Учреждение образования

«Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

 К ЗАЩИТЕ ДОПУСТИТЬ

\_\_\_\_\_\_\_\_\_\_\_ М. Г. Базанов

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

«Кроссплатформенный мессенджер с возможностью голосового общения»

по дисциплине

«Системное программное обеспечение вычислительных машин»

БГУИР КР 1-40 02 01 525 ПЗ

Выполнил:                                                                   Руководитель:

студент гр. 650505                                                     Базанов М. Г.

Пташник Е. В.

МИНСК 2018

Учреждение образования

«Белорусский государственный университет информатики

и радиоэлектроники»

Факультет: Компьтерных систем и сетей

Кафедра: Электронных вычислительных машин

Специальность: 40 02 01 «Вычислительные машины, системы и сети»

Предмет: Системное программное обеспечение электронных вычислительных машин

УТВЕРЖДАЮ Заведующий кафедрой ЭВМ \_\_\_\_\_\_\_\_\_ Д. И. Самаль

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_ 20\_\_ г.

ЗАДАНИЕ

по курсовому проектированию студента

Пташника Егора Владимировича

1. Тема проекта: «Кроссплатформенный мессенджер с возможностью голосового общения».

2. Дата выдачи задания: 20.02.2018 г.

3. Предоставление студентом готового проекта: 26.05.2018 г.

4. Решаемые задачи и функционал разрабатываемого ПО:

4.1. Многопользовательский текстовый чат.

4.1.1. Регистрация пользователей.

4.1.2. Возможность добавления и удаления собеседников.

4.1.3. Возможность создания групповых чатов.

4.1.3. Привилегии администратора.

4.2. Голосовой чат.

4.3. Возможность изменения имени.

5. ОС и средства разработки:

5.1. *Qt Creator.*

5.2. *Windows* 10.

5.3. *Ubuntu* 16.04.3.

6. Другое:

7. Содержание пояснительной записки:

Введение. 1. Обзор литературы. 2. Структурное проектирование. 3. Функциональное проектирование. 4. Разработка программных модулей. 5. Руководство пользователя. 6. Тестирование. Заключение. Литература. Приложения.

8. Перечень графического материала:

8.1. Схема структурная. Формат А4.

8.2. Диаграмма классов. Формат А3.

8.3. Оповещение о добавлении собеседника в чат. Схема программы. Формат А4.

8.4. Поиск пользователя по имени. Схема программы. Формат А4.

8.5. Листинг программы. Формат А4.

9. Контрольные точки готовности частей проекта:

9.1. 01.03 – введение, глава 1 (постановка задачи), лист задания.

9.2. 01.04 – предварительная структурная схема и диаграмма классов.

9.3. 01.05 – предварительная демонстрация ПО.

9.4. 10.05 – структурная схема, диаграмма классов, главы 1-6.

РУКОВОДИТЕЛЬ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Базанов М. Г.

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Пташник Е. В.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc515953974)

[1 ОБЗОР ЛИТЕРАТУРЫ 6](#_Toc515953975)

[1.1 Выбор технологий для создания проекта 6](#_Toc515953976)

[1.2 Основные термины 6](#_Toc515953977)

[1.3 Основные функции создаваемого ПО 7](#_Toc515953978)

[2 СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ 8](#_Toc515953979)

[2.1 Общее структурное описание состава ПО 8](#_Toc515953980)

[2.2 Описание деления проекта 8](#_Toc515953981)

[2.3 Описание сторонних программных компонентов 9](#_Toc515953982)

[3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 10](#_Toc515953983)

[3.1 Общее описание функционирования ПО 10](#_Toc515953984)

[3.2 Описание форматов внешних файлов данных 11](#_Toc515953985)

[4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 13](#_Toc515953986)

[4.1 Описание теоретических основ для разработки алгоритмов 13](#_Toc515953987)

[4.2 Описание алгоритмов, разработанных самостоятельно 14](#_Toc515953988)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 16](#_Toc515953989)

[5.1 Системные требования 16](#_Toc515953990)

[5.2 Запуск программы 16](#_Toc515953991)

[5.3 Использование программы 16](#_Toc515953992)

[6 ТЕСТИРОВАНИЕ 20](#_Toc515953993)

[6.1 Проверка работоспособности 20](#_Toc515953994)

[6.2 Качественные и количественные характеристики 20](#_Toc515953995)

[ЗАКЛЮЧЕНИЕ 21](#_Toc515953996)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22](#_Toc515953997)

[ПРИЛОЖЕНИЕ А 23](#_Toc515953998)

[ПРИЛОЖЕНИЕ Б 24](#_Toc515953999)

[ПРИЛОЖЕНИЕ В 25](#_Toc515954000)

[ПРИЛОЖЕНИЕ Г 26](#_Toc515954001)

[ПРИЛОЖЕНИЕ Д 27](#_Toc515954002)

**ВВЕДЕНИЕ**

Темой данного курсового проекта по предмету «Системное программное обеспечение вычислительных машин» был выбран «Кроссплатформенный мессенджер с возможностью голосового общения». Целью программного обеспечения, которое должно быть разработано, является улучшение и упрощение общения между людьми. Сегодня, в информационный век, человек находится в постоянной коммуникации с остальными и ему доставляет большой дискомфорт отсутствие возможности здесь и сейчас связаться с нужными людьми. Данная разработка должна справиться с этой задачей.

Целевой аудиторией продукта будут офисные работники и молодёжь, то есть люди, в основном быстро разбирающиеся в компьютерных технологиях, но в то же время требовательные к качеству продукта. По этой причине программное обеспечение обязано иметь понятный и простой интерфейс, а также хорошую стабильность, чтобы не отпугнуть пользователей и дать им возможность разобраться в работе без дополнительных источников. Эта разработка выделяет свободное время в графике рабочего, освобождая его от части неосновных собраний и походов к коллегам, с которыми нужно обсудить различные вопросы.

На сегодняшний день мессенджеры являются очень популярными приложениями. Кроме простого общения, они предоставляют возможность передачи аудио- и видеосообщений, файлов любого формата, добавляют интерактивные элементы в виде смайлов и стикеров, сохраняют тайну переписки посредством шифрования

Работа над курсовым проектом по данной теме позволяет развить свои навыки в программировании, в особенности улучшить знание языка С++, научиться работать с базами данных, с сетями, а также подготавливает к дипломному проектированию. Данная работа является вторым опытом в разработке полноценного программного обеспечения.

1. **ОБЗОР ЛИТЕРАТУРЫ**
   1. **Выбор технологий для создания проекта**

Для разработки программного обеспечения (ПО) был выбран язык С++. Основными преимуществами данного языка является поддержка такого подхода, как объектно-ориентированное программирование, а также совокупность свойств и низкоуровневых, и высокоуровневых языков. В сочетании с большим количеством реализаций языка это даёт большую гибкость в написании проекта.

Средой разработки был использован Qt Creator. Среди всех аналогов (таких, как Microsoft Visual Studio и Borland) можно выделить его открытый доступ, большое количество библиотек, поддерживаемых хорошим описанием и справками, а также широкий спектр возможностей для создания графического интерфейса. Например, в Microsoft Visual Studio для создания пользовательского интерфейса необходимо подключать библиотеку MFC (Microsoft Foundation Classes), функционал которой, по сравнению с Qt, остаётся бедным и менее удобным (занимает больше строк). Ещё преимуществами выбранной среды являются его кроссплатформенность и наличие классов, которые позволяют работать с базами данных на основе языка SQL и сетевыми протоколами (TCP и UDP).

* 1. **Основные термины**

Под кроссплатформенностью подразумевается способность ПО работать на нескольких платформах или операционных системах. Для её реализации нужно использование высокоуровневых языков программирования, их сред разработки, которые поддерживают компиляцию на различных платформах.

Среда разработки – это комплекс программных средств (в них входят текстовый редактор, компилятор, интерпретатор, средства автоматизации сборки, отладчик), который используется для создания программного обеспечения.

База данных (БД) – это набор данных, связанных между собой и структурированных в нужном порядке, с которыми может взаимодействовать (считывать, добавлять, удалять, редактировать и другие) пользователь через устройства с установленным на них ПО. Одним из самых удобных языком программирования для работы с базами данных является SQL (Structured Query Language). Он хорошо стандартизирован и лёгок в изучении.

Так как конечным продуктом курсового проекта будет мессенджер, то необходимо рассмотреть его работу и как эта работа будет выполняться в виде функций.

Мессенджер – это программа, которая позволяет мгновенно обмениваться информацией через интернет. Соединение между пользователями строится либо в формате клиент-клиент (удобно при аудио- и видеообщении), либо в формате клиент-сервер-клиент (необходимо для сохранения сообщений и файлов, информации об участниках). Также в мессенджерах можно создавать беседы, в которых может общаться больше двух людей. Всё это должно быть учтено и правильно разбито на функции для корректной работы программы.

В дальнейшем в пояснительной записке также будут использоваться следующие термины:

* парсинг – последовательный синтаксический анализ информации;
* ID (*identifier* – опознаватель) – уникальный признак (чаще всего номер), позволяющий отличить элемент от других ему подобных в одной структуре;
* виджет – базовый элемент графического пользовательского интерфейса, имеющий стандартный вид и выполняющий стандартные действия;
* фреймворк – программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта;
* протокол – набор правил интерфейса, которые определяют обмен данными между элементами и обработку ошибок.
  1. **Основные функции создаваемого ПО**

Для хранения информации о пользователях будет создана база данных, в которой будут храниться их ID, логины, пароли, имена и статусы. Также в базе данных будет содержаться информация о диалогах и беседах, а взаимосвязь между ними и пользователями будет представлена в виде перекрёстной таблицы. Сервер посредством реализованных функций должен уметь работать с БД.

Со стороны клиента и сервера основными будут функции формирования, отправления и обработки запросов.

При включении программы, клиент должен авторизоваться. Для этого ему необходимо отправить свой логин и пароль серверу, который по базе данных верифицирует пользователя и отправит обратно сообщение либо о разрешении на вход, либо о запрете.

Далее со стороны клиента нужно реализовать функции отправления и добавления приходящих сообщений, оповещения об изменениях, добавления и исключения собеседников, создания и удаления бесед, старта и окончания звонков. Со стороны сервера – занесение в БД информации о новых и учёт изменений о старых пользователях, приём и рассылку сообщений, создание и удаление файлов, связанных с беседами, предоставление IP-адреса, а также оповещение собеседника об окончании звонка.

1. **СТРУКТУРНОЕ ПРОЕКТИРОВАНИЕ**
   1. **Общее структурное описание состава ПО**

Разрабатываемое ПО состоит из 5 основных модулей:

* пользовательский графический интерфейс;
* клиент;
* сервер;
* база данных;
* данные переписок.

Полное строение проекта в виде блоков изображено в структурной схеме, которая представлена на рисунке 2.1 и на чертеже ГУИР.400201.525 С1 (приложение А).



Рис. 2.1 – Структурная схема разрабатываемого проекта

* 1. **Описание деления проекта**

*Пользовательский графический интерфейс* отвечает за контакт пользователя с приложением. Он связан только с клиентской частью и отображает информацию, обработанную клиентом, а также команды (добавление, удаление, редактирование), которые можно произвести с этой информацией.

*Клиент* отвечает за отправление запросов и обработку всех данных, которые программа получает от сервера, а именно: разрешение на вход, сообщения, изменение своего статуса или статуса других пользователей, добавление в беседы, оповещение о входящем звонке. Также клиент может обмениваться информацией напрямую между другими клиентами в случае с голосовым общением.

*Сервер* представляет собой ядро, обрабатываемое поступающие от всех клиентов запросы. Для быстрого ответа на запрос он имеет прямой доступ к базе данных и файлам, хранящим данные переписок. В случае, если сервер прекращает свою работу, останавливается функционирование всего мессенджера. Поэтому этот модуль является связывающим звеном всего проекта.

*База данных* представляет собой архив всей информации, отвечающей за пользователей и их бесед. Она связана только с сервером и ему единственному разрешено редактировать внутренние данные.

*Данные переписок* отвечают за хранение текстовых сообщений. Они хранятся в виде каталога файлов, каждый из которых связан с отдельной беседой, имеющих свой идентификационный номер. Так же, как и база данных, предоставляет доступ только серверу.

* 1. **Описание сторонних программных компонентов**

Основной компонентой, использованной для упрощения разработки проекта, является *Qt* – фреймворк со встроенным редактором графического интерфейса и детально проработанной документацией. К удобствам можно отнести возможность изменения дизайна базовых виджетов через механизм *QSS*, который является аналогом языка описания внешнего вида документов *CSS.* Ключевым удобством является его полная кроссплатформенность, ускоряющая во много раз разработку проекта.

Для проверки правильного создания и изменения баз данных, созданных на базе *SQL*, использовалась программа *SQLiteStudio*. Преимуществами данного приложения является удобный интерфейс. Стоит отметить, что эта программа также была написана с помощью фреймворка Qt.

1. **ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

**3.1 Общее описание функционирования ПО**

В данном разделено будет рассмотрено функционирование программы с начала её запуска. Все упомянутые ниже классы и методы изображены на диаграмме классов (см. чертёж ГУИР.400201.525 РР– приложение Б).

В корневой папке сервера хранится база данных. Её открытие происходит сразу после запуска в конструкторе поля *Server::database* через метод *QSqlDatabase::open().* Далее сервер настраивается на прием запросов через определенный порт, который можно выбрать в методе *QTcpServer::listen(QHostAddress address, quint16 port)*. После этого программа готова к работе и ожидает клиентов.

При включении клиентской части проекта сначала происходит подключение к серверу (метод *Client::connectToServer(QString IP, int port)*). При успешной попытке, пользователь может отправлять различные запросы, которые состоят из трёх частей– размер блока (один байт), ID операции (всего имеется 17 видов, также один байт) и сам блок передаваемой информации (не имеет фиксированного размера). Отправив запрос с помощью метода *Client::sendPacket(int id, QString block)*, сервер сразу принимает его реализованным методом-сигналом (более подробно рассмотрены в подразделе 4.1) *Server::dataReceived()* и в зависимости от операции вызывает далее нужную функцию. После обработки он формирует ответ клиенту, который схож клиентскому запросу. Отличие заключается в том, что к нему добавляется один байт, свидетельствующий об успехе или неуспехе операции (метод *Server::sendPacket(int id, int status, QString block)*). В итоге пользователь получает запрашиваемую информацию через такой же метод *Client::dataReceived()*.

Все операции хранятся в пространстве имён *ServerCommands*, в отдельном перечисляемом типе *ControlCommands*:

* регистрация (ID\_SIGIN, 0);
* авторизация (ID\_AUTHO, 1);
* изменение пароля (ID\_CHNPW, 2);
* получение информации о переписке (ID\_GETCV, 3);
* получение всего текста переписки (ID\_GTFLL, 4);
* отправление сообщения (ID\_SENDM, 5);
* вход в сеть (ID\_USRON, 6);
* выход из сети (ID\_LGOUT, 7);
* отказ в подключении (ID\_CNNCT, 8);
* изменение имени (ID\_CHGNM, 9);
* создание беседы (ID\_CRTCV, 10);
* добавить собеседника (ID\_ADDMR, 11);
* исключить собеседника (ID\_LFTCV, 12);
* получение списка всех пользователей (ID\_ALLUR, 13);
* получение IP-адреса собеседника (ID\_GETIP, 14);
* исходящий звонок (ID\_INCCL, 15);
* завершение звонка (ID\_ENDCL, 16).

Например, при попытке авторизации клиент отправляет серверу логин и пароль, тот в свою очередь сверяет полученную информацию с базами данных и отправляет обратно разрешение или отказ в доступе. В зависимости от этого у пользователя либо появляется сообщение о неправильно заполненных полях, либо открывается главное окно мессенджера. На таком принципе основан обмен информацией между двумя программами.

При голосовом общении один из пользователей, нажав на клавишу «Позвонить», вызывает метод *Client::on\_callButon\_clicked()*, в котором посылается запрос серверу об исходящем звонке. Тот в ответ отправляет IP-адрес потенциального собеседника, а тому собеседнику сигнал о входящем звонке с IP-адресом позвонившего. Если второй пользователь соглашается на запрос, оба клиента подключается напрямую друг к другу через методы *AudioReceiver::startCall()* и *AudioTransmitter::startCall()* благодаря сокетам, работающим по UDP протоколу. Порты заранее выбраны программой, но в случае необходимости их можно изменить. Окончание разговора проходит по обратной схеме с участием сервера.

В случае, если сервер неожиданно прекращает свою работу, у клиентов реализован метод-сигнал *Client::loggedOut()*, который извещает пользователя, вернув окно авторизации с выводом сообщения о потере соединения с сервером.

**3.2 Описание форматов внешних файлов данных**

Во время своей работы приложение взаимодействует с файлами четырёх расширений: *\*.txt*, *\*.sqlite*, *\*.jpg*, *\*.css*. О каждом подробнее.

В данном проекта текстовый формат *\*.txt* используется для хранения базы данных читателей. Он является стандартным текстовым документом *Windows*. Содержит текст без форматирования, что позволяет легко считывать информацию. Может быть открыт любой программой для работы с текстом (это можно отнести как к преимуществу, так и к недостатку). Существенных ограничений при работе с данным форматом нет, кроме выбора кодировки: для правильной работы из-за требований фреймворка *Qt* была выбрана *UTF-8*.

База данных книг хранится в файле формата *\*.sqlite*. Название расширения соответствует типу БД, которая поддерживает полный набор команд *SQL* и не накладывает никаких существенных ограничений на использование (в случае необходимости, их можно увеличивать). Возможные типы полей: *INTEGER, REAL, TEXT, BLOB* (аналог *bool*). Примеры ограничений: максимальное количество колонок (2000), максимальная длина *SQL*-выражения (1000000), максимальная глубина дерева выражений (1000), максимальное количество аргументов функции (100), максимальное количество объединённых выражений с *SELECT* (500) и другие.

Для хранения фотографий используется формат *\*.jpg* – один из самых популярных графических форматов. Алгоритм *JPEG* выполняет сжатие изображения. Файл JPEG содержит последовательность *маркеров*, каждый из которых начинается с байта, свидетельствующего о начале маркера, и байта-идентификатора. Такая структура файла позволяет быстро отыскать маркер с необходимыми данными (например, с длиной строки, числом строк и числом цветовых компонентов сжатого изображения). К недостаткам сжатия по стандарту JPEG следует отнести появление на восстановленных изображениях при высоких степенях сжатия характерных артефактов: изображение рассыпается на блоки размером 8x8 пикселей, в областях с высокой пространственной частотой (например, на контрастных контурах и границах изображения) возникают артефакты в виде шумовых ореолов. Однако, несмотря на недостатки, JPEG получил очень широкое распространение из-за достаточно высокой степени сжатия, поддержке сжатия полноцветных изображений и относительно невысокой вычислительной сложности [8].

Как писалось ранее в подразделе 2.3, в Qt реализована возможность изменения дизайна виджетов через механизм *QSS.* Так как он является аналогом *CSS*, расширение файлов осталось таким же – *\*.css*. Внутри него хранится описание внешнего вида на языке разметки, то есть можно задать цвет, шрифт, положение, формы и другие разного рода параметры. К преимуществам данного языка можно отнести простой синтаксис, ключевые слова которого являются просто словами из английского языка. Для выполнения этого действия необходимо подключить его к проекту. Пример такого описания на *CSS*:

QPushButton /\*название описываемого объекта\*/

{

background-color: #d81b1b; /\*задаётся цвет заднего фона\*/

border-width: 2px; /\*задаётся толщина рамки\*/

border-color: #3a3633; /\*задаётся цвет рамки\*/

border-style: solid; /\*задаётся стиль рамки\*/

padding: 5px; /\*задаётся выравнивание относительно сторон\*/

font: 10pt "Tahoma"; /\*задаётся шрифт текста и его размер\*/

color: white; /\*задаётся цвет текста\*/

}

QPushButton:pressed /\*задаётся изменение состояния при нажатии\*/

{

background-color: rgb(255,255,255); /\*при нажатии изменится цвет объекта\*/

}

1. **РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**
   1. **Описание теоретических основ для разработки алгоритмов**

Основной особенностью фреймворка Qtявляется принцип сигналов и слотов. Техника заключается в том, что при определённых условиях (например, изменение данных, изменение состояния объекта либо прямой вызов) автоматически вызывается сигнал, который в свою очередь вызывает слот. То есть сигнал отслеживает условие вызова слота, хотя и сигнал, и слот являются обычными методами класса. Примером может следующая ситуация: при нажатии на кнопку «Открыть» должно открыться окно. В этом случае нажатие на кнопку будет являться сигналом, а функция создания окна – слотом.

Проект состоит из 11 классов. Все они представлены в диаграмме классов (см. чертёж ГУИР.400201.525 РР– приложение Б). Рассмотрим основные классы.

*Server* – класс, объект которого обрабатывает и передает информацию клиентам. Методы этого класса представляют собой взаимодействие с БД и формирование ответных запросов. Основные поля данного класса: сервер (*QTcpServer\* server*), текущий клиент (*User\* currClient*), база данных (*DataBase\* database*), список пользователей, находящихся в сети (*QList<User> users*), тип операции (*int operation*). Основные методы:

* *void login(QString buffer)* – авторизовать пользователя;
* *void signin(QString buffer)* – зарегистрировать пользователя;
* *void sendPacket(int id, int status QString block)* – отправляет ответ клиенту;
* *void sendMessage(QString message)* – отправляет сообщение;
* *void createConversation(QListWidgetItem\*)* – создает новую беседу;
* *void dataReceived()* – обрабатывает полученную от клиента информацию.

*Client* – класс, объект которого обрабатывает серверную информацию и отображает её пользователю. Методы данного класса представляют собой работу с графическом интерфейсом и формирование запросов. Основные поля класса: ID (*int id*), имя (*QString nickname*), логин (*QString login*), пароль (*QString password*), список участников текущей беседы (*QList<User> users*), текущая беседа (*UserItem\* currConversation*), ID голосового собеседника (*int idCaller)*. Основные методы:

* *void sendPacket(int id, QString block)* – отправляет запрос на сервер;
* *void on\_sendButton\_clicked()* – отправляет сообщение;
* *void on\_callButton\_clicked()* – звонит другому пользователю;
* *void on\_addMemberButton \_clicked()* – добавляет собеседника в чат;
* *void createDialog(QListWidgetItem\*)* – создает новую беседу;
* *void on\_listWidge\_Bookt\_itemClicked(QListWidgetItem \*item)* – по нажатию на книгу заполняет все поля в информационной области;
* *void dataReceived()* – обрабатывает полученную от сервера информацию;
* *void fillMessageArea()* – заполняет текстовое поле полученными сообщениями переписки.

*DataBase* – класс, который работает (считывает данные, вносит изменения) с базой данных. Основные методы:

* *QString formConversations(QString name)* – формирует список бесед клиента;
* *QString formMembers(QString name –* формирует список участников беседы;
* *void makeOnline(int id)* – изменяет статус пользователя на «в сети»;
* *QString addToChat(QString idConv, QString idPerson)* – добавляет нового участника в беседу.

Для извлечения информации из базы данных использовались также классы *QSqlDatabase* и *QSqlQuery* из библиотеки Qt *-* ***<****QSQLite****>***. Первый необходим для получения доступа к данным, второй – для их извлечения и внесения изменений посредством отправки SQL-запросов.

* 1. **Описание алгоритмов, разработанных самостоятельно**

Два алгоритма работы программ разобраны в виде блок-схем и представлены на чертежах ГУИР.400201.525 ПД1 (приложение В, оповещение о добавлении в чат собеседника) и ГУИР.400201.525 ПД2 (приложение Г, поиск пользователя по имени).

В данном подразделе рассмотрен алгоритм авторизации пользователя на сервере. Пошаговое объяснение:

Шаг 1. Извлечение из пришедшего от клиента блока его логина и пароля. Действие над блоком данных *buffer*.

Шаг 2. Проверка полученных данных с данными в базе данных. Действие над базой данных *database.*

Шаг 3. Если проверка прошла успешно, то клиенту отправляется код успешной операции и сформированный список контактов и бесед, запоминается ID текущего клиента, в базу данных заносится его IP-адрес. В обратном случае лишь отправляется клиенту код неуспешной операции. Действие над сообщением *message*, базой данных *database* и текущим клиентом *currClient*.

Шаг 4. Если пользователь был авторизован, сервер рассылает всем его контактам, которые сами онлайн, оповещение о том, что он теперь «в сети». Действие над базой данных *database* и клиентами через указатель *currClient*.

Объяснение того же алгоритма на коде с комментариями:

void Server::login(QString buffer)

{

//разделенение полученного блока на логин и пароль

QStringList separate = buffer.split('|');

QString login = separate.at(0);

QString password = separate.at(1);

QString message = "";

//проверка с данными, хранящимися в базе данных

int status = database->checkPasswotd(login, password);

int ID;

if (status == 2)//если логин и пароль совпали

{

ID = database->getID(login); //получение ID

//формирование списка контактов и бесед

message = database->formConversations(QString::number(ID));

currClient->setId(ID); //сохранение ID

checkIP(); //сохранение IP-адреса

}

//отправление блока авторизованному клиенту

qDebug() << "Mess: " << message << endl;

sendPacket(ID\_AUTHO, status, message);

if (status < 2) return; //если не авторизован, то выход из функции

database->makeOnline(ID); //установление статуса онлайн

//формирование списка контактов

QList<int> contactList = database->getContactList(ID);

User user;

int idUser;

//контактам отправляется ID пользователя, который стал онлайн

QString id = QString::number(currClient->getId());

foreach(int idContact, contactList)

{

user.setId(idContact);

idUser = users.indexOf(user, 0); //поиск в списке тех, кто в сети

if (idUser < 0) continue; //если не найден, то поиск следующего

currClient = &users[idUser]; //ставится указатель на найденного

sendPacket(ID\_USRON, 1, id); //отправляется пакет

}

}

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**
   1. **Системные требования**

Минимальные системные требования для программы:

* операционная система: Windows 7 64-bit и выше, Linux;
* оперативная память: 256 MB;
* жёсткий диск: 200 MB свободного места.
  1. **Запуск программы**

Для запуска программы необходимо выполнить следующие действия:

1. Вставить CD-диск в дисковод.
2. Открыть папку в зависимости от операционной системы.
3. Запустить исполняемый файл сервера (*sever.exe* или *server*).
4. Запустить исполняемый файл мессенджера (*messenger.exe* или *messenger*).

Все необходимые библиотеки находятся внутри проекта.

* 1. **Использование программы**

После запуска сервера должно появиться небольшое окно (рис 5.1). Так как для него не предусмотрено никаких настраиваемых параметров и пользователь никак напрямую не взаимодействует с сервером, имеется лишь одна кнопка «Выход» для удобного завершения программы.

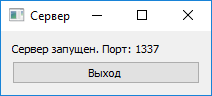


Рис 5.1 – Окно сервера

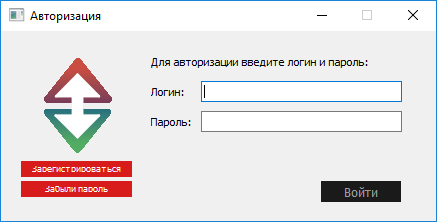


Рис 5.2 – Окно авторизации

После запуска мессенджера появится окно авторизации (рис 5.2). Кнопка «Войти» становится активной только в случае заполнения всех полей (логина и пароля). Также имеются две кнопки «Зарегистрироваться» и «Забыли пароль», которые ведут к одинаковому окну (рис. 5.3), где появляется дополнительное поле повторного ввода пароля. Оно необходимо для точной уверенности в том, что пользователь верно вводит данные.

Каждое окно имеет свою строку состояния, в которой, если есть потребность, выводится наводящая информация.

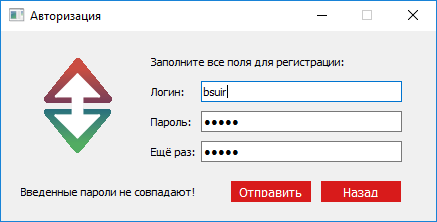


Рис 5.3 – Окно регистрации

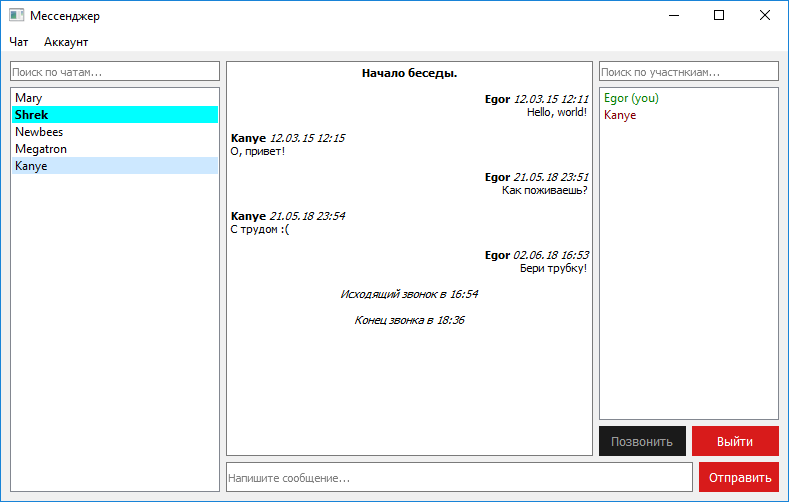


Рис 5.4 – Главное окно мессенджера

После подтверждения авторизация появляется главное окно приложения (рис 5.4). Сначала ни одна беседа не выбрана, поэтому все кнопки являются неактивными. Но при выборе чата с левой стороны мессенджера основные кнопки активируются, заполняются поле сообщений и правый список, который представляет собой участников выбранной беседы. Кнопку «Позвонить» можно нажать в том случае, если собеседник находится в сети. Снизу имеется поле для ввода и отправки сообщений.

Если беседа подсвечена бирюзовым цветом, то это говорит о том, что в ней произошли изменения (например, пришло сообщение или собеседник покинул чат). Также имеется звуковое сопровождение, если изменение произошло во время пользовательской сессии.

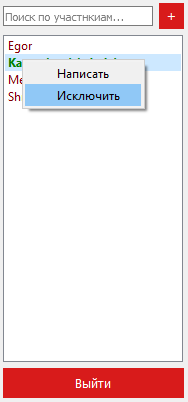


Рис 5.5 – Дополнительные опции

При нажатии правой кнопкой мыши на участника чата (рис. 5.5), можно выбрать опцию «Написать», которая сразу перенесет в диалог с ним (если такового не было, то он будет автоматически создан). В случае, если пользователь является администратором беседы, в опциях есть также кнопка «Исключить», а справа от поля поиска по участникам имеется кнопка «+», которая разрешает добавлять в чат людей из своего списка контактов.

Нажав на «Выйти», клиент покидает беседу, но у него есть возможность вернуться до тех пор, пока все не покинут её (так как тогда вся информация удаляется).

В правом верхнем углу можно выбрать разделы меню (рис. 5.6 и 5.7). Кратко о каждом:

* «Вся переписка» выводит полный текст чата, если он занимает много места и не помещается в фиксированный размер первоначального запроса;
* «Найти людей» показывает список всех пользователей мессенджера (кроме самого клиента и его контактов), чтобы можно было создать диалог;

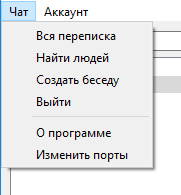


Рис 5.6 – Окно меню «Чат»

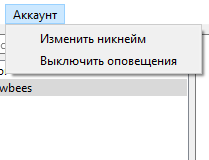


Рис 5.7 – Окно меню «Аккаунт»

* «Создать беседу» запрашивает название новой беседы и создает её, пользователь сразу становится администратором;
* «Выйти» выходит из сети, возвращая к окну авторизации, тем самым позволяя зайти под другим логином и паролем;
* «О программе» выводит базовую информацию о приложении;
* «Изменить порты» необходимо для корректной работы голосового общения (в частности, при запуске нескольких клиентов на одном компьютере);
* «Изменить никнейм» разрешает изменить имя пользователя;
* «Выключить оповещения» отключает все звуковые сопровождения.

1. **ТЕСТИРОВАНИЕ**
   1. **Проверка работоспособности**

Программа в большинстве случаев тестировалась на устройстве со следующими характеристиками:

* операционная система: Windows 10 Корпоративная;
* процессор: Intel® Core™ i5-4590 CPU 3.30 GHz;
* оперативная память: 16 Гб.

Также для проверки работоспособности программа запускалась на 5 других устройствах и во всех случаях неправильного функционирования не наблюдалось.

Если во время работы мессенджера пропадёт соединение с сервером, программа спрячет главное окно и предложит пройти повторную авторизацию, при которой можно будет восстановить связь.

* 1. **Качественные и количественные характеристики**

Во время проведения качественных характеристик (ввода некорректных данных в поля, попытка выбора несоответствующих расширений файлов) отклонений и нарушений в работе программы не было обнаружено.

Во время проведения количественных характеристик (подключения баз данных с большим количеством элементов) программа оставалась работоспособной. Единственным отклонением являлся долгий процесс поиска элементов при использовании поля поиска. После этого алгоритм был пересмотрен, что позволило сократить время поиска. Данный факт будет учтён в заключении.

**ЗАКЛЮЧЕНИЕ**

Темой данного курсового проекта была выбрана «Кроссплатформенный мессенджер с возможностью голосового общения». В соответствии с ней ставились такие задачи, как реализация обмена текстовыми сообщениями, голосовое общение, создание аккаунта с возможностью последующего редактирования, а также разработка простого и удобного графического интерфейса. Со всеми поставленными задачи итоговый проект справляется. В конечном варианте были добавлены и другие функции, предназначенные для комфортного использования.

Во время тестирования программы на её количественные характеристики, было выявлено достаточно сильное увеличение времени поиска элементов. Проблема была частично решена, но для полного устранения этого недостатка необходимо полностью пересмотреть алгоритм поиска, а также попытаться использовать более эффективные методы (например, уже написанные и встроенные в стандартные библиотеки Qt)*.*

В дальнейшем приложение можно усовершенствовать, исправив всевозможное недочёты. На сегодняшний день имеется большое количество мессенджеров (такие как Telegram, Viber, WhatsApp). Для того, чтобы аналог, созданный в процессе работы над курсовым проектом, не отставал по качеству, в будущем его можно доработать и подстроить под определенные тенденции (например, добавление видеочатов, секретных чатов, пересылку файлов, шифрование и другое).

Также данный проект позволил усовершенствовать навыки написания программ, углубить знания в области языка С++ и в области объектно-ориентированного программирования. Он научил правильно использовать справочную и научно-техническую литературу, осуществлять поиск нужной информации, ознакомил с правилами оформления текстовой и графической документации, показал объем знаний и времени, необходимых для выполнения больших проектов.

Таким образом, выполнение как основной задачи, так и выше перечисленных, позволяет объявить об успешном завершении работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алексеев, Е. Р. Программирование на языке С++ в среде Qt Creator / Е. Р. Алексеев [и др.]. – М. : ALT Linux, 2015. – 448 с. : ил. – (Библиотека ALT Linux).
2. Страуструп, Б. Язык программирования С++ / Б. Страуструп – М. : Бином, 2003. – 369 с.
3. Керниган, Б. Язык программирования Си / Б. Керниган, Д. Ритчи – М.: Вильямс, 2013. –304 с.
4. Лафоре, Р. Объектно-ориентированное программирование в C++ / Р. Лафоре. – М. : СПб: Питер; Издание 4-е, 2003. – 928 c.
5. Скляров, В. А. Язык С++ и объектно-ориентированное программирование: справ. пособие / В. А. Скляров. – Мн. : Вышэйшая школа, 1997. – 478 с.
6. Документация фреймворка Qt [Электронный ресурс]. – 2018. – Режим доступа : http://doc.qt.io/.
7. Документация по QSS и QSQL [Электронный ресурс]. – 2018. – Режим доступа : http://doc.crossplatform.ru/qt/4.6.x/.
8. Wikipedia – формат JPEG [Электронный ресурс]. – 2018. – Режим доступа : https://ru.wikipedia.org/wiki/JPEG/.

# **ПРИЛОЖЕНИЕ А**

*(обязательное)*

Структурная схема

# **ПРИЛОЖЕНИЕ Б**

*(обязательное)*

Диаграмма классов

# **ПРИЛОЖЕНИЕ В**

*(обязательное)*

Блок-схема алгоритма оповещения о добавлении в чат собеседника

# **ПРИЛОЖЕНИЕ Г**

*(обязательное)*

Блок-схема алгоритма поиска пользователя по имени

# **ПРИЛОЖЕНИЕ Д**

*(обязательное)*

Листинг программы

#include "client.h"

Client::Client()

{

setupUi(this);

socket = new QTcpSocket(this);

connect(socket, SIGNAL(readyRead()), this, SLOT(dataReceived()));

connect(socket, SIGNAL(connected()), this, SLOT(connectUser()));

connect(socket, SIGNAL(disconnected()), this, SLOT(loggedOut()));

connect(socket, SIGNAL(error(QAbstractSocket::SocketError)), this, SLOT(socketError(QAbstractSocket::SocketError)));

connect(userList, SIGNAL(customContextMenuRequested(QPoint)), this, SLOT(showContextMenu(QPoint)));

userList->setContextMenuPolicy(Qt::CustomContextMenu);

setWindowTitle(tr("Мессенджер"));

messagesList->setPlaceholderText("Выберите чат для начала общения");

message->setPlaceholderText("Напишите сообщение...");

filterChat->setPlaceholderText("Поиск по чатам...");

filterMember->setPlaceholderText("Поиск по участнкиам...");

sendButton->setEnabled(false);

deleteButton->setEnabled(false);

callButon->setEnabled(false);

endButton->setVisible(false);

addMemberButton->hide();

selectedUser = new UserItem();

call = new VoIPClient();

call->setLocalPort(1338);

call->setRemotePort(1339);

connect(call, SIGNAL(endCall()), this, SLOT(endCall()));

messageLength = (quint16) 0;

operation = (quint16) -1;

status = (quint16) -1;

idCaller = -1;

soundOff = false;

suboperation = NULL;

searchWindow = NULL;

}

Client::~Client()

{

socket->disconnectFromHost();

delete selectedUser;

delete call;

}

void Client::on\_sendButton\_clicked()

{

if (message->text().isEmpty() == true) return;

QString packet = QString::number(currConversation->info.id)

+ ',' + QString::number(id) + "|||" + message->text().replace('|', '\\');

sendPacket(ServerCommand::ID\_SENDM, packet);

message->clear();

message->setFocus();

}

void Client::on\_message\_returnPressed()

{

sendButton->click();

}

void Client::dataReceived()

{

QDataStream in(socket);

if(messageLength == 0) {

if(socket->bytesAvailable() < (int) sizeof(quint16)) {

return;

}

in >> messageLength;

}

if(operation == (quint16) -1) {

if(socket->bytesAvailable() < (int) sizeof(quint16)) {

return;

}

in >> operation;

}

if(status == (quint16) -1) {

if(socket->bytesAvailable() < (int) sizeof(quint16)) {

return;

}

in >> status;

}

if(socket->bytesAvailable() < messageLength) {

return;

}

QString packet;

in >> packet;

qDebug() << messageLength << " " << operation << " " << status << " " << packet << endl;

switch (operation)

{

case ID\_SIGIN:

emit statusOperation(ID\_SIGIN, status);

break;

case ID\_AUTHO:

emit statusOperation(ID\_AUTHO, status);

if (status == 2)

{

this->show();

this->setFocus();

}

fillList(packet);

break;

case ID\_CHNPW:

emit statusOperation(ID\_CHNPW, status);

break;

case ID\_GETCV:

fillMessageArea(packet);

break;

case ID\_SENDM:

getMessage(packet);

break;

case ID\_USRON:

changeOnlineStatus(packet);

break;

case ID\_CHGNM:

suboperation->getStatus(status);

changeNickname(packet, status);

break;

case ID\_CRTCV:

addNewConversation(packet);

break;

case ID\_ADDMR:

addToChat(packet);

break;

case ID\_LFTCV:

cleanConversation(packet);

break;

case ID\_ALLUR:

chooseNewConversation(packet);

break;

case ID\_GETIP:

startTalking(packet, status);

break;

case ID\_INCCL:

incomingCall(packet);

break;

case ID\_ENDCL:

endCall();

break;

default:

break;

}

messageLength = (quint16) 0;

operation = (quint16) -1;

status = (quint16) -1;

}

void Client::connectUser()

{

qDebug() << "Successfully connected!";

}

void Client:: logOut()

{

operation = (quint16) ID\_LGOUT;

socket->disconnectFromHost();

}

void Client::loggedOut()

{

this->hide();

emit backOffline(operation);

message->clear();

messagesList->clear();

userList->clear();

conversationsList->clear();

filterChat->clear();

filterMember->clear();

sendButton->setEnabled(false);

deleteButton->setEnabled(false);

callButon->setEnabled(false);

operation = (quint16) -1;

qDebug() << "Logged out.";

}

void Client::socketError(QAbstractSocket::SocketError error)

{

switch(error) {

case QAbstractSocket::HostNotFoundError:

qDebug() << "Error: Could not find server.";

break;

case QAbstractSocket::ConnectionRefusedError:

qDebug() << "Error: Connection refused.";

emit statusOperation(ID\_CNNCT, 0);

break;

case QAbstractSocket::RemoteHostClosedError:

qDebug() << "Error: Connection closed.";

break;

default:

qDebug() << "Error: " << socket->errorString();

}

}

void Client::connectToServer(QString IP, int port) {

qDebug() << "Connecting...";

socket->abort();

//Connection:

socket->connectToHost(IP, port);

}

void Client::setLoginInfo(QString login, QString password)

{

this->login = login;

this->password = password;

}

void Client::sendPacket(int operation, QString buffer)

{

QByteArray packet;

QDataStream out(&packet, QIODevice::WriteOnly);

out << (quint16) 0;

out << (quint16) operation;

out << buffer;

out.device()->seek(0);

out << (quint16) (packet.size() - 2\*sizeof(quint16));

socket->write(packet);

}

void Client::logIn()

{

QString logInfo = login + '|' + password;

this->sendPacket(ID\_AUTHO, logInfo);

}

void Client::sigIn()

{

QString logInfo = login + '|' + password;

this->sendPacket(ID\_SIGIN, logInfo);

}

void Client::changePassword()

{

QString logInfo = login + '|' + password;

this->sendPacket(ID\_CHNPW, logInfo);

}

void Client::fillList(QString packet)

{

if (packet.isEmpty() == true) return;

QStringList separateUser = packet.split('|');

QStringList separateInfo;

User user;

int checkRead;

separateInfo = separateUser.value(0).split(',');

id = separateInfo.value(0).toInt();

nickname = separateInfo.value(1);

if (nickname.isEmpty() == true) checkNickname();

for (int i = 1; i < separateUser.size(); i++)

{

separateInfo = separateUser.value(i).split(',');

user.id = separateInfo.value(0).toInt();

user.nickname = separateInfo.value(1);

checkRead = separateInfo.value(3).toInt();

conversations << user;

UserItem\* item = new UserItem();

conversationsList->addItem(item);

item->info.id = user.id;

item->info.nickname = user.nickname;

item->adminID = separateInfo.value(2).toInt();

item->setText(user.nickname);

if (checkRead == 0) changeFont(ID\_NEW, item);

//conversationsList->setCurrentItem(item);

}

}

void Client::on\_conversationsList\_itemClicked(QListWidgetItem \*item)

{

sendButton->setEnabled(true);

deleteButton->setEnabled(true);

changeFont(ID\_STD, item);

currConversation = (UserItem\*) item;

int id = currConversation->info.id;

sendPacket(ID\_GETCV, QString::number(id));

}

void Client::fillMessageArea(QString packet)

{

QStringList separateInfo = packet.split("||");

QStringList infoHeader = separateInfo.value(1).split('|');

QStringList person;

int adminID = separateInfo.value(0).toInt();

currConversation->adminID = adminID;

userList->clear();

users.clear();

callButon->show();

if (adminID == id)

{

addMemberButton->show();

filterMember->setMaximumWidth(150);

}

else

{

addMemberButton->hide();

filterMember->setMaximumWidth(180);

}

if (idCaller == currConversation->info.id)

{

callButon->setVisible(false);

endButton->setVisible(true);

}

else

{

callButon->setVisible(true);

endButton->setVisible(false);

}

for (int i = 0; i < infoHeader.size(); i++)

{

person = infoHeader.value(i).split(',');

UserItem\* item = new UserItem();

item->info.id = person.value(0).toInt();

item->info.nickname = person.value(1);

users << item->info;

if (person.value(3) == '1') continue;

item->setText(person.value(1));

if (item->info.id == id) item->setText(item->text() + " (you)");

if (item->info.id == adminID)

{

QFont font;

font.setWeight(80);

item->setFont(font);

item->setText(item->text() + " (admin)");

callButon->hide();

}

if (person.value(2).toInt() == 1)

{

item->setForeground(Qt::darkGreen);

callButon->setEnabled(true);

}

else

{

item->setForeground(Qt::darkRed);

callButon->setEnabled(false);

}

userList->addItem(item);

}

userList->sortItems(Qt::AscendingOrder);

messagesList->clear();

QStringList messages = separateInfo.value(2).split("\n");

QTextCursor cursor = messagesList->textCursor();

QTextBlockFormat textBlockFormat = cursor.blockFormat();

QString info;

User user;

int i = 1;

textBlockFormat.setAlignment(Qt::AlignCenter);

cursor.mergeBlockFormat(textBlockFormat);

messagesList->setTextCursor(cursor);

messagesList->setFontWeight(86);

if (messages.value(0).at(0) == '\_') messagesList->append("Начало беседы.");

else messagesList->append("Выберите \"Вся переписка\"");

messagesList->setFontWeight(50);

messagesList->append("");

for (i; i < messages.size(); i++)

{

separateInfo = messages.value(i).split('|');

qDebug() << separateInfo << endl;

user.id = separateInfo.value(0).toInt();

if (user.id < 0)

{

addInfoMessage(user.id, separateInfo.value(1));

continue;

}

info = users.value(users.indexOf(user, 0)).nickname;

info = "<b>" + info + "</b>" + " <em>" + separateInfo.value(1) + " " + separateInfo.value(2) + "</em>";

if (user.id == id) textBlockFormat.setAlignment(Qt::AlignRight);

else textBlockFormat.setAlignment(Qt::AlignLeft);

cursor.mergeBlockFormat(textBlockFormat);

messagesList->setTextCursor(cursor);

messagesList->append(info);

messagesList->append(separateInfo.value(3) + "\n");

}

if (adminID == -1 && userList->count() == 1)

{

UserItem\* item = new UserItem();

item->setText(currConversation->text());

item->setForeground(Qt::darkGray);

userList->addItem(item);

User user;

user.id = -1;

user.nickname = currConversation->text();

users << user;

callButon->setEnabled(false);

}

}

void Client::addInfoMessage(int operation, QString id)

{

QTextCursor cursor = messagesList->textCursor();

QTextBlockFormat textBlockFormat = cursor.blockFormat();

textBlockFormat.setAlignment(Qt::AlignCenter);

cursor.mergeBlockFormat(textBlockFormat);

messagesList->setTextCursor(cursor);

QString info;

if (operation > -3)

{

User user;

user.id = id.toInt();

info = users.value(users.indexOf(user, 0)).nickname;

}

switch(operation)

{

case -1:

info = "Пользователь " + info + " был добавлен в чат.";

break;

case -2:

info = "Пользователь " + info + " больше не состоит в чате.";

break;

case -3:

info = "Исходящий звонок в " + id;

break;

case -4:

info = "Конец звонка в " + id;

break;

default:

break;

}

info = "<em>" + info + "</em>";

messagesList->append(info);

messagesList->append("");

cursor = messagesList->textCursor();

textBlockFormat = cursor.blockFormat();

textBlockFormat.setAlignment(Qt::AlignLeft);

cursor.mergeBlockFormat(textBlockFormat);

messagesList->setTextCursor(cursor);

}

void Client::on\_filterChat\_textChanged(const QString &text)

{

QString enteredText = text.toLower();

QListWidgetItem \*item;

QString name;

for (int i = 0; i < conversationsList->count(); i++)

{

item = conversationsList->item(i);

name = item->text().toLower();

if (name.indexOf(enteredText) == 0)

{

if (item->isHidden()) item->setHidden(false);

}

else item->setHidden(true);

}

}

void Client::on\_filterMember\_textChanged(const QString &text)

{

QString enteredText = text.toLower();

QListWidgetItem \*item;

QString name;

for (int i = 0; i < userList->count(); i++)

{

item = userList->item(i);

name = item->text().toLower();

if (name.indexOf(enteredText) == 0)

{

if (item->isHidden()) item->setHidden(false);

}

else item->setHidden(true);

}

}

void Client::getMessage(QString packet)

{

QStringList separateInfo = packet.split("||");

User conversation;

conversation.id = separateInfo.value(0).toInt();

if (conversation.id != currConversation->info.id)

{

int num = conversations.indexOf(conversation);

conversation.nickname = conversations.value(num).nickname;

QList<QListWidgetItem\*> list = conversationsList->findItems(conversation.nickname, Qt::MatchExactly);

if (list.isEmpty() == true)

{

QString packet = separateInfo.value(0) + '|' + separateInfo.value(2) + '|';

conversation.nickname = separateInfo.value(2);

addNewConversation(packet);

list = conversationsList->findItems(conversation.nickname, Qt::MatchExactly);

}

QListWidgetItem \*item = list.first();

changeFont(ID\_NEW, item);

if (soundOff == false) QSound::play("alarm\_001.wav");

}

else

{

separateInfo = separateInfo.value(1).split('|');

QTextCursor cursor = messagesList->textCursor();

QTextBlockFormat textBlockFormat = cursor.blockFormat();

cursor = messagesList->textCursor();

textBlockFormat = cursor.blockFormat();

if (id == separateInfo.value(0).toInt()) textBlockFormat.setAlignment(Qt::AlignRight);

else textBlockFormat.setAlignment(Qt::AlignLeft);

cursor.mergeBlockFormat(textBlockFormat);

messagesList->setTextCursor(cursor);

User user;

QString info;

user.id = separateInfo.value(0).toInt();

if (users.indexOf(user, 0) == -1) user.id = -1;

info = users.value(users.indexOf(user, 0)).nickname;

info = "<b>" + info + "</b>" + " <em>" + separateInfo.value(1) + " " + separateInfo.value(2) + "</em>";

messagesList->append(info);

messagesList->append(separateInfo.value(3));

}

}

void Client::changeFont(int idOperation, QListWidgetItem \*item)

{

QFont font;

if (idOperation == ID\_STD)

{

font.setWeight(50);

item->setBackground(Qt::white);

}

else

{

font.setWeight(80);

item->setBackground(Qt::cyan);

}

item->setFont(font);

}

void Client::changeOnlineStatus(QString idUser)

{

User user;

user.id = idUser.toInt();

int num = users.indexOf(user);

if (num == -1) return;

if (userList->count() == 0) return;

QListWidgetItem \*item = userList->findItems(users[num].nickname, Qt::MatchStartsWith).first();

if (item->textColor() == Qt::darkGray) return;

if (item->textColor() == Qt::darkGreen)

{

item->setTextColor(Qt::darkRed);

}

else

{

item->setTextColor(Qt::darkGreen);

callButon->setEnabled(true);

}

}

void Client::on\_exit\_triggered()

{

QMessageBox::StandardButton reply;

reply = QMessageBox::question(this, "Выход", "Вы уверены, что хотите выйти?",

QMessageBox::Yes|QMessageBox::No);

if (reply == QMessageBox::Yes) logOut();

}

void Client::on\_disableSound\_triggered(bool checked)

{

soundOff = checked;

}

void Client::on\_about\_triggered()

{

QMessageBox::about(this, "О программе", "Данная программа просто супер мегакласс, как говорится, шик, блекс, бурлеск.\nПользуемся. девочки!");

}

void Client::on\_changeNick\_triggered()

{

suboperation = new Interaction();

suboperation->setModal(true);

suboperation->chooseMode(MODE\_NAME);

connect(suboperation, SIGNAL(sendInfo(QString)), this, SLOT(getNewNickname(QString)));

connect(suboperation, SIGNAL(rejected()), this, SLOT(checkNickname()));

suboperation->setNickname(nickname);

suboperation->setFixedSize(300, 100);

suboperation->show();

}

void Client::getNewNickname(QString newNickanme)

{

sendPacket(ID\_CHGNM, newNickanme);

}

void Client::on\_newConversation\_triggered()

{

suboperation = new Interaction();

suboperation->setModal(true);

suboperation->chooseMode(MODE\_CONV);

connect(suboperation, SIGNAL(sendInfo(QString)), this, SLOT(getConvName(QString)));

suboperation->setFixedSize(300, 100);

suboperation->show();

}

void Client::getConvName(QString name)

{

QString packet = "-1|" + name;

sendPacket(ID\_CRTCV, packet);

disconnect(suboperation, SIGNAL(sendInfo(QString)), this, SLOT(getNewNickname(QString)));

delete suboperation;

}

void Client::on\_userList\_itemClicked(QListWidgetItem \*item)

{

UserItem\* user = (UserItem\*) item;

selectedUser->info.id = user->info.id;

selectedUser->info.nickname = item->text();

}

void Client::on\_userList\_itemPressed(QListWidgetItem \*item)

{

on\_userList\_itemClicked(item);

}