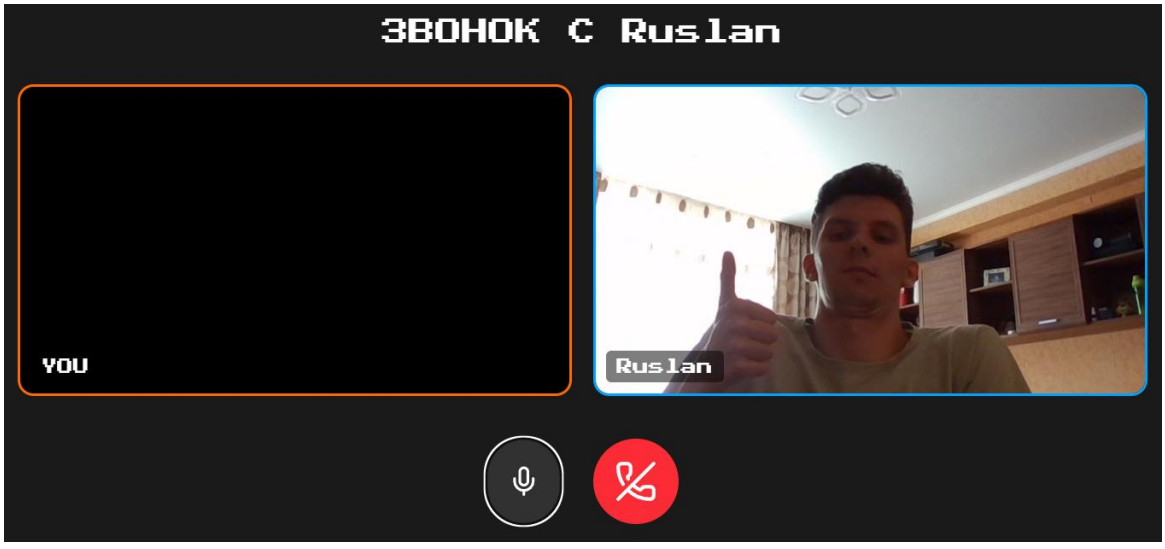


Рисунок 1.55 – Работающая веб-камера

При добавлении администратором в групповой чат пользователь увидит его в списке чатов и сможет общаться в нём с другими добавленными пользователями. Это показано на рисунке 1.56.

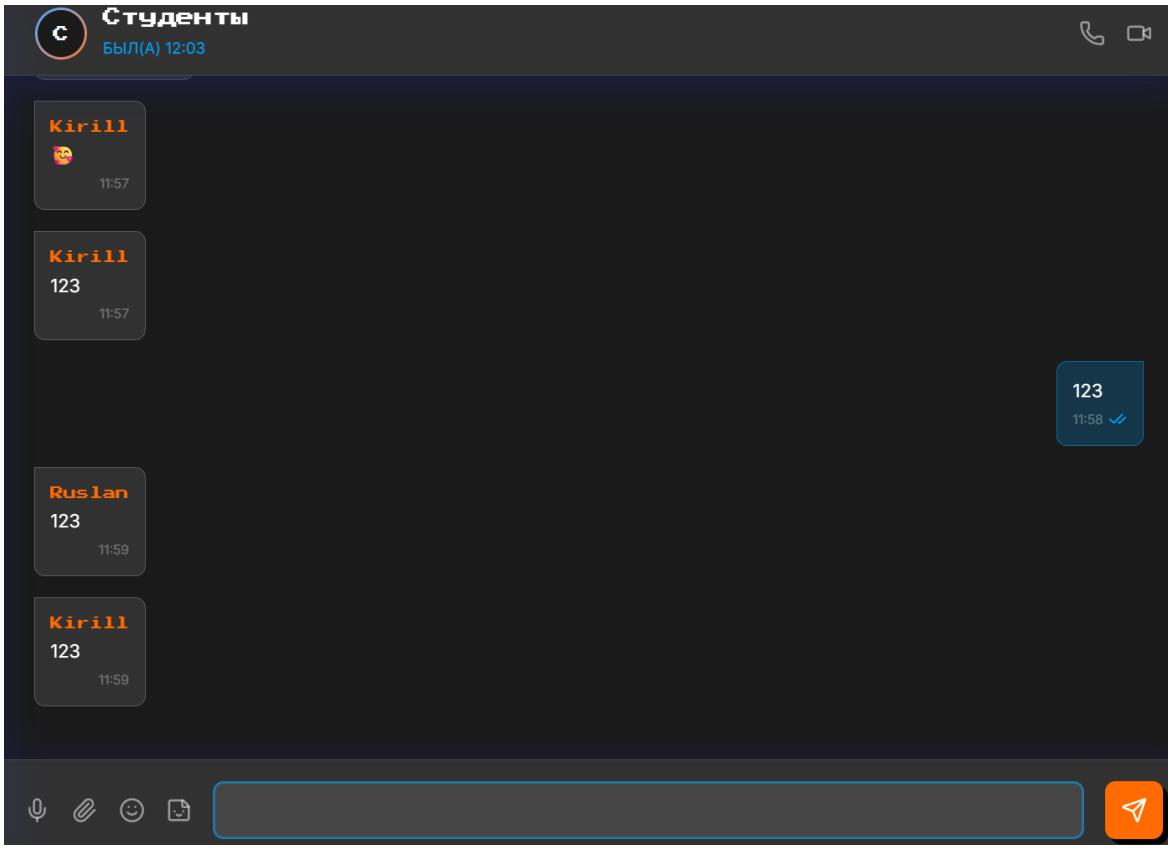


Рисунок 1.56 – Групповой чат

2 Охрана труда

2.1 Анализ негативных факторов, воздействующих на оператора ПЭВМ и способы их устранения

Работа с вычислительной техникой происходит в условиях ограниченной подвижности, что приводит к дистрофии мышц от нахождения в статическом напряжении. Также пользователь подвержен перенапряжению зрительных органов. Воздействие всего этого на человека приводит к ряду неблагоприятных факторов, которые снижают его работоспособность,

приводит к переутомлению, появляется недомогание и общая усталость организма.

Для снижения вредного воздействия на человека нужно соблюдать ряд установленных правил использования ПЭВМ, например:

– правильная организация рабочего пространства;

– правильное освещение;

– соблюдение правил применения электрических устройств;

– соблюдение правил пожарной безопасности;

– соблюдение режима труда и отдыха.

2.1.1 Оборудование рабочего места

Рабочее место пользователя должно удовлетворять некоторым требованиям, среди которых:

– расстояние от экрана до глаз должно быть не менее 600 – 700 мм;

– монитор должен быть расположен ниже уровня глаз, при этом угол направления взгляда на экран должен быть в районе 10 – 15 градусов;

– рабочий стол должен иметь возможность регулировки по высоте в пределах 600 – 800 мм, а при отсутствии такой возможности его высота должна составлять в районе 725 мм;

– расположение клавиатуры должно быть на поверхности стола примерно в 100 – 300 мм от края со стороны пользователя;

– рабочий стол должен иметь пространство для ног с высотой не менее 600 мм, глубиной не менее 450 мм на уровне колен и не менее 650 мм на уровне вытянутых ног.

На рисунке 2.1 изображена организация рабочего места по указанным требованиям.

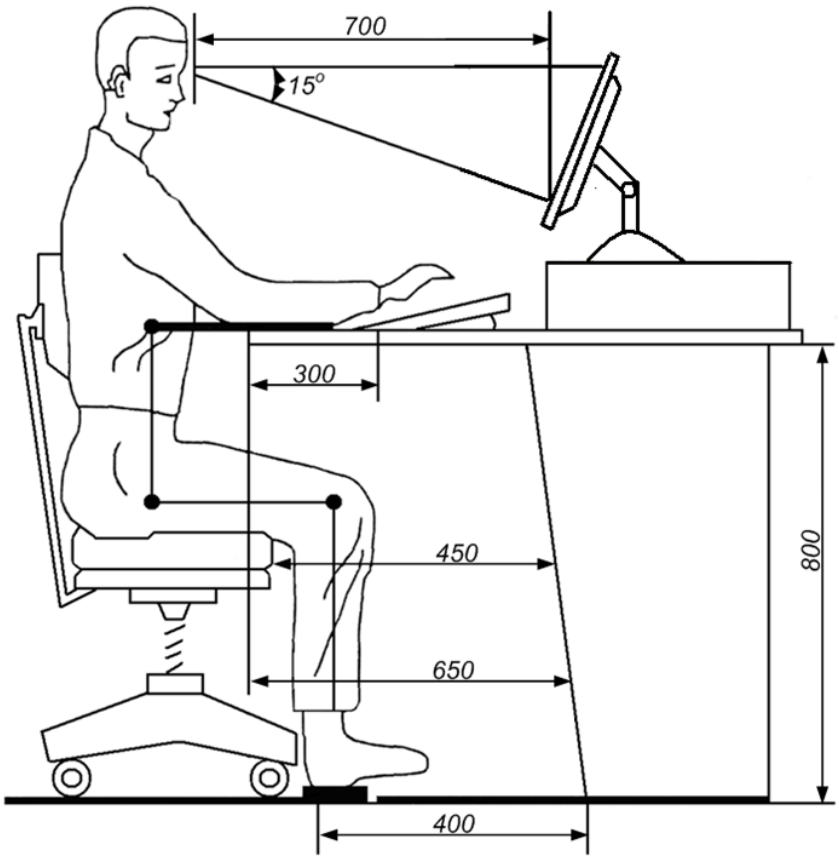


Рисунок 2.1 – Организация рабочего места

При правильной организации рабочего места обеспечивается правильная посадка пользователя ПЭВМ, которая показана на рисунке 2.2.

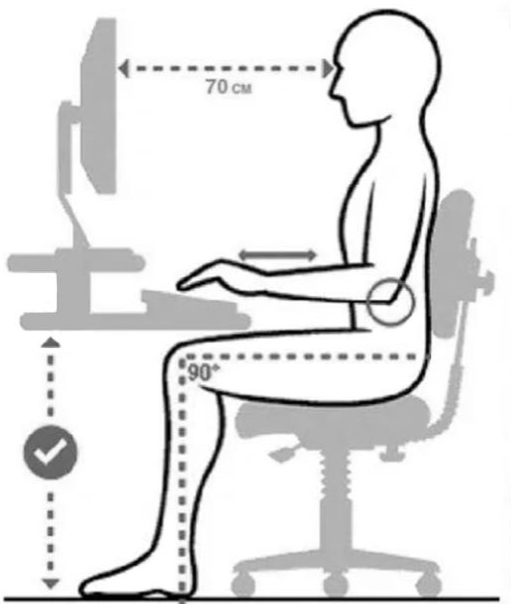


Рисунок 2.2 – Правильная посадка

При посадке за ПК нужно обеспечивать некоторые условия:

– плечи находятся в расслабленном состоянии;

– запястья находятся в прямом положении;

– локти нужно держать полностью на столе;

2.1.2 Освещённость рабочего пространства

Нормы освещённости устанавливаются в зависимости от характера выполняемой работы, места работы и поверхности. Для пультов ЭВМ и дисплеев – освещение не менее 400 люкс с общим освещением не менее 200 люкс.

Рабочее место должно обеспечиваться равномерной яркостью с исключением ярких и блестящих предметов. Для снижения монотонности можно использовать пестрые поверхности.

Для освещения чаще всего используются потолочные или настольные светильники с люминесцентными лампами. Для исключения засвечивания экранов светильники должны располагаться сбоку от рабочего места, параллельно линии зрения пользователя.

2.1.3 Электробезопасность

К электробезопасности предъявляются требования по ГОСТ 12.1.019-2017 «ЭЛЕКТРОБЕЗОПАСНОСТЬ».

ПК – это комплекс устройств, работающих от сети переменного тока с частотой 50 Гц и напряжением 230 В. Такое напряжение опасно для жизни. Для безопасности при работе с ПК нужно соблюдать следующие требования:

– питание всех устройств должно производиться от одной фазы электрической сети;

– сетевое электропитание устройств должно быть произведено от розеток типа «Европа» с заземляющими контактами;

– все электрические розетки, к которым подключаются устройства ПК, должны иметь маркировку по напряжению;

– значение номинального напряжения сети должно быть нанесено яркой краской, крупными символами на стене или щите возле розетки.

В процессе работы со средствами вычислительной техники каждый работник должен внимательно относиться к электропроводке, приборами и аппаратам и иметь понимание о том, что несоблюдение техники безопасности угрожает жизни и здоровью человека.

Необходимо регулярно следить за состоянием электропроводки, выключателей и розеток на своем рабочем месте и содержать их в исправном состоянии.

При обнаружении неисправности необходимо обесточить электрооборудование. Продолжить работу с элементом можно только после полного устранения неисправности.

2.1.4 Пожарная безопасность

При эксплуатации ПЭВМ самыми частыми причинами пожаров являются:

– короткие замыкания в проводках и электрических устройствах;

– воспламенение горючих материалов вблизи устройств;

– в местах контактных соединений наличие больших переходных сопротивлений.

Согласно федеральному закону от 21.12.1994 №69-ФЗ (ред. от 28.09.2010) «О пожарной безопасности» проводятся пожарные меры профилактики. Для обеспечения пожарной безопасности необходимо выполнить ряд мероприятий, которые направлены на обеспечение безопасности людей и предотвращение пожара с его дальнейшим распространением, а также на создание условий для успешной ликвидации пожара. К таким мероприятиям относятся:

– отсутствие загромождений на проходах в помещения и запасные выходы, препятствующих свободному прохождению;

– наличие свободного доступа к средствам пожаротушения;

– наличие устройств автоматического отключения системы вентиляции при пожаре, а также огне задерживающего клапана;

– наличие установленной автоматической пожарной сигнализации и средств пожаротушения, находящихся в исправном состоянии и в соответствии с типовыми правилами технического содержания.

2.1.5 Режим труда и отдыха

Работающий с ПК человек должен соблюдать режим труда и отдыха в зависимость от продолжительности деятельности, её вида и категории.

Существуют следующие виды трудовой деятельности:

– группа А – работа по считыванию информации с экрана дисплея с предварительным запросом;

– группа Б – работа по вводу информации;

– группа В – творческая работа в режиме диалога с ПЭВМ.

За основную работу с компьютером следует принимать вид деятельности, занимающий не менее 50% времени в течение всей рабочей смены или дня.

Для видов трудовой деятельности устанавливается 3 категории тяжести и напряжённости работы с компьютером, определяющиеся следующим образом:

– группа А – по суммарному числу считываемых знаков за рабочую смену (не более 60000 знаков);

– группа Б – по суммарному числу считываемых или вводимых знаков за рабочую смену (не более 40000 знаков);

– группа В – по суммарному времени непосредственной работы с компьютером за рабочую смену (не более 6 часов).

Заключение

В ходе выполнения выпускной квалификационной работы был разработан корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети на языке TypeScript. Графический интерфейс был реализован с помощью библиотеки React.

Разработанное приложение позволяет безопасно обмениваться сообщениями и файлами в реальном времени между пользователями, а также совершать аудио- и видеозвонки с использованием удобного и понятного графического интерфейса.

Одним из достоинств разработанного приложения является работа в условиях локальной сети, позволяющая достичь высокого уровня безопасности данных и исключить риск утечки.

В дальнейшем разработанное приложение можно усовершенствовать, добавив больше возможностей для кастомизации окна, улучшить имеющиеся системы до уровня ведущих мессенджеров. Реализовать более гибкую настройку аккаунта, добавить больше вариантов смены языка интерфейса.

Список использованных источников

1. bitrix24.ru: [Электронный ресурс]. URL: https://www.bitrix24.ru (дата обращения: 18.04.2026)
2. 360.yandex.ru: [Электронный ресурс]. URL: https://360.yandex.ru (дата обращения: 18.04.2026)
3. telegram.org: [Электронный ресурс]. URL: https://web.telegram.org (дата обращения: 18.04.2026)
4. max.ru: [Электронный ресурс]. URL: https://max.ru (дата обращения: 18.04.2026)
5. visualstudio.com: [Электронный ресурс]. URL: https://code.visualstudio.com (дата обращения: 18.04.2026)
6. react.dev: [Электронный ресурс]. URL: https://react.dev (дата обращения: 20.04.2026)
7. nodejs.org: [Электронный ресурс]. URL: https://nodejs.org (дата обращения: 19.04.2026)
8. prisma.io: [Электронный ресурс]. URL: https://www.prisma.io (дата обращения: 19.04.2026)
9. vite.dev: [Электронный ресурс]. URL: https://vite.dev (дата обращения: 21.04.2026)

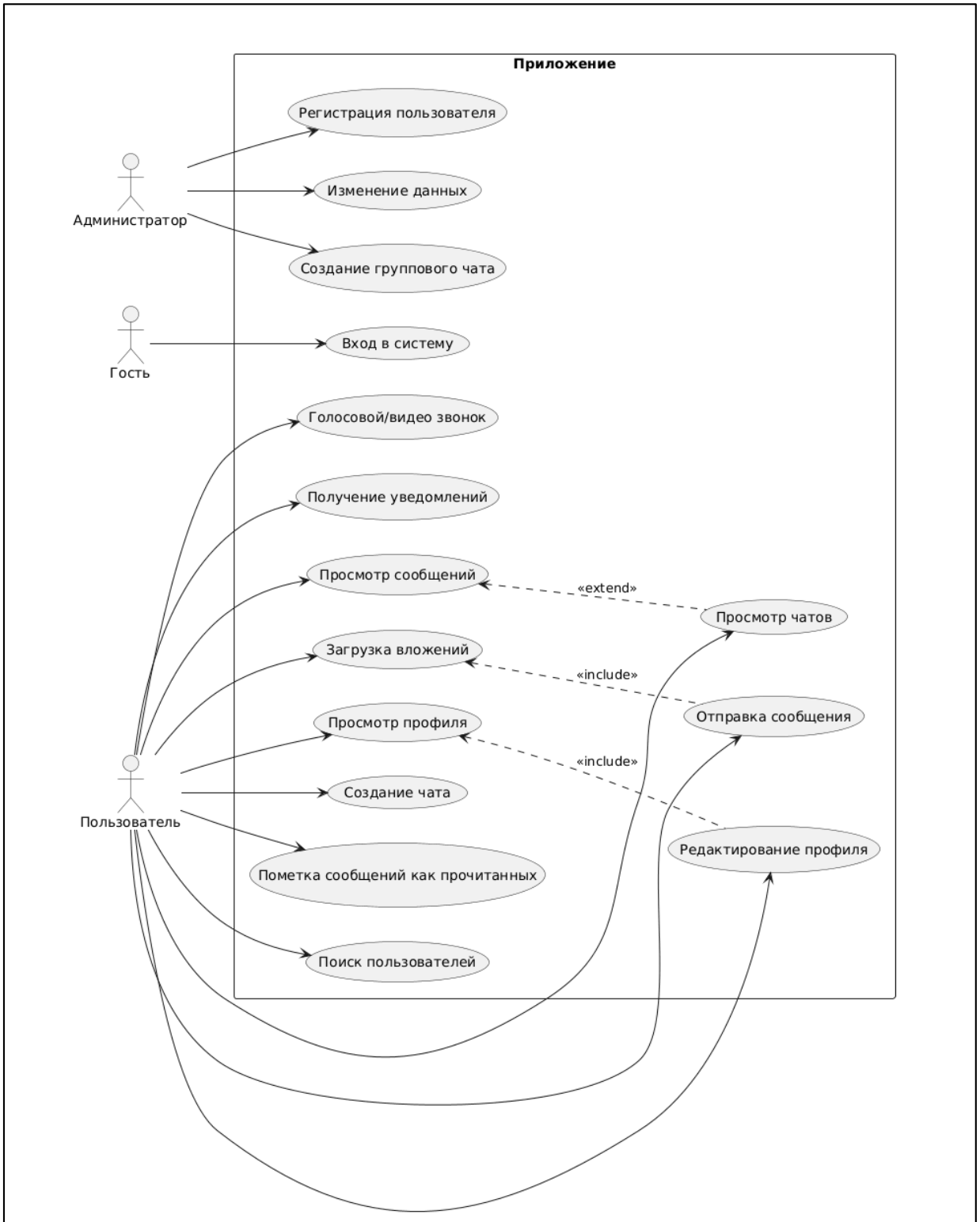
Приложение А

(обязательное)

Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети

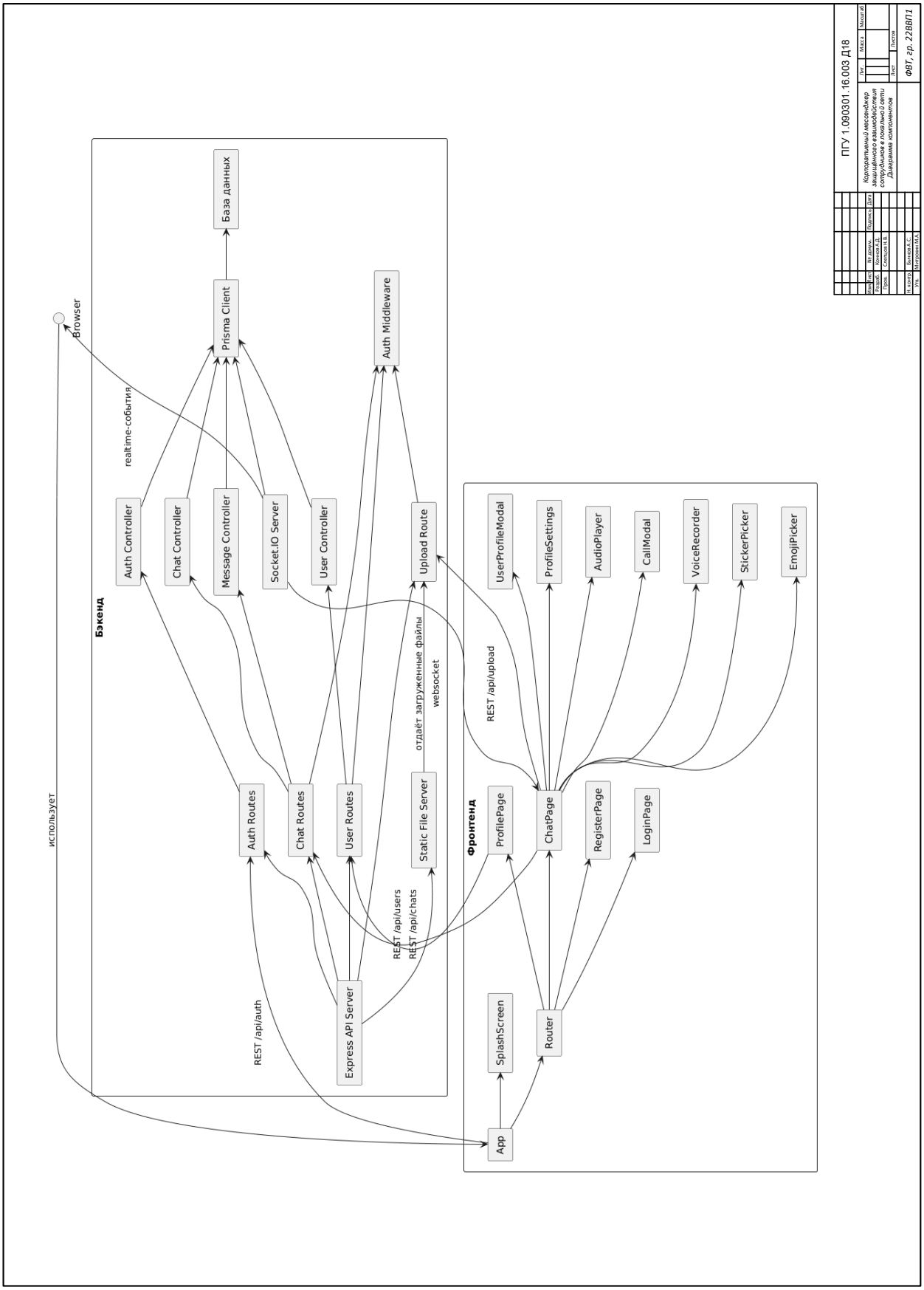
UML диаграммы

Приложение А.1 – UML-диаграмма прецедентов

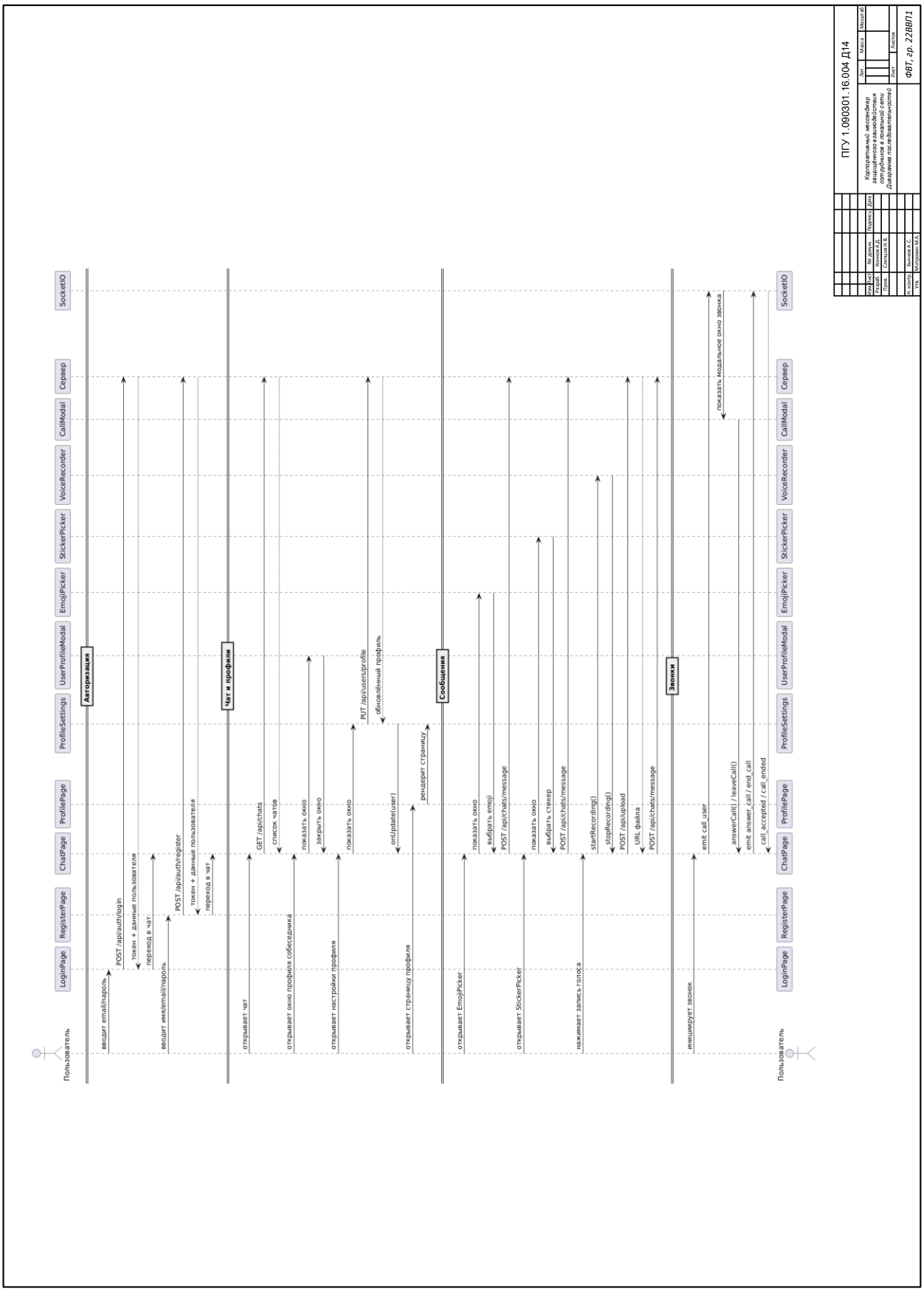


					ПГУ 1.090301.16.002 Д13	
Изм.	Лист	№ докум.	Подпись	Дата	*Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети Диаграмма прецедентов*	Лит. / Масса / Масштаб
Разраб.		Коннов А.Д.				
Пров.		Слепцов Н.В.				Лист / Листов
Н. контр.		Бычков А.С.				*ФВТ, гр. 22ВВП1*
Утв.		Митрохин М.А.				

Приложение А.2 – UML-диаграмма компонентов



Приложение А.3 – UML-диаграмма последовательностей



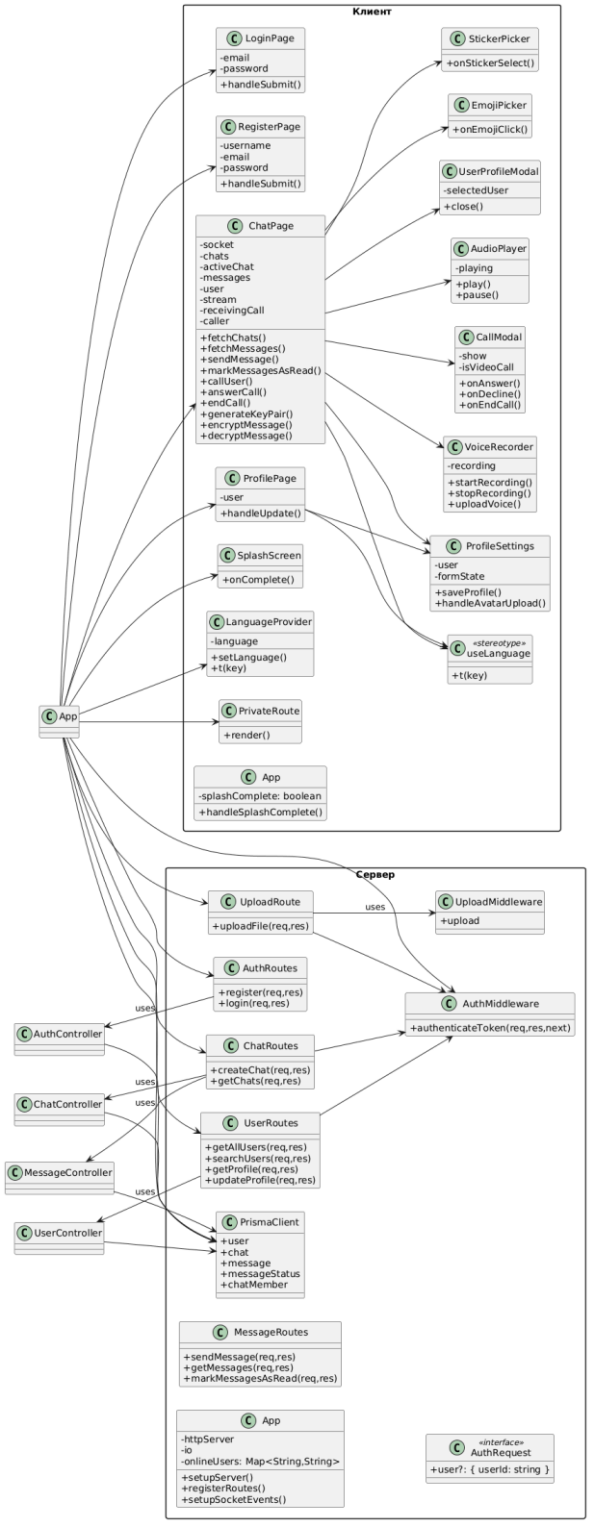
ПГУ 1.090301.16.004 Д14

Клиент-серверное приложение защищённого взаимодействия сотрудников в локальной сети

Диаграмма последовательностей

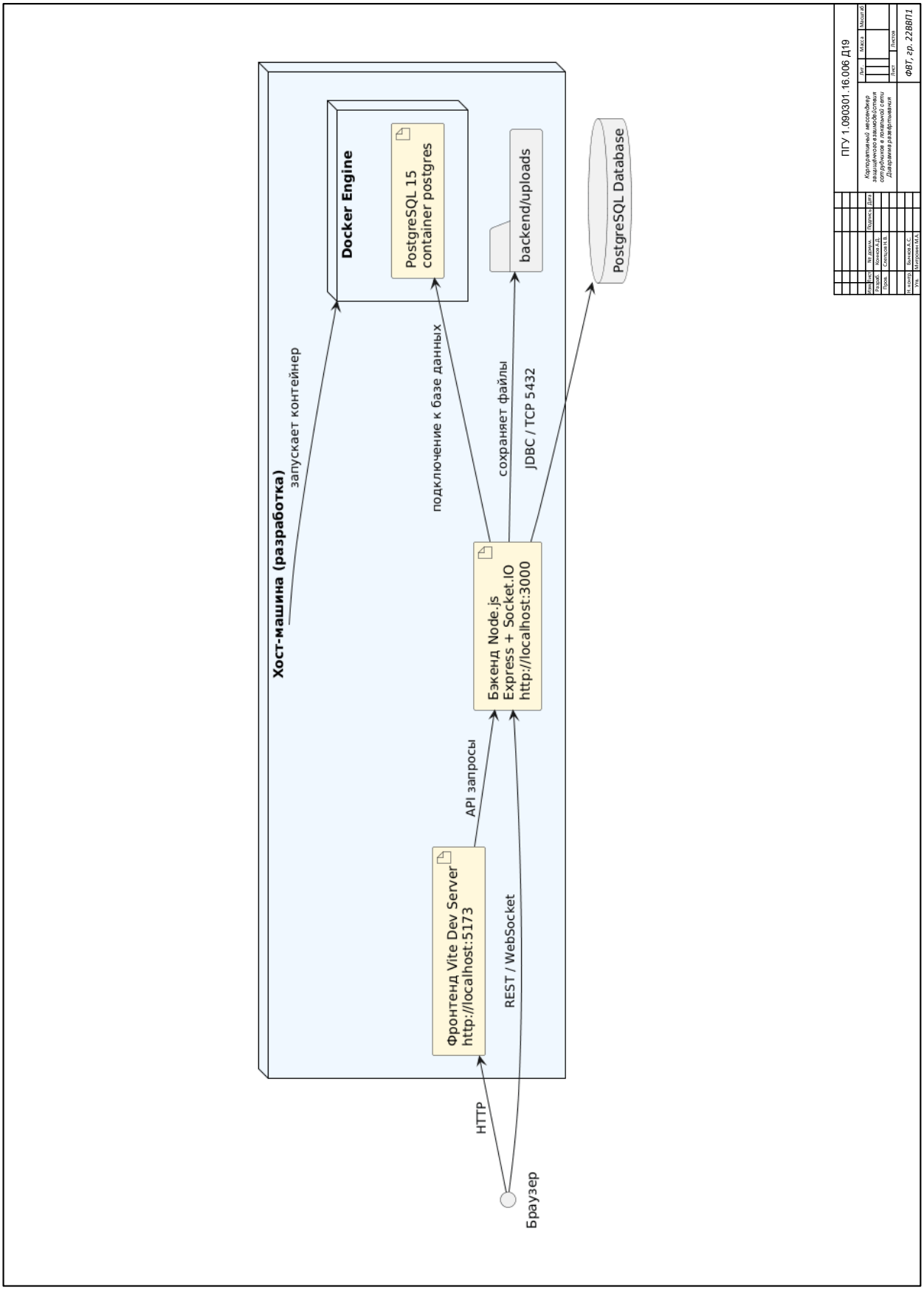
ФВТ, гр. 22ВВП1

Приложение А.4 – UML-диаграмма классов



					ПГУ 1.090301.16.005 Д11	
Изм.	Лист	№ докум.	Подпись	Дата	*Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети Диаграмма классов*	Лит. / Масса / Масштаб
Разраб.		Коннов А.Д.				Лист / Листов
Пров.		Слепцов Н.В.				
Н. контр.		Бычков А.С.				*ФВТ, гр. 22ВВП1*
Утв.		Митрохин М.А.				

Приложение А.5 – UML-диаграмма развёртывания



Приложение Б

(обязательное)

Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети

Листинг программы

Файл «Chat.tsx»

```
import React, { useEffect, useState, useRef } from 'react';
import { io, Socket } from 'socket.io-client';
import { Send, LogOut, User, Search, Plus, Settings, Smile,
Sticker, Phone, Video, Check, CheckCheck, Image as ImageIcon,
Paperclip } from 'lucide-react';
import { useNavigate } from 'react-router-dom';
import VoiceRecorder from '../components/VoiceRecorder';
import CustomEmojiPicker from '../components/EmojiPicker';
import StickerPicker from '../components/StickerPicker';
import AudioPlayer from '../components/AudioPlayer';
import CallModal from '../components/CallModal';
import ProfileSettings from '../components/ProfileSettings';
import UserProfileModal from '../components/UserProfileModal';
import Peer from 'simple-peer';
import type { EmojiClickData } from 'emoji-picker-react';
import { useLanguage } from '../context/LanguageContext';

const generateKeyPair = async (): Promise<{ publicKey:
CryptoKey; privateKey: CryptoKey }> => {
  const keyPair = await window.crypto.subtle.generateKey({
    name: 'RSA-OAEP',
    modulusLength: 2048,
    publicExponent: new Uint8Array([1, 0, 1]),
    hash: 'SHA-256',
  }, true, ['encrypt', 'decrypt']);

  return {
    publicKey: keyPair.publicKey,
    privateKey: keyPair.privateKey,
  };
};

const encryptMessage = async (message: string, publicKey:
CryptoKey): Promise<string> => {
  const encodedMessage = new TextEncoder().encode(message);
  const encryptedData = await window.crypto.subtle.encrypt(
    {
      name: 'RSA-OAEP',
    },
    publicKey,
    encodedMessage
  );
  return btoa(String.fromCharCode(...new
Uint8Array(encryptedData)));
};

const decryptMessage = async (encryptedMessage: string,
privateKey: CryptoKey): Promise<string> => {
  const encryptedData = Uint8Array.from(atob(encryptedMessage),
(c) => c.charCodeAt(0));
```

```
  const decryptedData = await window.crypto.subtle.decrypt(
    {
      name: 'RSA-OAEP',
    },
    privateKey,
    encryptedData
  );
  return new TextDecoder().decode(decryptedData);
};

interface Message {
  id: string;
  content: string;
  type: string;
  fileUrl?: string;
  senderId: string;
  chatId: string;
  createdAt: string;
  sender: { username: string; avatar: string | null };
  statuses: { userId: string; status: string }[];
}

interface Chat {
  id: string;
  name: string | null;
  chatMembers: { user: { id: string; username: string; avatar:
string | null; isOnline: boolean; lastSeen: string | null } }[];
  messages: Message[];
  unreadCount: number;
}

const Chat: React.FC = () => {
  const { t } = useLanguage();
  const [socket, setSocket] = useState<Socket | null>(null);
  const [chats, setChats] = useState<Chat[]>([]);
  const [activeChat, setActiveChat] = useState<Chat |
null>(null);
  const [messages, setMessages] = useState<Message[]>([]);
  const [newMessage, setNewMessage] = useState('');
  const [user, setUser] = useState<any>(null);
  const [searchQuery, setSearchQuery] = useState('');
  const [searchResults, setSearchResults] = useState<any[]>([]);
  const [showEmojiPicker, setShowEmojiPicker] = useState(false);
  const [showStickerPicker, setShowStickerPicker] =
useState(false);
  const [showProfileSettings, setShowProfileSettings] =
useState(false);

  const [selectedUserProfile, setSelectedUserProfile] =
useState<any>(null);
  const fileInputRef = useRef<HTMLInputElement>(null);
```

```
  const [stream, setStream] = useState<MediaStream |
null>(null);
  const [receivingCall, setReceivingCall] = useState(false);
  const [caller, setCaller] = useState('');
  const [callerSignal, setCallerSignal] = useState<any>();
  const [callAccepted, setCallAccepted] = useState(false);
  const [callEnded, setCallEnded] = useState(false);
  const [name, setName] = useState('');
  const [showCallModal, setShowCallModal] = useState(false);
  const [isVideoCall, setIsVideoCall] = useState(false);

  const myVideo = useRef<HTMLVideoElement>(null);
  const userVideo = useRef<HTMLVideoElement>(null);
  const connectionRef = useRef<Peer.Instance | null>(null);
  const messagesEndRef = useRef<HTMLDivElement>(null);
  const navigate = useNavigate();

  useEffect(() => {
    const token = localStorage.getItem('token');
    const storedUser = localStorage.getItem('user');

    if (!token || !storedUser) {
      navigate('/login');
      return;
    }

    try {
      const parsedUser = JSON.parse(storedUser);
      setUser(parsedUser);

      const newSocket = io('http://localhost:3000', {
        transports: ['websocket'],
        reconnection: true,
      });
      setSocket(newSocket);

      newSocket.on('connect', () => {
        console.log('Connected to socket');
        newSocket.emit('setup', parsedUser.id);
      });

      fetchChats(token);

      return () => {
        newSocket.disconnect();
      };
    } catch (error) {
      console.error('Error parsing user data:', error);
      navigate('/login');
    }
```

```
  }, [navigate]);

  useEffect(() => {
    if (socket && activeChat) {
      socket.emit('join_chat', activeChat.id);
      fetchMessages(activeChat.id);

      socket.on('new_message', (message: Message) => {
        if (activeChat && message.chatId === activeChat.id) {
             setMessages((prev) => [...prev, message]);
             scrollToBottom();
             markMessagesAsRead(activeChat.id);
        } else {
            if (Notification.permission === 'granted') {
                const storedUser = localStorage.getItem('user');
                if (storedUser) {
                    const parsedUser = JSON.parse(storedUser);
                    if (parsedUser.notificationsEnabled !==
false) {
                        new
Notification(`${t('new_message_from')}
${message.sender.username}`, {
                          body: message.content ||
(message.type === 'VOICE' ? t('voice_message') : t('sticker')),
                          icon: '/logo.png'
                        });

                        // Play notification sound
                        const audio = new Audio('/blink.mp3');
                        audio.volume = 0.5;
                        audio.load();
                        audio.play().catch(e =>
console.error("Audio play failed", e));
                    }
                }
            }
        }
        fetchChats(localStorage.getItem('token') || '');
      });

      socket.on('messages_read', ({ chatId }) => {
          fetchChats(localStorage.getItem('token') || '');
      });

      socket.on('user_online', (userId: string) => {
        setChats(prev => prev.map(chat => ({
          ...chat,
          chatMembers: chat.chatMembers.map(member =>
            member.user.id === userId ? { ...member, user: {
...member.user, isOnline: true } } : member
          )
```

```
        }));
      });

      socket.on('user_offline', (userId: string) => {
        setChats(prev => prev.map(chat => ({
          ...chat,
          chatMembers: chat.chatMembers.map(member =>
            member.user.id === userId ? { ...member, user: {
...member.user, isOnline: false, lastSeen: new
Date().toISOString() } } : member
          )
        })));
      });

      socket.on('messages_read', ({ chatId, userId }) => {
        if (activeChat && activeChat.id === chatId) {
            setMessages(prev => prev.map(msg => {
                if (msg.senderId === user.id) {
                    const existingStatus = msg.statuses.find(s
=> s.userId === userId);
                    if (existingStatus) {
                        return {
                            ...msg,
                            statuses: msg.statuses.map(s =>
s.userId === userId ? { ...s, status: 'READ' } : s)
                        };
                    } else {
                        return {
                            ...msg,
                            statuses: [...msg.statuses, {
userId, status: 'READ' }]
                        };
                    }
                }
                return msg;
            }));
        }
      });

      socket.on('call_user', (data) => {
          setReceivingCall(true);
          setCaller(data.from);
          setName(data.name);
          setCallerSignal(data.signal);
          setShowCallModal(true);
      });

      socket.on('call_ended', () => {
          setCallEnded(true);
          setShowCallModal(false);
          connectionRef.current?.destroy();
```

```
          window.location.reload();
        });

        return () => {
          socket.off('new_message');
          socket.off('user_online');
          socket.off('user_offline');
          socket.off('messages_read');
          socket.off('call_user');
          socket.off('call_ended');
        };
      }
    }, [socket, activeChat, user]);

    const fetchChats = async (token: string) => {
      try {
        const response = await
  fetch('http://localhost:3000/api/chats', {
          headers: { Authorization: `Bearer ${token}` },
        });
        const data = await response.json();
        if (Array.isArray(data)) {
          setChats(data);
        } else {
          console.error('Failed to fetch chats:', data);
          setChats([]);
        }
      } catch (error) {
        console.error('Error fetching chats:', error);
      }
    };

    const fetchMessages = async (chatId: string) => {
      try {
        const token = localStorage.getItem('token');
        const response = await
  fetch(`http://localhost:3000/api/chats/${chatId}/messages`, {
          headers: { Authorization: `Bearer ${token}` },
        });
        const data = await response.json();
        if (Array.isArray(data)) {
          setMessages(data);
          scrollToBottom();
          markMessagesAsRead(chatId);
        } else {
          console.error('Failed to fetch messages:', data);
          setMessages([]);
        }
      } catch (error) {
        console.error('Error fetching messages:', error);
      }
```

```
  };

  const markMessagesAsRead = async (chatId: string) => {
    try {
        const token = localStorage.getItem('token');
        await
fetch(`http://localhost:3000/api/chats/${chatId}/read`, {
            method: 'POST',
            headers: { Authorization: `Bearer ${token}` }
        });
        setChats(prev => prev.map(c => c.id === chatId ? { ...c,
unreadCount: 0 } : c));
    } catch (error) {
        console.error('Error marking messages as read:', error);
    }
  };

  const sendMessage = async (e: React.FormEvent) => {
    e.preventDefault();
    if (!newMessage.trim() || !activeChat) return;
    await sendTextMessage(newMessage);
    setNewMessage('');
    setShowEmojiPicker(false);
  };

  const onEmojiClick = (emojiData: EmojiClickData) => {
    setNewMessage((prev) => prev + emojiData.emoji);
  };

  const onStickerClick = async (stickerUrl: string) => {
    if (!activeChat) return;
    try {
      const token = localStorage.getItem('token');
      await fetch('http://localhost:3000/api/chats/message', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({
          chatId: activeChat.id,
          content: 'Sticker',
          type: 'STICKER',
          fileUrl: stickerUrl,
        }),
      });
      setShowStickerPicker(false);
    } catch (error) {
      console.error('Error sending sticker:', error);
    }
  };
```

```
  const handleFileUpload = async (e:
React.ChangeEvent<HTMLInputElement>) => {
    const file = e.target.files?.[0];
    if (!file || !activeChat) return;

    try {
      const token = localStorage.getItem('token');
      const formData = new FormData();
      formData.append('file', file);

      const uploadResponse = await
fetch('http://localhost:3000/api/upload', {
        method: 'POST',
        headers: { Authorization: `Bearer ${token}` },
        body: formData,
      });
      const { url } = await uploadResponse.json();

      const type = file.type.startsWith('image/') ? 'IMAGE' :
file.type.startsWith('video/') ? 'VIDEO' : 'DOCUMENT';

      await fetch('http://localhost:3000/api/chats/message', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({
          chatId: activeChat.id,
          content: file.name,
          type,
          fileUrl: url,
        }),
      });
    } catch (error) {
      console.error('Error uploading file:', error);
    }
  };

  const sendTextMessage = async (content: string) => {
    if (!activeChat) return;
    try {
      const token = localStorage.getItem('token');
      await fetch('http://localhost:3000/api/chats/message', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({
```

```
          chatId: activeChat.id,
          content,
          type: 'TEXT',
        }),
      });
    } catch (error) {
      console.error('Error sending message:', error);
    }
  };

  const sendVoiceMessage = async (audioBlob: Blob) => {
    if (!activeChat) return;
    try {
      const token = localStorage.getItem('token');
      const formData = new FormData();
      formData.append('file', audioBlob, 'voice-message.webm');

      const uploadResponse = await
fetch('http://localhost:3000/api/upload', {
        method: 'POST',
        headers: { Authorization: `Bearer ${token}` },
        body: formData,
      });
      const { url } = await uploadResponse.json();

      await fetch('http://localhost:3000/api/chats/message', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({
          chatId: activeChat.id,
          content: 'Voice Message',
          type: 'VOICE',
          fileUrl: url,
        }),
      });
    } catch (error) {
      console.error('Error sending voice message:', error);
    }
  };

  const scrollToBottom = () => {
    messagesEndRef.current?.scrollIntoView({ behavior: 'smooth'
});
  };

  const handleLogout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
```

```
    navigate('/login');
  };

  const handleUpdateProfile = (updatedUser: any) => {
    localStorage.setItem('user', JSON.stringify(updatedUser));
    setUser(updatedUser);
  };

  const getChatName = (chat: Chat) => {
    const partner = chat.chatMembers?.find((m) => m.user.id !==
user?.id);
    return partner?.user.username || t('unknown_user');
  };

  const getChatAvatar = (chat: Chat) => {
    const partner = chat.chatMembers?.find((m) => m.user.id !==
user?.id);
    return partner?.user.avatar;
  };

  const getChatStatus = (chat: Chat) => {
    const partner = chat.chatMembers?.find((m) => m.user.id !==
user?.id);
    if (!partner) return null;
    if (partner.user.isOnline) return t('online');
    if (partner.user.lastSeen) {
        const date = new Date(partner.user.lastSeen);
        return `${t('was_online')} ${date.toLocaleTimeString([],
{hour: '2-digit', minute:'2-digit'})}`;
    }
    return t('offline');
  };

  const getMessageStatusIcon = (msg: Message) => {
    if (msg.senderId !== user?.id) return null;

    const isRead = msg.statuses?.some(s => s.status === 'READ'
&& s.userId !== user.id);
    if (isRead) return <CheckCheck size={14} className="text-
flick-blue" />;
    const chat = chats.find(c => c.id === msg.chatId);
    const partner = chat?.chatMembers.find(m => m.user.id !==
user.id);
    const isDelivered = partner?.user.isOnline;
    if (isDelivered) return <CheckCheck size={14}
className="text-white/30" />;
    return <Check size={14} className="text-white/30" />;
  };

  const handleSearch = async (query: string) => {
    setSearchQuery(query);
```

```
    if (query.length > 2) {
      try {
        const token = localStorage.getItem('token');
        const response = await
fetch(`http://localhost:3000/api/users/search?query=${query}`, {
          headers: { Authorization: `Bearer ${token}` },
        });
        const data = await response.json();
        if (Array.isArray(data)) {
          setSearchResults(data);
        } else {
          console.error('Failed to search users:', data);
          setSearchResults([]);
        }
      } catch (error) {
        console.error('Error searching users:', error);
      }
    } else {
      setSearchResults([]);
    }
  };

  const startChat = async (partnerId: string) => {
    try {
      const token = localStorage.getItem('token');
      const response = await
fetch('http://localhost:3000/api/chats', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({ partnerId, isGroup: false }),
      });
      const chat = await response.json();
      setChats((prev) => {
        if (prev.find((c) => c.id === chat.id)) return prev;
        return [chat, ...prev];
      });
      setActiveChat(chat);
      setSearchQuery('');
      setSearchResults([]);
    } catch (error) {
      console.error('Error creating chat:', error);
    }
  };

  const callUser = (isVideo: boolean) => {
    setIsVideoCall(isVideo);
    setShowCallModal(true);
```

```
    navigator.mediaDevices.getUserMedia({ video: isVideo, audio:
true }).then((currentStream) => {
      setStream(currentStream);
      if (myVideo.current) {
        myVideo.current.srcObject = currentStream;
      }

      const peer = new Peer({
        initiator: true,
        trickle: false,
        stream: currentStream,
      });

      peer.on('signal', (data) => {
        if (activeChat && socket) {
          const partner = activeChat.chatMembers.find((m) =>
m.user.id !== user?.id);
          if (partner) {
            socket.emit('call_user', {
              userToCall: partner.user.id,
              signalData: data,
              from: user.id,
              name: user.username,
            });
          }
        }
      });

      peer.on('stream', (currentStream) => {
        if (userVideo.current) {
          userVideo.current.srcObject = currentStream;
        }
      });

      socket?.on('call_accepted', (signal) => {
        setCallAccepted(true);
        peer.signal(signal);
      });

      connectionRef.current = peer;
    });
  };

  const answerCall = () => {
    setCallAccepted(true);
    navigator.mediaDevices.getUserMedia({ video: isVideoCall,
audio: true }).then((currentStream) => {
      setStream(currentStream);
      if (myVideo.current) {
        myVideo.current.srcObject = currentStream;
      }
```

```
      const peer = new Peer({
        initiator: false,
        trickle: false,
        stream: currentStream,
      });

      peer.on('signal', (data) => {
        socket?.emit('answer_call', { signal: data, to: caller
});
      });

      peer.on('stream', (currentStream) => {
        if (userVideo.current) {
          userVideo.current.srcObject = currentStream;
        }
      });

      peer.signal(callerSignal);
      connectionRef.current = peer;
    });
  };

  const leaveCall = () => {
    setCallEnded(true);
    setShowCallModal(false);
    connectionRef.current?.destroy();
    stream?.getTracks().forEach(track => track.stop());
    setStream(null);
    if (activeChat && socket) {
       const partner = activeChat.chatMembers.find((m) =>
m.user.id !== user?.id);
       if (partner) {
         socket.emit('end_call', { to: partner.user.id });
       }
    }
  };

  let publicKey: CryptoKey;
  let privateKey: CryptoKey;

  (async () => {
    const keyPair = await generateKeyPair();
    publicKey = keyPair.publicKey;
    privateKey = keyPair.privateKey;
  })();

  const sendEncryptedMessage = async (content: string):
Promise<void> => {
    if (!activeChat) return;
    try {
```

```
      const encryptedContent = await encryptMessage(content,
publicKey);
      const token = localStorage.getItem('token');
      await fetch('http://localhost:3000/api/chats/message', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          Authorization: `Bearer ${token}`,
        },
        body: JSON.stringify({
          chatId: activeChat.id,
          content: encryptedContent,
          type: 'TEXT',
        }),
      });
    } catch (error) {
      console.error('Error sending encrypted message:', error);
    }
  };

  const receiveAndDecryptMessage = async (encryptedMessage:
string): Promise<string> => {
    return await decryptMessage(encryptedMessage, privateKey);
  };

  return (
    <div className="flex h-screen bg-flick-dark overflow-
hidden">
      {selectedUserProfile && (
        <UserProfileModal
            user={selectedUserProfile}
            onClose={() => setSelectedUserProfile(null)}
        />
      )}
      {showProfileSettings && (
        <ProfileSettings
            user={user}
            onClose={() => setShowProfileSettings(false)}
            onUpdate={handleUpdateProfile}
        />
      )}
      {showCallModal && (
        <CallModal
          stream={stream}
          callAccepted={callAccepted}
          callEnded={callEnded}
          userVideo={userVideo as
React.RefObject<HTMLVideoElement>}
          myVideo={myVideo as React.RefObject<HTMLVideoElement>}
          name={name || (activeChat ? getChatName(activeChat) :
'Unknown')}
```

```
          callUser={() => callUser(isVideoCall)}
          leaveCall={leaveCall}
          answerCall={answerCall}
          receivingCall={receivingCall}
          isVideo={isVideoCall}
        />
      )}
      {/* Sidebar */}
      <div className={`w-full md:w-80 glass-panel border-r
border-white/10 flex flex-col flex-shrink-0 ${activeChat ?
'hidden md:flex' : 'flex'}`}>
        <div className="p-4 border-b border-white/10 flex
justify-between items-center">
          <div className="flex items-center gap-3 cursor-
pointer" onClick={() => setShowProfileSettings(true)}>
            <div className="w-10 h-10 rounded-full bg-gradient-
to-br from-flick-orange to-flick-blue p-[2px]">
              <div className="w-full h-full rounded-full bg-
flick-dark flex items-center justify-center overflow-hidden">
                {user?.avatar ? (
                  <img
                    src={user.avatar.startsWith('http') ?
user.avatar : `http://localhost:3000${user.avatar}`}
                    alt={user.username}
                    className="w-full h-full object-cover"
                  />
                ) : (
                  <User size={20} className="text-white" />
                )}
              </div>
            </div>
            <span className="font-pixel text-xs text-
white">{user?.username}</span>
          </div>
          <div className="flex gap-2">
            <button onClick={() => setShowProfileSettings(true)}
className="text-white/50 hover:text-flick-blue transition-
colors">
              <Settings size={20} />
            </button>
            <button onClick={handleLogout} className="text-
white/50 hover:text-flick-orange transition-colors">
              <LogOut size={20} />
            </button>
          </div>
        </div>

        <div className="p-4 flex gap-2 items-center">
          <div className="relative">
            <Search className="absolute left-3 top-1/2 -
translate-y-1/2 text-white/50" size={16} />
```

```
            <input
              type="text"
              value={searchQuery}
              onChange={(e) => handleSearch(e.target.value)}
              placeholder={t('search')}
              className="pixel-input w-full pl-10 rounded-lg
text-xs"
            />
          </div>
          {searchResults.length > 0 && (
            <div className="absolute z-50 w-full mt-2 glass-
panel rounded-lg overflow-hidden">
              {searchResults.map((result) => (
                <div
                  key={result.id}
                  onClick={() => startChat(result.id)}
                  className="p-3 hover:bg-white/10 cursor-
pointer flex items-center gap-3"
                >
                  <div className="w-8 h-8 rounded-full bg-
white/10 flex items-center justify-center">
                    <span className="font-pixel text-[10px]
text-white">
                      {result.username.charAt(0).toUpperCase()}
                    </span>
                  </div>
                  <span className="font-pixel text-xs text-
white">{result.username}</span>
                  <Plus size={14} className="ml-auto text-flick-
blue" />
                </div>
              ))}
            </div>
          )}
        </div>

        <div className="flex-1 overflow-y-auto">
          {chats.map((chat) => {
            const avatar = getChatAvatar(chat);
            return (
              <div
                key={chat.id}
                onClick={() => setActiveChat(chat)}
                className={`p-4 cursor-pointer hover:bg-white/5
transition-colors border-b border-white/5 ${
                  activeChat?.id === chat.id ? 'bg-white/10
border-l-4 border-l-flick-orange' : ''
                }`}
              >
                <div className="flex items-center gap-3">
                  <div className="relative">
```

```
                    <div className="w-10 h-10 rounded-full bg-
white/10 flex items-center justify-center overflow-hidden">
                        {avatar ? (
                        <img
                            src={avatar.startsWith('http') ?
avatar : `http://localhost:3000${avatar}`}
                            alt={getChatName(chat) || ''}
                            className="w-full h-full object-
cover"
                        />
                        ) : (
                        <span className="font-pixel text-xs
text-flick-blue">
                            {(getChatName(chat) ||
'?').charAt(0).toUpperCase()}
                        </span>
                        )}
                    </div>
                    <div className={`absolute bottom-0 right-0
w-3 h-3 rounded-full border-2 border-flick-dark ${
                      chat.chatMembers.find(m => m.user.id !==
user?.id)?.user.isOnline
                       ? 'bg-green-500'
                       : 'bg-gray-500'
                    }`} />
                  </div>
                  <div className="flex-1 min-w-0">
                    <div className="flex justify-between items-
center mb-1">
                        <h3 className="font-pixel text-xs text-
white truncate">{getChatName(chat)}</h3>
                        {chat.messages?.[0] && (
                            <span className="text-[10px] text-
white/30">
                                {new
Date(chat.messages[0].createdAt).toLocaleTimeString([], {hour:
'2-digit', minute:'2-digit'})}
                            </span>
                        )}
                    </div>
                    <div className="flex justify-between items-
center">
                        <p className="text-xs text-white/50
truncate w-32">
                        {chat.messages?.[0] ? (
                            <span className={chat.unreadCount >
0 ? 'text-white font-bold' : ''}>
                                {chat.messages[0].senderId ===
user?.id && <span className="text-flick-blue">Вы: </span>}
                                {chat.messages[0].type ===
'VOICE' ? t('voice_message') :
```

```
                                    chat.messages[0].type ===
'STICKER' ? t('sticker') :
                                    chat.messages[0].content}
                                </span>
                            ) : (
                                chat.chatMembers.find(m => m.user.id
!== user?.id)?.user.isOnline ? (
                                    <span className="text-flick-
blue">{t('online')}</span>
                                ) : (
                                    <span className="text-
white/30">{t('offline')}</span>
                                )
                            )}
                            </p>
                            {chat.unreadCount > 0 && (
                                <div className="w-5 h-5 rounded-full
bg-flick-blue flex items-center justify-center">
                                    <span className="text-[10px]
font-bold text-white">{chat.unreadCount}</span>
                                </div>
                            )}
                        </div>
                      </div>
                    </div>
                  </div>
                );
              })}
            </div>
          </div>

          {/* Chat Area */}
          <div className={`flex-1 flex flex-col relative
${!activeChat ? 'hidden md:flex' : 'flex'}`}>
            {activeChat ? (
              <>
                {/* Header */}
                <div className="h-16 glass-panel border-b border-
white/10 flex items-center px-4 md:px-6 justify-between z-10">
                  <div className="flex items-center gap-3">
                    <button
                        onClick={() => setActiveChat(null)}
                        className="md:hidden text-white/50
hover:text-white mr-2"
                    >
                        <svg xmlns="http://www.w3.org/2000/svg"
width="24" height="24" viewBox="0 0 24 24" fill="none"
stroke="currentColor" strokeWidth="2" strokeLinecap="round"
strokeLinejoin="round"><path d="m15 18-6-6 6-6"/></svg>
                    </button>
                    <div
```

```
                    className="flex items-center gap-3 cursor-
pointer hover:opacity-80 transition-opacity"
                    onClick={() => {
                        const partner =
activeChat.chatMembers.find((m) => m.user.id !== user?.id);
                        if (partner) {
                            // Fetch full user details including
bio
                            fetch(`http://localhost:3000/api/use
rs/search?query=${partner.user.username}`, {
                                headers: { Authorization:
`Bearer ${localStorage.getItem('token')}` }
                            })
                            .then(res => res.json())
                            .then(data => {
                                const fullUser = data.find((u:
any) => u.id === partner.user.id);
                                if (fullUser)
setSelectedUserProfile(fullUser);
                            });
                        }
                    }}
                >
                    <div className="w-10 h-10 rounded-full bg-
gradient-to-br from-flick-blue to-flick-orange p-[2px]">
                    <div className="w-full h-full rounded-full
bg-flick-dark flex items-center justify-center overflow-hidden">
                        {getChatAvatar(activeChat) ? (
                        <img
                            src={getChatAvatar(activeChat)!.star
tsWith('http') ? getChatAvatar(activeChat)! :
`http://localhost:3000${getChatAvatar(activeChat)}`}
                            alt={getChatName(activeChat) || ''}
                            className="w-full h-full object-
cover"
                        />
                        ) : (
                        <span className="font-pixel text-xs
text-white">
                            {(getChatName(activeChat) ||
'?').charAt(0).toUpperCase()}
                        </span>
                        )}
                    </div>
                    </div>
                    <div>
                    <h2 className="font-pixel text-sm text-
white">{getChatName(activeChat)}</h2>
                    <span className="text-xs text-flick-
blue">{getChatStatus(activeChat)}</span>
                    </div>
```

```
                </div>
              </div>
              <div className="flex gap-4">
                <button
                  onClick={() => callUser(false)}
                  className="text-white/50 hover:text-flick-
orange transition-colors"
                >
                  <Phone size={20} />
                </button>
                <button
                  onClick={() => callUser(true)}
                  className="text-white/50 hover:text-flick-blue
transition-colors"
                >
                  <Video size={20} />
                </button>
              </div>
            </div>

            {/* Messages */}
            <div className="flex-1 overflow-y-auto p-6 space-y-
4">
              {messages.map((msg) => {
                const isMe = msg.senderId === user?.id;
                return (
                  <div key={msg.id} className={`flex ${isMe ?
'justify-end' : 'justify-start'}`}>
                    <div
                      className={`max-w-[70%] p-3 rounded-lg
backdrop-blur-sm ${
                        isMe
                          ? 'bg-flick-blue/20 border border-
flick-blue/50 rounded-tr-none'
                          : 'bg-white/10 border border-white/20
rounded-tl-none'
                      }`}
                    >
                      {!isMe && (
                        <p className="text-[10px] text-flick-
orange font-pixel mb-1">
                          {msg.sender.username}
                        </p>
                      )}
                      {msg.type === 'VOICE' ? (
                        <AudioPlayer src={msg.fileUrl || ''} />
                      ) : msg.type === 'STICKER' ? (
                        <img src={msg.fileUrl} alt="Sticker"
className="w-32 h-32 object-contain" />
                      ) : msg.type === 'IMAGE' ? (
```

```
                        <img src={msg.fileUrl} alt="Image"
className="max-w-full rounded-lg" />
                      ) : msg.type === 'VIDEO' ? (
                        <video src={msg.fileUrl} controls
className="max-w-full rounded-lg" />
                      ) : msg.type === 'DOCUMENT' && msg.fileUrl
? (
                        <a
                          href={msg.fileUrl}
                          download
                          target="_blank"
                          rel="noopener noreferrer"
                          className="text-flick-blue underline
block"
                        >
                          {msg.content}
                        </a>
                      ) : (
                        <p className="text-sm text-
white">{msg.content}</p>
                      )}
                      <div className="flex items-center justify-
end gap-1 mt-1">
                        <p className="text-[10px] text-
white/30">
                          {new
Date(msg.createdAt).toLocaleTimeString([], { hour: '2-digit',
minute: '2-digit' })}
                        </p>
                        {getMessageStatusIcon(msg)}
                      </div>
                    </div>
                  </div>
                );
              })}
              <div ref={messagesEndRef} />
            </div>

            {/* Input */}
            <div className="p-4 glass-panel border-t border-
white/10 relative">
              {showEmojiPicker && <CustomEmojiPicker
onEmojiClick={onEmojiClick} />}
              {showStickerPicker && <StickerPicker
onStickerClick={onStickerClick} />}
              <div className="flex gap-2 md:gap-4 items-center">
                <VoiceRecorder onSend={sendVoiceMessage} />
                <button
                  onClick={() => fileInputRef.current?.click()}
                  className="text-white/50 hover:text-flick-blue
transition-colors"
```

```
                >
                  <Paperclip size={20} />
                </button>
                <input
                    type="file"
                    ref={fileInputRef}
                    onChange={handleFileUpload}
                    className="hidden"
                    accept="image/*,video/*"
                />
                <button
                  onClick={() => {
                    setShowEmojiPicker(!showEmojiPicker);
                    setShowStickerPicker(false);
                  }}
                  className="text-white/50 hover:text-flick-
orange transition-colors hidden md:block"
                >
                  <Smile size={20} />
                </button>
                <button
                  onClick={() => {
                    setShowStickerPicker(!showStickerPicker);
                    setShowEmojiPicker(false);
                  }}
                  className="text-white/50 hover:text-flick-blue
transition-colors hidden md:block"
                >
                  <Sticker size={20} />
                </button>
                <form onSubmit={sendMessage} className="flex-1
flex gap-2 md:gap-4">
                  <input
                    type="text"
                    value={newMessage}
                    onChange={(e) =>
setNewMessage(e.target.value)}
                    placeholder=""
                    className="pixel-input flex-1 rounded-lg"
                  />
                  <button
                    type="submit"
                    className="bg-flick-orange hover:bg-flick-
blue text-white p-3 rounded-lg transition-colors shadow-pixel
active:translate-x-[2px] active:translate-y-[2px] active:shadow-
none"
                  >
                    <Send size={20} />
                  </button>
                </form>
              </div>
```

```
            </div>
          </>
        ) : (
          <div className="flex-1 flex flex-col items-center
justify-center text-white/30">
            <div className="w-150 h-64 mb-6 animate-pulse">
              <img src="/logo.png" alt="ЧАТ" className="w-full
h-full object-contain opacity-50" />
            </div>
            <p className="font-pixel text-xs text-
white/20">{t('select_chat')}</p>
          </div>
        )}
      </div>
    </div>
  );
};

export default Chat;
```

Файл «Login.tsx»

```
import React, { useState } from 'react';
import { Link, useNavigate } from 'react-router-dom';

const Login: React.FC = () => {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();
    try {
      const response = await
fetch('http://localhost:3000/api/auth/login', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ email, password }),
      });
      const data = await response.json();
      if (response.ok) {
        localStorage.setItem('token', data.token);
        localStorage.setItem('user', JSON.stringify(data.user));
        navigate('/chat');
      } else {
        alert(data.message);
      }
    } catch (error) {
      console.error('Login error:', error);
    }
  };

  return (
```

```
    <div className="min-h-screen flex items-center justify-
center p-4 bg-flick-dark">
      <div className="glass-panel p-6 md:p-8 rounded-2xl w-full
max-w-md relative overflow-hidden">
        <div className="absolute top-0 left-0 w-full h-2 bg-
gradient-to-r from-flick-orange to-flick-blue"></div>

        <h2 className="text-2xl md:text-3xl font-pixel text-
center mb-8 text-transparent bg-clip-text bg-gradient-to-r from-
flick-orange to-flick-blue">
          ВХОД В АККАУНТ
        </h2>

        <form onSubmit={handleSubmit} className="space-y-6">
          <div>
            <label className="block font-pixel text-xs mb-2
text-flick-blue">EMAIL</label>
            <input
              type="email"
              value={email}
              onChange={(e) => setEmail(e.target.value)}
              className="pixel-input w-full rounded-lg"
              placeholder="ВВЕДИТЕ EMAIL..."
            />
          </div>

          <div>
            <label className="block font-pixel text-xs mb-2
text-flick-orange">ПАРОЛЬ</label>
            <input
              type="password"
              value={password}
              onChange={(e) => setPassword(e.target.value)}
              className="pixel-input w-full rounded-lg"
              placeholder="ВВЕДИТЕ ПАРОЛЬ..."
            />
          </div>

          <button type="submit" className="pixel-btn w-full
rounded-lg text-sm py-3 mt-4">
            НАЧАТЬ ОБЩЕНИЕ
          </button>
        </form>


      </div>
    </div>
  );
};

export default Login;
```

Файл «Profile.tsx»

```
import React, { useState, useEffect } from 'react';
import { useNavigate } from 'react-router-dom';
import { ArrowLeft } from 'lucide-react';
import ProfileSettings from '../components/ProfileSettings';

const Profile: React.FC = () => {
  const navigate = useNavigate();
  const [user, setUser] = useState<any>(null);

  useEffect(() => {
    const storedUser = localStorage.getItem('user');
    if (storedUser) {
      try {
        const parsedUser = JSON.parse(storedUser);
        setUser(parsedUser);
      } catch (error) {
        console.error('Error parsing user data:', error);
        localStorage.removeItem('user');
        navigate('/login');
      }
    } else {
      navigate('/login');
    }
  }, [navigate]);

  const handleUpdate = (updatedUser: any) => {
    localStorage.setItem('user', JSON.stringify(updatedUser));
    setUser(updatedUser);
  };

  if (!user) return null;

  return (
    <div className="min-h-screen bg-flick-dark flex flex-col
items-center p-4">
      <div className="w-full max-w-md">
        <button
          onClick={() => navigate('/chat')}
          className="flex items-center gap-2 text-white/50
hover:text-flick-orange transition-colors mb-6 font-pixel text-
xs"
        >
          <ArrowLeft size={16} />
          НАЗАД В ЧАТ
        </button>

        <ProfileSettings
            user={user}
            onClose={() => navigate('/chat')}
            onUpdate={handleUpdate}
```

```
            isPage={true}
          />
        </div>
      </div>
    );
};

export default Profile;
```

Файл «AuthController.ts»

```
import { Request, Response } from 'express';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import prisma from '../utils/prisma';

export const register = async (req: Request, res: Response) => {
  try {
    const { username, email, password } = req.body;

    const existingUser = await prisma.user.findFirst({
      where: {
        OR: [
          { email },
          { username }
        ]
      }
    });

    if (existingUser) {
      res.status(400).json({ message: 'User already exists' });
      return;
    }

    const hashedPassword = await bcrypt.hash(password, 10);

    const user = await prisma.user.create({
      data: {
        username,
        email,
        password: hashedPassword
      }
    });

    const token = jwt.sign({ userId: user.id },
process.env.JWT_SECRET as string, { expiresIn: '1h' });
    const refreshToken = jwt.sign({ userId: user.id },
process.env.JWT_REFRESH_SECRET as string, { expiresIn: '7d' });

    res.status(201).json({
      message: 'User created successfully',
```

```
      user: { id: user.id, username: user.username, email:
user.email },
      token,
      refreshToken
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};

export const login = async (req: Request, res: Response) => {
  try {
    const { email, password } = req.body;

    const user = await prisma.user.findUnique({ where: { email }
});
    if (!user) {
      res.status(400).json({ message: 'Invalid credentials' });
      return;
    }

    const isMatch = await bcrypt.compare(password,
user.password);
    if (!isMatch) {
      res.status(400).json({ message: 'Invalid credentials' });
      return;
    }

    const token = jwt.sign({ userId: user.id },
process.env.JWT_SECRET as string, { expiresIn: '1h' });
    const refreshToken = jwt.sign({ userId: user.id },
process.env.JWT_REFRESH_SECRET as string, { expiresIn: '7d' });

    res.json({
      message: 'Login successful',
      user: { id: user.id, username: user.username, email:
user.email, avatar: user.avatar },
      token,
      refreshToken
    });
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

Файл «chatController.ts»

```
import { Response } from 'express';
import prisma from '../utils/prisma';
import { AuthRequest } from '../middleware/authMiddleware';
```

```
export const createChat = async (req: AuthRequest, res:
Response) => {
  try {
    const { partnerId, isGroup, name, members } = req.body;
    const userId = req.user?.userId;

    if (!userId) {
      res.status(401).json({ message: 'Unauthorized' });
      return;
    }

    if (isGroup) {
      if (!Array.isArray(members) || members.length < 1) {
        res.status(400).json({ message: 'Members array required
for group chat' });
        return;
      }
      const uniqueMembers = Array.from(new
Set(members.filter((id: string) => id !== userId)));
      if (uniqueMembers.length < 1) {
        res.status(400).json({ message: 'At least one other
member required' });
        return;
      }
      const chat = await prisma.chat.create({
        data: {
          isGroup: true,
          name: name || 'New Group',
          chatMembers: {
            create: [
              { userId: userId, role: 'ADMIN' },
              ...uniqueMembers.map((id: string) => ({ userId:
id, role: 'MEMBER' }))
            ]
          }
        },
        include: {
          chatMembers: {
            include: {
              user: {
                select: { id: true, username: true, avatar:
true, isOnline: true, lastSeen: true }
              }
            }
          }
        }
      });
      res.status(201).json(chat);
    } else {
      const existingChat = await prisma.chat.findFirst({
        where: {
```

```
          isGroup: false,
          AND: [
            { chatMembers: { some: { userId: userId } } },
            { chatMembers: { some: { userId: partnerId } } }
          ]
        },
        include: {
          chatMembers: {
            include: {
              user: {
                select: { id: true, username: true, avatar:
true, isOnline: true, lastSeen: true }
              }
            }
          }
        }
      });

      if (existingChat) {
        res.json(existingChat);
        return;
      }

      const chat = await prisma.chat.create({
        data: {
          isGroup: false,
          chatMembers: {
            create: [
              { userId: userId },
              { userId: partnerId }
            ]
          }
        },
        include: {
          chatMembers: {
            include: {
              user: {
                select: { id: true, username: true, avatar:
true, isOnline: true, lastSeen: true }
              }
            }
          }
        }
      });
      res.status(201).json(chat);
    }
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

```
export const getChats = async (req: AuthRequest, res: Response)
=> {
  try {
    const userId = req.user?.userId;

    if (!userId) {
      res.status(401).json({ message: 'Unauthorized' });
      return;
    }

    const chats = await prisma.chat.findMany({
      where: {
        chatMembers: {
          some: { userId }
        }
      },
      include: {
        chatMembers: {
          include: {
            user: {
              select: { id: true, username: true, avatar: true,
isOnline: true, lastSeen: true }
            }
          }
        },
        messages: {
          orderBy: { createdAt: 'desc' },
          take: 1
        }
      },
      orderBy: { updatedAt: 'desc' }
    });

    const chatsWithUnread = await Promise.all(chats.map(async
(chat: any) => {
      const unreadCount = await prisma.message.count({
        where: {
          chatId: chat.id,
          senderId: { not: userId },
          statuses: {
            none: {
              userId: userId,
              status: 'READ'
            }
          }
        }
      });
      return { ...chat, unreadCount };
    }));
```

```
    res.json(chatsWithUnread);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

Файл «messageController.ts»

```
import { Response } from 'express';
import prisma from '../utils/prisma';
import { AuthRequest } from '../middleware/authMiddleware';
import { io } from '../app';

const toString = (value: any): string | null => {
  if (!value) return null;
  if (Array.isArray(value)) return value[0];
  return value;
};

export const sendMessage = async (req: AuthRequest, res:
Response) => {
  try {
    const chatId = toString(req.body.chatId);
    const { content, type, fileUrl } = req.body;
    const userId = req.user?.userId;

    if (!userId) return res.status(401).json({ message:
'Unauthorized' });
    if (!chatId) return res.status(400).json({ message: 'chatId
is required' });

    const message = await prisma.message.create({
      data: {
        chatId,
        senderId: userId,
        content,
        type: type || 'TEXT',
        fileUrl: fileUrl || null,
        statuses: {
          create: {
            userId,
            status: 'SENT',
          },
        },
      },
      include: {
        sender: {
          select: { id: true, username: true, avatar: true },
        },
        statuses: true,
      },
    });
```

```
    await prisma.chat.update({
      where: { id: chatId },
      data: { updatedAt: new Date() },
    });

    io.to(chatId).emit('new_message', message);

    return res.status(201).json(message);
  } catch (error) {
    console.error(error);
    return res.status(500).json({ message: 'Server error' });
  }
};

export const getMessages = async (req: AuthRequest, res:
Response) => {
  try {
    const chatId = toString(req.params.chatId);
    const userId = req.user?.userId;

    if (!userId) return res.status(401).json({ message:
'Unauthorized' });
    if (!chatId) return res.status(400).json({ message: 'chatId
is required' });

    const isMember = await prisma.chatMember.findUnique({
      where: {
        chatId_userId: {
          chatId,
          userId,
        },
      },
    });

    if (!isMember) {
      return res.status(403).json({ message: 'Not a member of
this chat' });
    }

    const messages = await prisma.message.findMany({
      where: { chatId },
      include: {
        sender: {
          select: { id: true, username: true, avatar: true },
        },
        statuses: true,
      },
      orderBy: { createdAt: 'asc' },
    });
```

```
    return res.json(messages);
  } catch (error) {
    console.error(error);
    return res.status(500).json({ message: 'Server error' });
  }
};

export const markMessagesAsRead = async (req: AuthRequest, res:
Response) => {
  try {
    const chatId = toString(req.params.chatId);
    const userId = req.user?.userId;

    if (!userId) return res.status(401).json({ message:
'Unauthorized' });
    if (!chatId) return res.status(400).json({ message: 'chatId
is required' });

    const messagesToUpdate = await prisma.message.findMany({
      where: {
        chatId,
        senderId: { not: userId },
        statuses: {
          none: {
            userId,
            status: 'READ',
          },
        },
      },
    });

    await Promise.all(
      messagesToUpdate.map(async (msg) => {
        const existingStatus = await
prisma.messageStatus.findUnique({
          where: {
            messageId_userId: {
              messageId: msg.id,
              userId,
            },
          },
        });

        if (existingStatus) {
          await prisma.messageStatus.update({
            where: { id: existingStatus.id },
            data: { status: 'READ' },
          });
        } else {
          await prisma.messageStatus.create({
            data: {
```

```
              messageId: msg.id,
              userId,
              status: 'READ',
            },
          });
        }
      })
    );

    io.to(chatId).emit('messages_read', { chatId, userId });

    return res.json({ message: 'Messages marked as read' });
  } catch (error) {
    console.error(error);
    return res.status(500).json({ message: 'Server error' });
  }
};
```

Файл «userController.ts»

```
export const getAllUsers = async (req: AuthRequest, res:
Response) => {
  try {
    const userId = req.user?.userId;
    const users = await prisma.user.findMany({
      where: {
        id: { not: userId }
      },
      select: {
        id: true,
        username: true,
        avatar: true,
        bio: true,
        relationshipStatus: true,
        isOnline: true,
        lastSeen: true
      }
    });
    res.json(users);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
import { Response } from 'express';
import prisma from '../utils/prisma';
import { AuthRequest } from '../middleware/authMiddleware';

export const searchUsers = async (req: AuthRequest, res:
Response) => {
  try {
    const { query } = req.query;
    const userId = req.user?.userId;
```

```
    if (!query || typeof query !== 'string') {
      res.status(400).json({ message: 'Search query is required'
});
      return;
    }

    const users = await prisma.user.findMany({
      where: {
        username: {
          contains: query
        },
        NOT: {
          id: userId
        }
      },
      select: {
        id: true,
        username: true,
        avatar: true,
        bio: true,
        relationshipStatus: true,
        isOnline: true,
        lastSeen: true
      },
      take: 10
    });

    res.json(users);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};

export const getProfile = async (req: AuthRequest, res:
Response) => {
  try {
    const userId = req.user?.userId;
    const user = await prisma.user.findUnique({
      where: { id: userId },
      select: {
        id: true,
        username: true,
        email: true,
        avatar: true,
        bio: true,
        relationshipStatus: true,
        isOnline: true,
        lastSeen: true,
        notificationsEnabled: true,
```

```
        createdAt: true
      }
    });
    res.json(user);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};

export const updateProfile = async (req: AuthRequest, res:
Response) => {
  try {
    const userId = req.user?.userId;
    const { username, avatar, bio, relationshipStatus,
notificationsEnabled } = req.body;

    const user = await prisma.user.update({
      where: { id: userId },
      data: {
        username,
        avatar,
        bio,
        relationshipStatus,
        notificationsEnabled
      },
      select: {
        id: true,
        username: true,
        email: true,
        avatar: true,
        bio: true,
        relationshipStatus: true,
        isOnline: true,
        lastSeen: true,
        notificationsEnabled: true
      }
    });
    res.json(user);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: 'Server error' });
  }
};
```

Приложение В

(обязательное)

Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети

Презентация

Тема работы:
Корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети

Автор работы: студент группы 22ВВП1 Коннов А.Д.

Научный руководитель: к.т.н., доцент кафедры ВТ ПГУ Слепцов Н.В.

Пенза 2026

Цель и объект работы

Цель работы

Разработка корпоративного мессенджера, позволяющего взаимодействовать сотрудникам в локальной сети

Объект работы

Приложение, выполняющее следующие функции:

– авторизация пользователей;

– возможность вести диалог с другими пользователями;

– наличие удобного графического интерфейса;

– обеспечение безопасности данных.

Постановка задачи

В процессе разработки должны быть решены следующие задачи:

– выбор и настройка инструментов разработки;

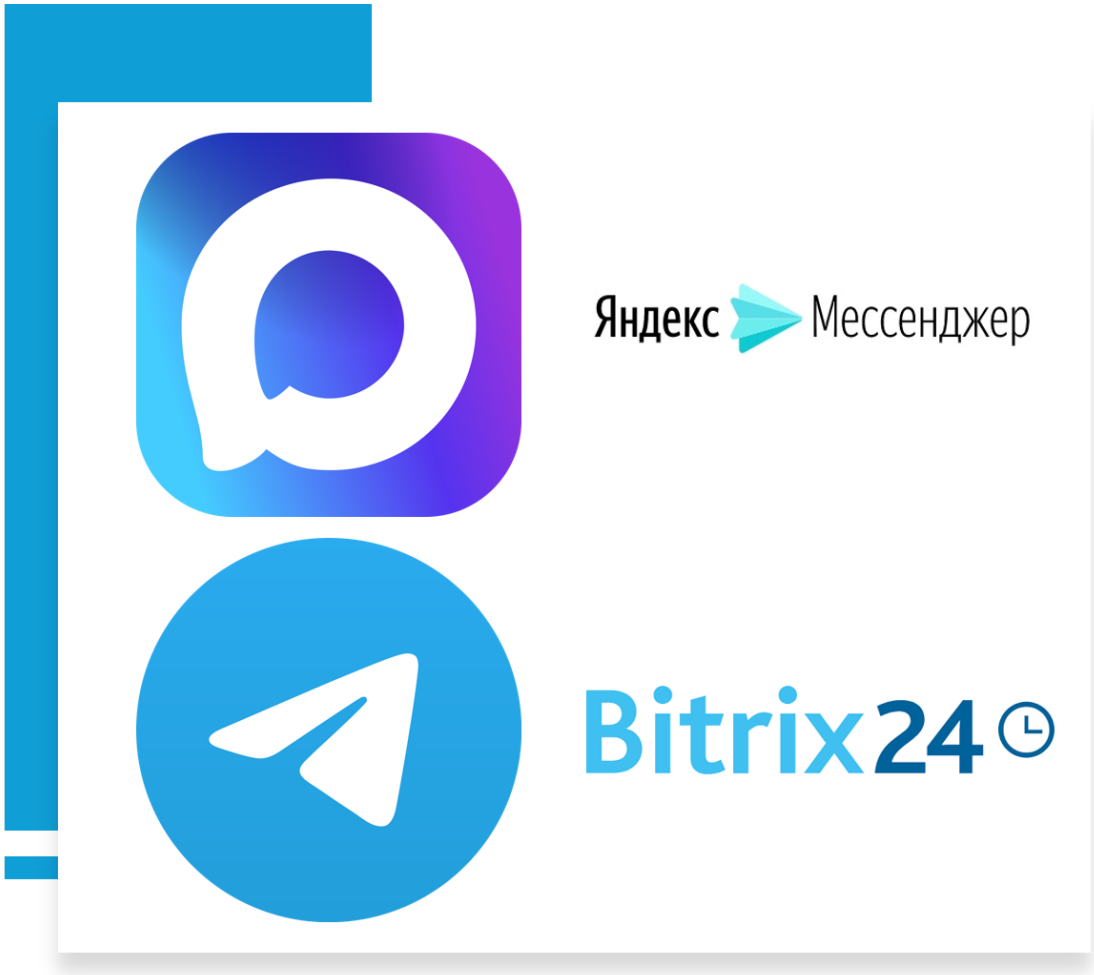
– создание удобного и понятного графического интерфейса;

– разработка базы данных;

– добавление возможности кастомизации профиля пользователя;

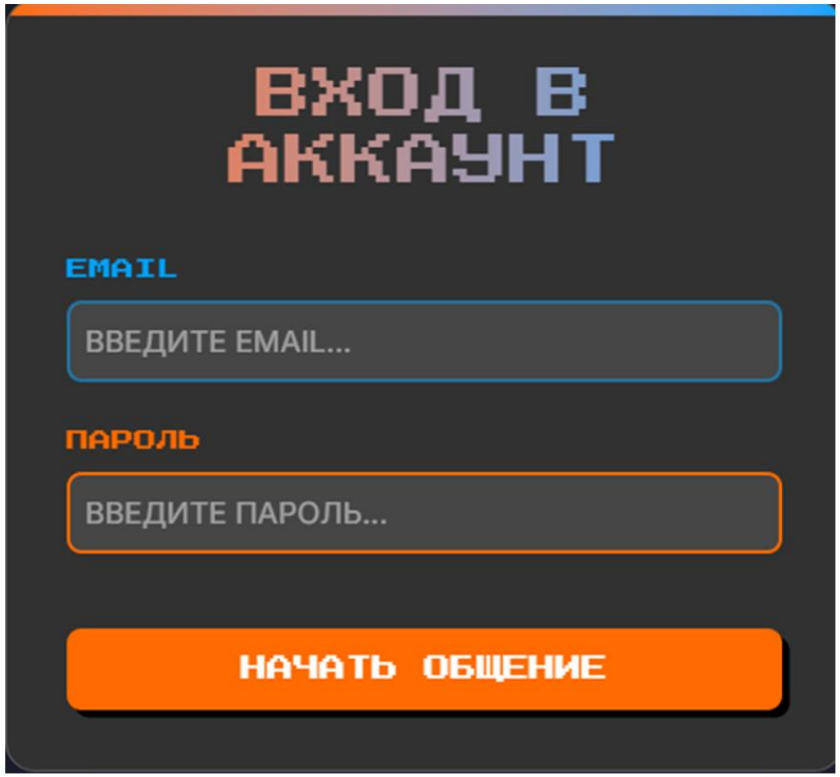
– обеспечение безопасности данных путём шифрования;

– создание условий для работы в локальной сети.

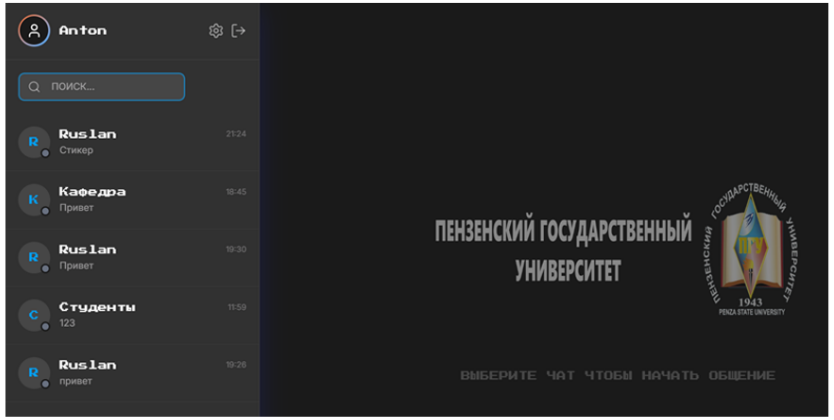


Аналоги

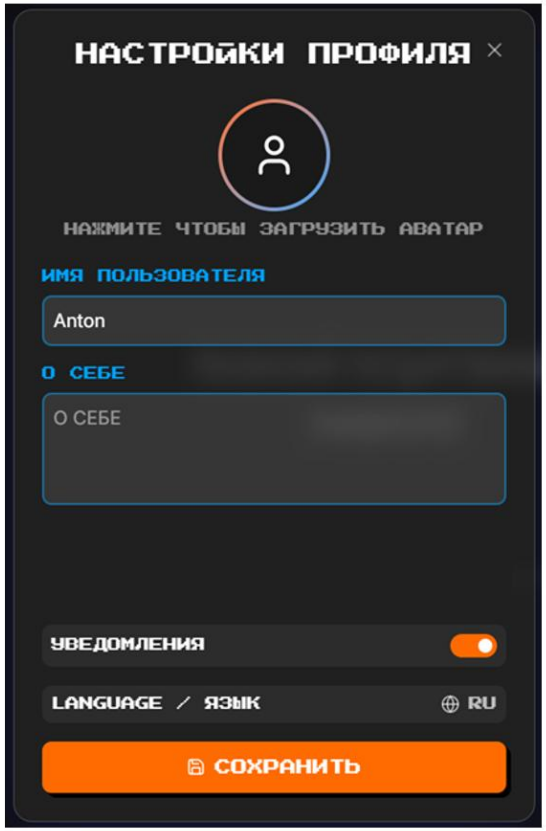
III Окно авторизации



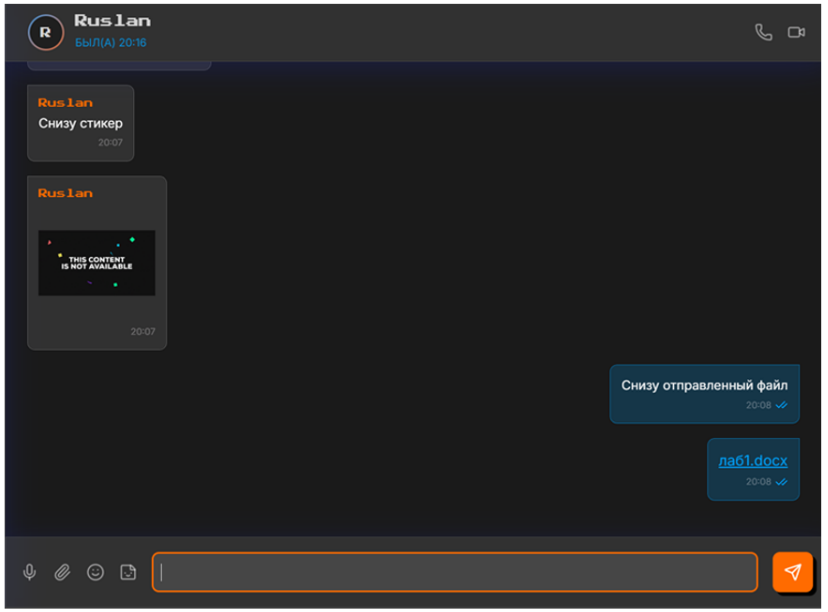
III Главное окно чата



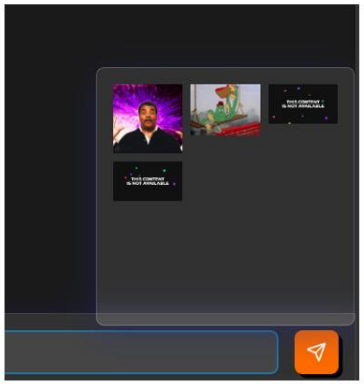
Профиль

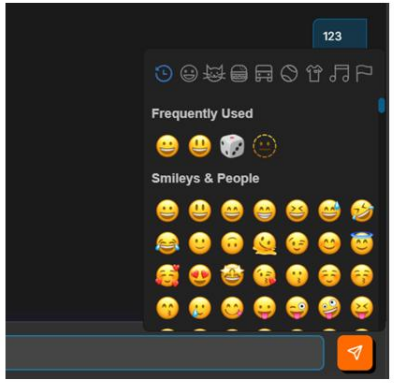

НАСТРОЙКИ ПРОФИЛЯ
НАЖМИТЕ ЧТОБЫ ЗАГРУЗИТЬ АВАТАР
ИМЯ ПОЛЬЗОВАТЕЛЯ
Anton
О СЕБЕ
О СЕБЕ
УВЕДОМЛЕНИЯ
LANGUAGE / ЯЗЫК
RU
СОХРАНИТЬ

III Окно диалога

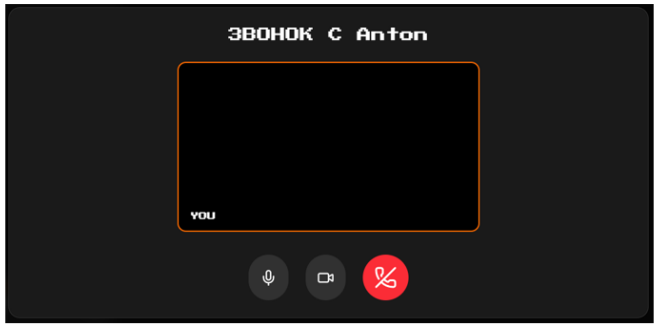


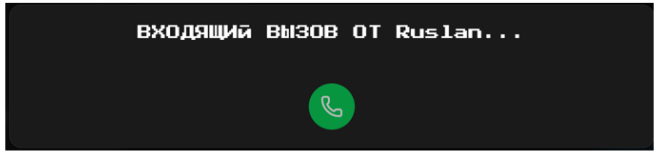
Набор эмодзи и стикеров



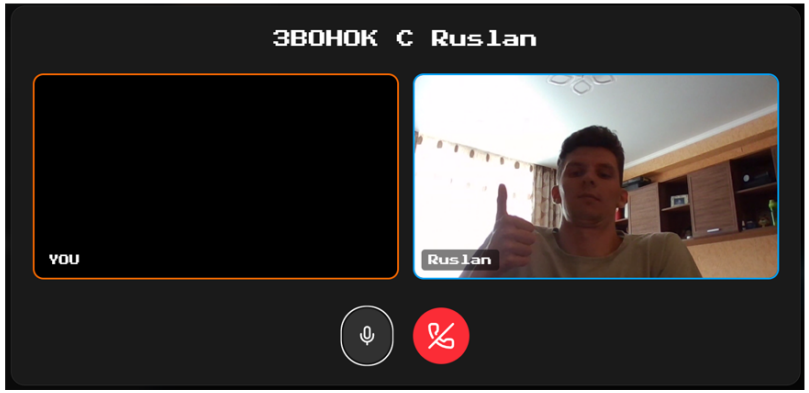


III Входящий звонок



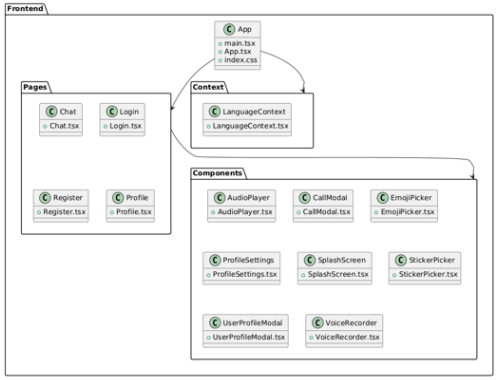


III Активный звонок



Стек технологий

III Архитектура приложения



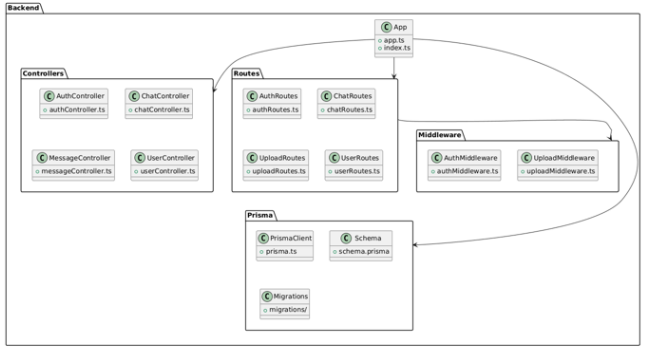


Диаграмма вариантов использования

Диаграмма последовательностей

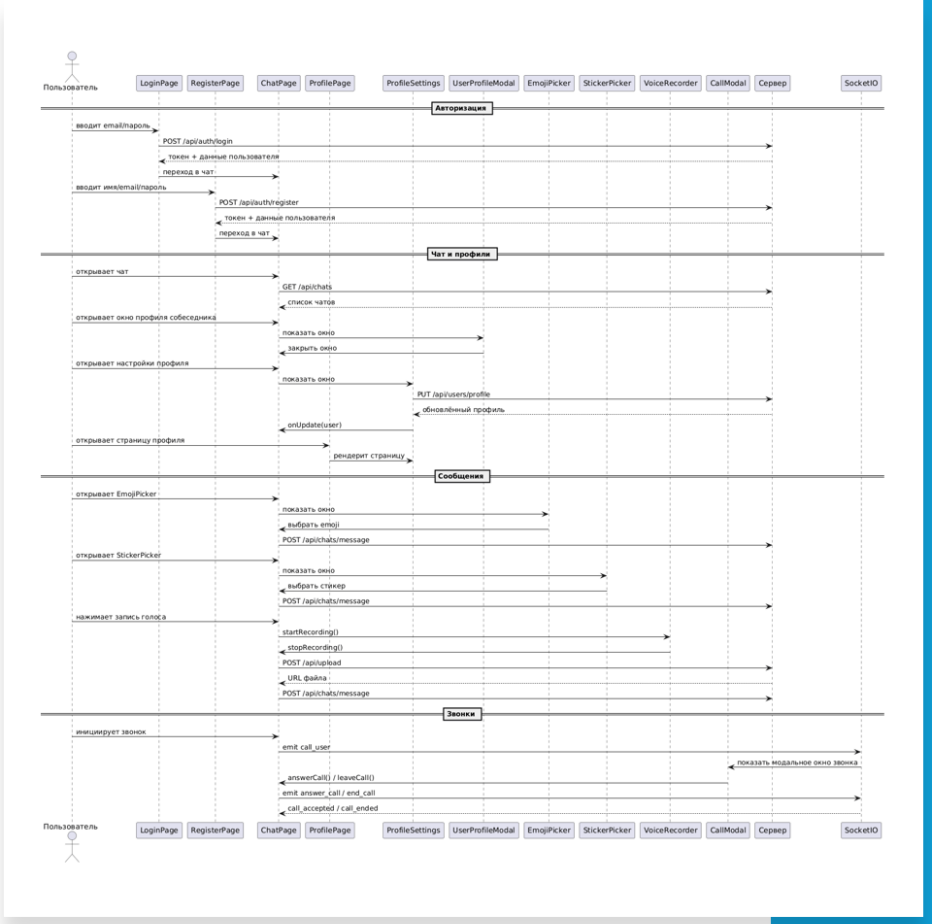


Диаграмма классов

Заключение

В ходе выполнения выпускной квалификационной работы был разработан корпоративный мессенджер защищённого взаимодействия сотрудников в локальной сети на языке *TypeScript*. Графический интерфейс был реализован с помощью библиотеки *React*.

Разработанное приложение позволяет безопасно обмениваться сообщениями и файлами в реальном времени между пользователями, а также совершать аудио- и видеозвонки с использованием удобного и понятного графического интерфейса.

Одним из достоинств разработанного приложения является работа в условиях локальной сети, позволяющая достичь высокого уровня безопасности данных и исключить риск утечки.

В дальнейшем разработанное приложение можно усовершенствовать, добавив больше возможностей для кастомизации окна, улучшить имеющиеся системы до уровня ведущих мессенджеров. Реализовать более гибкую настройку аккаунта, добавить больше вариантов смены языка интерфейса.

Спасибо за внимание!