**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

Оглавление

[**ВВЕДЕНИЕ** 3](#_Toc72346934)

[**ГЛАВА I. ТЕОРЕТИЧЕСКИЕ И МЕТОДОЛОГИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ КЛИЕНТ – СЕРВЕРНЫХ ПРИЛОЖЕНИЙ** 5](#_Toc72346935)

[**1.1.** **Основные понятия** 5](#_Toc72346936)

[**1.2. Кроссплатформенный фреймворк Qt** 6](#_Toc72346937)

[**1.3. Объектная модель. Сигналы и слоты** 10](#_Toc72346938)

[**1.4. Структура проекта в Qt** 11](#_Toc72346939)

[**1.5. Клиент – серверная (распределённая) архитектура.** 12](#_Toc72346940)

[**ГЛАВА II. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА КРОССПЛАТФОРМЕННОГО РАСПРЕДЕЛЁННОГО ПРИЛОЖЕНИЯ «ЧАТ»** 14](#_Toc72346941)

[**2.1. Техническое задание** 14](#_Toc72346942)

[**2.2. Структура приложения** 16](#_Toc72346943)

[**2.3.1. Архитектура клиентского приложения** 17](#_Toc72346944)

[**2.3.2. Архитектура серверного приложения** 19](#_Toc72346945)

[**2.4. База данных** 20](#_Toc72346946)

[**2.5. Этапы создания приложения** 21](#_Toc72346947)

[**ЗАКЛЮЧЕНИЕ** 37](#_Toc72346948)

[**СПИСОК ЛИТЕРАТУРЫ** 38](#_Toc72346949)

# **ВВЕДЕНИЕ**

**Актуальность.** 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.

Для достижения заданной цели выделим следующие **задачи**:

* Изучение предметной области;
* Анализ методов проектирования кроссплатформенных приложений;
* Реализация.

**Структура работы.** Работа будет состоять из двух глав. В первой затронем теоретический фундамент и методы подхода. Во второй главе будут описаны этапы проектирования и разработки кроссплатформенного клиент – серверного приложения.

# **ГЛАВА I. ТЕОРЕТИЧЕСКИЕ И МЕТОДОЛОГИЧЕСКИЕ ОСНОВЫ РАЗРАБОТКИ КЛИЕНТ – СЕРВЕРНЫХ ПРИЛОЖЕНИЙ**

## **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.

## **1.2. Кроссплатформенный фреймворк Qt**

Данный фреймворк включает в себя около 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.х: 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.х.х;
* 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 1.1 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.

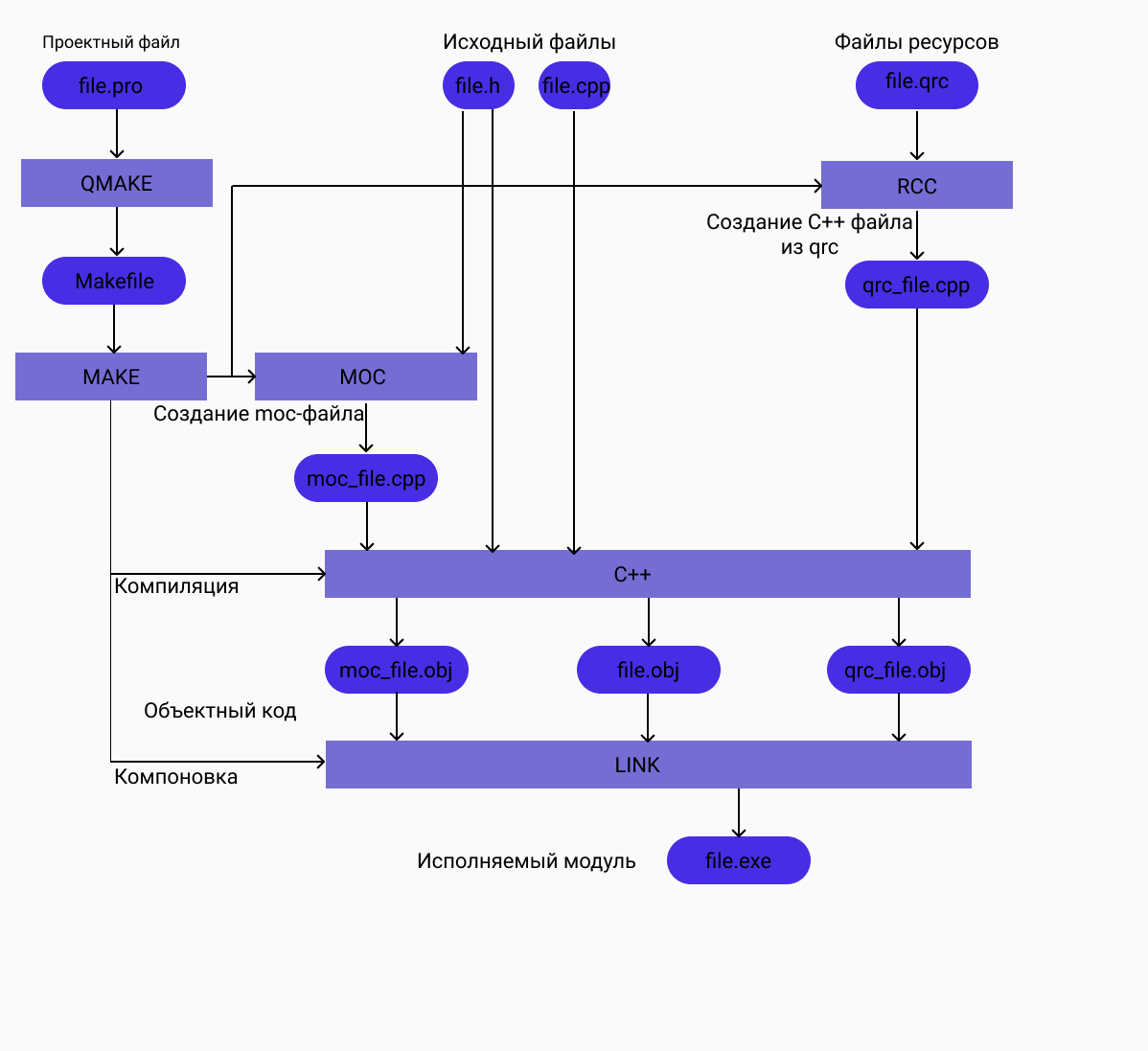


Рисунок 1. Структура Qt проекта

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 приведу простейший пример клиент – серверного взаимодействия.

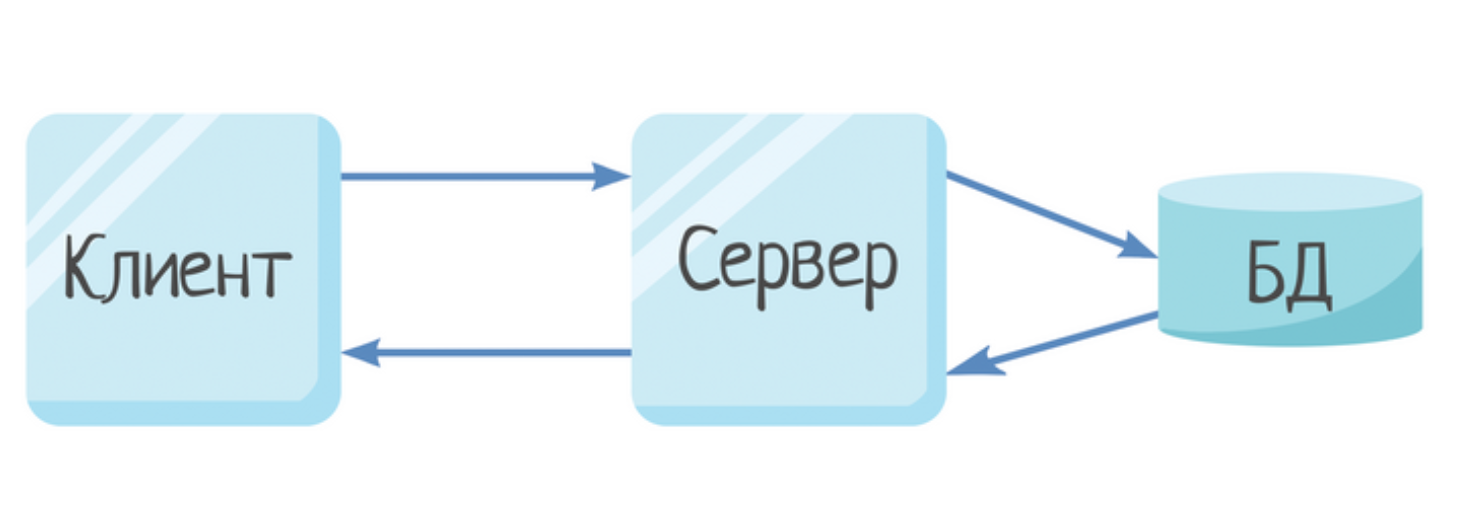


Рисунок 2. Клиент – серверная архитектура.

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.

# **ГЛАВА II. ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА КРОССПЛАТФОРМЕННОГО РАСПРЕДЕЛЁННОГО ПРИЛОЖЕНИЯ «ЧАТ»**

## **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.

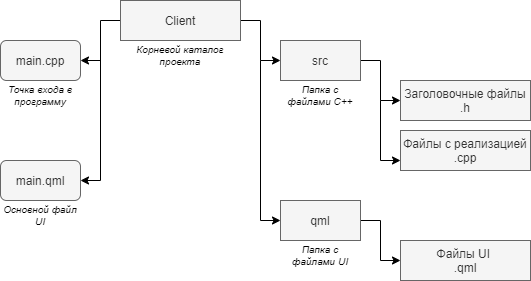


Рисунок 3. Структура проекта клиентского приложения

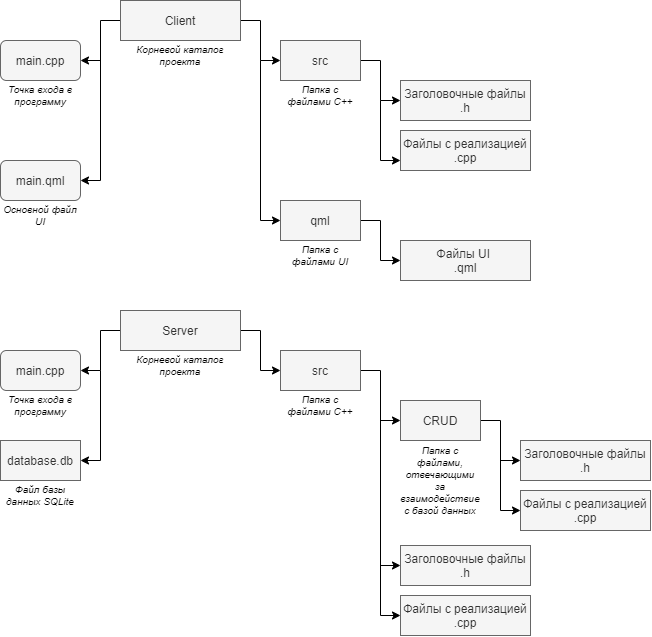


Рисунок 4. Структура проекта серверного приложения

## **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.

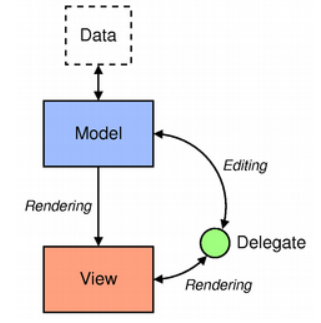


Рисунок 5. Model View Controller

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.

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**

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.

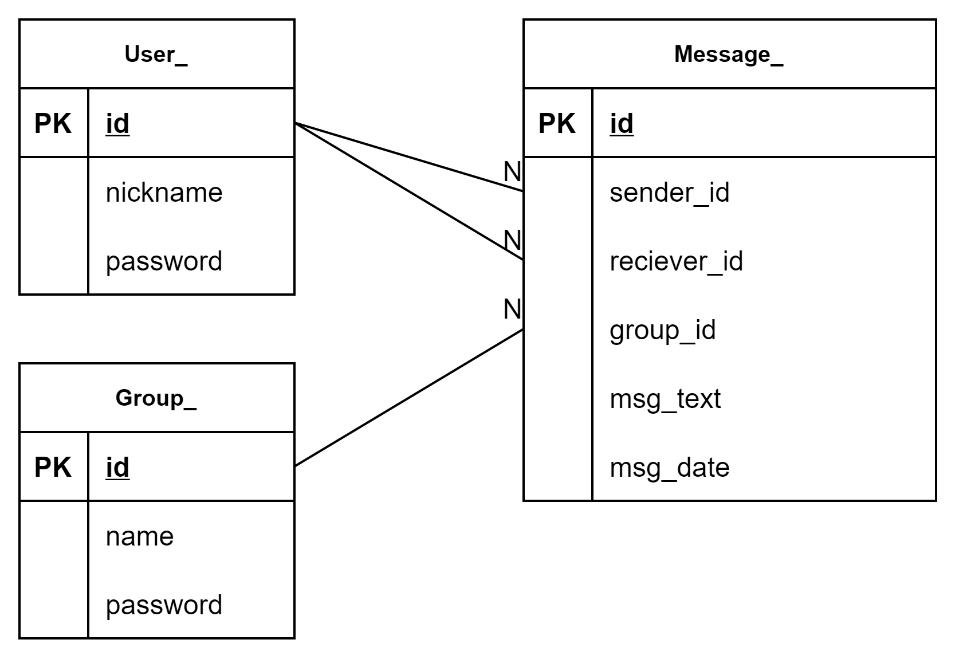


Рисунок 6. Структура базы данных

## **2.5. Этапы создания приложения**

**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. В проекте используется перечисление (*enum*) для удобства. Ниже привожу все доступные типы сообщений:

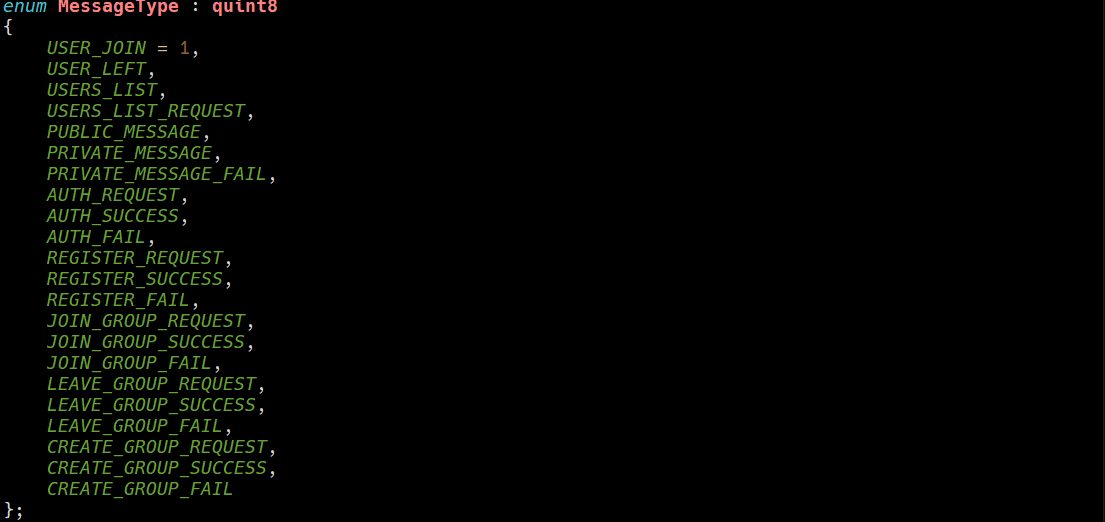


Рисунок 7. Перечисление типов сообщений

После типа сообщения будет указываться получатель, если сообщение адресовано конкретному пользователю. После будет записан текст сообщения. Ниже приведу реализацию функции, формирующей сообщения.

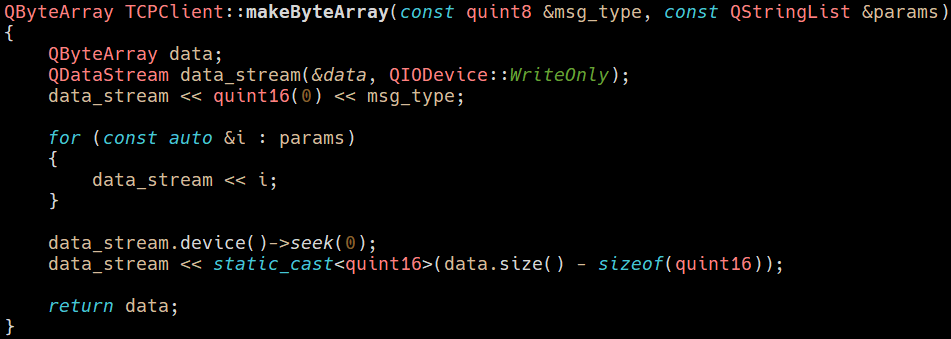


Рисунок 8. Формирование сообщений

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 требуется 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. 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. Ниже приеведена реализация этого слота.

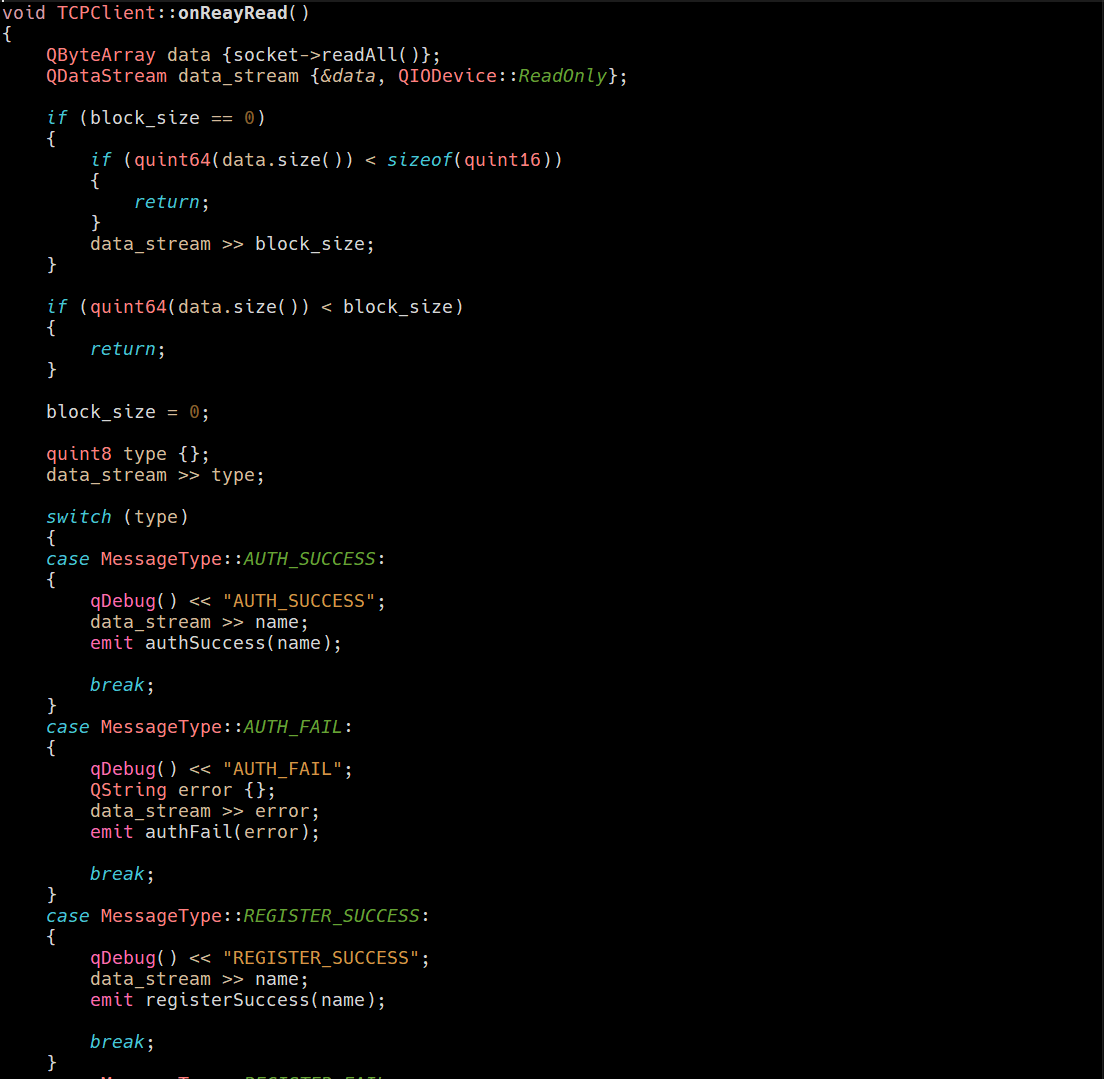


Рисунок 9. Слот, обрабатывающий полученные в сокет данные



Рисунок 10. Слот, обрабатывающий полученные в сокет данные

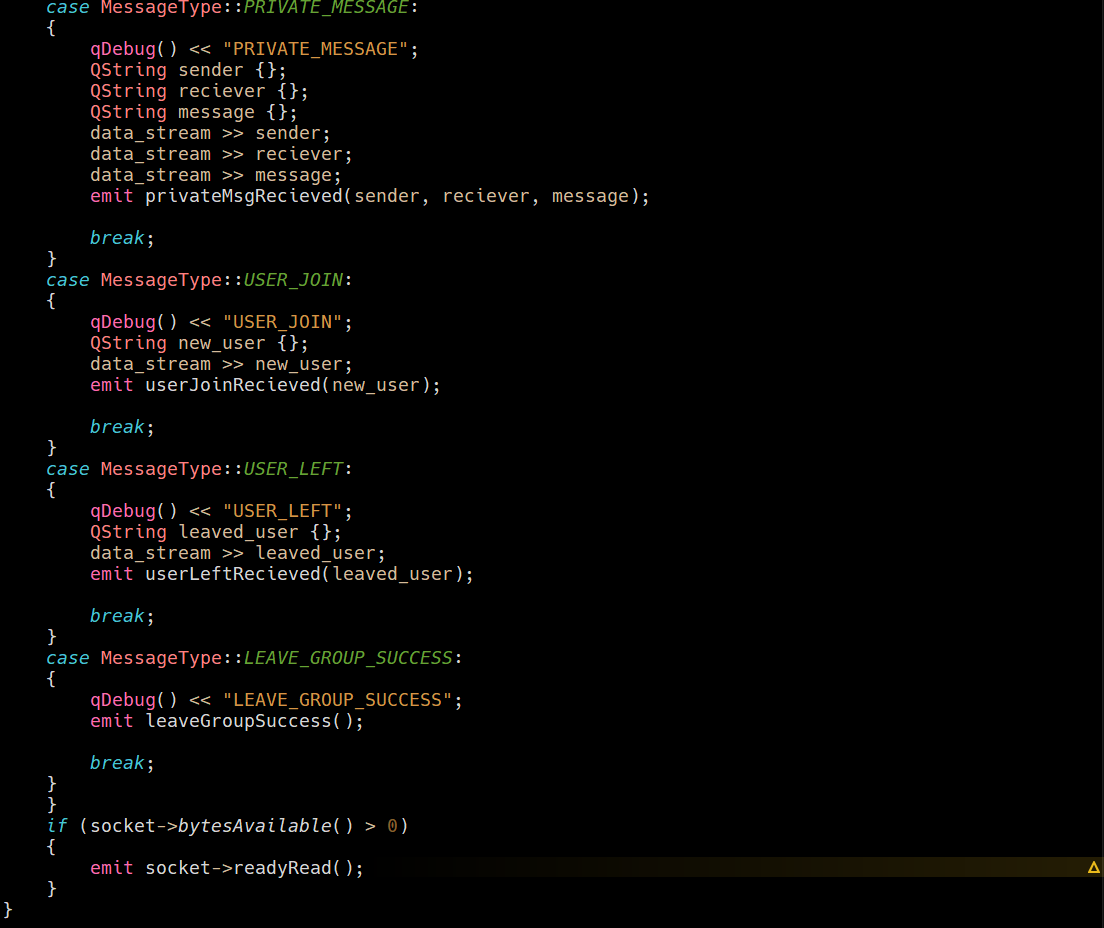


Рисунок 11. Слот, обрабатывающий полученные в сокет данные

Реализация этих методов на сервере и клиенте не отличается ничем, кроме набора кейсов в операторе *switch()*.

**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.



Рисунок 12 Заголовочный файл класса TCPServer

Класс содержит приватные свойства:

* *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;

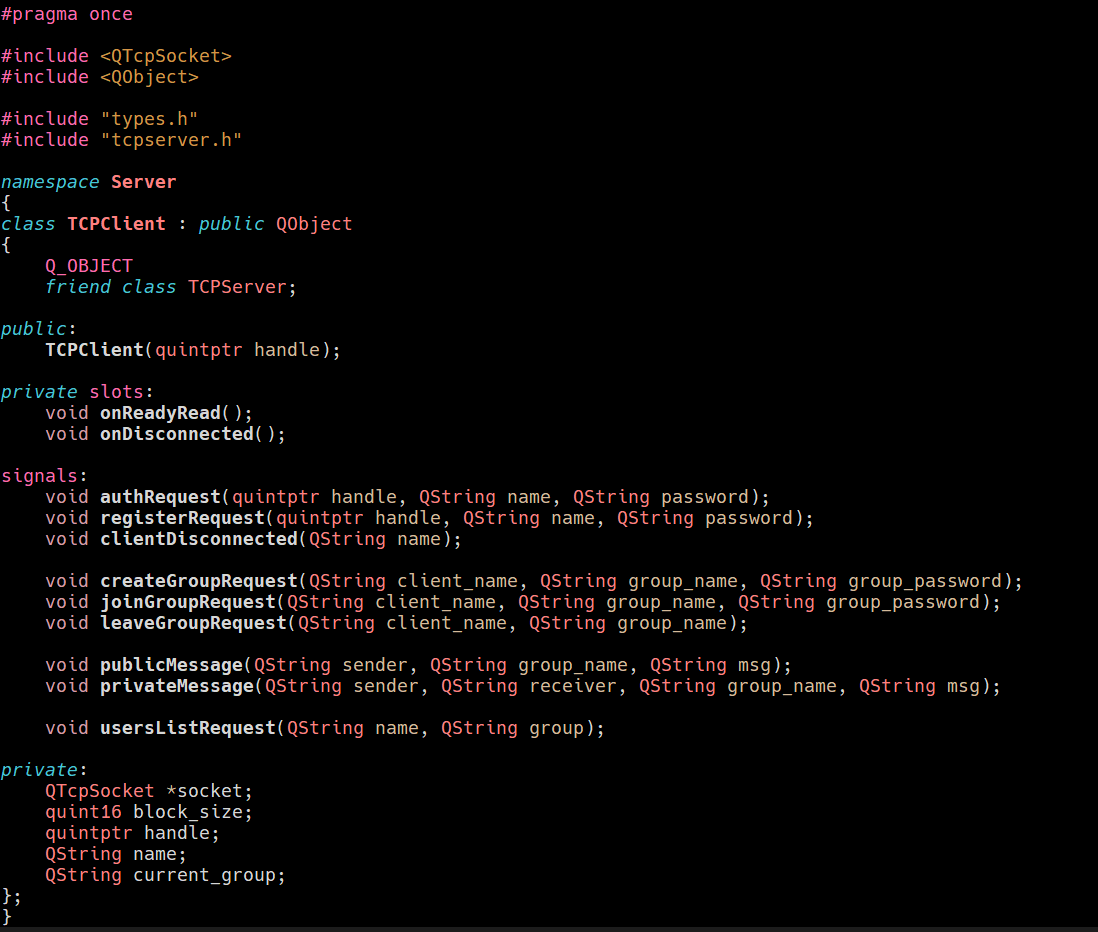


Рисунок 13. Заголовочный файл класса TCPClient

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).* Рассмотрим его реализацию.

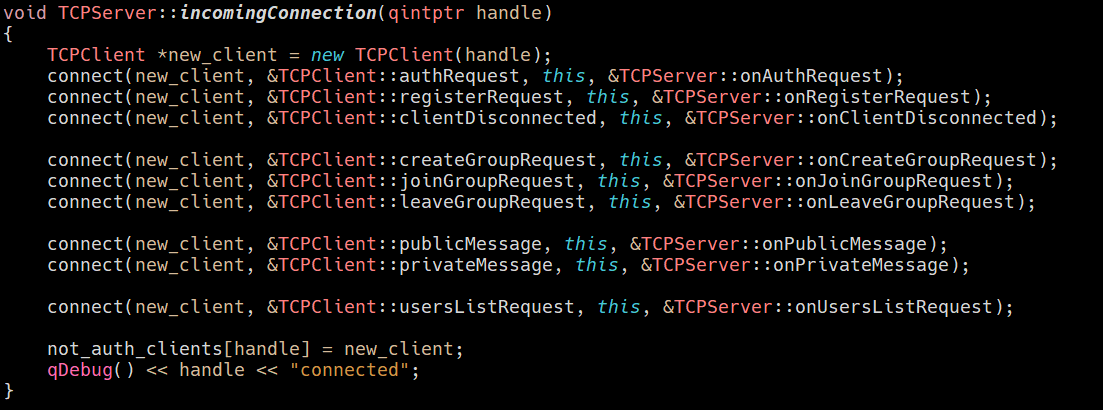


Рисунок 14. Реализация слота TCPServer::incomingConnetcion

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 захешированный пароль и вызывается метод попытки авторизации. Ниже его реализация.



Рисунок 15. Метод, реализующий попытку 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.

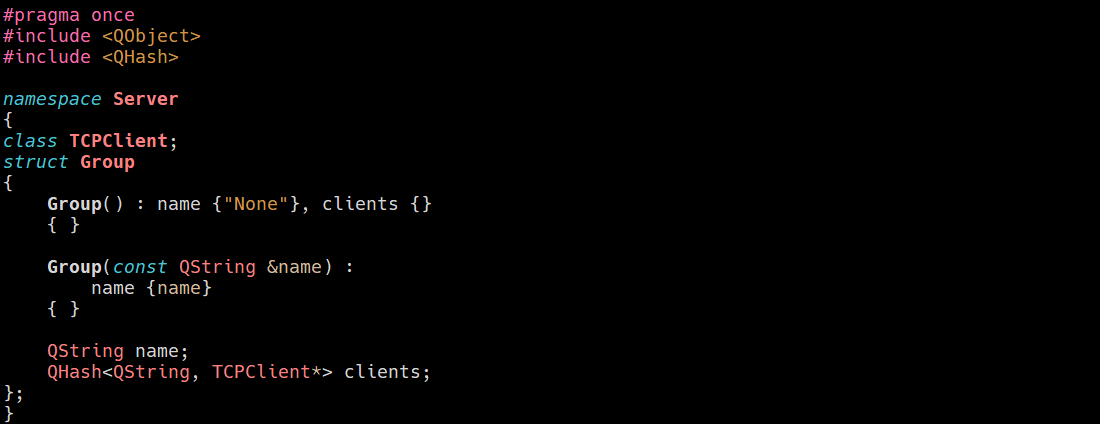


Рисунок 16. Структура Group

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*. Ниже приведена его реализация.



Рисунок 17. Реализация метода onJoinGroupRequest

Реализация похожа на авторизацию 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.

======================================================

Событие *onAuthFlagChanged* определяет поведение приложения после того, как флаг *authFlag,* инициализированный в модели данных*,* изменит свое значение.

Страница авторизации реализуется с помощью *ColumnLayout* (расположение элементов в столбце), который автоматически определяет положение элементов, учитывая указанные свойства (рисунок 2.9). Результат представлен на рисунке 2.10.

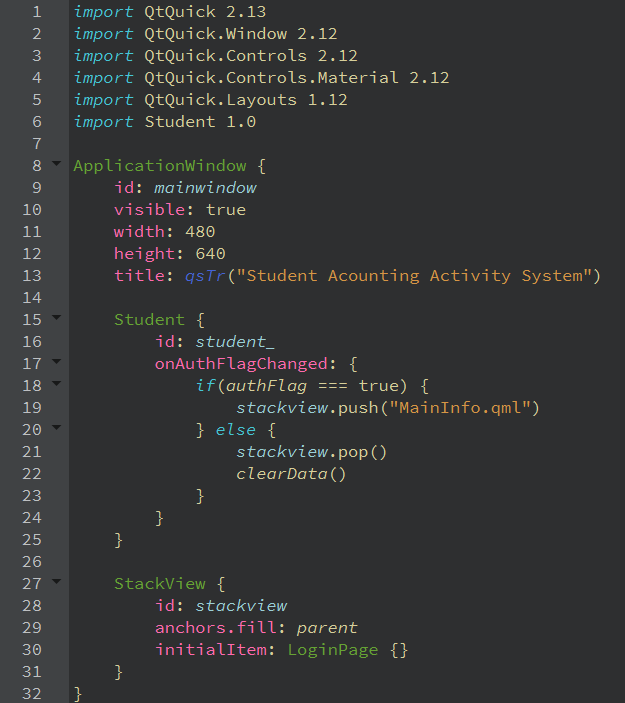
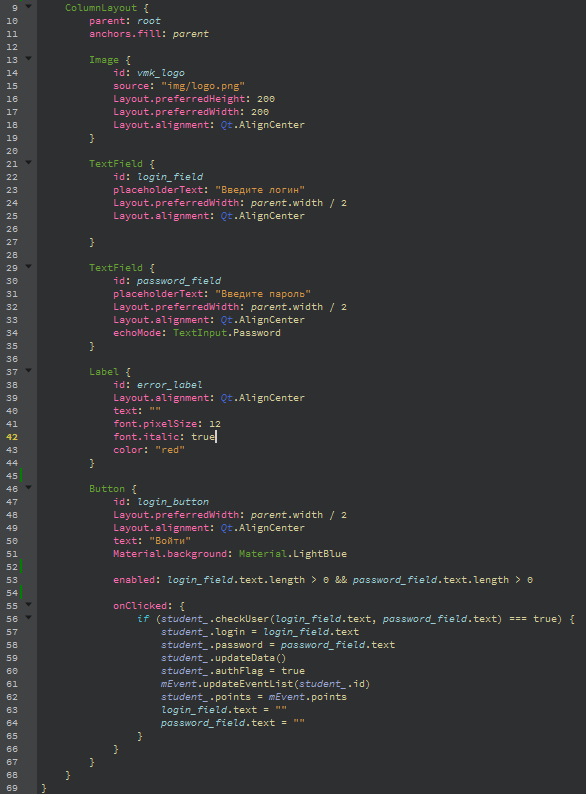
 

Рисунок 2.9. Реализация страницы авторизации

Рисунок 2.8. Определение main.qml

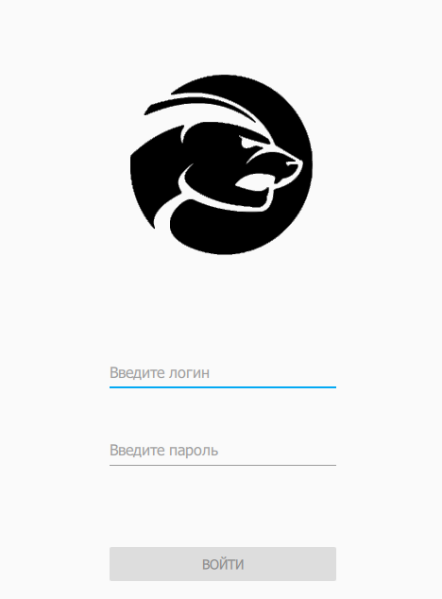


Рисунок 2.10. Страница авторизации

Кнопка «Войти» становится активной после того, как все необходимые поля будут заполнены. По нажатию на кнопку, вызывается событие на проверку пользовательских данных. Если такой пользователь найден, изменяется флаг *authFlag*, который, в свою очередь, вызывает событие, кладущее в стек новую страницу. Новая страница становится текущей и отображается пользователю.

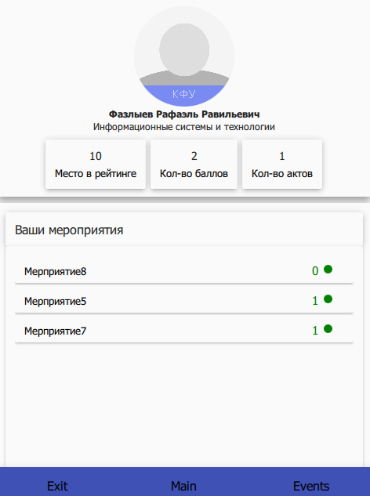


Рисунок 2.11. Главная страница

Для того, чтобы сделать изображение студента круглым, необходимо наложить маску (рисунок 2.12).



Рисунок 2.12. Наложение маски на изображение

Для реализации динамического списка мероприятий используется объект *ListView*, который получает доступ к модели данных и добавляет в делегат элементы, количество которых соответствует количеству элементов в модели данных (рисунок 2.13).

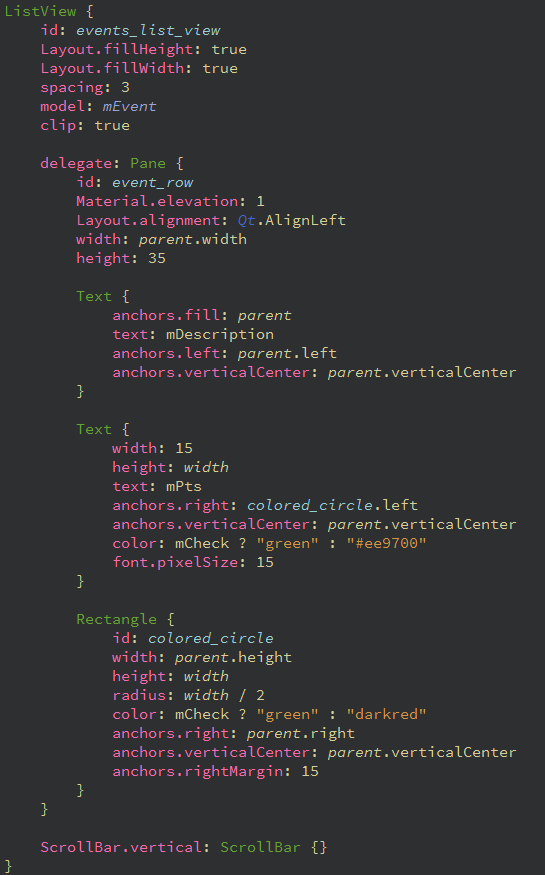


Рисунок 2.13. Реализация списка мероприятий

Список доступных мероприятий реализуется аналогично списку мероприятий. Отличие только в том, что этот список расположен в отдельной странице (рисунок 2.13).

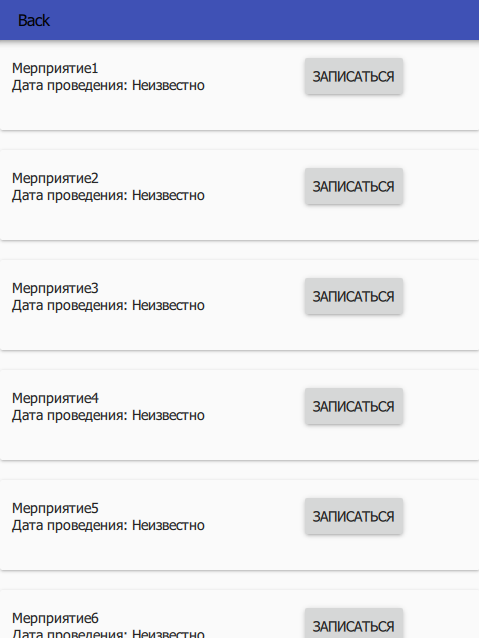


Рисунок 2.13. Список доступных мероприятий

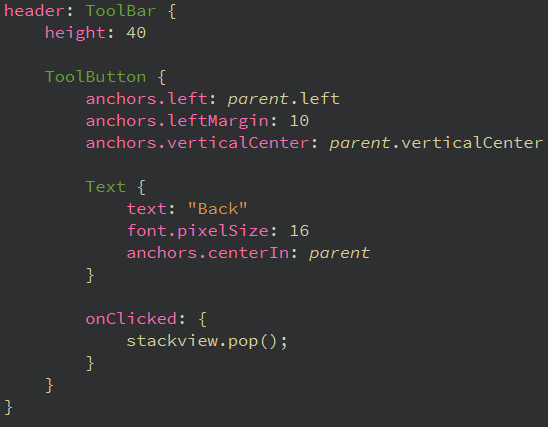


Рисунок 2.14. Реализация заголовка

Заголовок, содержащий кнопку возвращения, реализуется при помощи ключевого слова *header*, в котором определяется *ToolBar* (рис. 2.14)*.*

**Определение заголовочных файлов (классов)**

На данном этапе происходит определение основных классов, моделей данных и вспомогательных заголовочных файлов.

Класс *Student*, определенный в заголовочном файле *student.h* (рисунок 2.15), является основным классом, который отвечает за процедуру авторизации и хранение пользовательских данных. С помощью макроса *Q\_PROPERTY* устанавливаются переменные, которые будут доступны для получения, изменения переменных на стороне *QML*. К каждой из переменных указываются сигналы, которые будут вызываться в момент изменения значения переменных, а также методы для получения и изменения значения. Методы, помеченные макросом *Q\_INVOKABLE*, доступны для вызова на стороне *QML*.



Рисунок 2.15. Заголовочный файл student.h

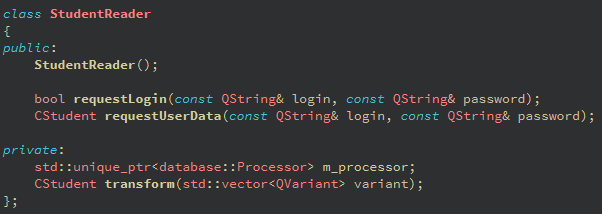


Рисунок 2.16. Заголовочный файл studentreader.h

Заголовочный файл *studentreader.h* (рисунок 2.16), в котором определен класс *StudentReader*, выполняет функцию считывания и трансформирования данных. Метод *requsetLogin* отправляет запрос на проверку введенных пользовательских данных, а метод *requestUserData* запрашивает полную информацию о пользователе. После получения, данные преобразуются к нужному формату с помощью метода *transform*.

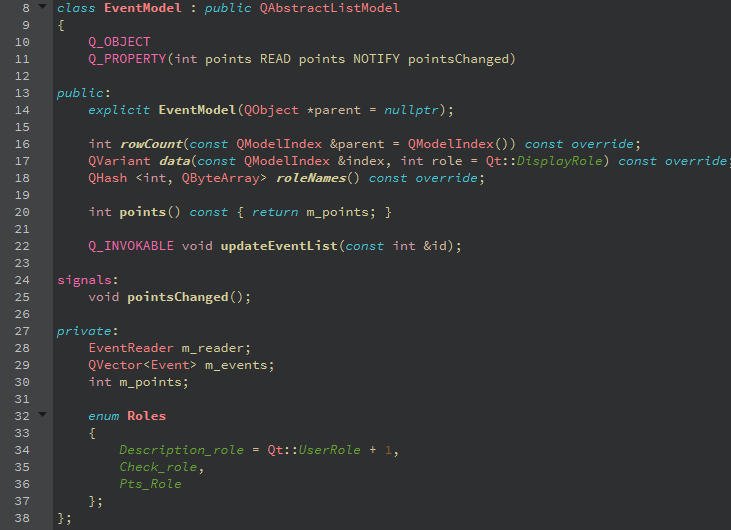


Рисунок 2.17. Заголовочный файл eventmodel.h

Класс *EventModel,* определенный в заголовочном файле *eventmodel.h* (рисунок 2.17), является моделью данных и содержит список мероприятий. В качестве контейнера (хранилища) выступает вектор, который хранит в себе список структур Event. Важным моментом является то, что данный класс наследуется от *QAbstractListModel* и переопределяет методы *rowCount*, *data*, *roleNames*. Эти методы необходимы для работы модели данных. Данный класс также имеет класс для чтения данных, он реализуется аналогично *StudentReader*.

Аналогично *EventModel, EventReader* определяется и реализуется модель *AvailableEventsModel* (список доступных мероприятий) и класс для чтения *AvailableEventsReader*.

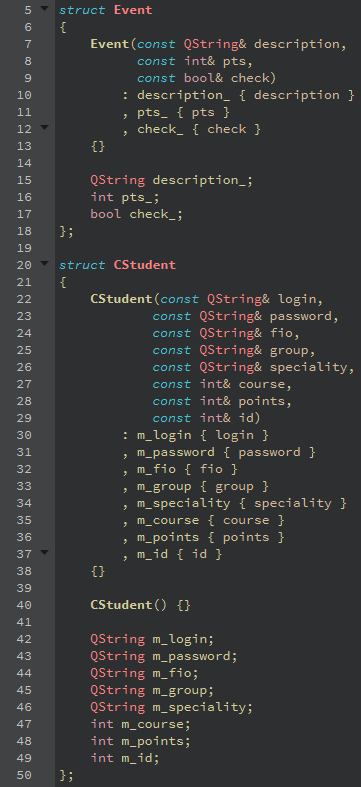


Рисунок 2.18. Заголовочный файл saastypes.h

В заголовочном файле *saastypes.h* (рисунок 2.18) определены структуры *Event* и *CStudent*.

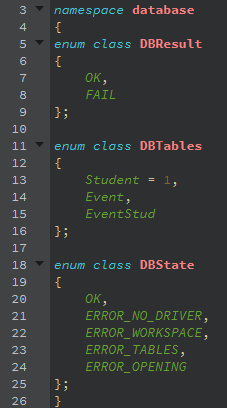


Рисунок 2.19. Заголовочный файл dbtypes.h

В заголовочном файле *dbtypes.h* (рисунок 2.19), в пространстве имен *database*, определены строго типизированные перечисления, содержащие результат выполнения запросов, наименования таблиц и текущее состояние подключения к базе данных.

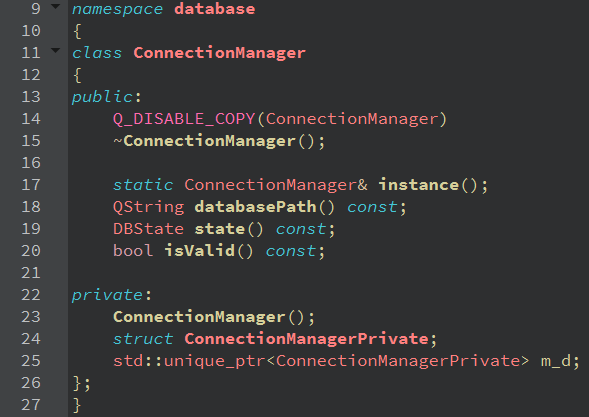


Рисунок 2.20. Заголовочный файл connectionmanager.h

В заголовочном файле connectionmanager.h (рисунок 2.20) определен класс *ConnectionManager*, отвечающий за соединение с базой данных. Причем, данный класс имеет гарантировано только один экземпляр, так как в нем реализован паттерн *Singleton* (одиночка).



Рисунок 2.21. Заголовочный файл processor.h

В заголовочном файле *processor.h* (рисунок 2.21) определен класс *Processor*, который делегирует запросы в *Selector* и *Manipulator*, в зависимости и типа действия и возвращается данные в запакованном виде.

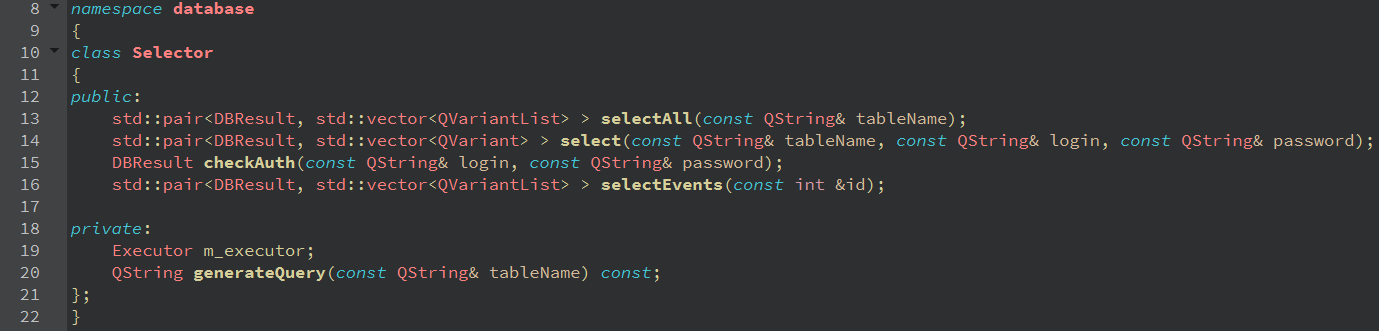


Рисунок 2.22. Заголовочный файл selector.h

В заголовочном файле *selector.h* (рисунок 2.22) определен класс *Selector*, формирующий *SELECT* запросы и передающий их на выполнение.

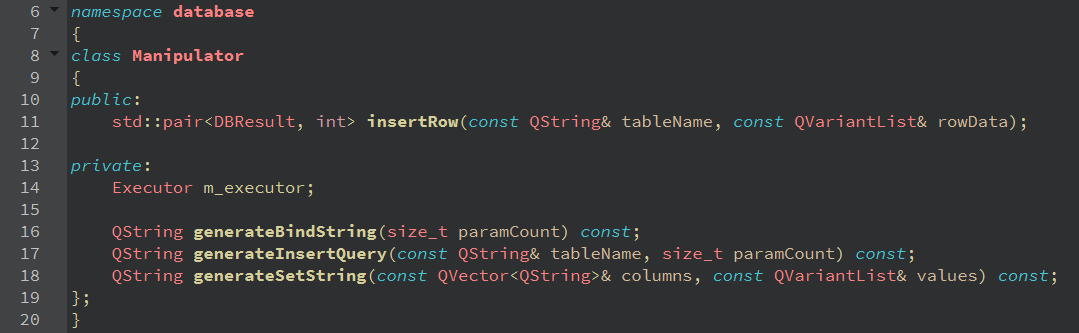


Рисунок 2.23. Заголовочный файл manipulator.h

В заголовочном файле *manipulator.h* (рисунок 2.23) определен класс *Manipulator*, формирующий *UPDATE* и *DELETE* запросы, и передающий их на выполнение.

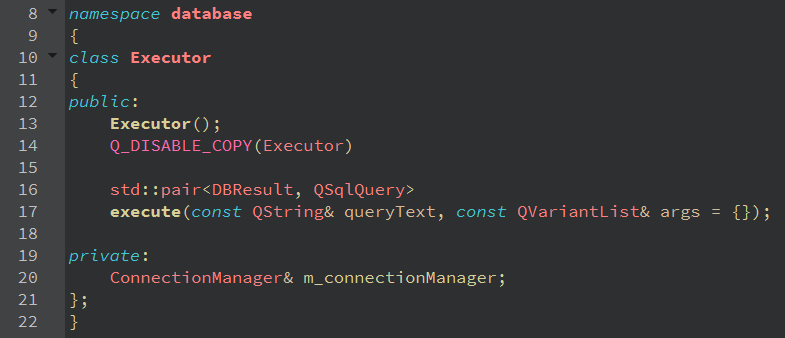


Рисунок 2.24. Заголовочный файл executor.h

В заголовочном файле *executor.h* (рисунок 2.24) определен класс Executor, который принимает сформированные запросы от *Manipulator* и *Selector* и возвращает результат выполнения запроса. Макрос *Q\_DISABLE\_COPY* отключает возможность копирования, то есть удаляет конструктор копирования.

**Реализация классов**

Данный этап подразумевает реализацию основных классов, методов и функций.

В исходном файле *eventmodel.cpp* (рисунок 2.25) реализуются все методы, которые были определены в заголовочном файле. Метод *rowCount* возвращает количество элементов в модели, *data* – конкретный элемент по индексу и роли, *roleNames* – список ролей (наименований), с помощью который можно получить или изменить значение в модели со стороны *QML*. С помощью вызова метода *readEvents*, который доступен для объекта m\_reader типа *EventReader*, обновляется контейнер, содержащий данные о мероприятиях.

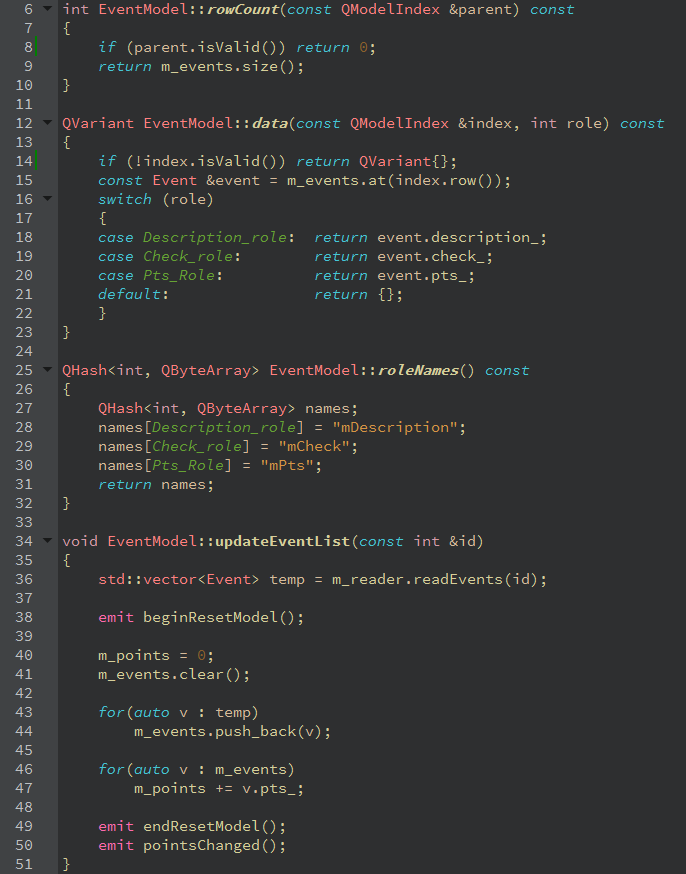


Рисунок 2.25. Исходный файл eventmodel.cpp

Для класса *EventReader* в исходном файле evenreader.cpp (рис. 2.26), реализуются методы *readEvents* и *transform*. Путем вызова метода *requestEventData*, который доступен объекту *m\_processor* типа *Processor*, контейнеру *list* присваиваются запакованные данные. Далее, метод *tansform* распаковывает эти данные и приводит их к необходимому типу.

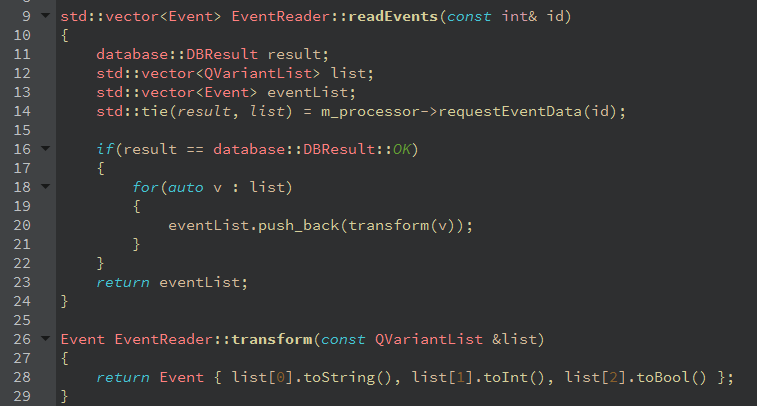


Рисунок 2.26. Исходный файл eventreader.cpp

Аналогичным образом реализуются классы *AvailableEventsModel*, *AvailableEventsReader* и *StudentReader*.

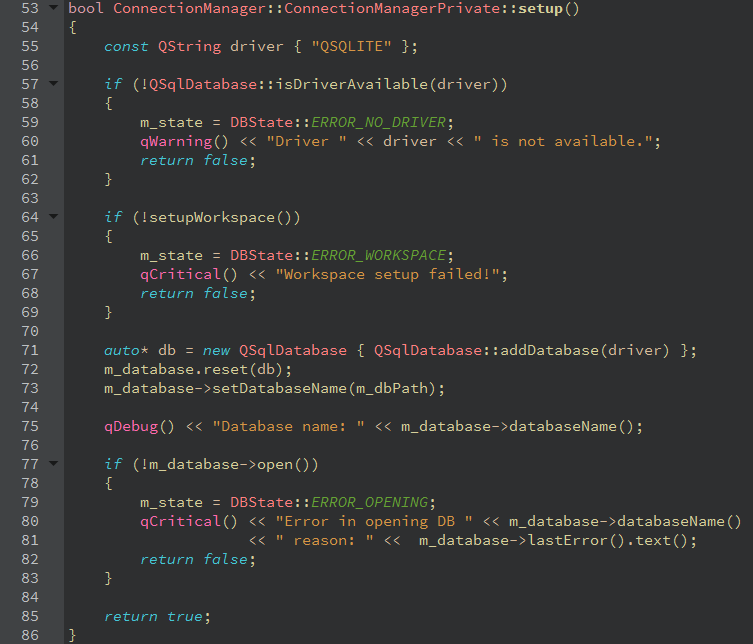


Рисунок 2.27. Исходный файл connectionmanager.cpp

Метод *setup*, класса *ConnectionManager* (рисунок 2.27), выполняет проверку на наличие драйверов, корректность рабочего пространства и осуществляет соединение с базой данных. Результатом выполнения является логическая переменная.

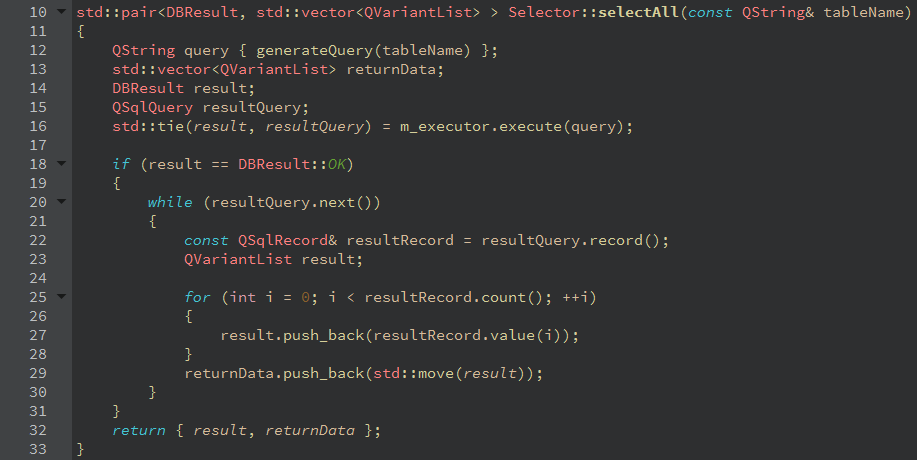


Рисунок 2.28. Исходный файл selector.cpp

Метод *selectAll*, класса Selector (рисунок 2.28), формирует запрос с помощью метода *generateQuery* на получение всей таблицы, и возвращает результат в *Processor*. Остальные методы класса имеют практически идентичную реализацию.



Рисунок 2.29. Исходный файл manipulator.cpp

Метод *insertRow* (рисунок 2.29) выполняет вставку строки в базу данных. Запрос формируется при помощи методов *generateInsertQuery* и *generateBindString*.



Рисунок 2.30. Исходный файл executor.cpp

Метод *execute* (рисунок 2.30) является основным в классе *Executor* и выполняет отправку запроса в базу данных, с последующей проверкой на ошибки и предупреждения.



Рисунок 2.31. Исходный файл main.cpp

В функции *main*, исходного файла *main.cpp* (рисунок 2.31) инициализируются объекты основных элементов библиотеки *Qt*, а также создаются экземпляры моделей данных, указатели которых устанавливаются как контекстное свойство для ядра *QML*.

# **ЗАКЛЮЧЕНИЕ**

Кроссплатформенное приложение, разработанное посредством использования библиотеки Qt, доступно для сборки под Windows, MacOS, iOS, Android и Linux без потери функциональности. Причем, учитываются все современные требования безопасной и эффективной передачи данных. Приложение позволяет быстро получить информацию студентам об их положении в общем рейтинге, количестве заработанных баллов и нарушений. Доступность на самых популярных платформах и операционных системах дает возможность каждому студенту пользоваться данным приложением, вне зависимости от его местоположения и типа устройства.

# **СПИСОК ЛИТЕРАТУРЫ**

1. Autodesk Maya [Электронный ресурс]. – Режим доступа: <https://www.autodesk.ru/products/maya/overview>, свободный (Дата обращения 01.05.2020)
2. Oracle VM VirtualBox [Электронный ресурс]. – Режим доступа: https:/virtualbox.org/, свободный (Дата обращения 01.05.2020)
3. Qt Documentation [Электронный ресурс]. – Режим доступа: <https://doc.qt.io/>, свободный (Дата обртащения 01.05.2020)
4. Skype [Электронный ресурс]. – Режим доступа: https:/skype.com/, свободный (Дата обращения 01.05.2020)
5. Telegram [Электронный ресурс]. – Режим доступа: <https://telegram.org/>, ограниченный (Дата обращения 01.05.2020)
6. Книга: Qt 5.10. Профессиональное программирование на С++. Автор: Шлее Макс.