Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования
«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт вычислительной математики и информационных технологий

Кафедра системного анализа и информационных технологий

Направление подготовки: 10.03.01 – Информационная безопасность

Профиль: Безопасность компьютерных систем

КУРСОВАЯ РАБОТА

РАЗРАБОТКА РАСПРЕДЕЛЁННОГО ПРИЛОЖЕНИЯ "ЧАТ" С ЗАЩИТОЙ ХРАНЕНИЯ И ПЕРЕДАЧИ ДАННЫХ

Студент 3 курса

группы 09-841

«___» _____ 2021 г. _____ Пьянков А.А.

Научный руководитель

доцент, к.н., КФУ

«___» _____ 2021 г. _____ Андрианова А.А.

Казань-2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В текущее время трудно себе представить повседневную жизнь без обмена всевозможными видами информации. Большую часть из них составляет обмен мгновенными сообщениями через различные типы мессенджеров.

При разработке таких приложений используются всевозможные подходы для организации обмена данными. Довольно простым и в то же время эффективным способом построения мессенджера является клиент – серверная архитектура, где сообщения и прочие данные от конечных пользователей, то есть клиентов, отправляются сначала на сервер, которых может быть несколько. Там эти сообщения обрабатываются и уходят дальше одному или нескольким получателям.

Так же используются различные инструменты для построения качественного пользовательского интерфейса – UI, и эффективной и быстрой «начинки» – логики приложения.

Использование различных инструментов для разработки того или иного продукта занимает довольно длительное время. А если пытаться использовать сторонние библиотеки, которые, с первого взгляда, должны облегчать процесс разработки, не дают той простоты, удобства и эффективности, которую хотелось бы иметь.

Согласно прогнозам аналитиков, кроссплатформенная разработка программного обеспечения является будущим индустрии информационных технологий. Выпуская свое приложение на разные платформы и операционные системы, можно увеличить количество пользователей. Преимущество кроссплатформенной разработки заключается в том, время и сложность разработки значительно сокращаются. В первую очередь это связано с тем, что разработчику не нужно делать разный исходный код программы под каждую платформу и операционную систему, а достаточно работать внутри конкретного фреймворка. Также отпадает необходимость знать тонкости и нюансы каждой из платформ. Учитывая всё это,

затрачиваемые на разработку трудовые ресурсы будут уменьшены и, следовательно, уменьшаться финансовые затраты на разработку.

Цель работы – реализация и изучение процесса разработки клиентского приложения и сервера, которые будут являться кроссплатформенными. Так же все данные, передающиеся по сети, должны быть надёжно защищены криптографическими методами.

Для достижения заданной цели выделим следующие задачи:

– Изучение предметной области;

– Анализ методов проектирования кроссплатформенных приложений;

– Анализ и выбор метода создания зашифрованного канала;

– Реализация.

1.1. Основные понятия

Клиентское приложение – программа, работающая на устройстве пользователя и обеспечивающее его интерактивное взаимодействие с удалённой системой.

Сервер – программный компонент вычислительной системы, выполняющий сервисные и обслуживающие функции по запросу одного или нескольких клиентов, предоставляя им доступ к определённым ресурсам и услугам.

Кроссплатформенность – возможность работы программного обеспечения на двух или более платформах. Эта возможность обеспечивается благодаря использованию в процессе разработки высокоуровневых языков программирования и специальных сред разработки, которые, в свою очередь, поддерживают условную компиляцию, компоновку и выполнение кода для двух или более платформ. Самым простым примером является программное обеспечение, которое может работать как на ОС Windows, так и на ОС Linux, причем для сборки программного обеспечения используется один и тот же программный код.

Библиотеки и среды выполнения – множество библиотек, которые включаются определенными средами разработки. Входят в состав компиляторов, операционных систем или сред разработки, как сопутствующие элементы. Главной задачей является поддержка функций во время выполнения программы, от запуска до завершения ее работы.

Яркими представителями кроссплатформенных библиотек являются: Qt, Boost, STL (Standard Template Library), OpenGL и другие.

Криптография – наука о шифровании, сокрытии информации от посторонних, не обладающих специальным секретом – ключом. Много столетий назад криптографию начали использовать для передачи секретных сообщений в военных целях и на государственной службе. Далее, с появлением всемирной сети Интернет появилась необходимость уметь налаживать каналы защищённой передачи данных, чтобы третьи лица не

могли похитить конфиденциальную информацию пользователей во время её передачи и хранения. Как следствие, сейчас почти что любое использование электронно-вычислительного оборудования будет включать в себя использование алгоритмов криптографии. Данная наука очень обширна, но нам для рассмотрения интересны несколько нижеприведённых термина:

– Аутентификация – процедура проверки подлинности, например, пользователя по его логину и паролю.

– Симметричное шифрование – способ, в котором для шифрования и дешифрования используется один и тот же ключ. Алгоритмы симметричного шифрования характеризуются высокой скоростью, но обладают одним существенным недостатком – необходимость каким-то образом обеспечить обмен секретными ключами между сторонами.

– Асимметричное шифрование – метод шифрования, предполагающий использование двух ключей – public и private. Public ключ распространяется открыто и с его помощью шифруются данные. Но после этого расшифровать их возможно только с помощью private ключа, который, в свою очередь, нигде не распространяется открыто. Характеризуется довольно низкой скоростью шифрования и поэтому используется, в основном, для обмена ключами для симметричных шифров.

– Электронно-цифровая подпись – некий набор данных, полученный криптографическими алгоритмами, позволяющий подтвердить авторство пользователя.

– Хеш – отображение некоторых данных, получаемое с помощью хеш-функции. При этом зная хеш, невозможно найти исходные данные. Т.е. хеш-функцию можно считать односторонней.

1.2. Кроссплатформенный фреймворк Qt

Данный фреймворк включает в себя большое количество всевозможных классов. Эти классы имеют строгую иерархию, которая представлена в строгой внутренней структуре, которая в свою очередь охватывает основную часть функциональных возможностей. Она не является монолитной и имеет множество модулей, которые могут взаимодействовать между собой.

Ниже представлен список модулей, доступных в Qt.

– QtCore — классы ядра библиотеки, используемые другими модулями;

– QtGui — компоненты графического интерфейса;

– QtWidgets — содержит классы для классических приложений на основе виджетов, модуль выделен из QtGui в Qt 5;

– Qt QML — модуль для поддержки QML;

– QtNetwork — набор классов для сетевого программирования. Поддержка различных высокоуровневых протоколов может меняться от версии к версии. В версии 4.2.x присутствуют классы для работы с протоколами FTP и HTTP. Для работы с протоколами TCP/IP предназначены такие классы, как QTcpServer, QTcpSocket для TCP и QUdpSocket для UDP;

– QtOpenGL — набор классов для работы с OpenGL;

– QtSql — набор классов для работы с базами данных с использованием SQL. Основные классы данного модуля в версии 4.2.x: QSqlDatabase — класс для предоставления соединения с базой, для работы с какой-нибудь конкретной базой данных требует объект, унаследованный от класса QSqlDriver — абстрактного класса, который реализуется для конкретной базы данных и может требовать для компиляции SDK базы данных. Например, для сборки драйвера под СУБД Firebird или InterBase требуются .h-файлы и библиотеки статической компоновки, входящие в комплект поставки данной СУБД;

– QtScript — классы для работы с Qt Scripts;

– QtSvg — классы для отображения и работы с данными Scalable Vector Graphics (SVG);

– QtXml — модуль для работы с XML, поддерживаются модели SAX и DOM;

– QtDesigner — классы создания расширений для своих собственных виджетов;

– QtUiTools — классы для обработки в приложении форм Qt Designer;

– QtAssistant — справочная система;

– Qt3Support — модуль с классами, необходимыми для совместимости с библиотекой Qt версии 3.x.x;

– QtTest — классы для поддержки модульного тестирования;

– QtWebKit — модуль WebKit, интегрированный в Qt и доступный через её классы. (Начиная с Qt 5.6 признан устаревшим);

– QtWebEngine — модуль Chromium, интегрированный в Qt и доступный через её классы.

– QtXmlPatterns — модуль для поддержки XQuery 1.0 и XPath 2.0;

– Phonon — модуль для поддержки воспроизведения и записи видео и аудио, как локально, так и с устройств и по сети (Начиная с Qt 5 заменён на QtMultimedia);

– QtMultimedia — модуль для поддержки воспроизведения и записи видео и аудио, как локально, так и с устройств и по сети;

– QtCLucene — модуль для поддержки полнотекстового поиска, применяется в новой версии Assistant в Qt 4.4;

– ActiveQt — модуль для работы с ActiveX и COM технологиями для Qt-разработчиков под Windows.

– QtDeclarative — модуль, предоставляющий декларативный фреймворк для создания динамичных, настраиваемых пользовательских интерфейсов.

В таблице 1 ниже показаны платформы, поддерживаемые данным фреймворком.

Таблица 1 - основные модули

Платформа	Описание
Linux/Unix	
X11	Qt для оконного менеджера X (Linux, FreeBSD, HP-UX, Solaris, AIX, и т. д.).
Wayland	Qt для Wayland. Приложения на Qt могут переключаться между графическими бэкэндами вроде X и Wayland во время загрузки, если добавить опцию командной строки –platform. Это позволяет приложениям незаметно переходить с X11 на Wayland.
Встраиваемые Linux-системы	Qt для встраиваемых систем: КПК, смартфонов, и т. д. Существует в виде нескольких платформ, в зависимости от технологии отрисовки. DirectFB, LinuxFB и EGLFS (EGL Full Screen).
Android	Qt для Android, ранее известный как Necessitas.
Платформы Microsoft	
Windows	Qt для Microsoft Windows XP, Vista, 7, 8 и 10.
Windows CE	Qt для Windows CE 6 и Windows Embedded Compact 7.
Windows RT	Поддержка для основанных на WinRT приложениях для Windows 8 и Windows Phone 8. Начиная с версии 5.4: Windows Phone 8.1.

1.3. Объектная модель. Сигналы и слоты

Объектная модель подразумевает то, что все построено на объектах. Класс QObject является базовым и практически все классы библиотеки Qt являются его наследниками. Если планируется использовать механизм сигналов и слотов, то класс должен быть наследником класса QObject.

Сигналы и слоты - средство, с помощью которых можно построить эффективное, простое для понимания и в то же время абстрактное взаимодействие между объектами путем вызова событий, вырабатываемых объектами.

Механизм сигналов и слотов заменяет и улучшает старую концепцию callback функций, при этом являясь объектно – ориентированным подходом.

Старая концепция callback функций использует процедурный подход и обычные функции, которые вызываются в результате некоторых действий. Использование данного подхода сильно усложняет исходный код программы и делает его трудно читаемым. Так же отсутствует возможность проверки типа возвращаемых значений. Это связано с тем, что во всех случаях возвращается указатель на void и надо собственноручно приводить типы.

Возможность соединять объекты становится одной из основных концепций написания программ с использованием Qt. Каждый класс, который унаследован от QObject имеет возможность как отправлять, так и принимать сигналы и иметь неограниченное количество сигналов и слотов. Сигналы могут вызывать другие сигналы и слоты, привязанные к этому сигналу, в том числе, если объекты исполняются в разных потоках. Это очень сильно упрощает доступ к общим ресурсам. Также сообщения, отправляемые с помощью сигналов, могут иметь множество аргументов любого типа. Определить соединение сигналов и слотов можно в абсолютно любой части программы.

1.4. Структура проекта в Qt

Структура Qt проекта достаточно проста. Помимо исходных файлов в проекте находится файл проекта с расширением pro. Он необходим для вызова утилиты qmake и последующего создания make-файла. Он хранит в себе заранее предусмотренные инструкции, при помощи которых создается исполняемый модуль. Подробнее с этапами сборка можно ознакомится в официальной документации Qt [1]. Схематичное изображение ниже (Рисунок 1).

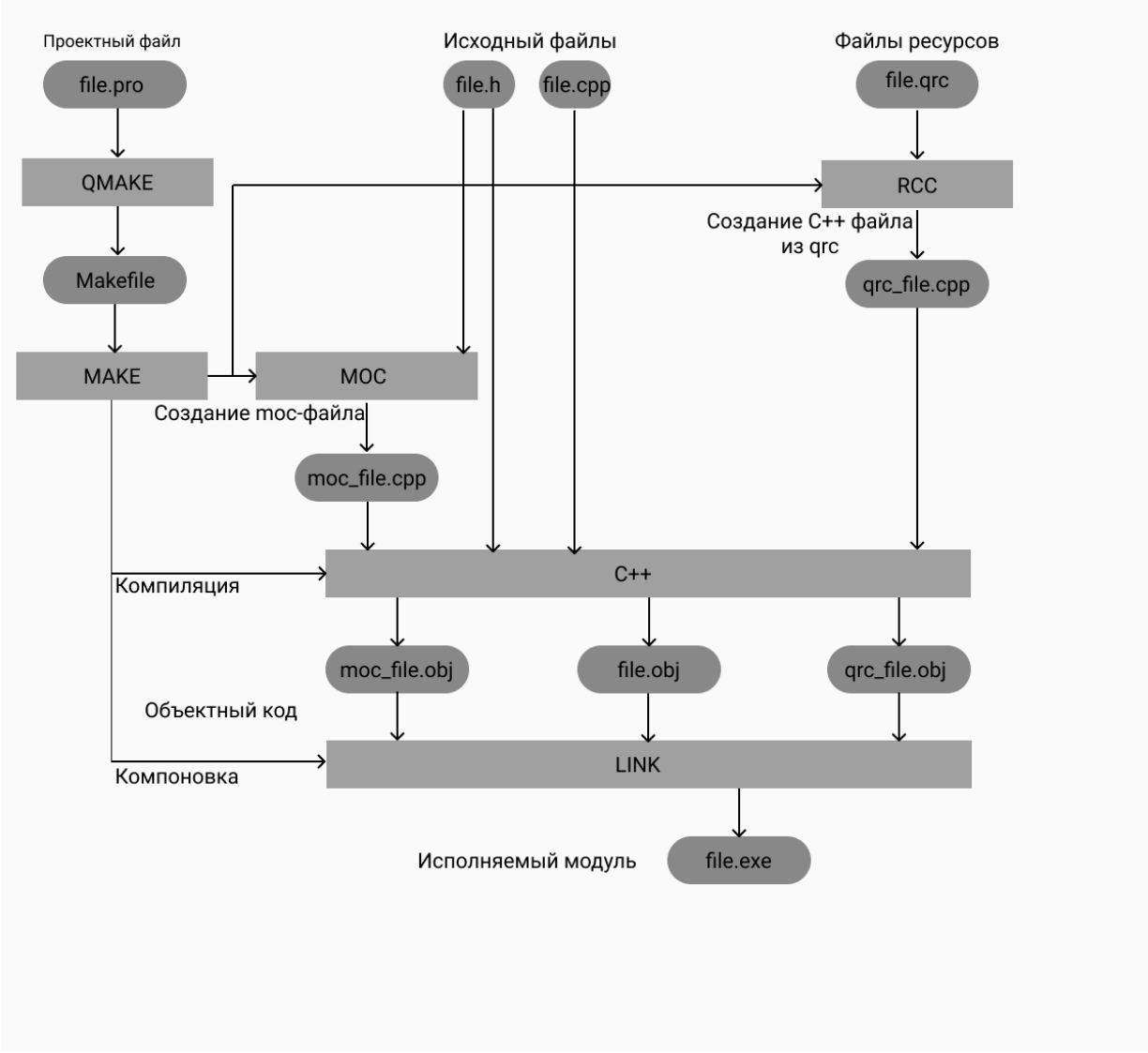

Рисунок 1 - Структура Qt проекта

В случае, если в проекте имеются дополнительные ресурсы, такие как, например, картинки и шрифты, то также будет создан и файл ресурсов. После этих процедур происходит процесс компиляции в файлы объектного кода. Они, в свою очередь, объединяются линкером, в результате чего мы и получаем исполняемый файл.

1.5. Клиент – серверная (распределённая) архитектура

При использовании данной архитектуры сетевая нагрузка распределяется между поставщиками услуг или серверами, и получателями услуг – клиентами. Фактически и клиент, и сервер являются программным обеспечением. В большинстве случаев они располагаются разных устройствах и общаются между собой средствами сетевых протоколов.

Сервер напрямую влияет на общую эффективность работы клиентов, так как на сервере должны обрабатываться запросы и отправляться ответы всем подключённым клиентам. По этой причине при наличии большого количества клиентов программа сервера должна обладать большим быстродействием и, желательно, быть установленной на высокопроизводительной машине. Данные на стороне сервера обычно хранятся в базе данных. ниже (Рисунок 2) приведу простейший пример клиент – серверного взаимодействия.

Рисунок 2 - Клиент – серверная архитектура

Клиентское приложение, однако, может быть установлено и на малопроизводительных системах. Но благодаря тому, что вычисления и обработка данных происходит на удалённом сервере, приложение будет обладать приемлемым быстродействием.

В качестве примера можно привести любой сервис онлайн – карт. Если бы они не обладали клиент – серверной архитектурой, то весь внушительный объём карт приходилось бы хранить на стороне пользователя. Так же появилась бы проблема синхронизации и обновления данных карт.

Клиент – серверная архитектура решает эти проблемы. Клиент отправляет на сервер запрос необходимой зоны карты, а сервер в ответ отправляет ему данные этой зоны.

1.6. Выбор криптографического протокола

При выборе протокола защиты канала передачи данных стоит учитывать криптостойкость, скорость и то, что этот протокол будет регулярно обновляться и поддерживаться. Исходя из этого был выбран протокол SSL - secure socket layer(уровень защищённых сокетов). Начиная с версии 3.0 протокол был заменён на TLS - transport layer security(безопасность транспортного уровня), но название "SSL" настолько прижилось, что сейчас при упоминании SSL скорее всего подразумевается TLS.

Цель протокола SSL/TLS - обеспечение защищённой передачи данных. Для этого используется ассиметричные алгоритмы шифрования для аутентификации и симметричные для передачи данных. Данная комбинация позволяет защищённо обменяться ключами с использованием асимметричных шифров и далее использовать эти ключи, используя симметричное шифрование данных для их быстрой передачи.

Рассмотрим поэтапно работу протокола SSL 3.0/TLS.

1. Клиент инициирует защищённое соединение, запрашивая информацию о SSL сертификате сервера.
2. Сервер высылает клиенту копию своего сертификата вместе с открытым ключём. Клиент сверяет полученый сертификат с адресом/названием сервера, удостоверяясь, что он пришёл именно от туда, от куда был запрошен. Так же проверяется срок действия сертификата и "корневой сертификат" - подтверждение того, что сертификат сервера выдан надёжным источником.
3. Если все проверки пройдены успешно, то клиент генерирует pre-master secret(предварительный секрет) для текущей сессии, шифрует его открытым ключом сервераи отправляет назад.
4. Сервер получает и расшифровывает своим закрытым ключом сгенерированный клиентом ключ. Далее его можно использовать для симметричного шифрования данных.

Наглядная схема протокола представлена далее (Рисунок 3).

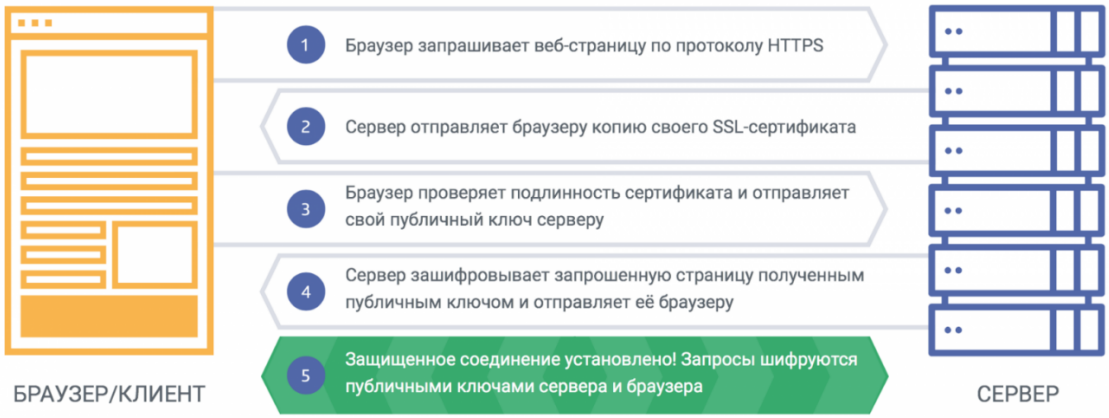

Рисунок 3 – Протокол TLS

В рассматриваемом протоколе используются два типа шифров - асимметричный и симметричный. От их выбора так же сильно будет зависеть криптостойкость протокола. В качестве ассиметричного алгоритма выбран алгоритм Диффи-Хеллмана. Если не вдаваться в математические подробности, то данный алгоритм основан на предположении необратимости

дискретного логарифмирования. Алгоритм используют в связке с ЭЦП, чтобы исключить возможность атаки "Man-in-the-middle". Для симметричного шифрования выбирается, например, AES или RC4, которые обладают таким важным свойством, как скорость.

Не лишним будет рассмотреть слабые и сильные стороны протокола TLS.К плюсам стоит отнести большую распространённость, возможность работы при отсутствии постоянного соединения между клиентом и сервером, "невидимость" для протоколов более высокого уровня, возможность создания защищённых TCP туннелей. Из минусов стоит упомянуть то, что программное обеспечение должно отвечать ряду требований для обеспечения поддержки TLS.

Одной из задач для разработки стоит безопасность. Поэтому необходимо организовать защищённое соединение между клиентами и сервером. Ранее в качестве криптографического протокола был выбран SSL 3.0/TLS. Благодаря тому, что фреймворк Qt содержит в себе множество классов для проектирования программного обеспечения, разработчику не придётся самому программировать алгоритмы шифрования. Можно использовать уже готовые классы, которые были разработаны и протестированы профильными специалистами. Один из таких классов - QSslSocket. Стоит обратить внимание, что для использования данного класса требуется наличие библиотеки OpenSSL. QSslSocket сделан на основе QTcpSocket и реализует протокол SSL/TLS поверх TCP. Достаточно установить соединение между двумя сокетами и дождаться сигнала QSslSocket::encrypted(), и после этого между сокетами образуется надёжно защищённый канал передачи данных. Сокет на стороне сервера будет являться серверным и именно он будет отвечать за хранение и поддержание актуальности SSL сертификата, что, в свою очередь, не будет создавать дополнительную нагрузку на клиентскую часть.

Следуя документации, QSslSocket является наследником QTcpSocket. Это позволяет сделать одну интересную вещь - вызовом одного метода

полностью отказываться от шифрования, а также обратно возвращаться к нему. Благодаря этому имеется возможность позволить пользователю переключаться в незащищённый режим, что может иметь смысл, когда будет важна скорость передачи.

2.1. Техническое задание

Процесс разработки приложения включает в себя следующие основные этапы:

– Формирование технического задания;

– Разработка;

– Тестирование;

Мною был сформировано техническое задание.

Клиентская часть должна требовать минимальное количество вычислительных ресурсов, иметь понятный и минималистичный интерфейс, понятный любому пользователю. Должны иметься возможность нахождения в разных каналах, отправка как личных сообщений, так и общих всем участникам канала. Так же должна быть возможность создать канал. Форма аутентификации и регистрации, создания и присоединения к каналам. Возможность использования криптографии при передаче данных.

Серверная часть должна состоять из консольной программы и файла локальной базы данных, в которой хранятся аутентификационные данные пользователя и данные созданных групп. Сервер должен поддерживать одновременное подключение нескольких клиентов и работу с ними без конфликтов. В консоль приложения программа должна отображать лог действий, происходящих на сервере в реальном времени, таких как подключение, авторизация, регистрация, создание групп, подключение к ним, отправитель, тип сообщения, получатель. Возможность использования криптографии при передаче данных.

Общие сведения о приложении: распределённое приложение «чат». Основной задачей приложения является удобный обмен мгновенными сообщениями с возможностью выбора получателя. Так же должна быть предусмотрена возможность разделения пользователей на группы или комнаты. Интерфейс должен быть интуитивно понятен и лаконичен.

Серверная часть должна быть максимально производительной и иметь информативные логи. Так же должен быть предусмотрен способ защищённой передачи данных от клиента серверу и обратно.

Цели и задачи приложения:

– Возможность обмена мгновенными сообщениями;
– Защищённость данных от перехвата;
– Кроссплатформенность клиентской и серверной части;

Целевая аудитория: пользователи персональных компьютеров и носимой электроники.

Структура серверной части проекта:

– Консольное приложение сервера;
– Файл с базой данных;

Структура клиентской части проекта:

– Страница с формой авторизации и регистрации;
– Страница с формой создания комнаты;
– Страница с формой подключения к комнате;
– Страница с формой обмена мгновенными сообщениями;
– Страница, на которой отображаются пользователи, находящиеся в текущей комнате**.**

2.2. Структура приложения

Файловые структуры не собранных в исполняемые файлы проектов представлена на рисунках ниже (Рисунок 4 и Рисунок 55) .

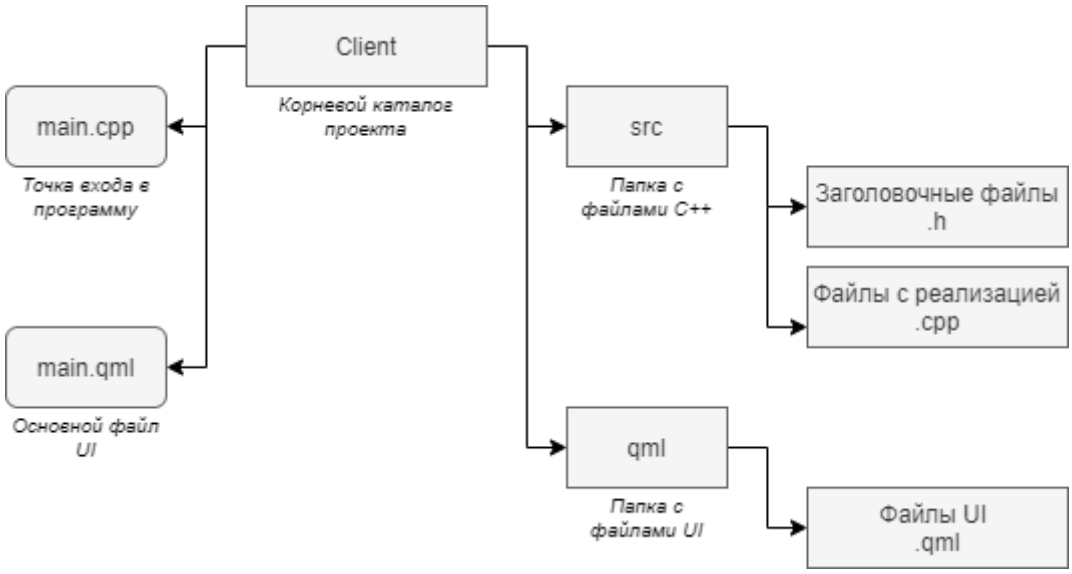

Рисунок 4 - Структура проекта клиентского приложения

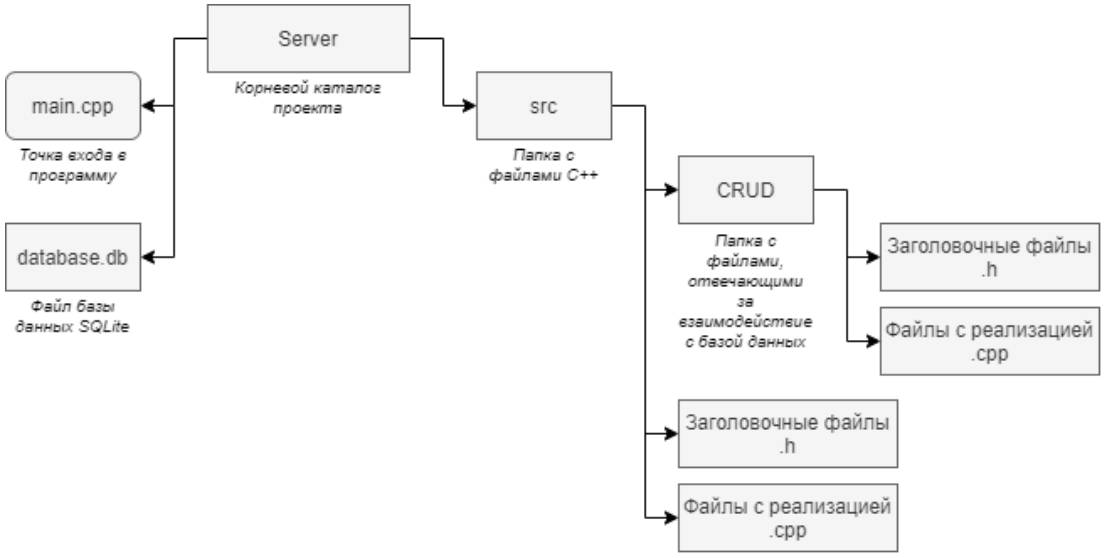

Рисунок 5 - Структура проекта серверного приложения

2.3.1. Архитектура клиентского приложения

Во время проектирования архитектуры клиентского приложения было учтено то, что основной задачей является обеспечить быстрое взаимодействие пользователя с данными и их отображение. Исходя из этого был выбран паттерн проектирования Model View Controller (MVC). Основной идеей этого шаблона является разделение хранения данных, логики взаимодействия пользователя с ними и их отображения. Со схематическим представлением этого шаблона можно ознакомиться ниже на Рисунке 6.

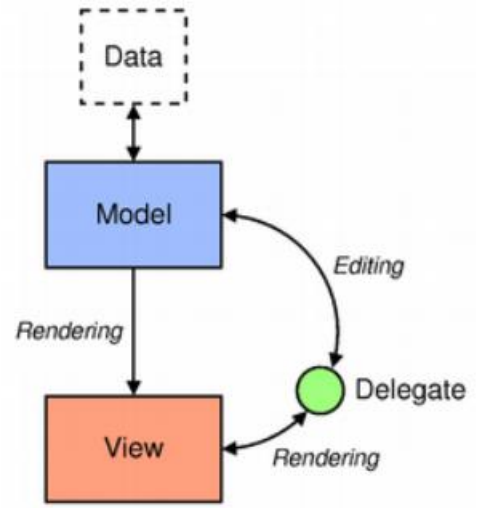

Рисунок 6 - Model View Controller

Рассмотрим компоненты данного паттерна.

– Model (модель) отвечает за хранение и правильный доступ к данным.

– View (представление) отображает данные в пользовательском интерфейсе и обеспечивает правильное взаимодействие с моделью.

– Delegate (делегат) является элементом представления и определяет то, как именно будут отображаться данные.

Паттерн Model-View является основополагающим при проектировании графического приложения в фреймворке Qt. За доступ, хранение и управление данными отвечает ядро программы, реализованное на C++. За представление этих данных же отвечает часть, написанная на QML. При разработке использовался большой раздел об разработке Qt приложений по паттерну Model – View из книги Макса Шлее «Qt 5.10. Профессиональное программирование на C++»[3].

Приложение состоит из трёх основных модулей.

Модуль, отвечающий за хранение данных о текущей сессии чата. Там находятся отправленные и полученные сообщения. Так же оно отвечает за хранение данных о текущем пользователе. Реализовано в классе ChatModel.

Модуль, который отображает подключённых в данный момент пользователей к текущей комнате – класс UserListModel.

Модуль, в котором происходит защищённое взаимодействие с сервером по протоколу TCP. Реализовано классом TCPClient. Для того, чтобы в программе существовал только один объект данного класса, при его реализации использовался паттерн Singleton (одиночка).

Обе модели – ChatModel и UserListModel содержат в себе указатель на объект TCPClient, чтобы независимо друг от друга отправлять и получать запросы к серверу.

2.3.2. Архитектура серверного приложения

Сервер должен взаимодействовать с клиентами и базой данных. Для работы с ней было принято решение использовать принцип CRUD – взаимодействие с данными основывается на четырёх операциях: Create, Read, Update, Delete. В языке SQL эти операции являются операторами INSERT, SELECT, UPDATE и DELETE. Этот функционал был вынесен в отдельный модуль, к которому модуль взаимодействия с клиентами имеет доступ через указатель.

Класс, отвечающий за взаимодействие других модулей с базой данных реализован с помощью паттерна Singleton (одиночка), чтобы гарантировать одну точку подключения и доступа к данным.

2.4. База данных

В качестве СУБД мною была выбрана SQLite. Это локальная база данных, которой не нужен сервер. Все данные хранятся в одном файле. При проектировании, построении и написании запросов пользовался официальный сайт [4]. База данных имеет очень простую структуру. Имеются три таблицы: Group_, User_ и Message_. В первой хранится информация о созданных комнатах. Во второй – о зарегистрированных клиентах. Таблица Message_ хранит информацию о всех сообщениях, прошедших через сервер: отправитель, получатель, комната и содержимое сообщения. Реляционная схема ниже (Рисунок 7).

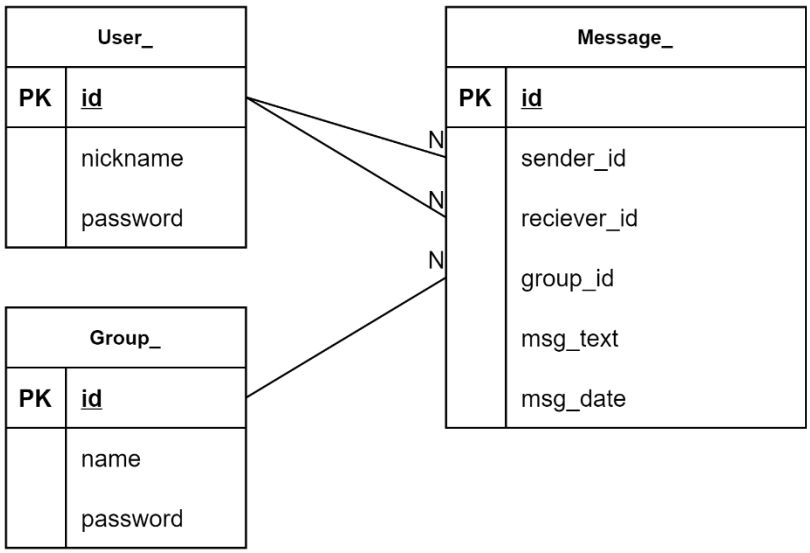

Рисунок 7 - Структура базы данных

2.5.1. Выбор формата сообщений между клиентом и сервером

Прежде всего стоит определиться с тем, каким образом будут формироваться сообщения перед отправкой по протоколу TCP. Для того, чтобы выделять отдельное сообщение из всех данных, которые поступили в сокет, будем добавлять в начало отправляемого пакета его размер. После него будем указывать тип данного сообщения. В качестве типа будем использовать восьмибитное число. Ниже привожу все доступные типы сообщений:

```
enum MessageType : quint8
{
    USER_JOIN = 1,
    USER_LEFT,
    USERS_LIST,
    USERS_LIST_REQUEST,
    PUBLIC_MESSAGE,
    PRIVATE_MESSAGE,
    PRIVATE_MESSAGE_FAIL,
    AUTH_REQUEST,
    AUTH_SUCCESS,
    AUTH_FAIL,
    REGISTER_REQUEST,
    REGISTER_SUCCESS,
    REGISTER_FAIL,
    JOIN_GROUP_REQUEST,
    JOIN_GROUP_SUCCESS,
    JOIN_GROUP_FAIL,
    LEAVE_GROUP_REQUEST,
    LEAVE_GROUP_SUCCESS,
    LEAVE_GROUP_FAIL,
    CREATE_GROUP_REQUEST,
    CREATE_GROUP_SUCCESS,
    CREATE_GROUP_FAIL
};
```

После типа сообщения будет указываться получатель, если сообщение адресовано конкретному пользователю. После будет записано тело сообщения. На вход функция получает константные ссылки на переменные, содержащие в себе тип сообщения и список строк с параметрами, которые будут записаны по порядку после типа сообщения. В начало вставляется вычисляемый размер сообщения. Т.к. данные передаются в виде набора байт,

используется встроенный класс QByteArray и класс для облегчения работы с ним QDataStream.

Для передачи таких сообщений используется класс QTcpSocket, который реализует абстрактный интерфейс сетевого взаимодействия по протоколу TCP/IP. При ознакомлении с работой с данным классом использовался пост «Клиент-серверный чат, используя сокеты Qt/C++» с электронного ресурса «Habr» [5] и примеры из официальной документации к Qt [1]. Для защиты передаваемой информации использовалась модифицированная версия этого класса, обеспечивающая защиту данных при передаче. Для отправки сообщения необходимо сначала сформировать массив байт по ранее рассмотренному формату и передать его в метод QTcpSocket::write(const QByteArray &data) сокета, который подключён к сокету на принимающей стороне.

Принимающий сокет испускает сигнал QTcpSocket::readyRead(), к которому подключён слот класса – обёртки TCPClient::onReadyRead(). Внутри этого слота происходит разбор полученных данных. После этого, в зависимости от его типа, вызываются другие методы. Реализация этого метода на сервере и клиенте не отличается ничем, кроме обрабатываемых типов сообщений.

2.5.2. Реализация серверной части

В качестве основного модуля реализован класс TCPServer, являющийся наследником встроенного класса QTcpServer. Используется перегрузка метода virtual void incomingConnection(qintptr handle) для того, чтобы корректно обрабатывать входящие подключения.

Класс содержит приватные поля:

– QHash<quintptr, TCPClient*> not_auth_clients – хеш-таблица, содержащая сокеты не авторизированных пользователей;

– QHash<QString, Group> groups – хеш-таблица, содержащая объекты чат-комнат;

– CRUD::Processor *crud_processor – указатель на объект, через который происходит взаимодействие с базой данных;

Класс TCPClient служит для обёртки стандартного класса QTcpSocket. Не отличается от реализации на клиентской стороне. Содержит в себе служебные приватные поля, динамический объект QTcpSocket, имя пользователя и название группы, в которой он состоит.

При входящем подключении к TCPServer, т.е. при попытке некого сокета подключиться к серверу, вызывается слот void TCPServer::incomingConnection(qintptr handle).

В метод в качестве параметра передаётся дескриптор подключающегося сокета. С помощью него создаётся ответный сокет, находящийся в классе TCPClient, производятся нужные соединения сигналов и слотов, далее объект new_client добавляется в хеш-таблицу не авторизированных клиентов.

После этого от клиента ожидается авторизация. При получении сообщения с запросом авторизации из него берутся никнейм пользователя и хэш пароля и вызывается метод попытки авторизации.

В нём первым делом производится поиск полученного никнейма по всем группам чтобы один пользователь не мог зайти одновременно с двух разных устройств. Далее производится запрос к базе данных, проверяющий существование записи с соответствующими данными. При успехе клиент, отправивший запрос, перемещается в хеш-таблицу клиентов группы “None”. При отрицательном результате поиска по базе данных клиенту отправляется сообщение о неудачной авторизации, и он остаётся в контейнере не авторизированных пользователей. Так же в консоль приложения выводятся соответствующие сообщения.

Так же от клиента может прийти запрос на вход в комнату, содержащий в себе её название и хешированный пароль. Для реализации групп объекты TCPClient раскладываются по соответствующим объектам Group и сообщения между ними происходят только в рамках этой комнаты.

```
struct Group
{
    Group() : name {"None"}, clients {}
    { }

    Group(const QString &name) :
        name {name}
    { }

    QString name;
    QHash<QString, TCPClient*> clients;
};
```

Структура Group содержит в себе своё название и хеш-таблицу с подключёнными к ней клиентами.

При получении запроса на вход в комнату вызывается слот TCPServer::onJoinGroupRequest.

Реализация похожа на метод авторизации клиента. Сначала проверяется существование такой группы, потом к базе отправляется запрос для проверки соответствия названия и пароля. В зависимости от результата, клиент либо помещается в нужную комнату и ему отправляется сообщение об успехе, либо остаётся в группе «None» и получает сообщение о неудаче.

Далее рассмотрим принцип работы сервера с базой данных. Главный объект в работе с ней – ConnectionManager. В этом классе происходит конфигурация, настройка и слежение за валидным состоянием подключения.

За работу с базой отвечает объект стандартного класса QSqlDatabase. Ему необходимо указать название драйвера, который работает с базой и параметры подключения. В случае СУБД SQLite нужно указать только путь к файлу базы. В данном методе сначала проверяется валидность драйвера, производятся некоторые сервисные действия. Далее происходит попытка подключиться к базе. Метод возвращает результат успешности этой попытки.

Все операции, входящие в CRUD, вызываются классом Processor. Далее, в зависимости типа операции (SELECT, INSERT и т.д.), вызываются методы классов Selector или Manipulator. Они отвечают за формирование запросов к базе. За SELECT отвечает Selector, за INSERT – Manipulator. Эти

запросы обрабатывает объект класса Executor. Ниже рассмотрим реализацию метода Executor::execute.

Сначала происходит проверка состояния подключения к базе. Далее обрабатываются параметры функции. queryText содержит текст запроса, список args хранит в себе значения, которые нужно подставить в запрос. Для исполнения запроса его необходимо обернуть в класс QSqlQuery. В объект этого класса нужно передать текст запроса, после методом QSqlQuery::bindValue в запрос подставляются значения. При вызове метода QSqlQuery::exec запрос выполняется в текущей подключённой базе данных, и в этом же объекте оказывается возвращаемое значение запроса – некоторая таблица. Доступ к нему можно получить, поочерёдно перебирая записи таблицы методом QSqlQuery::next и обрабатывая их методом QSqlQuery::value. Метод имеет две перегрузки. В первой значение поля записи можно получить по его индексу по порядку. Во второй – по названию.

При работе серверная программа выводит в консоль протоколы всех происходящих событий. Пример приведён ниже (Рисунок 8).

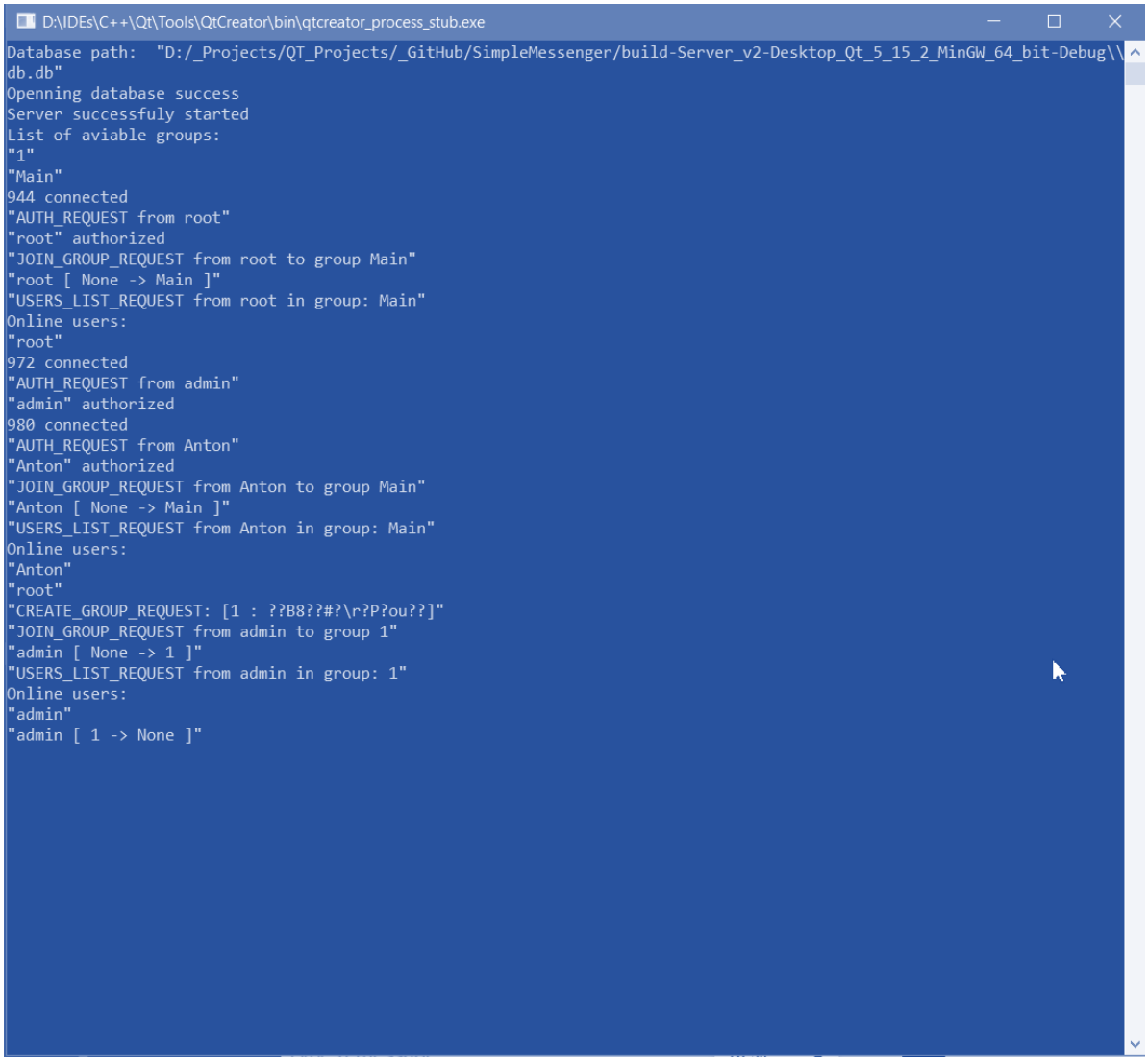

Рисунок 8 - Протоколы сервера

2.5.3. Реализация клиентской части

Главным классом клиентского приложения является модель данных чата ChatModel. Объект данного класса отвечает за взаимодействие с пользователем. Класс имеет довольно много различных методов, но самое основное – приватные поля данного класса.

– QList <MessageItem> m_messages_list – список сообщений, полученных и отправленных в текущую сессию и отображаемых в интерфейсе.

– TCPClient *client – объект, содержащий в себе клиентский сокет, через который происходит клиент – серверное взаимодействие. Реализация совпадает с серверной частью.

– QString m_nickname – никнейм текущего пользователя. Получается при успешной авторизации от сервера.

– QString m_group – название комнаты, в которой в данный момент находится клиент. Так же получается от сервера при успешном подключении к комнате.

– bool m_isAuth и bool m_isJoined – флаги, указывающие на то, авторизован ли пользователь и подключён ли он к некоторой комнате.

Далее будет рассмотрим реализацию пользовательского интерфейса – UI. Основным его элементом является объект ApplicationWindow – главное окно, в котором будут отображаться элементы интерфейса. Находится этот объект в основном файле main.qml. Ниже приведён код.

```
import QtQuick 2.13
import QtQuick.Controls 2.12
import QtQuick.Layouts 1.12

import ChatModel 1.0
import UserListModel 1.0
import "qml/"

ApplicationWindow {
    id: root
    visible: true
    width: 600
    height: 800
```

```
    ChatModel {
        id:  chat_model

        onIsAuthChanged: {
            if (isAuth && isJoined) {
                stack_view.clear()
                stack_view.push("qml/ChatPage.qml")
            }
            else if (isAuth && !isJoined) {
                stack_view.clear()
                stack_view.push("qml/GroupJoinPage.qml")
            }
            else {
                stack_view.clear()
                stack_view.push("qml/LoginPage.qml")
            }
        }

        onIsJoinedChanged: {
            if (isAuth && isJoined) {
                stack_view.clear()
                stack_view.push("qml/ChatPage.qml")
            }
            else if (isAuth && !isJoined) {
                stack_view.clear()
                stack_view.push("qml/GroupJoinPage.qml")
            }
            else {
                stack_view.clear()
                stack_view.push("qml/LoginPage.qml")
            }
        }
    }

    UserListModel {
        id: users_model
    }

    StackView {
        id: stack_view
        anchors.fill: parent
        clip: false
        initialItem: LoginPage {}
    }
}
```

У ApplicationWindow указываются начальные значения свойств ширины и высоты. Внутри создаётся объект ChatModel, с будет происходить взаимодействие по схеме Model – View. Определяются необходимые

действия, происходящие при изменении значения флагов. Далее создаётся объект UserListModel, который отвечает за отображение списка подключённых пользователей. После чего создаётся элемент графического интерфейса StackView, который будет контролировать отображение одной из выбранных страниц приложения. При разработке UI для компонентов управления использовался стиль Material, часто используемый в приложениях компании Google. Для его использования необходимо подключить модуль quickcontrols2 в .pro файле проекта, с помощью вызова QQuickStyle::setStyle("Material"); указать соответствующий стиль в main.cpp и подключить модуль QtQuick.Controls.Material. Рассмотрим формы авторизации (Рисунок 9) и регистрации (Рисунок 10).

Рисунок 9 - Форма авторизации

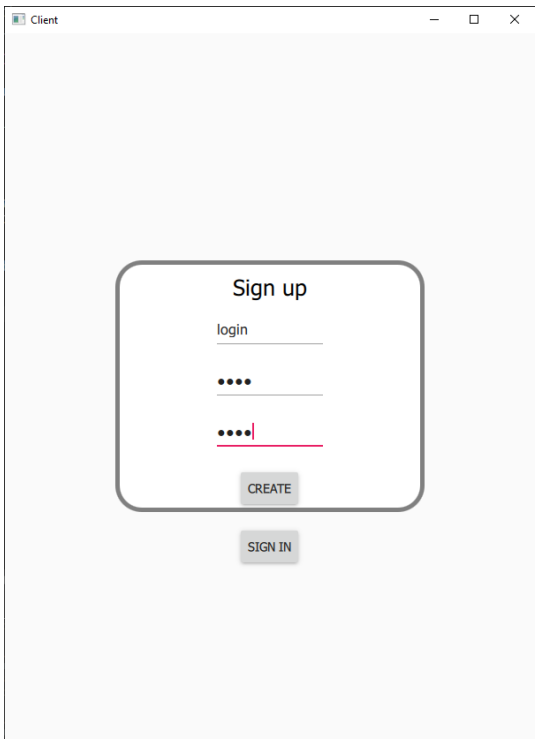

Рисунок 10 - Форма регистрации

При успешной авторизации пользователь попадает на форму присоединения и создания комнаты. Пример ниже (Рисунок 111).

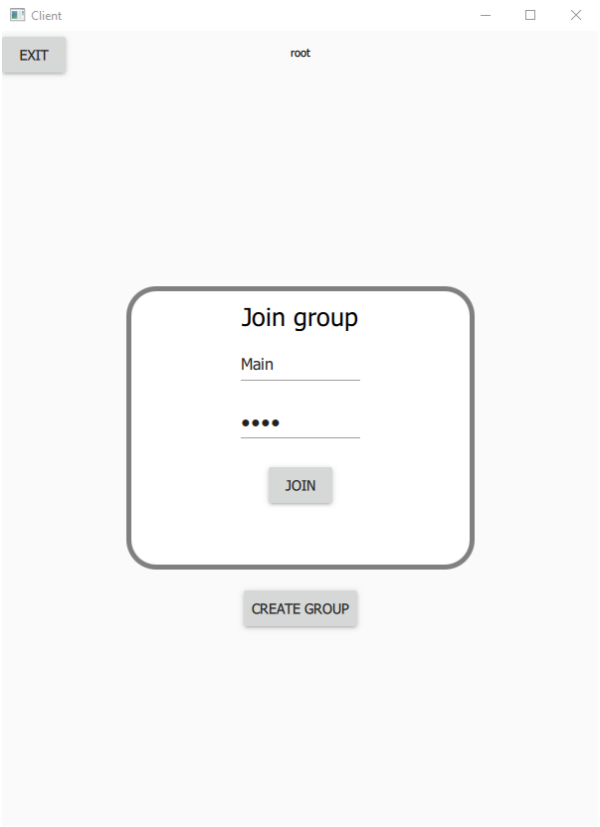

Рисунок 11 - Форма выбора комнаты

После успешного подключения к комнате пользователь переходит на страницу с чатом (Рисунок *12*12). При подключении или отключении его от комнаты пользователя всем остальным приходит соответствующее сообщение.

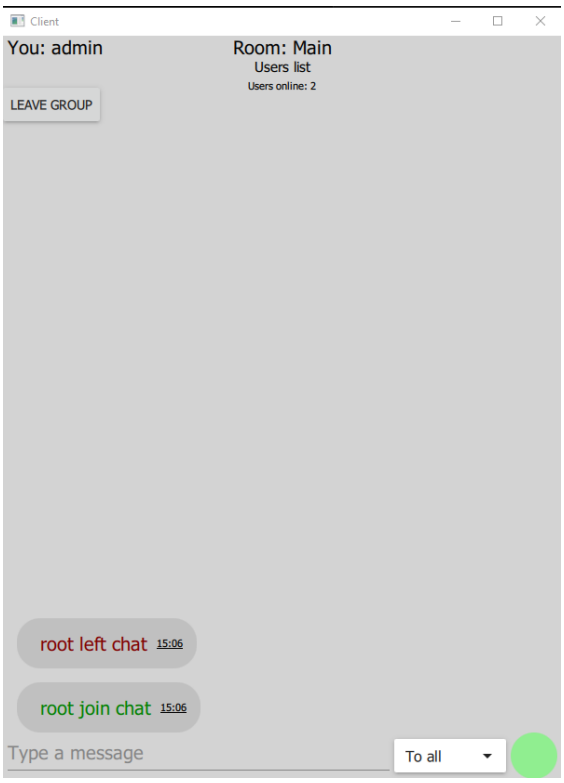

Рисунок 12 - Страница чата

В верхней части формы отображается никнейм, название текущей комнаты, количество подключённых к ней пользователей. При нажатии на надпись «Users list» откроется окно со списком подключённых пользователей, выполненный в простейшем дизайне. Так же на странице чата имеется combobox с выбором типа сообщения (Рисунок *13*13). При выборе приватного появляется второй combobox с выбором адресата (Рисунок *14*14).

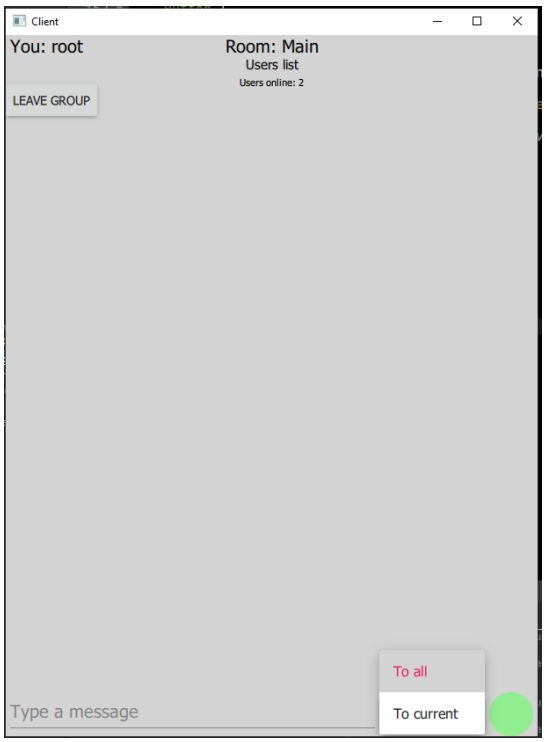

Рисунок 13 - Виды сообщений

Рисунок 14 - Виды сообщений

Отображение сообщений можно увидеть ниже (Рисунок *15*15).

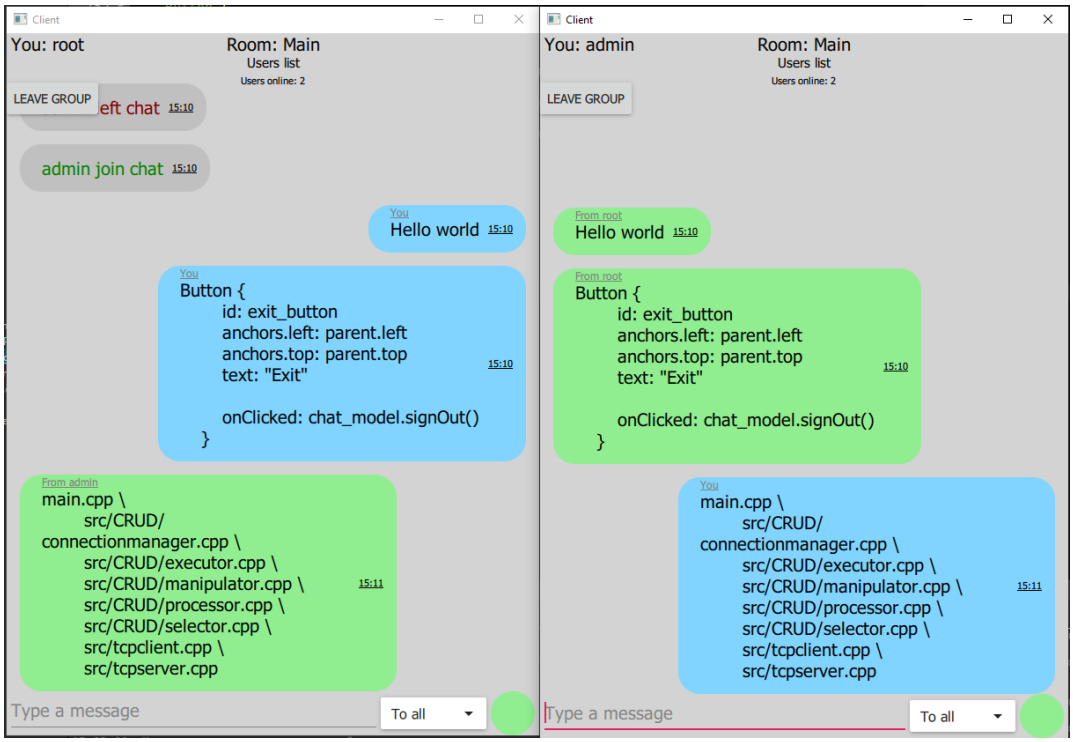

Рисунок 15 - Отображение сообщений

Для входящих и исходящих сообщений предусмотрен разный цвет и положение на форме. В сообщении указывается его текст, отправитель и дата.

ЗАКЛЮЧЕНИЕ

Благодаря гибкости и разнообразию фреймворка Qt в частности, и эффективности C++ стандарта 2017 года в целом, даже без использования многопоточного программирования можно быстро и качественно разрабатывать достаточно комплексные и сложные проекты. Изучив клиент – серверную архитектуру и сетевое взаимодействие, разработчику не составит труда средствами протокола TCP/IP и встроенных классов Qt реализовать распределённое приложение любой сложности. Благодаря наличию огромного количества встроенных в фреймворк классов и официальной документации появляется возможность без особых трудностей начинить разрабатываемую программу любым необходимым функционалом, который, благодаря скорости языка C++ и кроссплатформенности Qt, будет работать даже на самых слабых машинах и почти любой операционной системе и платформе. Так же, изучив базовые принципы криптографии, симметричных и асимметричных шифров можно без труда организовать защищённый обмен данными между клиентом и сервером.

Шифр компетенции	Расшифровка приобретаемой компетенции	Расшифровка освоения компетенции
ОК-8	способность к самоорганизации и самообразованию	Работа была декомпозирована и обозначена на определённые интервалы времени.
ОПК-2	способность применять соответствующий математический аппарат для решения профессиональных задач	В работе были выделены математические задачи и решены с помощью соответствующих математических аппаратов.
ОПК-5	способность использовать нормативные правовые акты в профессиональной деятельности	Проведено ознакомление с соответствующими правовыми актами.
ОПК-6	способность применять приемы оказания первой помощи, методы и средства защиты персонала предприятия и населения в условиях чрезвычайных ситуаций, организовать мероприятия по охране труда и технике безопасности	Были соблюдены все нормы карантинных мероприятий во время написания курсовой работы.

СПИСОК ЛИТЕРАТУРЫ

1. Qt Documentation [Электронный ресурс]. – Режим доступа: https://doc.qt.io/, свободный (Дата обращения 01.05.2020)

2. OpenSSL Cryptography and SSL/TLS Toolkot[Электронный ресурс]. Режим доступа: https://www.openssl.org/, свободный. (Дата обращения 01.05.2020)

3. Книга: Qt 5.10. Профессиональное программирование на C++. Автор: Шлее Макс.

4. Сайт sqlite.org [Электронный ресурс]. – Режим доступа: https://www.sqlite.org/index.html.

5. Сайт habr.com [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/131585/.

ПРИЛОЖЕНИЯ

Основной программный код серверной части.

.pro файл:

```
QT -= gui
QT += core sql network

CONFIG += c++17 console
CONFIG -= app_bundle

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000 # disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
        main.cpp \
        src/CRUD/connectionmanager.cpp \
        src/CRUD/executor.cpp \
        src/CRUD/manipulator.cpp \
        src/CRUD/processor.cpp \
        src/CRUD/selector.cpp \
        src/tcpclient.cpp \
        src/tcpserver.cpp

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

HEADERS += \
    src/CRUD/connectionmanager.h \
```

```
    src/CRUD/crudmapper.h \
    src/CRUD/crudtypes.h \
    src/CRUD/executor.h \
    src/CRUD/manipulator.h \
    src/CRUD/processor.h \
    src/CRUD/selector.h \
    src/group.h \
    src/tcpclient.h \
    src/tcpserver.h \
    src/types.h
```

main.cpp:

```
#include <QCoreApplication>
#include "src/tcpserver.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    Server::TCPServer myserver;
    myserver.start();
    return a.exec();
}
```

tcpserver.h:

```
#pragma once

#include <QObject>
#include <QTcpServer>
#include <QTcpSocket>
#include <QDataStream>
#include <QByteArray>
#include <QHash>
#include <QCryptographicHash>
#include <QDebug>
```

```
#include "tcpclient.h"
#include "group.h"
#include "CRUD/processor.h"

namespace Server
{
class TCPServer : public QTcpServer
{
    Q_OBJECT
public:
    explicit TCPServer(QObject *parent = nullptr);

    virtual void incomingConnection(qintptr handle) override;
    bool start();
    void stop();

private slots:
    void onRegisterRequest(quintptr handle, QString name, QString password);
    void onAuthRequest(quintptr handle, QString name, QString password);
    void onClientDisconnected(QString name);

    void onCreateGroupRequest(QString client_name, QString group_name, QString group_password);
    void onJoinGroupRequest(QString client_name, QString group_name, QString group_password);
    void onLeaveGroupRequest(QString client_name, QString group_name);

    void onPublicMessage(QString sender, QString group_name, QString msg);
    void onPrivateMessage(QString sender, QString receiver, QString group_name, QString msg);
```

```
        void onUsersListRequest(QString client_name, QString
group_name);


    private:

        static QByteArray makeByteArray(const quint8 &msg_type,
const QStringList &params);

        static QByteArray makeByteArray(const quint8 &msg_type,
const QString &param = {});


        QHash<quintptr, TCPClient*> not_auth_clients;

        QHash<QString, Group>       groups;

        CRUD::Processor             *crud_processor;


    };

    }
```

tcpclient.h:

```
    #pragma once

    #include <QTcpSocket>
    #include <QSslSocket>
    #include <QObject>
    #include <QSslKey>

    #include "types.h"
    #include "tcpserver.h"

    namespace Server
    {
    class TCPClient : public QObject
    {
        Q_OBJECT
        friend class TCPServer;

    public:
        TCPClient(quintptr handle);

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

    signals:
        void authRequest(quintptr handle, QString name,
QString password);
```

```
        void registerRequest(quintptr handle, QString name,
QString password);
        void clientDisconnected(QString name);

        void createGroupRequest(QString client_name, QString
group_name, QString group_password);
        void joinGroupRequest(QString client_name, QString
group_name, QString group_password);
        void leaveGroupRequest(QString client_name, QString
group_name);

        void publicMessage(QString sender, QString
group_name, QString msg);
        void privateMessage(QString sender, QString receiver,
QString group_name, QString msg);

        void usersListRequest(QString name, QString group);

    private:
        QSslSocket  *socket;
        quint16     block_size;
        quintptr    handle;
        QString     name;
        QString     current_group;
    };
    }
```

CRUD/processor.h:

```
#pragma once

#include <memory>
#include <QMutex>
#include <QDebug>

#include "selector.h"
#include "manipulator.h"

namespace CRUD
{
class ProcessorPrivate
{
public:
    Manipulator manipulator;
    Selector selector;
};

class Processor
{
public:
    Q_DISABLE_COPY(Processor)
    ~Processor();

    static Processor *instance();
```

```
        std::pair<RESULT, std::vector<QVariantList>>
requestTableData(TABLES table);
        bool userExist(const QString &nickname);
        bool groupExist(const QString &group_name);
        bool registerUser(const QString &nickname, const QString
&password);
        bool registerGroup(const QString &group_name, const
QString &group_password);
        bool checkUserPassword(const QString &nickname, const
QString &password);
        bool checkGroupPassword(const QString &group_name, const
QString &group_password);
        QStringList requestGroupsList();

    private:
        Processor();
        std::unique_ptr<ProcessorPrivate> m_processor_private;
    };
    }
```

Клиентская часть.

.pro файл:

```
    QT += quick core network quickcontrols2

    CONFIG += c++14

    # The following define makes your compiler emit warnings if
you use
    # any Qt feature that has been marked deprecated (the exact
warnings
    # depend on your compiler). Refer to the documentation for
the
    # deprecated API to know how to port your code away from it.
    DEFINES += QT_DEPRECATED_WARNINGS

    # You can also make your code fail to compile if it uses
deprecated APIs.
    # In order to do so, uncomment the following line.
    # You can also select to disable deprecated APIs only up to
a certain version of Qt.
    #DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    #
disables all the APIs deprecated before Qt 6.0.0

    SOURCES += \
            main.cpp \
            src/chatmodel.cpp \
            src/tcpclient.cpp \
            src/userlistmodel.cpp

    RESOURCES += qml.qrc
```

```
    # Additional import path used to resolve QML modules in Qt
Creator's code model
    QML_IMPORT_PATH =

    # Additional import path used to resolve QML modules just
for Qt Quick Designer
    QML_DESIGNER_IMPORT_PATH =

    # Default rules for deployment.
    qnx: target.path = /tmp/$${TARGET}/bin
    else: unix:!android: target.path = /opt/$${TARGET}/bin
    !isEmpty(target.path): INSTALLS += target

    HEADERS += \
        src/chatmodel.h \
        src/messageitem.h \
        src/tcpclient.h \
        src/types.h \
        src/userlistitem.h \
        src/userlistmodel.h
```

main.cpp:

```
    #include <QGuiApplication>
    #include <QQmlApplicationEngine>
    #include <QQmlContext>
    #include <QQuickStyle>

    #include "src/tcpclient.h"
    #include "src/chatmodel.h"
    #include "src/userlistmodel.h"

    int main(int argc, char *argv[])
    {

QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);

        QGuiApplication app(argc, argv);

        QQuickStyle::setStyle("Material");
        QQmlApplicationEngine engine;

        qmlRegisterType<ChatModel>("ChatModel", 1, 0,
"ChatModel");
        qmlRegisterType<UserListModel>("UserListModel", 1, 0,
"UserListModel");

        const QUrl url(QStringLiteral("qrc:/main.qml"));
        QObject::connect(&engine,
&QQmlApplicationEngine::objectCreated,
                         &app, [url](QObject *obj, const QUrl
&objUrl) {
            if (!obj && url == objUrl)
                QCoreApplication::exit(-1);
```

```
        }, Qt::QueuedConnection);
        engine.load(url);

        return app.exec();
    }
```

chatmodel.h:

```
    #pragma once

    #include "tcpclient.h"

    #include <QAbstractListModel>
    #include <QTime>
    #include <QCryptographicHash>
    #include <QByteArray>
    #include <QHostAddress>

    #include "messageitem.h"


    class ChatModel : public QAbstractListModel
    {
        Q_OBJECT
        Q_PROPERTY(bool    isAuth   READ isAuth   NOTIFY
isAuthChanged)
        Q_PROPERTY(bool    isJoined READ isJoined NOTIFY
isJoinedChanged)
        Q_PROPERTY(QString group    READ group    NOTIFY
groupChanged)
        Q_PROPERTY(QString nickname READ nickname NOTIFY
nicknameChanged)

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

        enum Roles
        {
            SenderRole = Qt::UserRole + 1,
            MessageRole,
            TimeRole,
            IsMyRole,
            FontColorRole,
            BackColorRole
        };

        int rowCount(const QModelIndex &parent = QModelIndex())
const override;
        QVariant data(const QModelIndex &index, int role =
Qt::DisplayRole) const override;
        QHash<int, QByteArray> roleNames() const override;

        bool isAuth() const      { return m_isAuth;   }
        bool isJoined() const    { return m_isJoined; }
```

```
        QString nickname() const { return m_nickname; }
        QString group() const    { return m_group;    }

    public slots:
        void sendPrivateMsg(const QString &reciever, const
QString &message);
        void sendPublicMsg(const QString &message);

        void signUp(const QString &nickname, const QString
&password);
        void signIn(const QString &nickname, const QString
&password);
        void signOut();
        void joinGroup(const QString &group_name, const QString
&password);
        void leaveGroup();
        void createGroup(const QString &group_name, const
QString &password);

    signals:
        void isAuthChanged(bool isAuth);
        void isJoinedChanged(bool isJoined);
        void nicknameChanged(QString nickname);
        void groupChanged(QString group);
        void gs();

    private slots:
        void onPublicMessageRecieved (QString sender, QString
message);
        void onPrivateMessageRecieved(QString sender, QString
reciever, QString message);

        void onUserJoinRecieved(QString user);
        void onUserLeftRecieved(QString user);

        void onAuthSuccess(QString nickname);
        void onAuthFail(QString error);
        void onRegisterSuccess(QString nickname);
        void onRegisterFail(QString error);

        void onJoinGroupSuccess(QString group);
        void onJoinGroupFail(QString error);
        void onLeaveGroupSuccess();


    private:
        void addMsgToList(const MessageItem &msg_item);
        QByteArray hashPassword(const QString &password) const;

        QList <MessageItem> m_messages_list;
        TCPClient           *client;
        QString             m_nickname;
        QString             m_group;
```

```
        bool                m_isAuth;
        bool                m_isJoined;
    };
```

userlistmodel.h:

```
    #pragma once

    #include <QAbstractListModel>
    #include <QList>

    #include "userlistitem.h"
    #include "tcpclient.h"

    class UserListModel : public QAbstractListModel
    {
        Q_OBJECT
        Q_PROPERTY(int usersOnline READ usersOnline WRITE
setUsersOnline NOTIFY usersOnlineChanged)

    public:
        explicit UserListModel(QObject *parent = nullptr);

        enum Roles
        {
            NicknameRole = Qt::UserRole + 1,
            IsOnlineRole,
            ColorRole
        };

        int rowCount(const QModelIndex &parent = QModelIndex())
const override;
        QVariant data(const QModelIndex &index, int role =
Qt::DisplayRole) const override;
        QHash<int, QByteArray> roleNames() const override;
        int usersOnline() const { return m_users_online; }

    public slots:
        void setUsersOnline(int users_online);

    signals:
        void usersOnlineChanged(int users_online);

        void test();

    private slots:
        void onUserJoinRecieved(QString nickname);
        void onUserLeftRecieved(QString nickname);
        void onUsersListRecieved(QStringList users_list);

    private:
        QList<UserListItem> m_users_list;
        TCPClient           *tcp_client;
        int                 m_users_online;
```

```
};
```

userlistitem.h:

```
#pragma once

#include <QString>
#include <QColor>

struct UserListItem
{
    UserListItem() : nickname {""}, online {false}
    { }

    UserListItem(const QString &nickname, const QColor
color, const bool &online) :
        nickname {nickname}, color {color}, online {online}
    { }

    bool operator==(const UserListItem &item1)
    {
        return item1.nickname == nickname;
    }

    bool operator==(const QString &nick)
    {
        return nickname == nick;
    }

    friend bool operator==(const QString &nick, const
UserListItem &item)
    {
        return nick == item.nickname;
    }

    QString nickname;
    QColor  color;
    bool    online;
};
```

messageitem.h:

```
#pragma once

#include <QString>
#include <QColor>

struct MessageItem
{
    MessageItem() :
        sender {""}, message {""}, fontColor {Qt::black},
time {"00:00"}, isMine {true}
    { }
```

```
        MessageItem(const QString &sender, const QString
&message, const QColor &fontColor,
                    const QColor &backColor, const bool &isMy =
false, const QString &time = "00:00") :
            sender {sender}, message {message}, fontColor
{fontColor},
            backColor {backColor}, time {time}, isMine {isMy}
        { }

        QString sender;
        QString message;
        QColor  fontColor;
        QColor  backColor;
        QString time;
        bool    isMine;
    };
```

tcpclient.h:

```
    #pragma once

    #include <QSslSocket>
    #include <QTcpSocket>
    #include <QDebug>
    #include <vector>

    class TCPClient : public QObject
    {
        Q_OBJECT
        Q_DISABLE_COPY(TCPClient)
        friend class UserListModel;
        friend class ChatModel;

    public:
        static TCPClient* instance();

        void sendPublicMsg(const QString &message_text);
        void sendPrivateMsg(const QString &reciever, const
QString &message_text);

        void singUp(const QHostAddress &host, const int &port,
const QString &nickname, const QString &password);
        void signIn(const QHostAddress &host, const int &port,
const QString &nickname, const QString &password);
        void signOut();
        void joinGroup(const QString &group, const QString
&password);
        void leaveGroup();
        void createGroup(const QString &group, const QString
&password);

        static QByteArray makeByteArray(const quint8 &msg_type,
const QStringList &params);
```

```
        static QByteArray makeByteArray(const quint8 &msg_type,
const QString &param = {});

    private slots:
        void onReayRead();
        void onDisconnected();

    signals:
        void publicMsgRecieved (QString sender, QString
message);
        void privateMsgRecieved(QString sender, QString
reciever, QString message);

        void userJoinRecieved(QString sender);
        void userLeftRecieved(QString sender);
        void usersListRecieved(QStringList users_list);

        void authSuccess(QString nickname);
        void authFail(QString error);
        void registerSuccess(QString nickname);
        void registerFail(QString nickname);

        void joinGroupSuccess(QString group);
        void joinGroupFail(QString error);

        void leaveGroupSuccess();

    private:
        TCPClient();

        QSslSocket  *socket;
        QByteArray  data;
        QString     name;
        QString     current_group;
        quint16     block_size;
    };
```

main.qml:

```
import QtQuick 2.13
import QtQuick.Controls 2.12
import QtQuick.Layouts 1.12

import ChatModel 1.0
import UserListModel 1.0
import "qml/"

ApplicationWindow {
    id: root
    visible: true
    width: 600
    height: 800

    ChatModel {
```

```
        id:  chat_model

        onIsAuthChanged: {
            if (isAuth && isJoined) {
                stack_view.clear()
                stack_view.push("qml/ChatPage.qml")
            }
            else if (isAuth && !isJoined) {
                stack_view.clear()
                stack_view.push("qml/GroupJoinPage.qml")
            }
            else {
                stack_view.clear()
                stack_view.push("qml/LoginPage.qml")
            }
        }

        onIsJoinedChanged: {
            if (isAuth && isJoined) {
                stack_view.clear()
                stack_view.push("qml/ChatPage.qml")
            }
            else if (isAuth && !isJoined) {
                stack_view.clear()
                stack_view.push("qml/GroupJoinPage.qml")
            }
            else {
                stack_view.clear()
                stack_view.push("qml/LoginPage.qml")
            }
        }
    }

    UserListModel {
        id: users_model
    }

    StackView {
        id: stack_view
        anchors.fill: parent
        clip: false
        initialItem: LoginPage {}
    }
}
LoginPage.qml:
import QtQuick 2.13
import QtQuick.Layouts 1.12
import QtQuick.Controls 2.12
import QtQuick.Controls.Material 2.12

Page {
    id: root_page
    background: Rectangle {
```

```
            anchors.fill: parent
            color: "lightgray"
        }

        Button {
            id: exit_button
            anchors.left: parent.left
            anchors.top: parent.top
            anchors.topMargin: 50
            text: "Leave group"
            z: 5

            onClicked: {
                message_type_select.currentIndex = 0
                chat_model.leaveGroup()
                text_input.clear()
            }
        }

        Text {
            id: nickname_text
            text: "You: " + chat_model.nickname
            anchors.top: parent.top
            anchors.left: parent.left
            anchors.leftMargin: 5
            font.pixelSize: 20
        }

        Text {
            id: roomname_text
            text: "Room: " + chat_model.group
            anchors.top: parent.top
            anchors.horizontalCenter: parent.horizontalCenter
            font.pixelSize: 20
        }

        Text {
            id: users_list_text
            text: "Users list"
            anchors.top: roomname_text.bottom
            anchors.horizontalCenter:
roomname_text.horizontalCenter
            font.pixelSize: roomname_mousearea.pressed ? 17 : 15
            font.bold: roomname_mousearea.containsMouse

            MouseArea {
                id: roomname_mousearea
                anchors.fill: parent
                hoverEnabled: true

                onClicked: stack_view.push("UsersListPage.qml")
            }
        }
```

```
        Text {
            id: users_count
            text: "Users online: " + users_model.usersOnline
            anchors.horizontalCenter:
users_list_text.horizontalCenter
            anchors.top: users_list_text.bottom
            anchors.topMargin: 5
        }

        ListView {
            id: list_view
            anchors.fill: parent
            anchors.bottomMargin: 50
            anchors.topMargin: 50

            spacing: 15
            clip: true
            verticalLayoutDirection: ListView.BottomToTop

            model: chat_model

            delegate: MessageBox {
                isMy: isMy_
                sender: sender_
                message: message_
                time: time_
                fontColor: font_color
                backColor: back_color
            }
        }

        RowLayout {
            id: root_row
            anchors.bottom: parent.bottom
            width: parent.width

            TextField {
                id: text_input
                height: 50
                Layout.alignment: Qt.AlignCenter
                Layout.leftMargin: 5
                Layout.fillWidth: true
                font.pixelSize: 20

                placeholderText: "Type a message"
            }

            ComboBox {
                id: message_type_select

                model: ListModel {
```

```
                    ListElement {
                        text: "To all"
                    }

                    ListElement {
                        text: "To current"
                    }
                }
            }

            ComboBox {
                id: reciever_select
                property string selectedNickname: ""
                visible: message_type_select.currentIndex == 1 ?
true : false
                model: users_model

                delegate: Rectangle {
                    height: 50
                    width: parent.width
                    color: delegate_mousearea.containsMouse ?
"lightgray" : "transparent"

                    Text {
                        id: nickname_field
                        anchors.left: parent.left
                        anchors.verticalCenter:
parent.verticalCenter
                        anchors.leftMargin: 15

                        color: color_
                        text: nickname_ == chat_model.nickname ?
"You" : nickname_
                        font.pixelSize: 14
                    }

                    MouseArea {
                        id: delegate_mousearea
                        anchors.fill: parent
                        hoverEnabled: true

                        onClicked: {
                            reciever_select.selectedNickname =
nickname_field.text
                            reciever_select.displayText =
nickname_field.text
                            reciever_select.update()
                        }
                    }
                }
            }

            Rectangle {
```

```
                id: send_button
                color: mouse_area.containsMouse ? "green" :
"lightgreen"
                width: 50
                height: width
                radius: 25
                Layout.rightMargin: 5

                MouseArea {
                    id: mouse_area
                    anchors.fill: parent
                    enabled: text_input.length > 0

                    onPressed: {
                        if (message_type_select.currentIndex ==
0)

chat_model.sendPublicMsg(text_input.text)

                        else if
(reciever_select.selectedNickname.length > 0)

chat_model.sendPrivateMsg(reciever_select.selectedNickname,
text_input.text)

                        text_input.text = ""
                    }
                }
            }
        }
    }
```

ChatPage.qml:

```
import QtQuick 2.13
import QtQuick.Layouts 1.12
import QtQuick.Controls 2.12
import QtQuick.Controls.Material 2.12

Page {
    id: root_page
    background: Rectangle {
        anchors.fill: parent
        color: "lightgray"
    }

    Button {
        id: exit_button
        anchors.left: parent.left
        anchors.top: parent.top
        anchors.topMargin: 50
        text: "Leave group"
        z: 5
```

```
            onClicked: {
                message_type_select.currentIndex = 0
                chat_model.leaveGroup()
                text_input.clear()
            }
        }

        Text {
            id: nickname_text
            text: "You: " + chat_model.nickname
            anchors.top: parent.top
            anchors.left: parent.left
            anchors.leftMargin: 5
            font.pixelSize: 20
        }

        Text {
            id: roomname_text
            text: "Room: " + chat_model.group
            anchors.top: parent.top
            anchors.horizontalCenter: parent.horizontalCenter
            font.pixelSize: 20
        }

        Text {
            id: users_list_text
            text: "Users list"
            anchors.top: roomname_text.bottom
            anchors.horizontalCenter:
roomname_text.horizontalCenter
            font.pixelSize: roomname_mousearea.pressed ? 17 : 15
            font.bold: roomname_mousearea.containsMouse

            MouseArea {
                id: roomname_mousearea
                anchors.fill: parent
                hoverEnabled: true

                onClicked: stack_view.push("UsersListPage.qml")
            }
        }

        Text {
            id: users_count
            text: "Users online: " + users_model.usersOnline
            anchors.horizontalCenter:
users_list_text.horizontalCenter
            anchors.top: users_list_text.bottom
            anchors.topMargin: 5
        }

        ListView {
            id: list_view
```

```
        anchors.fill: parent
        anchors.bottomMargin: 50
        anchors.topMargin: 50

        spacing: 15
        clip: true
        verticalLayoutDirection: ListView.BottomToTop

        model: chat_model

        delegate: MessageBox {
            isMy: isMy_
            sender: sender_
            message: message_
            time: time_
            fontColor: font_color
            backColor: back_color
        }
    }

    RowLayout {
        id: root_row
        anchors.bottom: parent.bottom
        width: parent.width

        TextField {
            id: text_input
            height: 50
            Layout.alignment: Qt.AlignCenter
            Layout.leftMargin: 5
            Layout.fillWidth: true
            font.pixelSize: 20

            placeholderText: "Type a message"
        }

        ComboBox {
            id: message_type_select

            model: ListModel {

                ListElement {
                    text: "To all"
                }

                ListElement {
                    text: "To current"
                }
            }
        }

        ComboBox {
            id: reciever_select
```

```
                property string selectedNickname: ""
                visible: message_type_select.currentIndex == 1 ?
true : false
                model: users_model

                delegate: Rectangle {
                    height: 50
                    width: parent.width
                    color: delegate_mousearea.containsMouse ?
"lightgray" : "transparent"

                    Text {
                        id: nickname_field
                        anchors.left: parent.left
                        anchors.verticalCenter:
parent.verticalCenter
                        anchors.leftMargin: 15

                        color: color_
                        text: nickname_ == chat_model.nickname ?
"You" : nickname_
                        font.pixelSize: 14
                    }

                    MouseArea {
                        id: delegate_mousearea
                        anchors.fill: parent
                        hoverEnabled: true

                        onClicked: {
                            reciever_select.selectedNickname =
nickname_field.text
                            reciever_select.displayText =
nickname_field.text
                            reciever_select.update()
                        }
                    }
                }
            }

            Rectangle {
                id: send_button
                color: mouse_area.containsMouse ? "green" :
"lightgreen"
                width: 50
                height: width
                radius: 25
                Layout.rightMargin: 5

                MouseArea {
                    id: mouse_area
                    anchors.fill: parent
                    enabled: text_input.length > 0
```

```
                    onPressed: {
                        if (message_type_select.currentIndex ==
0)

chat_model.sendPublicMsg(text_input.text)

                        else if
(reciever_select.selectedNickname.length > 0)

chat_model.sendPrivateMsg(reciever_select.selectedNickname,
text_input.text)

                        text_input.text = ""
                    }
                }
            }
        }
    }
```

UserListPage.qml:

```
import QtQuick 2.13
import QtQuick.Controls 2.12
import QtQuick.Layouts 1.12

Page {
    id: root_page

    Button {
        anchors.left: parent.left
        anchors.top: parent.top
        text: "Back"

        onClicked: {
            stack_view.pop()
        }
    }

    Text {
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.top: parent.top
        anchors.topMargin: 5
        text: "Users online: " + users_model.usersOnline
    }

    ListView {
        id: list_model
        anchors.fill: parent
        anchors.topMargin: 50
        clip: true
        spacing: 10
        model: users_model
```

```
        delegate: UserBox {
            nickname: nickname_
            color: color_
        }
    }
}
```