Министерство науки и высшего образования РФ
Федеральное государственное автономное образовательное учреждение высшего образования
«Казанский (Приволжский) Федеральный Университет»

ИНСТИТУТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ТЕОРЕТИЧЕСКОЙ КИБЕРНЕТИКИ
Направление: 01.03.02. Прикладная математика и информатика
Профиль: Прикладная математика и информатика

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Разработка десктопного клиент - серверного мессенджера на Cpp с функциями шифрования

Работа завершена:
Студент гр.09-812 ИВМиИТ
«__»_____20__г. _____ И. Н. Садыков

Работа допущена к защите:
Научный руководитель,
к.ф.-м. н, доцент КТК ИВМиИТ
«__»_____20__г. _____ В. Р. Байрашева

Заведующий кафедрой,
д.ф.м.н., профессор
«__»_____20__г. _____ Ф. М. Аблаев

Казань – 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность работы. Сейчас сложно представить современный мир без сети интернет. В нем мы делаем покупки, получаем образование, развлекаемся, общаемся. Свободный обмен информацией является одним из основных преимуществ глобальной паутины. Большую часть времени в интернете люди проводят в различных социальных сетях, мессенджерах. Мессенджер — это простая программа для обмена текстовыми сообщениями, видео и фотографиями. Мессенджеры должны иметь возможность своевременно принимать важную информацию, помогать пользователю следить за новостями и позволять группам людей проводить переговоры.

Основой таких приложений и всего интернета в целом является стек протоколов TCP/IP. Стандартная архитектура таких приложений является клиент-серверной. Архитектура клиент-сервер — это вычислительная модель, в которой сервер хранит и обрабатывает большую часть ресурсов и служб потребляемых клиентом. В архитектуре этого типа один или несколько клиентских компьютеров подключены к центральному серверу через сеть. Поверх транспортного протокола TCP для приложения часто пишут свой более высокоуровневый протокол обмена данными, который адаптирован под конкретную бизнес логику.

Современные тенденции требуют быстрой разработки программного обеспечения, чтобы в короткие сроки его можно было доставить пользователю или заказчику и получать прибыль. В связи с этим очень выгодно разрабатывать кроссплатформенное программное обеспечение. Развертывая приложение на разных платформах, под разные операционные системы, можно добиться увеличения количества пользователей, конверсии, а также сократить количество затраченных ресурсов на разработку, потому что приложение нужно разработать всего один раз. Для разработки таких приложений используют интерпретируемые языки, которые не завязаны на конкретную платформу и исполняют код на виртуальной машине. Но их производительность не так хороша, как на компилируемых языках. Наш

выбор при написании приложения пал на язык C++ с использованием фреймворка Qt. На языке C++ при грамотном использовании можно добиться потрясающей производительности, при этом используя высокоуровневые конструкции и ООП, но его стандартная библиотека довольно небольшая и простая, поэтому мы прибегли к использованию кроссплатформенного фреймворка, в котором есть все, что нужно для разработки сетевого приложения с графическим интерфейсом.

Цель работы – реализация прикладного программного обеспечения клиент-серверного кроссплатформенного мессенджера.

Задачи, решаемые для достижения цели:

- Изучение сетевых технологий и методов разработки сетевых приложений
- Проектирование архитектуры клиент-серверного приложения
- Определение модели данных и протокола, на которых будет основан клиент-сервер
- Непосредственно реализация приложения в программном коде
- Тестирование

Анализ существующих кроссплатформенных мессенджеров

Среди кроссплатформенных мессенджеров наиболее популярными в мире являются: WhatsApp, WeChat, Telegram. Рассмотрим десктопные версии данных приложений.

Рисунок 1 – основное окно приложения Telegram

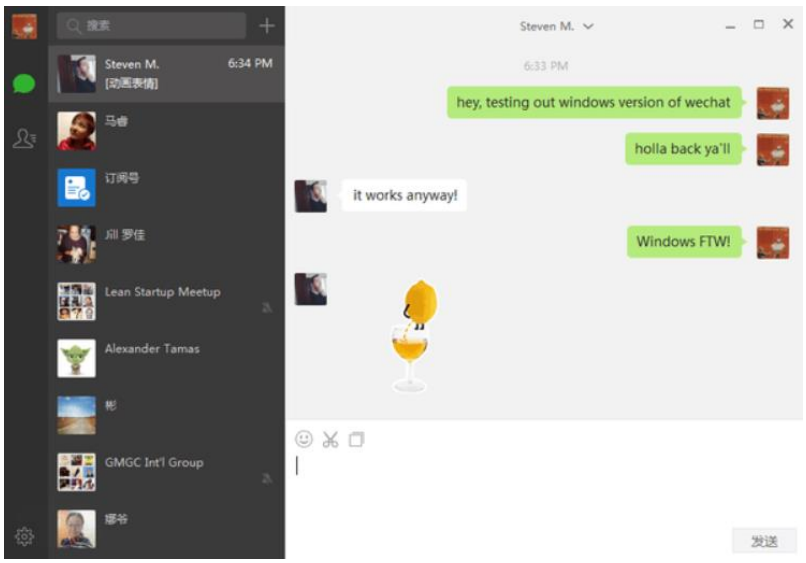

Рисунок 2 – основное окно приложения WeChat

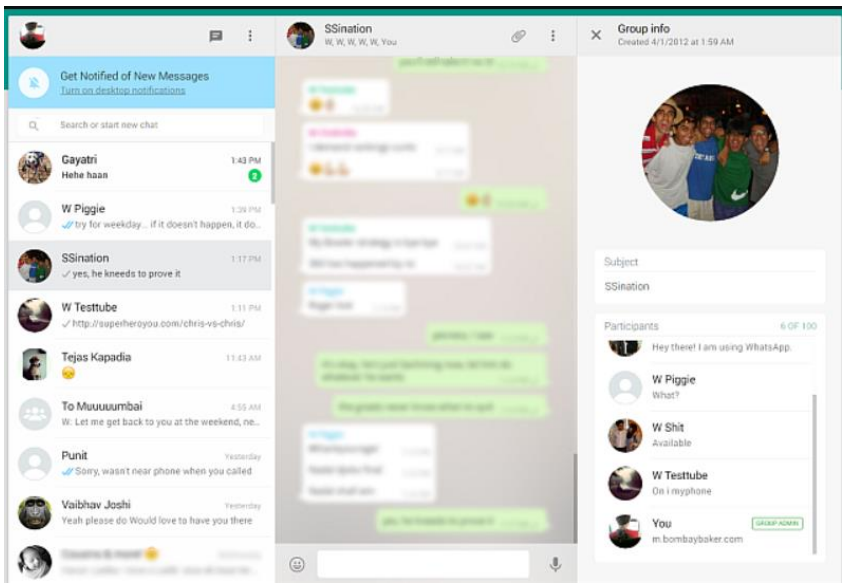

Рисунок 3 – основное окно приложения WhatsApp

Все эти приложения работают на операционных системах Windows, macOS и Unix-подобных операционных системах. Также они поддерживают x86, x64 аппаратные платформы.

В основном приложения имеют похожие элементы и возможности (см. рисунки 1, 2, 3): большое пространство для чата, возможность выбирать группы для общения, возможность посылать сообщения, фотографии, видео. Обычно сообщения, отправляемые нами, находятся справа, а сообщения вашего собеседника слева. Можно видеть время отправки сообщения.

Визуально данные приложения похожи, но у каждого из них есть свои особенности. Telegram позиционируется как наиболее безопасный мессенджер. WeChat позволяет пересылать деньги прямо в чате, а также имеет интеллектуальную ленту новостей. WhatsApp очень простой в управлении, но данный мессенджер уже отстаёт от конкурентов по введению новых функциональных особенностей.

Популярными мессенджерами владеют и управляют крупные компании и государства, что может ставить под сомнение приватность данных. Помимо этого всегда есть риск того, что мессенджер может стать доступным только по платной подписке или вовсе может быть заблокирован, что создает риски для вашего бизнеса. Собственная реализация мессенджера позволяет быть

уверенным в безопасности данных. Так компания, которая хочет иметь внутренний чат для общения, может развернуть свой собственный сервер, и для неё вероятность утечки данных будет меньше. Или если требуется какая-то новая функциональная возможность, которая специфична для компании, и она вряд ли будет введена в популярном мессенджере, то ничего не ограничивает возможность добавить эту особенность в свое приложение. Например, наше приложение позволяет создавать группы для общения, в которых сообщения не будут храниться на сервере и будут только на физической памяти у клиентов. Также есть возможность устанавливать пароль для группы и периодически его менять. В любой момент можно подключить необходимый сервис, который будет удовлетворять нуждам пользователя.

Глава I. Проектирование приложения

Основные понятия, используемые при разработке приложения

• Клиент – это программа, запущенная на устройстве пользователя. Клиентская программа обращается к серверу для исполнения различных задач и служб.

• Сервер – программа, которая чаще всего находится на удаленной машине, но может быть и на локальной, ждет запросов от клиентов, а затем отвечает на них.

• Параллелизм – это возможность одновременного исполнения вычислений. Чаще всего организуется с использованием множества процессов, потоков.

• Кроссплатформенная разработка приложения – это создание одного приложения, которое может работать в разных операционных системах, вместо разработки разных версий приложения для каждой платформы. Кроссплатформенность может быть достигнута с помощью интерпретаторов, которые переводят байтовый код в машинный код, а также фреймворков, которые в рамках высокоуровневых конструкций могут собирать код, подходящий для целевой платформы.

• Фреймворк – это абстракция, реализующая общие функции, каркас для будущего приложения, обычно набор функций классов, которые разработчик может расширять и изменять для достижения желаемого поведения.

Язык C++

При выборе основного инструмента для разработки мы остановились на языке C++. В проекте используется стандарт C++17.

C++ – это один из самых популярных языков программирования в мире, используется в современных операционных системах, графических пользовательских интерфейсах и встроенных системах [5]. C++ мультипарадигменный, компилируемым язык со статической типизацией. Чаще всего используют объектно-ориентированный подход, что дает четкую структуру программам и позволяет повторно использовать код, снижая затраты на разработку [5]. C++ является переносимым и может использоваться для разработки приложений, которые могут исполняться на различных платформах [5].

Фреймворк Qt

Qt – это кроссплатформенный фреймворк для разработки настольных, встроенных и мобильных приложений. Поддерживаемые платформы включают Linux, Windows, Android, iOS, BlackBerry и другие.

Сам Qt не является языком программирования. Это среда разработки, написанная на C++. Препроцессор, MOC (компилятор метаобъектов) используются для расширения языка C++ такими функциями, как сигналы и слоты. Перед этапом компиляции MOC анализирует исходные файлы, написанные на расширенном Qt C++, и генерирует из них соответствующие стандарту исходные коды C++. Таким образом, сам фреймворк и приложения/библиотеки, которые его используют, могут быть скомпилированы с помощью любого стандартного компилятора C++, такого как Clang, GCC, MinGW и MSVC [4].

Все классы в Qt наследуются от класса QObject (см. рисунок 4).

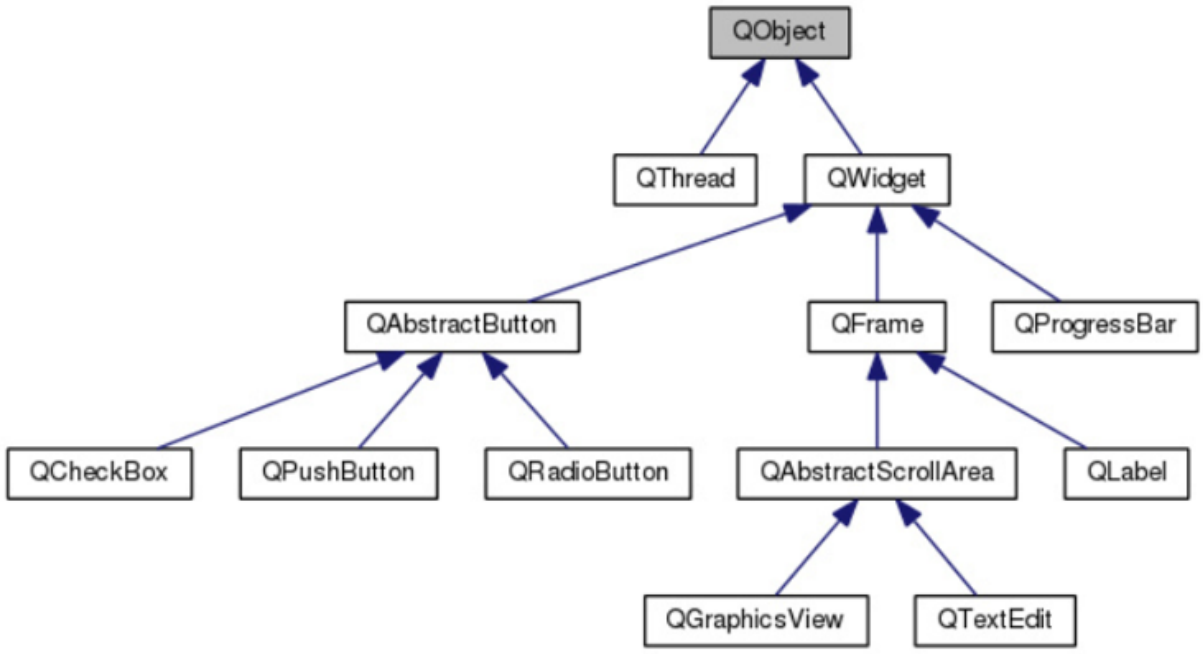

Рисунок 4 – Диаграмма наследования классов в Qt

Краткое описание основных модулей Qt приведено в таблице 1.

Таблица 1 – модули Qt

Модуль	Описание
Qt Core	Единственный необходимый модуль Qt, содержащий классы, используемые другими модулями, включая мета-объектную систему, параллелизм и многопоточность, контейнеры, систему событий, плагины и средства ввода-вывода.
Qt GUI	Центральный модуль графического интерфейса. В Qt 5 этот модуль зависит от OpenGL, но больше не содержит классов виджетов.
Qt Widgets	Содержит классы для классических приложений с графическим интерфейсом на основе виджетов и классы QSceneGraph.
Qt QML	Модуль для языков QML и JavaScript.
Qt Quick	Модуль для приложения с графическим интерфейсом, написанный с использованием QML2.
Qt Quick Controls	Виджет, подобный элементам управления Qt Quick, предназначен в основном для настольных приложений.
Qt Quick Layouts	Макеты для размещения элементов в Qt Quick.
Qt Network	Слой сетевой абстракции. В комплекте с поддержкой TCP, UDP, HTTP, TLS, SSL (в Qt 4) и SPDY (начиная с Qt 5.3).
Qt Multimedia	Классы по работе с аудио, видео, радио и камерой.
Qt Multimedia Widgets	Виджеты от Qt Multimedia.
Qt SQL	Содержит классы для интеграции базы данных с использованием SQL.

Хотя приложения, использующие Qt, обычно пишутся на C++, существуют привязки QML к другим языкам. Они не являются частью Qt, но

предоставляются различными третьими сторонами. Например, Riverbank Computing предоставляет коммерческие и бесплатные привязки Python для программного обеспечения (PyQt).

Основные модули Qt, используемые для разработки open-source программного обеспечения, являются бесплатными. Для открытого исходного кода обычно используется лицензия GPL, которая предоставляют следующие преимущества:

- Свобода запускать программу для любых целей.
- Свобода изучать, как работает программа, и адаптировать ее к конкретным потребностям.
- Свобода распространять копии.
- Свобода улучшать программу и публиковать свои улучшения для всего сообщества.

Сборка проекта с использованием Qt

Затронем тему сборки с использованием фреймворка Qt. Помимо исходных файлов в проекте помещается файл CMakeLists.txt. Он необходим для вызова утилиты CMake и последующего создания make файлов. Он хранит в себе заранее предусмотренные инструкции, в которых прописывается название проекта, иерархия модулей, утилиты необходимые для сборки проекта [2].

Сравнивая системы сборки C++ и Qt, можно увидеть, что система сборки C++ (серые прямоугольники) осталась неизменной (см. рисунок 5). Мы все еще пишем код на C++. Однако мы добавляем больше источников и заголовков (зеленые прямоугольники) (см. рисунок 5). Здесь задействованы три генератора кода:

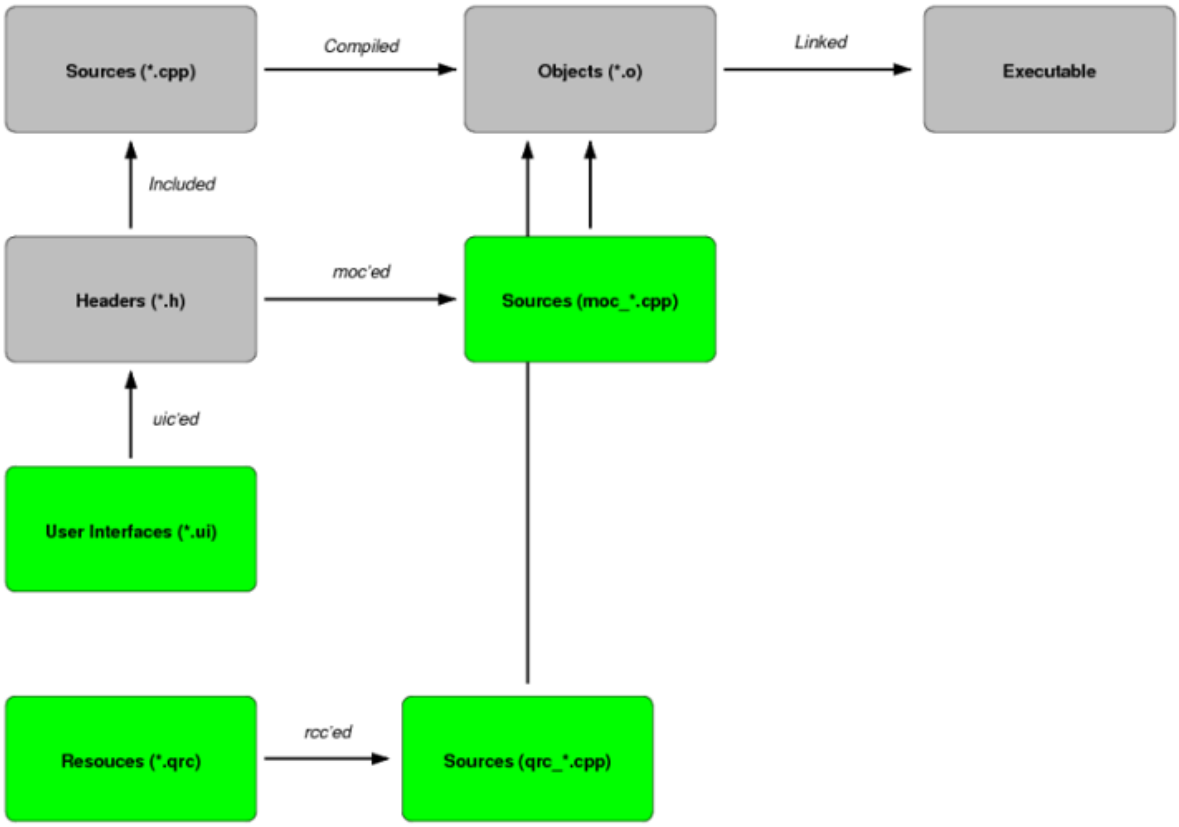

Рисунок 5 – Диаграмма этапов сборки в Qt

Метаобъектная компиляция

Компилятор метаобъектов (moc) принимает все классы, начиная с макроса Q_OBJECT, и генерирует исходный файл C++ moc_*.cpp. Этот файл содержит информацию о моделируемом классе, такую, как имя класса, дерево наследования и т. д., а также реализацию сигналов. Это означает, что, когда вы посылаете сигнал, вы фактически вызываете функцию, сгенерированную moc.

Компиляция пользовательского интерфейса

Компилятор пользовательского интерфейса (uic) берет проекты из Designer и создает файлы заголовков. Эти файлы заголовков затем, как обычно, включаются в исходные файлы, что позволяет вызвать setupUi для создания экземпляра дизайна пользовательского интерфейса.

Компиляция ресурсов

Компилятор ресурсов Qt (rcc) позволяет встраивать изображения, текстовые файлы и т.д. в исполняемый файл, но при этом иметь доступ к ним как к файлам.

Далее, после компиляции файлов в файлы *.h и *.cpp, относящихся к фреймворку Qt, берутся все файлы и производятся стандартные действия по сборке, свойственные для языка C++.

Препроцессинг

Препроцессор работает с одним исходным файлом C++ за раз, заменяя директивы #include содержимым соответствующих файлов (обычно это просто объявления), выполняя замену макросов (#define) и выбирая разные части текста в зависимости от #if, Директивы #ifdef и #ifndef.

Компиляция

Шаг компиляции выполняется на каждом выходе препроцессора. Компилятор анализирует чистый исходный код C++ (теперь без каких-либо директив препроцессора) и преобразует его в код сборки, создавая фактический двоичный файл в некотором формате (ELF, COFF, a.out, ...). Этот объектный файл содержит скомпилированный код (в двоичной форме)

символов, определенных во входных данных. Символы в объектных файлах называются по имени.

Созданные объектные файлы могут быть помещены в специальные архивы, называемые статическими библиотеками, для облегчения повторного использования в дальнейшем.

Компоновка

Компоновщик производит окончательный вывод компиляции из объектных файлов, созданных компилятором. Этот вывод может быть либо статической или динамической библиотекой, либо исполняемым файлом. Он связывает все объектные файлы, заменяя ссылки на неопределенные символы правильными адресами. Каждый из этих символов может быть определен в других объектных файлах или в библиотеках.

Механизм сигналов и слотов.

Сигналы и слоты используются для связи между объектами. Данный механизм – это основная особенность Qt, то есть именно та часть, которая больше всего отличается от функционала, предоставляемого другими фреймворками.

В программировании с графическим пользовательским интерфейсом, когда мы меняем состояние одного виджета, мы хотим, чтобы уведомлялся другой виджет. К примеру, если пользователь нажимает кнопку «Закрыть», мы хотим, чтобы вызывалась функция окна close(). В старых библиотеках такая связь достигается с помощью функций обратного вызова (callback). Обратный вызов – это указатель на функцию. Если вы хотите, чтобы функция обработки уведомляла вас о каком-либо событии, вы передаете указатель на другую функцию в функцию обработки. Затем функция обработки при необходимости вызывает обратный вызов. Функции обратного вызова имеют два главных недостатка: первое – они небезопасны по типу, второе – обратный вызов сильно связан с функцией обработки [4].

В Qt имеется альтернатива методу обратного вызова – механизм сигналов и слотов (см. рисунок 6). Сигнал излучается, когда происходит некоторое событие. Слот – это функция, которая вызывается в ответ на определенный сигнал.

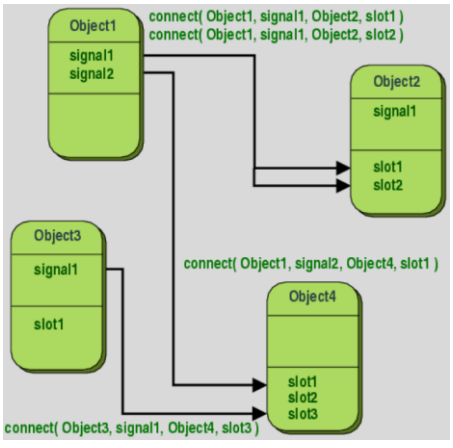

Рисунок 6 – Пример взаимодействия сигналов и слотов

Реализация данного механизма не тривиальна и требует метаобъектного компилятора, упомянутого выше. Если кратко, то метаобъектный компилятор генерирует для каждого специально помеченного класса мета таблицу, с помощью которой во время исполнения можно просматривать атрибуты объекта, то есть возможна рефлексия. У каждого объекта такого класса есть специальный список (см. рисунок 7), хранящий информацию о слотах, которые необходимо вызвать по излучению сигнала. С помощью такой реализации появляется возможность соединять сигналы и слоты во время исполнения программы, что дает большую гибкость приложению.

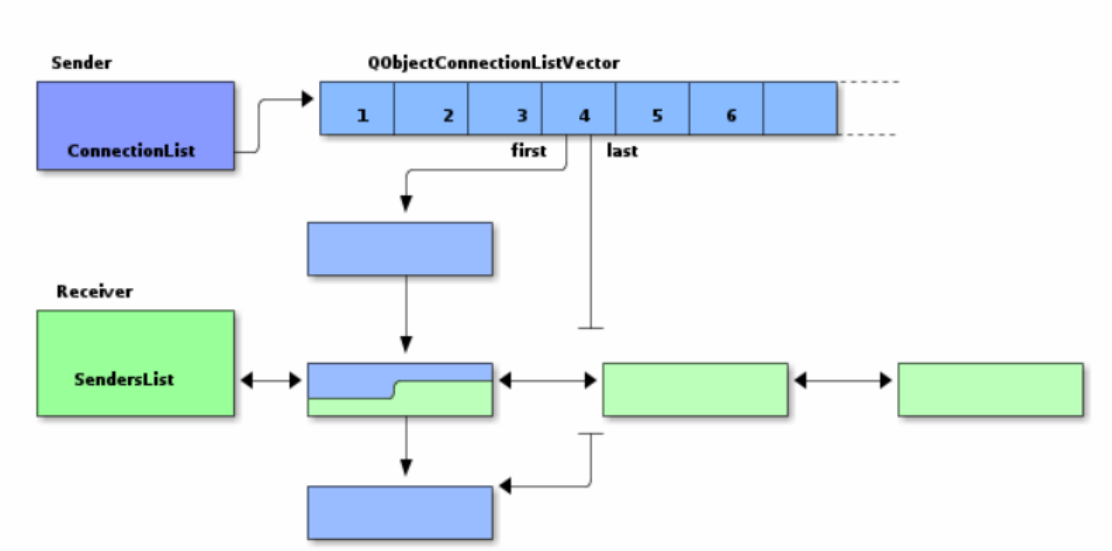

Рисунок 7 – Иллюстрация двусвязного списка, хранящего сигналы и слоты

Архитектура приложения

Архитектура для сетевого приложения заложена классическая, клиент-серверная.

Основные преимущества такого подхода:

1) Централизация. В Peer to Peer архитектуре нет централизованного администрирования, но в архитектуре клиент-сервер есть возможность централизованного управления. Серверы помогают в администрировании всей системы. Права доступа и распределение ресурсов тоже осуществляется серверами [1].
2) Правильное управление. Поскольку все файлы хранятся в одном месте, управление файлами упрощается.
3) Возможность резервного копирования и восстановления. Сделать резервную копию всех данных легко, поскольку данные хранятся на сервере. Предположим, что произошла какая-то поломка и данные утеряны, их можно легко и эффективно восстановить. В одноранговых вычислениях мы должны делать резервные копии на каждой рабочей станции.
4) Обновление и масштабируемость. Если мы хотим внести изменения, нам нужно будет просто обновить сервер. Кроме того, мы можем добавлять новые ресурсы и системы, внося необходимые изменения на сервере.
5) Доступность. С различных платформ в сети к серверу можно получить удаленный доступ.
6) По мере того, как новая информация загружается в базу данных, у каждой рабочей станции нет необходимости увеличивать собственную емкость хранения (как это может иметь место в одноранговых системах). Все изменения производятся только на центральном компьютере, на котором существует база данных сервера.

7) Безопасность. Правила, определяющие безопасность и права доступа, могут быть определены во время настройки сервера.
8) Серверы могут играть разные роли для разных клиентов.

Общий план архитектуры описываемого Мессенджера представлен на рисунке 8.

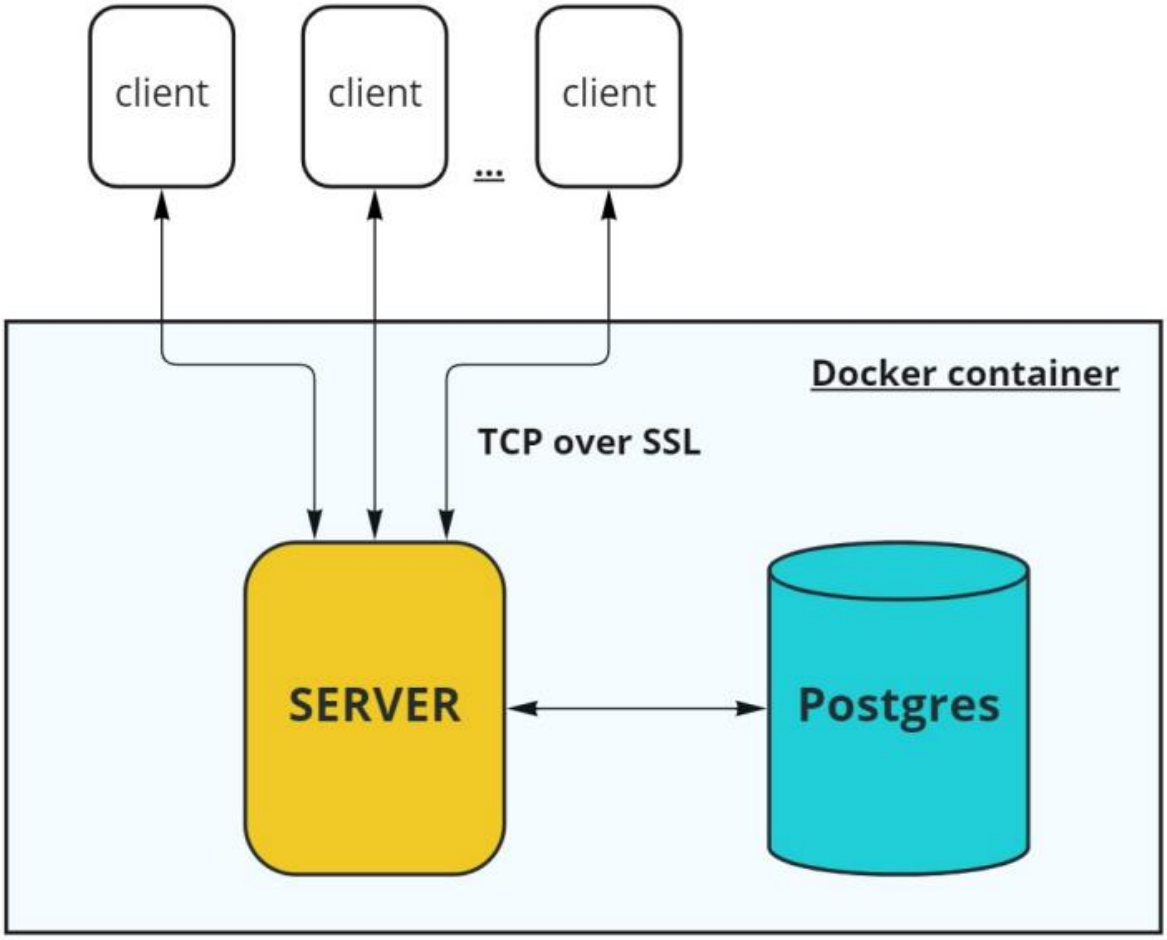

Рисунок 8 – Диаграмма компонентов системы

Сервер работает на удаленной машине и взаимодействует с СУБД PostgreSQL для работы с данными. Клиенты могут исполняться как на разных машинах, так и на одной. Клиенты не общаются с друг другом напрямую, всё взаимодействие происходит только с сервером. Сервер обрабатывает все события: соединения пользователей, отсоединения, появление сообщений от пользователей и т.д.

Глава II. Реализация приложения

План проекта

Создание приложения осуществлялось по следующему плану:

1. Исследование в области реализации подобных сетевых приложений.
2. Проектирование архитектуры.
3. Подготовка инфраструктуры для разработки приложения (настройка системы сборки, системы контроля версий, настройка IDE, настройка Docker контейнера для запуска сервера).
4. Сборка библиотеки для возможности подключения к СУБД Postgres на языке C++.
5. Разработка схемы базы данных для приложения.
6. Разработка протокола на базе JSON для передачи сообщений между клиентом и сервером.
7. Разработка базовой части сервера.
8. Разработка пула потоков для сервера.
9. Разработка пула соединений для сервера.
10. Разработка таймера для освобождения неиспользуемых соединений.
11. Разработка базовой части клиента.
12. Разработка графического интерфейса клиента.
13. Разработка регистрации в приложении.
14. Разработка авторизации в приложении.
15. Добавление возможности создания множества групп для общения.
16. Переход на SSL сокеты для безопасной передачи пакетов в сети.
17. Тестирование системы.

Приложение должно удовлетворять следующим требованиям:

Клиентская часть должна использовать минимальное количество вычислительных ресурсов и занимать минимум памяти на диске, иметь понятный и минималистичный интерфейс. Должна быть возможность регистрации и авторизации в приложении, создания групп и присоединения к

ним с помощью пароля, и отправка сообщений в группу. Передача данных должна быть зашифрована.

Серверная часть должна состоять из основного консольного приложения и СУБД PostgreSQL, в которой хранятся данные о группах, пользователях. Сервер должен поддерживать одновременное подключение множества клиентов и работу с ними без конфликтов. Сервер должен писать информативные логи, с помощью которых можно узнать, по какой причине произошел сбой, смоделировать причину и устранить её. Передача данных должна быть зашифрована.

Структура проекта

Файловая структура клиентской части представлена на рисунке 9.

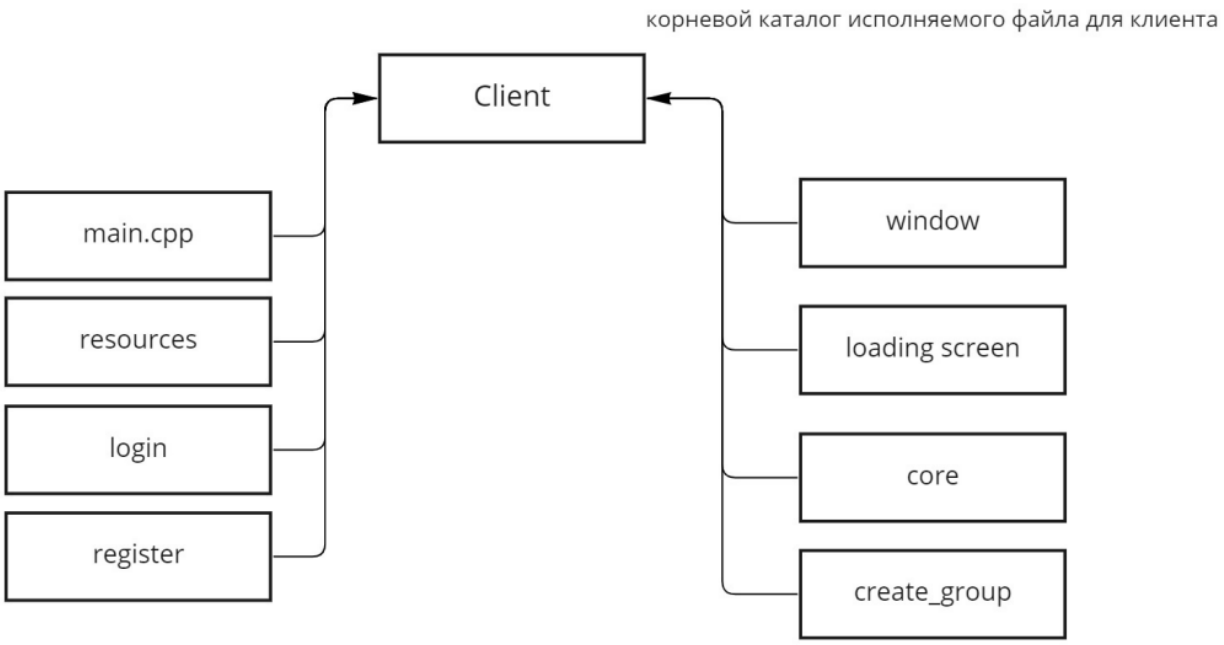

Рисунок 9 - Файловая структура клиентской части

Файловая структура серверной части представлена на рисунке 10.

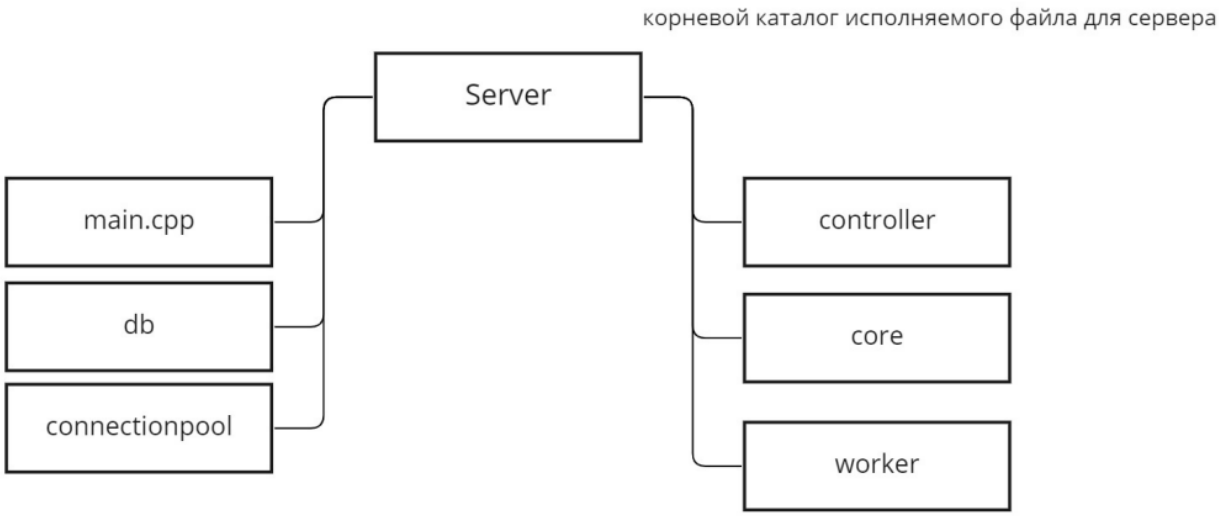

Рисунок 10 - Файловая структура серверной части

Взаимодействие компонентов изображено на рисунке 11.

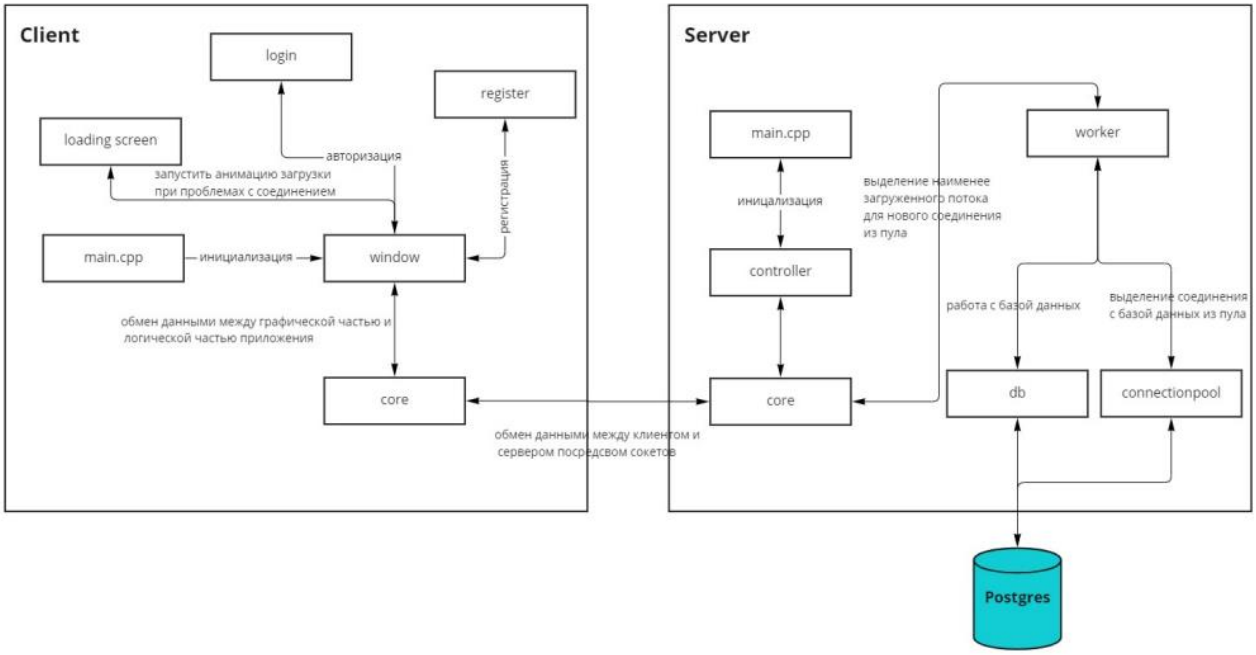

Рисунок 11 - Диаграмма потоков данных

База данных и взаимодействие с ней

В качестве СУБД выбрана PostgreSQL так как она является самой передовой реляционной базой данных с открытым исходным кодом [3]. Она также доступна почти для любой операционной системы. Для работы с PostgreSQL в Qt используются классы QSqlDatabase и QSqlQuery. Также к серверу присоединяется статическая библиотека libpq.lib которая необходима для работы с Postgres.

Схема базы данных, описанная с помощью ERD диаграммы изображена на рисунке 12.

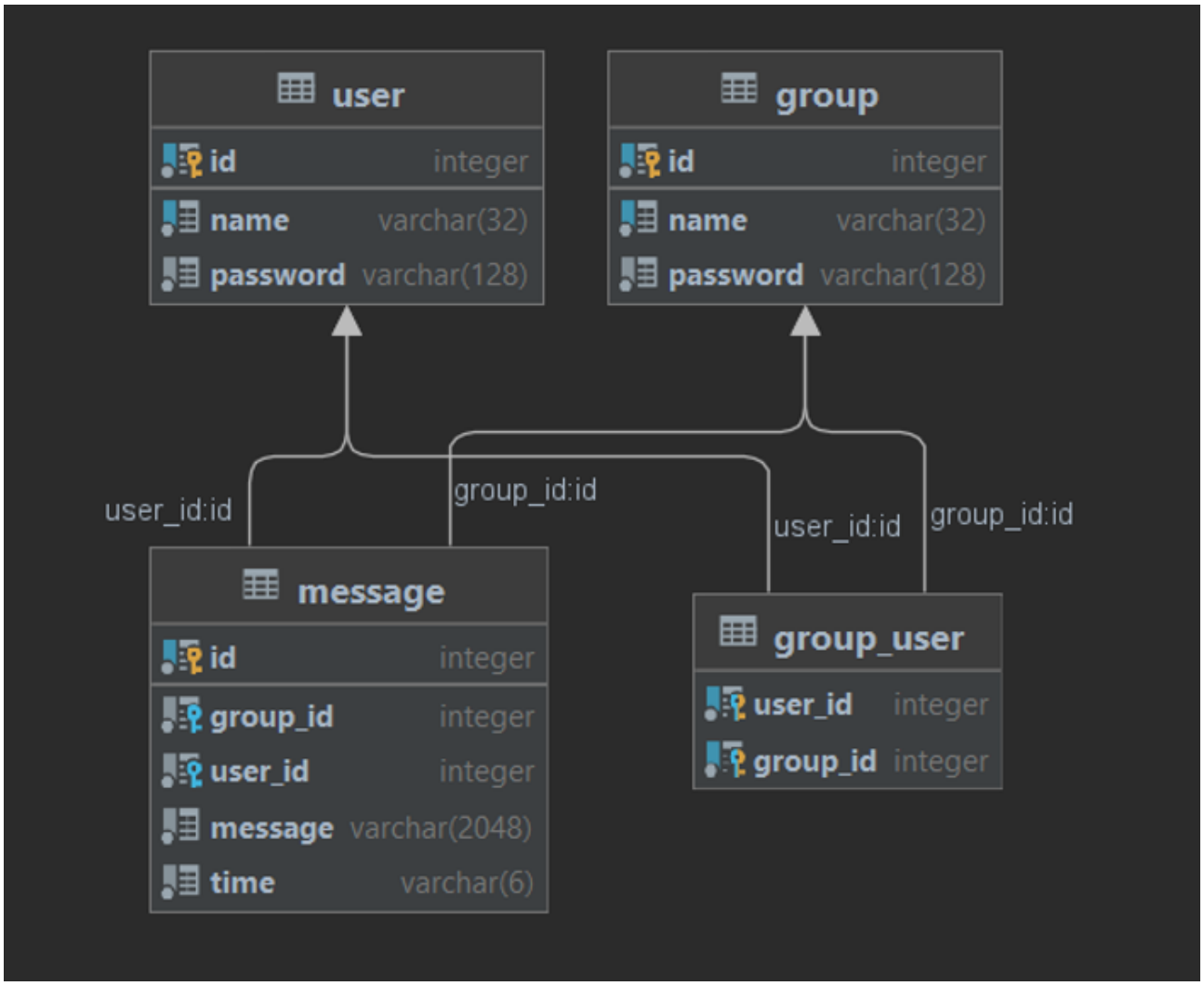

Рисунок 12 - Схема БД

Для эффективной работы с базой данных на стороне сервера был создан класс пула соединений (см. рисунок 13).

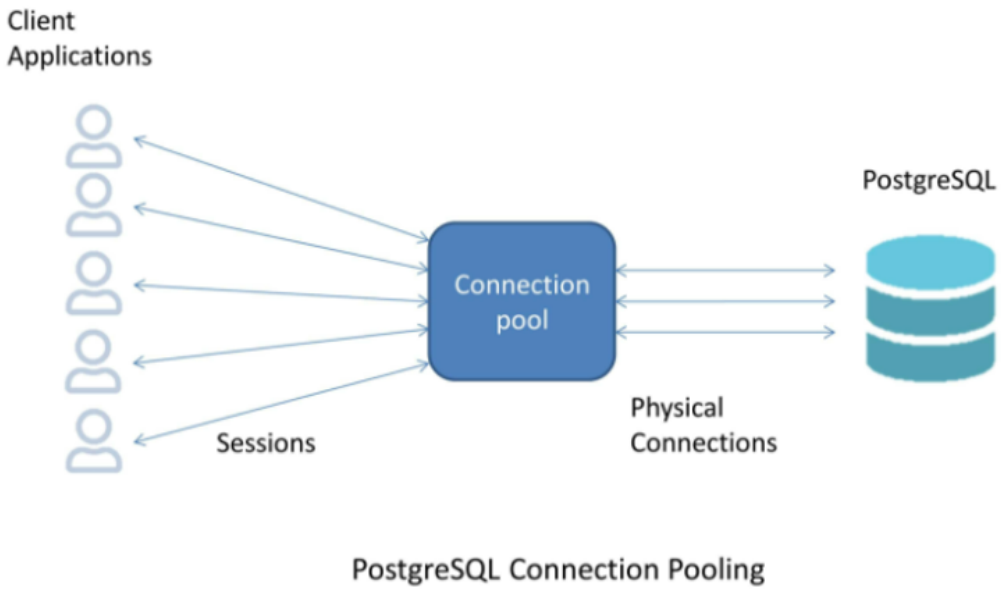

Рисунок 13 - Схема работы с пулом соединений

Характеристики пула соединений с базой данных, которые необходимо достичь при реализации:

- Не нужно знать имя соединения при установлении соединения
- Поддержка многопоточности, чтобы гарантировать, что полученное соединение не используется другими потоками
- Создание соединений по запросу
- Возможность создавать несколько подключений
- Возможность контролировать количество подключений
- Возможность повторно использовать соединения вместо того, чтобы каждый раз воссоздавать новое соединение
- Автоматическое переподключение после отключения
- Когда соединение недоступно, поток, получающий соединение, будет ждать определенное количество времени, чтобы попытаться продолжить получение, и не будет возвращать недопустимое соединение, пока не истечет время ожидания

- Легкое закрытие соединений.
- Автоматическое освобождение соединений после прохождения заданного времени

Интерфейс для работы с пулом соединений представлен на таблице 2.

Таблица 2 – открытый интерфейс пула соединений

метод	код
Получить соединение	auto conn = ConnectionPool::getConnection();
Освободить соединение	ConnectionPool::releaseConnection(conn);
Освободить пул соединений	ConnectionPool::release();

Пример использования интерфейса пула соединений в функции проверки существования пользователя (см. рисунок 14).

```
bool db::isUserExist(const QString& userName)
{
    auto conn = ConnectionPool::getConnection();
    QSqlQuery query(conn);
    query.prepare(R"(select id from "Messenger".public.user where "name" = :name)");
    query.bindValue(":name", userName);
    query.exec();

    int id = 0;
    while (query.next()) {
        id = query.value("id").toInt();
    }
    ConnectionPool::releaseConnection(conn);

    return id != 0;
}
```

Рисунок 14 – Метод isUserExist

Для эффективного использования ресурсов базы данных соединения освобождаются после таймаута в 300 секунд. Для этого используется специальный таймер *QTimer*, который срабатывает каждые 300 секунд и проверяет все соединения. Если находится не используемое соединение, он его освобождает. Реализация функции, вызываемой с помощью таймера (см. рисунок 15).

```
void ConnectionPool::releaseUnusedConnections()
{
    QMutexLocker locker(&mutex);
    for (auto& unusedConnection : unusedConnections) {
        if (!unusedConnection.released) {
            QSqlDatabase::removeDatabase(unusedConnection.connectionName);
            qInfo() << "released connection: " << unusedConnection.connectionName;
            unusedConnection.released = true;
        }
    }
}
```

Рисунок 15 – метод releaseUnusedConnections

Протокол взаимодействия клиента сервера

Для передачи информации по сети используются TCP сокеты, для которых нужно указать лишь адрес и порт назначения. Но просто сокеты могут передавать лишь байты, что не очень удобно при реализации приложения. Поэтому в качестве формата передачи сообщений был выбран формат JSON. JSON – это облегченный формат обмена данными. Людям легко читать и писать с помощью него. Машины также легко анализируют и генерируют его.

JSON состоит из двух структур:

- Коллекция пар имя/значение. На разных языках это реализовано как объект, запись, структура, словарь, хеш-таблица, список с ключами или ассоциативный массив.
- Упорядоченный список значений. В большинстве языков это реализовано как массив, вектор, список или последовательность.

Для передачи сообщений между клиентом и сервером формируется JSON структура, у которой могут быть следующие типы и данные внутри сообщения (см. рисунок 16).

```cpp
namespace Packet {
    namespace Type {
        constexpr const char* const TYPE          = "type";
        constexpr const char* const LOGIN         = "login";
        constexpr const char* const REGISTER      = "register";
        constexpr const char* const CONNECT_GROUP = "connect_group";
        constexpr const char* const CREATE_GROUP  = "create_group";
        constexpr const char* const USER_JOINED   = "user_joined";
        constexpr const char* const USER_LEFT     = "user_left";
        constexpr const char* const MESSAGE       = "message";
        constexpr const char* const INFORM_JOINER = "inform_joiner";
    } // namespace Type
    namespace Data {
        constexpr const char* const USERNAME   = "username";
        constexpr const char* const GROUP_NAME = "group_name";
        constexpr const char* const PASSWORD   = "password";
        constexpr const char* const TEXT       = "text";
        constexpr const char* const SENDER     = "sender";
        constexpr const char* const SUCCESS    = "success";
        constexpr const char* const REASON     = "reason";
        constexpr const char* const USERNAMES  = "usernames";
        constexpr const char* const MESSAGES   = "messages";
        constexpr const char* const TIME       = "time";
    } // namespace Data
} // namespace Packet
```

Рисунок 16 - Типы сообщений

Пример формирования JSON сообщения и отправки его пользователю (см. рисунок 17).

```cpp
if (isUserLoggedIn(userName)) {
    QJsonObject errorPacket;
    errorPacket[Packet::Type::TYPE]    = Packet::Type::LOGIN;
    errorPacket[Packet::Data::SUCCESS] = false;
    errorPacket[Packet::Data::REASON]  = "user with such name already logged in";
    sendPacket(sender, errorPacket);
    return;
}
```

Рисунок 17 - Код формирования сообщения

После формирования сообщения оно сериализуется и отправляется по сети через TCP сокет (см. рисунок 18).

```
void ClientCore::sendMessage(const QString& message, const QString& time)
{
    QDataStream clientStream(clientSocket);
    clientStream.setVersion(serializerVersion);

    QJsonObject packet;
    packet[Packet::Type::TYPE]   = Packet::Type::MESSAGE;
    packet[Packet::Data::SENDER] = name;
    packet[Packet::Data::TEXT]   = message;
    packet[Packet::Data::TIME]   = time;
    clientStream << QJsonDocument(packet).toJson();
}
```

Рисунок 18 - Метод sendMessage

Парсинг структуры, рассмотренной выше (см. рисунок 19).

```
void ClientCore::handleMessagePacket(const QJsonObject& packet)
{
    const QJsonValue senderVal = packet.value(QLatin1String(Packet::Data::SENDER));
    if (senderVal.isNull() || !senderVal.isString()) {
        return;
    }
    const QJsonValue textVal = packet.value(QLatin1String(Packet::Data::TEXT));
    if (textVal.isNull() || !textVal.isString()) {
        return;
    }
    const QJsonValue timeVal = packet.value(QLatin1String(Packet::Data::TIME));
    if (timeVal.isNull() || !timeVal.isString()) {
        return;
    }
    emit messageReceived({senderVal.toString(), textVal.toString(), timeVal.toString()});
}
```

Рисунок 19 - Метод handleMessagePacket

Структура выше отображается в класс Message (см. рисунок 20), который содержит имя отправителя, непосредственно сообщение и время отправки сообщения.

```
class Message
{
public:
    Message() = default;
    Message(const QString& sender, const QString& message, const QString& time);
    [[nodiscard]] const QString& getSender() const;
    [[nodiscard]] const QString& getMessage() const;
    [[nodiscard]] const QString& getTime() const;

private:
    QString sender;
    QString message;
    QString time;
};
```

Рисунок 20 - Класс Message

Реализация Сервера

Основным классом является класс ServerCore (см. рисунок 21), который наследуется от класса QTcpServer. Используется перегрузка виртуального метода *incomingConnection* для того, чтобы удобно обрабатывать входящие подключения.

```
class ServerCore : public QTcpServer
{
    Q_OBJECT
    Q_DISABLE_COPY(ServerCore)
public:
    explicit ServerCore(QObject* parent = nullptr);
    ~ServerCore() override;

protected:
    void incomingConnection(qintptr socketDescriptor) override;

private:
    const int idealThreadCount;
    QVector<QThread*> threads;
    QVector<int> threadLoadFactor;
    QVector<ServerWorker*> clients;
private slots:
    void unicast(const QJsonObject& packet, ServerWorker* receiver);
    void broadcast(const QString& group, const QJsonObject& packet, const ServerWorker* exclude);
    void packetReceived(ServerWorker* sender, const QJsonObject& packet);
    void userDisconnected(ServerWorker* sender, int threadIdx);
    static void userError(ServerWorker* sender);
public slots:
    void stopServer();

private:
    void loginUser(ServerWorker* sender, const QJsonObject& packet);
    bool isUserLoggedIn(const QString& username);
```

Рисунок 21 - Класс ServerCore

Запускается основной модуль из объекта класса ServerController (см. рисунок 22).

```
class ServerController
{
public:
    ServerController();
    ~ServerController();
    void startServer();

private:
    ServerCore* serverCore;
};
```

Рисунок 22 - Класс ServerController

Основная работа ServerCore – это обработка событий от клиентов (соединение, прием сообщения, регистрация, авторизация, …).

В классе содержится пул потоков *threads* (см. рисунок 23) для присоединяющихся клиентов. Как только присоединяется новый клиент под него выделяется наименее загруженный поток из пула. При старте пул пустой - потоки не выделены. Далее, при присоединении новых клиентов, создаются новые потоки до ограничения *threadLoadFactor*, который задается с помощью функции *QThread::idealThreadCount.*

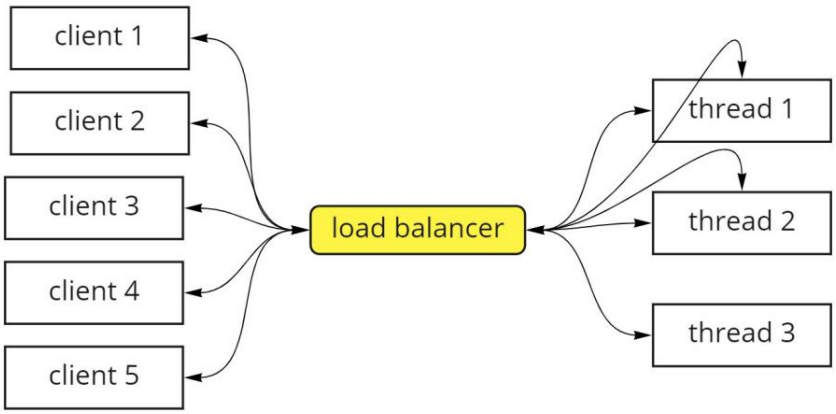

Рисунок 23 - Схема работы пула потоков

Под каждого клиента в сервере выделяется сокет, который хранится в объекте класса ServerWorker, в свою очередь worker работает в потоке, выделившимся из пула потоков.

Объявление класса ServerWorker (см. рисунок 24).

```
class ServerWorker : public QObject
{
    Q_OBJECT
    Q_DISABLE_COPY(ServerWorker)
public:
    explicit ServerWorker(QObject* parent = nullptr);
    ~ServerWorker() override;

    virtual bool setSocketDescriptor(qintptr socketDescriptor);
    QString getUserName() const;
    void setUserName(const QString& name);
    QString getGroupName() const;
    void setGroupName(const QString& name);
    void sendPacket(const QJsonObject& packet);
public slots:
    void disconnectFromClient();
private slots:
    void onReadyRead();
signals:
    void packetReceivedSig(const QJsonObject& packet);
    void disconnectedFromClientSig();
    void errorSig();

private:
    QSslSocket* serverSocket;
    QString userName;
```

Рисунок 24 - Класс ServerWorker

Прослушивание входящих пакетов в виде JSON структуры происходит в методе onReadyRead (см. рисунок 25).

```
void ServerWorker::onReadyRead()
{
    QByteArray jsonData;
    QDataStream socketStream(serverSocket);
    socketStream.setVersion(serializerVersion);
    while (true) {
        socketStream.startTransaction();
        socketStream >> jsonData;
        if (socketStream.commitTransaction()) {
            QJsonParseError parseError  = {0};
            const QJsonDocument jsonDoc = QJsonDocument::fromJson(jsonData, &parseError);
            if (parseError.error == QJsonParseError::NoError) {
                if (jsonDoc.isObject()) {
                    emit packetReceived(jsonDoc.object());
                } else {
                    qInfo() << qPrintable(QString("invalid message: ") + QString::fromUtf8(jsonData));
                }
            } else {
                qInfo() << qPrintable(QString("invalid message: ") + QString::fromUtf8(jsonData));
            }
        } else {
            break;
        }
    }
}
```

Рисунок 25 - Метод onReadyRead

Далее после получения полноценной структуры она отправляется дальше необходимому обработчику на основе типа сообщения.

Реализация Клиента

Клиентская часть разделена на два основных класса. За графический интерфейс и взаимодействие с пользователем отвечает класс ClientWindow, за общение с сервером отвечает класс ClientCore.

Класс ClientWindow (см. рисунок 26) наследуется от класса QWidget и имеет графический интерфейс, реализованный в ui формате, который компилируется в h файл с помощью компилятора uic.

```
class ClientWindow : public QWidget
{
    Q_OBJECT
    Q_DISABLE_COPY(ClientWindow)
public:
    explicit ClientWindow(QWidget* parent = nullptr);
    ~ClientWindow() override;

private:
    Ui::ClientWindow* ui;
    ClientCore* clientCore;
    QStandardItemModel* chatModel;
    QString lastUserName;
    Login* loginWindow;
    Register* registerWindow;
    CreateGroup* createGroupWindow;
    LoadingScreen* loadingScreen;
    bool logged;
    static constexpr int minWindowWidth    = 750;
    static constexpr int minWindowHeight   = 500;
    static constexpr int maxMessageRowSize = 50;
    static constexpr int maxMessageSize    = 2048;
private slots:
    void connected();
    void loggedIn();
```

Рисунок 26 - Класс ClientWindow

Ui файл (см. рисунок 27) форматом похож на XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>ClientWindow</class>
 <widget class="QWidget" name="ClientWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>750</width>
    <height>500</height>
   </rect>
  </property>
  <property name="minimumSize">
   <size>
    <width>750</width>
    <height>500</height>
   </size>
  </property>
  <property name="windowTitle">
   <string>Messenger</string>
  </property>
  <layout class="QGridLayout" name="gridLayout" columnstretch="40,100">
   <item row="0" column="0">
    <layout class="QVBoxLayout" name="verticalLayout">
     <item>
```

Рисунок 27 - Ui код

Графический интерфейс, который получается с помощью такого файла представлен на рисунке 28.

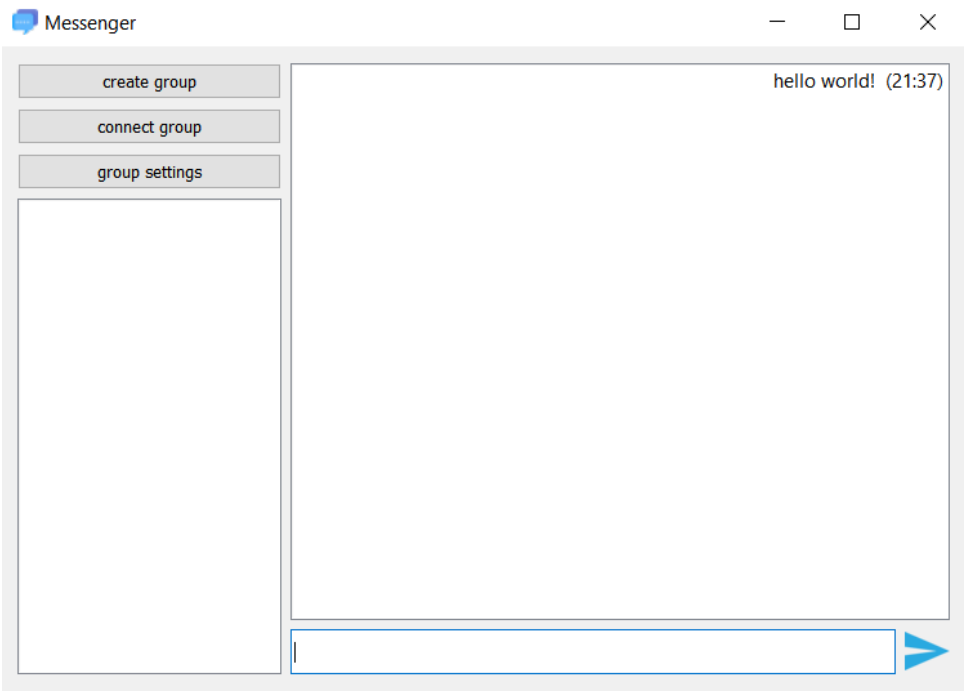

Рисунок 28 - Главное окно приложения

Класс ClientCore (см. рисунок 29) взаимодействует с сервером для обмена сообщениями, а также с классом ClientWindow, чтобы отображать состояние в графическом интерфейсе.

```
class ClientCore : public QObject
{
    Q_OBJECT
    Q_DISABLE_COPY(ClientCore)
public:
    explicit ClientCore(QObject* parent = nullptr);
    ~ClientCore() override;
    [[nodiscard]] QString getName() const;
    void connectToServer(const QHostAddress& address, quint16 port);
    void login(const QString& username, const QString& password);
    void registerUser(const QString& username, const QString& password);
    void connectGroup(const QString& groupName, const QString& password);
    void createGroup(const QString& groupName, const QString& password);
    void sendMessage(const QString& message, const QString& time);
    void disconnectFromHost();

private slots:
    void onReadyRead();
signals:
    void connectedSig();
    void disconnectedSig();
    void loggedInSig();
    void registeredSig();
    void connectedToGroupSig();
    void createdGroupSig();
    void loginErrorSig(const QString& reason);
    void registerErrorSig(const QString& reason);
    void connectToGroupErrorSig(const QString& reason);
    void createdGroupErrorSig(const QString& reason);
```

Рисунок 29 - Класс ClientCore

Продемонстрируем работу приложения:

Если нет соединения с сервером, клиент будет пытаться присоединиться к серверу. При это будет гореть окно с анимацией загрузки (см. рисунок 30).

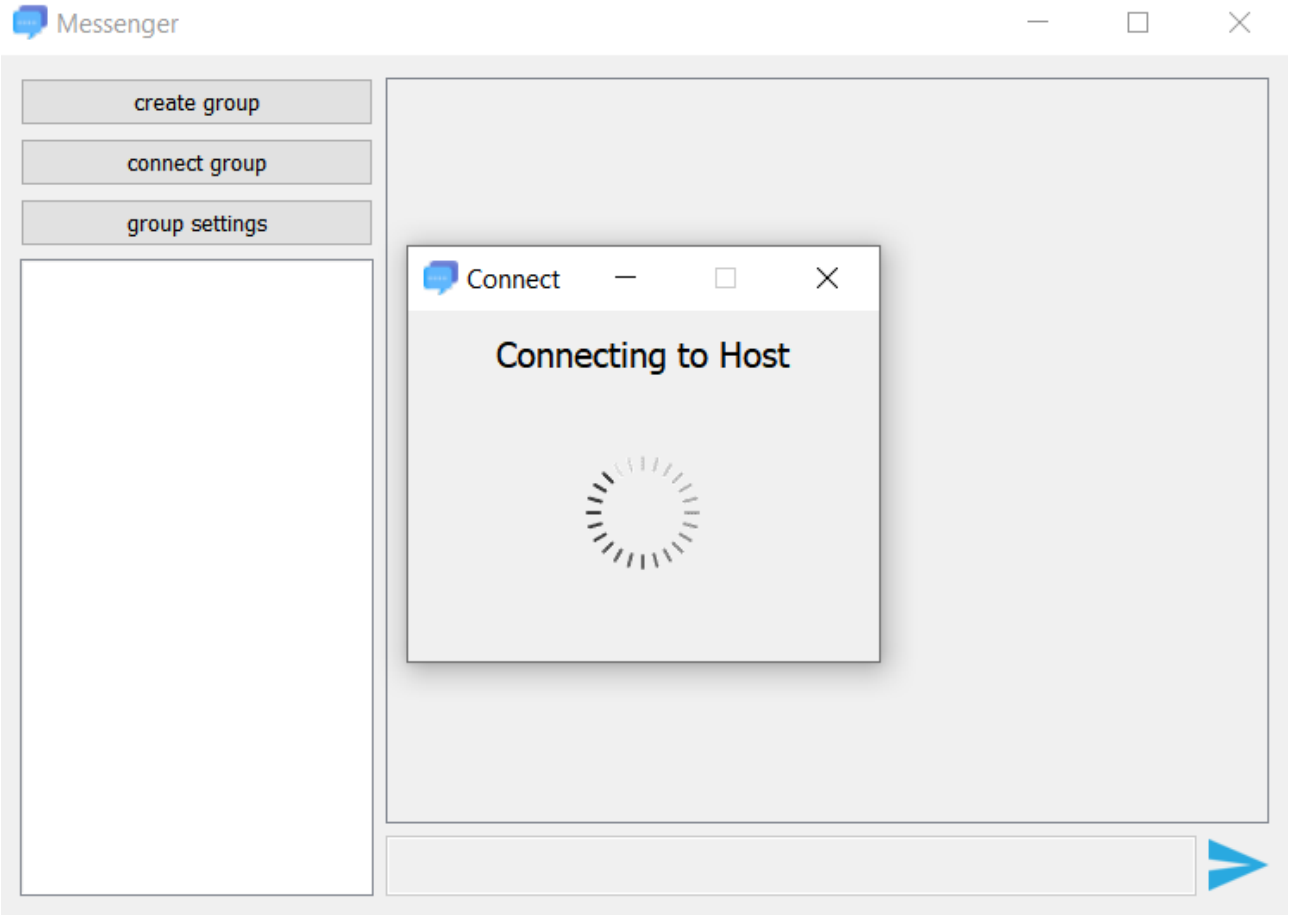

Рисунок 30 - Окно загрузки

После того, как соединение прошло, нужно указать свое имя и пароль или пройти регистрацию (см. рисунок 31).

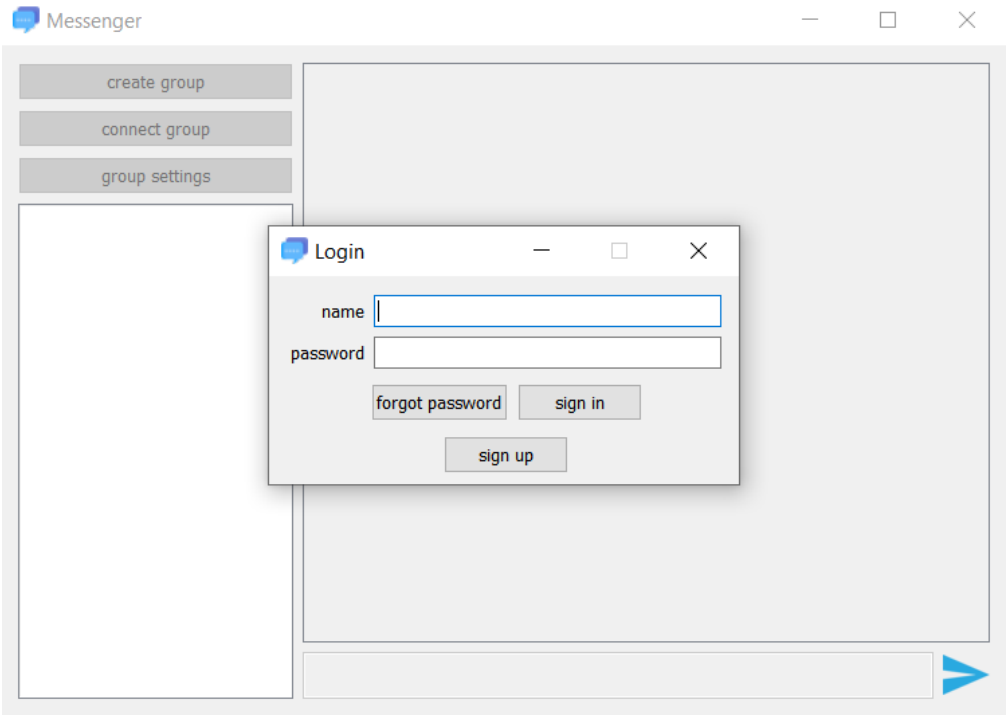

Рисунок 31 - Окно авторизации

Если пароль неверный, приложение предупредит об этом пользователя и предложит ввести реквизиты повторно (см. рисунок 32).

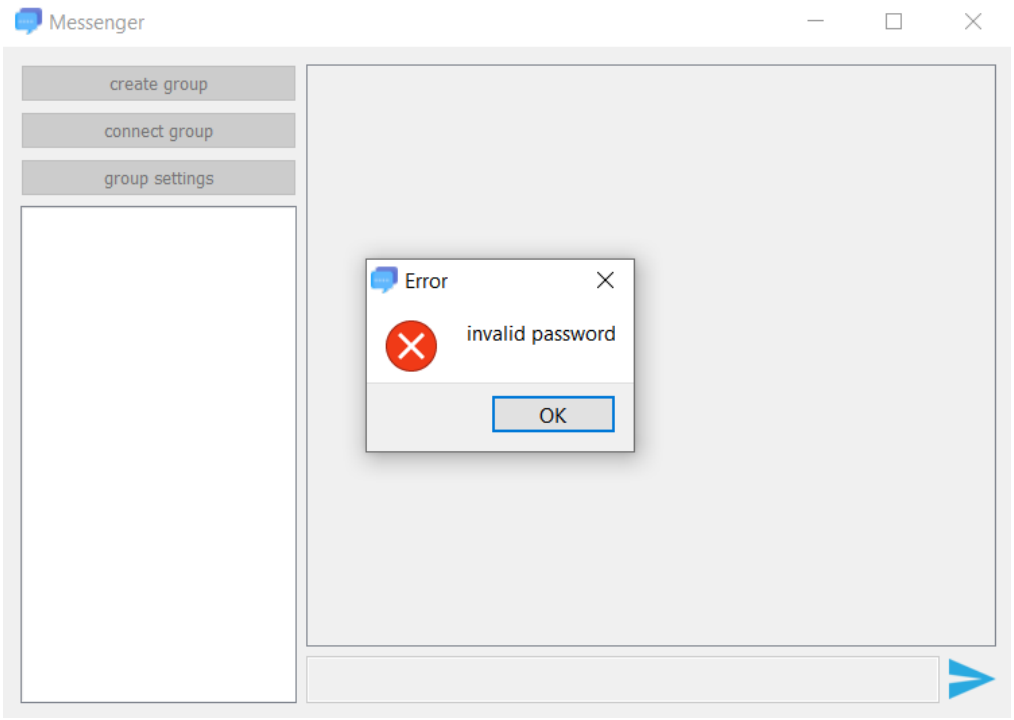

Рисунок 32 - Окно ошибки

Если пароль верный, у нас открывается доступ к созданию группы или присоединения к группе (см. рисунок 33).

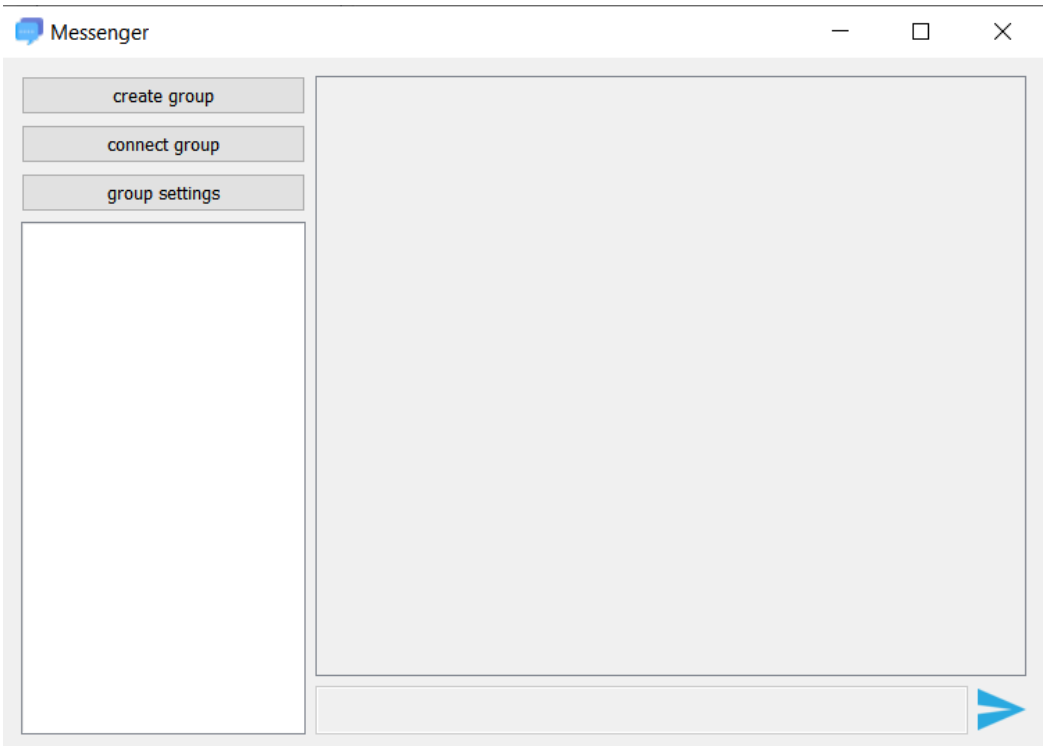

Рисунок 33 - Окно с чатом

Для регистрации необходимо нажать кнопку sign up, после чего выйдет окно, в которое нужно ввести имя пользователя и пароль (см. рисунок 34).

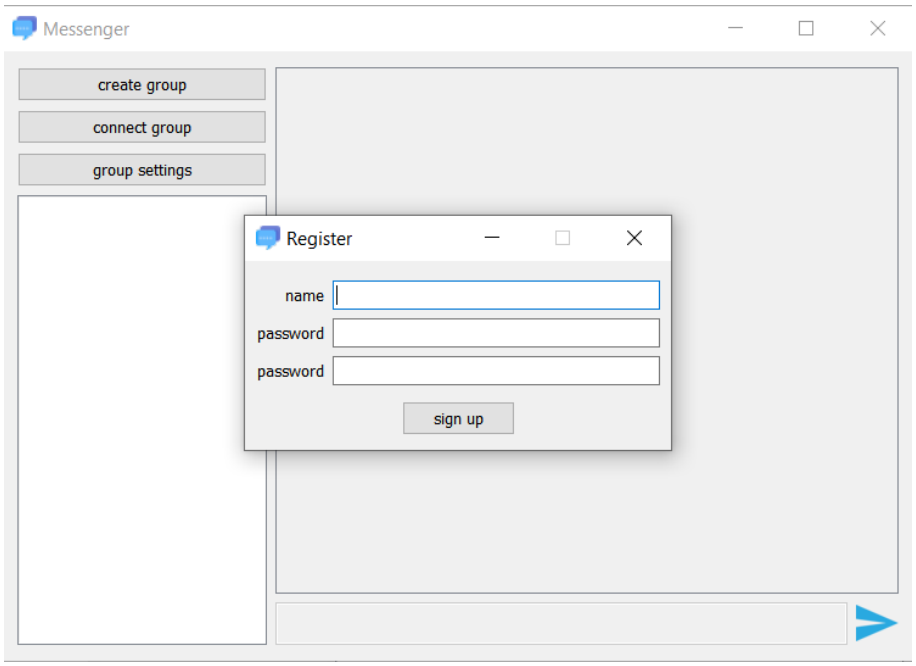

Рисунок 34 – Окно регистрации

При вводе слабого пароля выйдет предупреждение (см. рисунок 35).

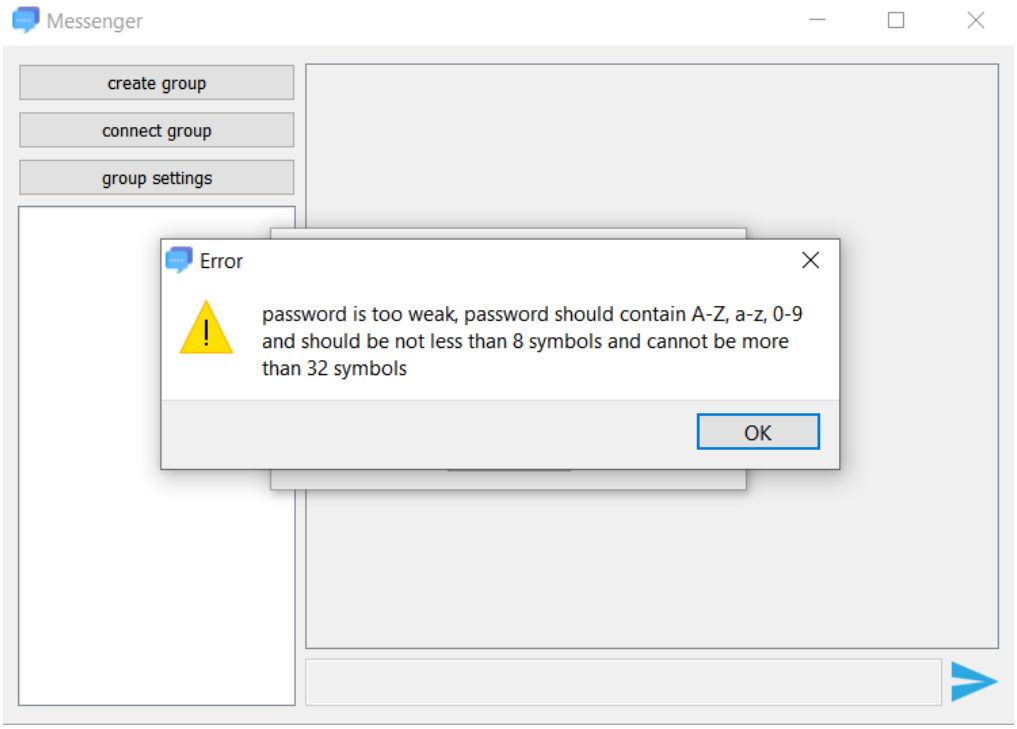

Рисунок 35 – предупреждение об использовании слабого пароля

При вводе некорректного имени с использованием различных символов отступов также выйдет предупреждение (см. рисунок 36).

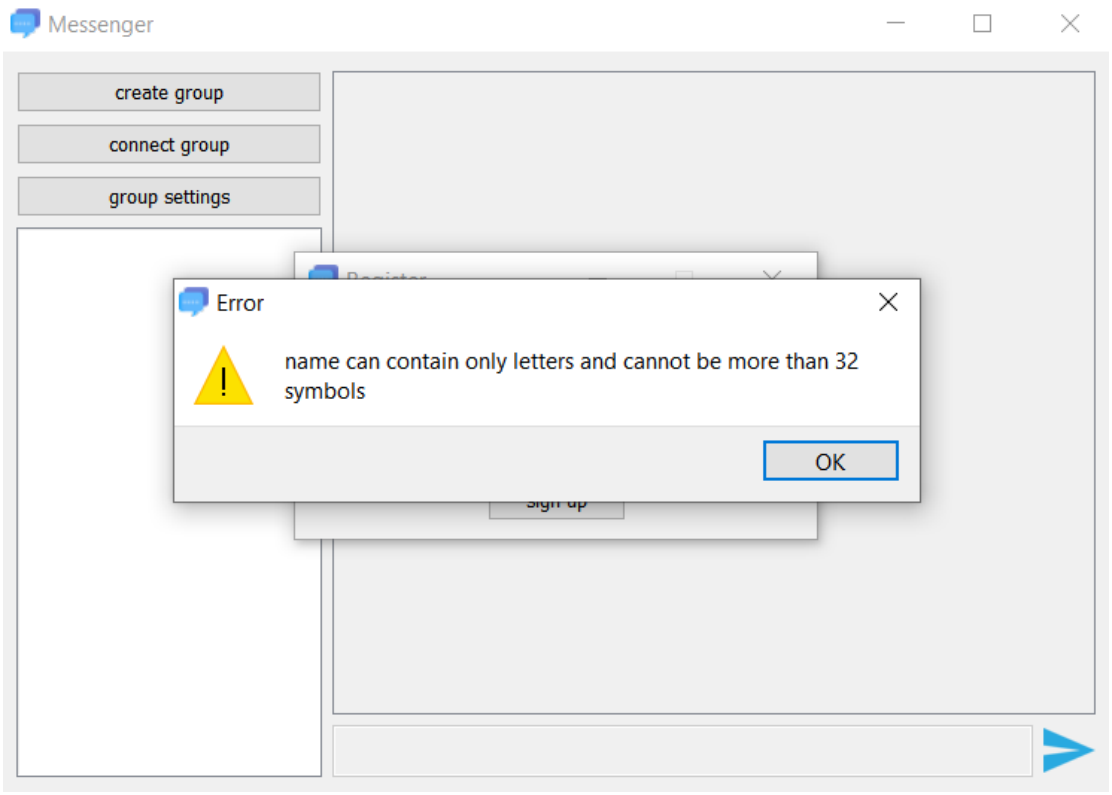

Рисунок 36 – предупреждение о вводе некорректного имени

Если попытаться создать пользователя с именем, которое уже существует в системе, также выйдет информативное предупреждение (см. рисунок 37).

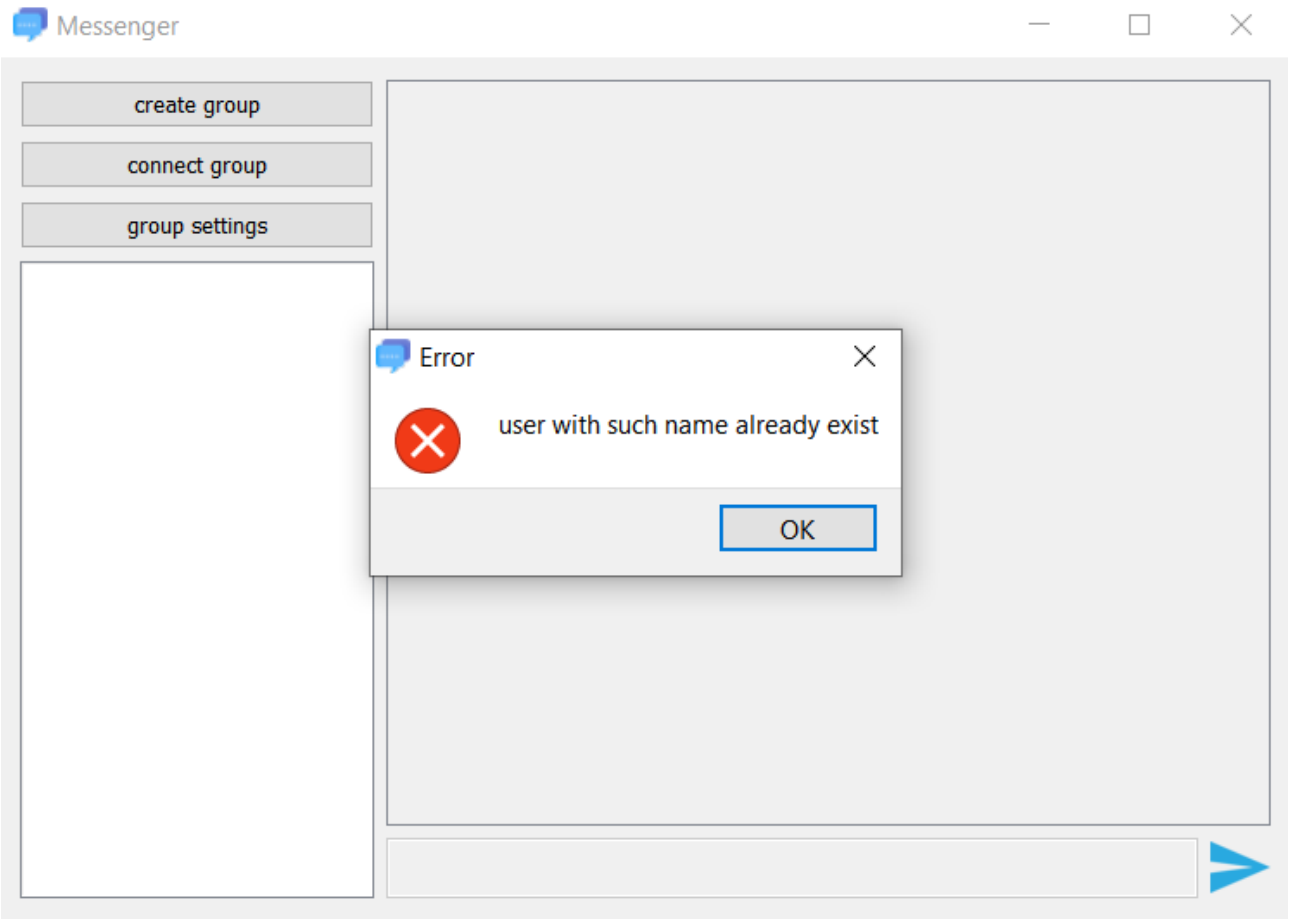

Рисунок 37 – предупреждение о существовании пользователя

Для создания группы необходимо нажать кнопку create group, пользователю будет предложено ввести название группы и опционально пароль (см. рисунок 38).

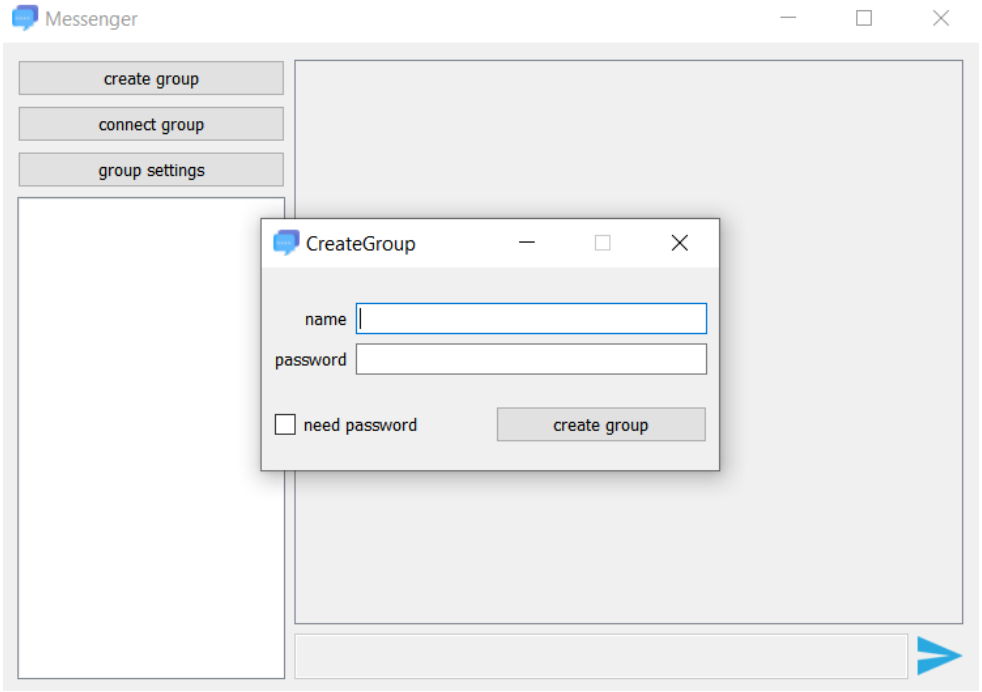

Рисунок 38 – Окно создания группы.

Для присоединения к группе необходимо нажать кнопку connect group (см. рисунок 39).

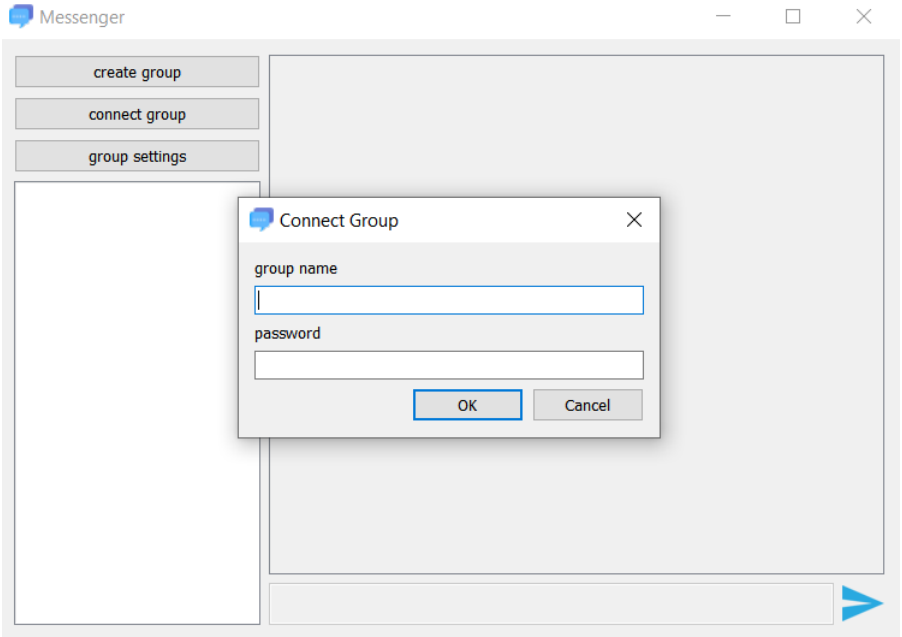

Рисунок 39 – Окно присоединения к группе

Откроем еще несколько клиентов для симуляции активного общения.

При присоединении нового пользователя, имя пользователя, который онлайн, появляется в боковом виджете. В чате видно, что присоединился новый пользователь (см. рисунок 40).

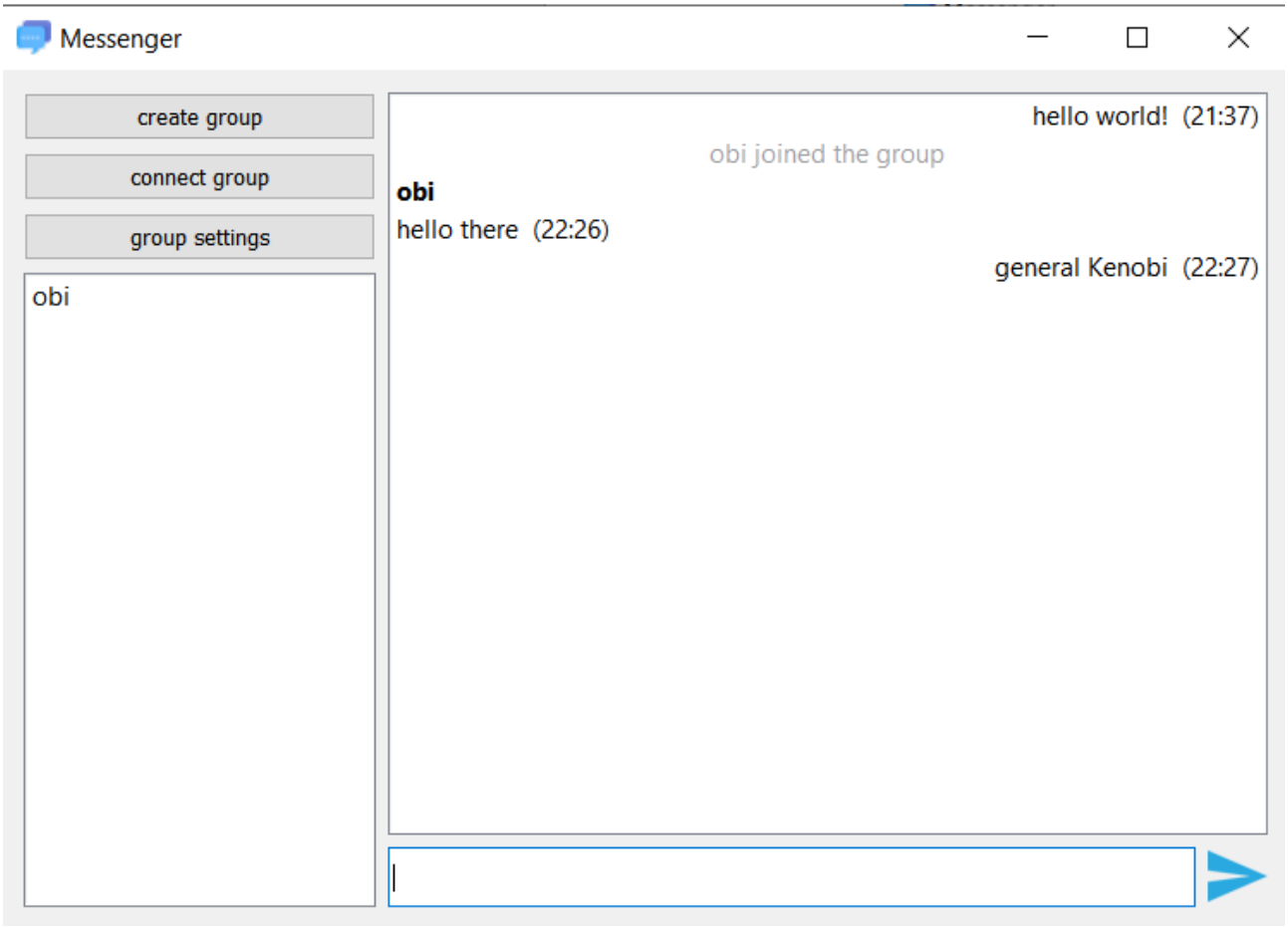

Рисунок 40 - Окно с чатом

Общение нескольких клиентов продемонстрировано на рисунке 41.

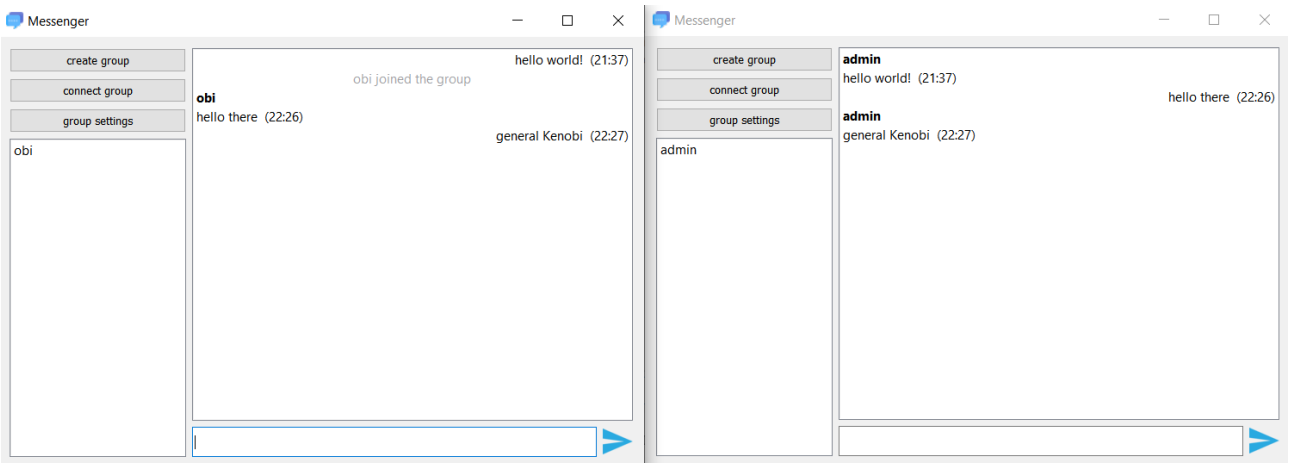

Рисунок 41 - Общение нескольких клиентов

Общение большого количества клиентов продемонстрировано на рисунке 42.

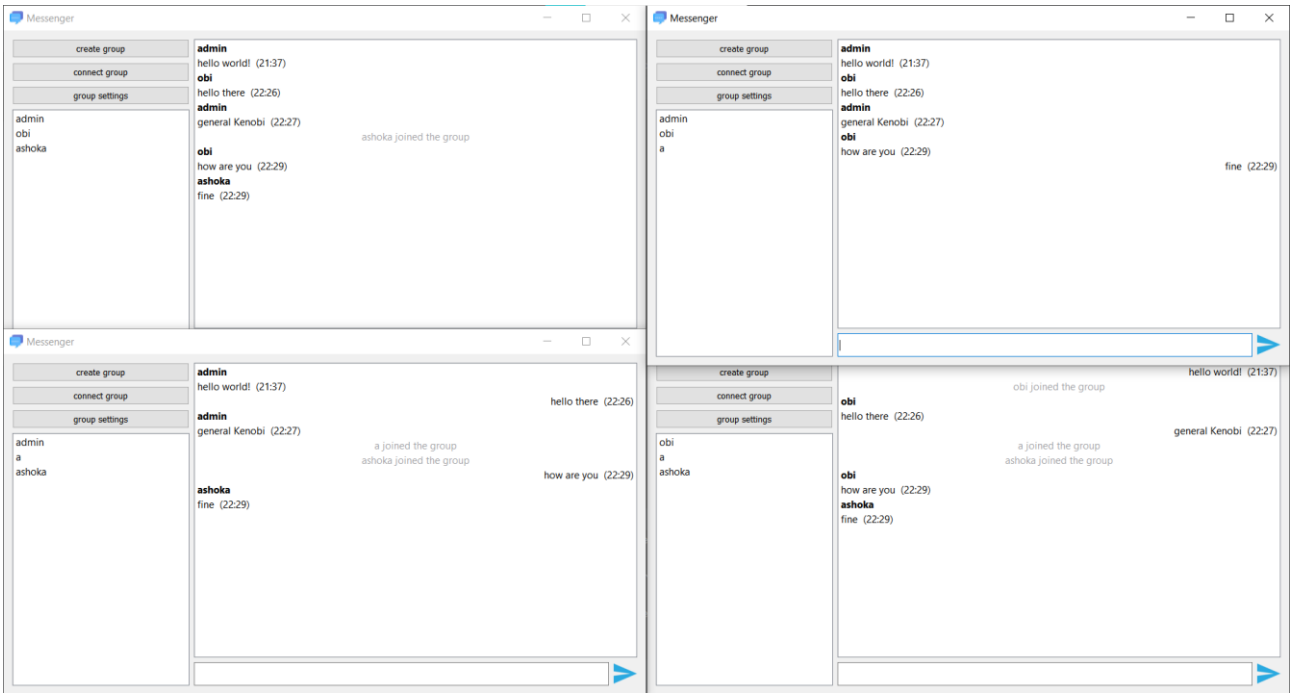

Рисунок 42 - Общение множества клиентов

При отсоединении пользователя его имя исчезает с боковой панели других клиентов. Кроме того, в чате появляется сообщение о данном событии (см. рисунок 43).

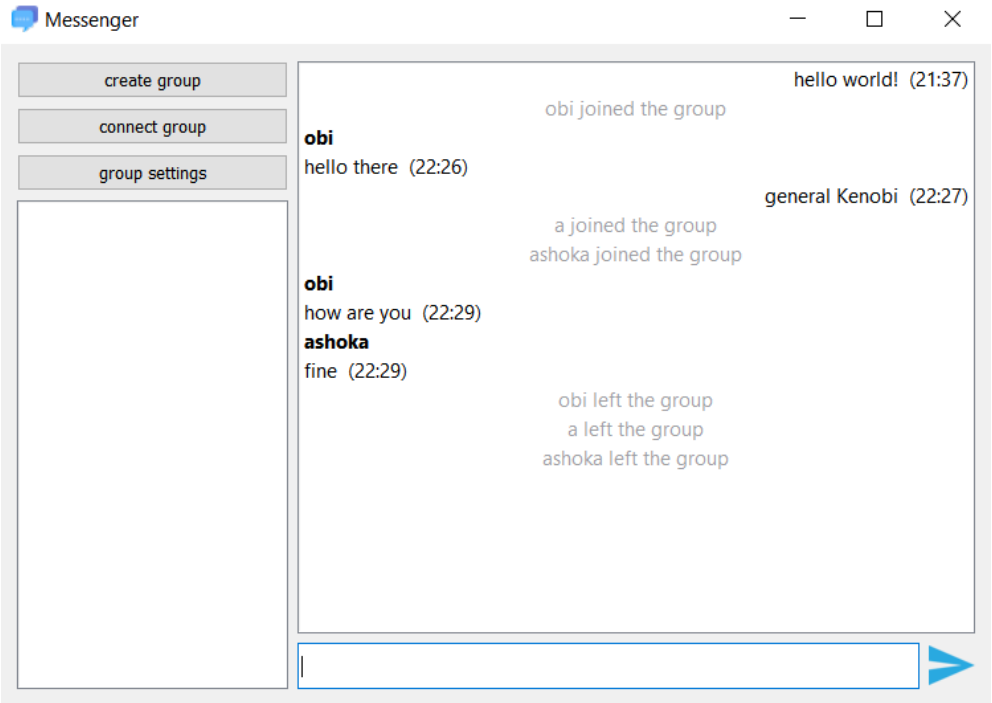

Рисунок 43 - Окно чата

При заходе клиента в чат у него появляется вся история прошлых сообщений (см. рисунок 44).

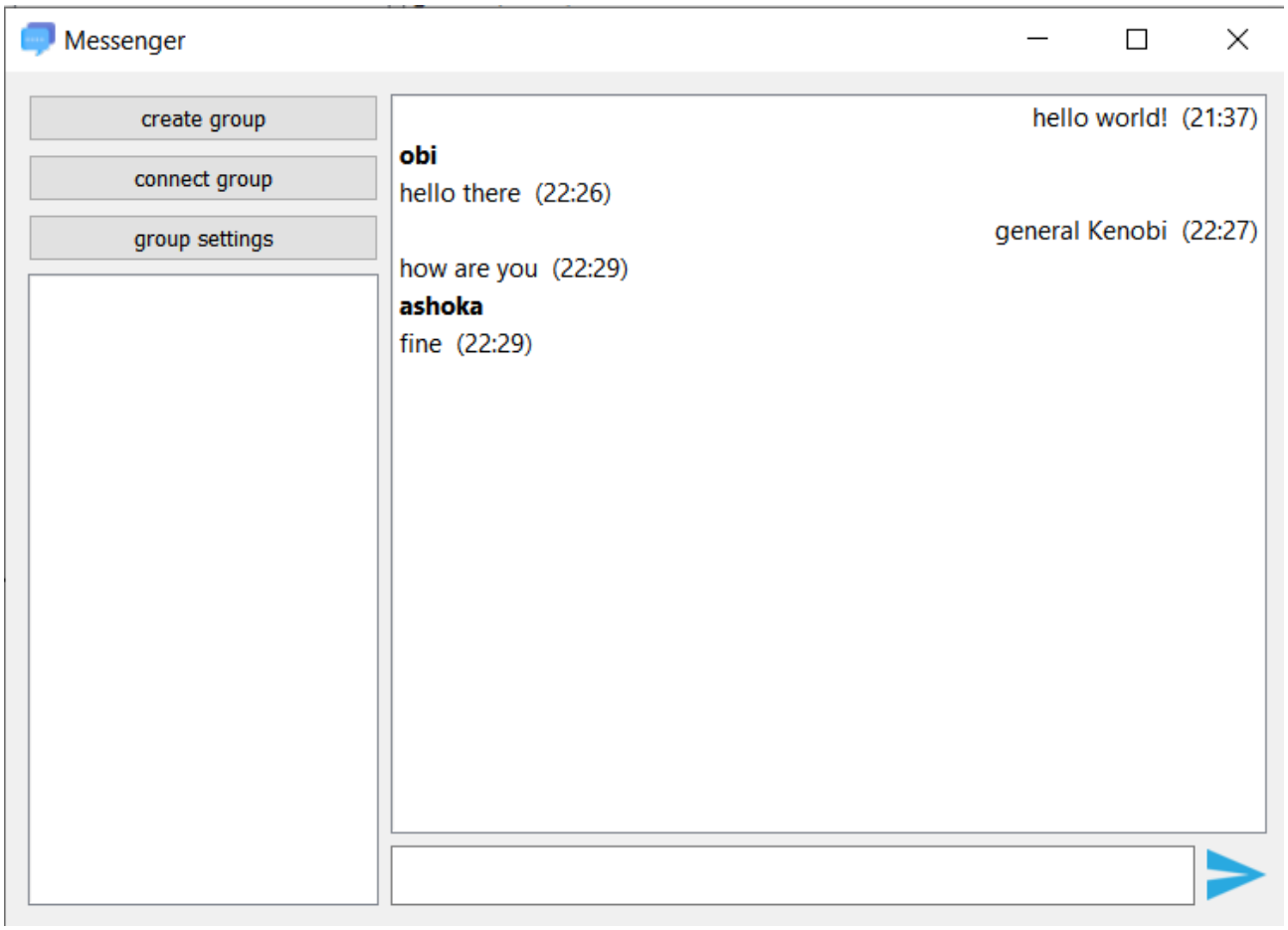

Рисунок 44 - Окно чата

Шифрование

Безопасность передачи данных в приложении обеспечивается с помощью использования SSL сокетов. SSL – это криптографический протокол, который предоставляет безопасную пересылку пакетов в сети. Он использует асимметричную шифрование для аутентификации ключей и симметричное шифрование для сохранения конфиденциальности. SSL работает поверх существующего потока TCP после того, как сокет входит в состояние соединения (см. рисунок 45).

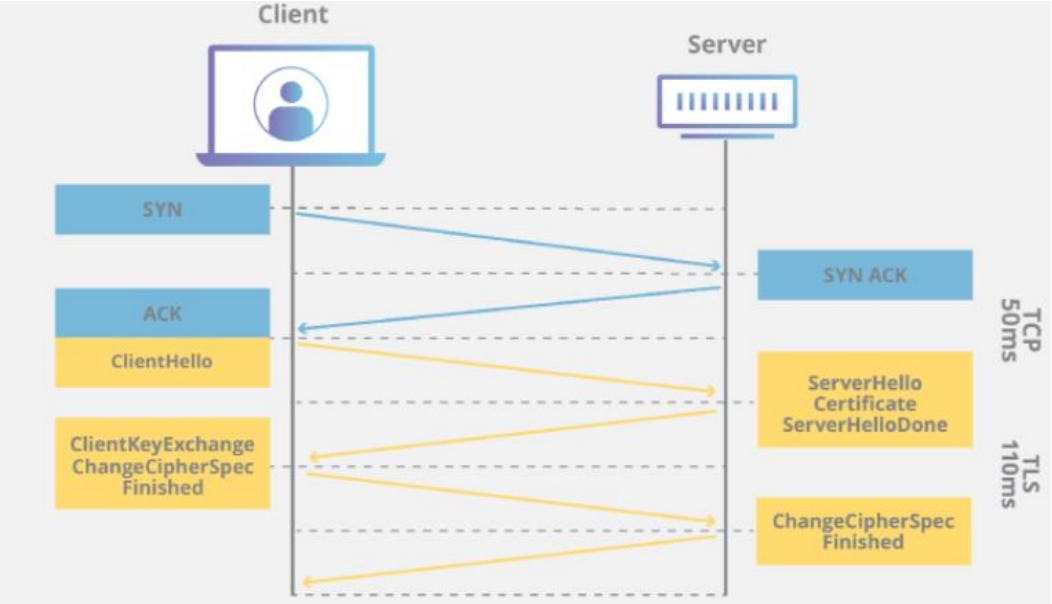

Рисунок 45 – SSL рукопожатие

В приложении для передачи данных с помощью протокола SSL используется класс QSslSocket. Для этого требуется собирать приложение с криптографической библиотекой OpenSSL, которая предоставляет функции шифрования, генерации сертификатов.

Безопасность данных обеспечивается не только от атак снаружи с помощью шифрования пакетов, но и внутри в серверной части приложения. Пароли пользователя содержатся на сервере зашифрованными с помощью алгоритма sha256. В коде после обработки данных с паролями они

немедленно затираются нулями, чтобы у злоумышленника была меньше вероятность считать чувствительные данные с физической памяти.

ЗАКЛЮЧЕНИЕ

В дипломной работе было проведено исследование в области реализации сетевых приложений и работы компьютерных сетей, были изучены протоколы стека TCP/IP, проведена разработка схемы базы данных для приложения, разработка протокола на базе JSON для передачи сообщений между клиентом и сервером, разработка сервера и клиента, разработка пула потоков и пула соединений, разработка таймера для освобождения неиспользованных соединений. В процессе проектирования и разработки использовались проверенные временем концепции и методологии. В программном обеспечении применялись самые актуальные технологии, которые используются в промышленной разработке. Данное приложение обладает потенциалом для дальнейшего развития путем добавления новых функциональных возможностей.

Таким образом, все поставленные задачи были решены, цель дипломной работы достигнута.

В таблице 3 указаны приобретенные компетенции.

Таблица 3 – компетенции

УК-1	Способен осуществлять поиск, критический анализ и синтез информации, применять системный подход для решения поставленных задач	Была найдена необходимая библиотека на C++ для коммуникации с СУБД PostgreSQL
УК-2	Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	Были выбран оптимальный способ подключения клиентов к серверу
УК-3	Способен осуществлять социальное взаимодействие и реализовывать свою роль в команде	Были исполнены роли лидера, разработчика, тестировщика и дизайнера в команде.

УК-4	Способен осуществлять деловую коммуникацию в устной и письменной формах на государственном языке Российской Федерации и иностранном(ых) языке(ах)	Была осуществлена деловая коммуникация с научным руководителем.
УК-5	Способен воспринимать межкультурное разнообразие общества в социально-историческом, этическом и философском контекстах	Проведен поиск по открытым источникам в международных ресурсах.
УК-6	Способен управлять своим временем, выстраивать и реализовывать траекторию саморазвития на основе принципов образования в течение всей жизни	Был продумана дорожная карта проекта и работа была выполнена к сроку.
УК-7	Способен поддерживать должный уровень физической подготовленности для обеспечения полноценной социальной и профессиональной деятельности	Были проведены тренировки для поддержания физической формы во время выполнения выпускной квалификационной работы.
УК-8	Способен создавать и поддерживать безопасные условия жизнедеятельности, в том числе при возникновении чрезвычайных ситуаций	Был проведен инструктаж по технике безопасности во время прохождения учебной практики.
ОПК-1	Способен применять фундаментальные знания, полученные в области математических и (или) естественных наук, и использовать их в профессиональной деятельности	Были исследованы основы компьютерных сетей и стека протоколов TCP/IP.
ОПК-2	Способен использовать и адаптировать существующие методы и системы программирования для разработки и реализации алгоритмов решения	Были использован фреймворк с открытым исходным кодом Qt.

	прикладных задач	
ОПК-3	Способен применять и модифицировать математические модели для решения задач в области профессиональной деятельности	Был проведен анализ возможных решений поставленной задачи, из которых был выбран наилучший вариант для конкретной задачи, с учетом всех преимуществ и недостатков.
ОПК-4	Способен решать задачи профессиональной деятельности с использованием существующих информационно-коммуникационных технологий и с учетом основных требований информационной безопасности	Был произведен поиск по самой актуальной литературе из открытых источников. Были использованы последние технические решения из области информационно-коммуникационных технологий.
ПК-1	Проверка работоспособности и рефакторинг кода программного обеспечения, интеграция программных модулей и компонент и верификация выпусков программного обеспечения	Была написана информационная система с клиент-серверной архитектурой, которая подвергалась неоднократному рефакторингу.
ПК-2	Мониторинг функционирования интеграционного решения в соответствии с трудовым заданием, работа обращениями пользователей по вопросам функционирования интеграционного решения в соответствии с трудовым заданием	Был выбран оптимальный способ реализации клиентской части для обеспечения лучшего пользовательского опыта в приложении.
ПК-3	Проверка и отладка программного кода, тестирование информационных ресурсов с точки зрения логической целостности (корректность ссылок, работа элементов форм)	Приложение было протестировано и отлажено, пользовательский интерфейс отзывается быстро.
ПК-4	Ведение информационных баз данных	Была использована базы данных PostgreSQL.
ПК-5	Обеспечение функционирования	Была использована libpq для лучшего взаимодействия с базой данных. База

	баз данных	данных была развернута в отдельном Docker контейнере.
ПК-6	Педагогическая деятельность по проектированию и реализации общеобразовательных программ	Были изучены последние практики разработки и проектирования программного обеспечения.
ПК-7	Разработка и документирование программных интерфейсов, разработка процедур сборки модулей и компонент программного обеспечения, разработка процедур развертывания и обновления программного обеспечения	Были разработан программный интерфейс. Код закомментирован. Архитектура приложения описана с помощью диаграмм. Было развернуто приложение на удаленном сервере.
ПК-8	Применять методы и средства сборки модулей и компонент программного обеспечения, разработки процедур для развертывания программного обеспечения, миграции и преобразования данных, создания программных интерфейсов	Были продуман программные интерфейсы протокол который описывают взаимодействие между клиентом и сервером. Были изучены методы сборки приложений написанных на C++.
ПК-9	Описание возможной архитектуры развертывания каждого компонента, включая оценку современного состояния предлагаемых архитектур, оценка архитектур с точки зрения надежности правовой поддержки	Развертывание происходит с помощью docker контейнеров. Правила развертывание описаны yaml файле. Образ описан в dockerfile.
ПК-10	Документальное предоставление прослеживаемости требований, согласованности с системными требованиями; приспособленность стандартов и методов проектирования; осуществимость, функционирования и сопровождения; осуществимость программных составных частей	Была использована платформа Miro для создания диаграмм архитектуры приложения. Были использованы последние практики построения приложений.

ПК-11	Техническое сопровождение возможных вариантов архитектуры компонентов, включающее описание вариантов и технико-экономическое обоснование выбранного варианта	Было проведено техническое сопровождение ключевых архитектурных компонентов.
ПК-12	Выполнение работ по созданию (модификации) и сопровождению ИС, автоматизирующих задачи организационного управления и бизнес-процессы	Была продумана информационная система. Были разработаны компоненты информационной системы. Были продуманы бизнес-процессы.
ПК-13	Создание и сопровождение требований и технических заданий на разработку и модернизацию систем и подсистем малого и среднего масштаба и сложности	Были созданы технические задания на разработку и модернизацию системы.
ПК-14	Способность использовать основы экономических знаний в профессиональной деятельности	Был выбраны оптимальный способ интеграции клиентского приложения, который обеспечит лучший пользовательский опыт.
ПК-15	Способность к коммуникации, восприятию информации, умение логически верно, аргументировано и ясно строить устную и письменную речь на русском языке для решения задач профессиональной коммуникации	Развита способность к восприятию сложной технической документации и информации. Развита устная и письменная речь.
ПК-16	Способность находить организационно-управленческие решения в нестандартных ситуациях и готовность нести за них ответственность	Развита способность находить нестандартные решения в различных ситуациях.
ПК-17	Знание своих прав и обязанностей как гражданина своей страны, способностью использовать действующее законодательство и другие правовые документы в своей	Было разработано приложение с целью развития пользовательского опыта в десктопных мессенджерах. Были использованы свободные и законные источники информации.

	профессиональной деятельности, демонстрировать готовность и стремление к совершенствованию и развитию общества на принципах гуманизма, свободы и демократии	

СПИСОК ЛИТЕРАТУРЫ

1. Компьютерные сети. 4-е изд. / Э. Таненбаум. — СПб.: Питер, 2003. — 992 с: ил. — (Серия «Классика computer science»).
2. CMake Documentation [Электронный ресурс]. – Режим доступа: https://cmake.org/documentation/ (Дата обращения 12.05.2022)
3. PostgreSQL Documentation [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/ (Дата обращения 12.05.2022)
4. Qt Documentation [Электронный ресурс]. – Режим доступа: https://doc.qt.io/ (Дата обращения 12.05.2021)
5. Страуструп, Б. Язык программирования C++. Специальное издание / Б. Страуструп. – Издательство: Бином, 2008. – 1104 с
6. Садыков, И. Н. Разработка десктопного клиент - серверного мессенджера на Cpp [Текст] / Ин-т ВМиИТ. – Казань, 2021. – 83 с.