

**СОДЕРЖАНИЕ**

Введение ……………………………………………………………………………...3

1 Предметная область ……………………………………………………………….5

[1.1 Язык программирования Python](#_Toc477445703) ……………………………………………..5

[1.2 Среда разработки PyQt](#_Toc477445704) ………………………………………………………..6

1.3 Библиотека для Telegram – Telethon …………………………………………7

[1.4 Библиотека для ВКонтакте – vk\_api](#_Toc477445706) ………………………………………...7

[1.5 Таблица стилей Qt QSS ……………………………………………………….8](#_Toc477445704)

2 Концепция приложения …………………………………………………………...9

[2.1 Создание структуры приложения …………………………………………..10](#_Toc477445708)

2.2 Графическая составляющая приложения …………………………………..13

[2.3 Логическая составляющая приложения ……………………………………16](#_Toc477445710)

Заключение………………………………………………………………………….24

Список использованных источников……………………………………………...25

**ВВЕДЕНИЕ**

Сложно представить современный мир без компьютеров, информационных технологий, и интернета, так как они уже успели занять особое место в жизни каждого человека. Большой популярностью пользуются социальные сети. Сейчас у каждого человека есть аккаунт в не одной социальной сети, и он посещает их несколько раз по дню. Это позволяет осуществлять коммуникации между родными и близкими людьми, быстро передавать необходимую информацию и даже помогает человеку заниматься саморазвитием.

В нашем случае нас интересует возможность использования средств коммуникации не в одной социальной сети, а сразу в нескольких. Для реализации данной функции существует мультипротокольные чат-клиенты или мультипротокольные мессенджеры.

Актуальность мультипротокольных чат-клиентов в современном мире состоит в том, что на сегодняшний день очень много людей используют мессенджеры, чаты, социальные сети для передачи какой-либо информации между собой.

Главное преимущество мультипротокольных чат-клиентов состоит в том, что они позволяют одновременное пользоваться средствами коммуникаций в нескольких чатах, мессенджерах и даже в социальных сетях. Это позволяет экономить время, которое тратится на переключение между приложениями и это удобно в плане общения, так как пользователь может сразу связаться с нужным ему контактом в нескольких чатах и будет больше вероятности, что информация дойдет до адресата.

Недостаток мультипротокольных чат-клиентов чаще состоит в том, что они чаще всего не поддерживают все функции официального клиента, например не отображаются стикеры или же нельзя прослушать голосовые сообщения, а также могут не поддерживаться групповые беседы.

Целью данного курсового проекта является разработка программного обеспечения, которое позволит в рамках одного приложения пользоваться средствами коммуникации в минимум двух чат-клиентах, мессенджерах или социальных сетях, в нашем случае «Telegram» и «ВКонтакте».

**1 Предметная область**

Первым этапом разработки приложения является анализ предметной области. Необходимо определить комплекс задач и целей для реализации, которых создается данное приложение.

Необходимо изучить структуру чат-клиентов, мессенджеров или социальной сети. В данном курсовом проекте будет использоваться социальная сеть «Вконтакте», а точнее её API и мессенджер «Telegram», я выбрал их потому что на данный момент это самая популярная социальная сеть и мессенджер, а также у них имеется открытый код в свободном доступе и документации, что очень упрощает процесс разработки. Для того чтобы разработать приложения нужно подробно изучить документацию и библиотеки, которые позволяют работать с «Вконтакте» и «Telegram», также выбрать язык программирования и среду разработки, где будет создаваться приложение. Для того чтобы у приложения был спрос нужно продумать логическую структуру, так чтобы у приложения не было серьезных ошибок, либо они были незаметны пользователю, а также разработать понятный и удобный интерфейс, чтобы пользователь не терялся и понимал как работать с приложением.

* 1. **Язык программирования Python**

Python – высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объем полезных функций.

Python поддерживает несколько парадигм программирования: структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты – динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений и удобные высокоуровневые структуры данных. Код в Python организовывается в функции и классы, которые могут объединяться в модули[1].

* 1. **Среда разработки PyQt**

PyQt – набор привязок графического фреймворка Qt для языка программирования Python, выполненный в виде расширения Python. PyQt работает на всех платформах, поддерживаемых Qt: Linux и другие UNIX-подобные OC, Mac OS X и Windows. Существует 2 версии: PyQt5, поддерживающий Qt 5, и PyQt4, поддерживающий Qt4. PyQt практически полностью реализует возможности Qt. А это более 600 классов, более 6000 функций и методов, включая:

* cуществующий набор виджетов графического интерфейса;
* cтили виджетов;
* доступ к базам данных с помощью SQL;
* qScintilla, основанный на Scintilla виджет текстового редактора;
* поддержку интернационализации (i18n);
* парсер XML;
* поддержку SVG;
* интеграцию с WebKit, движком рендеринга HTML;
* поддержку воспроизведения видео и аудио.

PyQt также включает в себя Qt Designer – дизайнер графического интерфейса пользователя. Программа pyuic генерирует Python код из файлов, созданных в Qt Designer. Это делает PyQt очень полезным инструментом для быстрого прототипирования. Кроме того, можно добавлять новые графические элементы управления, написанные на Python, в Qt Designer[6].

* 1. **Библиотека для Telegram – Telethon**

Цель библиотеки состоит в том, чтобы иметь существующую библиотеку MTProto, имеющую практически любые зависимости (действительно, везде, где доступен Python, можно запустить эту библиотеку).

Написание на Python означает, что производительность будет нигде не близка к другим реализациям, написанным, например, Java, C ++, Rust или почти любым другим скомпилированным языком. Однако библиотека оказывается довольно приличной для обычных операций, таких как отправка сообщений, получение обновлений или других сценариев. Загрузка файлов может быть заметно медленнее

Если libssl доступно в системе, библиотека будет использовать его для ускорения некоторых критических частей, таких как шифрование и дешифрование сообщений. Файлы будут отправляться и загружаться быстрее.

Основное внимание уделяется тому, чтобы все было чисто и просто, чтобы каждый мог понять, как работает работа с MTProto и Telegram.

* 1. **Библиотека для ВКонтакте - vk\_api**

Работать с API «ВКонтакте» доводилось многим, и каждый может сказать: работать с ним напрямую неудобно. Разумеется, было написано множество библиотек, предоставляющих удобный интерфейс работы с ним. Их используют для облегчения работы и ускорения разработки. Одна из таких библиотек vk\_api для Python, она содержит все методы, которые находится в документации в разделе «Разработчикам» на сайте vk.com, это очень сильный инструмент, который существенно упрощает процесс разработки.

* 1. **Таблица стилей Qt QSS**

QSS - это механизм, позволяющий настраивать внешний вид [виджетов](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B6%D0%B5%D1%82" \o "Виджет) [библиотеки](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) [Qt](https://ru.wikipedia.org/wiki/Qt" \o "Qt).

QSS в значительной части был вдохновлён каскадными таблицами стилей CSS для HTML, вследствие чего имеет похожий синтаксис. В частности, как и в CSS, в QSS можно изменять форму, цвета, прозрачность элемента, а также визуальную реакцию на события (такие, как нажатие кнопки). Стили можно присоединять как к отдельному компоненту так и к ко всему приложению, с помощью метода ::setStyleSheet(), имеющегося как у отдельных виджетов, так и у объекта QApplication.

Qt Designer предоставляет возможность интеграции QSS-стилей, что упрощает их тестирование и разработку. Также, при запуске приложения на Qt, можно применить к нему таблицу стилей, воспользовавшись синтаксисом командной строки вида:MyApp -stylesheet MyStyle.qss

Поддержка QSS появилась в Qt начиная с версии 4.3, однако до версии 4.5 этот механизм не работал на платформе Mac OS X

Механизм QSS позволяет полностью отделить визуальный дизайн от разработки приложения и привлечь к стилизации приложения веб-дизайнеров. Наряду с подсистемами QtScript и QML — это один из шагов по сближению Qt-программирования с web-разработкой.

1. **Концепция приложения**

Приложения оформлено в темных тонах, они были выбраны, потому что они имеют больше спрос у пользователей и не бросаются сильно своей яркостью в глаза.

Во время разработки приложения были использованы принципы объектно-ориентированного программирования. Когда пользователь открывает приложение, перед ним открывается главное меню приложения, где справа расположены вкладки с выбором канала в моем случаем «Telegram», «ВКонтакте». Внутри каждой вкладки расположены кнопки для входа, при нажатии на эту кнопку происходит обращения к классу авторизации «Telegram» или «ВКонтакте» и запускает окно для авторизации.

Если пользователю нужно будет авторизоваться в «Telegram», то необходимо будет ввести номер телефона и его логин. Для авторизации в «ВКонтакте» пользователь должен ввести, свой номер телефона по которому он регистрировался и пароль. В каждом окне авторизации пользователь может посмотреть свой пароль, чтобы убедиться в его правильности, а также может сохранить свою учетную запись для того чтобы постоянно не вводить свои данные.

После успешной авторизации загружаются контакты пользователя вместе с их фотографиями. Теперь вместо кнопки авторизации во вкладке «Telegram» отображается два списка, один отображает контакты пользователя, с которыми он общался, другой каналы на которые он подписан. Во вкладке «ВКонтакте» отображается также контакты пользователя, с которыми он общался и его беседы. Все в этих списках отсортировано.

При выборе одного из контактов во вкладке «Telegram» или «ВКонтакте», на элементе расположенным справа открывается вкладка с именем этого контакта и внутри этой вкладки загружается сообщения пользователя с этим контактом. Имя пользователя и контакта выделены жирным шрифтом, чтобы можно было легко сориентироваться кому принадлежит то или иное сообщение. Внизу этого компонента с вкладками расположено поле для набора сообщений, а также кнопка для их отправки.

* 1. **Создание структуры приложения**

Для решения этой задачи будет использован способ динамического создания объектов на языке Python в среде разработки PyQt. Технология этого способа заключается в том, что все компоненты приложения будут созданы посредством написания определенного кода, а не с помощью QtDesigner. В текстовом редакторе, в моем случае geany, создается файл с расширением .py. Для того чтобы работать в PyQt, необходимо подключить библиотеки, которые нужны будут в процессе разработки. Подключение необходимо библиотек представлено на листинге 1.

from PyQt5 import QtWidgets, QtCore

from PyQt5.QtWidgets import QApplication, QGridLayout,QTabWidget, QWidget, QCheckBox, QLineEdit, QLabel, QListView, QVBoxLayout, QHBoxLayout, QListWidget, QListWidgetItem, qApp, QAction

from PyQt5 import QtCore, QtGui, uic

from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5.Qt import QImage, QLabel, QPixmap, QPainter, QItemSelectionModel

from PyQt5.QtGui import QStandardItem, QFont, QStandardItemModel, QPixmap, QIcon, QTextCursor, QTextImageFormat

from PyQt5.QtCore import QTimer, QSize, Qt

Листинг 1 - Подключение библиотек

Ниже расположен листинг 2, отвечающий за динамическое создание объектов, которые пригодятся в разработке приложения.

self.layout\_chats = QtWidgets.QGridLayout(self)

self.layout\_chats.setAlignment(Qt.AlignTop)

self.telegram\_list = QtWidgets.QListWidget(self)

self.telegram\_list.setObjectName("telegram\_list")

self.telegram\_list.setFixedSize(220,700)

self.tab\_chats = QtWidgets.QTabWidget(self)

self.tab\_chats.setFixedSize(210, 750)

self.tab\_chats.setObjectName("tab\_chats")

self.users\_telegram = QtWidgets.QListWidget(self)

self.users\_telegram.setObjectName("users\_telegram")

self.users\_telegram.setIconSize(QSize(32,32))

self.users\_telegram.setFixedSize(200,340)

self.users\_telegram.itemClicked.connect(self.item\_changed\_telegram)

self.channels\_telegram = QtWidgets.QListWidget(self)

self.channels\_telegram.setObjectName("channels\_telegram")

self.channels\_telegram.setIconSize(QSize(32,32))

self.channels\_telegram.setFixedSize(200,340)

self.channels\_telegram.itemClicked.connect(self.item\_changed\_telegram\_channels)

self.wgt\_us\_chan = QtWidgets.QWidget()

self.users\_channels = QtWidgets.QVBoxLayout(self)

self.users\_channels.addWidget(self.users\_telegram)

self.users\_channels.addWidget(self.channels\_telegram)

self.wgt\_us\_chan.setLayout(self.users\_channels)

self.item\_users\_channels = QListWidgetItem(self.telegram\_list)

self.item\_users\_channels.setSizeHint(self.wgt\_us\_chan.sizeHint())

self.telegram\_list.setItemWidget(self.item\_users\_channels, self.wgt\_us\_chan)

self.telegram\_connect = QtWidgets.QPushButton("Войти в Telegram!", self)

self.telegram\_connect.setFixedSize(180, 30)

self.telegram\_connect.clicked.connect(self.clicked\_chats\_telegram)

self.wgt\_telegram = QtWidgets.QWidget()

self.layout\_users\_telegram = QtWidgets.QVBoxLayout(self)

self.layout\_users\_telegram.addWidget(self.telegram\_connect)

self.wgt\_telegram.setLayout(self.layout\_users\_telegram)

self.item\_telegram = QListWidgetItem(self.users\_telegram)

self.item\_telegram.setSizeHint(self.wgt\_telegram.sizeHint())

self.users\_telegram.setItemWidget(self.item\_telegram, self.wgt\_telegram)

self.tab\_chats.addTab(self.telegram\_list, "Telegram")

Листинг 2 – Динамическое создание объектов

Как видно из листинга 2 сначала создается виджет QGridLayout, который позволяет нам удобно расположить объекты в приложении. Далее создается виджет QListWidget, в котором будут располагаться списки контактов и каналов, у виджета задается фиксированный размер, чтобы он не растягивался по всему окну. Следующим шагом создается два отдельных QListWidget, в которых по отдельности расположатся спикски контактов и каналов, эти списки помещаются в виджет QVBoxLayout, чтобы расположить их в QListWidget, который создавался до этого. Потом соответственно QListWidget с готовыми списками, добавляется в QGridLayout, для отображения на форме приложения.

На рисунке 1 можно увидеть как расположились объекты. Но в данный момент используется стандартный интерфейс, что не выглядит визуально привлекательно и пользователь может не заинтересоваться нашим приложением. Эту проблему можно решить с помощью инструмента QSS[2].

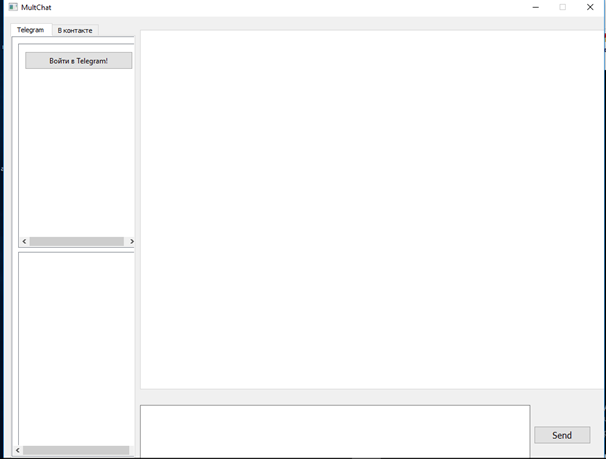


Рисунок 1 – Вид приложения со стандартным интерфейсом

**2.2 Графическая составляющая приложения**

Ниже приведен листинг 3, который задает характеристики заднего фона, вкладок, кнопок, текстовых редакторов, списков виджетов, а также различные события при наведение курсора на компонет и нажатия на него.

QWidget

{

color: #b1b1b1;

background-color: #323232;

}

QWidget:item:hover

{

background-color: QLinearGradient( x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #ffa02f, stop: 1 #ca0619);

color: #000000;

}

QWidget:item:selected

{

background-color: QLinearGradient( x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #ffa02f, stop: 1 #d7801a);

}

QPushButton

{

color: #b1b1b1;

background-color: QLinearGradient( x1: 0, y1: 0, x2: 0, y2: 1, stop: #565656, stop: 0.1 #525252, stop: 0.5 #4e4e4e, stop: 0.9 #4a4a4a, stop: 1 #464646);

border-width: 1px;

border-color: #1e1e1e;

border-style: solid;

border-radius: 6;

padding: 3px;

font-size: 12px;

padding-left: 5px;

padding-right: 5px;

}

QPushButton:pressed

{

background-color: QLinearGradient( x1: 0, y1: 0, x2: 0, y2: 1, stop: 0

#2d2d2d, stop: 0.1 #2b2b2b, stop: 0.5 #292929, stop: 0.9 #282828, stop: 1 #252525);

}

QPushButton:hover

{

border: 2px solid QLinearGradient( x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #ffa02f, stop: 1 #d7801a);

}

QTextEdit:focus

{

border: 2px solid QLinearGradient( x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #ffa02f, stop: 1 #d7801a);

}

QTextEdit

{

background-color: #242424;

}

Листинг 3 – задание стилей для виджетов

В листинге 3 задаются стили для объектов, а также псевдоклассы, когда происходят различные события с этими объектами. В данном листинге можно видеть что, для компонента item задается псевдокласс hover, который обрабатывает события наведения курсора на объект, а также задаются стили на событие, когда нажата кнопка.

Как видим на рисунке 2, используя для виджетов инструмент QSS, приложение стало выглядеть визуально привлекательно, у такого приложения уже будет больше спрос у пользователей[4].

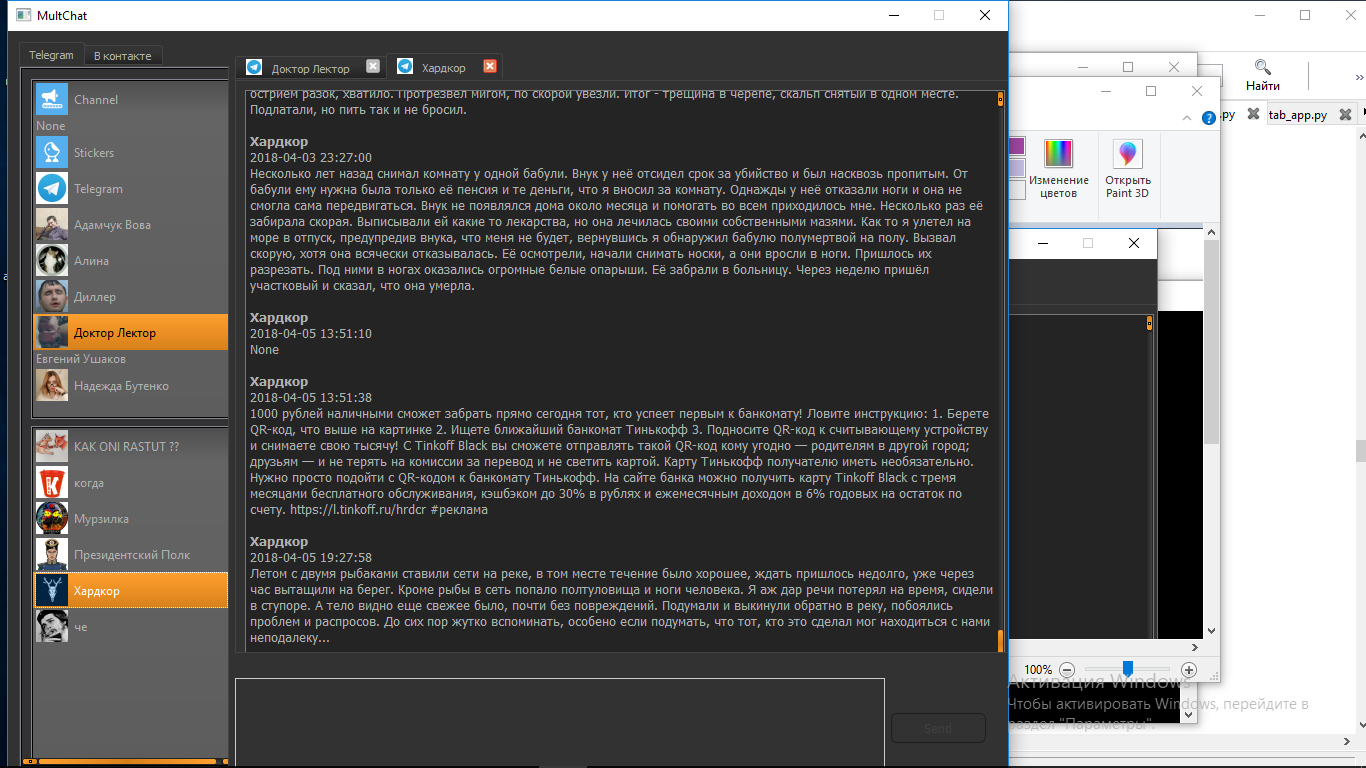


Рисунок 2 – Вид приложения с использованием инструмента QSS

* 1. **Логическая составляющая приложения**

Когда пользователь пытается зайти в официальный клиент «Telegram» или «ВКонтакте» перед ним появляется поля, куда он может ввести свои данные для авторизации. Для того чтобы в приложение была возможность для авторизации, был использован принцип объектно-ориентированного программирования. Созданы два класса, которые передавали данные для авторизации пользователя в определенные функции главного класса. На листинге 4 продемонстрирована функция, которая позволяет авторизоваться пользователю в «Telegram»:

def connect\_telegram(self):

if self.flag\_telegram == False:

if len(self.login\_telegram) and len(self.phone\_telegram):

self.client=TelegramClient(str(self.login\_telegram), api\_id,api\_hash)

self.client.connect()

if not self.client.is\_user\_authorized():

self.client.send\_code\_request(self.phone\_telegram)

self.client.sign\_in(self.phone\_telegram,input('Enter the code:)

self.result1=self.client(GetDialogsRequest(offset\_date=None,offset\_id=0, offset\_peer = InputPeerEmpty(),limit = 200))

self.flag\_telegram = True

self.user.extend(self.result1.users)

self.channel.extend(self.result1.chats)

if self.flag\_photo\_telegram == False:

try:

for user\_value in range(0, len(self.user), 1):

users = self.user[user\_value]

self.client.download\_profile\_photo(users)

for channels\_value in range(0,len(self.channel), 1):

channels = self.channel[channels\_value]

self.client.download\_profile\_photo(channels)

except ValueError:

pass

except TimeoutError:

pass

else:

self.timer\_autoriziting\_telegram.stop()

self.telegram\_contact()

self.channels\_connect()

Листинг 4 – Авторизация в «Telegram»

В описанном листинге выше, сначала принимаются данные с класса авторизации, которые передают телефон и логин пользователя. Если пользователь авторизуются в данном приложении, в канале «Telegram» впервые, то на телефон придет сообщение с кодом для авторизации. Когда авторизации прошла успешно, происходит загрузка сообщений и фотографий контактов и каналов[8].

После авторизации надо отобразить контакты пользователя и каналы. На листинге 5 продемонстрированы эти функции.

def telegram\_contact(self):

self.users\_telegram.clear()

for user\_value in range(0, len(self.user), 1):

users = self.user[user\_value]

if users.last\_name!=None:

user\_n = str(users.first\_name) + " " + str(users.last\_name)

self.listItem = QListWidgetItem(user\_n)

else:

user\_n = str(users.first\_name)

self.listItem = QListWidgetItem(user\_n)

if users.username!=None:

self.listItem.setIcon(QIcon(str(users.username) + ".jpg"))

else:

self.listItem.setIcon(QIcon(str(users.first\_name) + ".jpg"))

self.users\_telegram.addItem(self.listItem)

self.users\_telegram.sortItems()

self.users\_telegram.show()

def channels\_connect(self):

self.channels\_telegram.clear()

for channels\_value in range(0,len(self.channel), 1):

channels = self.channel[channels\_value]

self.listItem = QListWidgetItem(channels.title)

self.listItem.setIcon(QIcon(str(channels.username)+".jpg)

self.channels\_telegram.addItem(self.listItem)

self.channels\_telegram.sortItems()

self.channels\_telegram.show()

Листинг 5 – Отображение контактов и каналов «Telegram»

В данных функциях загружаются контакты, беседы и каналы в компонент QListWidget, для отображения фотографий используется компонент QIcon, также с помощью метода sortItems все данные в QListWidget сортируется по алфавиту, что позволяет пользователю быстро найти нужным ему контакт, канал или беседу[9].

При выборе нужного пункта в списках, происходит загрузка сообщений, для того что обновлять сообщения используется таймер. В приложения используется всего один таймер для обновления сообщений в контактах, каналах и беседах, потому что при использовании нескольких таймеров приложения начинает тормозить и им становится невозможно пользоваться. Для того чтобы сообщения обновляясь в определенном месте, используются логические переменные, так называемые флаги, которые могут принимать всего два значения – True и False. Например, если пользователь выбирает контакт из «Telegram», то ответственный за контакты флаг принимает значение True, а все остальные False. Функция, реализующая обновления сообщений в «Telegram» представленная на листинге 6.

def message\_telgram(self):

login\_telegram = ""

users\_telegram = ""

if self.flag\_telegram\_message == True:

self.message\_telegram.clear()

self.message\_telegram.setVisible(True)

self.message\_telegram.setReadOnly(True)

for user\_val in range(0,self.users\_telegram.count(),1):

users = self.user[user\_val]

log\_tl = str(users.first\_name) + " " + str(users.last\_name)

if(self.tabBar.tabText(self.tab\_message.currentIndex())==str(users.firs t\_name)) or (self.tabBar.tabText(self.tab\_message.currentIndex())==log\_tl) or (self.tabBar.tabText(self.tab\_message.currentIndex())==str(users.username)):

result=self.client(GetHistoryRequest(users,limit=20000000,

offset\_date=None,offset\_id=0,max\_id=0,min\_id=0,add\_offset=0))

try:

message = result.messages

for message\_value in range(0, len(message), 1):

messages = message[message\_value]

if (self.my\_self.id == messages.from\_id):

if self.my\_self.last\_name == None:

login\_telegram = str(self.my\_self.first\_name)

else:

login\_telegram=str(self.my\_self.first\_name)

" "+str(self.my\_self.last\_name)

cursor=QTextCursor(self.message\_telegram.document())

cursor.insertHtml("<big>"+"<b>"+login\_telegram+"</b>"+"<br>"+str(messages.date) +"<br>"+str(messages.message) +"<br><br>"+"</big>")

self.message\_telegram.moveCursor(QtGui.QTextCursor.End)

else:

if users.last\_name == None:

users\_telegram=str(users.first\_name)

else:

users\_telegram=str(users.first\_name)+

" "+str(users.last\_name)

cursor=QTextCursor(self.message\_telegram.document())

cursor.insertHtml("<big>"+"<b>"+users\_telegram+"</b>"+"<br>"+str(messages.date) + "<br>" + str(messages.message) + "<br><br>"+"</big>")

self.message\_telegram.moveCursor(QtGui.QTextCursor.End)

except ValueError:

pass

except TimeoutError:

pass

except MemoryError:

pass

except OverflowError:

pass

Листинг 6 – Обновление сообщений «Telegram»

В данной функции сначала определяется, какой выбран контакт или канал, потом этот контакт сравнивается с теми контактами, которые загрузились при авторизации, если контакты совпали, то вызывается метод, который загружает сообщения с выбранным контактом. Далее происходит разделение сообщений на те, которые относятся к пользователю и которые относятся к выбранному контакту и происходит форматирование вывода сообщения c помощью языка разметки html. Процесс обновления сообщений в функции происходит следующим образом: сначала очищается все сообщения, которые вывелись, а потом снова вызывается метод для получения сообщений, так как в таймере задается маленький промежуток обновления, то пользователь не замечает очищения сообщений.

Для обновления сообщений в «ВКонтакте» используется метод LongPoll – это метод получает мгновенный ответ с сервера, то есть не надо будет каждый раз очищать старые сообщения и снова вызывать метод для их получения, сообщения будут отображаться по мере их поступления. Данный метод использовался в функции представленной на листинге 7.

def update\_message(self):

for event in self.longpoll.listen():

if event.type == VkEventType.MESSAGE\_NEW:

user = self.vk.users.get(user\_ids = event.user\_id)

user\_vk = user[0].get('first\_name') + " " + user[0].get('last\_name')

if event.from\_chat:

chats\_vk = self.vk.messages.getChat(chat\_id = event.chat\_id)

chats\_vk\_title = chats\_vk.get('title')

if self.tabBar.tabText(self.tab\_message.currentIndex())==chats\_vk\_title:

time = event.timestamp

mes = str(event.text)

data=datetime.datetime.fromtimestamp(int(time)).strftime(

'%Y-%m-%d %H:%M:%S')

user\_chat=self.vk.users.get(user\_ids=event.user\_id)

user\_vk\_chat=user[0].get('first\_name')+" "+ user[0].get('last\_name')

self.chats\_vk\_message.append("<big>"+"<b>"+user\_vk\_chat+"</b>" + " " +str(data) + "<br>" + mes + "<br><br>"+"</big>")

self.chats\_vk\_message.moveCursor(QtGui.QTextCursor.End)

if self.tabBar.tabText(self.tab\_message.currentIndex())==user\_vk:

if event.from\_me:

time = event.timestamp

mes = str(event.text)

data=datetime.datetime.fromtimestamp(int(time)).strftime

('%Y-%m-%d %H:%M:%S')

self.vk\_message.append("<big>"+ "<b>"+self.my\_self\_vk+

"</b>" + " "+str(data) + "<br>" + mes + "<br><br>"+"</big>")

self.vk\_message.moveCursor(QtGui.QTextCursor.End)

elif event.to\_me:

time = event.timestamp

mes = str(event.text)

data=datetime.datetime.fromtimestamp(int(time)).strftime

('%Y-%m-%d %H:%M:%S')

self.vk\_message.append("<big>"+"<b>"+user\_vk+"</b>"+" "+str(data)+"<br>"+mes+"<br><br>"+"</big>")

self.vk\_message.moveCursor(QtGui.QTextCursor.End)

Листинг 7 – Обновление сообщений в «ВКонтакте»

В приложение также есть возможность обмениваться сообщениями, но не с одним контактом из одного клиента, а сразу с несколькими и из разных клиентов, эта функция приложения реализована с помощью компонента QTabWidget. Функции для отправки сообщений в «Telegram», представлена на листинге 8:

def telegram\_send(self):

if (self.flag\_telegram\_message == True):

if (len(self.message\_text.toPlainText())):

for user\_val1 in range(0,self.users\_telegram.count(),1):

users = self.user[user\_val1]

log\_tl = str(users.first\_name) + " " + str(users.last\_name)

if(self.tabBar.tabText(self.tab\_message.currentIndex())==str(users.first\_name))or(self.tabBar.tabText(self.tab\_message.currentIndex())==log\_tl)or(self.tabBar.tabText(self.tab\_message.currentIndex())==str(users.username)):

self.client.send\_message(users, self.message\_text.toPlainText())

self.message\_text.clear()

self.timer\_message\_telegram.start(5000)

Листинг 8 – Отправка сообщений в «Telegram»

В данной функции тоже используют флаги для определения, какому контакту, каналу или в какую беседу хочет пользователь отправить сообщения, чтобы не произошло ошибки, и сообщение не отправилось в другой место, имя контакта, или название беседы берется с названия вкладки, если контакт с «Telegram», то полученного имя хватает. В случае с «ВКонтакте» нужно определить еще id выбранного контакта, если сообщение отправляется в беседу в «ВКонтакте», то к полученному id беседы прибавляется еще два миллиарда.

**ЗАКЛЮЧЕНИЕ**

В данном курсовом проекте мной были решены следующие задачи:

1. Анализ социальных сетей и выбор самых популярных для использования в приложении.
2. Анализ различных библиотек с исходным открытым кодом для дальнейшего использования.
3. Изучение высокоуровневого языка программирования Python.
4. Изучение среды разработки PyQt.
5. Ознакомление с механизм QSS, позволяющий настраивать внешний вид [виджетов](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%B4%D0%B6%D0%B5%D1%82" \o "Виджет) [библиотеки](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) [Qt](https://ru.wikipedia.org/wiki/Qt" \o "Qt).

В ходе написания данного курсового проекта было разработано приложение, которое позволить использовать средства коммуникации и обмениваться информацией между людьми.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Мэтиз Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения / Э. Мэтиз. – СПб.: Питер, 2017. – 496 с.

2 Доусон М. Программируем на Python / М.Доусон. – СПб.: Питер, 2014. – 416 с.

3 Лутц М. Изучаем Python, 4-е издание / М.Лутц. – СПб.: Символ-Плюс, 2011. – 1280 с.

4 Лутц М. Программирование на Python, том I, 4-е издание / М.Лутц. – СПб.: Символ-Плюс, 2011. – 992 с.

5 Лутц М. Программирование на Python, том II, 4-е издание / М.Лутц. – СПб.: Символ-Плюс, 2011. – 992 с.

6 Прохоренок Н.А. Python 3 и PyQt. Разработка приложений / Н. А. Прохоренок. – СПб.: БХВ-Петербург, 2012. – 704 с.

7 Вабищевич П. Н. Численные методы. Вычислительный практикум / Вабищевич. – СПб.: Питер, 2012 – 320 c.

8 Пилгрим М. Погружение в Python 3 / М. Пилгрим. – СПб.: БХВ-Петербург, 2010. – 603 с.

9 Чаплыгин А.Н. Учимся программировать вместе с питоном / А.Н. Чаплыгин. – СПб.: Символ-Плюс, 2010. – 503 с.

10 Шапошникова С. Основы программирования на Python / С. Шапошникова. – СПб.: Питер, 2015. – 426 с.