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

Специальность (направление): 01.03.02 – Прикладная математика и информатика

КУРСОВАЯ РАБОТА

Разработка десктопного клиент - серверного мессенджера на Cpp

Работа завершена:

Студент гр. 09-812

"___"_____ 20__ г. _____ И. Н. Садыков

Работа допущена к защите:

Научный руководитель

Доцент КТК

"___"_____ 20__ г. _____ В. Р. Байрашева

Казань – 2021

Содержание

Введение 3
Глава I. Проектирование приложения 5
Основные понятия и концепции, используемые при разработке приложения 5
Язык C++ 6
Фреймворк Qt 7
Сборка проекта с использованием Qt 11
Механизм сигналов и слотов. 14
Архитектура приложения 16
Глава II. Реализация приложения 18
План проекта 18
Структура проекта 20
База данных и взаимодействие с ней 22
Протокол взаимодействия клиента сервера 25
Реализация Сервера 28
Реализация Клиента 32
Заключение 40
Список литературы 41
Листинг 42

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.

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, которые потребляются клиентом. В архитектуре этого типа один или несколько клиентских компьютеров подключены к центральному серверу через сеть или подключение к Интернету. 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, заказчику и получать прибыль. В связи с этим очень выгодно разрабатывать кроссплатформенное программное обеспечение. Выпуская свое приложение на разные платформы и операционные системы, можно увеличить количество пользователей, а также сократить количество затраченных ресурсов на разработку, потому что приложение нужно разработать всего раз. Для таких целей используют интерпретируемые языки которые не завязаны на конкретной платформе и исполняют код на виртуальной машине. Но их производительность не такая хорошая 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:

- Проектирование архитектуры клиент – серверного приложения
- Определение модели данных, протокола на которых будет основан клиент сервер
- Непосредственно реализация приложения в программном коде
- Тестирование

Глава I. Проектирование приложения

Основные понятия и концепции, используемые при разработке приложения

• Клиент - программа, запущенная на машине пользователя. Клиентские программы предоставляют интерфейс, который позволяет пользователю компьютера запрашивать службы сервера и отображать результаты, возвращаемые сервером.

• Сервер – программа, которая чаще всего находится на удаленной машине, но может быть и на локальной, ждет запросов от клиентов, а затем отвечает на них. В идеале сервер предоставляет клиентам стандартизированный прозрачный интерфейс, чтобы клиенты не знали о специфике системы (то есть аппаратного и программного обеспечения), предоставляющей услугу.

• Параллелизм - возможность разбить программу на части, которые могут выполняться независимо друг от друга. Это означает, что задачи могут выполняться не по порядку, а результат будет таким, как если бы они выполнялись по порядку.

• Кросс-платформенная разработка приложений - это создание одного приложения, которое может работать в разных операционных системах, вместо разработки разных версий приложения для каждой платформы. Кросс-платформенность 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++ - один из самых популярных языков программирования в мире, его можно найти 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 можно адаптировать к нескольким платформам [5].

Фреймворк Qt

Qt - 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), используется 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. Фреймоворк имеет обширную поддержку интернационализации. Qt также предоставляет Qt Quick, который включает декларативный язык сценариев под названием QML, который позволяет использовать JavaScript для обеспечения логики [4]. Благодаря Qt Quick стала возможной быстрая разработка приложений для мобильных устройств, в то время как логика по-прежнему может быть написана с использованием собственного кода для достижения максимально возможной производительности [4].

Другие функции включают доступ к базе данных SQL, синтаксический анализ XML, анализ JSON, управление потоками и поддержку сети [4].

Все классы в Qt наследуются от класса QObject (рисунок 1)

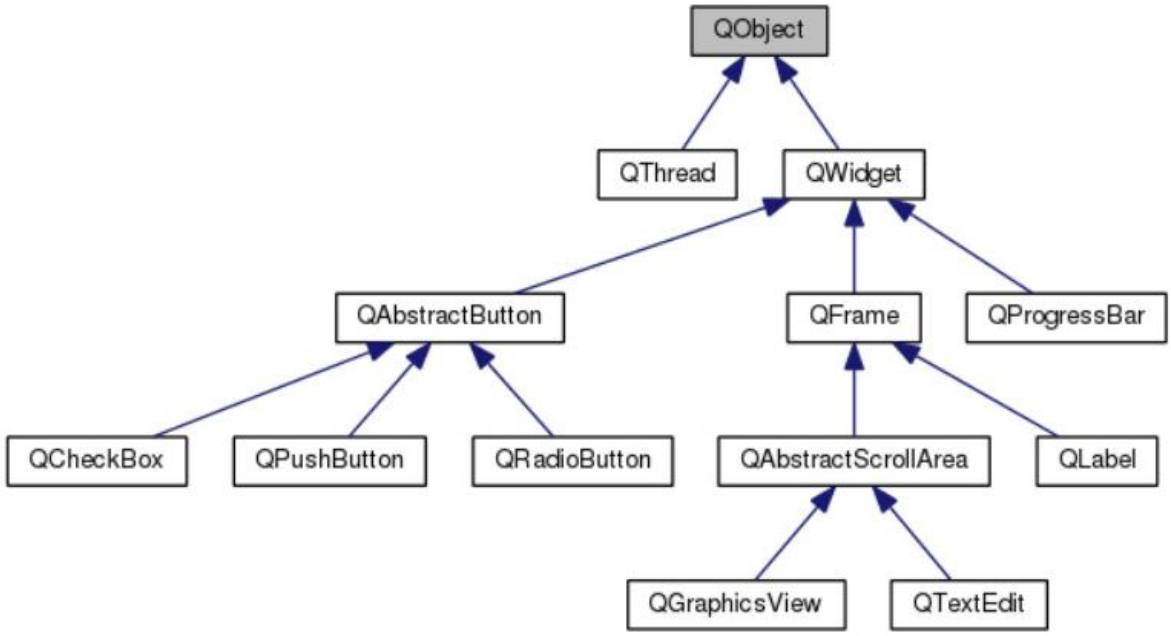

Рисунок 1 – Диаграмма наследования классов в Qt

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.

Продолжение таблицы 1

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.
Qt WebEngine	Набор Qt Widget и QML webview API на основе Chromium.
Qt Test	Классы для модульного тестирования приложений и библиотек Qt.

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 бесплатные для разработки open-source программного обеспечения. 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 систему сборки C++ с системой сборки Qt, видно, что система сборки C++ (серые прямоугольники) осталась неизменной (рисунок 2). Мы все еще пишем код на C++. Однако мы добавляем больше источников и заголовков (зеленые прямоугольники) (рисунок 2). Здесь задействованы три генератора кода:

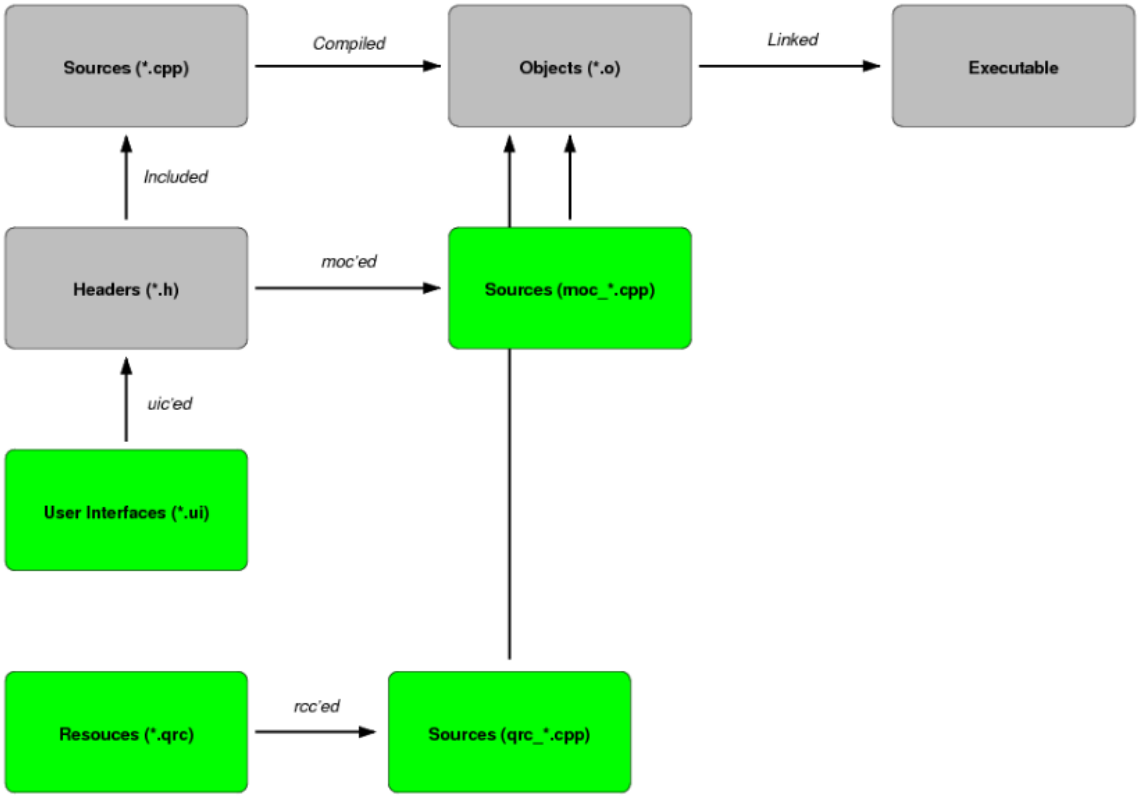

Рисунок 2 – Диаграмма этапов сборки в Qt

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 - это то, что 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 - центральная черта Qt и 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 (рисунок 3). Сигнал излучается, когда происходит некоторое событие. Слот - это функция, которая вызывается в ответ на определенный сигнал.

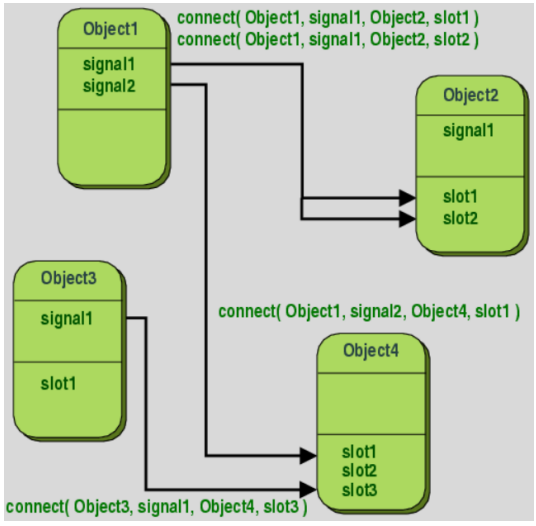

Рисунок 3 – Пример взаимодействия сигналов и слотов

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 (рисунок 4), хранящий информацию о слотах, которые необходимо вызвать по излучению сигнала. С помощью такой реализации есть возможность соединять сигналы и слоты во время исполнения программы, что дает большую гибкость приложению.

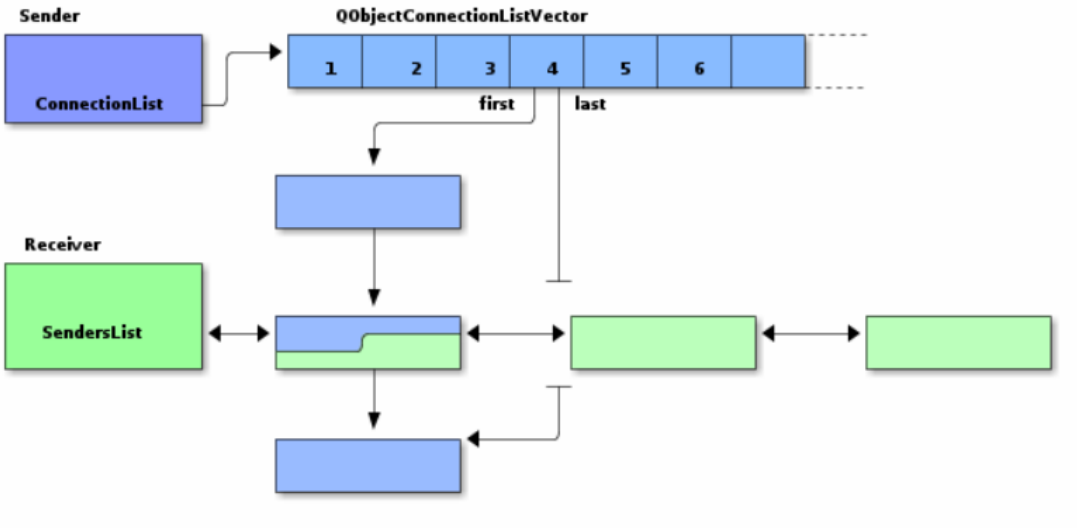

Рисунок 4 – Иллюстрация двусвязного списка хранящего сигналы и слоты

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, а также права доступа и распределение ресурсов 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 5.

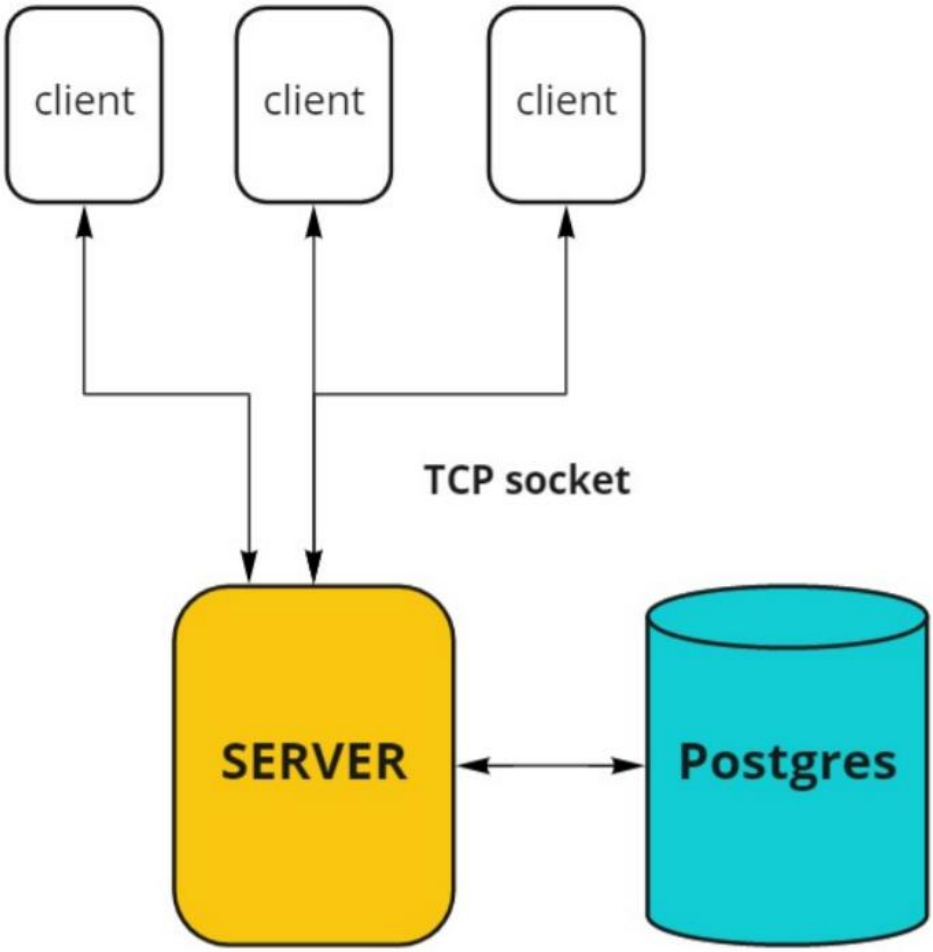

Рисунок 5 – Диаграмма компонентов системы

Сервер работает на удаленной машине и взаимодействует с СУБД PostgreSQL для работы с данными. Клиенты могут исполняться на разных машинах так и на одной. Клиенты не общаются с друг другом на прямую, 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).
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 базовой части клиента.
11. Разработка графического интерфейса клиента.
12. Переход на SSL сокеты для безопасной передачи пакетов в сети.
13. 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 иметься возможность присоединения к каналу с помощью пароля и отправка сообщений в канал. Передача данных должна быть зашифрована.

Серверная часть должна состоять из основного консольного приложения и СУБД PostgreSQL, в которой хранятся данные о канале, 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 6.

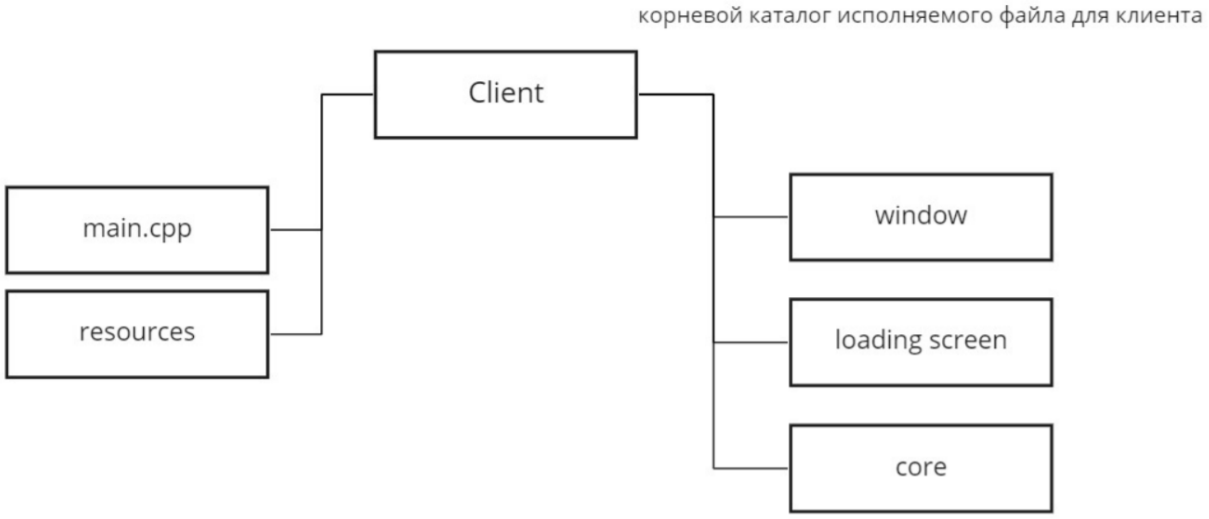

Рисунок 6 - Файловая структура клиентской части

Файловая структура серверной части представлена на рисунке 7.

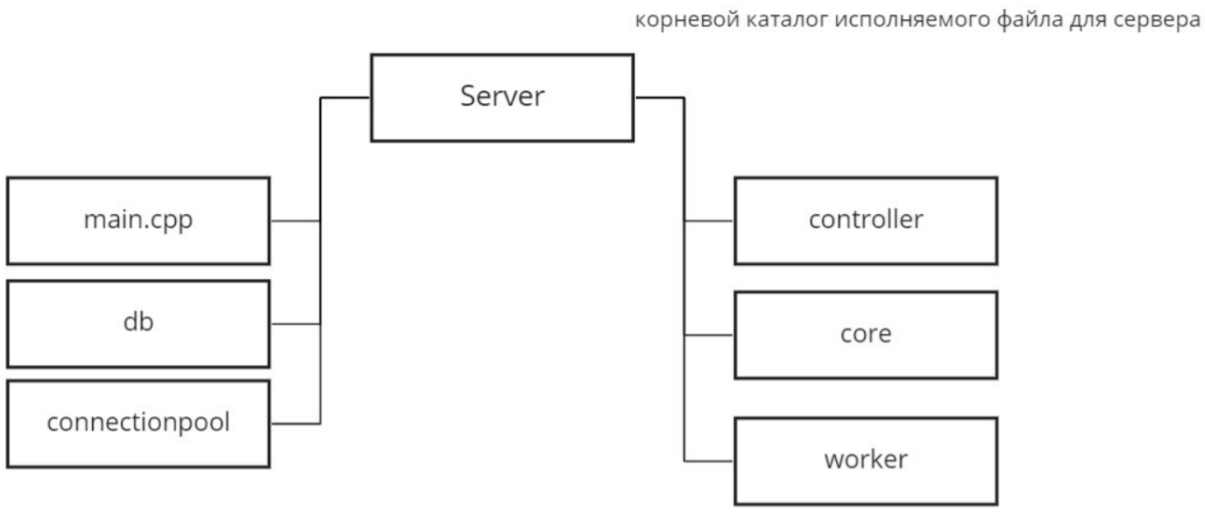

Рисунок 7 - Файловая структура серверной части

Взаимодействие компонентов изображено на рисунке 8.

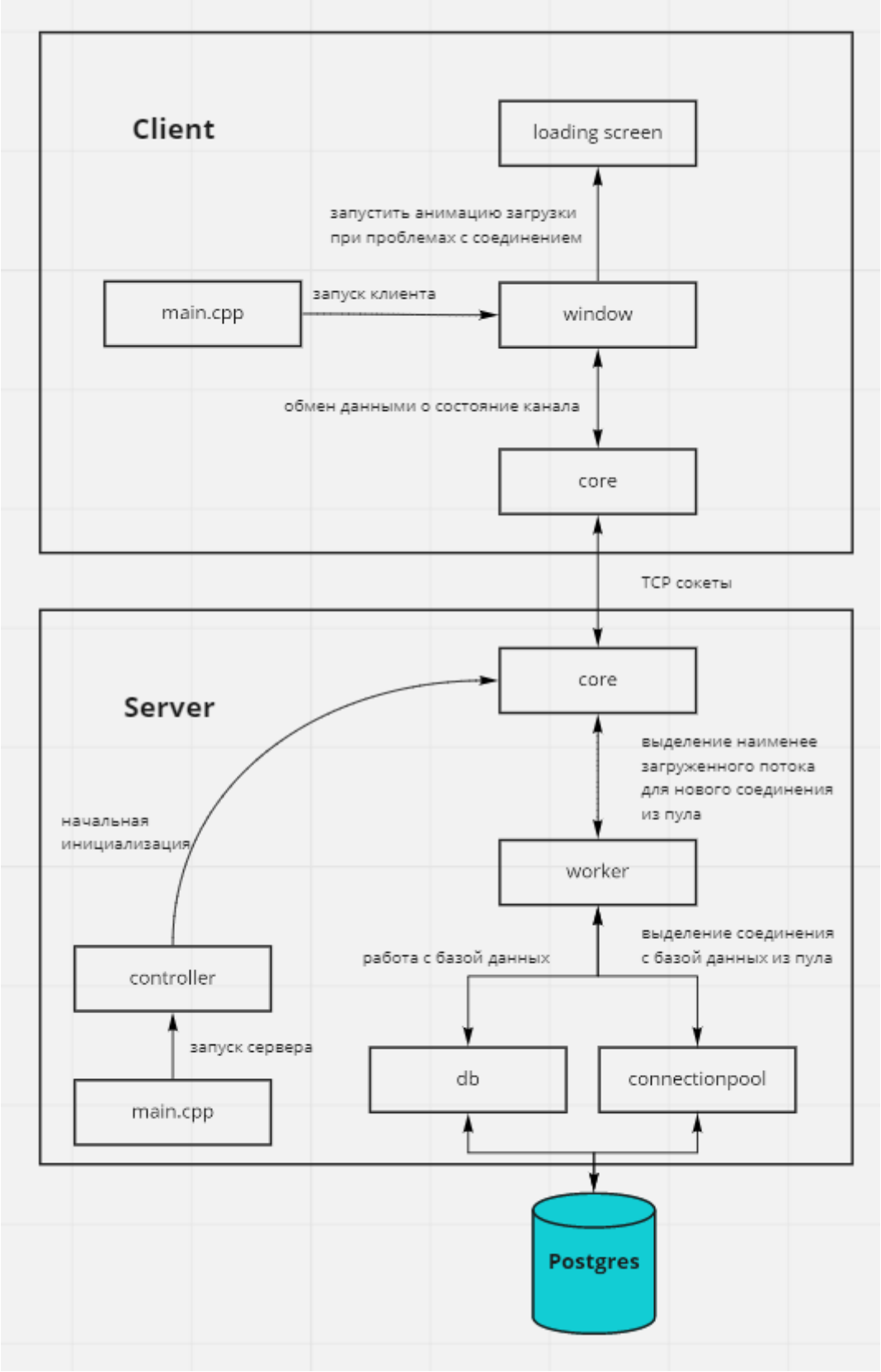

Рисунок 8 - Диаграмма потоков данных

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 9.

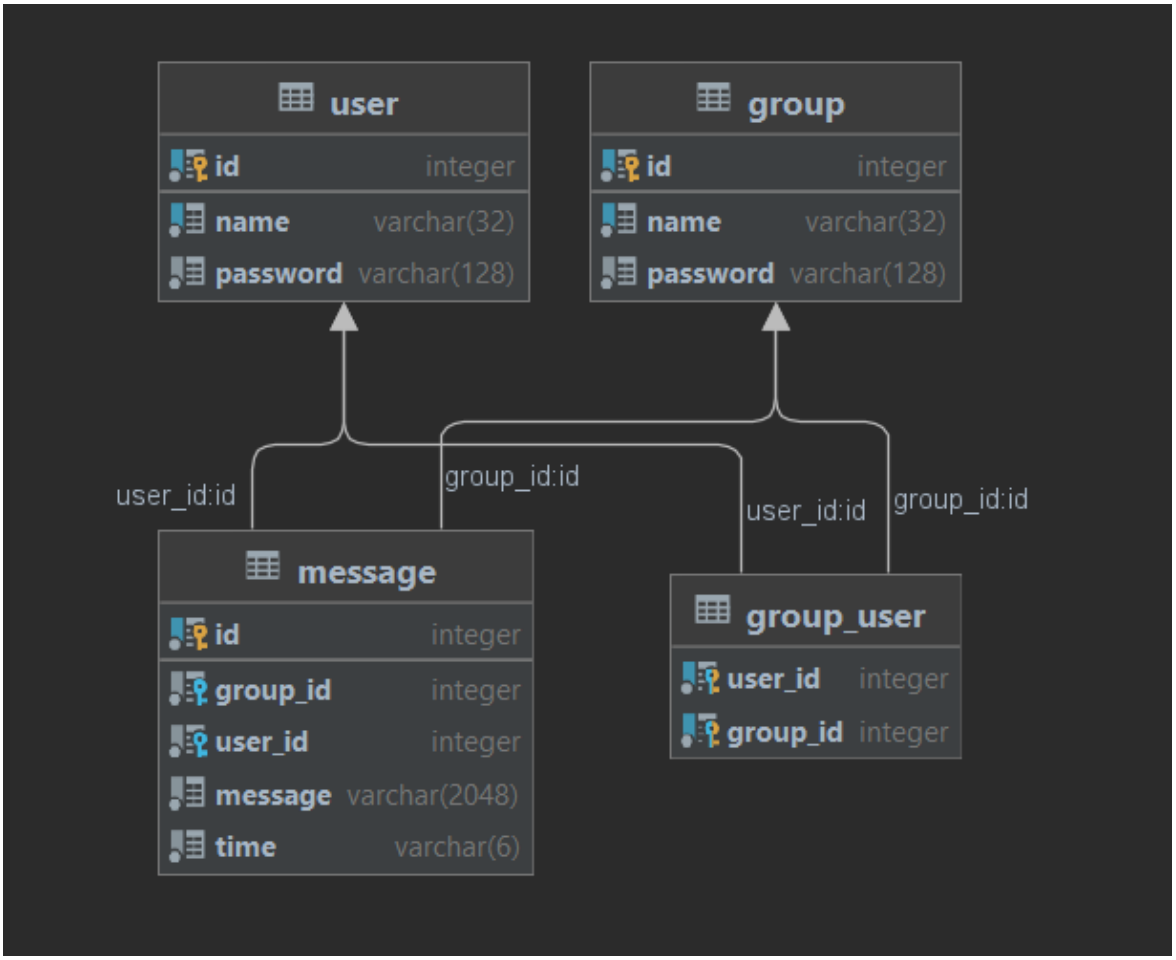

Рисунок 9 - Схема БД

Для эффективной работы с базой данных на стороне сервера был создан класс пула соединений (рисунок 10).

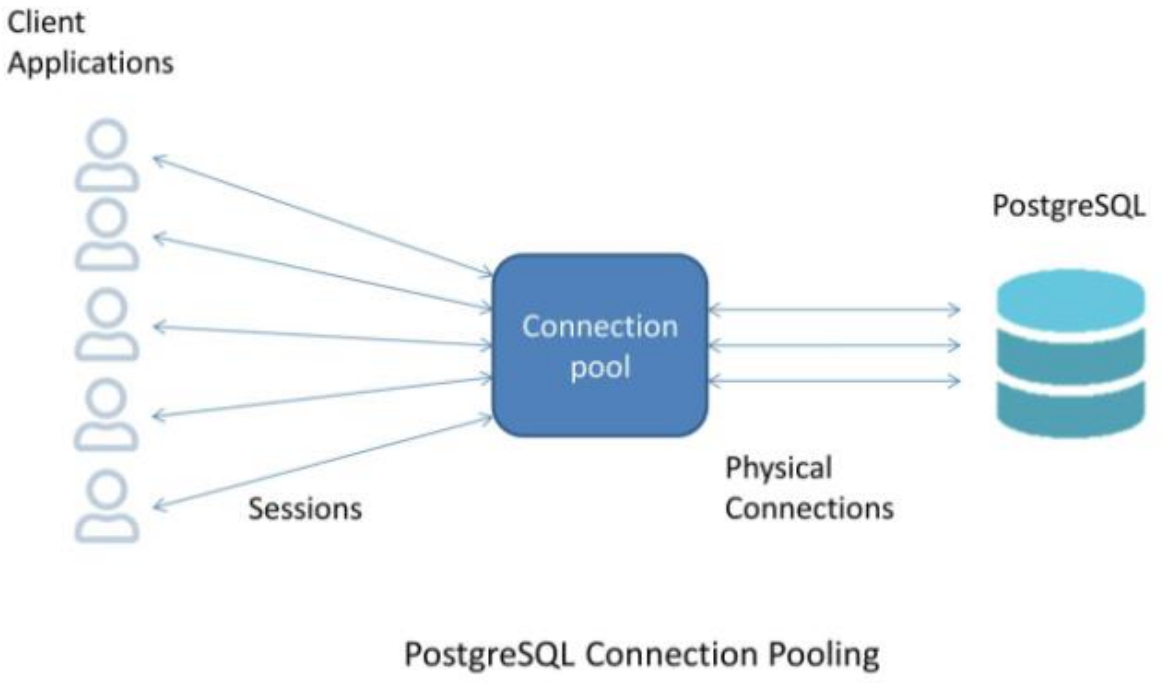

Рисунок 10 - Схема работы с пулом соединений

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, должен будет ждать определенное количество времени, чтобы

попытаться продолжить получение, и не будет возвращать недопустимое соединение, пока не истечет время ожидания

- Легкое закрытие соединений.

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 добавления сообщения в базу данных (рисунок 11).

```
void db::addMessage(const Message& message)
{
    auto conn = ConnectionPool::getConnection();
    QSqlQuery query(conn);
    query.prepare(R"(insert into message (group_id, sender_name, message, time)
values (1, :sender_name, :message, :time))");
    query.bindValue(":sender_name", message.getSender());
    query.bindValue(":message", message.getMessage());
    query.bindValue(":time", message.getTime());
    query.exec();
    ConnectionPool::releaseConnection(conn);
}
```

Рисунок 11 - Метод addMessage

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 - 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 (рисунок 12).

```
namespace Packet {
    namespace Type {
        constexpr const char* const TYPE          = "type";
        constexpr const char* const LOGIN         = "login";
        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 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 12 - Типы сообщений

Пример формирования JSON сообщения и отправки его всем пользователям в группе (рисунок 13).

```
QJsonObject broadcastPacket;
broadcastPacket[Packet::Type::TYPE]   = Packet::Type::MESSAGE;
broadcastPacket[Packet::Data::SENDER] = sender->getUserName();
broadcastPacket[Packet::Data::TEXT]   = text;
broadcastPacket[Packet::Data::TIME]   = time;
broadcast(broadcastPacket, sender);
```

Рисунок 13 - Код формирования сообщения

После формирования сообщения оно сериализуется и отправляется по сети через TCP сокет (рисунок 14).

```
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 14 - Метод sendMessage

Парсинг структуры рассмотренной выше (рисунок 15).

```
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 15 - Метод handleMessagePacket

Структура выше отображается в класс Message (рисунок 16), 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 16 - Класс Message

Реализация Сервера

Основным классом является класс ServerCore (рисунок 17), 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 QJsonObject& packet, 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 packetFromLoggedOut(ServerWorker* sender, const QJsonObject& packet);
    void packetFromLoggedIn(ServerWorker* sender, const QJsonObject& packet);
    QJsonArray getUsernames(ServerWorker* exclude) const;
    static QJsonArray getMessages() ;
    static void sendPacket(ServerWorker* destination, const QJsonObject& packet);
    static bool isEqualPacketType(const QJsonValue& jsonType, const char* strType);
```

Рисунок 17 - Класс ServerCore

Запускается основной модуль из объекта класса ServerController (рисунок 18).

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

private:
    ServerCore* serverCore;
};
```

Рисунок 18 - Класс ServerCore

Основная работа ServerCore это обработка событий от клиентов (соединение, прием сообщения).

В классе содержится пул потоков *threads* (рисунок 19) для присоединяющихся клиентов. Как только присоединяется новый клиент под него выделяется наименее загруженный поток из пула. При старте пул пустой - потоки не выделены, потом, при присоединении новых клиентов, создаются новые потоки до ограничения *threadLoadFactor*, который задается с помощью функции *QThread::idealThreadCount.*

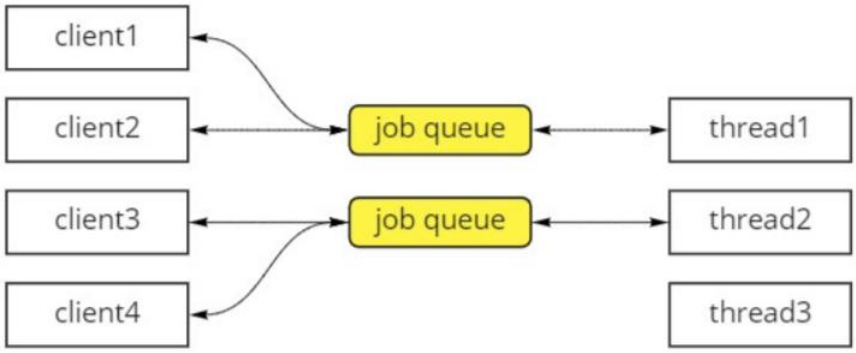

Рисунок 19 - Схема работы пула потоков

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

Объявление класса ServerWorker (рисунок 20).

```
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);
    void sendPacket(const QJsonObject& packet);
public slots:
    void disconnectFromClient();
private slots:
    void onReadyRead();
signals:
    void packetReceived(const QJsonObject& packet);
    void disconnectedFromClient();
    void error();

private:
    QSslSocket* serverSocket;
    QString userName;
    mutable QReadWriteLock userNameLock;
};
```

Рисунок 20 - Класс ServerWorker

Прослушивание входящих пакетов в виде JSON структуры происходит в методе onReadyRead (рисунок 21).

```
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 21 - 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 (рисунок 22) наследуется от класса QWidget и имеет графический интерфейс, который реализован в ui формате, который компилируется в h файл с помощью компилятора uic.

```
namespace Ui {
    class ClientWindow;
}

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;
    LoadingScreen* loadingScreen;
    bool logged;
    static constexpr int minWindowWidth    = 750;
    static constexpr int minWindowHeight   = 500;
    static constexpr int maxMessageRowSize = 50;
    static constexpr int maxMessageSize    = 2048;
    static constexpr int minUserNameSize   = 1;
    static constexpr int maxUserNameSize   = 32;
    static constexpr int minPasswordSize   = 4;
    static constexpr int maxPasswordSize   = 32;
```

Рисунок 22 - Класс ClientWindow

Ui файл (рисунок 23) 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 23 - Ui код

Графический интерфейс, который получается с помощью такого файла представлен на рисунке 24.

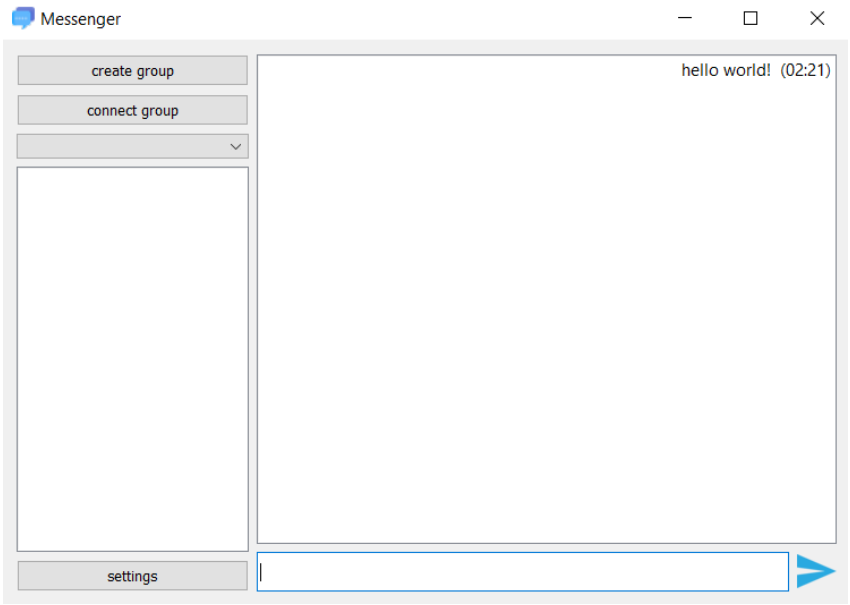

Рисунок 24 - Главное окно приложения

Класс ClientCore (рисунок 25) 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;
public slots:
    void connectToServer(const QHostAddress& address, quint16 port);
    void login(const QString& username, const QString& password);
    void sendMessage(const QString& message, const QString& time);
    void disconnectFromHost();
private slots:
    void onReadyRead();
signals:
    void connected();
    void disconnected();
    void loggedIn();
    void loginError(const QString& reason);
    void messageReceived(const Message& message);
    void error(QAbstractSocket::SocketError socketError);
    void userJoined(const QString& username);
    void userLeft(const QString& username);
    void informJoiner(const QStringList& usernames, const QList<Message>& messages);

private:
    QSslSocket* clientSocket;
    QString name;
```

Рисунок 25 - Класс ClientCore

Продемонстрируем работу приложения:

Если нет соединения с сервером, клиент будет пытаться присоединиться к серверу, и будет гореть окно с анимацией загрузки (рисунок 26).

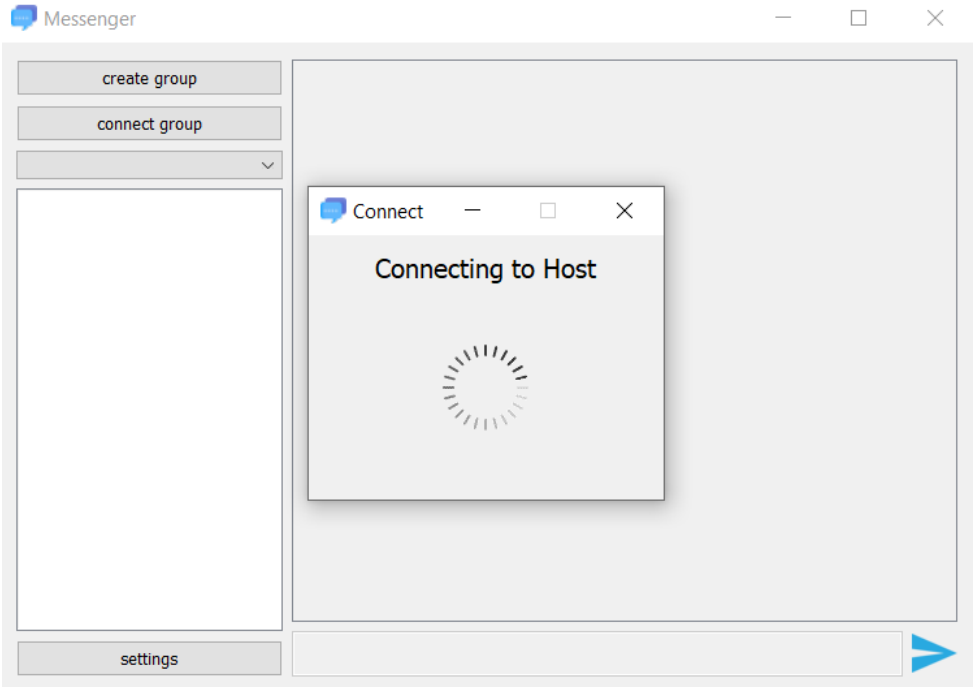

Рисунок 26 - Окно загрузки

После того как соединение прошло, нужно указать свое имя и пароль группы (рисунок 27).

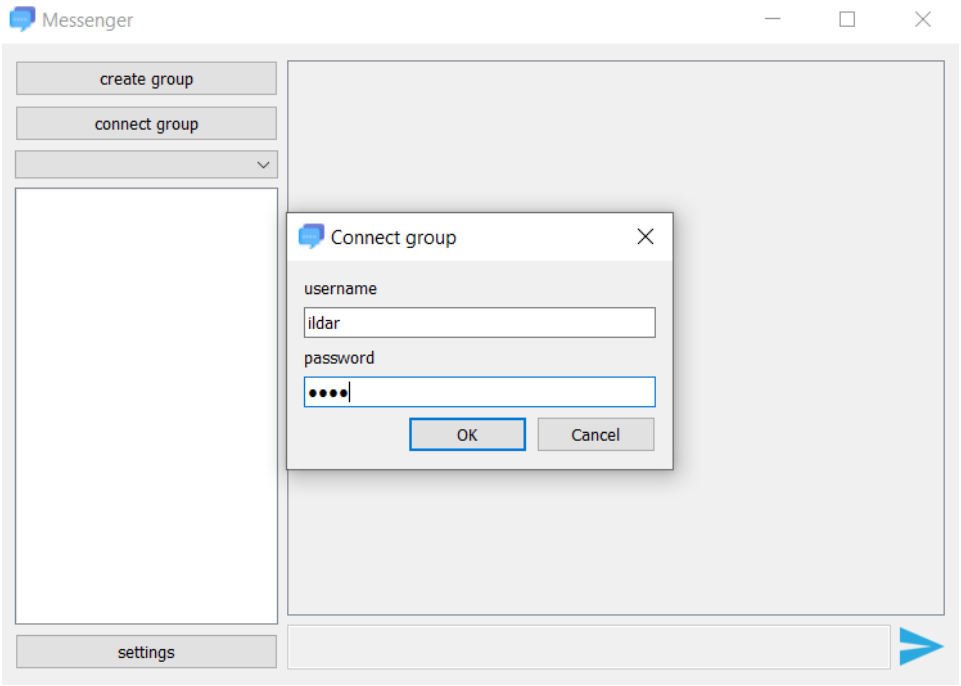

Рисунок 27 - Окно авторизации

Если пароль неверный, приложение предупредит об этом пользователя и предложит ввести реквизиты повторно (рисунок 28).

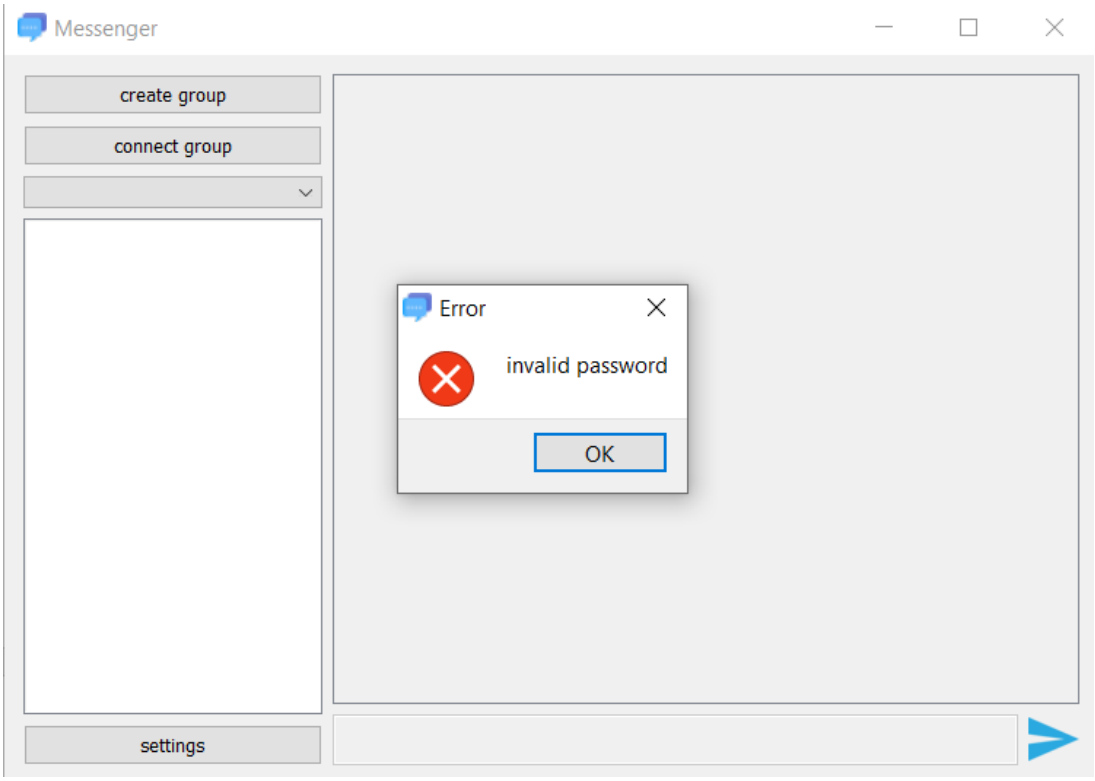

Рисунок 28 - Окно ошибки

Если пароль верный, у нас открывается доступ к чату (рисунок 29).

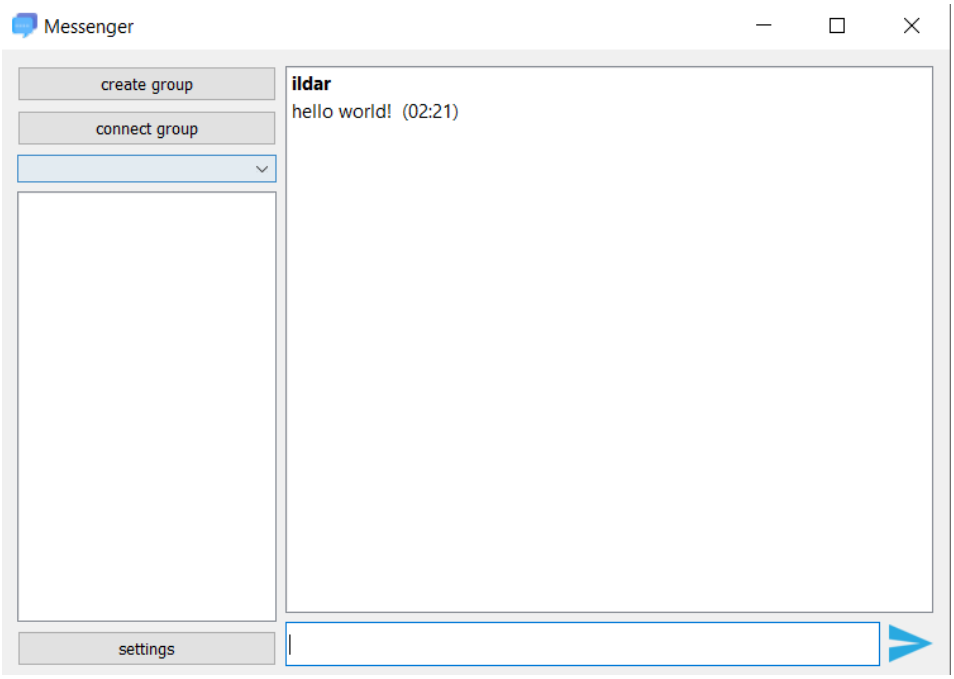

Рисунок 29 - Окно с чатом

Откроем еще несколько клиентов для симуляции активного общения.

При присоединении нового пользователя, имя пользователя, который онлайн, появляется в боковом виджете. Также в чате видно, что присоединился новый пользователь (рисунок 30).

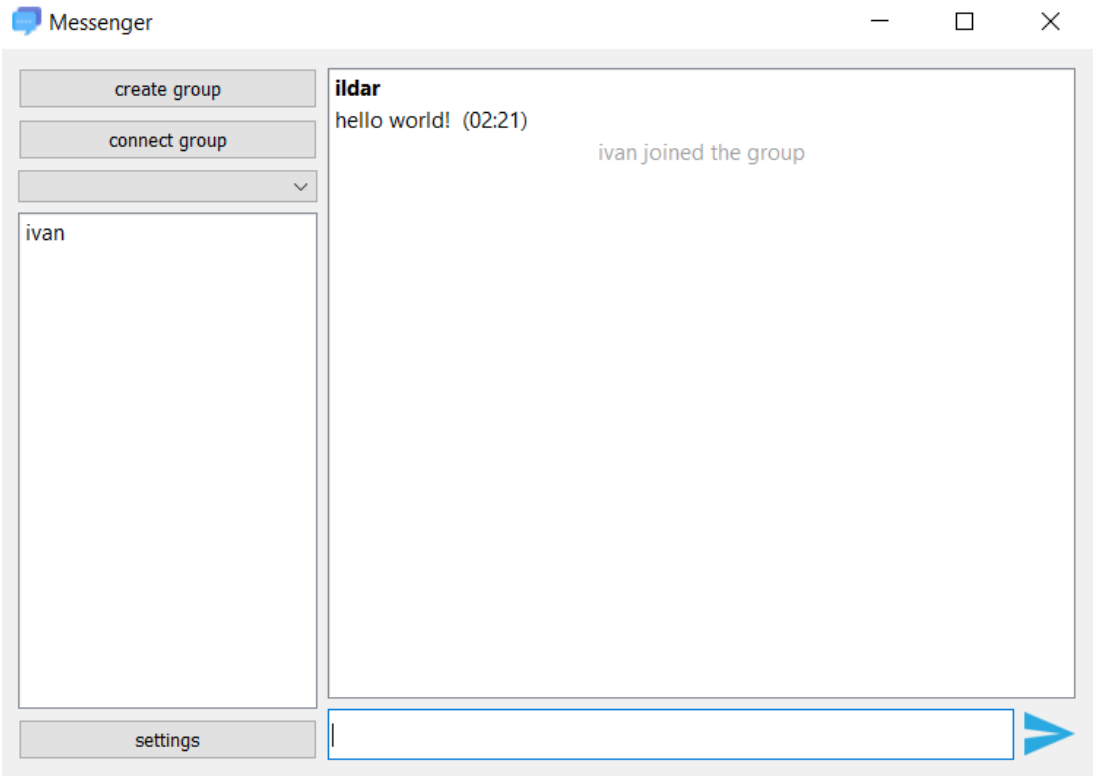

Рисунок 30 - Окно с чатом

Общение нескольких клиентов продемонстрировано на рисунке 31.

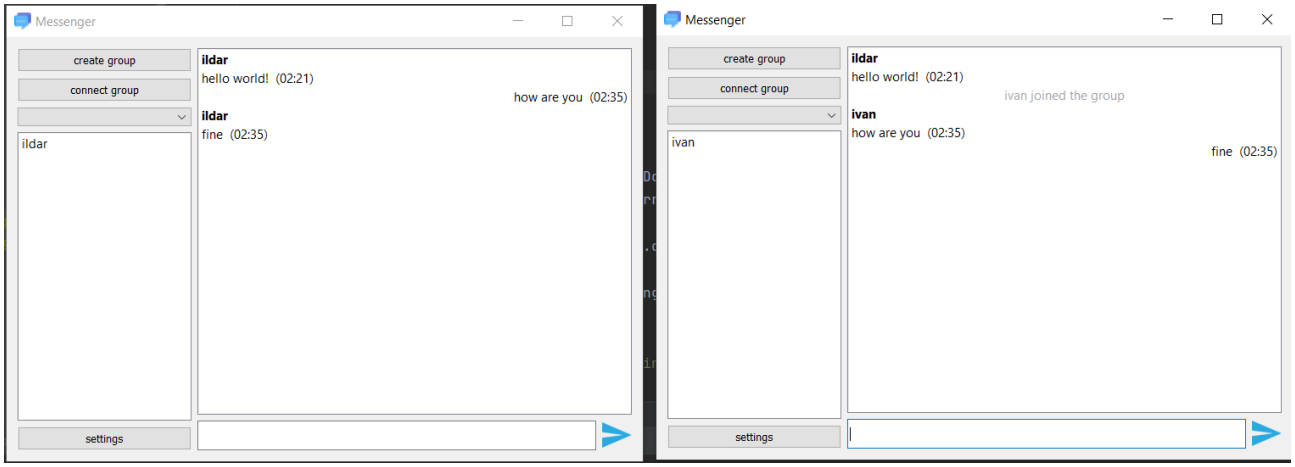

Рисунок 31 - Общение нескольких клиентов

Общение большого количества клиентов продемонстрировано на рисунке 32.

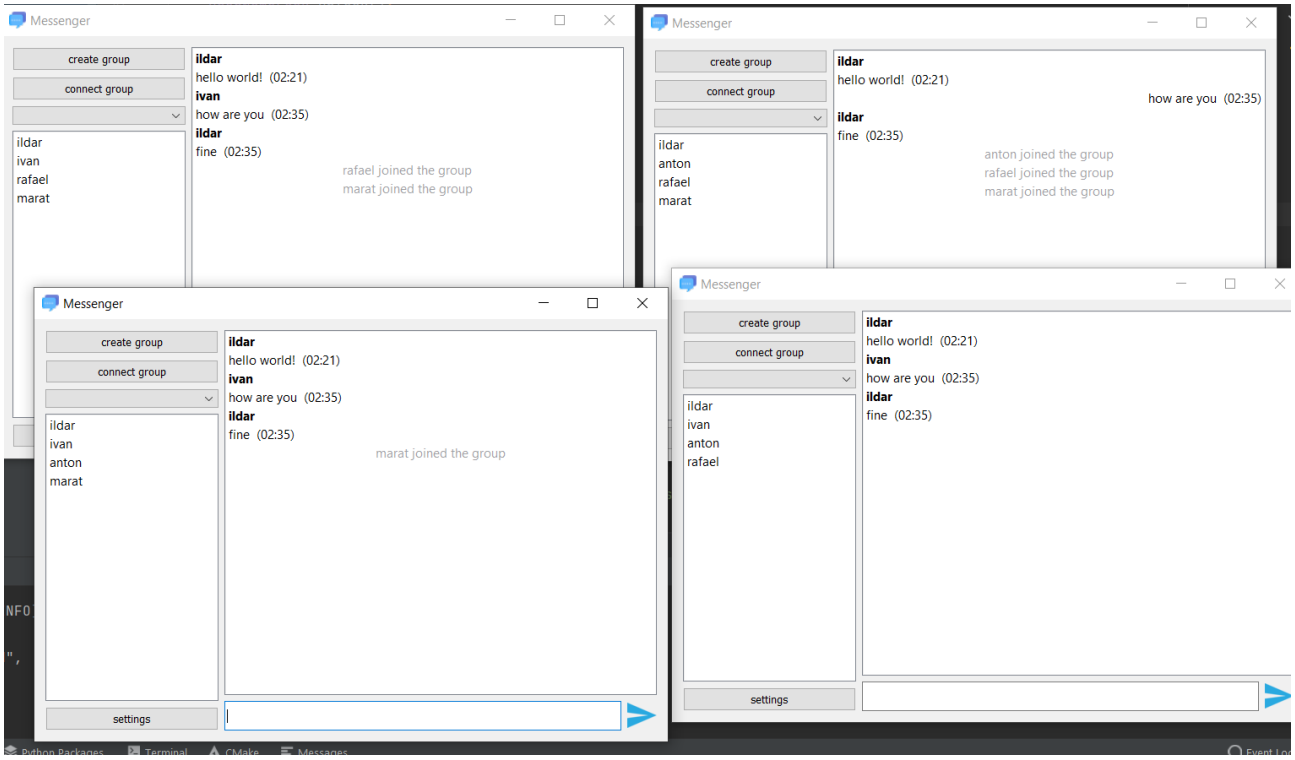

Рисунок 32 - Общение множества клиентов

При отсоединении пользователя его имя исчезает с боковой панели других клиентов. Также в чате появляется сообщение о данном событии (рисунок 33).

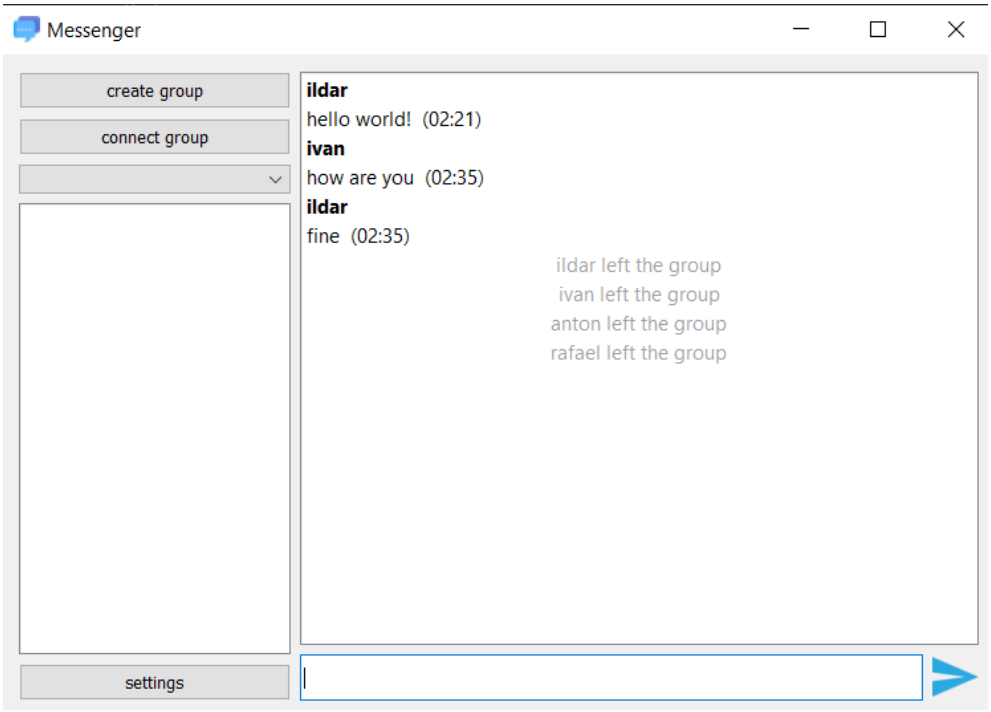

Рисунок 33 - Окно чата

При заходе клиента в чат у него появляется вся история прошлых сообщений (рисунок 34).

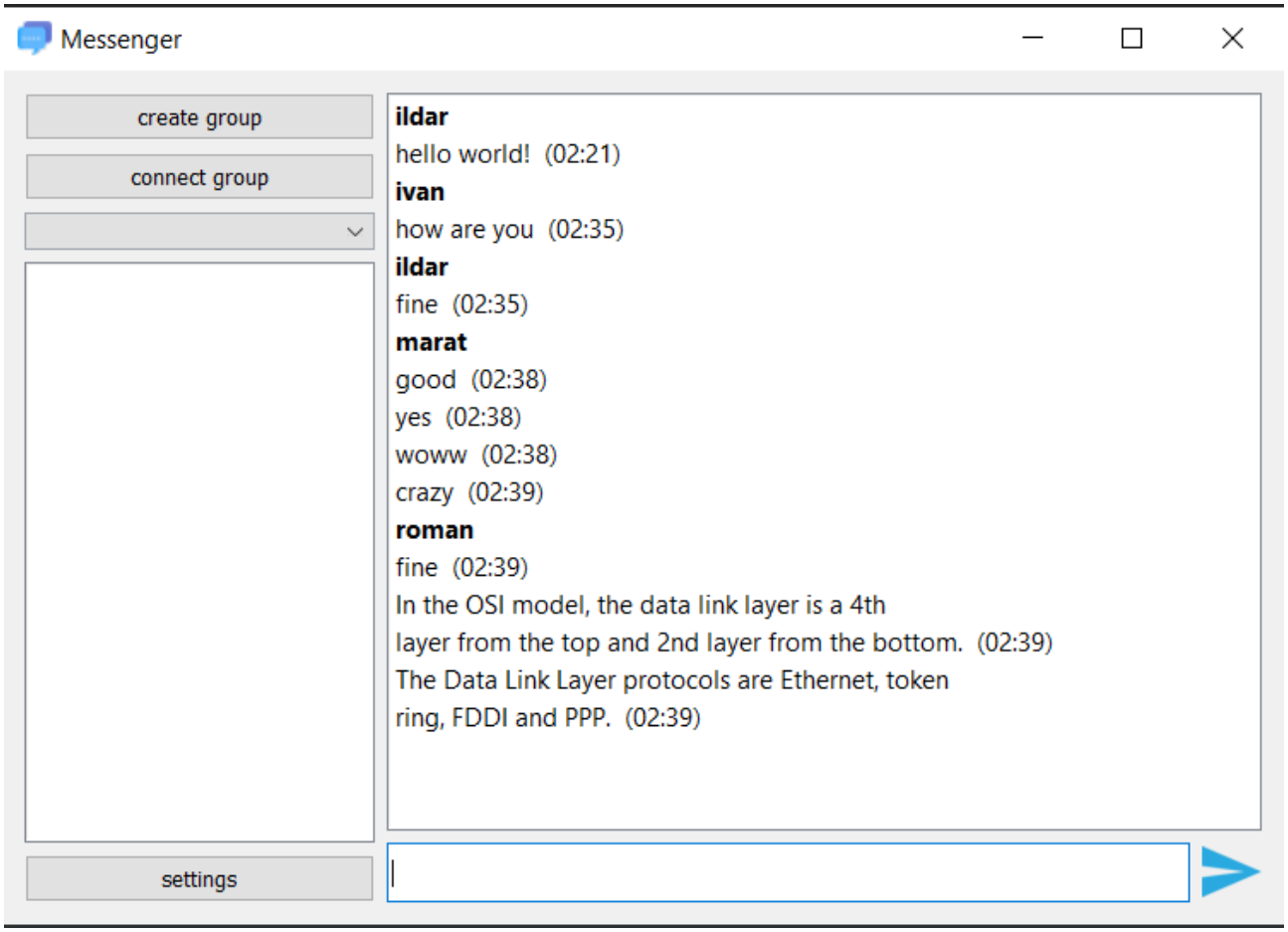

Рисунок 34 - Окно чата

Заключение

В курсовой работе было сделано: исследование в области реализации сетевых приложений, 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. В процессе проектирования и разработки использовались проверенные временем концепции и методологии. Данную программу можно развивать и дальше, добавив новые функциональные возможности.

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

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 20.12.2021)
3. PostgreSQL Documentation [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/ (Дата обращения 20.12.2021)
4. Qt Documentation [Электронный ресурс]. – Режим доступа: https://doc.qt.io/ (Дата обращения 20.12.2021)
5. Страуструп, Б. Язык программирования C++. Специальное издание / Б. Страуструп. – Издательство: Бином, 2008. – 1104 с

Листинг

clientwindow.cpp

```
#include <QStandardItemModel>
#include <QInputDialog>
#include <QMessageBox>
#include <QFormLayout>
#include <QLabel>
#include <QDialogButtonBox>
#include <QHostAddress>
#include <QDateTime>
#include <QTimer>
#include "ui_window.h"
#include "clientwindow.h"
#include "constants.h"


ClientWindow::ClientWindow(QWidget* parent)
    : QWidget(parent), ui(new Ui::ClientWindow), clientCore(new
ClientCore(this)),
      chatModel(new QStandardItemModel(this)), loadingScreen(new
LoadingScreen), logged(false)
{
    ui->setupUi(this);
    setSizePolicy(QSizePolicy::Minimum, QSizePolicy::Minimum);
    setMinimumSize(minWindowWidth, minWindowHeight);


    chatModel->insertColumn(0);
    ui->chatView->setModel(chatModel);
    ui->users->setFocusPolicy(Qt::NoFocus);


    // connect for handle signals from logic view of client
    connect(clientCore, &ClientCore::connected, loadingScreen,
&LoadingScreen::close);
```

```
    connect(clientCore, &ClientCore::connected, this,
&ClientWindow::connected);

    connect(clientCore, &ClientCore::loggedIn, this,
&ClientWindow::loggedIn);

    connect(clientCore, &ClientCore::loginError, this,
&ClientWindow::loginError);

    connect(clientCore, &ClientCore::messageReceived, this,
&ClientWindow::messageReceived);

    connect(clientCore, &ClientCore::disconnected, this,
&ClientWindow::disconnected);

    connect(clientCore, &ClientCore::error, this,
&ClientWindow::error);

    connect(clientCore, &ClientCore::userJoined, this,
&ClientWindow::userJoined);

    connect(clientCore, &ClientCore::userLeft, this,
&ClientWindow::userLeft);

    connect(clientCore, &ClientCore::informJoiner, this,
&ClientWindow::informJoiner);

    // connect for send message

    connect(ui->sendButton, &QPushButton::clicked, this,
&ClientWindow::sendMessage);

    connect(ui->messageEdit, &QLineEdit::returnPressed, this,
&ClientWindow::sendMessage);


    QTimer::singleShot(100, this, [this]() { this-
>attemptConnection(); });

}


ClientWindow::~ClientWindow()

{

    delete ui;

    delete clientCore;

    delete chatModel;

    delete loadingScreen;

}


QPair<QString, QString> ClientWindow::getConnectionCredentials()

{
```

```
    QDialog dialog(this);
    dialog.setWindowFlags(dialog.windowFlags() &
~Qt::WindowContextHelpButtonHint & ~Qt::WindowMaximizeButtonHint);
    dialog.setWindowTitle("Connect group");
    dialog.resize(300, 100);
    QFormLayout form(&dialog);

    QLabel usernameLabel("username");
    form.addRow(&usernameLabel);
    QLineEdit usernameLineEdit(&dialog);
    form.addRow(&usernameLineEdit);

    QLabel passwordLabel("password");
    form.addRow(&passwordLabel);
    QLineEdit passwordLineEdit(&dialog);
    passwordLineEdit.setEchoMode(QLineEdit::Password);
    form.addRow(&passwordLineEdit);

    QDialogButtonBox buttonBox(QDialogButtonBox::Ok |
QDialogButtonBox::Cancel, Qt::Horizontal, &dialog);
    form.addRow(&buttonBox);

    connect(&buttonBox, &QDialogButtonBox::accepted, &dialog,
&QDialog::accept);
    connect(&buttonBox, &QDialogButtonBox::rejected, &dialog,
&QDialog::reject);
    connect(clientCore, &ClientCore::disconnected, &dialog,
&QDialog::close);

    if (dialog.exec() == QDialog::Accepted) {
        return {usernameLineEdit.text(), passwordLineEdit.text()};
    }
    return {};
}
```

```
void ClientWindow::attemptConnection()
{
    loadingScreen->show();
    clientCore->connectToServer(QHostAddress(HOST), PORT);
}

void ClientWindow::connected()
{
    QPair<QString, QString> credentials;
    bool usernameOk;
    bool passwordOk;
    do {
        credentials             = getConnectionCredentials();
        const QString& username = credentials.first;
        const QString& password = credentials.second;
        usernameOk =
                (!username.isEmpty() && (username.size() >=
minUserNameSize && username.size() <= maxUserNameSize));
        passwordOk =
                (!password.isEmpty() && (password.size() >=
minPasswordSize && password.size() <= maxPasswordSize));
        if (!usernameOk) {
            QMessageBox::information(this, tr("username error"),
                                     tr("min %1 characters\nmax %2
characters")

.arg(QString::number(minUserNameSize),
QString::number(maxUserNameSize)));
            continue;
        }
        if (!passwordOk) {
            QMessageBox::information(this, tr("password error"),
                                     tr("min %1 characters\nmax %2
characters")
```

```
.arg(QString::number(minPasswordSize),
QString::number(maxPasswordSize)));
            continue;
        }
        attemptLogin(username, password);
        credentials.first.clear();
        credentials.second.clear();
        return;
    } while (true);
}

void ClientWindow::attemptLogin(const QString& username, const
QString& password)
{
    clientCore->login(username, password);
}

void ClientWindow::loggedIn()
{
    ui->sendButton->setEnabled(true);
    ui->messageEdit->setEnabled(true);
    ui->chatView->setEnabled(true);
    QTimer::singleShot(0, ui->messageEdit, SLOT(setFocus()));
    lastUserName.clear();
    logged = true;
}

void ClientWindow::loginError(const QString& reason)
{
    QMessageBox::critical(this, tr("Error"), reason);
    connected();
}
```

```
QStringList ClientWindow::splitString(const QString& str, const
int rowSize)
{
    QString temp = str;
    QStringList list;
    list.reserve(temp.size() / rowSize + 1);

    while (!temp.isEmpty()) {
        list.append(temp.left(rowSize).trimmed());
        temp.remove(0, rowSize);
    }
    return list;
}

QStringList ClientWindow::splitText(const QString& text)
{
    const QStringList words = text.split(QRegExp("[\r\n\t ]+"),
QString::SkipEmptyParts);
    const int wordsCount    = words.size();
    QStringList rows;
    rows.append("");
    for (int i = 0, j = 0; i < wordsCount; ++i) {
        if (words[i].size() > maxMessageRowSize - rows[j].size())
{
            if (words[i].size() > maxMessageRowSize) {
                QStringList bigWords = splitString(words[i],
maxMessageRowSize);
                for (const auto& bigWord : bigWords) {
                    if (rows[j].isEmpty()) {
                        rows[j] += bigWord + QString(" ");
                    }
                    rows.append(bigWord + QString(" "));
                    ++j;
                }
            } else {
```

```
                rows.append(words[i] + QString(" "));
                ++j;
            }
        } else {
            rows[j] += words[i] + QString(" ");
        }
    }
    return rows;
}

void ClientWindow::displayMessage(const QString& message, const
QString& time, const int lastRowNumber,
                                  const int alignMask)
{
    QStringList rows    = splitText(message);
    const int rowsCount = rows.size();
    int currentRow      = lastRowNumber;
    for (int i = 0; i < rowsCount; ++i) {
        chatModel->insertRow(currentRow);
        if (i == rowsCount - 1) {
            chatModel->setData(chatModel->index(currentRow, 0),
rows[i] + QString(" (") + time + QString(")"));
        } else {
            chatModel->setData(chatModel->index(currentRow, 0),
rows[i]);
        }
        chatModel->setData(chatModel->index(currentRow, 0),
int(alignMask | Qt::AlignVCenter), Qt::TextAlignmentRole);
        ui->chatView->scrollToBottom();
        ++currentRow;
    }
}

void ClientWindow::messageReceived(const Message& message)
{
```

```
    int currentRow = chatModel->rowCount();
    if (lastUserName != message.getSender()) {
        lastUserName = message.getSender();

        QFont boldFont;
        boldFont.setBold(true);

        chatModel->insertRow(currentRow);
        chatModel->setData(chatModel->index(currentRow, 0),
message.getSender());
        chatModel->setData(chatModel->index(currentRow, 0),
int(Qt::AlignLeft | Qt::AlignVCenter),
                           Qt::TextAlignmentRole);
        chatModel->setData(chatModel->index(currentRow, 0),
boldFont, Qt::FontRole);
        ++currentRow;
    }
    displayMessage(message.getMessage(), message.getTime(),
currentRow, Qt::AlignLeft);
}

void ClientWindow::sendMessage()
{
    const QString message = ui->messageEdit->text();
    if (message.isEmpty() || message.size() > maxMessageSize) {
        return;
    }
    const QString time =
QDateTime::currentDateTime().toString("hh:mm");
    clientCore->sendMessage(message, time);

    int currentRow = chatModel->rowCount();
    displayMessage(message, time, currentRow, Qt::AlignRight);

    ui->messageEdit->clear();
```

```
    ui->chatView->scrollToBottom();
    lastUserName.clear();
}

void ClientWindow::disconnected()
{
    qWarning() << "The host terminated the connection";

    ui->sendButton->setEnabled(false);
    ui->messageEdit->setEnabled(false);
    ui->chatView->setEnabled(false);
    lastUserName.clear();
    logged = false;
    chatModel->removeRows(0, chatModel->rowCount());
    ui->users->clear();
    if (isVisible()) {
        attemptConnection();
    }
}

void ClientWindow::userEventImpl(const QString& username, const
QString& event)
{
    const int newRow = chatModel->rowCount();
    chatModel->insertRow(newRow);
    chatModel->setData(chatModel->index(newRow, 0), tr("%1
%2").arg(username, event));
    chatModel->setData(chatModel->index(newRow, 0),
Qt::AlignCenter, Qt::TextAlignmentRole);
    chatModel->setData(chatModel->index(newRow, 0),
QBrush(Qt::gray), Qt::ForegroundRole);

    ui->chatView->scrollToBottom();
    lastUserName.clear();
}
```

```
void ClientWindow::userJoined(const QString& username)
{
    if (logged) {
        userEventImpl(username, "joined the group");
        ui->users->addItem(username);
    }
}

void ClientWindow::userLeft(const QString& username)
{
    userEventImpl(username, "left the group");
    QList<QListWidgetItem*> items = ui->users->findItems(username,
Qt::MatchExactly);
    if (items.isEmpty()) {
        return;
    }
    delete items.at(0);
}

void ClientWindow::informJoiner(const QStringList& usernames,
const QList<Message>& messages)
{
    for (const auto& username : usernames) {
        ui->users->addItem(username);
    }
    for (const auto& message : messages) {
        messageReceived(message);
    }
}

void ClientWindow::error(const QAbstractSocket::SocketError
socketError)
{
```

```
switch (socketError) {
    case QAbstractSocket::ConnectionRefusedError:
        qWarning() << "ConnectionRefusedError";
        break;
    case QAbstractSocket::RemoteHostClosedError:
        qWarning() << "RemoteHostClosedError";
        break;
    case QAbstractSocket::HostNotFoundError:
        qWarning() << "HostNotFoundError";
        break;
    case QAbstractSocket::SocketAccessError:
        qWarning() << "SocketAccessError";
        break;
    case QAbstractSocket::SocketResourceError:
        qWarning() << "SocketResourceError";
        break;
    case QAbstractSocket::SocketTimeoutError:
        qWarning() << "SocketTimeoutError";
        return;
    case QAbstractSocket::DatagramTooLargeError:
        qWarning() << "DatagramTooLargeError";
        break;
    case QAbstractSocket::NetworkError:
        qWarning() << "NetworkError";
        break;
    case QAbstractSocket::AddressInUseError:
        qWarning() << "AddressInUseError";
        break;
    case QAbstractSocket::SocketAddressNotAvailableError:
        qWarning() << "SocketAddressNotAvailableError";
        break;
    case QAbstractSocket::UnsupportedSocketOperationError:
        qWarning() << "UnsupportedSocketOperationError";
```

```
        break;
    case QAbstractSocket::UnfinishedSocketOperationError:
        qWarning() << "UnfinishedSocketOperationError";
        break;
    case QAbstractSocket::ProxyAuthenticationRequiredError:
        qWarning() << "ProxyAuthenticationRequiredError";
        break;
    case QAbstractSocket::SslHandshakeFailedError:
        qWarning() << "SslHandshakeFailedError";
        break;
    case QAbstractSocket::ProxyConnectionRefusedError:
        qWarning() << "ProxyConnectionRefusedError";
        break;
    case QAbstractSocket::ProxyConnectionClosedError:
        qWarning() << "ProxyConnectionClosedError";
        break;
    case QAbstractSocket::ProxyConnectionTimeoutError:
        qWarning() << "ProxyConnectionTimeoutError";
        break;
    case QAbstractSocket::ProxyNotFoundError:
        qWarning() << "ProxyNotFoundError";
        break;
    case QAbstractSocket::ProxyProtocolError:
        qWarning() << "ProxyProtocolError";
        break;
    case QAbstractSocket::OperationError:
        qWarning() << "OperationError";
        return;
    case QAbstractSocket::SslInternalError:
        qWarning() << "SslInternalError";
        break;
    case QAbstractSocket::SslInvalidUserDataError:
        qWarning() << "SslInvalidUserDataError";
```

```
            break;
        case QAbstractSocket::TemporaryError:
            qWarning() << "TemporaryError";
            break;
        default:
            Q_UNREACHABLE();
    }

    ui->sendButton->setEnabled(false);
    ui->messageEdit->setEnabled(false);
    ui->chatView->setEnabled(false);
    lastUserName.clear();
    logged = false;
    chatModel->removeRows(0, chatModel->rowCount());
    ui->users->clear();
    if (isVisible()) {
        attemptConnection();
    }
}
```

clientcore.cpp

```
#include <QTcpSocket>
#include <QDataStream>
#include <QJsonParseError>
#include <QJsonObject>
#include <QJsonArray>
#include <QFile>
#include "clientcore.h"
#include "constants.h"

ClientCore::ClientCore(QObject* parent) : QObject(parent),
clientSocket(new QSslSocket(this))
{
#ifdef SSL_ENABLE
    clientSocket->setProtocol(QSsl::SslV3);
```

```
    QByteArray certificate;
    QFile fileCertificate("ssl/ssl.cert");
    if (fileCertificate.open(QIODevice::ReadOnly)) {
        certificate = fileCertificate.readAll();
        fileCertificate.close();
    } else {
        qWarning() << fileCertificate.errorString();
    }
    QSslCertificate sslCertificate(certificate);
    clientSocket->setLocalCertificate(sslCertificate);
#endif


    connect(clientSocket, &QSslSocket::connected, this,
&ClientCore::connected);
    connect(clientSocket, &QSslSocket::disconnected, this,
&ClientCore::disconnected);


    connect(clientSocket, &QSslSocket::readyRead, this,
&ClientCore::onReadyRead);
    connect(clientSocket,
QOverload<QAbstractSocket::SocketError>::of(&QAbstractSocket::erro
r), this,
            &ClientCore::error);
}


ClientCore::~ClientCore()
{
    delete clientSocket;
}


void ClientCore::connectToServer(const QHostAddress& address,
const quint16 port)
{
    clientSocket->connectToHost(address, port);
#ifdef SSL_ENABLE
    clientSocket->startClientEncryption();
```

```
#endif
}

void ClientCore::login(const QString& username, const QString&
password)
{
    if (clientSocket->state() == QAbstractSocket::ConnectedState)
{
        this->name = username;
        QDataStream clientStream(clientSocket);
        clientStream.setVersion(serializerVersion);

        QJsonObject packet;
        packet[Packet::Type::TYPE]     = Packet::Type::LOGIN;
        packet[Packet::Data::USERNAME] = username;
        packet[Packet::Data::PASSWORD] = password;
        clientStream <<
QJsonDocument(packet).toJson(QJsonDocument::Compact);
    }
}

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();
}
```

```
void ClientCore::disconnectFromHost()
{
    clientSocket->disconnectFromHost();
}

bool ClientCore::isEqualPacketType(const QJsonValue& jsonType,
const char* const strType)
{
    return jsonType.toString().compare(QLatin1String(strType),
Qt::CaseInsensitive) == 0;
}

void ClientCore::handleLoginPacket(const QJsonObject& packet)
{
    const QJsonValue successVal =
packet.value(QLatin1String(Packet::Data::SUCCESS));
    if (successVal.isNull() || !successVal.isBool()) {
        return;
    }
    const bool loginSuccess = successVal.toBool();
    if (loginSuccess) {
        emit loggedIn();
        return;
    }
    const QJsonValue reasonVal =
packet.value(QLatin1String(Packet::Data::REASON));
    emit loginError(reasonVal.toString());
}

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()});
}

void ClientCore::handleUserJoinedPacket(const QJsonObject& packet)
{
    const QJsonValue usernameVal =
packet.value(QLatin1String(Packet::Data::USERNAME));
    if (usernameVal.isNull() || !usernameVal.isString()) {
        return;
    }
    emit userJoined(usernameVal.toString());
}

void ClientCore::handleUserLeftPacket(const QJsonObject& packet)
{
    const QJsonValue usernameVal =
packet.value(QLatin1String(Packet::Data::USERNAME));
    if (usernameVal.isNull() || !usernameVal.isString()) {
        return;
    }
    emit userLeft(usernameVal.toString());
}
```

```
void ClientCore::handleInformJoinerPacket(const QJsonObject&
packet)
{
    const QJsonValue usernamesVal =
packet.value(QLatin1String(Packet::Data::USERNAMES));
    if (usernamesVal.isNull() || !usernamesVal.isArray()) {
        return;
    }
    QJsonArray jsonUsernames = usernamesVal.toArray();
    QStringList usernames;
    for (const auto& jsonUsername : jsonUsernames) {
        usernames.push_back(jsonUsername.toString());
    }

    const QJsonValue messagesVal =
packet.value(QLatin1String(Packet::Data::MESSAGES));
    if (messagesVal.isNull() || !messagesVal.isArray()) {
        return;
    }

    QJsonArray jsonMessages = messagesVal.toArray();
    QList<Message> messages;
    for (const auto& jsonMessage : jsonMessages) {
        QJsonObject obj = jsonMessage.toObject();
        QString sender  = obj[Packet::Data::SENDER].toString();
        QString text    = obj[Packet::Data::TEXT].toString();
        QString time    = obj[Packet::Data::TIME].toString();
        messages.push_back({sender, text, time});
    }
    emit informJoiner(usernames, messages);
}

void ClientCore::packetReceived(const QJsonObject& packet)
{
```

```
    const QJsonValue packetTypeVal =
packet.value(QLatin1String(Packet::Type::TYPE));
    if (packetTypeVal.isNull() || !packetTypeVal.isString()) {
        return;
    }

    if (isEqualPacketType(packetTypeVal, Packet::Type::LOGIN)) {
        handleLoginPacket(packet);
    } else if (isEqualPacketType(packetTypeVal,
Packet::Type::MESSAGE)) {
        handleMessagePacket(packet);
    } else if (isEqualPacketType(packetTypeVal,
Packet::Type::USER_JOINED)) {
        handleUserJoinedPacket(packet);
    } else if (isEqualPacketType(packetTypeVal,
Packet::Type::USER_LEFT)) {
        handleUserLeftPacket(packet);
    } else if (isEqualPacketType(packetTypeVal,
Packet::Type::INFORM_JOINER)) {
        handleInformJoinerPacket(packet);
    }
}

void ClientCore::onReadyRead()
{
    QByteArray jsonData;
    QDataStream socketStream(clientSocket);
    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()) {
                    packetReceived(jsonDoc.object());
                }
            }
        } else {
            break;
        }
    }
}
```

constants.h

```
#ifndef MESSENGER_CONSTANTS_H
#define MESSENGER_CONSTANTS_H

#include "QDataStream"

constexpr auto serializerVersion = QDataStream::Qt_5_7;
constexpr quint16 PORT           = 30000;
constexpr const char* const HOST = "127.0.0.1";

namespace Packet {
    namespace Type {
        constexpr const char* const TYPE          = "type";
        constexpr const char* const LOGIN         = "login";
        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 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

#endif // MESSENGER_CONSTANTS_H
```

servercore.cpp

```
#include <QThread>
#include <functional>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QTimer>
#include "servercore.h"
#include "db.h"
#include "constants.h"
#include "message.h"

ServerCore::ServerCore(QObject* parent) : QTcpServer(parent),
idealThreadCount(qMax(QThread::idealThreadCount(), 1))
{
    threads.reserve(idealThreadCount);
    threadLoadFactor.reserve(idealThreadCount);
}

ServerCore::~ServerCore()
{
    for (QThread* singleThread : threads) {
        singleThread->quit();
        singleThread->wait();
```

```
    }
}

void ServerCore::incomingConnection(const qintptr
socketDescriptor)
{
    auto* worker = new ServerWorker;
    if (!worker->setSocketDescriptor(socketDescriptor)) {
        worker->deleteLater();
        return;
    }

    int threadIdx = threads.size();
    if (threadIdx < idealThreadCount) {
        threads.append(new QThread(this));
        threadLoadFactor.append(1);
        threads.last()->start();
    } else {
        threadIdx =
static_cast<int>(std::distance(threadLoadFactor.begin(),

std::min_element(threadLoadFactor.begin(),
threadLoadFactor.end())));
        ++threadLoadFactor[threadIdx];
    }

    worker->moveToThread(threads.at(threadIdx));
    connect(threads.at(threadIdx), &QThread::finished, worker,
&QObject::deleteLater);
    connect(worker, &ServerWorker::disconnectedFromClient, this,
            [this, worker, threadIdx] { userDisconnected(worker,
threadIdx); });

    connect(worker, &ServerWorker::error, this, [worker] {
userError(worker); });
```

```
    connect(worker, &ServerWorker::packetReceived, this, [this,
worker](auto&& placeholder) {
        packetReceived(worker,
std::forward<decltype(placeholder)>(placeholder));
    });
    connect(this, &ServerCore::stopAllClients, worker,
&ServerWorker::disconnectFromClient);
    clients.append(worker);
    qInfo() << "new client connected";
}

void ServerCore::sendPacket(ServerWorker* const destination, const
QJsonObject& packet)
{
    Q_ASSERT(destination);
    QTimer::singleShot(0, destination, [destination, packet] {
destination->sendPacket(packet); });
}

void ServerCore::unicast(const QJsonObject& packet, ServerWorker*
const receiver)
{
    auto worker = std::find(clients.begin(), clients.end(),
receiver);
    if (worker != clients.end()) {
        Q_ASSERT(*worker);
        sendPacket(*worker, packet);
    }
}

void ServerCore::broadcast(const QJsonObject& packet,
ServerWorker* const exclude)
{
    for (ServerWorker* worker : clients) {
        Q_ASSERT(worker);
        if (worker != exclude) {
```

```
            sendPacket(worker, packet);
        }
    }
}


void ServerCore::packetReceived(ServerWorker* sender, const
QJsonObject& packet)
{
    Q_ASSERT(sender);
    qInfo() << qPrintable("JSON received\n" +
QString::fromUtf8(QJsonDocument(packet).toJson(QJsonDocument::Inde
nted)));


    const QString& userName = sender->getUserName();
    if (userName.isEmpty()) {
        return packetFromLoggedOut(sender, packet);
    }
    packetFromLoggedIn(sender, packet);
}


void ServerCore::userDisconnected(ServerWorker* const sender,
const int threadIdx)
{
    --threadLoadFactor[threadIdx];
    clients.removeAll(sender);
    const QString& userName = sender->getUserName();
    if (!userName.isEmpty()) {
        QJsonObject packet;
        packet[Packet::Type::TYPE]     = Packet::Type::USER_LEFT;
        packet[Packet::Data::USERNAME] = userName;
        broadcast(packet, nullptr);
        qInfo() << qPrintable(userName + QString("
disconnected"));
    }
    sender->deleteLater();
```

```
}


void ServerCore::userError(ServerWorker* const sender)
{
    qWarning() << qPrintable(QString("error from <") + sender-
>getUserName() + QString(">"));
}


void ServerCore::stopServer()
{
    emit stopAllClients();
    close();
}


bool ServerCore::isEqualPacketType(const QJsonValue& jsonType,
const char* const strType)
{
    return jsonType.toString().compare(QLatin1String(strType),
Qt::CaseInsensitive) == 0;
}


QJsonArray ServerCore::getUsernames(ServerWorker* const exclude)
const
{
    QJsonArray usernames;
    for (ServerWorker* worker : clients) {
        Q_ASSERT(worker);
        if (worker != exclude) {
            QString username = worker->getUserName();
            if (!username.isEmpty()) {
                usernames.push_back(qMove(username));
            }
        }
    }
    return usernames;
```

```
}

QJsonArray ServerCore::getMessages()
{
    QList<Message> dbMessages = db::fetchMessages("main");
    QJsonArray messages;
    for (const auto& message : dbMessages) {
        QJsonObject leafObject;
        leafObject[Packet::Data::SENDER] = message.getSender();
        leafObject[Packet::Data::TEXT]   = message.getMessage();
        leafObject[Packet::Data::TIME]   = message.getTime();
        messages.push_back(leafObject);
    }
    return messages;
}

void ServerCore::packetFromLoggedOut(ServerWorker* const sender,
const QJsonObject& packet)
{
    Q_ASSERT(sender);
    const QJsonValue typeVal =
packet.value(QLatin1String(Packet::Type::TYPE));
    if (typeVal.isNull() || !typeVal.isString()) {
        return;
    }
    if (!isEqualPacketType(typeVal, Packet::Type::LOGIN)) {
        return;
    }

    const QJsonValue usernameVal =
packet.value(QLatin1String(Packet::Data::USERNAME));
    if (usernameVal.isNull() || !usernameVal.isString()) {
        return;
    }
```

```
    const QString newUserName =
usernameVal.toString().simplified();
    if (newUserName.isEmpty()) {
        return;
    }

    const QJsonValue passwordVal =
packet.value(QLatin1String(Packet::Data::PASSWORD));
    if (passwordVal.isNull() || !passwordVal.isString()) {
        return;
    }
    QString password = passwordVal.toString().simplified();
    if (password.isEmpty()) {
        return;
    }

    if (password != db::fetchGroupPassword("main")) {
        QJsonObject errorPacket;
        errorPacket[Packet::Type::TYPE]    = Packet::Type::LOGIN;
        errorPacket[Packet::Data::SUCCESS] = false;
        errorPacket[Packet::Data::REASON]  = "invalid password";
        sendPacket(sender, errorPacket);
        return;
    }

    for (ServerWorker* worker : qAsConst(clients)) {
        if (worker == sender) {
            continue;
        }
        if (worker->getUserName().compare(newUserName,
Qt::CaseInsensitive) == 0) {
            QJsonObject errorPacket;
            errorPacket[Packet::Type::TYPE]    =
Packet::Type::LOGIN;
            errorPacket[Packet::Data::SUCCESS] = false;
```

```
            errorPacket[Packet::Data::REASON]  = "duplicate
username";
            sendPacket(sender, errorPacket);
            return;
        }
    }

    sender->setUserName(newUserName);
    QJsonObject successPacket;
    successPacket[Packet::Type::TYPE]    = Packet::Type::LOGIN;
    successPacket[Packet::Data::SUCCESS] = true;
    sendPacket(sender, successPacket);

    QJsonObject unicastPacket;
    unicastPacket[Packet::Type::TYPE]      =
Packet::Type::INFORM_JOINER;
    unicastPacket[Packet::Data::USERNAMES] = getUsernames(sender);
    unicastPacket[Packet::Data::MESSAGES]  = getMessages();
    unicast(unicastPacket, sender);

    QJsonObject connectedBroadcastPacket;
    connectedBroadcastPacket[Packet::Type::TYPE]     =
Packet::Type::USER_JOINED;
    connectedBroadcastPacket[Packet::Data::USERNAME] =
newUserName;
    broadcast(connectedBroadcastPacket, sender);
}

void ServerCore::packetFromLoggedIn(ServerWorker* const sender,
const QJsonObject& packet)
{
    Q_ASSERT(sender);
    const QJsonValue typeVal =
packet.value(QLatin1String(Packet::Type::TYPE));
    if (typeVal.isNull() || !typeVal.isString()) {
        return;
```

```
    }
    if (!isEqualPacketType(typeVal, Packet::Type::MESSAGE)) {
        return;
    }

    const QJsonValue senderVal =
packet.value(QLatin1String(Packet::Data::SENDER));
    if (senderVal.isNull() || !senderVal.isString()) {
        return;
    }
    const QString senderName = senderVal.toString();
    if (senderName.isEmpty()) {
        return;
    }

    const QJsonValue textVal =
packet.value(QLatin1String(Packet::Data::TEXT));
    if (textVal.isNull() || !textVal.isString()) {
        return;
    }
    const QString text = textVal.toString().trimmed();
    if (text.isEmpty()) {
        return;
    }

    const QJsonValue timeVal =
packet.value(QLatin1String(Packet::Data::TIME));
    if (timeVal.isNull() || !timeVal.isString()) {
        return;
    }
    const QString time = timeVal.toString();
    if (time.isEmpty()) {
        return;
    }
```

```
    QJsonObject broadcastPacket;
    broadcastPacket[Packet::Type::TYPE]   = Packet::Type::MESSAGE;
    broadcastPacket[Packet::Data::SENDER] = sender->getUserName();
    broadcastPacket[Packet::Data::TEXT]   = text;
    broadcastPacket[Packet::Data::TIME]   = time;
    broadcast(broadcastPacket, sender);

    db::addMessage({senderName, text, time});
}
```

serverworker.cpp

```
#include <QDataStream>
#include <QJsonDocument>
#include <QJsonObject>
#include <QFile>
#include <QSslKey>
#include "serverworker.h"
#include "constants.h"

ServerWorker::ServerWorker(QObject* parent) : QObject(parent),
serverSocket(new QSslSocket(this))
{
#ifdef SSL_ENABLE
    serverSocket->setProtocol(QSsl::SslV3);
    QByteArray certificate;
    QFile fileCertificate("ssl/ssl.cert");
    if (fileCertificate.open(QIODevice::ReadOnly)) {
        certificate = fileCertificate.readAll();
        fileCertificate.close();
    } else {
        qWarning() << fileCertificate.errorString();
    }
    QSslCertificate sslCertificate(certificate);
    serverSocket->setLocalCertificate(sslCertificate);
```

```
    QByteArray privateKey;
    QFile fileKey("ssl/ssl.key");
    if (fileKey.open(QIODevice::ReadOnly)) {
        privateKey = fileKey.readAll();
        fileKey.close();
    } else {
        qWarning() << fileKey.errorString();
    }
    QSslKey sslKey(privateKey, QSsl::Rsa, QSsl::Pem,
QSsl::PrivateKey, "localhost");
    serverSocket->setPrivateKey(sslKey);
#endif


    connect(serverSocket, &QSslSocket::readyRead, this,
&ServerWorker::onReadyRead);
    connect(serverSocket, &QSslSocket::disconnected, this,
&ServerWorker::disconnectedFromClient);
    connect(serverSocket,
QOverload<QAbstractSocket::SocketError>::of(&QAbstractSocket::erro
r), this,
            &ServerWorker::error);
}


ServerWorker::~ServerWorker()
{
    delete serverSocket;
}


bool ServerWorker::setSocketDescriptor(const qintptr
socketDescriptor)
{
    const bool ret = serverSocket-
>setSocketDescriptor(socketDescriptor);
#ifdef SSL_ENABLE
    serverSocket->startServerEncryption();
```

```
#endif
    return ret;
}

void ServerWorker::sendPacket(const QJsonObject& packet)
{
    const QByteArray jsonData = QJsonDocument(packet).toJson();
    qInfo() << qPrintable(QString("sending JSON to ") +
getUserName() + QString("\n") + QString::fromUtf8(jsonData));

    QDataStream socketStream(serverSocket);
    socketStream.setVersion(serializerVersion);
    socketStream << jsonData;
}

void ServerWorker::disconnectFromClient()
{
    serverSocket->disconnectFromHost();
}

QString ServerWorker::getUserName() const
{
    userNameLock.lockForRead();
    QString result = userName;
    userNameLock.unlock();
    return result;
}

void ServerWorker::setUserName(const QString& name)
{
    userNameLock.lockForWrite();
    userName = name;
    userNameLock.unlock();
}
```

```
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;
        }
    }
}
```

connectionpool.cpp

```
#include <QUuid>
#include "connectionpool.h"
#include "config.h"
```

```
ConnectionPool::~ConnectionPool()
{
    foreach (QString connectionName, usedConnectionNames) {
        QSqlDatabase::removeDatabase(connectionName);
    }
}

ConnectionPool& ConnectionPool::getInstance()
{
    if (instance == nullptr) {
        QMutexLocker locker(&mutex);
        if (instance == nullptr) {
            instance = new ConnectionPool();
        }
    }
    return *instance;
}

void ConnectionPool::release()
{
    QMutexLocker locker(&mutex);
    delete instance;
    instance = nullptr;
}

QSqlDatabase ConnectionPool::getConnection()
{
    ConnectionPool& pool = ConnectionPool::getInstance();

    QMutexLocker locker(&mutex);
    int connectionCount = pool.unusedConnectionNames.size() +
pool.usedConnectionNames.size();
    for (int i = 0; i < ConnectionPool::maxWaitTime &&
pool.unusedConnectionNames.empty() &&
```

```
                    connectionCount ==
ConnectionPool::maxConnectionCount;
         i += ConnectionPool::waitInterval) {
        waitConnection.wait(&mutex, ConnectionPool::waitInterval);
        connectionCount = pool.unusedConnectionNames.size() +
pool.usedConnectionNames.size();
    }

    QString connectionName;
    if (!pool.unusedConnectionNames.empty()) {
        connectionName = pool.unusedConnectionNames.dequeue();
    } else if (connectionCount <
ConnectionPool::maxConnectionCount) {
        connectionName = QString(QUuid::createUuid().toString());
    } else {
        qInfo() << "cannot create more connections";
        return {};
    }

    QSqlDatabase db =
ConnectionPool::createConnection(connectionName);
    if (db.isOpen()) {
        pool.usedConnectionNames.enqueue(connectionName);
    }
    return db;
}

void ConnectionPool::releaseConnection(const QSqlDatabase&
connection)
{
    ConnectionPool& pool   = ConnectionPool::getInstance();
    QString connectionName = connection.connectionName();
    if (pool.usedConnectionNames.contains(connectionName)) {
        QMutexLocker locker(&mutex);
        pool.usedConnectionNames.removeOne(connectionName);
```

```
        pool.unusedConnectionNames.enqueue(connectionName);
        waitConnection.wakeOne();
    }
}

QSqlDatabase ConnectionPool::createConnection(const QString&
connectionName)
{
    if (QSqlDatabase::contains(connectionName)) {
        QSqlDatabase db = QSqlDatabase::database(connectionName);

        if (testOnBorrow) {
            QSqlQuery query(testOnBorrowQuery, db);
            if (query.lastError().type() != QSqlError::NoError &&
!db.open()) {
                qWarning() << qPrintable(QString("DB fail:") +
db.lastError().text());
                return {};
            }
        }

        return db;
    }

    QSqlDatabase db = QSqlDatabase::addDatabase(DB_TYPE,
connectionName);
    db.setHostName(DB_HOSTNAME);
    db.setDatabaseName(DB_NAME);
    db.setUserName(DB_USERNAME);
    db.setPassword(DB_PASSWORD);
    if (!db.open()) {
        qWarning() << qPrintable(QString("DB fail:") +
db.lastError().text());
        return {};
    }
```

```
    qInfo() << qPrintable(QString("DB connection ") +
connectionName + QString(" ok"));


    return db;
}
```

window.ui

```
<?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>
      <widget class="QPushButton" name="createGroup">
       <property name="cursor">
```

```
        <cursorShape>PointingHandCursor</cursorShape>
       </property>
       <property name="styleSheet">
        <string notr="true">*:hover {
  background-color: rgb(98,194,233);
}</string>
       </property>
       <property name="text">
        <string>create group</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QPushButton" name="connectGroup">
       <property name="cursor">
        <cursorShape>PointingHandCursor</cursorShape>
       </property>
       <property name="styleSheet">
        <string notr="true">*:hover {
  background-color: rgb(98,194,233);
}</string>
       </property>
       <property name="text">
        <string>connect group</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QComboBox" name="groups">
       <property name="cursor">
        <cursorShape>PointingHandCursor</cursorShape>
       </property>
       <property name="styleSheet">
```

```
        <string notr="true">* {
  font: bold;
}</string>
       </property>
      </widget>
     </item>
     <item>
      <widget class="QListWidget" name="users"/>
     </item>
     <item>
      <widget class="QPushButton" name="settings">
       <property name="cursor">
        <cursorShape>PointingHandCursor</cursorShape>
       </property>
       <property name="styleSheet">
        <string notr="true">*:hover {
  background-color: rgb(98,194,233);
}</string>
       </property>
       <property name="text">
        <string>settings</string>
       </property>
      </widget>
     </item>
    </layout>
   </item>
   <item row="0" column="1">
    <layout class="QVBoxLayout" name="verticalLayout_2"
stretch="0,0">
     <item>
      <widget class="QListView" name="chatView">
       <property name="enabled">
        <bool>false</bool>
       </property>
```

```
  <property name="editTriggers">
   <set>QAbstractItemView::NoEditTriggers</set>
  </property>
 </widget>
</item>
<item>
 <layout class="QHBoxLayout" name="horizontalLayout">
  <item>
   <widget class="QLineEdit" name="messageEdit">
    <property name="enabled">
     <bool>false</bool>
    </property>
    <property name="sizePolicy">
     <sizepolicy hsizetype="Expanding" vsizetype="Fixed">
      <horstretch>0</horstretch>
      <verstretch>0</verstretch>
     </sizepolicy>
    </property>
    <property name="minimumSize">
     <size>
      <width>0</width>
      <height>35</height>
     </size>
    </property>
    <property name="maximumSize">
     <size>
      <width>16777215</width>
      <height>16777215</height>
     </size>
    </property>
    <property name="styleSheet">
     <string notr="true"/>
    </property>
```

```
        </widget>
       </item>
       <item>
        <widget class="QPushButton" name="sendButton">
         <property name="enabled">
          <bool>false</bool>
         </property>
         <property name="minimumSize">
          <size>
           <width>35</width>
           <height>30</height>
          </size>
         </property>
         <property name="maximumSize">
          <size>
           <width>35</width>
           <height>30</height>
          </size>
         </property>
         <property name="cursor">
          <cursorShape>PointingHandCursor</cursorShape>
         </property>
         <property name="layoutDirection">
          <enum>Qt::LeftToRight</enum>
         </property>
         <property name="styleSheet">
          <string notr="true">* {
  border-image: url(:/sendButton.png) 3 10 3 10;
  border-top: 3px transparent;
  border-bottom: 3px transparent;
  border-right: 10px transparent;
  border-left: 10px transparent;
}</string>
```

```
          </property>
          <property name="text">
           <string/>
          </property>
          <property name="default">
           <bool>true</bool>
          </property>
         </widget>
        </item>
       </layout>
      </item>
     </layout>
    </item>
   </layout>
  </widget>
  <resources/>
  <connections/>
</ui>
```