

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

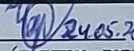
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

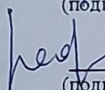
на тему

**«КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ
ОБМЕНА СООБЩЕНИЯМИ НА БАЗЕ
ФРЕЙМВОРКА QT»**

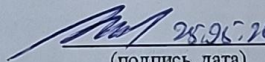
Студент

 24.05.26 Черников Кирилл Александрович
(подпись, дата) (ФИО полностью)

Руководитель

 24.05.26 Леонова Т.Ю.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

 28.05.26 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

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

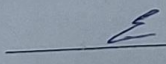
Заведующий кафедрой

 М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

11 июня 2026 г., № 26)

Секретарь ГЭК

 Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2026

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

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»



«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«07» апреля 2026 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Черников Кирилл Александрович гр. 22ВВП1 факультета ФВТ

2. Тема работы клиент-серверное приложение обмена сообщениями на базе фреймворка qt

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

3. Руководитель бакалаврской работы Леонова Т.Ю.

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

разработка клиент-серверного приложения для обмена сообщениями на базе фреймворка Qt, предназначенного для организации личного и группового общения пользователей в локальных и глобальных сетях. Разрабатываемая система должна обеспечивать независимость от сторонних облачных сервисов, централизованное хранение данных и полный контроль над инфраструктурой обмена сообщениями.

Исходные данные включают необходимость реализации серверной части для обработки подключений пользователей, хранения истории переписки и маршрутизации сообщений, а также клиентского приложения с графическим. Система должна поддерживать регистрацию и авторизацию пользователей, обмен текстовыми сообщениями, передачу файлов и голосовых сообщений, создание групповых чатов и хранение истории сообщений.

Разрабатываемое приложение решает задачу создания автономной коммуникационной среды, позволяющей обеспечить безопасность передаваемых данных, повысить надёжность внутреннего обмена информацией и исключить зависимость от коммерческих облачных мессенджеров. Использование клиент-серверной архитектуры позволяет централизованно управлять пользователями и данными, а применение фреймворка Qt обеспечивает удобство сопровождения программного продукта.

5. Перечень подлежащих разработке вопросов

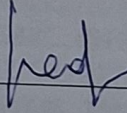
Требуется провести анализ предметной области: изучить существующие

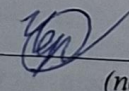
мессенджеры и системы обмена сообщениями, выявить их преимущества и недостатки.
 Определить требования пользователей к функциональности приложения.
 Выбрать язык программирования, среду разработки и технологии сетевого взаимодействия. Изучить возможности фреймворка Qt для создания приложений, раб
 графическим интерфейсом, сетевыми соединениями и базами данных.
 Требуется определить архитектуру приложения, спроектировать взаимодействие
 клиентской и серверной частей, разработать структуру базы данных и протокол обмен
 сообщениями.
 Дополнительно требуется изучить функциональные возможности среды разработки
 Microsoft Visual Studio и инструментов Qt для разработки, отладки и тестирования.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подп руковод
1	Изучение предметной области	32	07.04.2026 – 11.04.2026	Leo
2	Анализ требований	32	12.04.2026 – 15.04.2026	Leo
3	Создание макета графического интерфейса	30	16.04.2026 – 19.04.2026	Leo
4	Выбор технологий, применяемых при проектировании и разработке	40	20.04.2026 – 24.04.2026	Leo
5	Реализация графического интерфейса	66	25.04.2026 – 03.05.2026	Leo
6	Разработка основной части программы	70	04.05.2026 – 13.05.2026	Leo
7	Оформление пояснительной записки	54	14.05.2026 – 21.05.2026	Leo

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  07.04.2026 Леонова Т.Ю.
(подпись, дата)

Задание к исполнению принял студент  07.04.2026 Черников К.А.
(подпись, дата)

Работу к защите допустить
 Декан факультета



(подпись, дата)

Фионова Л.Р.

13.06.2026

ATK

або

ена

ДП

вод

e

ρ

Содержание

Перечень сокращений и обозначений.....	8
1 Клиент-серверное приложение обмена сообщениями на базе фреймворка Qt .	11
1.1 Анализ предметной области	11
1.1.1 Анализ темы	11
1.1.2 Анализ существующих аналогов	12
1.1.3 Постановка задачи	15
1.2 Выбор средств разработки приложения	17
1.2.1 Выбор языка программирования.....	17
1.2.2 Выбор среды разработки.....	18
1.2.3 Выбор способа обмена данными.....	19
1.2.4 Настройка среды разработки	19
1.3 Проектирование приложения	28
1.3.1 Паттерн MVP.....	28
1.3.2 Архитектура приложения.....	30
1.3.3 Проектирование базы данных	30
1.4 Разработка клиентского приложения.....	39
1.4.1 Инициализация пользовательского интерфейса.....	39
1.4.2 Сетевое взаимодействие и авторизация.....	40
1.4.3 Отображение и управление чатами.....	42
1.4.4 Отправка и прием текстовых сообщений.....	43
1.4.5 Передача файлов и голосовых сообщений.....	44
1.4.6 Управление группами и профилями	46
1.4.7 Вспомогательные классы	47
1.5 Разработка серверного приложения.....	48

1.5.1	Инициализация сервера.....	48
1.5.2	Обработка сетевых соединений и протокол обмена	49
1.5.3	Обработка сообщений	50
1.5.4	Управление чатами и группами.....	50
1.5.5	Работа с базой данных	51
1.6	Тестирование	51
1.7	Руководство пользователя.....	53
2	Безопасность жизнедеятельности.....	70
2.1	Общие положения	70
2.2	Опасные и вредные факторы при работе за ПК.....	70
2.3	Требования к организации рабочего места	70
2.4	Электробезопасность	71
2.5	Режим труда и отдыха	72
2.6	Пожарная безопасность.....	73
	Заключение	74
	Список использованных источников	75
	Приложение А – UML – диаграммы	76
	Приложение Б – Листинг программы	84
	Приложение В – Презентация.....	113

Перечень сокращений и обозначений

БД – база данных

СУБД – система управления базами данных

Фреймворк (Framework) – программная основа или набор инструментов, который предоставляет структуру и готовые компоненты для разработки программного обеспечения

FK (Foreign key) – внешний ключ, понятие теории реляционных баз данных

ID (Identifier) – уникальный номер объекта, идентификатор

IDE (Integrated development environment) – интегрированная среда разработки

JSON (JavaScript Object Notation) – текстовый формат обмена данными

MIME (Multipurpose Internet Mail Extensions) – многоцелевые расширения интернет-почты, стандарт, описывающий передачу различных типов данных

MVC (Model-View-Controller) – шаблон проектирования, разделяющий приложение на модель, представление и контроллер

MVP (Model-View-Presenter) – шаблон проектирования, производный от MVC, разделяет приложение на модель, представление и представителя

OpenSSL – свободная библиотека с открытым исходным кодом, реализующая протоколы SSL и TLS

PK (Primary key) – первичный ключ, понятие теории реляционных баз данных

Qt – кроссплатформенный фреймворк для разработки приложений

SQLite – встраиваемая реляционная система управления базами данных

SSL (Secure Sockets Layer) – криптографический протокол, обеспечивающий защищённое соединение

TCP (Transmission Control Protocol) – протокол управления передачей

TLS (Transport Layer Security) – протокол защиты транспортного уровня

Введение

Необходимость в надёжном, приватном и независимом от внешних сервисов канале связи встречается как в повседневной жизни, так и в учебной, корпоративной и профессиональной среде. В зависимости от требований к безопасности, скорости передачи данных и уровня контроля над инфраструктурой традиционные облачные мессенджеры перестают быть универсальным решением. Темы цифрового суверенитета и защиты личной информации набирают обороты не только на уровне обычных пользователей, но и в масштабах целых организаций. В связи с этим появляются системные проблемы, будь то риск утечки данных через сторонние серверы, внезапное изменение условий использования или полная потеря доступа к сервису из-за региональных ограничений.

Чтобы избежать данных проблем, правильным решением будет разработка программы, которая позволит развернуть собственный мессенджер с клиент-серверной архитектурой, снизить зависимость от внешних посредников и обеспечить полный контроль над обменом сообщениями. Пользователь при работе с такой системой должен рассматриваться как равноправный участник защищённого информационного пространства. Основной задачей управления данной системой является обеспечение надёжной и безопасной доставки сообщений, аутентификации пользователей и ведения истории переписки без участия третьих сторон.

Эффективность обратной связи между участниками рабочих групп напрямую зависит от доступности, конфиденциальности и целостности передаваемых данных. В условиях, когда коммерческие мессенджеры массово внедряют модели монетизации на основе анализа переписки, возникает острая потребность в автономной инфраструктуре. Существующая практика использования публичных облачных сервисов для внутренней коммуникации является уязвимым местом, ставящим под угрозу как персональные данные, так и коммерческую тайну.

Целью данной работы является разработка инструмента, автоматизирующего процесс обмена сообщениями в закрытой среде с полным контролем над данными.

					ПГУ 1.090301.25.001 ПЗ	Лист
						9
Изм.	Лист	№ докум.	Подпись	Дата		

Создаваемое приложение реализует клиент-серверную архитектуру с использованием кроссплатформенного фреймворка Qt, что позволяет развернуть сервер на собственном оборудовании (в локальной сети организации или на частном хостинге), а клиентские приложения запускать на Windows или Linux. Такой подход обеспечивает унификацию коммуникационной среды, исключает утечку данных за пределы контролируемой инфраструктуры и предоставляет пользователям привычный интерфейс, свободный от ограничений коммерческих аналогов.

					ПГУ 1.090301.25.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

1 Клиент-серверное приложение обмена сообщениями на базе фреймворка Qt

1.1 Анализ предметной области

1.1.1 Анализ темы

Процесс цифровой коммуникации в современных учебных и рабочих коллективах представляет собой сложную информационную деятельность, в рамках которой задействованы проприетарные облачные сервисы, функционирующие круглосуточно с высокой нагрузкой на серверные мощности зарубежных центров обработки данных. В условиях распространения удалённой работы и гибридных форматов обучения количество одновременно взаимодействующих участников исчисляется десятками и сотнями, а стандартные средства облачных мессенджеров, ориентированные на массового пользователя и не дающие администратору возможности полного контроля над серверной частью, создают критическую информационную уязвимость, повышая риск утечки служебной переписки или коммерческой тайны из-за неосторожного обращения с данными, использования сторонних плагинов или банальной передачи прав доступа коммерческому мессенджеру без согласования.

Разрабатываемое клиент-серверное приложение решает задачу создания полностью автономной, независимой от внешних облачных провайдеров коммуникационной среды, обеспечивая контроль над всеми этапами передачи и хранения сообщений за счёт прямого взаимодействия между сервером, размещённым под управлением владельца (в локальной сети предприятия, университета или на частном хостинге), и клиентскими приложениями, работающими на операционных системах пользователей. Накопление истории переписки в едином хранилище позволяет не только оперативно обмениваться текстовыми сообщениями, но и формировать архив рабочих обсуждений, на основе которого возможно анализировать эффективность групповых взаимодействий, восстанавливать ход принятия решений и планировать аудит коммуникаций.

					ПГУ 1.090301.25.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

С точки зрения организации, внедрение подобной системы обеспечивает переход от зависимой модели коммуникации, при которой критическая информация хранится на серверах третьих лиц, к суверенной стратегии, направленной на полный контроль над всеми информационными потоками. Кроме того, создание собственного мессенджера даёт значительные технические и экономические преимущества: кроссплатформенность позволяет использовать единую кодовую базу для настольных операционных систем, что снижает затраты на разработку и поддержку; нативный интерфейс обеспечивает высокую отзывчивость и привычный пользовательский опыт; система авторизации и поддержка групповых чатов с неограниченным числом участников закрывают типовые потребности рабочих коллективов.

Таким образом, создание собственного клиент-серверного мессенджера на Qt позволяет получить современный, безопасный и полностью управляемый инструмент общения, свободный от ограничений и рисков, присущих коммерческим облачным решениям.

1.1.2 Анализ существующих аналогов

Среди аналогов программ, реализующих функции личного и группового обмена сообщениями можно выделить следующие решения:

- Telegram;
- Discord;
- TeamSpeak.

Рассмотрим возможности мессенджера Telegram.

Telegram – это современный быстрый и безопасный мессенджер, который объединяет в себе функции обмена текстовыми и голосовыми сообщениями, голосовых и видеозвонков, создания каналов и групп. Приложение поддерживает обширный функционал ботов, передачу файлов размером до 2 ГБ, а также синхронизацию сообщений между всеми устройствами пользователя. Общий вид данной программы представлен на рисунке 1.

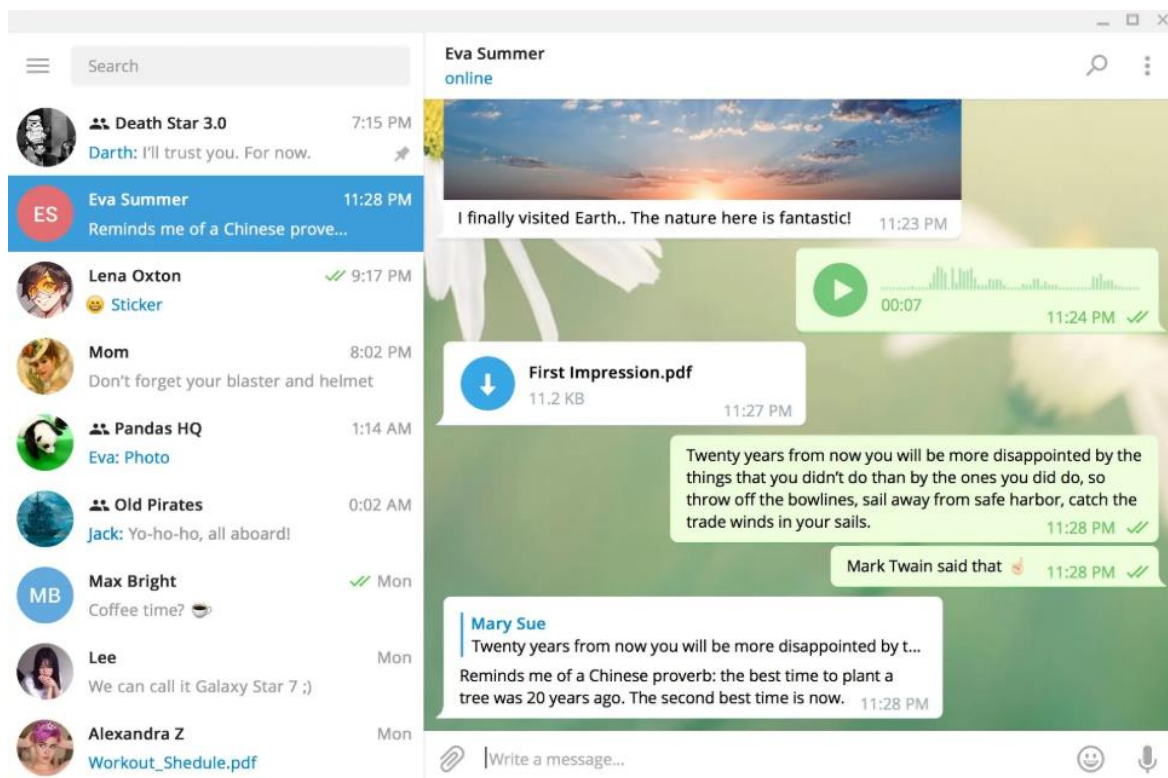


Рисунок 1 – Мессенджер «Telegram»

Discord – это многоплатформенное приложение для видеосвязи, голосового и текстового общения. Поддерживает голосовые каналы с низкой задержкой, текстовые чаты, серверы с гибкой системой ролей и прав и трансляцию экрана. Интерфейс программы представлен на рисунке 2.

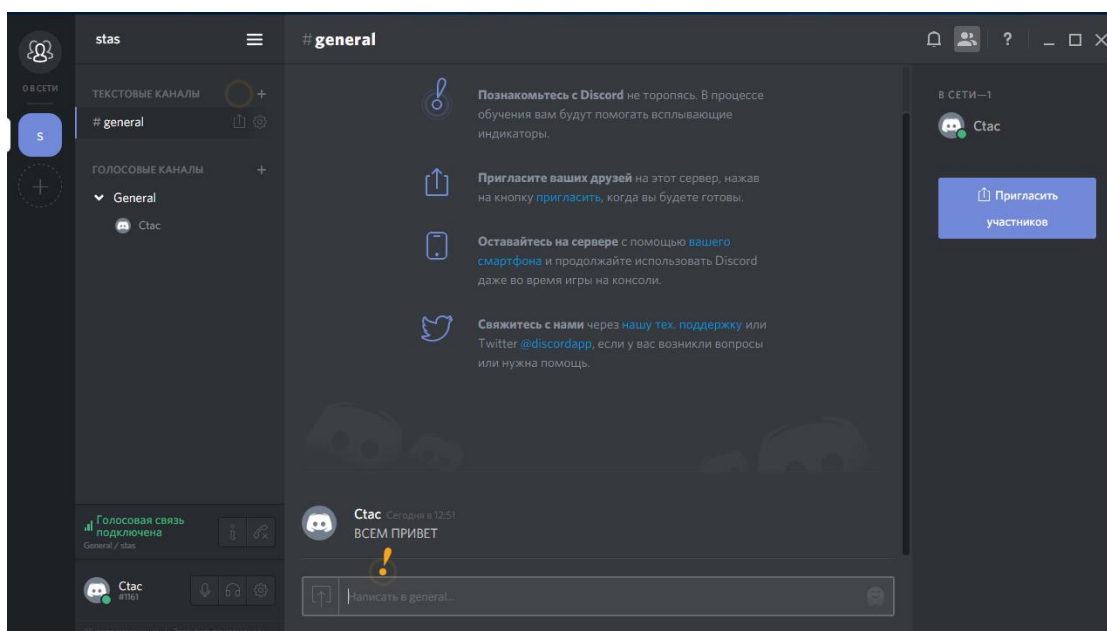


Рисунок 2 – Discord

TeamSpeak – программа, предназначенная для голосового общения, широко используемая в киберспорте, корпоративной среде и игровых сообществах. Главной особенностью TeamSpeak является высокое качество звука и низкая задержка. Приложение поддерживает создание серверов с системой каналов, шифрование трафика. В последних версиях добавлены функции обмена текстовыми сообщениями, передача файлов и современный интерфейс. Интерфейс программы представлен на рисунке 3.

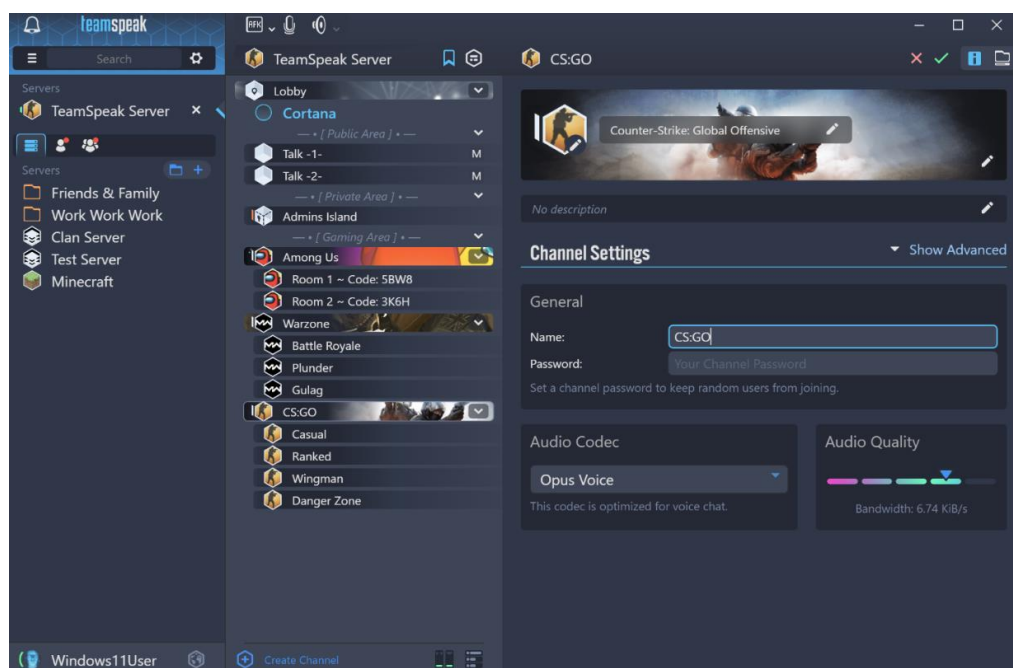


Рисунок 3 – TeamSpeak

Сравнения преимуществ и недостатков представлены в таблице 1.

Таблица 1 – Сравнительная таблица аналогов

Мессенджер	Преимущества	Недостатки
Telegram	Скорость работы, удобный интерфейс, поддержка больших групп и каналов	Полная зависимость от облачных серверов компании, нет возможности создания собственного сервера

Мессенджер	Преимущества	Недостатки
Discord	Развитая система сообществ, голосовые и видеочаты, гибкая система ролей и каналов	Высокая ресурсоёмкость, централизованная модель, нет возможности создания собственного выделенного сервера
TeamSpeak	Возможность создания собственного выделенного сервера, настраиваемое качество голосовой связи, возможность использования в локальных сетях	Отсутствие синхронизации сообщений на серверах, ограниченные возможности чатов

На основании проведённого анализа выявлен потенциал для разработки нового решения. Существующие аналоги либо полностью облачные и закрытые, либо ориентированы на преимущественно голосовое взаимодействие. Разрабатываемое приложение направлено на устранение выявленных ограничений и предлагает клиент-серверную архитектуру с возможностью полного контроля над данными и инфраструктурой.

1.1.3 Постановка задачи

Требуется разработать клиент-серверное приложение для личного и группового обмена сообщениями на базе фреймворка Qt, обеспечивающего обмен сообщениями в локальных и глобальных сетях с возможностью полного контроля над данными. Для достижения поставленной цели решаются следующие задачи, разделённые по компонентам приложения.

Функции клиентской части:

- реализация графического интерфейса с удобной навигацией (список чатов, окно переписки, панель контактов и поиска);
- авторизация пользователей (регистрация и вход в систему);
- создание и участие в личных и групповых чатах;
- отправка и получение текстовых сообщений;
- отправка и получение голосовых сообщений;
- удаление сообщений;
- загрузка и отображение истории переписки при открытии чата;
- уведомления о новых сообщениях.

Функции серверной части:

- обработка входящих подключений от клиентов;
- управление авторизацией пользователей;
- хранение данных пользователей, списков чатов и истории сообщений в базе данных;
- маршрутизация сообщений между участниками чатов;
- создание, редактирование и управление групповыми чатами;
- обеспечение доставки сообщений;
- обеспечение удаления сообщений;
- обработка ошибок соединения.

Технические требования:

- операционная система: Windows 10 и более поздние версии;
- оперативная память: не менее 512 МБ ОЗУ;
- дисковое пространство: не менее 512 МБ ПЗУ;
- процессор: Intel Core i3-8100 3,6 GHz.

На рисунке 4 изображена диаграмма вариантов использования приложения клиента.



Рисунок 4 – Диаграмма прецедентов

1.2 Выбор средств разработки приложения

1.2.1 Выбор языка программирования

В качестве основного языка программирования для разработки приложения выбран C++.

C++ является мощным, высокопроизводительным языком программирования общего назначения. Он широко используется для создания высоконагруженных систем, включая клиент-серверные приложения реального времени (мессенджеры, системы обмена сообщениями и т.д.) [1].

Язык сочетает в себе возможности объектно-ориентированного, процедурного программирования, а также поддерживает лямбда-выражения, умные указатели и многопоточность. Благодаря высокой производительности и низкому уровню абстракции C++ идеально подходит для реализации сетевого взаимодействия, работы с базами данных и графического интерфейса, требующих минимальных задержек при передаче сообщений [2].

Приложения на C++ с использованием Qt получают преимущества от кроссплатформенности: один и тот же код может быть скомпилирован и запущен под Windows, Linux и macOS без значительных изменений. Qt 6 предлагает современный, высокопроизводительный графический движок, а также мощные классы для работы с сетью, JSON и базами данных [3].

1.2.2 Выбор среды разработки

В качестве среды разработки выбрана Microsoft Visual Studio 2026 – мощная интегрированная среда (IDE), позволяющая проектировать, отлаживать, тестировать и развертывать приложения в едином рабочем пространстве. При работе с фреймворком Qt данная IDE обеспечивает глубокую интеграцию через официальное расширение Qt Visual Studio Tools [3].

Visual Studio 2026 предоставляет широкий набор инструментов для разработки на языке C++ и Qt:

- полная поддержка C++20/23 стандартов;
- визуальное проектирование интерфейсов через Qt Designer (работа с файлами формата .ui);
- средства отладки клиент-серверного взаимодействия;
- встроенная поддержка систем контроля версий (Git).

1.2.3 Выбор способа обмена данными

JSON (Java Script Object Notation) – стандартный текстовый формат для хранения и передачи структурированных данных. Он основан на синтаксисе объекта в JavaScript, но не привязан к нему [1].

JSON-формат хранения данных имеет ряд преимуществ:

- простой и легко читаемый синтаксис, который понятен и компьютеру, и человеку;
- JSON-файлы весят меньше, чем файлы других форматов, и загружаются быстрее;
- для работы с JSON можно использовать почти любой популярный язык программирования;
- JSON можно использовать для хранения массивов, упорядоченных списков и коллекций пар «ключ — значение»;
- в JSON можно включить метаданные, которые описывают представленные данные, упрощая работу с ними.

В данной бакалаврской работе JSON-формат будет использоваться для обмена данными между клиентской и серверной частями разрабатываемого приложения.

Для защиты сетевого обмена JSON-сообщения будут передаваться через защищённое TLS-соединение.

1.2.4 Настройка среды разработки

Необходимо установить среду разработки Microsoft Visual Studio 2026. Сайт Visual Studio представлен на рисунке 5.

Страница загрузки Visual Studio содержит подробное описание функциональных возможностей среды, технические требования к системе, архив предыдущих версий и элементы управления для загрузки установщика. Для начала работы необходимо нажать кнопку скачивания (рисунок 6).

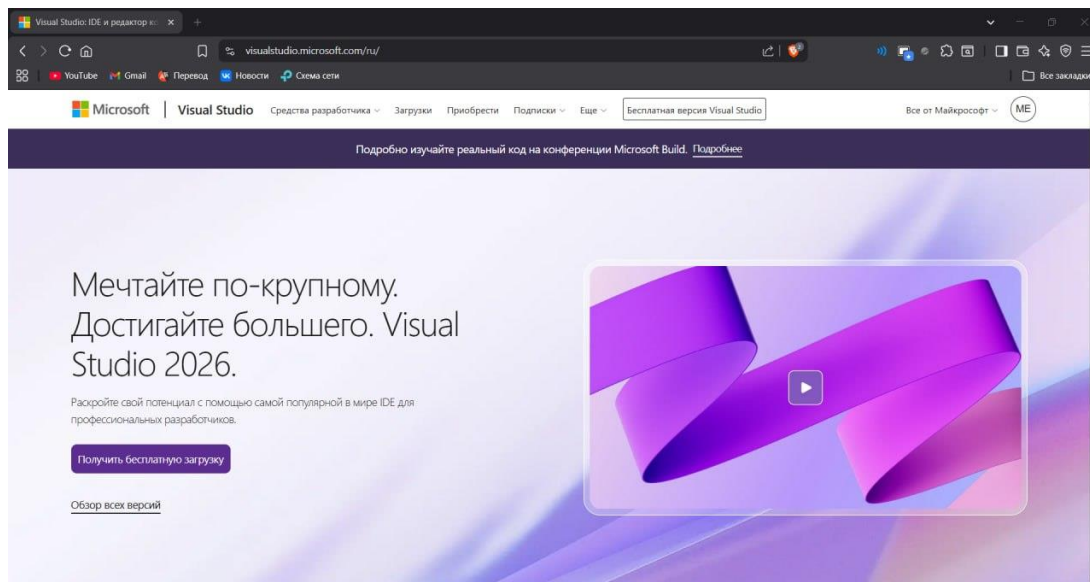


Рисунок 5 – Главная страница сайта Visual Studio

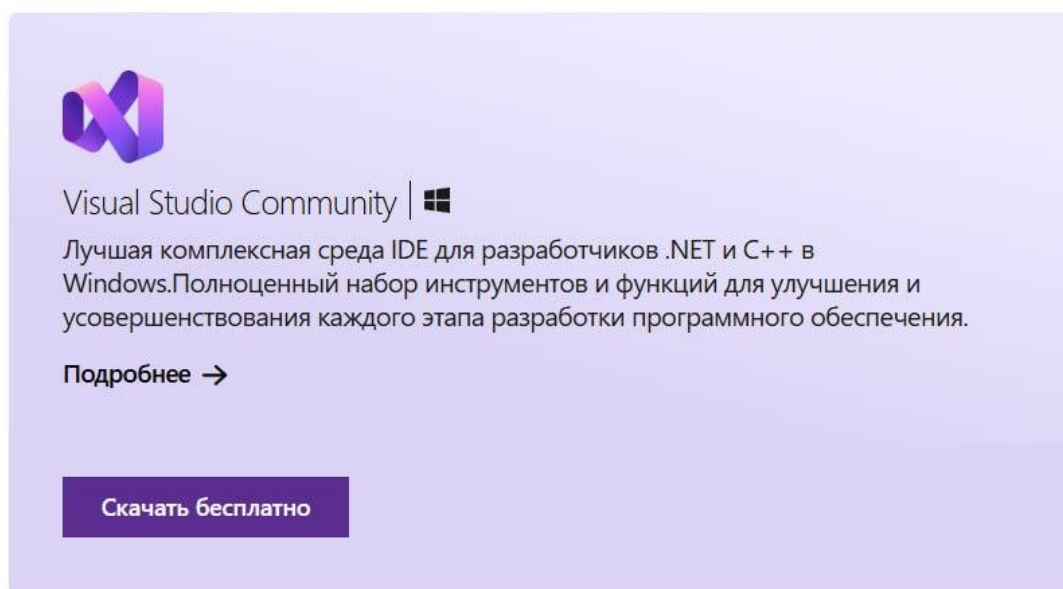


Рисунок 6 – Кнопка загрузки установщика

После перехода на страницу загрузки начинается скачивание исполняемого файла (рисунок 7), который затем необходимо запустить (рисунок 8).

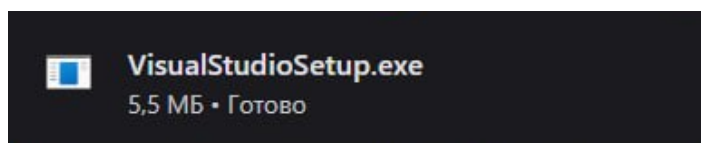


Рисунок 7 – Скачивание установщика

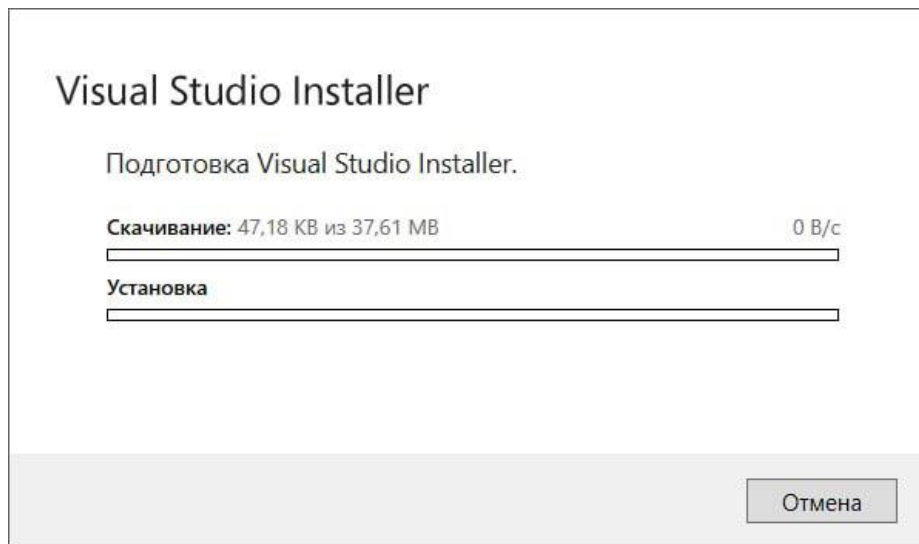


Рисунок 8 – Запуск установщика

В окне выбора компонентов следует выбрать «Разработка классических приложений на C++» (рисунок 9). При первом запуске Visual Studio 2026 открывается окно приветствия (рисунок 10), в котором для первичной настройки выбираем вариант «Продолжить без кода».

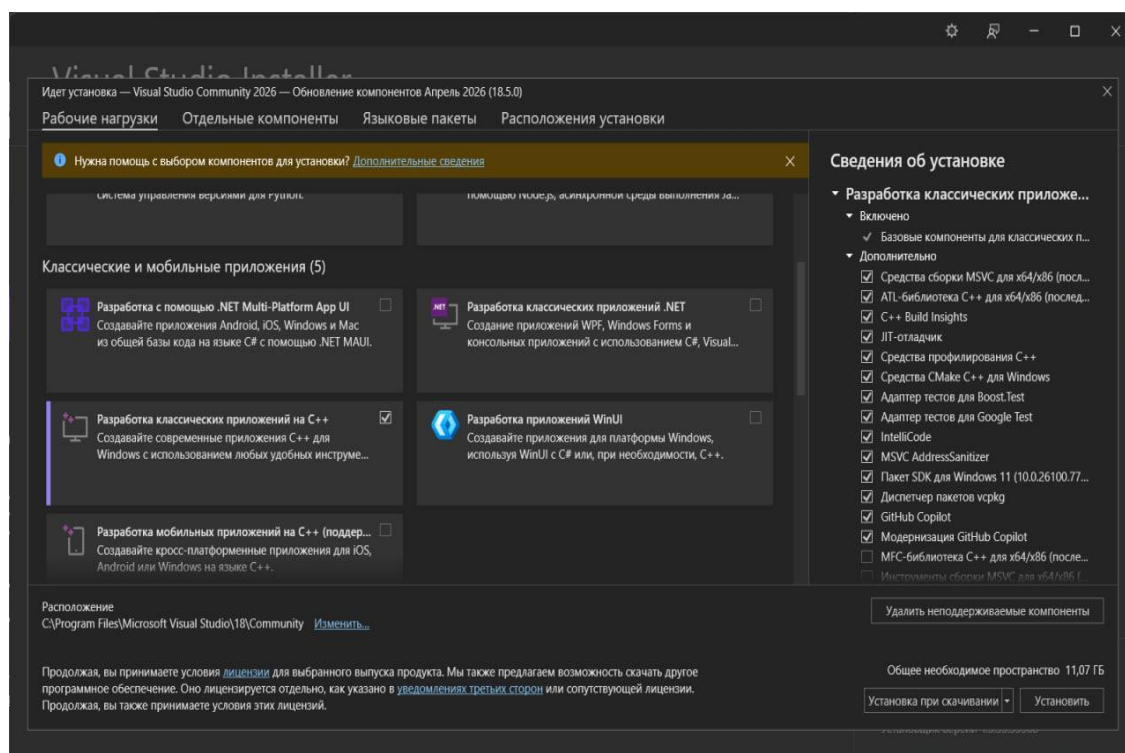


Рисунок 9 – Выбор компонентов в Visual Studio Installer

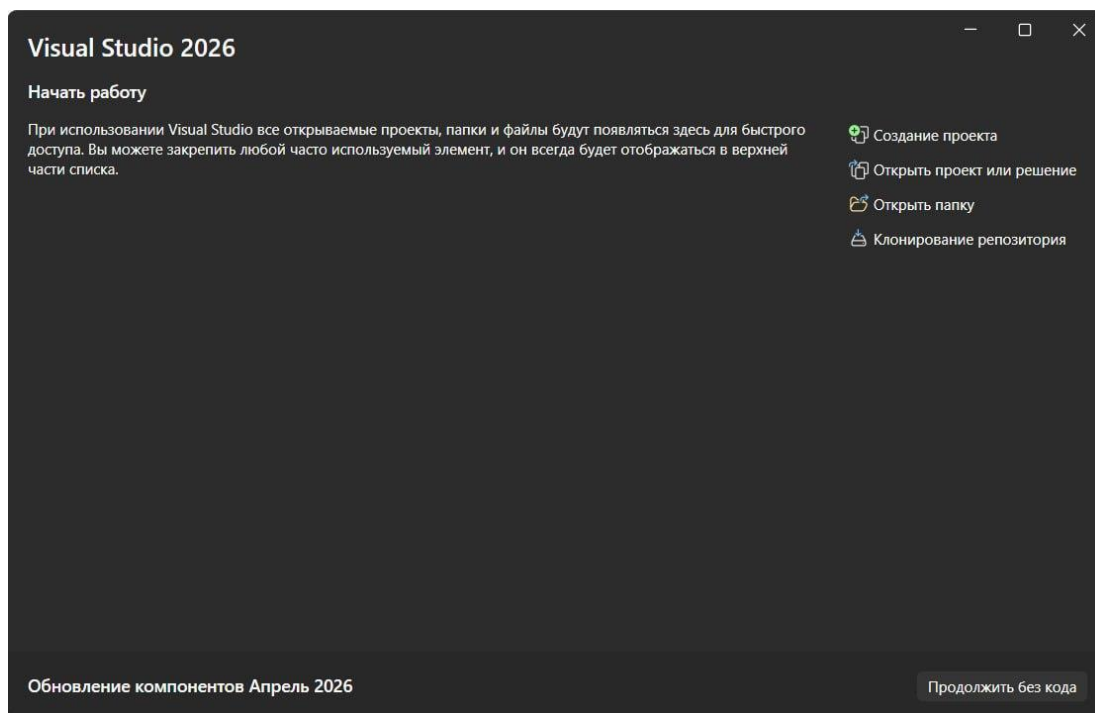


Рисунок 10 – Начальное окно запуска среды

Для разработки программ на базе Qt 6 необходимо установить соответствующее расширение [3]. Для этого используется меню «Расширения» (рисунок 11). В окне управления расширениями следует найти и установить «Qt Visual Studio Tools» (рисунок 12). Результат успешной установки показан на рисунке 13.

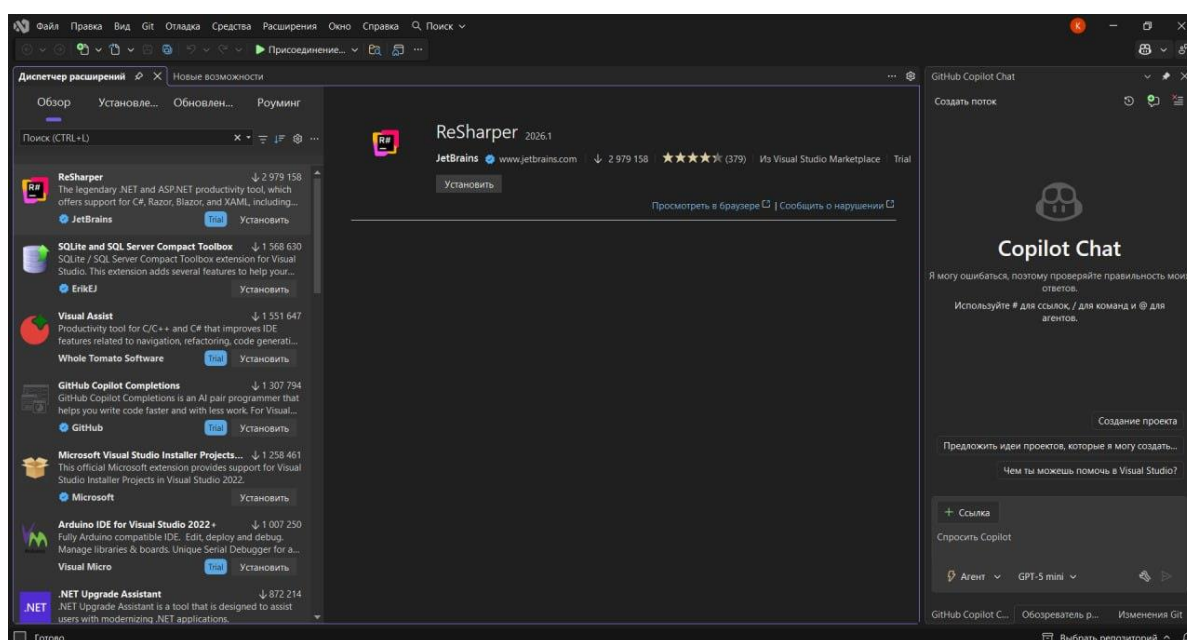


Рисунок 11 – Меню расширений

					ПГУ 1.090301.25.001 ПЗ	Лист 22
Изм.	Лист	№ докум.	Подпись	Дата		

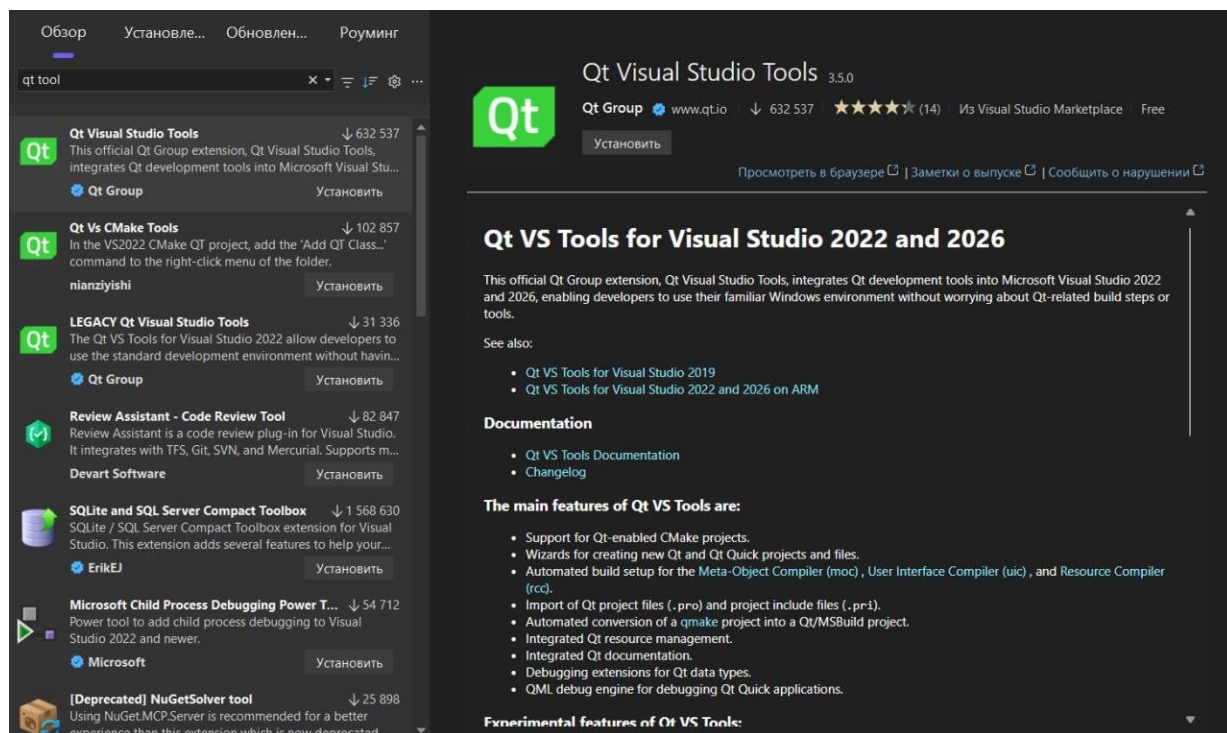


Рисунок 12 – Установка расширения «Qt Visual Studio tools»

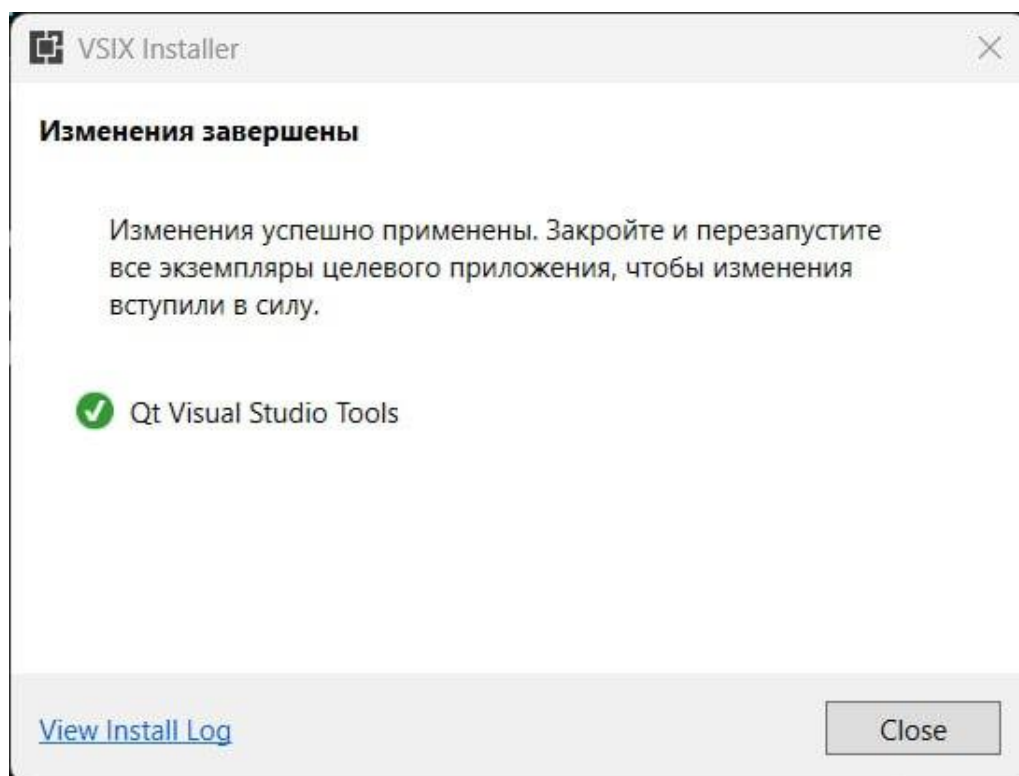


Рисунок 13 – Установка завершена

Помимо настройки IDE требуется установка самого фреймворка Qt с официального сайта (рисунок 14). Для загрузки необходимо выбрать целевую операционную систему (рисунок 15). В окне установщика требуется активировать режим «Выборочная установка» (рисунок 16).

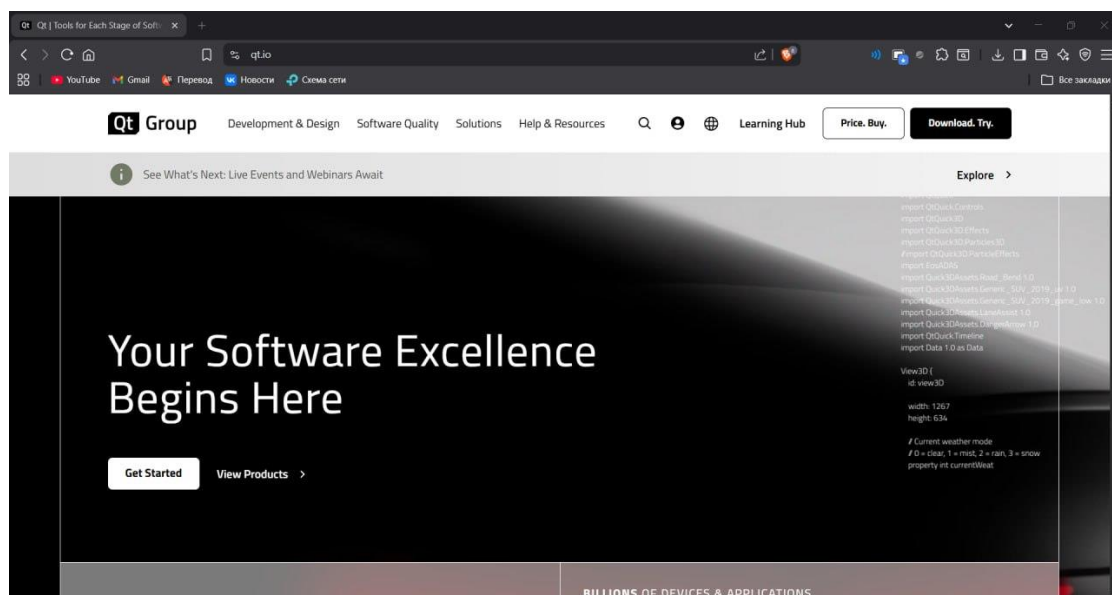


Рисунок 14 – Официальный сайт Qt

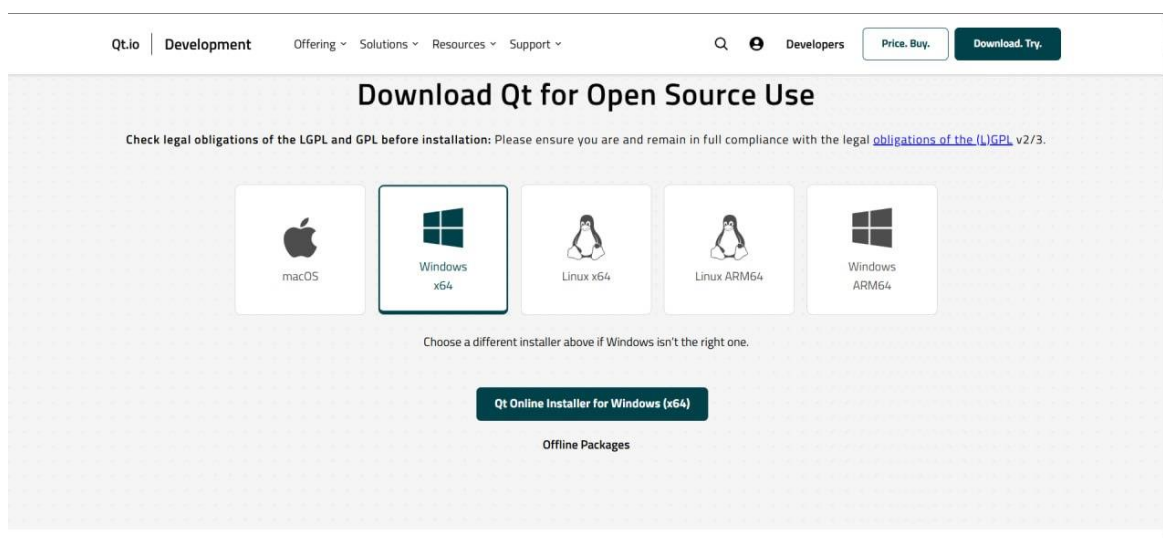


Рисунок 15 – Выбор целевой платформы

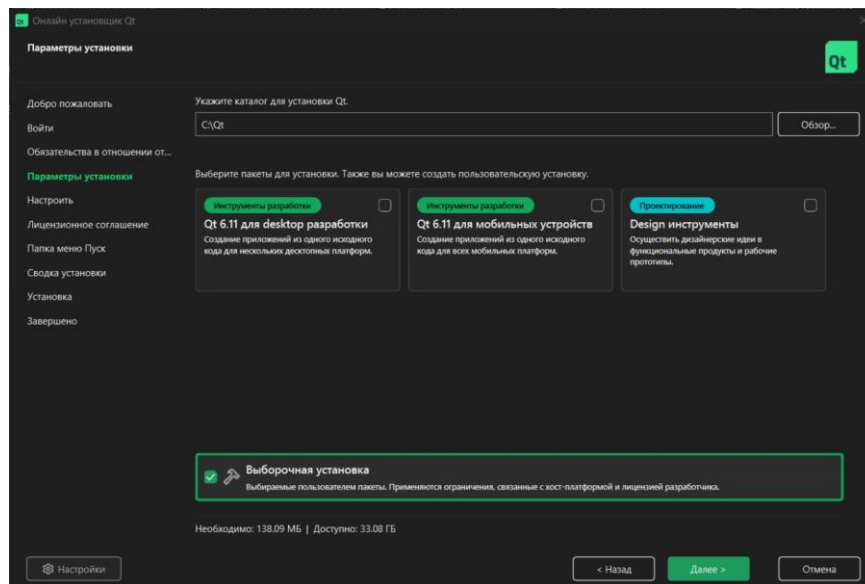


Рисунок 16 – Выбор типа установки

На этапе выбора компонентов необходимо отметить версию библиотеки Qt, соответствующую компилятору MSVC 2022 64-bit (рисунок 17).

По завершении установки в Visual Studio создается тестовый проект типа «Qt Quick Application» (рисунок 18). Если среда разработки не обнаружила версию Qt автоматически (рисунок 19), следует перейти по ссылке «Click here to browse for a Qt version» и указать путь к файлу qmake.exe в директории установленного фреймворка (рисунок 20).

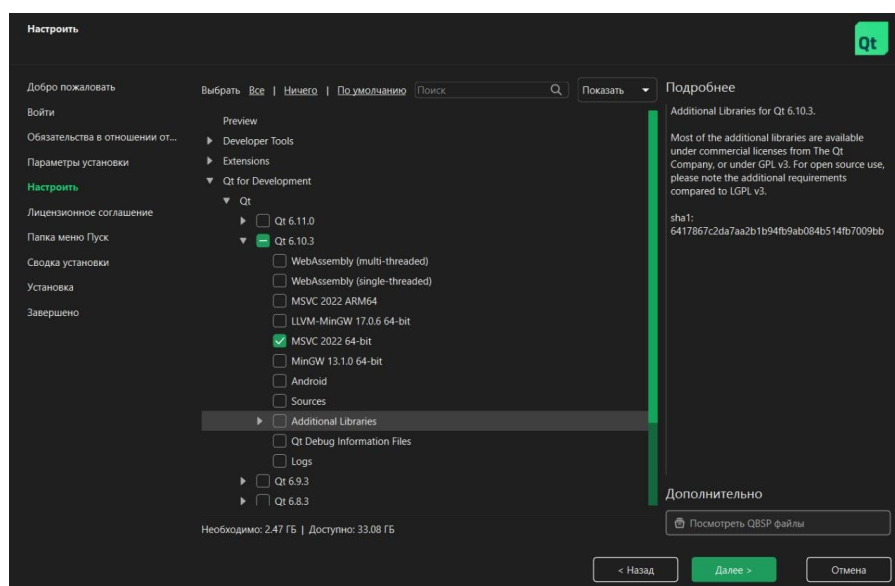


Рисунок 17 – Выбор версии библиотек Qt

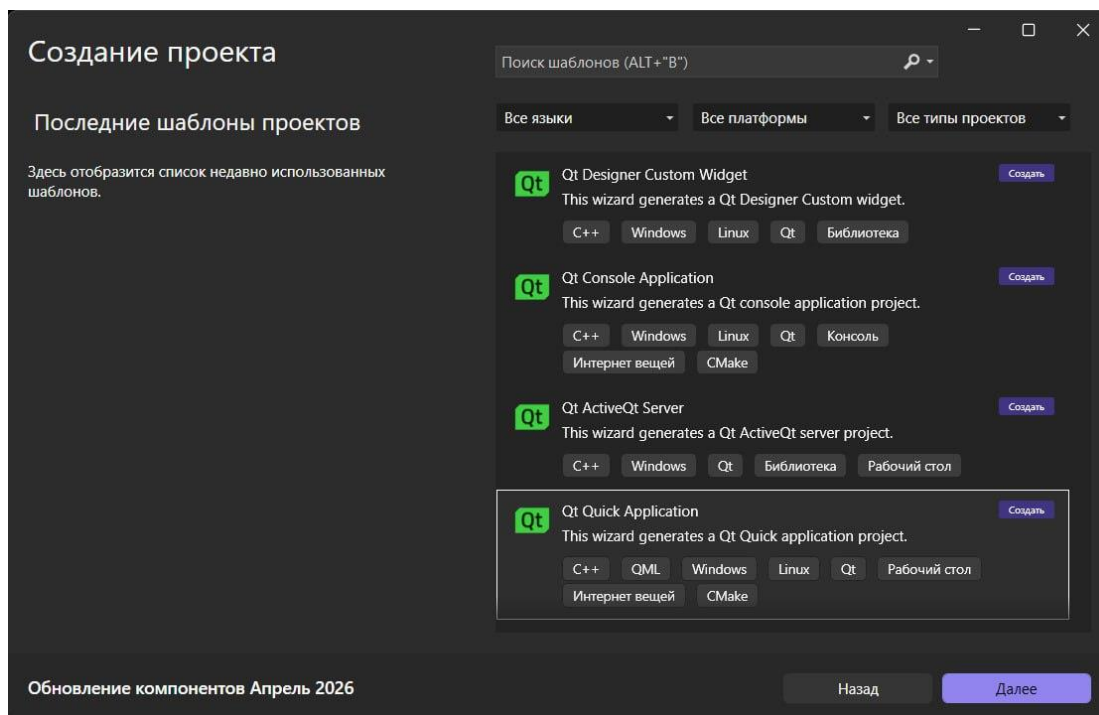


Рисунок 18 – Создание проекта «Qt Quick Application»

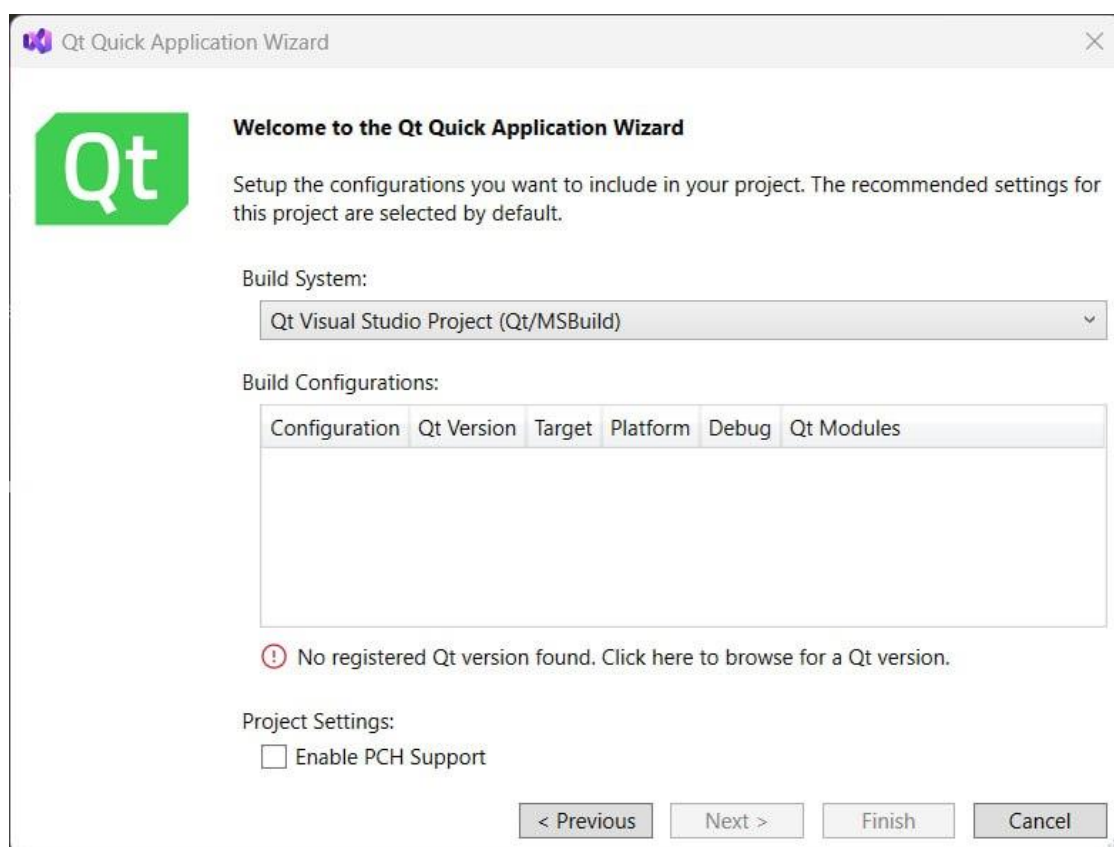


Рисунок 19 – Уведомление об отсутствии зарегистрированной версии Qt

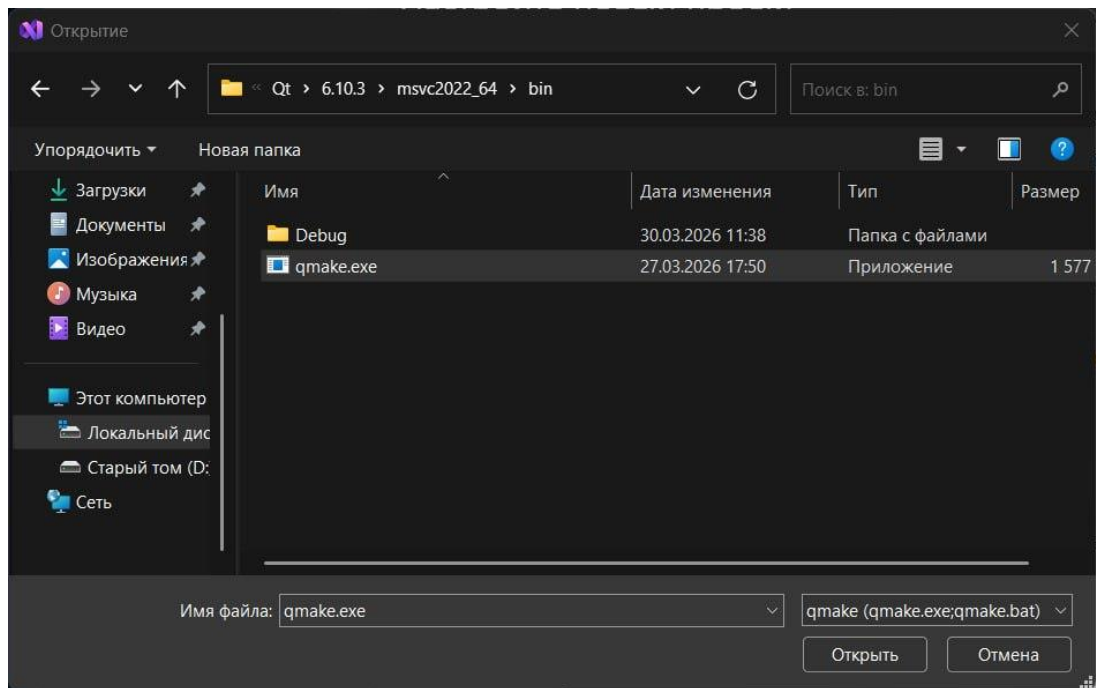


Рисунок 20 – Настройка пути к исполняемому файлу qmake.exe

После успешной конфигурации открывается рабочая область проекта (рисунок 21). Для проверки корректности настройки среды выполним сборку и запуск приложения. Для этого в панели инструментов выбираем пункты «Сборка», «Собрать решение», после чего запускаем «Локальный отладчик Windows». Появление окна готового приложения на экране (рисунок 22) подтверждает корректную настройку среды разработки.

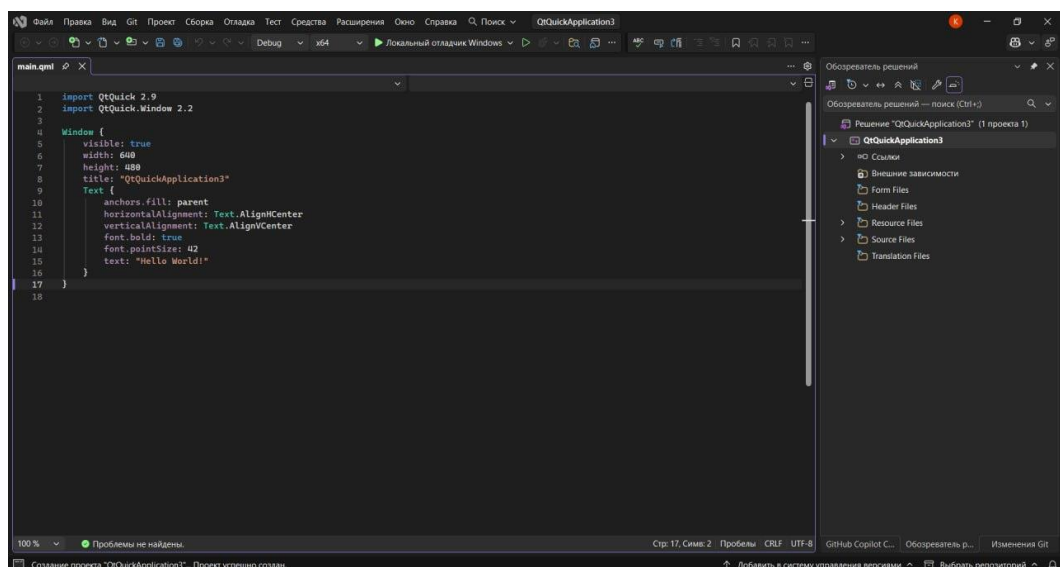


Рисунок 21 – Главное окно среды разработки

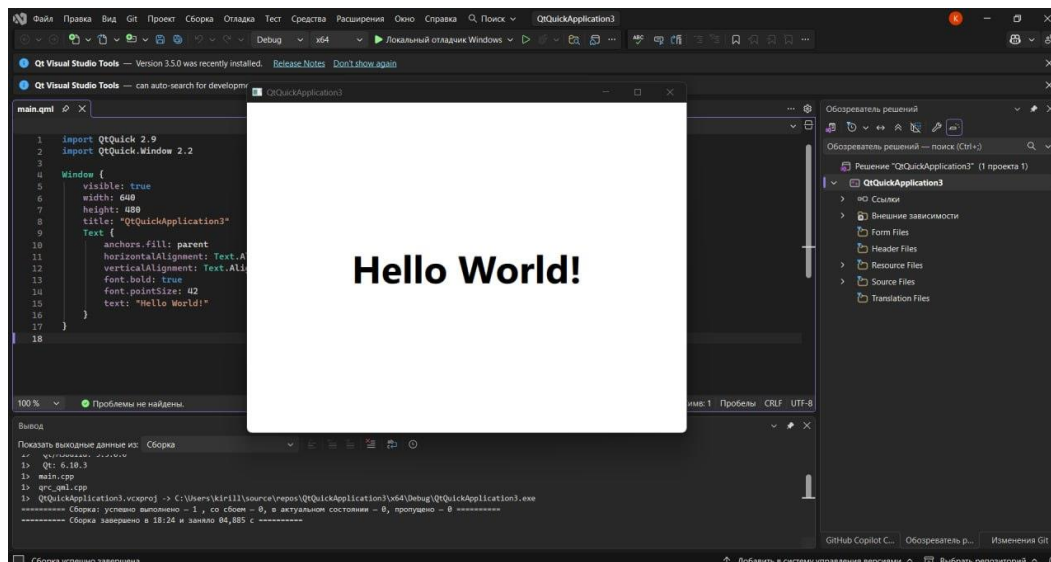


Рисунок 22 – Результат запуска скомпилированного приложения

1.3 Проектирование приложения

1.3.1 Паттерн MVP

В ходе проектирования клиент-серверной архитектуры разрабатываемой системы был произведен анализ распространенных паттернов представления (MVC, MVP, MVVM). Ключевыми критериями выбора являлись: чёткое разделение кода графического интерфейса и кода, отвечающего за взаимодействие с сервером, удобство тестирования сетевых сценариев, а также лёгкость обработки асинхронных ответов.

В качестве основного архитектурного шаблона для клиентской части был выбран Model-View-Presenter (MVP).

MVP (Model, View, Presenter) – это шаблон проектирования, делящий архитектуру приложения на три основных компонента: модель (Model), представление (View) и представитель (Presenter). С помощью данного шаблона представляется возможным отделяющий представление и логику приложения от графического интерфейса [4].

Модель реализует хранение данных и программную логику, не зависящую от пользовательского интерфейса. Она управляет состоянием приложения, предоставляет методы для изменения данных и уведомляет об изменениях. Модель

ничего не знает о существовании представления – с ней взаимодействует исключительно представитель [4].

Представление – это графический интерфейс, реализующий абстрактный. Оно определяет внешний вид приложения, отображает переданные ему данные и передает действия пользователя представителю, посредством вызовов его методов. Представление не содержит бизнес-логики и не обращается напрямую к модели, все обновления интерфейса выполняются представителем [4].

Представитель – выступает связующим звеном между моделью и представлением. Он получает данные модели и преобразует их для отображения в представлении. При этом представитель работает с представлением в абстрактном виде, через его интерфейс [4].

Логическая схема MVP-приложения представлена на рисунке 23.

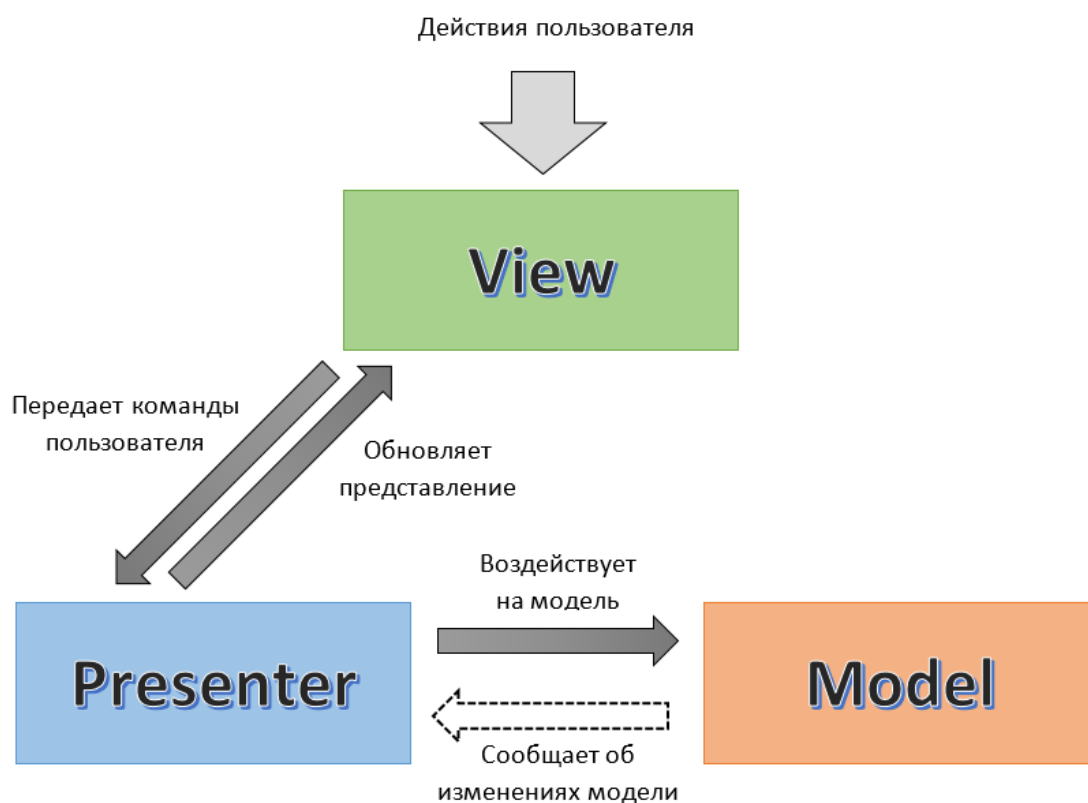


Рисунок 23 – Логическая схема MVP-приложения

1.3.2 Архитектура приложения

Во время работы с программой пользователь взаимодействует с элементами графического интерфейса (View). Любое действие – регистрация, авторизация, отправка текстового или файлового сообщения, управление групповыми чатами – инициируется через кнопки, поля ввода и другие элементы управления.

View не содержит собственной логики обработки, поэтому когда пользователь нажимает кнопку или подтверждает ввод, представление немедленно передаёт событие соответствующему компоненту Presenter.

Presenter выполняет всю необходимую обработку: проверяет корректность и полноту введённых данных, инициирует сетевые запросы и, при необходимости, обращается к слою Model.

Модели служат для хранения состояния приложения и не зависят от интерфейса. Presenter помещает в них новые данные (список чатов, сообщения, информацию о пользователях), а модели уведомляют об изменениях с помощью сигналов.

Получив сигнал от модели, Presenter подготавливает данные в формате, удобном для отображения, и передаёт их обратно в слой представления. View только отрисовывает полученные элементы, не занимаясь их самостоятельным извлечением или обработкой.

На рисунке 24 представлена диаграмма компонентов приложения.

1.3.3 Проектирование базы данных

Для работы мессенджера спроектированы две независимые реляционные базы данных под управлением SQLite: серверная обеспечивает централизованное хранение и целостность информации, а клиентская – локальный кэш для быстрого отображения данных и поддержки офлайн режима. Такое разделение позволяет разгрузить сервер, снизить сетевую задержку при открытии чатов и гарантировать сохранность данных даже при отсутствии соединения.

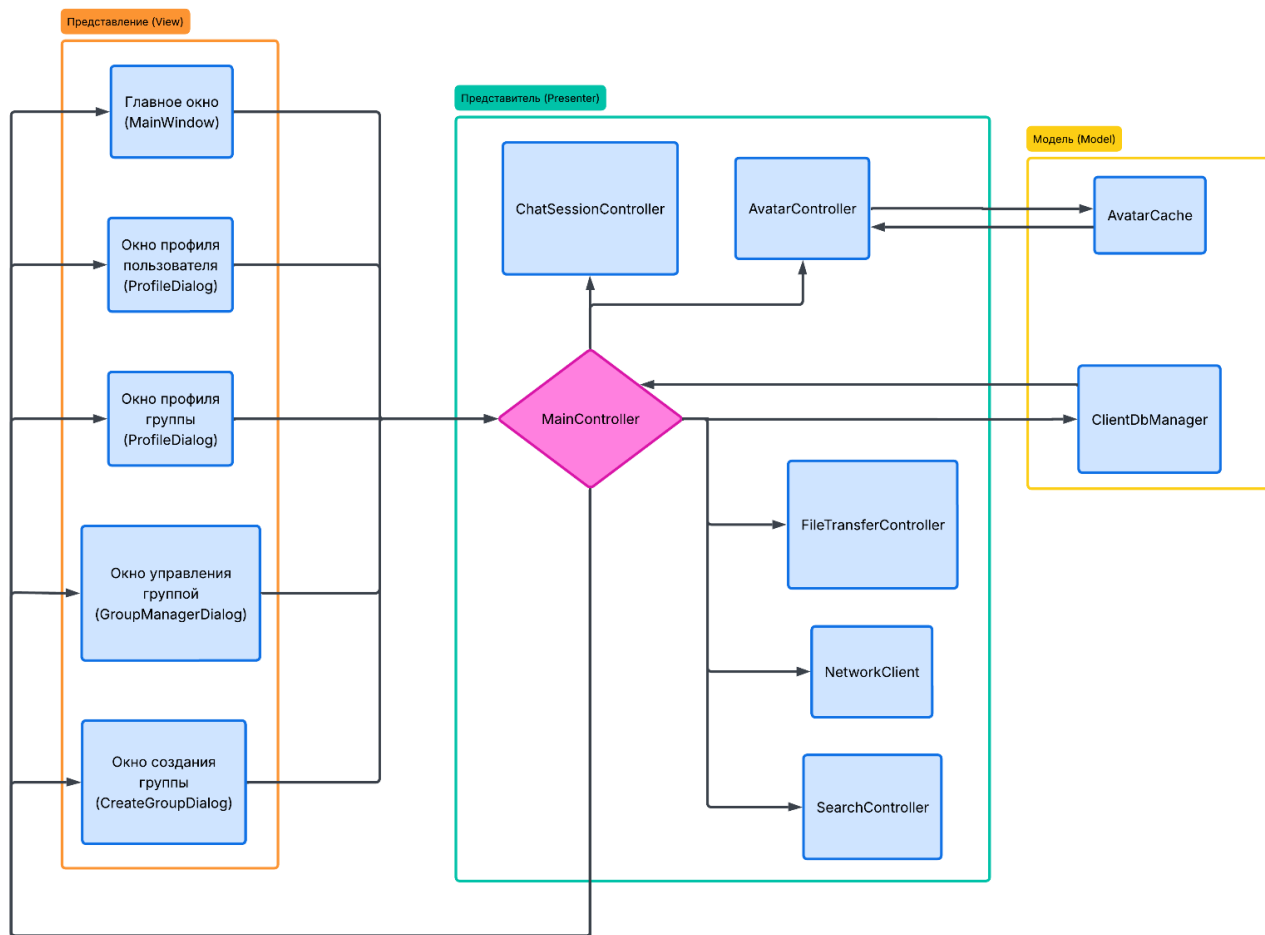


Рисунок 24 – Диаграмма компонентов

Серверная БД является основным хранилищем всей информации мессенджера. Она должна содержать следующие таблицы:

- users. Содержит учётные записи зарегистрированных пользователей;
- chats. Описывает все чаты;
- chat_participants. Связывает пользователей с чатами и хранит права администраторов чата;
- messages. Хранит текстовые сообщения;
- file_messages. Хранит информацию о файловых сообщениях;
- message_deletions. Хранит отметки об удалении сообщений «только для себя»;
- read_receipts. Хранит информацию о прочтении сообщений участниками чата;
- blocked_users. Хранит информацию о блокировках пользователей.

Таблица 1 – Атрибуты таблицы users

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Уникальный идентификатор пользователя	PK	1
username	TEXT	Логин для входа		petrov
password	TEXT	Хеш пароля		ab12:3f4d5e...
email	TEXT	Электронная почта		petrov@example.com
last_seen	INTEGER	Время последней активности		1715200000
avatar_id	TEXT	Идентификатор файла аватара		a1b2c3d4

Таблица 2 – Атрибуты таблицы chats

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Уникальный идентификатор чата	PK	101
name	TEXT	Название (для групп)		Проектная команда
is_group	INTEGER	Признак группового чата (0/1)		1
description	TEXT	Описание группы		Обсуждение задач
avatar_id	TEXT	Идентификатор аватара группы		g1a2b3c4

Таблица 3 – Атрибуты таблицы chat_participants

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
chat_id	INTEGER	Идентификатор чата	FK → chats.id	101
user_id	INTEGER	Идентификатор участника	FK → users.id	1

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
is_admin	INTEGER	Является ли администратором (0/1)		1

Таблица 4 – Атрибуты таблицы messages

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Уникальный идентификатор сообщения	PK	1001
chat_id	INTEGER	Чат, в котором отправлено	FK → chats.id	101
sender_id	INTEGER	Отправитель	FK → users.id	2
text	TEXT	Текст сообщения		Привет!
timestamp	INTEGER	Время отправки		1715200500
is_edited	INTEGER	Признак редактирования (0/1)		0
edited_at	INTEGER	Время последней правки		0
reply_to_id	INTEGER	ID цитируемого сообщения		1000
reply_to_sender_name	TEXT	Имя автора цитируемого сообщения		ivanov
reply_to_preview	TEXT	Превью цитируемого текста		Всем привет

Таблица 5 – Атрибуты таблицы file_messages

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Уникальный идентификатор файлового сообщения	PK	2001

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
sender_id	INTEGER	Отправитель	FK → users.id	1
file_id	TEXT	Уникальный идентификатор файла на сервере		f123abc
file_name	TEXT	Оригинальное имя файла		Отчёт.pdf
file_size	INTEGER	Размер в байтах		204800
mime_type	TEXT	Тип файла		application/pdf
is_voice	INTEGER	Признак голосового сообщения (0/1)		0
duration	INTEGER	Длительность голосового сообщения		0
timestamp	INTEGER	Время отправки		1715200600

Таблица 6 – Атрибуты таблицы message_deletions

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
message_id	INTEGER	Идентификатор сообщения	FK → messages.id	1001
user_id	INTEGER	Пользователь, скрывший сообщение	FK → users.id	1

Таблица 7 – Атрибуты таблицы read_receipts

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
chat_id	INTEGER	Чат	FK → chats.id	101
user_id	INTEGER	Пользователь	FK → users.id	1
last_msg_id	INTEGER	ID последнего прочитанного сообщения		1005

Таблица 8 – Атрибуты таблицы blocked_users

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
blocker_id	INTEGER	Кто заблокировал	FK → users.id	1
blocked_id	INTEGER	Кого заблокировали	FK → users.id	3

На рисунке 25 изображена ER-диаграмма базы данных сервера.



Рисунок 25 – ER-диаграмма БД сервера

Клиентская БД представляет собой облегчённый локальный кэш. Её таблицы хранят только те данные, которые нужны для быстрой загрузки интерфейса без постоянных обращений к серверу. Она должна содержать следующие таблицы:

- users. Сокращённая версия серверной таблицы: отсутствуют поля пароля и email, так как они не нужны на клиенте;
- chats. Клиентская копия списка чатов, дополненная полем владельца для локальной изоляции данных;
- chat_admins. Локальный кэш администраторов чатов;
- messages. Совмещает в себе серверные таблицы messages и file_messages. В отличие от серверной версии использует временный идентификатор temp_id для отображения сообщений в процессе отправки, пока сервер не подтвердит;
- deleted_for_me. Локальный аналог серверной message_deletions, хранит ID сообщений, удалённых «только у себя».

Таблица 9 – Атрибуты таблицы users

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор пользователя	PK	1
username	TEXT	Логин		petrov
avatar_id	TEXT	Идентификатор аватара		a1b2c3d4
is_online	INTEGER	Онлайн статус (0/1)		1
last_seen	INTEGER	Время последней активности		1715200000

Таблица 10 – Атрибуты таблицы chats

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
id	INTEGER	Идентификатор чата	PK	101

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
owner_id	INTEGER	Локальный владелец записи	FK → users.id	1
type	TEXT	Тип: 'private' или 'group'		group
name	TEXT	Название чата		Проектная команда
avatar_id	TEXT	ID аватара		g1a2b3c4
is_pinned	INTEGER	Закреплён ли чат (0/1)		0

Таблица 11 – Атрибуты таблицы chat_admins

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
chat_id	INTEGER	Чат	FK → chats.id	101
user_id	INTEGER	Администратор	FK → users.id	1

Таблица 12 – Атрибуты таблицы messages

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
temp_id	TEXT	Временный ID	PK	{uuid}
real_id	INTEGER	Подтверждённый сервером ID		1001
chat_id	INTEGER	Чат	FK → chats.id	101
sender_id	INTEGER	Отправитель	FK → users.id	2
sender_name	TEXT	Имя отправителя		ivanov
text	TEXT	Текст сообщения		Привет
timestamp	INTEGER	Время отправки		171520
is_edited	INTEGER	Признак редактирования		0
edited_at	INTEGER	Время правки		0
msg_type	TEXT	Тип сообщения		text
reply_to_id	INTEGER	ID цитируемого сообщения		1000
reply_to_sender_name	TEXT	Имя автора цитаты		ivanov

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
reply_to_preview	TEXT	Превью цитаты		Привет! Как дела?
file_id	TEXT	ID файла (если есть)		
file_name	TEXT	Имя файла		
file_size	INTEGER	Размер файла		
local_path	TEXT	Путь к локальному файлу		
is_voice	INTEGER	Голосовое сообщение		0
duration	INTEGER	Длительность голосового		0
status	TEXT	Статус отправки: 'pending', 'sent'		sent

Таблица 13 – Атрибуты таблицы deleted_for_me

Имя столбца	Тип данных	Описание	Роль атрибута	Пример
real_id	INTEGER	ID сообщения	FK → messages.real_id	1001
user_id	INTEGER	Локальный пользователь	FK → users.id	1

На рисунке 26 представлена ER-диаграмма базы данных клиента.

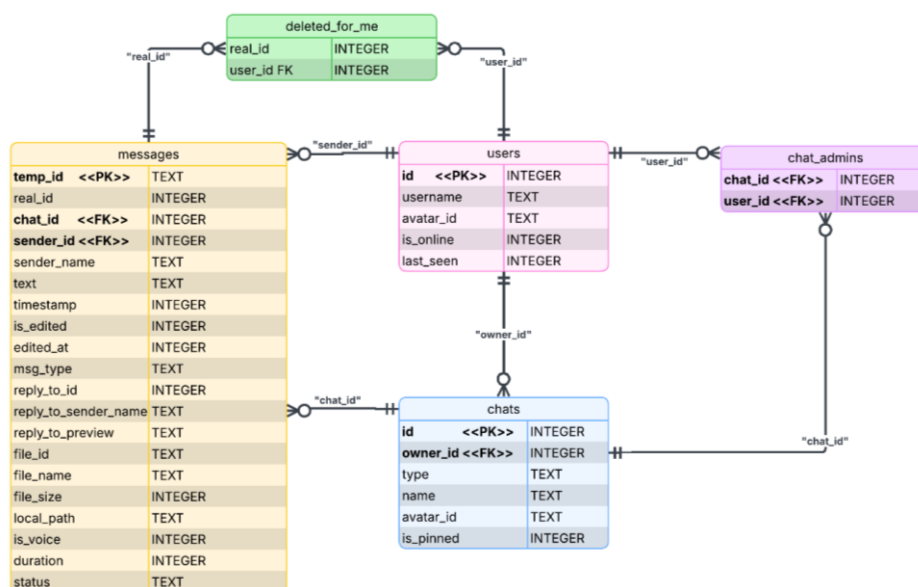


Рисунок 26 – ER-диаграмма БД клиента

1.4 Разработка клиентского приложения

Клиентская часть приложения разработана на языке C++ с использованием фреймворка Qt и реализует полнофункциональный графический интерфейс для обмена сообщениями, передачи файлов, создания и управления групповыми чатами.

1.4.1 Инициализация пользовательского интерфейса

Центральным элементом графического интерфейса является класс MainWindow, реализующий интерфейс IMainView. При запуске приложения после успешной авторизации вызывается метод initialize(MainController* controller), который выполняет первичную настройку всех элементов окна.

В ходе инициализации создаются пользовательские делегаты для отрисовки списков: ChatListDelegate для элементов списка чатов и MessageBubbleDelegate для сообщений. Делегаты назначаются соответствующим виджетам через вызов setItemDelegate(). Фрагмент настройки делегата сообщений MessageBubbleDelegate, отвечающий за формирование блоков сообщений с различным оформлением:

```
m_msgDelegate = new MessageBubbleDelegate(this);  
ui->listMessages->setItemDelegate(m_msgDelegate);
```

На рисунке 27 представлены диаграммы классов ChatListDelegate и MessageBubbleDelegate.

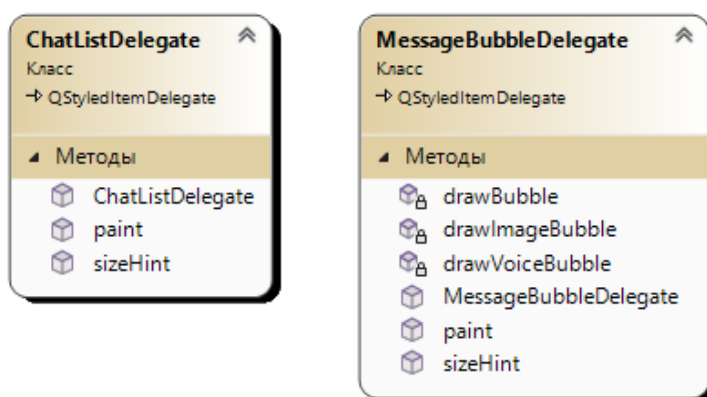


Рисунок 27 – Диаграммы классов ChatListDelegate и MessageBubbleDelegate

Главное окно содержит три основные панели: левую панель со списком чатов и строкой поиска, центральную панель с областью сообщений и заголовком активного чата, а также нижнюю панель ввода, включающую текстовое поле, кнопки прикрепления файлов, выбора эмодзи и голосового ввода.

В методе `setupInitialState()` выполняется начальная настройка: скрываются вспомогательные виджеты (поисковая выдача, панель редактирования), устанавливается контекстное меню для списка сообщений.

1.4.2 Сетевое взаимодействие и авторизация

Для обмена данными с сервером используется класс `NetworkClient`, который инкапсулирует защищённое TLS-соединение на базе `QSslSocket`, механизмы повторного подключения и аутентификации.

При установлении соединения клиент и сервер выполняют TLS-рукопожатие, после чего весь сетевой обмен осуществляется в зашифрованном виде. Для тестирования работы защищённого соединения потребовалось создать самоподписанный SSL-сертификат с использованием инструмента `OpenSSL` [5].

Диаграмма последовательностей, показывающая процесс регистрации и входа, изображена на рисунке 28.

Успешная авторизация приводит к созданию экземпляра `MainController`, которому передаются объекты `NetworkClient`, идентификатор и имя пользователя, а также указатель на `MainWindow`. `MainController` становится центральным узлом, владеющим всеми подконтроллерами, моделями данных и сервисами, и соединяет их с сетевым клиентом и представлением, используя сигналы и слоты.

Фрагмент инициализации контроллера:

```
MainController* controller = new MainController(client, userId, username, w);  
w->initialize(controller);
```

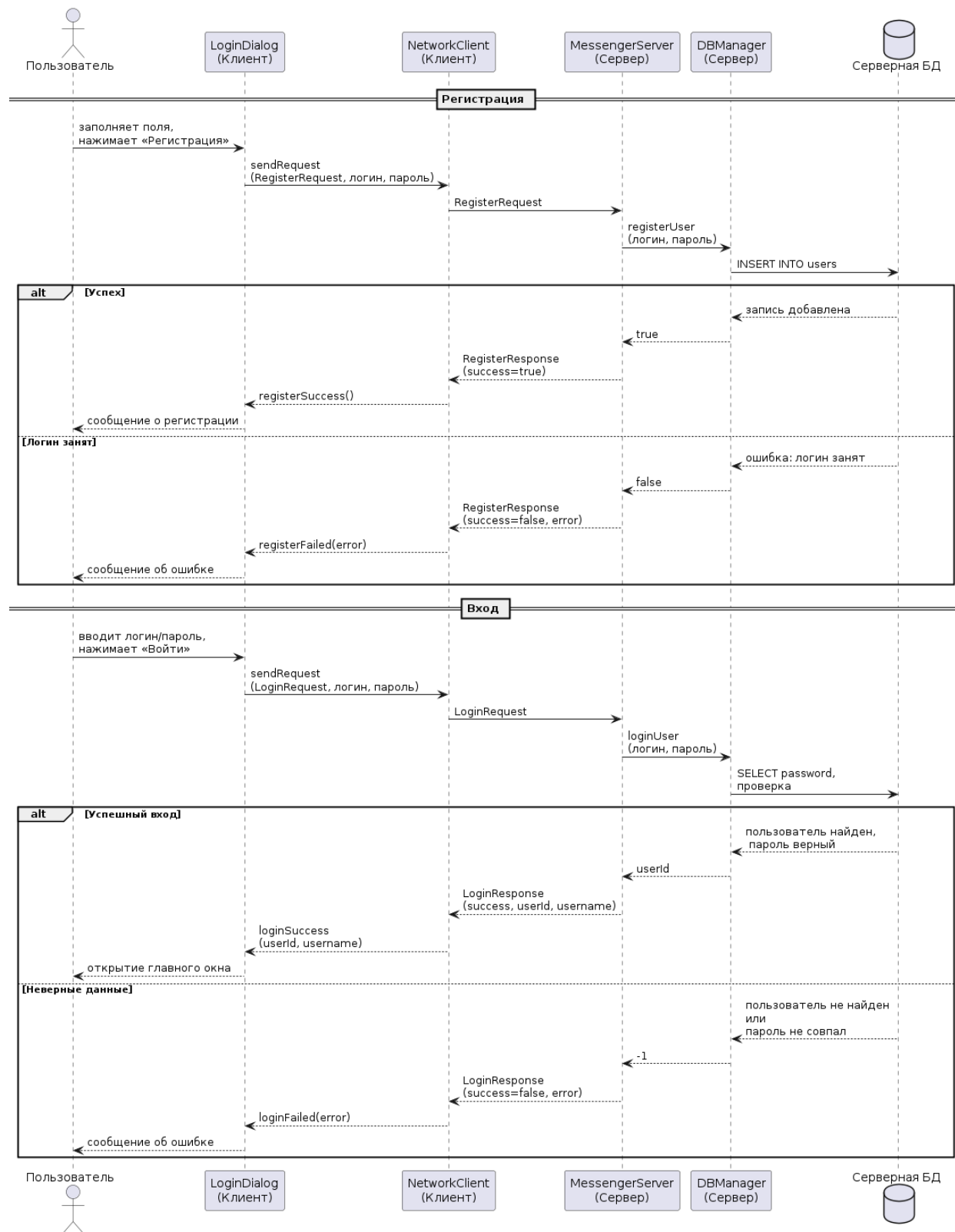


Рисунок 28 – Диаграмма последовательностей. Регистрация и вход

1.4.3 Отображение и управление чатами

Список чатов загружается с сервера с помощью запроса ChatListRequest и сохраняется в модель ChatModel, которая уведомляет представление о добавлении или изменении элементов. Каждый чат описывается структурой ChatDisplayData, содержащей имя, подзаголовок (последнее сообщение или статус), идентификатор аватара и количество непрочитанных сообщений. При получении данных MainController вызывает метод MainWindow::addChatItem(), который создаёт QListWidgetItem, после чего делегат ChatListDelegate отрисовывает элемент в виде строки с аватаром, именем и счётчиком непрочитанных сообщений.

Диаграмма класса ChatModel изображена на рисунке 29.

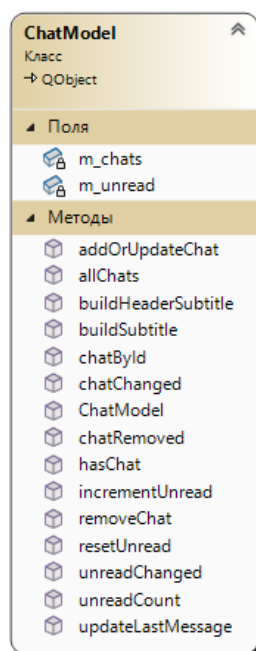


Рисунок 29 – Диаграмма класса ChatModel

При выборе чата пользователем вызывается слот onChatSelected(), который передаёт управление в ChatSessionController::openChat(). Последний выполняет очистку списка сообщений, сбрасывает счётчик непрочитанных сообщений и, при необходимости, запрашивает историю сообщений через NetworkClient::requestMessageHistory().

1.4.4 Отправка и прием текстовых сообщений

Отправка текстового сообщения инициируется нажатием кнопки «Отправить» или клавиши Enter в поле ввода.

Обработчик вызывает метод `MainController::sendTextMessage()`, который делегирует вызов `ChatSessionController`. В методе `sendTextMessage()` создается объект `Message` с временным идентификатором `tempId (UUID)` и статусом `"pending"`. Сообщение сохраняется в локальную БД через `ClientDbManager::saveMessage()`, добавляется в `MessageModel` и отображается в списке. Затем формируется JSON-запрос `TextMessage` и отправляется через `NetworkClient`.

При получении ответа от сервера с реальным идентификатором сообщения `ChatSessionController::onMessageReceived()` подтверждает отправку, обновляя идентификатор и статус в `MessageModel`. Для входящих сообщений, не являющихся собственными, модель вставляет новое сообщение и, если чат в данный момент открыт, автоматически помечает его как прочитанное, отправляя серверу `ReadReceipt` (подтверждение о прочтении).

Диаграмма класса `MessageModel` представлена на рисунке 30.

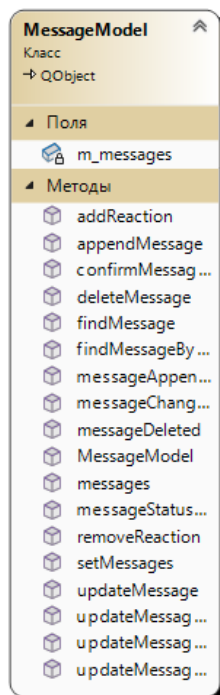


Рисунок 30 – Диаграмма класса `MessageModel`

1.4.5 Передача файлов и голосовых сообщений

Отправка файлов реализуется через FileTransferManager, который разбивает файл на фрагменты размером 64 КБ и передаёт их. При инициировании отправки вызывается FileTransferController::sendFile(), который запускает FileTransferManager::startUpload(). Последний создаёт временное сообщение с информацией о файле и сохраняет его в локальной БД.

Процесс отправки начинается с передачи метаданных файла серверу, после чего клиент получает подтверждение и начинает последовательную отправку фрагментов. Каждый фрагмент сопровождается подтверждением от сервера. По завершении передачи сервер сохраняет файловое сообщение в БД и возвращает итоговый объект сообщения, который заменяет временное сообщение.

Скачивание файла активируется двойным щелчком по файловому сообщению или выбором пункта «Сохранить» в контекстном меню. FileTransferManager::startDownload() создаёт контекст загрузки FileContext, запрашивает файл у сервера и принимает фрагменты, записывая их на диск. После завершения загрузки вызывается слот onFileDownloaded(), который обновляет локальный путь к файлу в модели и предоставляет пользователю доступ к нему.

Голосовые сообщения записываются через класс VoiceRecordingService, использующий AudioCapturer для захвата звука с микрофона. При отпускании кнопки записи аудиоданные сохраняются в формате WAV, и файл отправляется как обычное файловое сообщение с флагом isVoice. Воспроизведение осуществляется через класс VoicePlayerManager, который управляет сигналами для обновления интерфейса (полоса прогресса, кнопка воспроизведения).

Диаграммы классов FileTransferManager, VoiceRecordingService, AudioCapturer и VoicePlayerManager изображены на рисунке 31.

Процесс отправки и загрузки файла показан на диаграмме последовательностей (рисунки 32, 33).

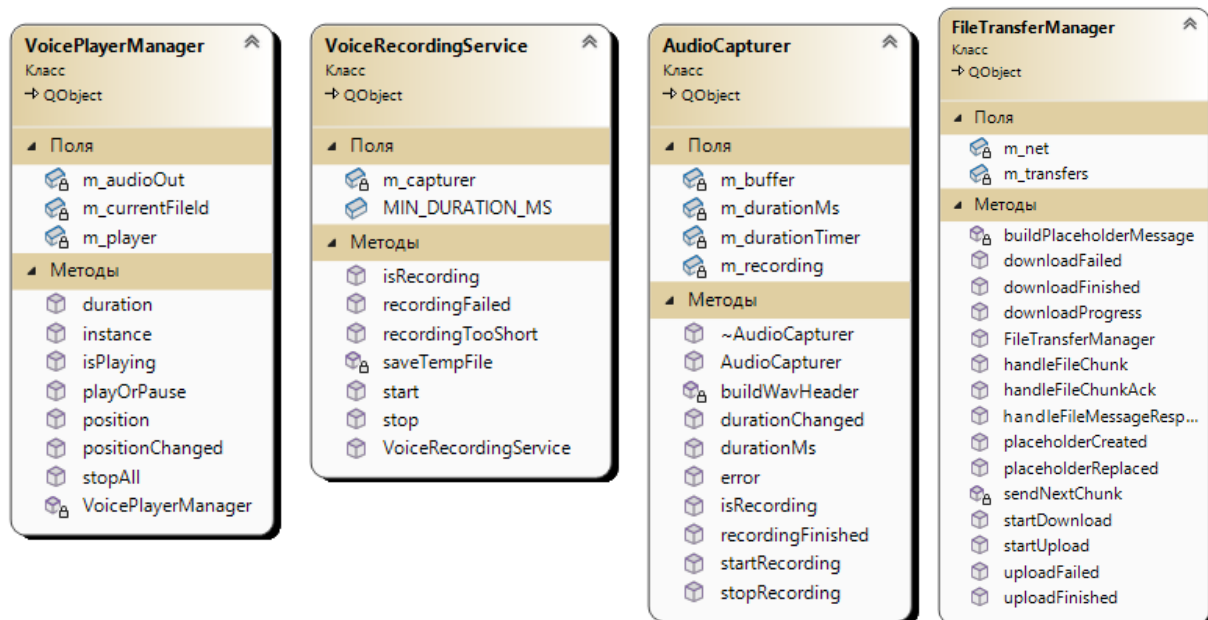


Рисунок 31 – Диаграммы классов FileTransferManager, VoiceRecordingService, AudioCapturer и VoicePlayerManager

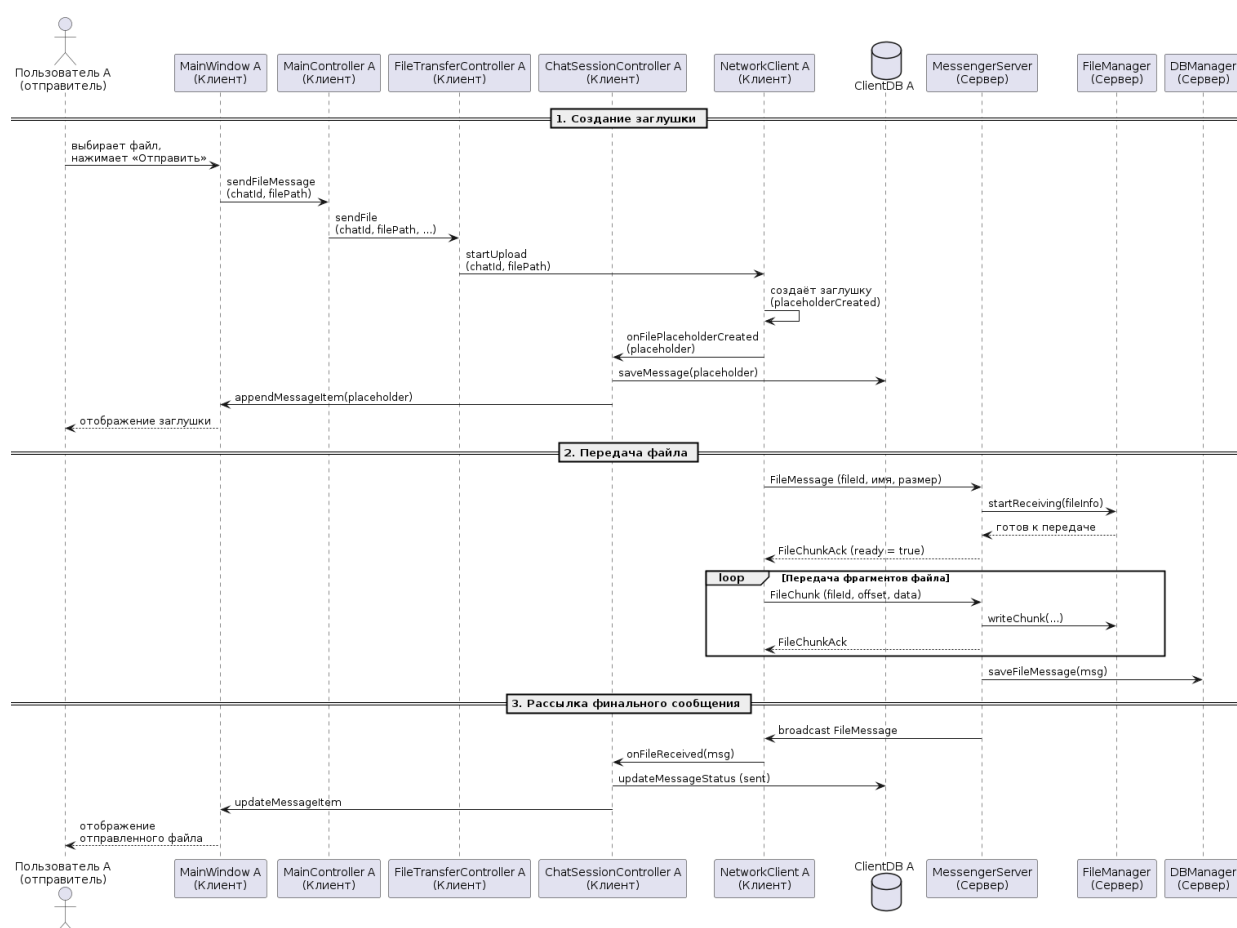


Рисунок 32 – Диаграмма последовательностей. Отправка файла

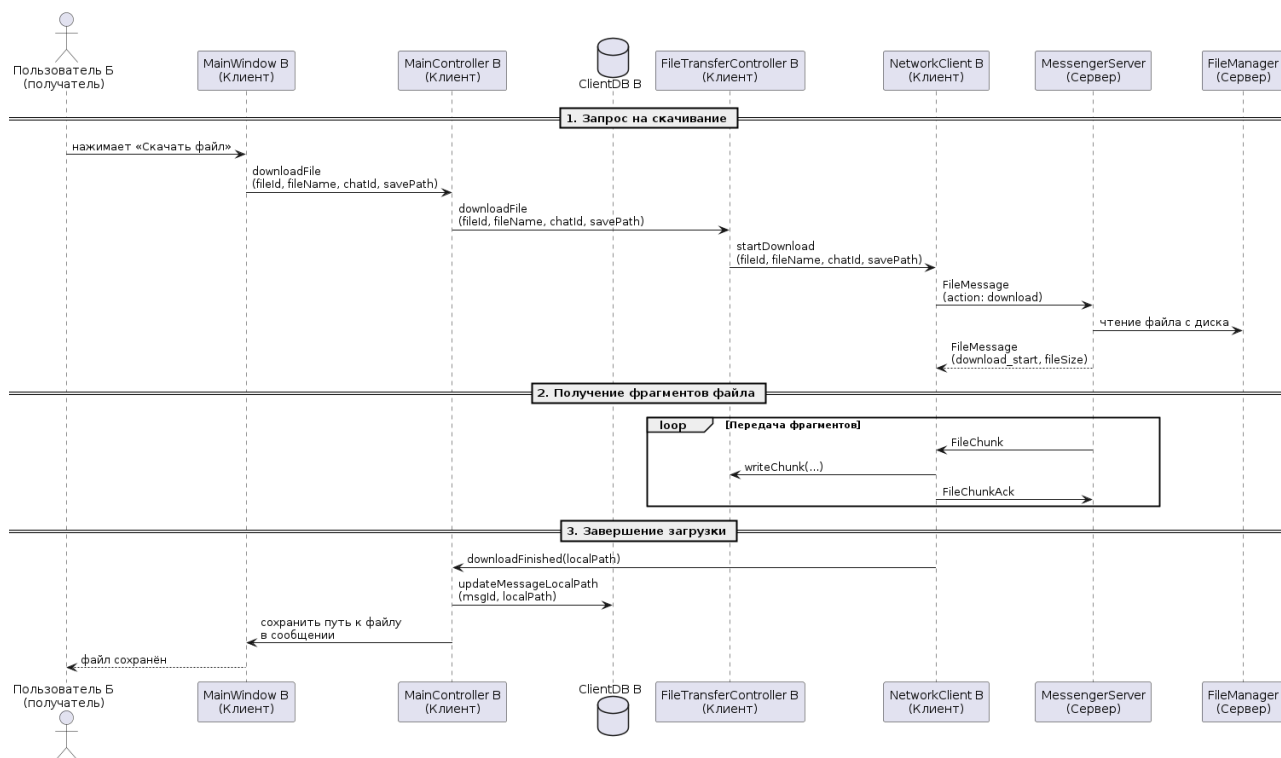


Рисунок 33 – Диаграмма последовательностей. Загрузка файла

1.4.6 Управление группами и профилями

Создание группового чата инициируется через диалоговое окно CreateGroupDialog, которое отображает список доступных пользователей с возможностью множественного выбора. После подтверждения серверу отправляется запрос CreateChatRequest с типом "group" и списком участников. Серверный ответ содержит идентификатор созданного чата, название, список участников и администраторов, после чего новый чат добавляется в модель ChatModel и открывается.

Управление существующей группой (добавление/удаление участников, смена названия, загрузка аватара) выполняется через диалоговое окно GroupManagerDialog, доступное только администраторам.

Для просмотра профиля пользователя или информации о группе используется ProfileDialog, отображающее детальные сведения: аватар, имя, статус, email, описание группы и список участников. Из диалогового окна профиля можно начать приватный чат или заблокировать пользователя.

На рисунке 34 представлены диаграммы классов CreateGroupDialog, GroupManagerDialog и ProfileDialog.

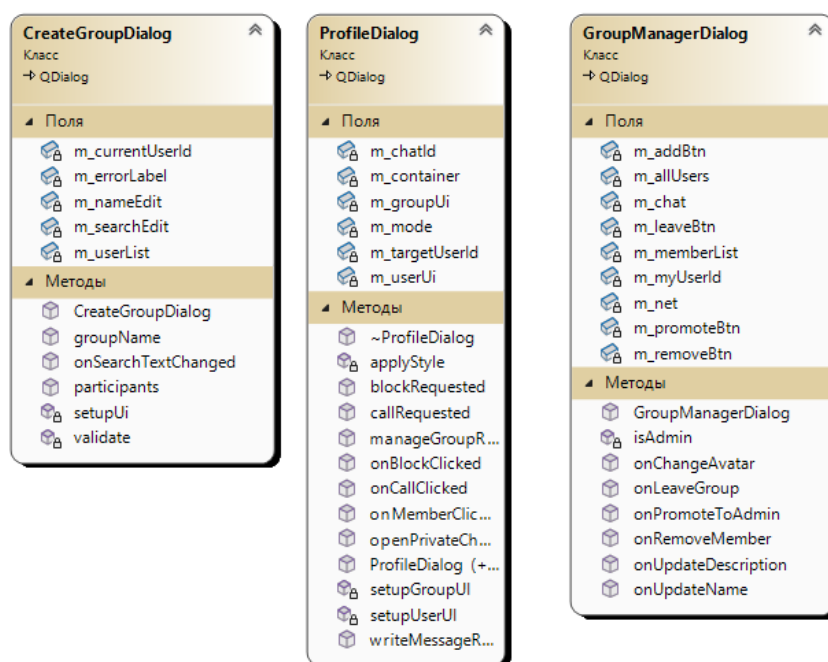


Рисунок 34 – Диаграммы классов CreateGroupDialog, GroupManagerDialog и ProfileDialog.

1.4.7 Вспомогательные классы

Индикация набора текста реализована классом TypingIndicator, который отслеживает локальный ввод текста и отправляет серверу уведомления TypingStart/TypingStop. При получении аналогичных уведомлений от других пользователей формируется текстовая строка (например, «Пользователь печатает...»), отображаемая в подзаголовке чата.

Подтверждения прочтения управляются ReadReceiptManager, который при открытии чата или прокрутке списка до последнего сообщения отправляет серверу ReadReceiptRequest с идентификатором последнего прочитанного сообщения. Сервер сохраняет эту информацию и рассылает её другим участникам, что позволяет отображать статус сообщений (отправлено / прочитано).

Поиск по чатам и контактам осуществляется через SearchController. При вводе текста в строку поиска локально фильтруются имеющиеся чаты и пользователи, а также отправляется запрос на сервер для поиска по всем

пользователям системы. Результаты отображаются в виде списка с подзаголовками «Чаты» и «Люди».

Диаграммы классов TypingIndicator, ReadReceiptManager и SerachController изображена на рисунке 35.

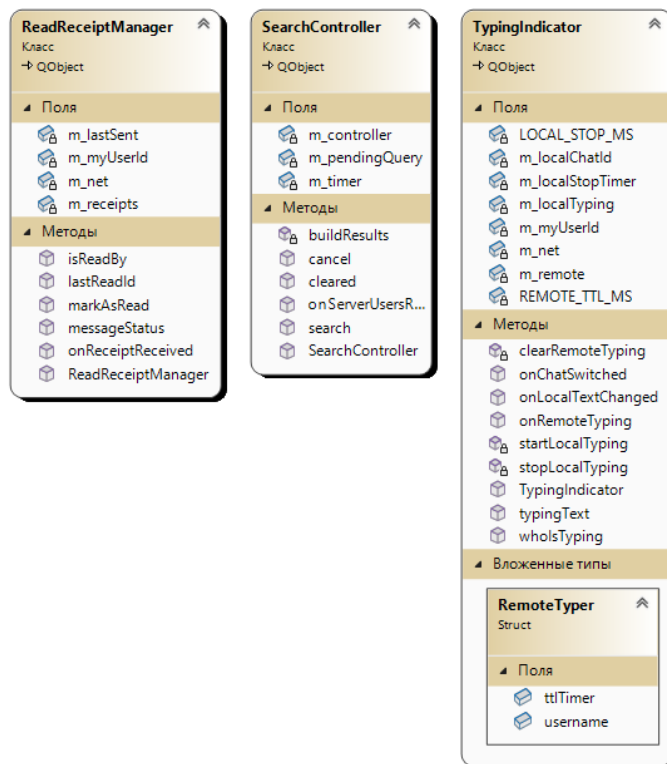


Рисунок 35 – Диаграммы классов TypingIndicator, ReadReceiptManager и SerachController.

1.5 Разработка серверного приложения

Серверная часть мессенджера представляет собой консольное приложение на языке C++ с использованием фреймворка Qt, реализующее централизованное хранение данных, маршрутизацию сообщений, аутентификацию пользователей и управление групповыми чатами.

1.5.1 Инициализация сервера

Сервер запускается из функции main(), где последовательно инициализируются классы для хранения данных и сетевого взаимодействия. Фрагмент кода запуска сервера:

```

if (!DBManager::instance().init("messenger.db")) {
    qCritical() << "Cannot initialize database";
    return -1;
}
if (!FileManager::instance().init("files")) {
    qCritical() << "Cannot init file storage";
    return -1;
}
MessengerServer server;
if (!server.start(12345)) {
    return -1;
}

```

После успешного запуска сервер переходит в цикл обработки событий Qt и ожидает входящие подключения. При новом подключении создаётся состояние клиента ClientState, включающее буфер приёма данных, идентификатор пользователя и таймер проверки активности.

1.5.2 Обработка сетевых соединений и протокол обмена

Все данные передаются в виде кадров: первые 4 байта содержат длину последующего JSON-сообщения. Класс Protocol предоставляет статические методы frame() и extractPackets() для формирования и извлечения полных пакетов из потока. При получении новых данных от клиента срабатывает слот onReadyRead(), который накапливает байты в буфере и по мере накопления полных кадров вызывает processPacket().

Для защиты сетевого трафика сервер использует TLS-соединения, реализованные с помощью QSslSocket и QSslConfiguration.

Метод processPacket() анализирует тип сообщения, извлечённый из JSON-поля type, и направляет запрос в соответствующий обработчик. Фрагмент маршрутизации запросов:

```

switch (type) {
case MessageType::LoginRequest:
    handleLogin(client, payload, state);    break;
case MessageType::TextMessage:
    handleTextMessage(client, payload, state.userId); break;
// ... остальные типы

```

1.5.3 Обработка сообщений

Обработчик `handleTextMessage()` принимает объект `Message` из JSON-запроса и проверяет состоит ли отправитель в чате. Сообщение сохраняется в базе данных через `DBManager::saveMessageGetId()`, после чего рассылается всем участникам чата методом `broadcastToChat()`.

Для поддержки цитирования сообщений извлекается информация об исходном сообщении (идентификатор, автор, часть текста) и заполняется структура `replyTo`. Это позволяет клиентам отображать ответы вместе со ссылкой на оригинал.

Передача файлов организована в два этапа: сначала клиент отправляет метаданные (имя, размер, MIME-тип, признак голосового сообщения), сервер подтверждает готовность к приёму, после чего идёт поток фрагментов. Класс `FileManager` управляет контекстами передачи: при старте создаётся файл на диске, каждый полученный фрагмент записывается по нужному смещению, а после завершения формируется запись в таблице `file_messages` базы данных.

По окончании приёма сервер формирует итоговое сообщение и рассылает его всем участникам чата. Скачивание файла инициируется клиентом запросом с действием "download". Сервер проверяет существование файла и отправляет его по чанкам, используя ту же инфраструктуру `FileTransferProtocol`.

1.5.4 Управление чатами и группами

Создание чатов обрабатывается методом `handleCreateChat()`. Для личного чата сервер сначала проверяет существование аналогичного чата между двумя участниками; если чат уже существует, повторное создание игнорируется. Для

группового чата создаётся новая запись в таблице chats, а инициатор назначается администратором. Все участники немедленно уведомляются о новом чате.

Управление членством в группах (добавление, удаление, выход) реализовано с проверкой прав администратора: добавлять и исключать участников может только администратор; удаление другого администратора запрещено. При выходе последнего участника чат автоматически удаляется вместе со всеми связанными данными.

1.5.5 Работа с базой данных

Работа с базой данных осуществляется классом DBManager, который реализует паттерн «одиночка» и предоставляет высокоуровневые методы для работы с таблицами.

При инициализации DBManager::init() автоматически создаются все необходимые таблицы, если они отсутствуют, и настраиваются индексы для ускорения запросов (например, idx_messages_chat для выборки сообщений конкретного чата).

1.6 Тестирование

При работе с программой пользователи могут допускать ошибки, приводящие к некорректной работе. Например, пользователь может при регистрации оставить некоторые поля пустыми. Подобные ошибки необходимо обрабатывать и, при необходимости, уведомлять о них пользователя. Для проверки работы обработчиков были составлены тест-кейсы (таблица 14).

Таблица 14 – Тест-кейсы

№	Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
1	Проверка наличия пустых полей при входе	Для входа необходимо заполнить поля «Логин» и «Пароль»	1. Запустить приложение. 2. Не заполняя хотя бы одно из полей («Логин» или «Пароль»), нажать кнопку «Войти».	На экране отобразилось уведомление: «Введите логин и пароль»

№	Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
3	Ввод неправильного пароля при входе в систему	Необходимо ввести верный пароль для успешной авторизации	1. Запустить приложение. 2. В поле «Логин» ввести существующий логин. 3. В поле «Пароль» ввести заведомо неверный пароль. 4. Нажать кнопку «Войти».	На экране появилось уведомление: «Неверный логин или пароль»
4	Попытка регистрации с занятым логином	Логин должен быть уникальным	1. Запустить приложение. 2. Заполнить все поля, указав в поле «Логин» уже существующее имя пользователя. 3. Нажать кнопку «Регистрация».	На экране появилось уведомление: «Имя пользователя уже занято»
5	Попытка регистрации с пустыми полями	Все поля регистрации обязательны для заполнения	1. Запустить приложение. 2. Оставить хотя бы одно поле пустым. 3. В окне входа нажать кнопку «Регистрация».	На экране появилось уведомление: «Заполните все поля»
6	Отправка пустого сообщения	Нельзя отправить сообщение без текста	1. Запустить приложение и выполнить вход. 2. Выбрать любой чат. 3. Оставив поле ввода пустым, нажать кнопку «Отправить».	Сообщение не отправляется, кнопка не активна.
7	Отправка сообщения без выбранного чата	Для отправки сообщения необходимо открыть чат	1. Запустить приложение и выполнить вход. 2. Не выбирая чат в списке, ввести текст в поле ввода и нажать кнопку «Отправить».	На экране появилось уведомление: «Выберите чат для отправки сообщения»
8	Создание группы без названия	Для создания группы необходимо указать название	1. Запустить приложение и выполнить вход. 2. Нажать кнопку создания группы («+») 3. В открывшемся диалоге оставить поле «Название группы» пустым. 4. Выбрать участников и нажать «Создать».	В окне появилось уведомление: «Введите название группы»
9	Создание группы без участников	Группа должна содержать хотя бы одного участника помимо создателя	1. Запустить приложение и выполнить вход. 2. Нажать кнопку создания группы («+») 3. Ввести название группы, не отмечать участников. 4. Нажать «ОК».	На экране появилось уведомление: «Выберите хотя бы одного участника!»

№	Наименование	Описание	Шаги для воспроизведения	Ожидаемый результат
10	Попытка изменения группы пользователем без прав администратора	Только администратор может открывать меню управления группой	1. Запустить приложение и выполнить вход под учётной записью, не являющейся администратором группы. 2. Открыть профиль группы. 3. Нажать кнопку «Управление группой».	Кнопка «Управление группой» не отображается.
11	Попытка отправки файла без выбранного чата	Файл можно отправить только в открытый чат	1. Запустить приложение и выполнить вход. 2. Не выбирая чат, нажать кнопку «Прикрепить файл».	На экране появилось уведомление: «Выберите чат»
12	Попытка скачать несуществующий файл	Файл должен быть загружен на сервер для скачивания	1. Запустить приложение и выполнить вход. 2. В истории сообщений найти файловое сообщение, файл которого был удалён с сервера. 3. Дважды нажать на сообщение.	На экране появилось уведомление: «Файл не найден на сервере»
13	Поиск пользователя в строке поиска по неполному совпадению	Поиск должен находить пользователей, чьи имена содержат введённую подстроку	1. Запустить приложение и выполнить вход. 2. В поле поиска в левой панели ввести часть имени существующего пользователя. 3. Дождаться отображения результатов.	В списке результатов отображаются пользователи с похожими именами.

1.7 Руководство пользователя

Чтобы начать работу с программой, необходимо пройти авторизацию (рисунок 36). Для этого в поля «Логин» и «Пароль» нужно ввести учетные данные пользователя, после чего нажать кнопку «Регистрация».

При необходимости указать адрес сервера можно нажав на строку «Настройки подключения» (рисунок 37).

Рисунок 36 – Окно входа

Рисунок 37 – Настройка IP-адреса сервера

Если регистрация прошла успешно, появится всплывающее окно (рисунок 38). В случае, когда логин уже занят или пароль слишком короткий, программа выдаст предупреждение об ошибке (рисунок 39).

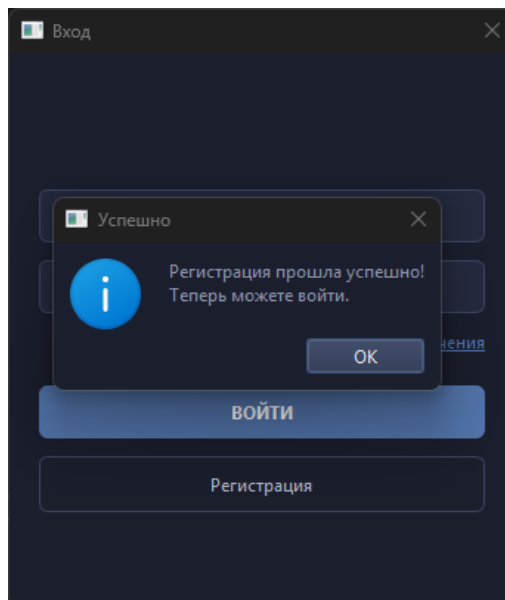


Рисунок 38 – Успешная регистрация

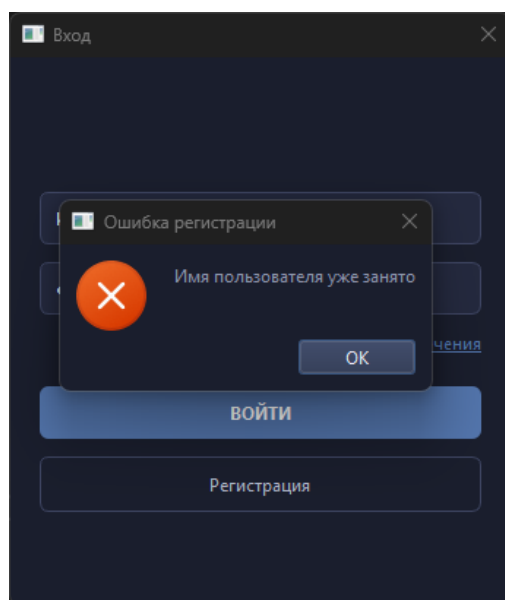


Рисунок 39 – Ошибка при регистрации

Для входа в существующую учётную запись требуется ввести логин и пароль и нажать кнопку «Войти». При совпадении данных откроется главное окно программы (рисунок 40). Если данные введены неверно, появится сообщение об ошибке (рисунок 41).

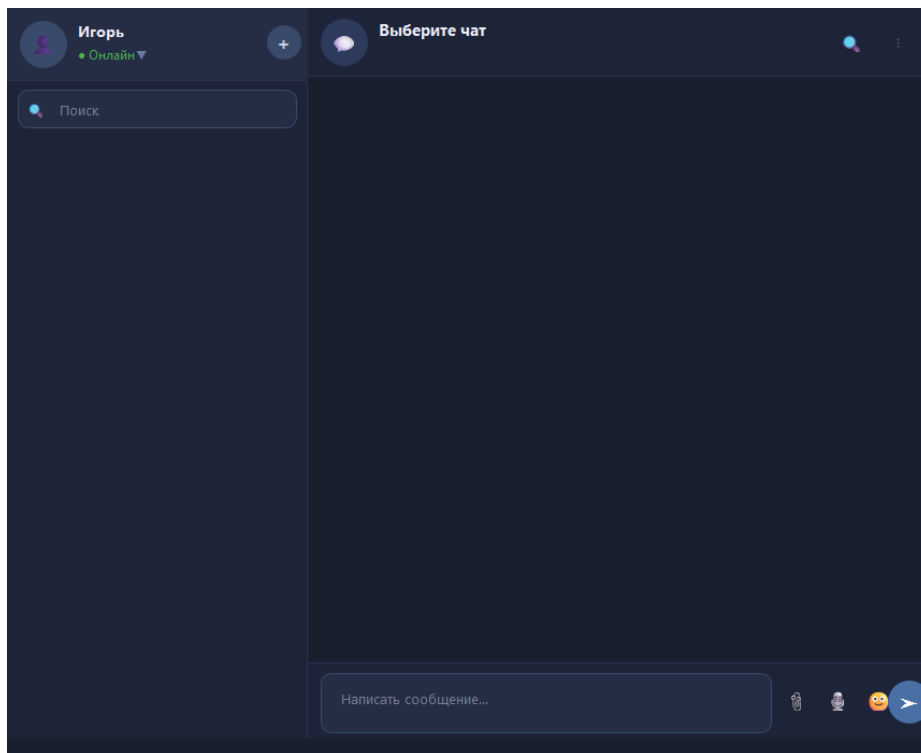


Рисунок 40 – Главное окно приложения

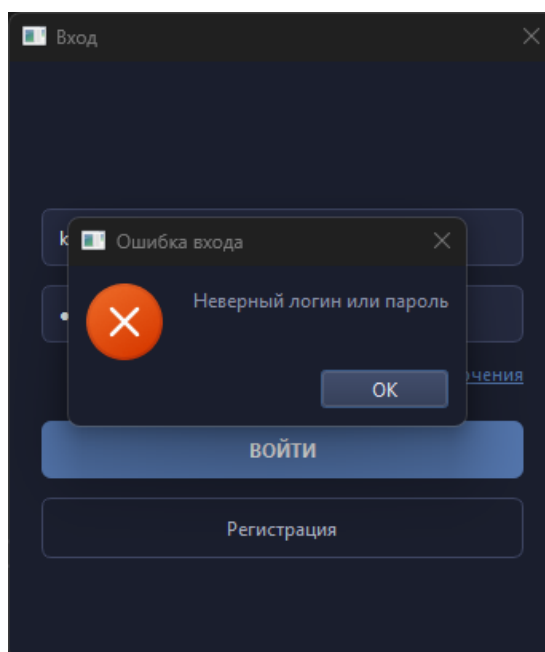


Рисунок 41 – Ошибка при входе

Для поиска пользователя или группы можно воспользоваться строкой поиска в левой панели. В списке отобразятся подходящие чаты и пользователи (рисунок 42). Нажав на найденного пользователя, программа откроет чат с ним (рисунок 43).

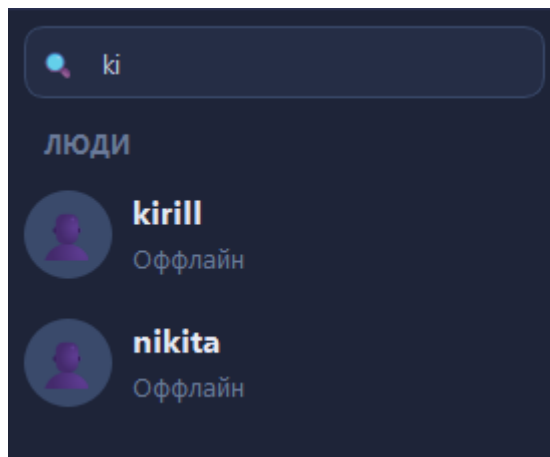


Рисунок 42 – Поиск пользователей

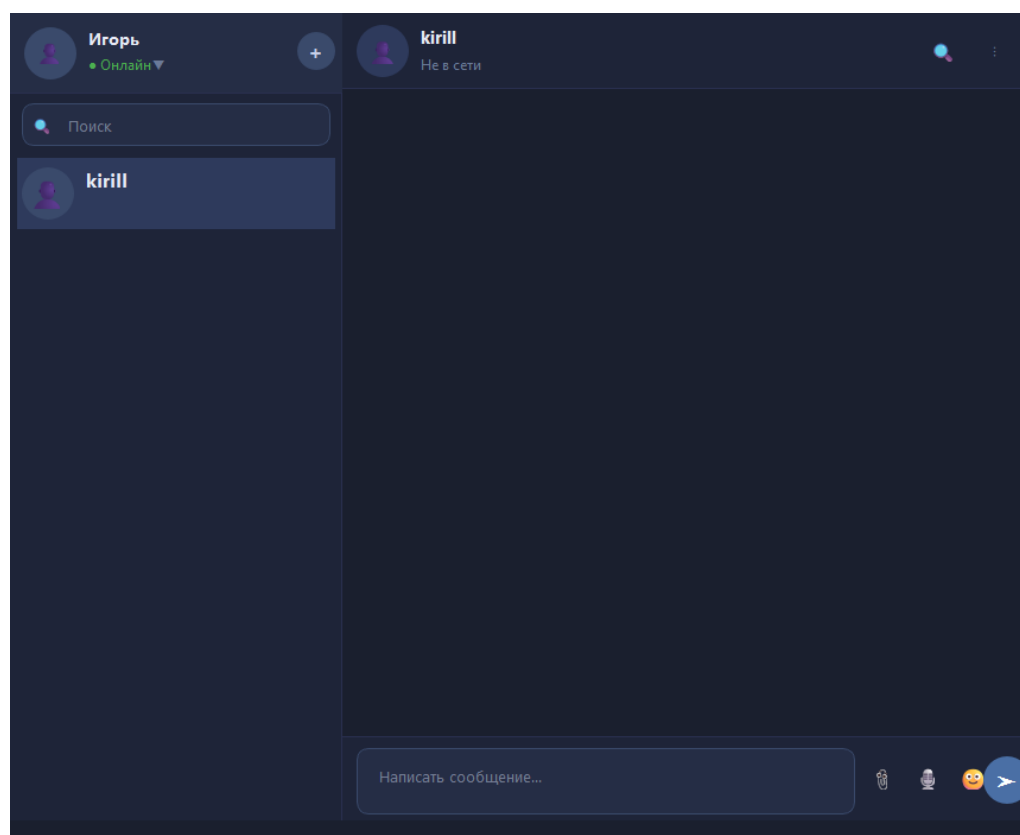


Рисунок 43 – Открытый чат

При нажатии на кнопку с тремя точками, можно увидеть информацию о пользователе (рисунок 44). Чтобы отправить сообщение, нужно напечатать текст в нижнем поле и нажать кнопку «Отправить» (или клавишу Enter). Сообщение сразу появится в чате (рисунок 46). Для вставки эмодзи можно открыть специальное меню, нажав на кнопку эмодзи рядом с полем ввода (рисунок 45).

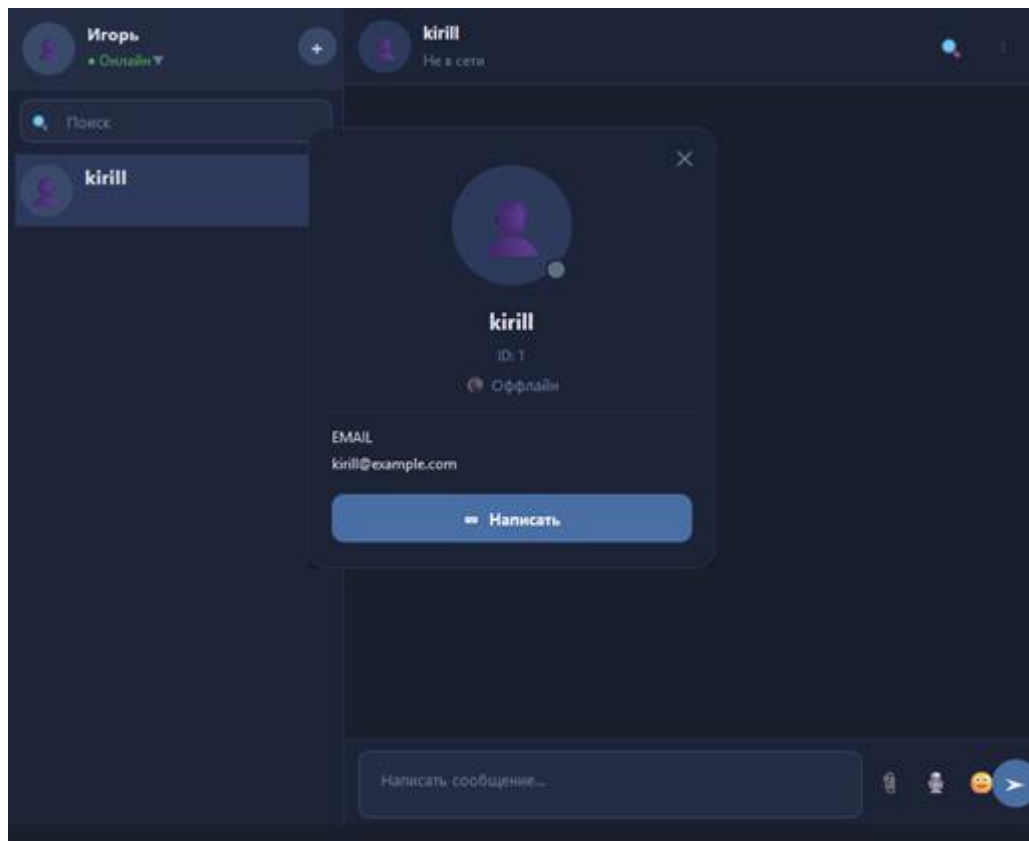


Рисунок 44 – Информация о пользователе

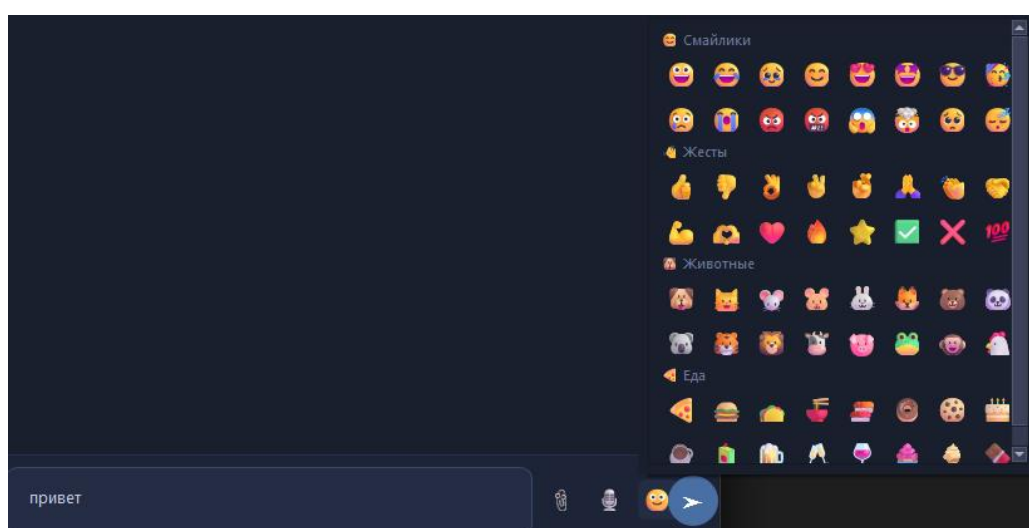


Рисунок 45 – Меню эмодзи

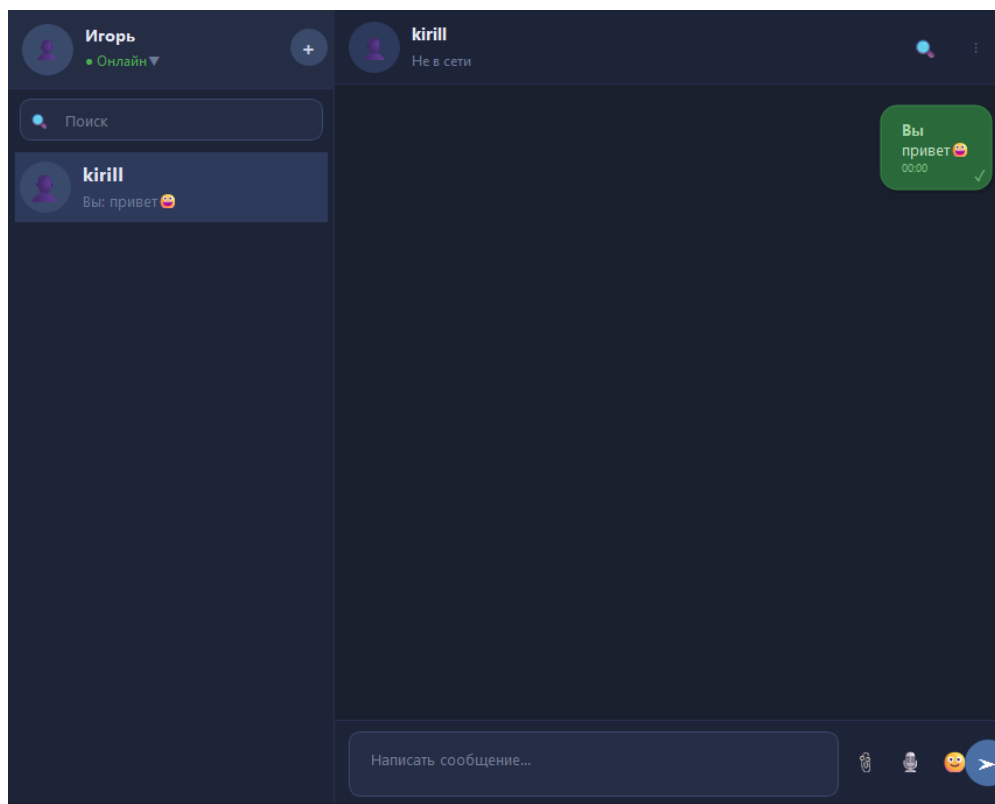


Рисунок 46 – Отправленное сообщение

Чтобы отправить файл, следует нажать кнопку «Прикрепить» (скрепка) в нижней панели и выбрать файл на компьютере. После завершения отправки файл отобразится в чате (рисунок 47).

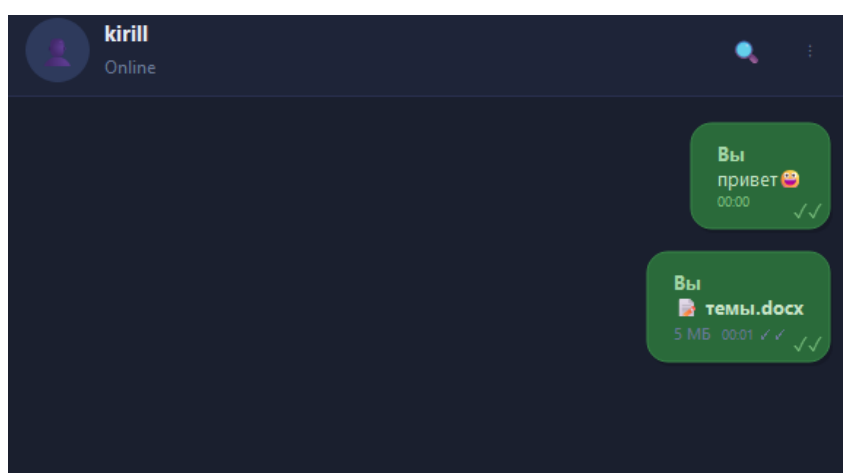


Рисунок 47 – Отправленный файл

При необходимости можно ответить на конкретное сообщение. При нажатии правой кнопки мыши на сообщение откроется контекстное меню (рисунок 48). В нём следует выбрать «Ответить». Над полем ввода появится строка с текстом (рисунок 49). После отправки ответ будет отображаться со ссылкой на исходное сообщение (рисунок 50).

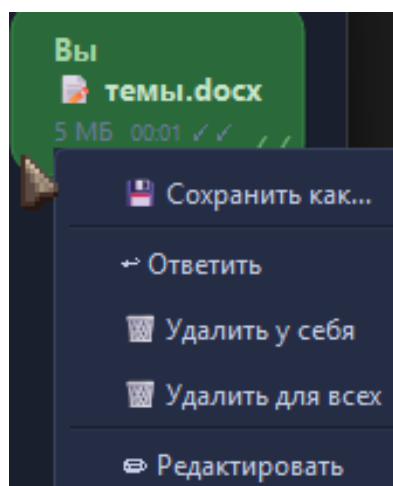


Рисунок 48 – Контекстное меню сообщений



Рисунок 49 – Ответ на сообщение

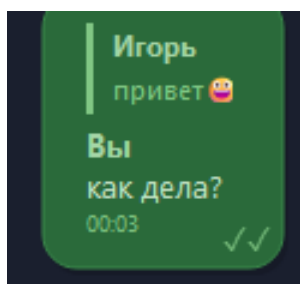


Рисунок 50 – Отображение ответа со ссылкой

Своё сообщение можно отредактировать или удалить через контекстное меню (рисунки 51, 52).

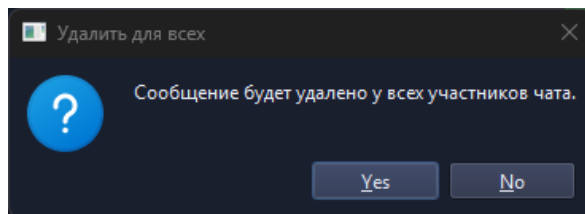


Рисунок 51 – Удаление сообщения



Рисунок 52 – Сообщение удалено

Если при нажатии на кнопку «Прикрепить» выбрать изображение, после загрузки, оно будет показано в виде уменьшенной копии (рисунок 53).

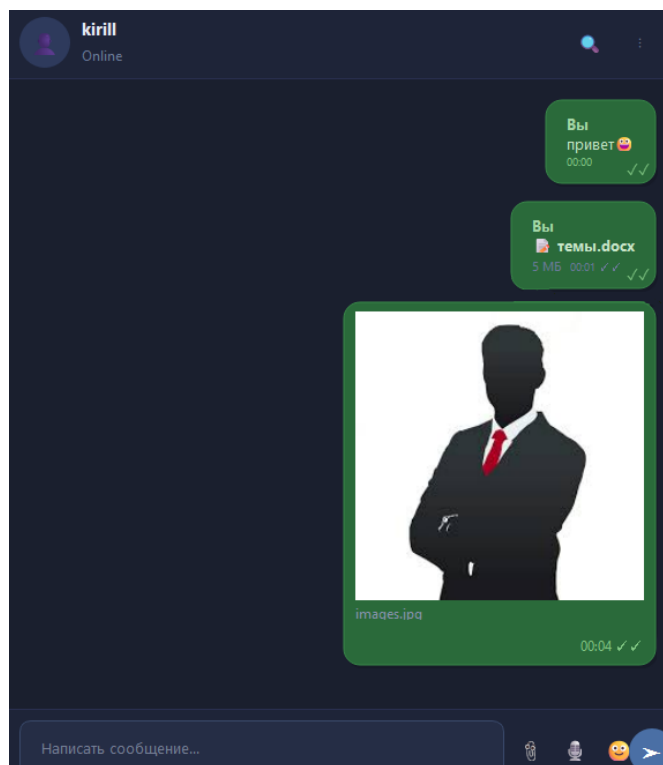


Рисунок 53 – Сообщение с изображением

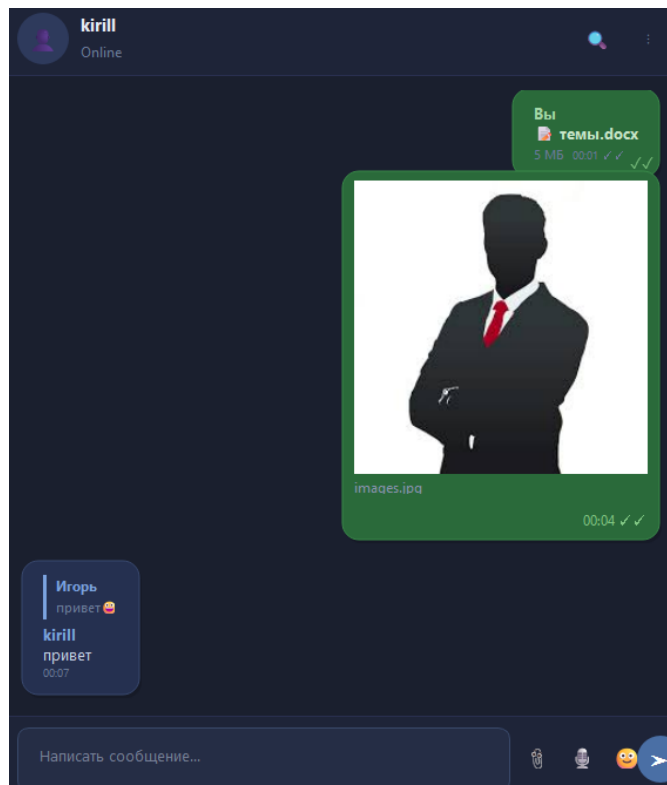


Рисунок 54 – Сообщение другого пользователя

Для записи голосового сообщения нужно зажать кнопку микрофона в панели ввода (рисунок 55). Когда кнопка будет отпущена сообщение сразу отправится (рисунок 56). При нажатии на кнопку прослушивания, файл сообщения загрузится с сервера (рисунок 57) и начнётся воспроизведение (рисунок 58), повторное нажатие приостановит воспроизведение.

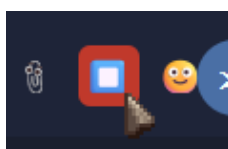


Рисунок 55 – Запись голосового сообщения

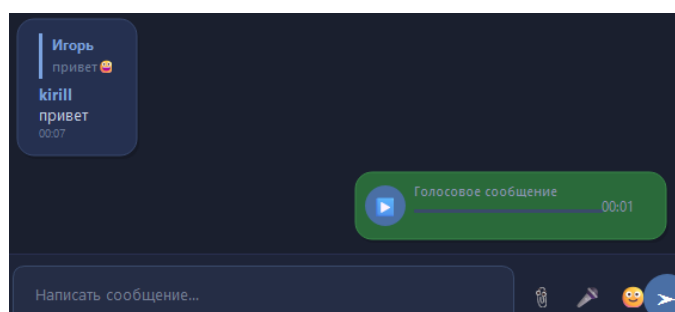


Рисунок 56 – Голосовое сообщение

Downloading voice_1778015300245.wav: 0%

Рисунок 57 – Статус загрузки файла

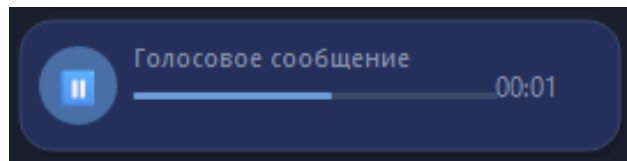


Рисунок 58 – Прослушивание голосового сообщения

Создать группу можно, нажав кнопку «+» в левой панели (рисунок 59). В открывшемся окне (рисунок 60) требуется задать название группы и отметить участников из списка контактов. После создания группа появится в общем списке чатов.

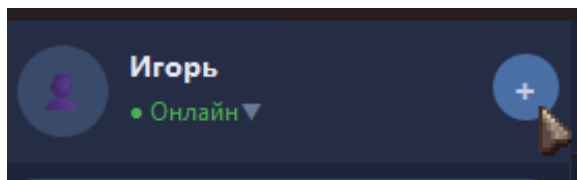


Рисунок 59 – Кнопка создания группы

Рисунок 60 – Меню создания группы

Информация о группе доступна через кнопку «⋮» в заголовке чата. Будет открыто окно с описанием группы, списком участников и кнопкой управления (рисунок 61).

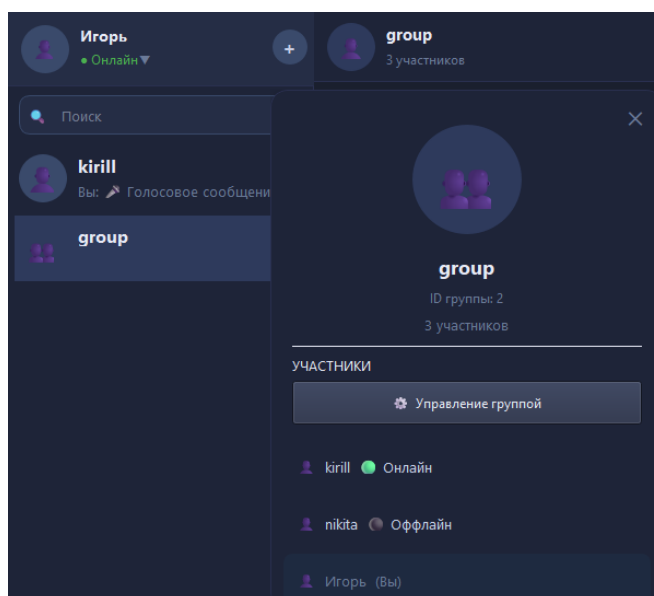


Рисунок 61 – Информация о группе

Если пользователь является администратором группы, ему доступно меню управления группой (рисунок 62), где можно назначить других участников администраторами (рисунки 63, 64) добавить или удалить пользователей (рисунки 65, 66).

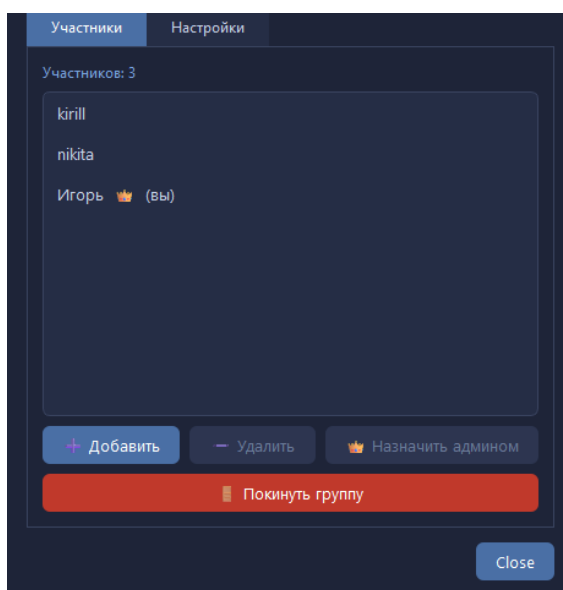


Рисунок 62 – Меню управления группой

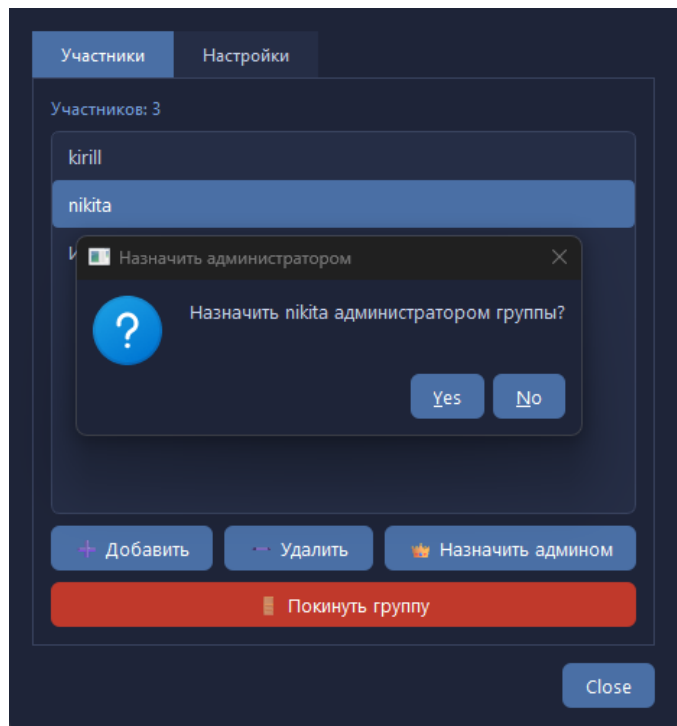


Рисунок 63 – Назначение пользователя администратором группы

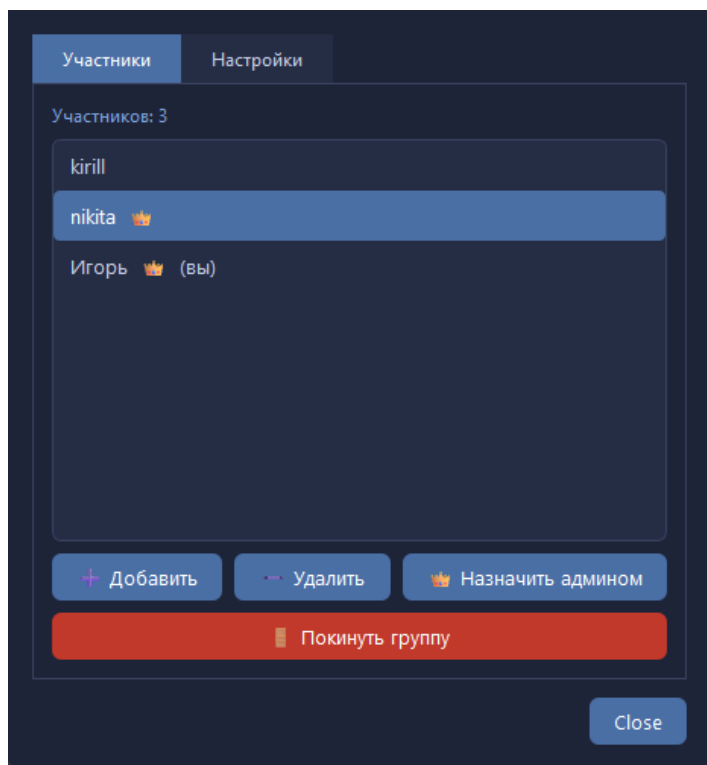


Рисунок 64 – Пользователь назначен администратором

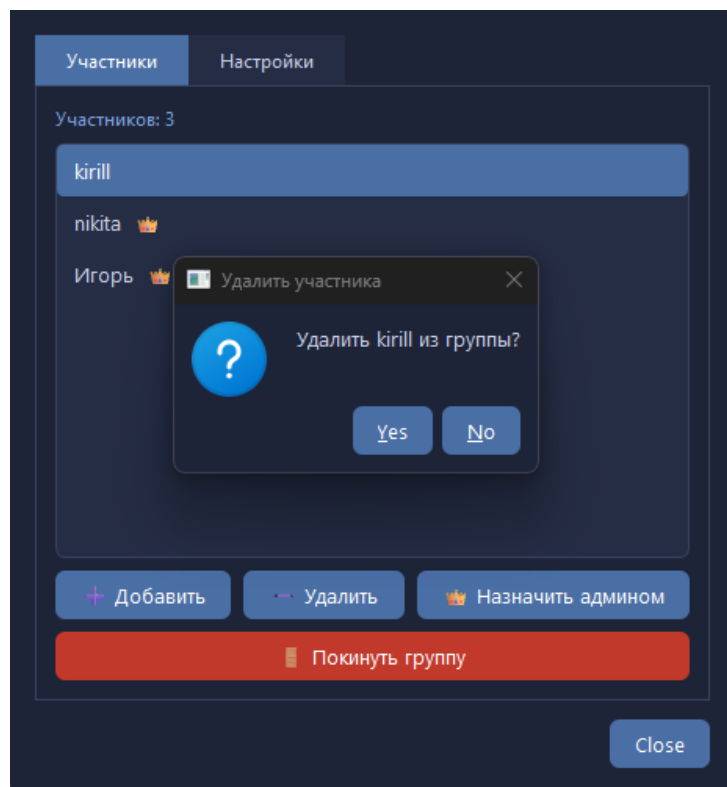


Рисунок 65 – Удаление пользователя из группы

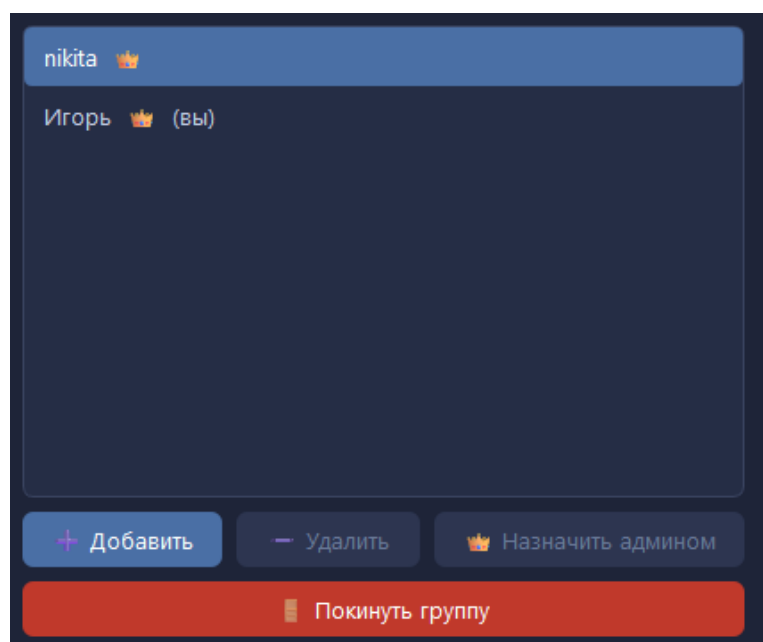


Рисунок 66 – Пользователь удален из группы

Также администратору доступно меню настроек группы (рисунок 67) может изменять название (рисунок 68) и аватар группы (рисунок 69).

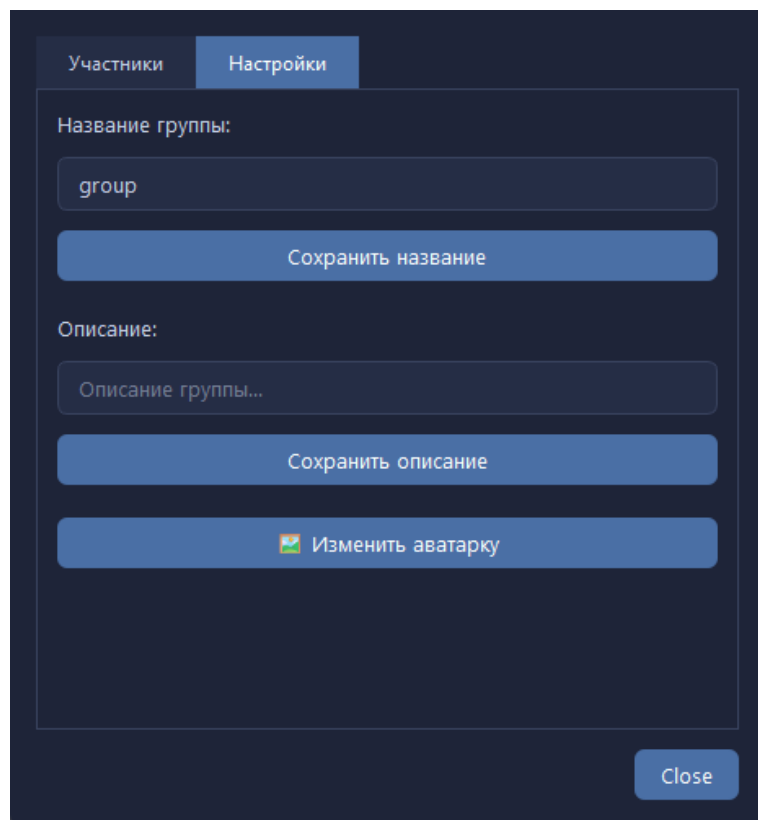


Рисунок 67 – Меню настроек группы

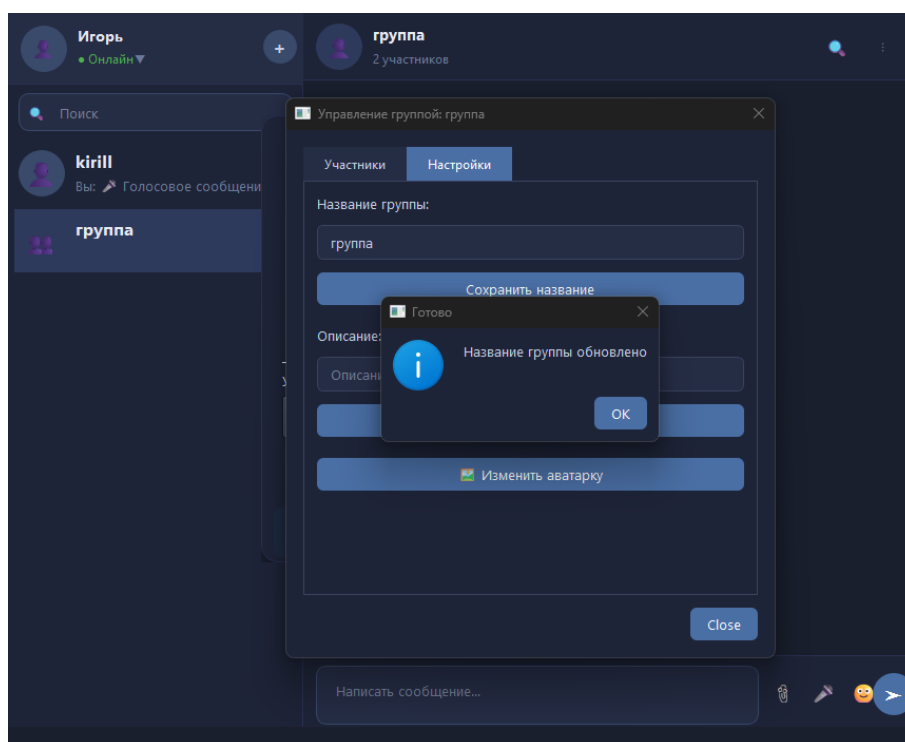


Рисунок 68 – Изменение имени группы

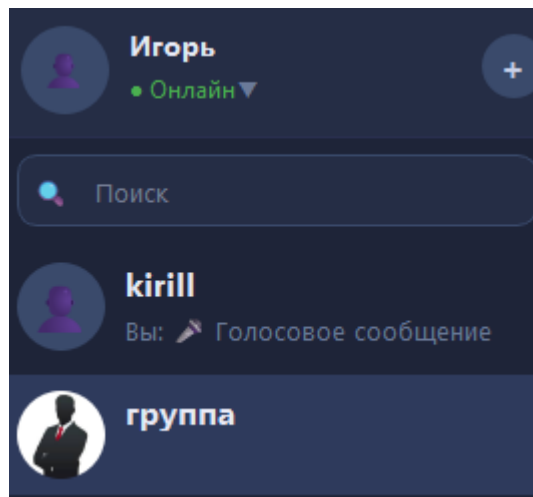


Рисунок 69 – Изменение аватара группы

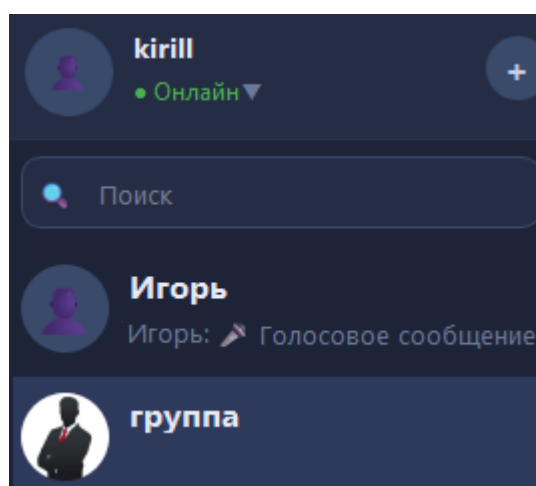


Рисунок 70 –Аватар группы от лица другого пользователя

Чтобы изменить собственный аватар, пользователю нужно навести мышь на область со своим именем и аватаркой в левой панели, появится меню (рисунок 71). Чтобы сменить аватар, требуется указать нужное изображение (рисунок 72). Для завершения работы с программой в меню можно нажать «Выйти».

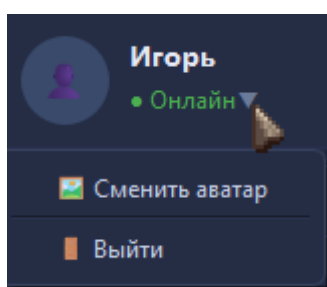


Рисунок 71 – Контекстное меню пользователя

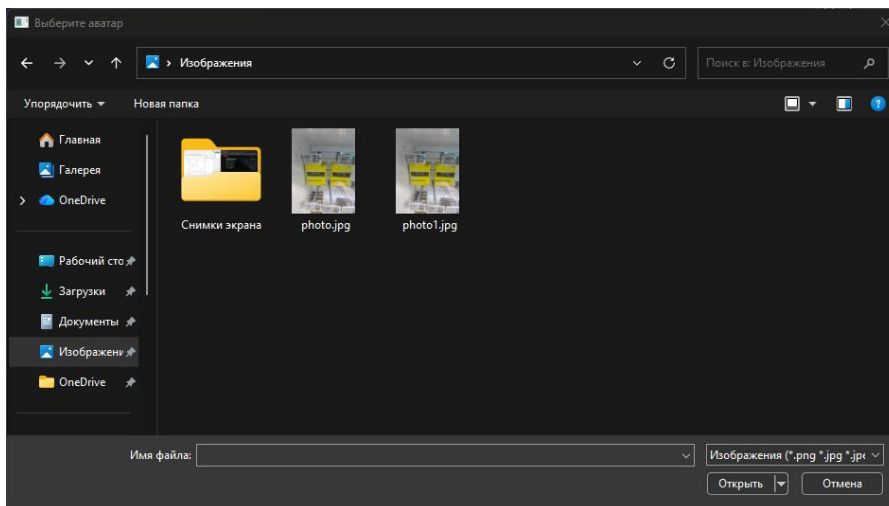


Рисунок 72 – Выбор файла для аватара

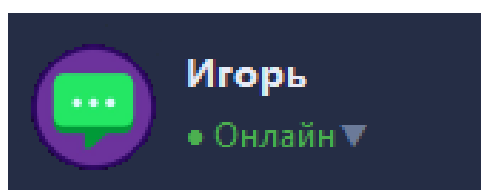


Рисунок 73 – Изменённый аватар

2 Безопасность жизнедеятельности

2.1 Общие положения

Работа с персональным компьютером связана с воздействием на пользователя ряда вредных и опасных факторов, способных оказывать негативное влияние на здоровье и работоспособность человека. К таким факторам относятся повышенная зрительная нагрузка, статическое напряжение мышц, психоэмоциональное перенапряжение, воздействие электромагнитного излучения и неблагоприятные параметры микроклимата.

Вопросы обеспечения безопасных условий труда регулируются Трудовым кодексом Российской Федерации, санитарными правилами СП 2.2.3670-20 «Санитарно-эпидемиологические требования к условиям труда.

2.2 Опасные и вредные факторы при работе за ПК

При работе за персональным компьютером на пользователя воздействуют следующие вредные производственные факторы:

- длительное напряжение органов зрения;
- статическая нагрузка на позвоночник и мышцы шеи;
- монотонность выполняемой работы;
- повышенное нервно-эмоциональное напряжение;
- недостаточная освещённость рабочего места;
- риск поражения электрическим током.

Длительная работа за компьютером может привести к ухудшению зрения, развитию остеохондроза, синдрому хронической усталости и заболеваниям опорно-двигательного аппарата [6].

2.3 Требования к организации рабочего места

Организация рабочего места должна соответствовать эргономическим требованиям и обеспечивать комфортные условия труда.

					ПГУ 1.090301.25.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

Монитор рекомендуется располагать на расстоянии 50–70 см от глаз пользователя. Верхняя граница экрана должна находиться на уровне глаз или немного ниже. Клавиатура размещается на расстоянии 10–15 см от края стола. Рабочее кресло должно иметь регулировку высоты и угла наклона спинки.

Освещение рабочего места должно быть достаточным и равномерным. Рекомендуемый уровень освещённости составляет 300–500 лк. Не допускается появление бликов на экране монитора.

Ниже представлена схема правильного положения тела при работе с персональным компьютером.

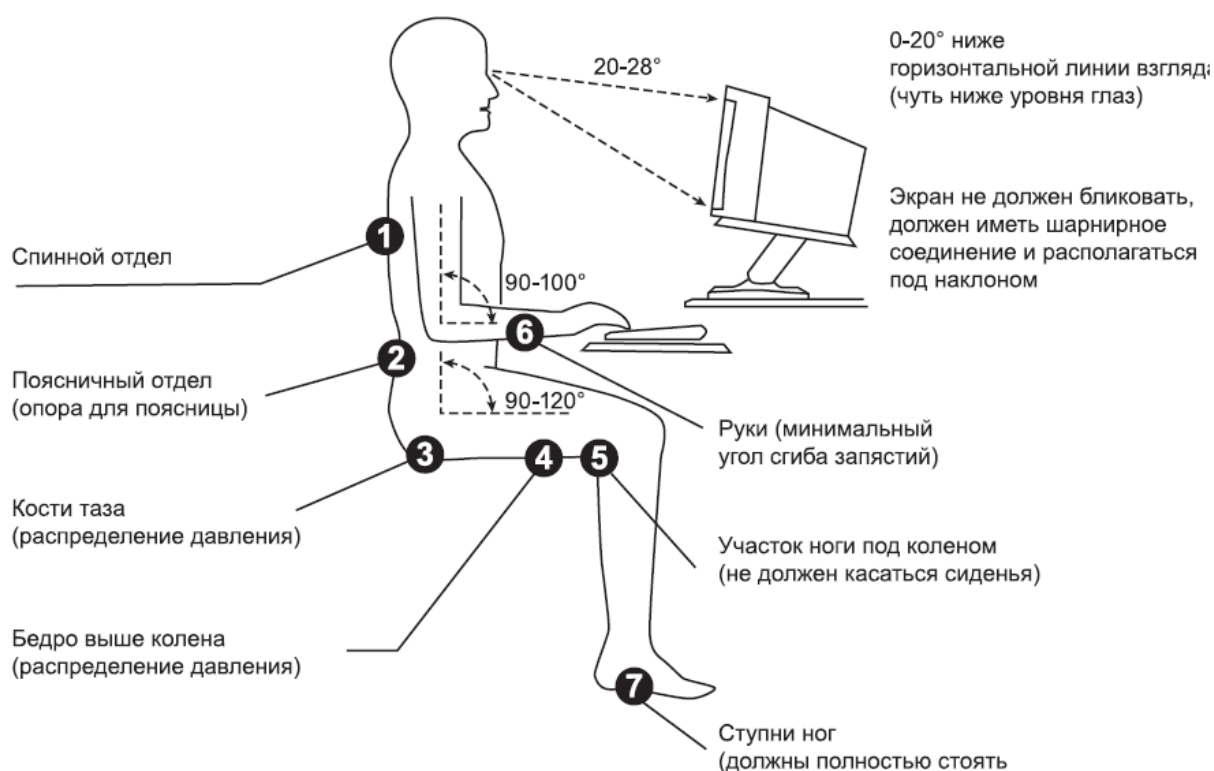


Рисунок 74 – Схема правильного положения тела при работе с персональным компьютером

2.4 Электробезопасность

Персональный компьютер является электротехническим оборудованием, поэтому при его эксплуатации необходимо соблюдать требования электробезопасности.

Пользователю запрещается:

- использовать неисправное оборудование;
- работать с повреждёнными кабелями и розетками;
- прикасаться к оборудованию влажными руками;
- самостоятельно разбирать устройства, подключённые к электросети.

Согласно требованиям охраны труда, оборудование должно быть подключено к исправной сети электропитания с заземлением. В случае обнаружения запаха гари, перегрева оборудования или искрения необходимо немедленно отключить питание и сообщить ответственному лицу.

2.5 Режим труда и отдыха

Для снижения утомляемости пользователей ПК необходимо соблюдать рациональный режим труда и отдыха. При продолжительной работе рекомендуется делать перерывы продолжительностью 5–10 минут после каждого часа работы.

Во время перерывов рекомендуется выполнять упражнения для глаз и разминку для мышц шеи, спины и кистей рук.

Соблюдение режима труда и отдыха способствует снижению зрительного и нервно-мышечного напряжения. На рисунке 75 представлены примеры упражнений для снижения зрительного напряжения

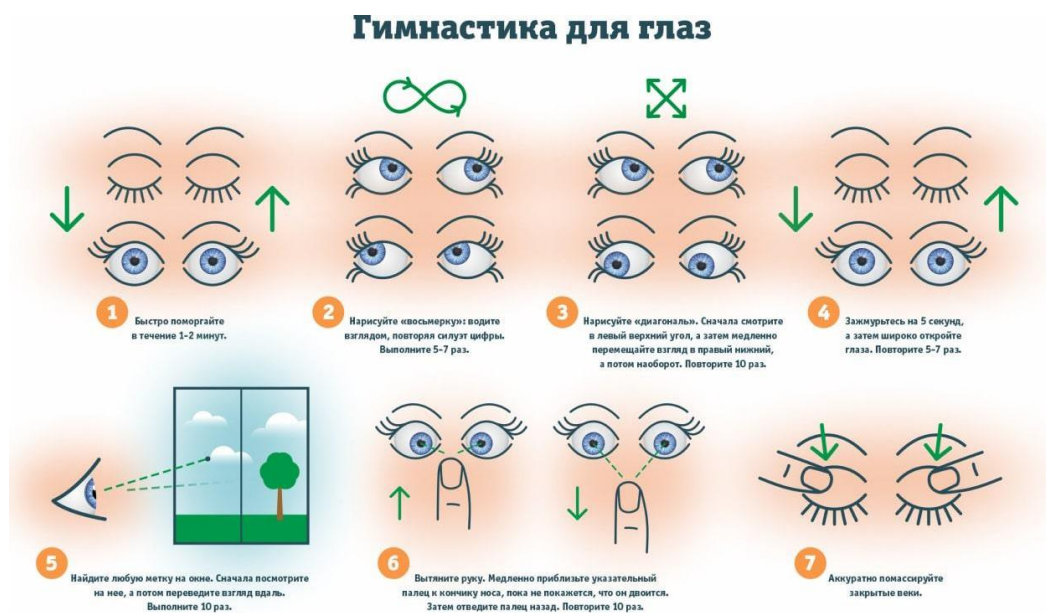


Рисунок 75 – Упражнения для снижения зрительного напряжения

Изм.	Лист	№ докум.	Подпись	Дата

2.6 Пожарная безопасность

Помещение, в котором размещается компьютерная техника, должно быть оборудовано средствами пожаротушения и соответствовать требованиям пожарной безопасности.

Запрещается:

- перегружать электрическую сеть;
- использовать неисправные удлинители;
- размещать легковоспламеняющиеся материалы рядом с оборудованием;
- оставлять включённую технику без присмотра.

При возникновении пожара необходимо:

- отключить электропитание;
- сообщить о возгорании в пожарную охрану;
- использовать первичные средства пожаротушения при отсутствии угрозы жизни;
- организовать эвакуацию людей из помещения.

Требования пожарной безопасности устанавливаются Федеральным законом от 21 декабря 1994 г. № 69-ФЗ «О пожарной безопасности».

					ПГУ 1.090301.25.001 ПЗ	Лист
						73
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В процессе выполнения задания были получены навыки разработки клиент-серверных приложений в среде Microsoft Visual Studio 2026 с использованием языка программирования C++ и фреймворка Qt, изучены методы проектирования сетевых приложений и организации взаимодействия между клиентом и сервером. За период выполнения работы были освоены основные технологии разработки графических интерфейсов, сетевого взаимодействия, обработки JSON-данных и работы с базами данных SQLite, изучены принципы обеспечения безопасности сетевого взаимодействия.

Во время выполнения данной работы были разработаны клиентская и серверная части приложения для обмена сообщениями, реализован графический интерфейс пользователя, система авторизации и регистрации, безопасный обмен данными через TLS-соединение, обмен текстовыми сообщениями, файлами и голосовыми сообщениями.

В дальнейшем проект может быть расширен путём внедрения сквозного шифрования сообщений, поддержки видеозвонков, мобильных клиентов, синхронизации между устройствами и интеграции дополнительных механизмов защиты и администрирования.

					ПГУ 1.090301.25.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1.Wikipedia.org [Электронный ресурс] / Режим доступа: <https://ru.wikipedia.org/>, свободный (дата обращения 10.05.2026)

2.Прохоренок Н. А. Qt 6. Разработка оконных приложений на C++. – Санкт-Петербург : БХВ-Петербург, 2022. – 512 с.

3.Doc.qt.io [Электронный ресурс] / Режим доступа: <https://doc.qt.io/>, свободный (дата обращения: 11.03.2026).

4.Фримен Э., Робсон Э., Сьерра К., Бейтс Б. Head First. Паттерны проектирования. – Санкт-Петербург : Питер, 2018. – 656 с.

5.Ключи, шифры, сообщения: как работает TLS [Электронный ресурс] / Режим доступа: <https://tls.dxdt.blog/tls.html>, свободный (дата обращения: 28.04.2026)

6.Об утверждении санитарных правил СП 2.2.3670-20 «Санитарно-эпидемиологические требования к условиям труда» : Постановление Главного государственного санитарного врача РФ от 02.12.2020 № 40. [Электронный ресурс] / Режим доступа: <https://docs.cntd.ru/document/573230583>, свободный (дата обращения 21.05.2026).

Приложение А

(обязательное)

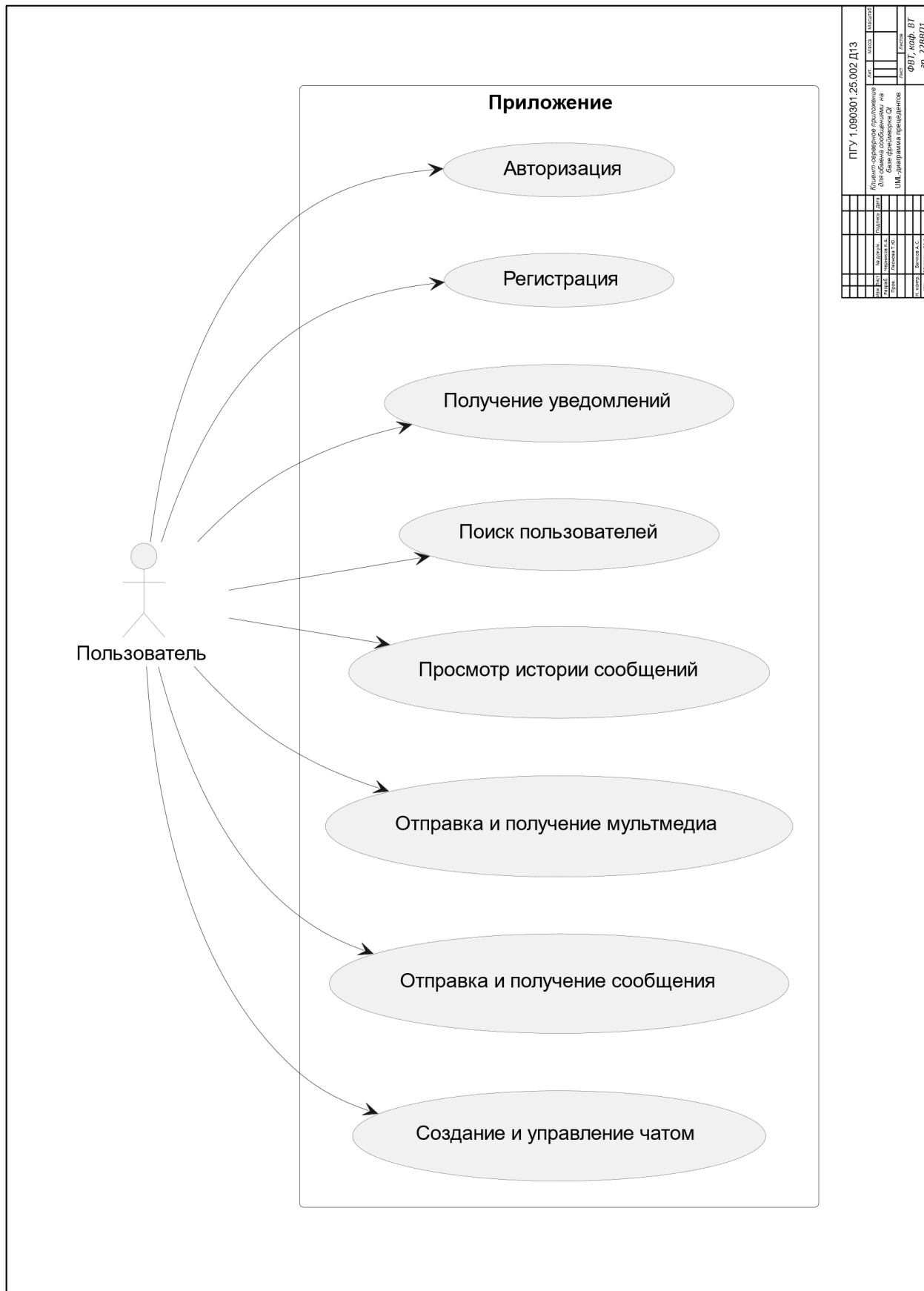
Клиент-серверное приложение обмена сообщениями

на базе фреймворка Qt

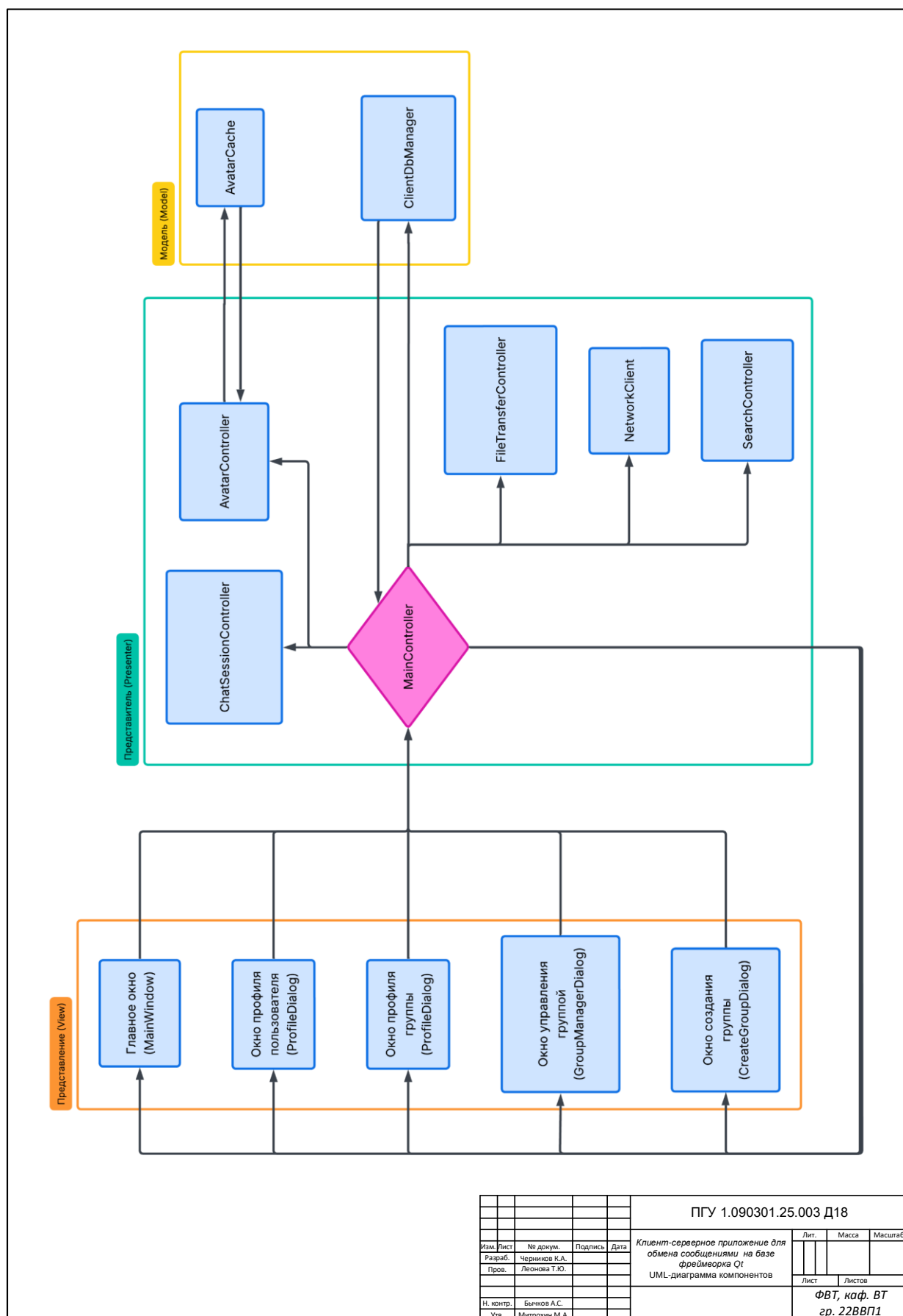
UML диаграммы и схемы

					ПГУ 1.090301.25.001 ПЗ	Лист
						76
Изм.	Лист	№ докум.	Подпись	Дата		

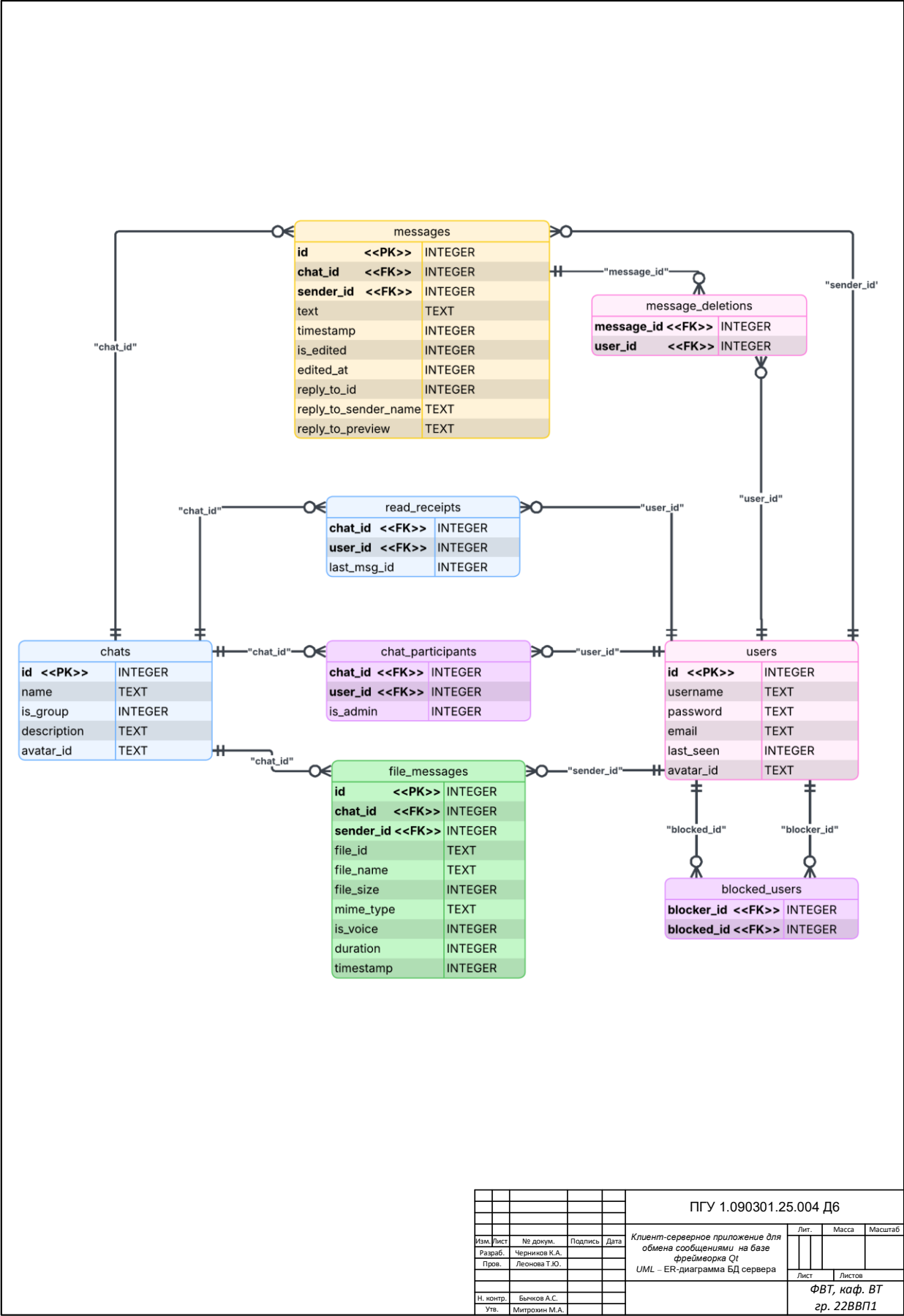
Приложение А.1 – UML – диаграмма прецедентов



Приложение А.2 – UML – диаграмма компонентов приложения

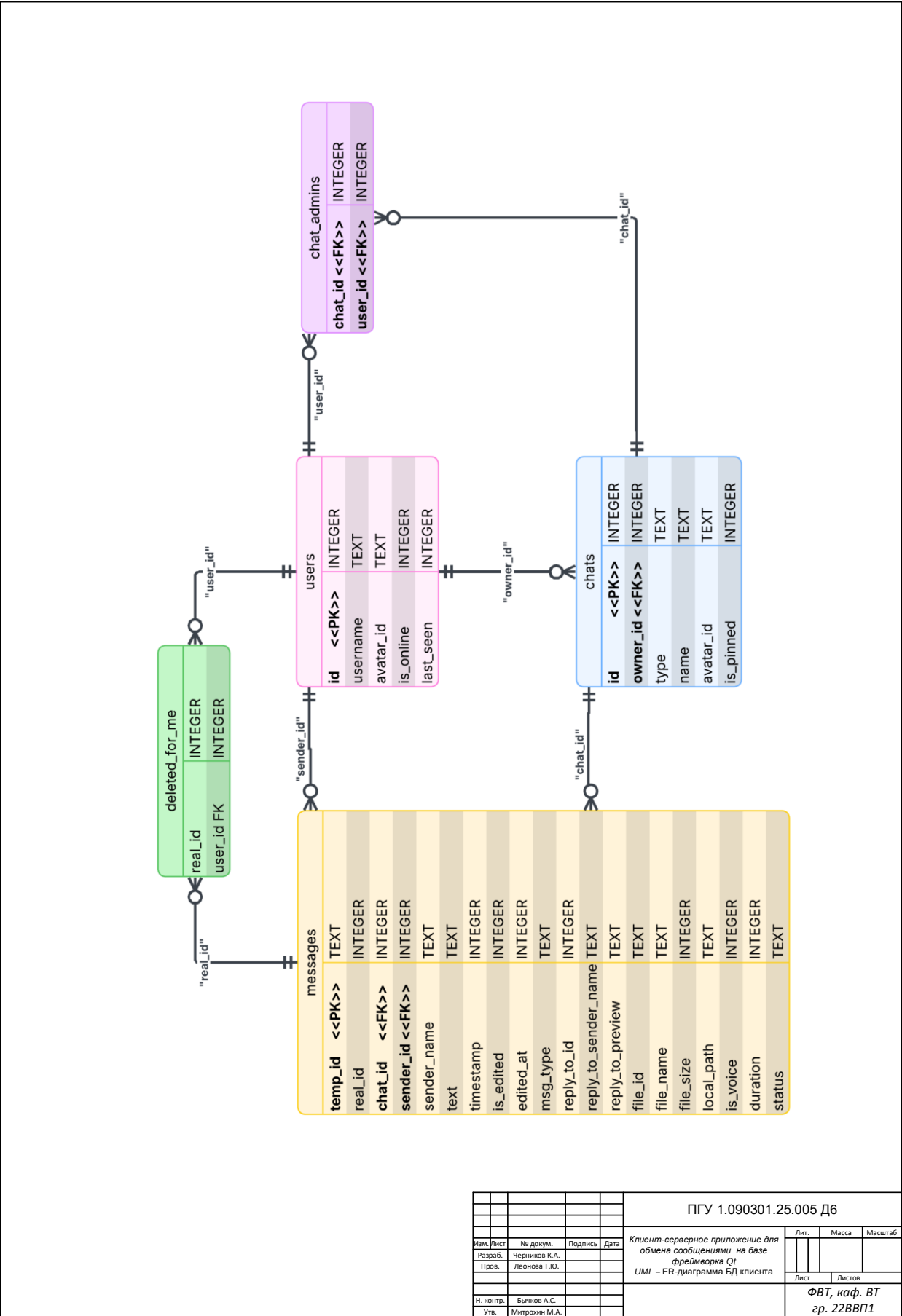


Приложение А.3 – UML – ER-диаграмма БД сервера

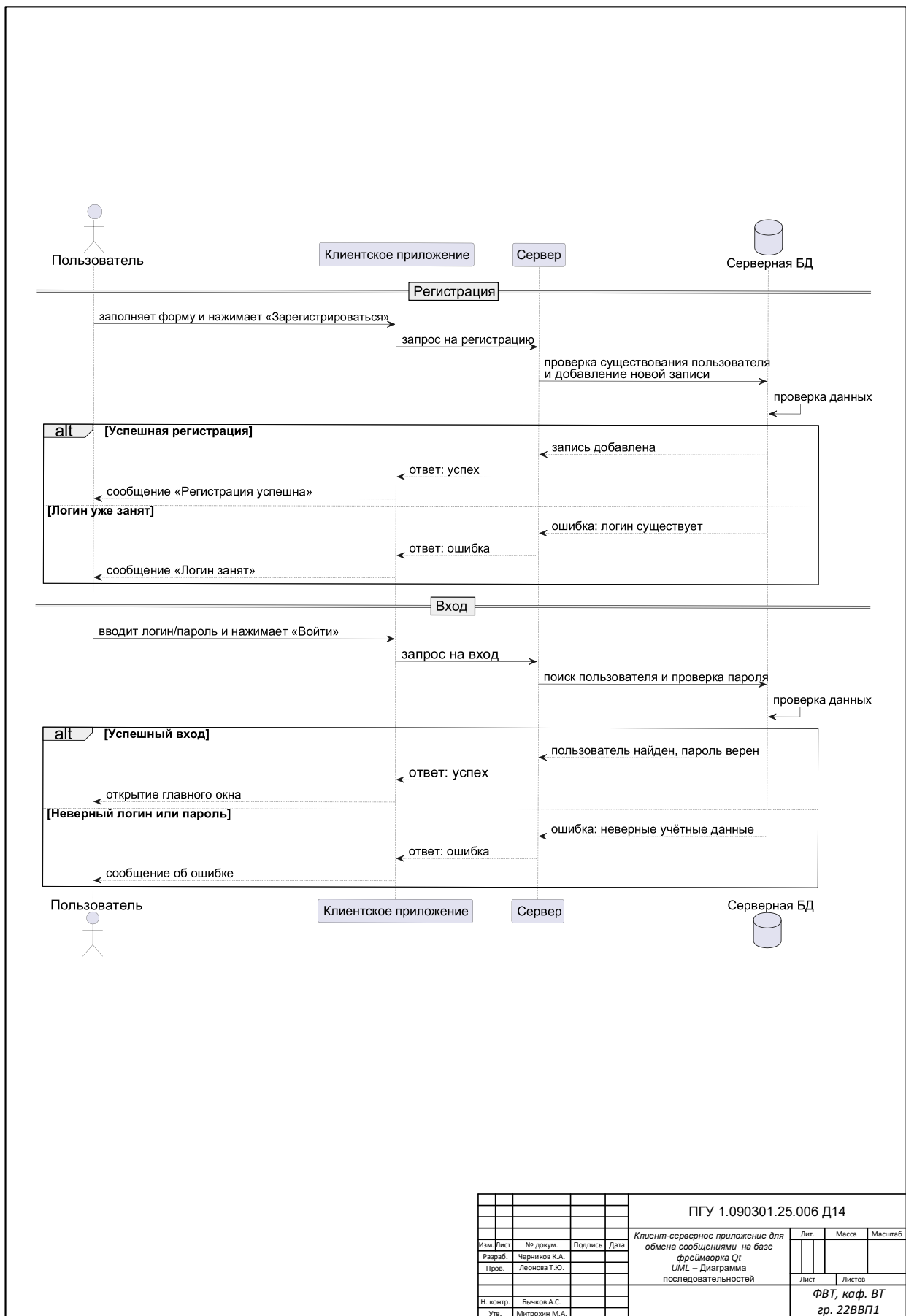


						ПГУ 1.090301.25.004 Д6			
Изм.	Лист	№ докум.	Подпись	Дата		Клиент-серверное приложение для обмена сообщениями на базе фреймворка Qt	Лит.	Масса	Масштаб
						UML – ER-диаграмма БД сервера			
Разраб.	Черников К.А.						Лист	Листов	
Пров.	Леонова Т.Ю.								
Н. контр.	Бычков А.С.								ФВТ, коп. ВТ
Утв.	Митрохин М.А.								гр. 22ВВП1

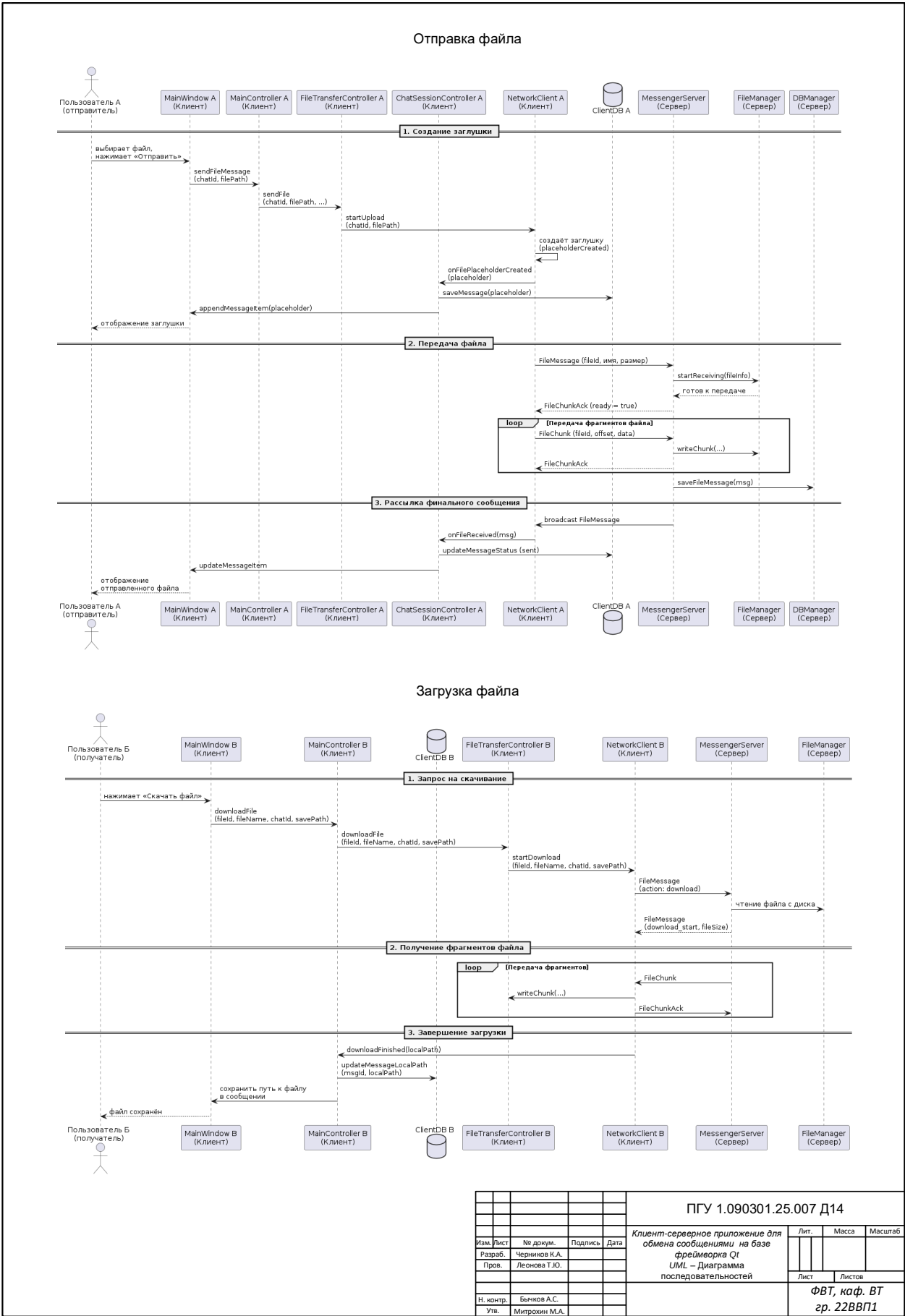
Приложение А.4 – UML – ER-диаграмма БД клиента



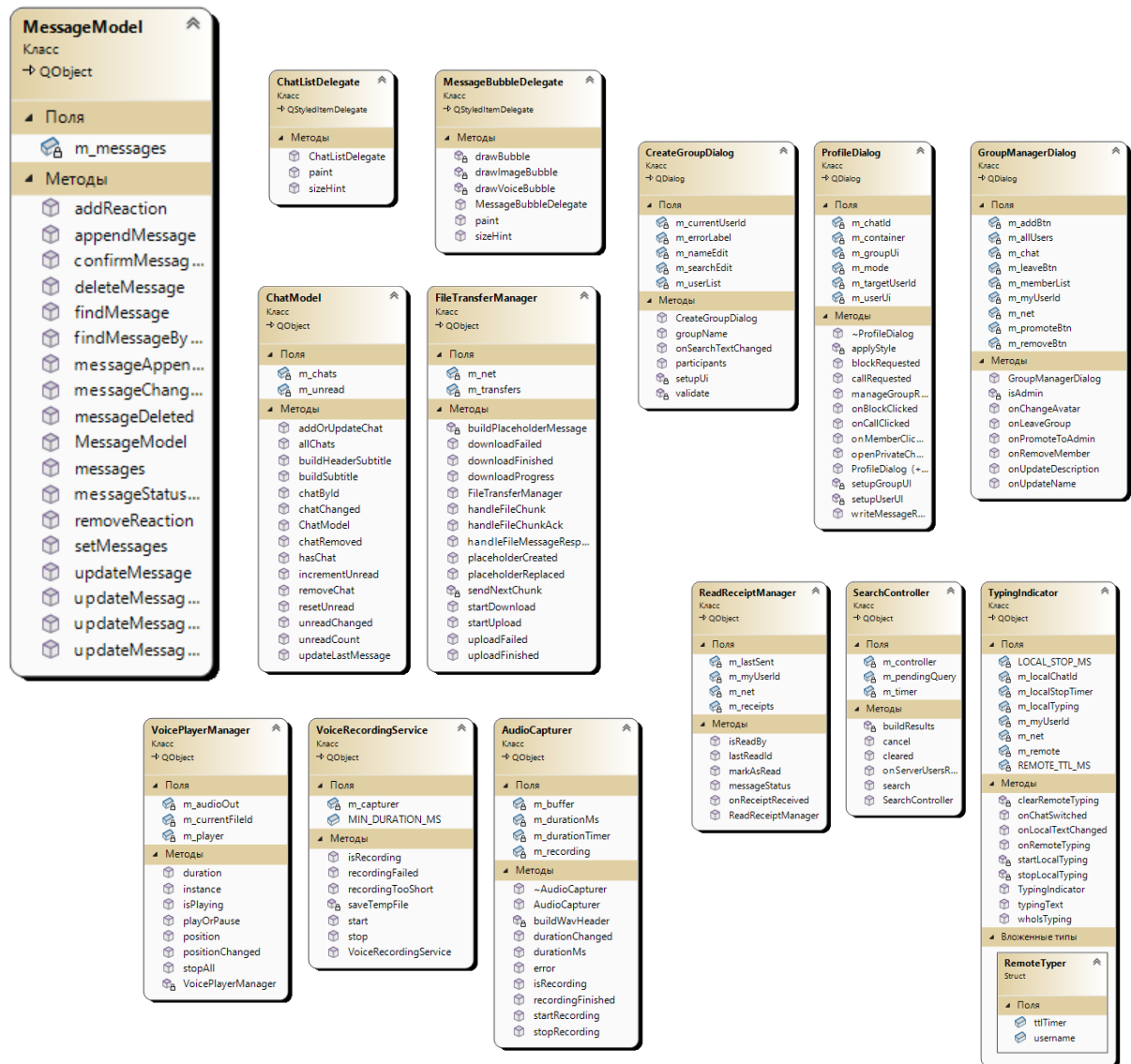
Приложение А.5 – UML – Диаграмма последовательностей. Регистрация и вход



Приложение А.6 – UML – Диаграммы последовательностей



Приложение А.7 – UML – Диаграммы классов



ПГУ 1.090301.25.008 Д11			
Изм.	Лист	№ докум.	Подпись
Разраб.	Черников К.А.	Леонова Т.Ю.	Дата
Пров.	Леонова Т.Ю.		
Н. контр.	Бычков А.С.		
Утв.	Митрохин М.А.		

Клиент-серверное приложение для
обмена сообщениями на базе
фреймворка Qt
UML – Диаграммы классов

Лит. Масса Масштаб
Лист Листов

ФВТ, каф. ВТ
гр. 22ВВП1

Изм.	Лист	№ докум.	Подпись	Дата
------	------	----------	---------	------

ПГУ 1.090301.25.001 ПЗ

Лист

83

Приложение Б

(обязательное)

Клиент-серверное приложение обмена сообщениями

на базе фреймворка Qt

Листинг программы

					ПГУ 1.090301.25.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «MainWindow.h»

```
#pragma once
#include <QMainWindow>
#include <QListWidgetItem>
#include <QPointer>
#include "controllers/MainController.h"
#include "delegates/messagebubbldelegate.h"
#include "delegates/chatlistdelegate.h"
#include "widgets/chatdroparea.h"
#include "controllers/SearchController.h"
#include "core/MessageRenderer.h"
#include "widgets/emojipicker.h"
#include "widgets/EditBar.h"
#include "IMainView.h"
#include "models/ChatDisplayData.h"
#include "controllers/tempItemDataRole.h"
#include "views/widgets/replypreviewwidget.h"

namespace Ui { class MainWindowClass; }

class MainWindow : public QMainWindow, public IMainView {
    Q_OBJECT
public:
    explicit MainWindow(QWidget* parent = nullptr);
    ~MainWindow();

    void initialize(MainController* controller);
    void addChatItem(const ChatDisplayData& data) override;
    void updateChatItem(int chatId, const ChatDisplayData& data) override;
    void removeChatItem(int chatId) override;
    void setChatUnreadCount(int chatId, int count) override;
    void setMyAvatar(const QPixmap& avatar) override;

    void setMessageList(const QList<QListWidgetItem*>& items) override;
    void appendMessageItem(QListWidgetItem* item) override;
    void updateMessageItem(int index, QListWidgetItem* item) override;
    void removeMessageItem(int index) override;
    void clearMessageList() override;
    void scrollToBottom() override;

    void setChatHeader(const QString& title, const QString& subtitle, const QPixmap&
avatar) override;
    void setConnectionStatus(bool online, const QString& message) override;
    void setTypingIndicator(const QString& text) override;

    void showSearchResults(const QList<SearchResult>& results) override;
    void hideSearch() override;
    void showStatusMessage(const QString& message, int timeoutMs) override;

    void showUserProfile(int userId) override;
    void showGroupProfile(int chatId) override;

    int currentChatId() const override;
    void activateChat(int chatId) override;

    void repaintChatList() override;
    void updateMessagePreview(const QString& fileId, const QPixmap& preview)
override;
    int messageListWidth() const override;
signals:
    void logoutRequested();
```

					ПГУ 1.090301.25.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		85

```

private slots:
    void onSearchResultsAvailable(int chatId, const QList<Message>& results);

    void onChatSelected(QListWidgetItem* item);
    void onSendMessageClicked();
    void onAttachClicked();
    void onVoiceButtonPressed();
    void onVoiceButtonReleased();
    void onChatInfoClicked();
    void onCreateGroupClicked();
    void onSearchTextChanged(const QString& text);
    void onLogoutClicked();
    void onMessageContextMenu(const QPoint& pos);
    void onMessageDoubleClicked(QListWidgetItem* item);
    void onLocalTextChanged();
    void onFileDropped(const QString& filePath);
    void onFileUploadProgress(const QString& fileId, int percent);
    void onFileDownloadProgress(const QString& fileId, const QString& fileName, int
percent);

    void onMessageClicked(QListWidgetItem* item);

protected:
    bool eventFilter(QObject* obj, QEvent* event) override;

private:
    SearchController* m_search;
    void addSearchResultItem(const SearchResult& r);
    void addSectionHeader(const QString& title);
    void addSeparator();
    void setupInitialState();
    void setupDelegates();
    void setupUiConnections();
    void setupDropArea();
    void updateChatHeader(const QString& name, const QString& subtitle, bool
isGroup);

    void setupEmojiPicker();
    void onSearchResultClicked(QListWidgetItem* item);
    void showUserMenu();
    void applySearchResults(const QList<SearchResult>& results);
    void fillFileItemData(QListWidgetItem* item,
        const Message& msg,
        const MessageRenderer::Result& rendered);

    ReplyPreviewWidget* m_replyPreview = nullptr;
    Ui::MainWindowClass* ui;
    MainController* m_controller;
    MessageBubbleDelegate* m_msgDelegate;
    ChatListDelegate* m_chatDelegate;
    ChatDropArea* m_dropArea;
    MessageRenderer* m_renderer;
    EmojiPicker* m_emojiPicker = nullptr;
    EditBar* m_editBar = nullptr;
    int m_editingMsgId = -1;
    int m_currentChatId = -1;
    int m_replyToId;
    QTimer* m_userMenuTimer = nullptr;
    QLabel* m_arrowLabel = nullptr;
};

```

Файл «IMainView.h»

```
#pragma once

#include <QString>
#include <QList>
#include <QPixmap>
#include "models/ChatDisplayData.h"
#include "types.h"

class QListWidgetItem;

class IMainView {
public:
    virtual ~IMainView() = default;

    virtual void setMyAvatar(const QPixmap& avatar) = 0;
    // === Список чатов ===
    virtual void addChatItem(const ChatDisplayData& data) = 0;
    virtual void updateChatItem(int chatId, const ChatDisplayData& data) = 0;
    virtual void removeChatItem(int chatId) = 0;
    virtual void setChatUnreadCount(int chatId, int count) = 0;

    // === Список сообщений ===
    virtual void setMessageList(const QList<QListWidgetItem*>& items) = 0;
    virtual void appendMessageItem(QListWidgetItem* item) = 0;
    virtual void updateMessageItem(int index, QListWidgetItem* item) = 0;
    virtual void removeMessageItem(int index) = 0;
    virtual void clearMessageList() = 0;
    virtual void scrollToBottom() = 0;

    // === Заголовок и строка состояния ===
    virtual void setChatHeader(const QString& title,
                               const QString& subtitle,
                               const QPixmap& avatar) = 0;
    virtual void setConnectionStatus(bool online,
                                       const QString& message) = 0;
    virtual void setTypingIndicator(const QString& text) = 0;

    // === Поиск и статус-бар ===
    virtual void showSearchResults(const QList<SearchResult>& results) = 0;
    virtual void hideSearch() = 0;
    virtual void showStatusMessage(const QString& message,
                                    int timeoutMs = 0) = 0;

    // === Диалоги профиля ===
    virtual void showUserProfile(int userId) = 0;
    virtual void showGroupProfile(int chatId) = 0;

    // === Текущий выбранный чат ===
    virtual int currentChatId() const = 0;
    virtual void activateChat(int chatId) = 0;
    virtual void repaintChatList() = 0;
    virtual void updateMessagePreview(const QString& fileId, const QPixmap& preview)
= 0;
    virtual int messageListWidth() const = 0;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		87

Файл «MainController.h»

```
#pragma once
#include <QObject>
#include "models/ChatModel.h"
#include "models/MessageModel.h"
#include "models/UserModel.h"
#include "services/notificationmanager.h"
#include "services/avatarservice.h"
#include "models/MessageViewBuilder.h"
#include "views/IMainView.h"
#include "controllers/ChatSessionController.h"
#include "controllers/FileTransferController.h"
#include "controllers/AvatarController.h"
#include "services/VoiceRecordingService.h"
#include "services/AvatarService.h"

class NetworkClient;
class FileTransferManager;
class TypingIndicator;
class ReadReceiptManager;
class NotificationManager;

class MainController : public QObject {
    Q_OBJECT
public:
    explicit MainController(NetworkClient* net,
        int currentUserId,
        const QString& currentUsername,
        IMainView* view,
        QObject* parent = nullptr);
    ~MainController();

    // модели
    ChatModel* chatModel() const { return m_chatModel; }
    MessageModel* messageModel() const { return m_messageModel; }
    UserModel* userModel() const { return m_userModel; }
    NetworkClient* networkClient() const { return m_net; }
    NotificationManager* notificationManager() const { return m_notifications; }

    int currentUserId() const { return m_currentUserId; }
    QString currentUsername() const { return m_currentUsername; }
    int currentChatId() const { return m_chatSession->currentChatId(); }

    void requestChatsAndUsers();
    void searchMessages(int chatId, const QString& query);
    // Действия пользователя
    void sendTextMessage(int chatId, const QString& text, int replyToId = -1);
    void sendFileMessage(int chatId, const QString& filePath, bool isVoice = false,
int durationMs = 0);
    void blockUser(int userId);
    void unblockUser(int userId);
    void openChat(int chatId);
    void closeChat(int chatId);
    void markChatRead(int chatId);
    void deleteMessage(int chatId, int msgId, bool forAll);
    void editMessage(int chatId, int msgId, const QString& newText);
    void addReaction(int chatId, int msgId, const QString& emoji);
    void removeReaction(int chatId, int msgId, const QString& emoji);
    void createGroupChat(const QString& name, const QList<int>& participantIds);
    void createPrivateChat(int otherUserId);
```

					ПГУ 1.090301.25.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

```

void addGroupMember(int chatId, int userId);
void removeGroupMember(int chatId, int userId);
void leaveGroup(int chatId);
void updateTyping(int chatId, const QString& currentText);
void downloadFile(const QString& fileId, const QString& fileName, int chatId,
const QString& savePath);
void startVoiceRecording();
void stopVoiceRecording();
void logout();
void checkAvatar(int userId, bool isGroup, const QString& newAvatarId);
void updateUserAvatar(int userId, const QString& newAvatarId);
void changeGroupAvatar(int chatId, const QString& path);
void renameGroup(int chatId, const QString& newName);

private slots:

void onAvatarUploaded(int entityId, bool isGroup, const QString& newAvatarId);
void onUserAvatarChanged(int userId, const QString& newAvatarId);
void onGroupAvatarChanged(int chatId, const QString& newAvatarId);
void onAvatarReady(const QString& avatarId, int entityId, bool isGroup);
void onChatAdded(int chatId);
void onChatChanged(int chatId);
void onChatRemoved(int chatId);
void onUnreadChanged(int chatId, int newCount);
void onConnected();
void onDisconnected();
void onReconnecting(int attempt);
void onConnectionStatusChanged(bool connected, const QString& msg);
void onUsersListReceived(const QList<User>& users);
void onChatListReceived(QList<Chat>& chats);
void onChatCreated(const Chat& chat);
void onGroupRenamed(int chatId, const QString& newName);
void onUserStatusChanged(int userId, bool isOnline, qint64 lastSeen);
void onLeftGroup(int chatId);
void onMemberAdded(int chatId, int userId, const QString& username);
void onMemberRemoved(int chatId, int userId, bool kicked);
void onSearchMessagesResult(int chatId, const QList<Message>& results);
private:
ChatDisplayData buildChatDisplayData(const Chat& c) const;
void loadInitialData();
void updateUnreadBadge();
QString m_myAvatarId;

NetworkClient* m_net;
int m_currentUserId;
QString m_currentUsername;
IMainView* m_view;

ChatModel* m_chatModel;
MessageModel* m_messageModel;
UserModel* m_userModel;

ChatSessionController* m_chatSession;
FileTransferController* m_fileCtrl;
AvatarController* m_avatarCtrl;
NotificationManager* m_notifications;

AvatarService* m_avatarService;
FileTransferManager* m_fileMgr;
TypingIndicator* m_typing;
ReadReceiptManager* m_receipts;
VoiceRecordingService* m_voiceService;
};

```

Файл «NetworkClient.h»

```
#pragma once
#include <QSslSocket>
#include <QSslConfiguration>
#include <QJsonObject>
#include <QTimer>
#include "protocol.h"
#include "services/FileTransferManager.h"
#include "serialization.h"
#include <QFile>

static constexpr int MAX_RECONNECT_INTERVAL_MS = 30000;
static constexpr int BASE_RECONNECT_MS = 2000;

class FileTransferManager;
class NetworkClient : public QObject
{
    Q_OBJECT
public:
    explicit NetworkClient(QObject* parent = nullptr);

    bool waitForConnected(int msTimeout = 5000);
    void connectToServer(const QString& host = "127.0.0.1",
        quint16 port = 12345);
    void disconnectFromHost();
    void setFileTransferManager(FileTransferManager* mgr) { m_fileMgr = mgr; }

    // Сессия
    void      setSessionToken(const QString& token);
    QString   sessionToken() const { return m_sessionToken; }
    void      trySessionLogin();
    void      logout();

    void blockUser(int userId);
    void unblockUser(int userId);

    // Удаление сообщения
    void      deleteMessage(int messageId, int chatId, bool forAll);

    void sendRequest(MessageType type, const QJsonObject& payload = {});
    void requestMessageHistory(int chatId, qint64 since = 0);
    void requestUsersList(const QString& searchQuery);
    void requestUsersList();
    void requestChatList();
    bool isConnected() const;
    void setCredentials(const QString& username, const QString& password);

    int findPrivateChatId(int otherUserId) const;

    void sendTypingStart(int chatId);
    void sendTypingStop(int chatId);

    // receipts
    void sendReadReceipt(int chatId, int lastMessageId);

    // Редактирование
    void editMessage(int messageId, int chatId,
        const QString& newText);

    // Реакции
    void addReaction(int messageId, int chatId,
```

					ПГУ 1.090301.25.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

```

        const QString& emoji);
void removeReaction(int messageId, int chatId,
    const QString& emoji);

// Управление группой
void addGroupMember(int chatId, int userId);
void removeGroupMember(int chatId, int userId);
void leaveGroup(int chatId);
void renameGroup(int chatId, const QString& newName);
void changeGroupAvatar(int chatId, const QString& avatarId);

void downloadAvatar(const QString& avatarId);
void uploadAvatar(int entityId, bool isGroup, const QByteArray& data);

// Поиск по сообщениям
void searchMessages(int chatId, const QString& query);

signals:

    void typingReceived(const TypingStatus& status);

    // receipts
    void readReceiptReceived(const ReadReceipt& receipt);

    // Редактирование
    void messageEdited(int messageId, Message msg);

    // Реакции
    void reactionAdded(int messageId, int chatId,
        const Reaction& reaction);
    void reactionRemoved(int messageId, int chatId,
        const Reaction& reaction);

    // Группы
    void memberAdded(int chatId, int userId,
        const QString& username);
    void memberRemoved(int chatId, int userId,
        bool wasKicked);
    void leftGroup(int chatId);
    void groupRenamed(int chatId, const QString& newName);
    void groupAvatarChanged(int chatId, const QString& avatarId);

    void avatarUploaded(int entityId, bool isGroup,
        const QString& avatarId);
    void avatarUploadFailed(const QString& error);
    void avatarDownloaded(const QString& avatarId, const QByteArray& data);
    void userAvatarChanged(int userId, const QString& avatarId);

    // Статус пользователя
    void userStatusChanged(int userId,
        bool isOnline,
        qint64 lastSeen);

    // Поиск
    void searchMessagesResult(int chatId,
        const QList<Message>& results);

    // Сессия / выход
    void sessionLoginSuccess(int userId,
        const QString& username,
        const QString& token);
    void sessionLoginFailed(const QString& error);

```

```

void logoutFinished();

// Удаление
void messageDeletedLocally(int messageId, int chatId);
void messageDeletedForAll(int messageId, int chatId);

void fileReceived(Message& msg);
void connectionStatusChanged(bool isConnected, const QString& message);
void registerSuccess();
void registerFailed(const QString& error);
void usersListReceived(const QList<User>& users);
void chatListReceived(QList<Chat>& chats);
void chatCreated(const Chat& chat);
void messageHistoryReceived(int chatId, QList<Message>& messages);
void loginSuccess(int userId, const QString& username);
void loginFailed(const QString& error);
void connected();
void disconnected();
void messageReceived(const Message& msg);
void currentUserInfoReceived(int userId, const QString& username);
void reconnecting(int attemptNumber);

private slots:
    void onReadyRead();
    void onConnected();
    void onDisconnected();
    void onReconnectTimer();

private:
    void setupSsl();
    void processPacket(const QByteArray& data);
    void scheduleReconnect();

    QSslSocket* socket;
    QByteArray m_buffer;
    FileTransferManager* m_fileMgr = nullptr;

    QString m_username;
    QString m_password;
    QString m_sessionToken;

    QString m_host;
    quint16 m_port = 12345;
    QTimer* m_reconnectTimer;
    bool m_initialConnection = true;
    int m_reconnectAttempt = 0;
    bool m_intentionalDisconnect = false;

    QMap<int, int> m_privateChatCache; // otherUserId -> chatId

    static constexpr qint64 CHUNK_SIZE = 64 * 1024;
};

```

Файл «ClientDbManager.h»

```
#pragma once
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>
#include <QList>
#include "types.h"

class ClientDbManager
{
public:
    static ClientDbManager& instance();

    bool init(const QString& dbPath);

    void saveUser(const User& user);
    void saveChat(const Chat& chat, const int currentUserId);
    QList<User> loadUsers();
    QList<Chat> loadChats(int currentUserId);

    void saveMessage(const Message& msg);
    void updateMessageId(const QString& tempId, int realId);
    void updateMessageStatus(const QString& tempId, const QString& status);
    void editMessage(int messageId, Message msg);
    void markDeletedForMe(int messageId, int myUserId);
    QList<Message> loadMessages(int chatId, int myUserId, int limit = 200);

    void updateMessageLocalPath(int messageId, const QString& localPath);

    void saveSession(const QString& token, int userId,
        const QString& username,
        const QString& host, quint16 port);
    bool loadSession(QString& outToken, int& outUserId,
        QString& outUsername,
        QString& outHost, quint16& outPort);
    void clearSession();

    bool chatExists(const int chatId, const int currentUserId) const;
    void updateUserAvatarId(int userId, const QString& avatarId);

    void confirmPendingMessage(const QString& oldTempId, int realId);

    bool isDeletedForMe(int messageId, int myUserId) const;

private:
    ClientDbManager() = default;
    void createTables();
    QSqlDatabase m_db;
};
```

Файл «ChatSessionController.h»

```
#pragma once
#include <QObject>
#include "models/ChatDisplayData.h"
#include "models/MessageViewBuilder.h"
#include "services/notificationmanager.h"
#include "views/IMainView.h"
#include "types.h"

class NetworkClient;
class ChatModel;
class MessageModel;
class UserModel;
class TypingIndicator;
class ReadReceiptManager;
class FileTransferManager;
class NotificationManager;
class IMainView;

class ChatSessionController : public QObject{
    Q_OBJECT
public:
    explicit ChatSessionController(NetworkClient* net,
        int currentUserId,
        const QString& currentUsername,
        IMainView* view,
        ChatModel* chatModel,
        MessageModel* messageModel,
        UserModel* userModel,
        TypingIndicator* typing,
        ReadReceiptManager* receipts,
        NotificationManager* notifications,
        QObject* parent = nullptr);

    void openChat(int chatId);
    void closeChat();
    int currentChatId() const { return m_currentChatId; }

    void sendTextMessage(int chatId, const QString& text, int replyToId = -1);
    void updateTyping(int chatId, const QString& currentText);
    void markChatRead(int chatId);
    void deleteMessage(int chatId, int msgId, bool forAll);
    void editMessage(int chatId, int msgId, const QString& newText);
    void addReaction(int chatId, int msgId, const QString& emoji);
    void removeReaction(int chatId, int msgId, const QString& emoji);

    void updateChatHeader(int chatId);

public slots:
    void onFilePlaceholderCreated(const Message& placeholder);

private slots:
    void onMessageReceived(const Message& msg);
    void onFileReceived(Message& msg);
    void onTypingReceived(const TypingStatus& status);
    void onReadReceiptReceived(const ReadReceipt& receipt);

    void onMessagesReset(int chatId);
    void onMessageAppended(int chatId, int index);
    void onMessageChanged(int chatId, int index);
    void onMessageDeleted(int chatId, int index);
```

```

void onMessageHistoryReceived(int chatId, QList<Message>& messages);
void onMessageDeletedLocally(int msgId, int chatId);
void onMessageDeletedForAll(int msgId, int chatId);
void onMessageEdited(int msgId, const Message& newMsg);
void onReactionRemoved(int msgId, int chatId, const Reaction& reaction);
    void onReactionAdded(int msgId, int chatId, const Reaction& reaction);

private:
    void updateUnreadBadge();
    void loadChatHistory(int chatId);

    ChatDisplayData buildChatDisplayData(const Chat& c) const;

    NetworkClient* m_net;
    IMainView* m_view;
    ChatModel* m_chatModel;
    MessageModel* m_messageModel;
    UserModel* m_userModel;
    TypingIndicator* m_typing;
    NotificationManager* m_notifications;
    ReadReceiptManager* m_receipts;
    MessageViewBuilder m_msgBuilder;
    int m_currentChatId = -1;
    int m_currentUserId;
    QString m_currentUsername;
};

```

					ПГУ 1.090301.25.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «ChatModel.h»

```
#pragma once
#include <QObject>
#include <QMap>
#include <QList>
#include "types.h"

class ChatModel : public QObject {
    Q_OBJECT
public:
    explicit ChatModel(QObject* parent = nullptr);

    void addOrUpdateChat(const Chat& chat);
    void removeChat(int chatId);
    void updateLastMessage(int chatId, const Message& msg);
    Chat chatById(int id) const;
    bool hasChat(int id) const;
    QList<Chat> allChats() const;

    int unreadCount(int chatId) const;
    void incrementUnread(int chatId);
    void resetUnread(int chatId);

    QString buildSubtitle(const Chat& chat, int currentUserId) const;
    QString buildHeaderSubtitle(const Chat& chat, int currentUserId) const;

signals:
    void chatAdded(int chatId);
    void chatChanged(int chatId);
    void chatRemoved(int chatId);
    void unreadChanged(int chatId, int newCount);

private:
    QMap<int, Chat> m_chats;
    QMap<int, int> m_unread;
};
```

Файл «MessageModel.h»

```
#pragma once
#include <QObject>
#include <QMap>
#include <QList>
#include <QSharedPointer>
#include "types.h"

class MessageModel : public QObject {
    Q_OBJECT
public:
    explicit MessageModel(QObject* parent = nullptr);

    void setMessages(int chatId, const QList<QSharedPointer<Message>>& msgs);
    void appendMessage(int chatId, const QSharedPointer<Message>& msg);
    void updateMessage(int chatId, int msgId, const QSharedPointer<Message>& newMsg);
    void updateMessageStatus(int chatId, const QString& tempId, int realId, const
    QString& status);
    void deleteMessage(int chatId, int msgId);

    QList<QSharedPointer<Message>> messages(int chatId) const;
    QSharedPointer<Message> findMessage(int chatId, int msgId) const;
    QSharedPointer<Message> findMessageByTempId(int chatId, const QString& tempId)
const;
    bool confirmMessageSent(int chatId, const QString& tempId, int realId);
    void addReaction(int chatId, int messageId, const Reaction& reaction);
    void removeReaction(int chatId, int messageId, int userId, const QString& emoji);
    void updateMessageLocalPreview(int chatId, int messageId, const QString&
    localPath);
    bool updateMessageByFileId(int chatId, const QString& fileId, Message& newMsg);

signals:
    void messagesReset(int chatId);
    void messageAppended(int chatId, int index);
    void messageChanged(int chatId, int index);
    void messageDeleted(int chatId, int index);
    void messageStatusChanged(int chatId, int index, const QString& newStatus);

private:
    QMap<int, QList<QSharedPointer<Message>>> m_messages;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		97

Файл «FileTransferManager.h»

```
#pragma once
#include <QObject>
#include <QMap>
#include "filetransferprotocol.h"
#include "types.h"

class NetworkClient;

class FileTransferManager : public QObject
{
    Q_OBJECT
public:
    explicit FileTransferManager(NetworkClient* net, QObject* parent = nullptr);

    QString startUpload(int chatId, const QString& filePath,
        int senderId, const QString& senderName,
        bool isVoice = false, int durationMs = 0);

    void startDownload(const QString& fileId, const QString& fileName, int chatId,
        const QString& savePath);

    void handleFileChunkAck(const QJsonObject& payload);
    void handleFileChunk(const QJsonObject& payload);
    void handleFileMessageResponse(const QJsonObject& root);

signals:
    void uploadProgress(const QString& fileId, int percent);
    void uploadFailed(const QString& fileId, const QString& error);
    void uploadFinished(const FileContext& file);

    void downloadProgress(const QString& fileId, const QString& fileName, int
        percent);
    void downloadFinished(const FileContext& file);
    void downloadFailed(const QString& fileId, const QString& error);

    void placeholderCreated(const Message& placeholder);
    void placeholderReplaced(const Message& finalMessage);

private:
    void sendNextChunk(const QString& fileId);
    Message buildPlaceholderMessage(const FileContext& ctx,
        int senderId, const QString& senderName,
        bool isVoice = false, int durationMs = 0);

    NetworkClient* m_net;
    QMap<QString, FileContext> m_transfers;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
						98
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «FileTransferController.h»

```
#pragma once
#include <QObject>
#include "views/IMainView.h"
#include "services/FileTransferManager.h"

class FileTransferManager;
class IMainView;

class FileTransferController : public QObject{
    Q_OBJECT
public:
    FileTransferController(FileTransferManager* fileMgr, IMainView* view, int
currentUserId, QObject* parent = nullptr);

    void sendFile(int chatId, const QString& filePath, const QString& senderName,
        bool isVoice = false, int durationMs = 0);
    void downloadFile(const QString& fileId, const QString& fileName, int chatId,
const QString& savePath);

signals:
    void das();

private slots:
    void onUploadProgress(const QString& fileId, int percent);
    void onDownloadProgress(const QString& fileId, const QString& fileName, int
percent);
    void onError(const QString& fileId, const QString& error);

private:
    FileTransferManager* m_fileMgr;
    IMainView* m_view;
    int m_currentUserId;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
						99
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «AvatarController.h»

```
#pragma once
#include <QObject>
#include <QPixmap>
#include "data/AvatarCache.h"

class AvatarService;
class IMainView;

class AvatarController : public QObject
{
    Q_OBJECT
public:
    explicit AvatarController(AvatarService* avService,
                             IMainView* view,
                             int currentUserId,
                             QObject* parent = nullptr);

    void checkAvatar(int entityId, bool isGroup, const QString& avatarId);

private slots:
    void onAvatarReady(const QString& avatarId, int entityId, bool isGroup);

private:
    AvatarService* m_avatarService;
    IMainView* m_view;
    int m_currentUserId;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
						100
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «VoiceRecordingService.h»

```
#pragma once
#include <QObject>
#include <QString>

class AudioCapturer;

class VoiceRecordingService : public QObject {
    Q_OBJECT
public:
    static constexpr int MIN_DURATION_MS = 500;

    explicit VoiceRecordingService(QObject* parent = nullptr);

    void start();
    void stop();
    bool isRecording() const;

signals:
    void recordingReady(const QString& filePath, int durationMs);
    void recordingTooShort();
    void recordingFailed(const QString& reason);

private slots:
    void onRawDataReady(const QByteArray& wavData, int durationMs);

private:
    QString saveTempFile(const QByteArray& data) const;

    AudioCapturer* m_capturer = nullptr;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		101

Файл «AudioCapturer.h»

```
#pragma once
#include <QObject>
#include <QByteArray>
#include <QTimer>

class AudioCapturer : public QObject
{
    Q_OBJECT
public:
    explicit AudioCapturer(QObject* parent = nullptr);
    ~AudioCapturer();

    void    startRecording();
    void    stopRecording();
    bool    isRecording() const { return m_recording; }
    int     durationMs() const;

signals:
    void    recordingStarted();
    void    durationChanged(int ms);
    void    recordingFinished(const QByteArray& data,
        int durationMs);
    void    error(const QString& msg);

private:
    QByteArray buildWavHeader(qint32 dataSize,
        int sampleRate = 16000,
        int channels = 1,
        int bitsPerSample = 16);
    bool        m_recording = false;
    QByteArray  m_buffer;
    QTimer*     m_durationTimer = nullptr;
    int         m_durationMs = 0;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		102

Файл «VoicePlayerManager.h»

```
#pragma once
#include <QObject>
#include <QMediaPlayer>
#include <QAudioOutput>
#include <QMap>

class VoicePlayerManager : public QObject
{
    Q_OBJECT
public:
    static VoicePlayerManager& instance();

    void playOrPause(const QString& fileId,
                    const QString& localPath);

    void stopAll();

    bool isPlaying(const QString& fileId) const;
    qint64 position(const QString& fileId) const;
    qint64 duration(const QString& fileId) const;

signals:
    void stateChanged(const QString& fileId);
    void positionChanged(const QString& fileId, qint64 pos);

private:
    explicit VoicePlayerManager(QObject* parent = nullptr);

    QString      m_currentFileId;
    QMediaPlayer* m_player = nullptr;
    QAudioOutput* m_audioOut = nullptr;
};
```

Файл «TypingIndicator.h»

```
#pragma once
#include <QObject>
#include <QMap>
#include <QTimer>
#include <QStringList>
#include "types.h"

class NetworkClient;

class TypingIndicator : public QObject
{
    Q_OBJECT
public:
    explicit TypingIndicator(NetworkClient* net,
        int myUserId,
        QObject* parent = nullptr);

    void onLocalTextChanged(int chatId, const QString& text);

    void onChatSwitched(int newChatId);

    void onRemoteTyping(const TypingStatus& status);

    QStringList whoIsTyping(int chatId) const;

    QString typingText(int chatId) const;

signals:
    void typingChanged(int chatId);

private slots:
    void onLocalStopTimer();

private:
    void startLocalTyping(int chatId);
    void stopLocalTyping(int chatId);
    void clearRemoteTyping(int chatId, int userId);

    NetworkClient* m_net;
    int m_myUserId;

    QTimer* m_localStopTimer;
    int m_localChatId = -1;
    bool m_localTyping = false;

    struct RemoteTyper {
        QString username;
        QTimer* ttlTimer = nullptr;
    };
    QMap<int, QMap<int, RemoteTyper>> m_remote;

    static constexpr int LOCAL_STOP_MS = 2000;
    static constexpr int REMOTE_TTL_MS = 5000;
};
```

Файл «ReadReceiptManager.h»

```
#pragma once
#include <QObject>
#include <QMap>
#include "types.h"

class NetworkClient;

class ReadReceiptManager : public QObject
{
    Q_OBJECT
public:
    explicit ReadReceiptManager(NetworkClient* net,
                                int myUserId,
                                QObject* parent = nullptr);

    void markAsRead(int chatId, int lastMessageId);
    void onReceiptReceived(const ReadReceipt& receipt);
    bool isReadBy(int userId, int messageId) const;

    QString messageStatus(int messageId,
                           int chatId,
                           int senderId) const;
    int lastReadId(int chatId, int userId) const;

signals:
    void receiptUpdated(int chatId, int userId, int lastReadId);

private:
    NetworkClient* m_net;
    int m_myUserId;

    QMap<int, QMap<int, int>> m_receipts;
    QMap<int, int> m_lastSent;
};
```

Файл «SearchController.h»

```
#pragma once
#include <QObject>
#include <QTimer>
#include <QList>
#include "types.h"

class MainController;

class SearchController : public QObject {
    Q_OBJECT
public:
    explicit SearchController(MainController* controller, QObject* parent = nullptr);
    void search(const QString& query);
    void cancel();

signals:
    void resultsReady(const QList<SearchResult>& results);
    void cleared();

private slots:
    void onDebounceTimeout();
    void onServerUsersReceived(const QList<User>& users);

private:
    QList<SearchResult> buildResults(const QString& query) const;

    MainController* m_controller;
    QTimer* m_timer;
    QString m_pendingQuery;
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		106

Файл «MessengerServer.h»

```
#pragma once
#include <QSslServer>
#include <QSslSocket>
#include <QSslConfiguration>
#include <QSslKey>
#include <QSslCertificate>
#include <QMap>
#include <QSet>
#include <QTimer>
#include <QJsonObject>
#include "protocol.h"
#include "filemanager.h"
#include "dbmanager.h"
#include "serialization.h"

struct ClientState {
    QByteArray buffer;
    int      userId = -1;
    QTimer* pingTimer = nullptr;
    bool      waitingPong = false;
};

class MessengerServer : public QSslServer
{
    Q_OBJECT
public:
    explicit MessengerServer(QObject* parent = nullptr);
    bool start(quint16 port = 12345);

protected:
    void incomingConnection(qintptr socketDescriptor) override
    {
        QSslSocket* socket = new QSslSocket(this);

        if (!socket->setSocketDescriptor(socketDescriptor)) {
            delete socket;
            return;
        }

        socket->setPrivateKey(privateKey);
        socket->setLocalCertificate(certificate);

        socket->startServerEncryption();

        addPendingConnection(socket);
    }

private slots:
    void onNewConnection();
    void onReadyRead();
    void onClientDisconnected();

private:
    bool initSsl();

    void processPacket(QSslSocket* client, const QByteArray& data);

    bool handleUnauthorized(QSslSocket* client,
        MessageType      type,
        const QJsonObject& payload,
        ClientState& state);
};
```

					ПГУ 1.090301.25.001 ПЗ	Лист
						107
Изм.	Лист	№ докум.	Подпись	Дата		

```

void handleRegister(QSslSocket* client,
    const QJsonObject& payload);

void handleLogin(QSslSocket* client,
    const QJsonObject& payload,
    ClientState& state);

void handleSessionLogin(QSslSocket* client,
    const QJsonObject& payload,
    ClientState& state);

void handleUsersList(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleChatList(QSslSocket* client, int userId);

void handleCreateChat(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleTextMessage(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleMessageHistory(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleDeleteMessage(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleEditMessage(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleReaction(QSslSocket* client,
    const QJsonObject& payload,
    int userId, bool isAdd);

void handleReadReceipt(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleSearchMessages(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleAddMember(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleRemoveMember(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleLeaveGroup(QSslSocket* client,
    const QJsonObject& payload,
    int userId);

void handleFileMessage(QSslSocket* client,

```

```

        const QJsonObject& payload,
        int senderId);

void handleFileDownloadRequest(QSslSocket* client, const QJsonObject& payload,
int userId);

void handleFileChunk(QSslSocket* client,
        const QJsonObject& payload,
        int senderId);

void handleTyping(QSslSocket* client,
        const QJsonObject& payload,
        int            userId,
        bool            isTyping);

void handleLogout(QSslSocket* client,
        const QJsonObject& payload,
        int userId);

void handlePing(QSslSocket* client, ClientState& state);

void handleUploadAvatar(QSslSocket* client, const QJsonObject& payload, int
userId);

void handleDownloadAvatar(QSslSocket* client, const QJsonObject& payload, int
userId);

void handleRenameGroup(QSslSocket* client, const QJsonObject& payload, int
userId);

void handleBlockUser(QSslSocket* client, const QJsonObject& payload, int userId);
void handleUnblockUser(QSslSocket* client, const QJsonObject& payload, int
userId);

void            sendToClient(QSslSocket* client,
        MessageType type,
        const QJsonObject& payload);

void            sendError(QSslSocket* client,
        const QString& message);

bool            checkMembership(QSslSocket* client,
        int chatId, int userId);

void broadcastToChat(int chatId, MessageType type, const QJsonObject& payload,
QSslSocket* exclude = nullptr);

void broadcastToGroupMembers(int chatId, int excludeUserId, MessageType type,
const QJsonObject& payload);

void broadcastToContacts(int userId, MessageType type, const QJsonObject&
payload);

void            broadcastUserStatus(int userId, bool isOnline);

QList<int>       getChatMemberIds(int chatId);
QSslSocket*     findSocketByUserId(int userId);
QString         getChatDisplayName(int chatId, int forUserId);

QJsonObject     buildMemberJson(const User& u);
QJsonArray      buildMembersArray(int chatId);

void            onPingTimer(QSslSocket* client);

```

```

QSSLConfiguration sslConfig;
QSSLCertificate certificate;
QSSLKey privateKey;
QMap<QSSLSocket*, ClientState> clientStates;
QMap<int, QString> userNames;
};

```

					ПГУ 1.090301.25.001 ПЗ	Лист
						110
Изм.	Лист	№ докум.	Подпись	Дата		

Файл «DBManager.h»

```
#pragma once
#include <QSqlDatabase>
#include <QSqlQuery>
#include <QSqlError>
#include <QMap>
#include "types.h"

class DBManager
{
public:
    static DBManager& instance();
    bool init(const QString& dbPath = "messenger.db");

    // Пользователи
    bool registerUser(const QString& username,
                     const QString& password,
                     const QString& email);

    int loginUser(const QString& username,
                 const QString& password);

    QString getUsername(int userId);
    QList<User> getAllUsers();
    QList<int> getChatAdmins(int chatId);
    void updateLastSeen(int userId);
    qint64 getLastSeen(int userId);

    bool saveAvatar(int userId, const QString& avatarId);
    QString getAvatar(int userId);

    // Сессии
    bool createSession(int userId, const QString& token);
    int validateSession(const QString& token);
    bool renewSession(const QString& token);
    bool deleteSession(const QString& token, int userId);

    // Чаты
    void deleteChat(int chatId);
    bool isGroup(int chatId) const;
    bool blockUser(int blockerId, int blockedId);
    bool unblockUser(int blockerId, int blockedId);
    bool isBlocked(int userId, int targetUserId);

    int createChat(int creatorId, bool isGroup,
                  const QString& name = "");
    int createPrivateChat(int user1Id, int user2Id);
    int findPrivateChat(int user1Id, int user2Id);

    bool addParticipant(int chatId, int userId, bool isAdmin);
    bool removeParticipant(int chatId, int userId);
    bool isMember(int chatId, int userId);

    bool setGroupAdmin(int chatId, int userId, bool isAdmin);
    bool isGroupAdmin(int chatId, int userId);

    void updateUserAvatar(int userId, const QString& avatarId);

    void updateGroupAvatar(int chatId, const QString& avatarId);

    void renameGroup(int chatId, const QString& newName);
```

```

QList<int> getGroupMembers(int chatId);

QList<int> getUserContacts(int userId);
bool    updateGroupInfo(int chatId,
    const QString& name,
    const QString& description);
QList<Chat> getUserChats(int userId);
QList<int>  getUserChatIds(int userId);
QList<User> getChatMembers(int chatId);
// Сообщения
bool    saveMessage(const Message& msg);
int     saveMessageGetId(const Message& msg);
bool    saveFileMessage(const Message& msg);
QString getMessageText(int messageId) const;
QList<Message> getMessageHistory(int    chatId,
    int    limit = 100,
    qint64 sinceTimestamp = 0);

QList<Message> getMessagesBefore(int    chatId,
    qint64 beforeTimestamp,
    int    limit = 50);

QList<Message> getLastMessages(const QList<int>& chatIds);

QList<Message> searchMessages(int            chatId,
    const QString& query,
    int            limit = 50);

struct MessageMeta {
    int  senderId = -1;
    int  chatId = -1;
    bool found = false;
};
MessageMeta getMessageMeta(int messageId);

bool    editMessage(int            messageId,
    int            requesterId,
    const QString& newText);

bool    deleteMessageForAll(int messageId, int requesterId);
bool    deleteMessageForUser(int messageId, int userId);

// Реакции
bool    addReaction(int            messageId,
    int            userId,
    const QString& emoji);

bool    removeReaction(int            messageId,
    int            userId,
    const QString& emoji);

QList<Reaction> getReactions(int messageId);

// receipts
bool    saveReadReceipt(int chatId, int userId, int lastMessageId);
QMap<int, int> getReadReceipts(int chatId);

private:
    DBManager() = default;
    void createTables();
    QSqlDatabase db;
};

```

Приложение В

(обязательное)

Клиент-серверное приложение обмена сообщениями
на базе фреймворка Qt
Презентация

					ПГУ 1.090301.25.001 ПЗ	Лист
						113
Изм.	Лист	№ докум.	Подпись	Дата		

Клиент-серверное приложение обмена сообщениями на базе фреймворка Qt

Автор работы:

Студент гр. 22ВВП1 Черников Кирилл Александрович

Научный руководитель:

к.п.н., доцент каф. ВТ Леонова Татьяна Юрьевна

Пенза, 2026

Актуальность темы

Цифровой суверенитет

Потребность в приватных каналах связи, независимых от коммерческих мессенджеров

Корпоративная безопасность

Обмен данными через внешние сервисы не отвечает требованиям безопасности при удалённой работе

Контроль инфраструктуры

Собственный сервер обеспечивает полный контроль над хранением и маршрутизацией сообщений

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.25.001 ПЗ

Цель и задачи работы

Цель работы – Разработать прототип клиент-серверного приложения с функцией защищённого обмена сообщениями на базе фреймворка Qt.

Задачи:

- Анализ предметной области и выбор инструментов разработки;
- Проектирование архитектуры приложения и базы данных;
- Реализация клиентской и серверной частей;
- Тестирование и разработка руководства пользователя.

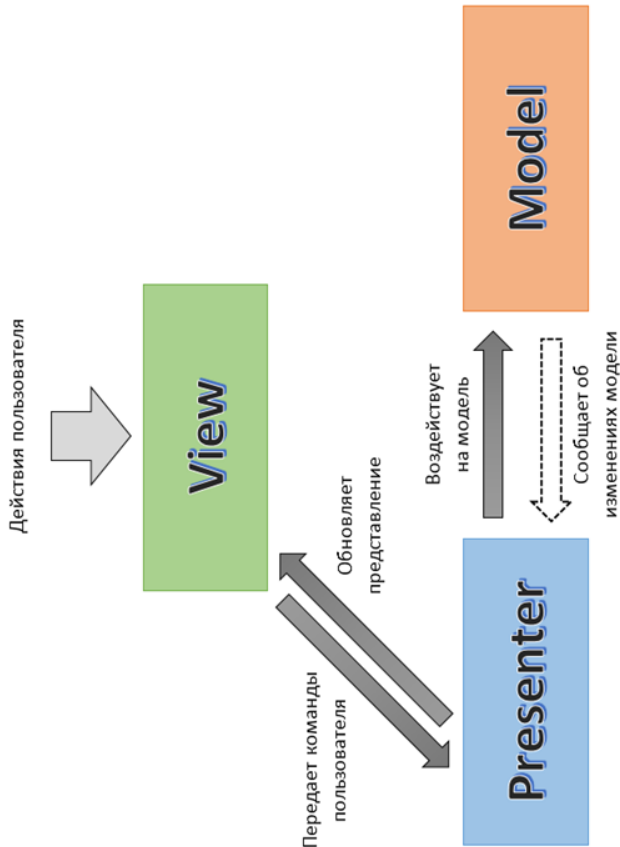
Сравнение аналогов

 <p>Telegram</p> <ul style="list-style-type: none"> + Быстрый, удобный интерфейс; + большие группы и каналы. - Зависимость от облачной инфраструктуры; - нет возможности создания собственного сервера. 	 <p>Discord</p> <ul style="list-style-type: none"> + Сообщества; + голосовые/видеоچаты; + гибкие роли; + каналы. - Высокое потребление ресурсов; - централизованная архитектура. 	 <p>teamspeak</p> <ul style="list-style-type: none"> + Возможность создания собственного сервера; + настройка качества связи; + работа в локальной сети. - Нет синхронизации сообщений; - ограниченный
---	--	---

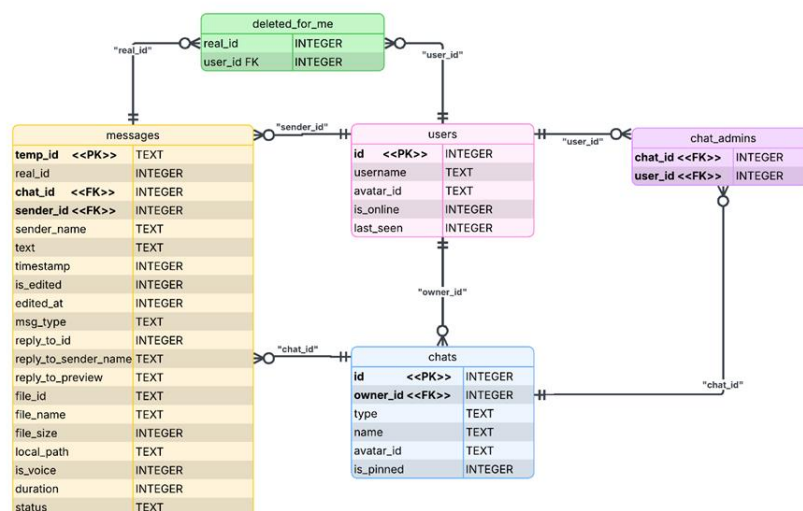
Используемые технологии



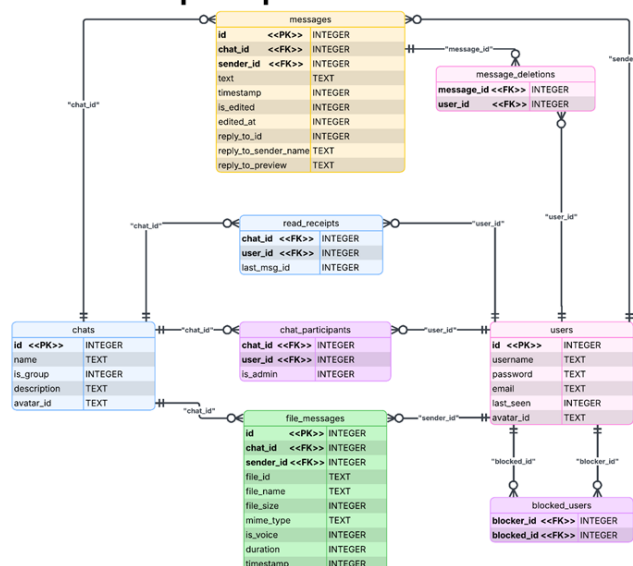
Архитектура приложения



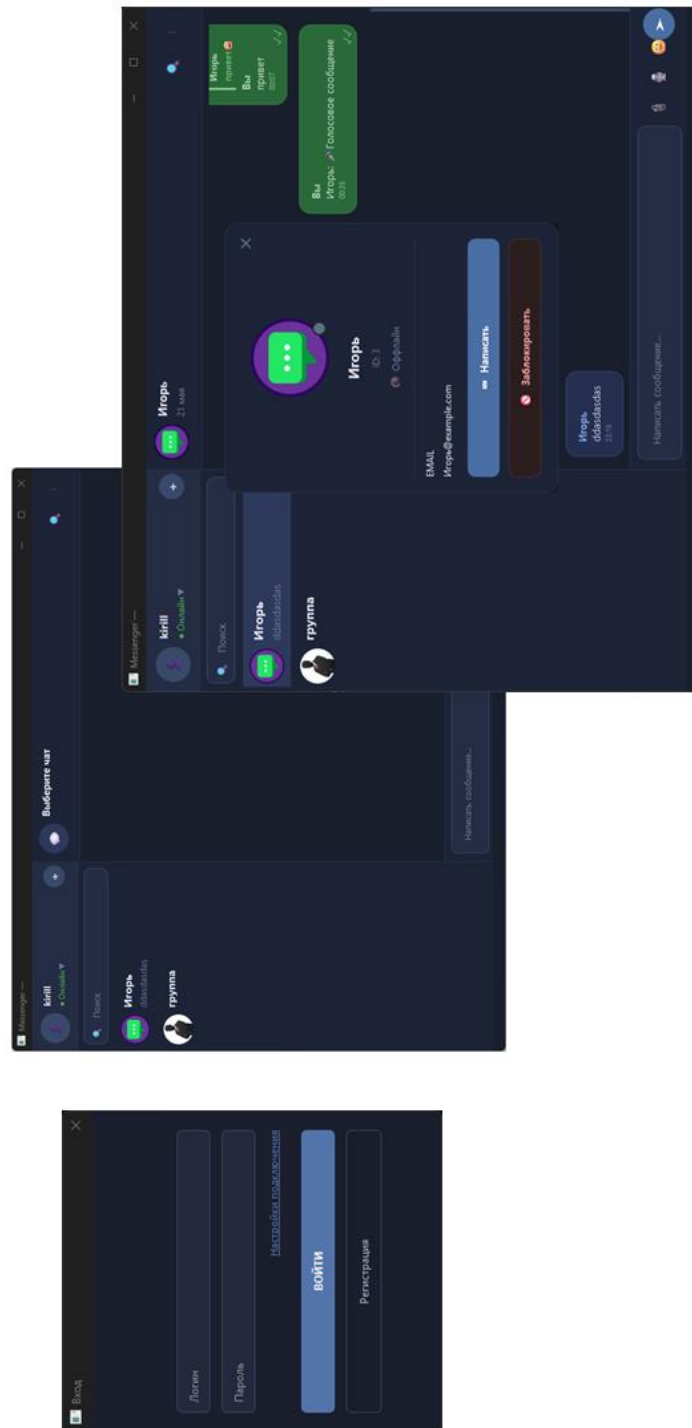
База данных клиента



База данных сервера



Клиентская часть



Серверная часть



Тестирование

Для проверки корректности работы было проведено функциональное тестирование, также были разработаны тест-кейсы для проверки обработчиков ошибок.

VerySleepy/
verysleepy

Very Sleepy, a sampling CPU profiler for Windows



Выполнено тестирование производительности с использованием профайлера VerySleepy, что позволило выявить и устранить проблемы с производительностью в коде.

Результаты

- Выполнен анализ предметной области;
- Рассмотрены аналоги;
- Разработано и отлажено серверное приложение;
- Разработано и отлажено клиентское приложение;
- Выполнено тестирование производительности.

В дальнейшем проект можно расширить за счёт внедрения сквозного шифрования, поддержки мобильных платформ и реализации аудио- и видеозвонков, синхронизации между устройствами и интеграции дополнительных механизмов защиты и администрирования.

Спасибо за внимание!

					ПГУ 1.090301.25.001 ПЗ	Лист
						125
Изм.	Лист	№ докум.	Подпись	Дата		