Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Современные платформы программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

Программное средство по учёту доходов и расходов с прогнозированием «Kwit»

БГУИР КП 1-40 01 01 120 ПЗ

Студент: гр. 451006 Снитовец М. В.

Руководитель: Дубко Н. А.

Минск 2017

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

Лапицкая Н.В.

(подпись)

2017 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту Снитовцу Михаилу Владимировичу

1. Тема работы Программное средство для учёта расходов и доходов с прогнозированием

2. Срок сдачи студентом законченной работы 12.06.2017

3. Исходные данные к работе Среда программирования Intellij IDEA

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение;

1. Анализ предметной области;

2. Используемые технологии и приёмы прогаммирования;

3. Разработка программного средства;

4. Руководство пользователя;

Заключение.

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема базы данных программного средства

6. Консультант по курсовой работе Дубко Н. А.

7. Дата выдачи задания 10.02.2017 г.

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

введение к 20.02.2017 – 10 % готовности работы;

раздел 1 к 15.03.2017 – 30 % готовности работы;

раздел 2 к 02.04.2017 – 60 % готовности работы;

раздел 3 к 26.04.2017 – 60 % готовности работы;

раздел 4 к 04.05.2017 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 12.05.2017- 100 % готовности работы.

Защита курсового проекта с 12 мая 2017.

РУКОВОДИТЕЛЬ Дубко Н. А.

(подпись)

Задание принял к исполнению М. В. Снитовец 10.02.2017 г.

(дата и подпись студента)

СОДЕРЖАНИЕ

Введение 4
1 Анализ предметной области 5
1.1 Обзор аналогов 5
1.2 Постановка задачи 8
2 Используемые технологии и приёмы программирования 9
2.1 Язык программирования Koltin 9
2.2 Шаблон проектирования Model-view-controller 10
2.3 База данных SQLite 11
3 Разработка программного средства 13
3.1 Общее описание архитектуры 13
3.2 Работа с сетью 14
3.3 Обработка запросов к API 16
3.4 Использование системы ресурсов Qt 17
3.5 Разработка графического интерфейса 18
3.6 Кэширование с использованием базы данных SQLite 20
3.7 Обработка ошибок 22
4 Руководство пользователя 24
Заключение 27
Список использованных источников 28

ВВЕДЕНИЕ

В наше время существует невероятное количество разных способов траты денег, некоторые из них являются необходимостью, например оплата коммунальных услуг или телефонной связи, а некоторые простым развлечением, начиная от большого количества различных съедобных вкусностей и заканчивая удивительными способами экстремального отдыха.

Появилась проблема контроля за расходами, учётом целей трат и их объемов. Для решения данной проблемы с течением времени использовались разные подходы, такие как простая запись раходов и доходов в тетрадь или ведение таблиц Excel. С развитием технологий разработки мобильных и веб-приложений стали появляться отдельные средства для полного контроля над личными финансами.

Широкое применение в области учёта финансов нашли аналитические технологии для прогнозирования и анализа данных, прогнозирование на основе эконометрических, регрессионных и нейросетевых алгоритмах.

Целью данного курсового проекта является разработка веб-приложения для простого и удобного ведения учёта личных доходов и расходов с внедрением возможностей оценки текущего баланса и прогнозированием будущих раходов. Внимание планируется сконцентрировать на простоту использования и прослеживания текущего состояния счетов.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе будет произведен краткий обзор существующих аналогов приложения; сформулированы требования к разрабатываемому программному средству.

1.1 Обзор аналогов

В результате анализа предметной области было выявлено большое количество приложений по личному учёту доходов и раходов. В данном подразделе будут приведены три ярких представителя.

Основными критериями сравнения являются:

– пользовательский интерфейс;
– удобство и быстрота использования;
– возможность категоризации тразакций;
– работа со счетами;
– возможность просмотра статистики;
– отображение текущего состояния счетов.

Классическим приложением по учёту доходов и расходов является система «Семейный бюджет», расположенная по адресу https://koshelek.org. Данная система предоставляет широкий функционал по контролю личного бюджета. Предоставляет большие возомжности по генерации отчётов, планированию будущих расходов. Однако в следствии широких возможностей системы пострадал пользовательский интерфес. Внешний вид сайта устарел (рисунок 1.1), что также сказывается и на удобстве использования. Также сайт перегружен множеством вложенных меню и мелких, неочевидных иконок, что затрудняет работу с ним неподготовленному пользователю.

Кроме этого существует система под названием «Drebedengi» расположенная по адресу http://drebedengi.org (рисунок 1.2). Данная система позволяет вести учёт доходов и расходов, перемещений между счетами, планировать бюджет и и контролировать текущее состояние счетов. Также приложение позволяет контролировать долги. Пользовательский интерфейс более приятный, по сравнению с «Семейным бюджетом». Однако всё равно присутствует сложность восприятия из-за большого количества чисел.

Третий аналог, отличающийся от описанных выше отсутсвием перегруженного интерфейса — система «Zenmoney» (http://zenmoney.ru). Данное приложение позволяет работать с личными счетами, категориями, транзакциями. Предоставляет возможности генерировать отчёты, планировать бюджет, просматривать прогноз баланса. Главным недостатком данного

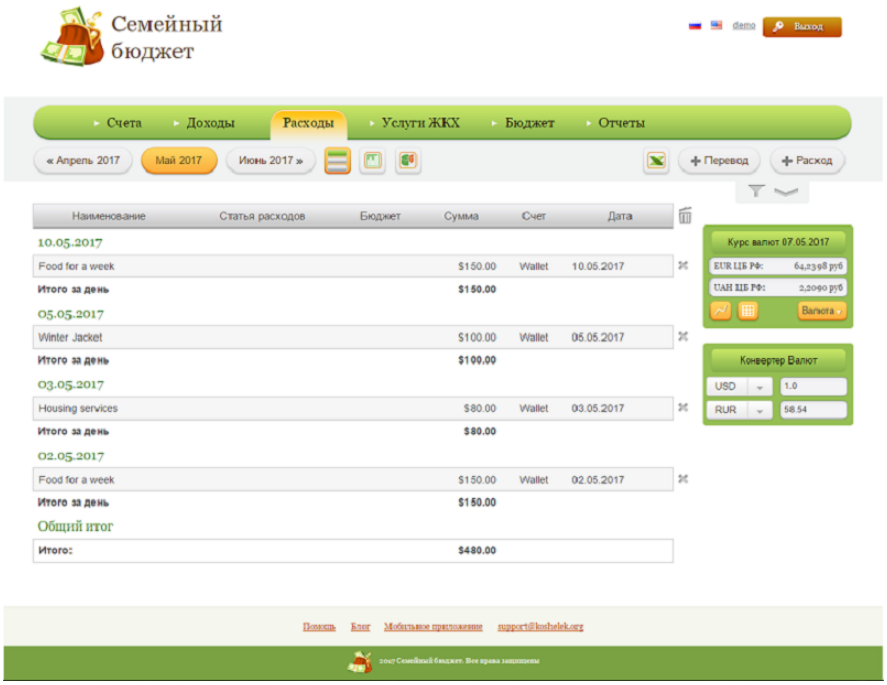

Рисунок 1.1 – Раздел «Доходы» сайта системы «Семейный бюджет»

приложения является внешний вид приложение — большинство элементов интерфейса предоставлены без какой-либо стилизации (рисунок 1.3).

Результат сравнения имеющихся аналогов с разрабатываемым приложением приведён в таблице 1.1.

Таблица 1.1 – Сравнение приведеных аналогов с разрабатываемым приложением

Функция	Kwit	Koshelek.org	Drebedengi.ru	Zenmoney.ru
Современный интерфейс	+	-	+	-
Простота в использовании	+	-	-	+
Мультивалютность	+	+	+	+
Простота в использовании	+	-	-	+
Работа со счетами	+	+/-	+/-	+
Работа с категориями	+	+/-	+/-	+
Возможность генерировать отчёты	-	+	+	+
Прогнозирование	+	+	+	+

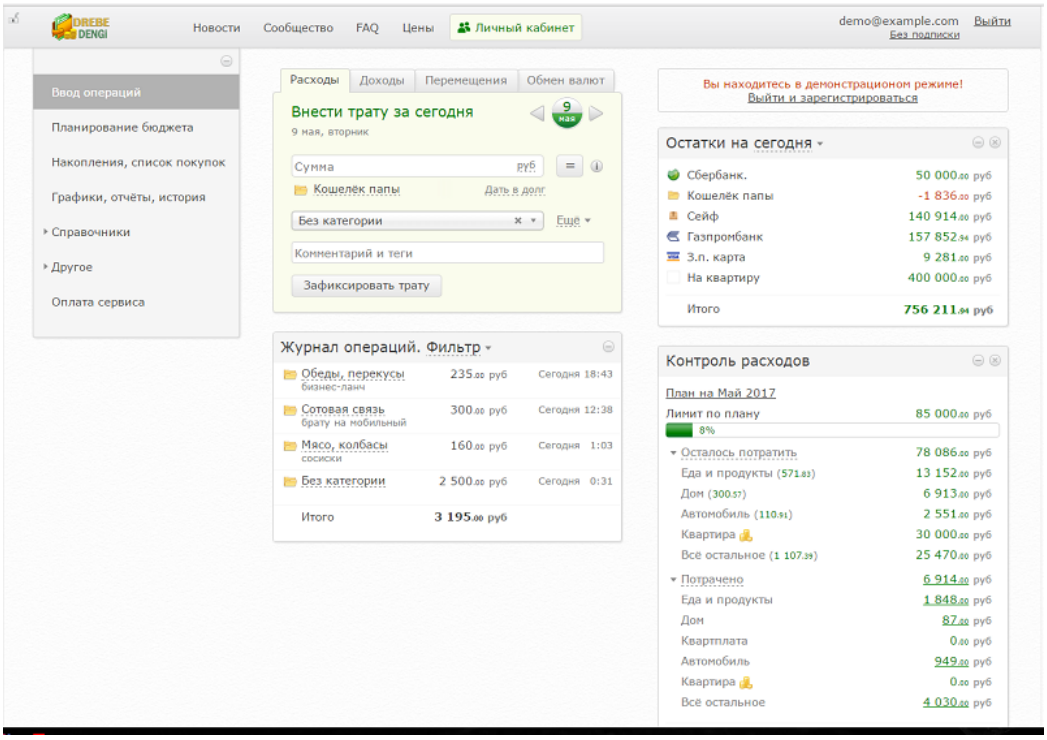

Рисунок 1.2 – Главная страница пользователя сайта системы «Drebedengi»

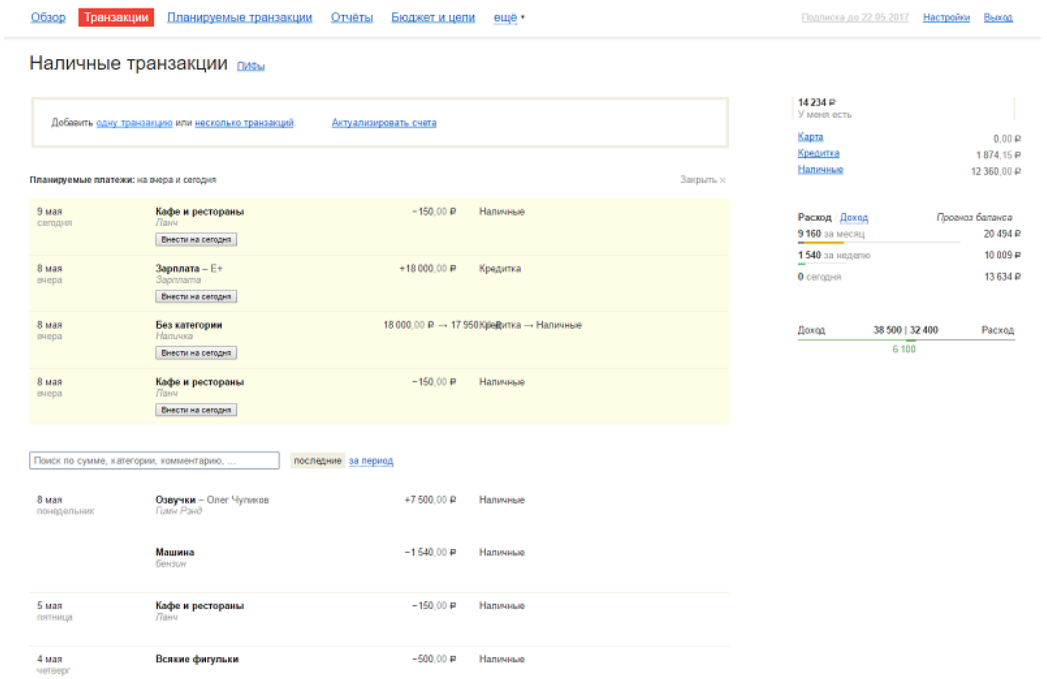

Рисунок 1.3 – Страница транзакций сайта системы «Zenmoney»

При составлении таблицы учитывался весь запланированный функционал, реализация некоторых функций возможна в версиях, которые будут разработаны вне данного курсового проекта.

1.2 Постановка задачи

Целью данного курсового проекта является разработка:

– добавление, удаление и изменение транзакций;

– добавление, удаление и изменение категорий;

– добавление, удаление и изменение счетов;

– возможность удаления категорий и счетов с переносом всех транзакций на другой счёт;

– возможжость подсчёта статистики по категориям за произвольный период всемени;

– подсчёт ежедневной суммы до зарплаты, прогноза будущих затрат за последнее время.

Программное средство должно представлять собой сервер с REST API и веб-клиент. Данное сочетание позволит в дальнейшем развить данное приложение в полноценную кроссплатформенную систему.

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ И ПРИЁМЫ ПРОГРАММИРОВАНИЯ

В данном разделе будет приведено описание языков программирования, фреймворков, используемых при разработке, и базы данных, применяемой в программе, а также изложен подход, применяемый к проектированию архитектуры приложения.

2.1 Язык программирования Koltin

Koltin (Котлин) — статически типизированный язык программирования, работающий поверх JVM и разрабатываемый компанией JetBrains.

Основные приемущества языка Koltin:

– краткость — кострукции и возможности языка разрабатывались с целью уменьшить количество кода, но при этом не уменьшая читаемости этого кода;

– поддержка защиты от NullPointerException на уровне языка;

– полная совместимость с языком программирования Java, что позволяет использовать весь набор библиотек и технологий, накопленных за долгое время существования Java;

– возможность расширять библиотеки, не изменяя их код, что позволяет дописывать необходимую функциональность без нарушений каких-либо лицензий либо поиска исходников уже скомпилированных библиотек (листинг 2.1);

– мультипарадигмость — Koltin можно писать код как в процедурном стиле, так и в объектно-ориентированном, а также благодаря поддержке функций высшего порядка можно писать код и в функциональном стиле. Сочетание лучших качеств у каждого из этих стилей позволяет разрабатывать приложения быстро и эффективно (листинг 2.2);

Листинг 2.1 – Пример Extension-функции на языке Koltin

```
public fun CharSequence.padEnd(length: Int, padChar: Char = ' '): CharSequence
    {
   if (length < 0)
      throw IllegalArgumentException("Desired length $length is less than
         zero.")
   if (length <= this.length)
      return this.subSequence(0, this.length)

   val sb = StringBuilder(length)
   sb.append(this)
   for (i in 1..(length - this.length))
```

```
        sb.append(padChar)
    return sb
}
```

Листинг 2.2 – Реализация популярной функции высшего порядка «map» на языке Koltin

```
fun <T, R> List<T>.map(transform: (T) -> R): List<R> {
    val result = arrayListOf<R>()
    for (item in this)
        result.add(transform(item))
    return result
}
```

2.2 Шаблон проектирования Model-view-controller

Достаточно часто в процессе разработки требуется вносить изменения в логику работы программы. Любые видоизменения в логике влекут за собой изменения в исходном коде. Как известно, при модификации исходного кода есть риск допустить ошибку. Очевидно, что программирование без изменения исходного кода невозможно, однако при проектировании архитектуры программы следует делать это так, чтобы переписывать как можно меньше кода при внесении изменений в логику работы. Иными словами, архитектура должна быть гибкой и расширяемой.

Паттерн проектирования — это идея решения проблемы проектирования в определенном, характерном контексте. Таким образом, само решение проблемы, то есть реализация какого-либо паттерна на языке программирования, все еще является задачей для разработчика, однако шаблон определяет некоторые аспекты отношения и взаимодействия между классами и объектами, снижая сложность разработки.

Шаблон проектирования Model-view-controller — это популярный подход к построению архитектуры приложения, когда модель приложения, интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные (рисунок 2.1).

Основная цель применения этой концепции состоит в отделении модели от её представления. За счет такого разделения повышается возможность повторного использования кода. Наиболее полезно применение данной концепции в тех случаях, когда пользователь должен видеть те же самые данные одновременно в различных контекстах. Например, некоторые данные могут быть одновременно представлены в виде электронной таблицы,

гистограммы и круговой диаграммы. Данный подход также упрощает совместную разработку, так как позволяет каждой команде разработчиков концентрироваться только на своей задаче. Поэтому возможно добиться того, что программисты, занимающиеся разработкой бизнес-логики («модели» в терминах MVC), вообще не будут осведомлены о том, какое представление будет использоваться.

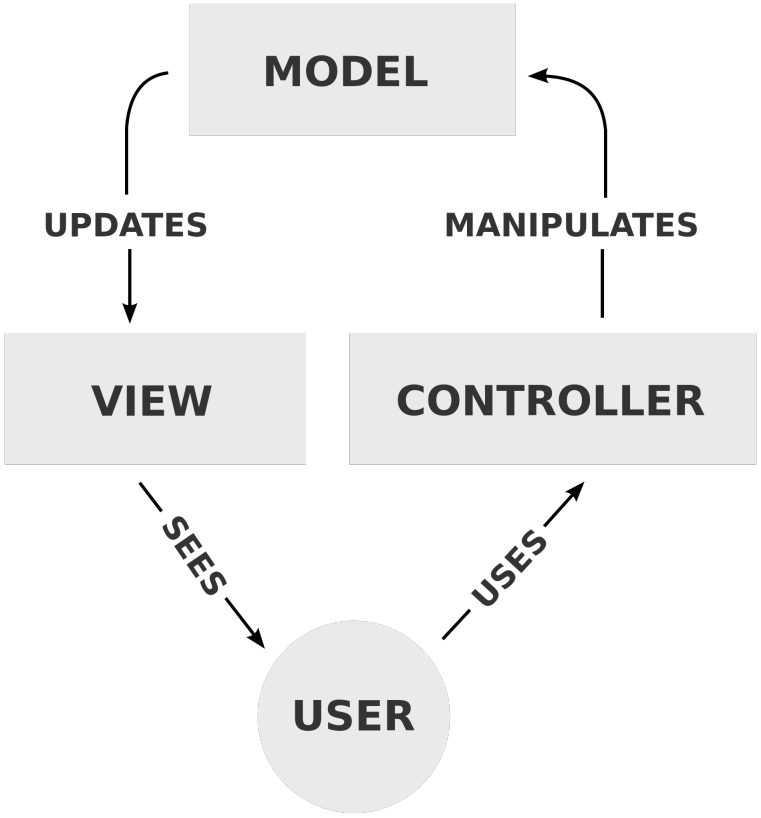

Рисунок 2.1 – Схема взаимодействия компонентов MVC

При правильном применении Model-view-controller существенно облегчает поддержку программы и снижает затраты на изменение или добавление функциональности.

2.3 База данных SQLite

Процесс загрузки большого количества данных через запросы с API может занять достаточно продолжительное время. Даже несмотря на возможность одновременной загрузки данных и вывода уже загруженных, необходимо минимизировать расход трафика и нагрузку на сеть в процессе работы с приложением путем кэширования уже загруженных данных и периодического обновления кэша по мере появления новой информации на

сервере.

Для этих целей отлично подойдет встраиваемая реляционная база данных SQLite. Данная база данных не использует клиент-серверную архитектуру, то есть не имеет отдельно работающего процесса, взаимодействующего с клиентской программой и базой данных, а предоставляет библиотеку, с которой программа компонуется. Таким образом движок становится составной частью программы. SQLite хранит всю базу данных в единственном файле, что уменьшает накладные расходы, время отклика и упрощает программу.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки.

SQLite отличается простотой использования и встраивания, в связи с чем активно используется во многих популярных приложениях.

3 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

В данном разделе будет описан процесс проектирования и программирования основных модулей программного средства

3.1 Общее описание архитектуры

С целью обеспечения гибкости архитектуры и модульности, принято решение разбить функционал программного средства на следующие модули:

– набор модулей для работы с сетью на уровне запросов;

– модуль для работы с API «ВКонтакте».

– модуль для работы с LongPoll сервером;

– модуль для хранения кэша загруженных данных и обеспечения связи с графическим интерфейсом;

– база данных для сохранения кэша на диск;

– модуль для взаимодействия с базой данных;

– набор классов для представления различных типов данных, используемых API социальной сети;

– набор модулей для реализации графического интерфейса.

Учитывая специфику асинхронной работы с сетью, важным моментом является логика обновления данных в представлении. Обновление данных может происходить в разном порядке по мере загрузки информации из сети. Кроме события получения новых сообщений в фоновом режиме возможно обновление фотографии пользователя, онлайн-статуса собеседника, статуса прочитанности сообщений и многой другой информации. Всё это должно не только своевременно отражаться в графическом интерфейсе, но и выборочно сохраняться в локальный кэш, который необходим для сокращения времени синхронизации с данными учётной записи.

Для решения обозначенной проблемы предназначен модуль хранения кэша загруженных данных, являющийся связующим звеном между модулями для работы с сетью, графическим интерфейсом и базой данных для сохранения кэша в постоянной памяти.

Взаимодействие хранилища и графического интерфейса осуществляется с применением паттерна «Наблюдатель». Хранилище обрабатывает постоянно загружаемые из сети данные и в случае необходимости уведомляет графический интерфейс об обновлении, что в свою очередь вызывает перерисовку нужных элементов.

С целью ослабления зависимостей между графическим интерфейсом и подсистемой работы с сетью создан модуль для работы с API «ВКонтак-

те», инакпсулирующий всю необходимую работу с сетью через соответствующие модули.

Такая декомпозиция помогает упростить процесс разработки и обеспечивает возможность независимой работы над различными функциональными компонентами программного средства.

3.2 Работа с сетью

Qt предлагает набор классов для удобной асинхронной работы с сетью с использованием сигнально-слотовой системы. Для загрузки определенной веб-страницы необходимо организовать взаимодействие объектов трёх классов: QNetworkAccessManager, QNetworkRequest и QNetworkReply.

Класс QNetworkAccessManager позволяет выполнять сетевые запросы, представляемые объектами класса QNetworkRequest и получать ответы на них в виде загруженных данных. Объект данного класса хранит сетевые настройки для запросов, которые будут отправляться с его помощью. Для обработки загруженных данных в классе имеется сигнал replyFinished, передающий объект класса QNetworkReply, содержащий все загруженные данные. Пример кода, иллюстрирующий данное описание, представлен в листинге 3.1.

Листинг 3.1 – Загрузка страницы с использованием Qt

```
QNetworkAccessManager *manager = new QNetworkAccessManager(this);
connect(manager, SIGNAL(finished(QNetworkReply*)),
        this, SLOT(replyFinished(QNetworkReply*)));
manager->get(QNetworkRequest(QUrl("http://vk.com")));
```

Работа с сетью в программе осуществляется следующим набором классов:

– VNetworkManager — основной класс для работы с сетью, объединяющий при помощи механизма композиции более узкоспециализированные классы;

– VNetworkLongPoll — класс для работы с LongPoll сервером;

– VNetworkSimple — класс, предназначенный для выполнения обычных GET запросов;

– VNetworkReply — вспомогательный класс, используемый в классе VNetworkSimple для обработки загруженных данных.

Разделение функционала данных классов и классов, выполняющих обработку загруженных данных, достигается при помощи системы сигналов и слотов Qt. Схема соединения сигналов и слотов, обеспечивающих обновление интерфейса, приведена на рисунке 3.1.

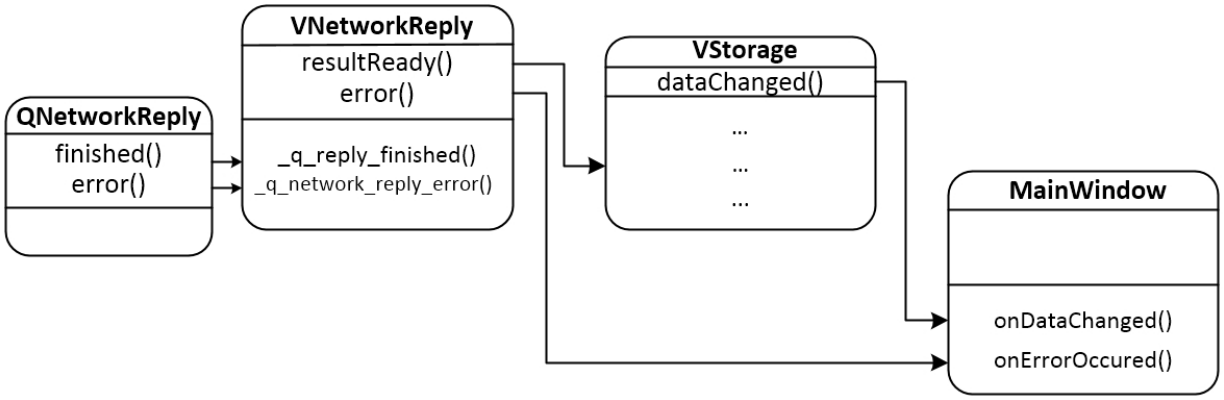

Рисунок 3.1 – Схема соединения сигналов и слотов, обеспечивающих обновление интерфейса

Из приведённой схемы видно, что библиотечный класс QNetworkReply связан с VNetworkReply. Такая агрегация позволяет выполнить предварительную низкоуровневую обработку ответа перед тем, как передавать его в класс, обеспечивающий кэширование. На данном этапе происходит создание JSON-объекта, используемого другими модулями программы, и отслеживание некоторых типов ошибок. Многоточия на схеме обозначают набор слотов, с одним или несколькими из которых будет соединён сигнал resultReady класса VNetworkReply. Выбор данного слота осуществляется в зависимости от типа загружаемых данных. В то же время выработка сигнала dataChanged от типа обновлённой информации зависит лишь вспомогательными для интерфейса данными, которые передаются в качестве параметров функции.

Доступа к методам классов VNetworkManager и доступных из него VNetworkSimple и VNetworkLongPoll организован с использованием паттерна «Одиночка». Данный паттерн обеспечивает наличие единственного экземпляра класса с глобальной точкой доступа в однопоточном приложении. Класс VNetworkManager гарантирует это благодаря классическому для C++ приёму, называемому синглтон Майерса: в публичном методе getInstance() объявляется статическая переменная, имеющая тип данного класса, а конструктор класса переносится в секцию private. Дополнительно к этому запрещается использование конструктора копий и оператора присваивания, для этого применяется ключевое слово delete, введенное в стандарте C++11. Похожим образом в классе VNetworkManager организован доступ к классу для выполнения простых GET запросов и классу для работы с LongPoll сервером. В качестве примера в листинге 3.2 приведен код для получения экземпляра класса VNetworkSimple.

Листинг 3.2 – Получение экземпляра класса VNetworkSimple

```
VNetworkSimple *VNetworkManager::simple()
{
      static VNetworkSimple networkSimpleInstance;
      return &networkSimpleInstance;
}
```

Класс хранения static гарантирует, что переменная, объявленная таким образом, будет создана и инициализирована только один раз перед первым использованием. Применительно к объекту это означает, что конструктор соответствующего класса будет вызван только один раз.

3.3 Обработка запросов к API

Для получения информации о состоянии учетной записи и поддержки данной информации в актуальном состоянии требуется совершать большое число запросов к API социальной сети, причем данные запросы могут находится в тесной связи друг с другом.

Необходимость отправки нового запроса может возникнуть не только по команде от пользовательского интерфейса, но и в процессе обработки другого запроса. Также их результаты могут по-разному обрабатываться в зависимости от текущего состояния программы и статуса завершенности зависимых запросов.

Все перечисленные факторы хорошо учитываются при организации запросов в виде дерева. Каждый запрос представлен узлом дерева. Дочерние узлы дерева представляют, соответственно, зависимые запросы. Сигнал о завершении запроса не инициируется до тех пор, пока не завершатся все его дочерние запросы, даже в том случае, когда сам запрос уже завершился. Реализация данной модели значительно упрощается благодаря сигнально-слотовой системе фреймворка Qt.

По завершении запроса возможно три варианта развития событий: отсутствие действий, уведомление представления, изменение модели. В соответствии с данными вариантами узлы дерева, представляющие запросы, имеют три типа. Зависимости от типа запроса, узел дерева может хранить дополнительную информацию, необходимую для корректной реакции на его завершение. Например, возможно уведомление представления о следующих событиях: отправка сообщения, получение нового сообщения, удаление сообщения, загрузка аватара пользователя, получение информации о необходимости заполнения капчи, загрузка изображения капчи и т. д.

3.4 Использование системы ресурсов Qt

Система ресурсов Qt – платформо-независимый механизм для внедрения двоичных файлов в исполняемый файл приложения — систему ресурсов. Это полезно, если для разрабатываемого программного средства всегда требуется определённый набор файлов, например, пиктограмм, файлов перевода и так далее, при этом риск потери файлов необходимо свести к минимуму. Ресурсы, связанные с приложением, указываются в файле коллекции ресурсов, имеющем расширение qrc. Формат файла основан на XML, в котором перечисляются файлы на диске и опционально присваивает им имя ресурса, используемое приложением для доступа к ресурсу. Листинг 3.3 показывает пример файла коллекции ресурсов:

Листинг 3.3 – Пример файла qrc

```
<!DOCTYPE RCC><RCC version="1.0">
      <qresource>
          <file>images/copy.png</file> <file>images/cut.png</file>
          <file>images/delete.png</file>
      </qresource>
 </RCC>
```

Файлы ресурсов, перечисленные в коллекции ресурсов, являются частью дерева исходников приложения. Указываемые пути являются относительными к каталогу, содержащему файл qrc, при этом перечисленные файлы ресурсов должны располагаться в том же каталоге, что и файл коллекции, или в одном из подкаталогов. Данные ресурса могут либо быть скомпилированы в двоичном виде и таким образом к ним можно получить доступ непосредственно в коде приложения, либо двоичный ресурс может быть создан и станет возможно позже ссылаться на него в коде приложения зарегистрированный с помощью системы ресурсов.

В листинге 3.4 демонстрируется метод, используемый для загрузки стиля QSS из ресурса.

Листинг 3.4 – Загрузка стиля для виджета из файла

```
void VHelper::loadStyleSheetFromFile(QString fileName, QWidget *target)
{
   QFile styleSheetFile(fileName);
   if (styleSheetFile.open(QIODevice::ReadOnly) )
   {
      QString qss = QLatin1String(styleSheetFile.readAll() );
      target->setStyleSheet(qss);
      styleSheetFile.close();
   }
}
```

3.5 Разработка графического интерфейса

При разработке графического интерфейса важно учитывать стилистические особенности оформления сайта и официальных клиентов «ВКонтакте». Все они выдержаны в одном стиле и в одной цветовой гамме, которые следует использовать и в Vk App for Desktop.

Для оформления графического интерфейса использовалась технология QSS, описанная в подразделе **??**. В качестве примера в листинге 3.5 приведён стиль для оформления кнопки входа.

Листинг 3.5 – QSS-стиль для оформления кнопки входа

```
QPushButton#buttonLogin {
	font-weight: 700;
	color: #fff;
	background-color: #6182aa;
	border-radius: 4px;
	padding-left: 3px;
	padding-right: 3px;
	padding-top: 8px;
	padding-bottom: 8px;
}
```

Результат применения данного стиля приведён на рисунке 3.2. Такое оформление в точности соответствует стилистике социальной сети.

Рисунок 3.2 – Кнопка входа, оформленная с использованием CSS

Отображение сообщений и списка диалогов является более сложной задачей из-за отсутствия готового компонента в библиотеке виджетов Qt. Для решения проблемы необходимо создание своего виджета, отображающего всю необходимую информацию в подходящей форме.

За отображение сообщений в диалогах отвечает класс VMessageView, использующий QCustomWidget и QCustomLabel. Вспомогательные классы предназначены для переопределения некоторых виртуальных методов, необходимых для обеспечения визуального отклика на действия пользователя. Данные методы принимают в качестве параметров объекты служебных классов QEvent, QMouseEvent и QPaintEvent. Для реализации необходимых эффектов потребовалось переопределить следующие виртуальные методы родительского класса QWidget:

– mousePressEvent — метод, обрабатывающий события клика мышью по виджету;

– enterEvent и leaveEvent — методы, в которых производится обработка наведения курсора мыши на виджет;

– paintEvent — метод, вызывающийся при необходимости перерисовки виджета.

Изменение цвета сообщения по наведению мыши достигается за счёт переопределения enterEvent и leaveEvent, а для выделения по клику используется пользовательский обработчик mousePressEvent.

Внешний вид получившегося виджета в разных состояниях приведён на рисунке 3.3.

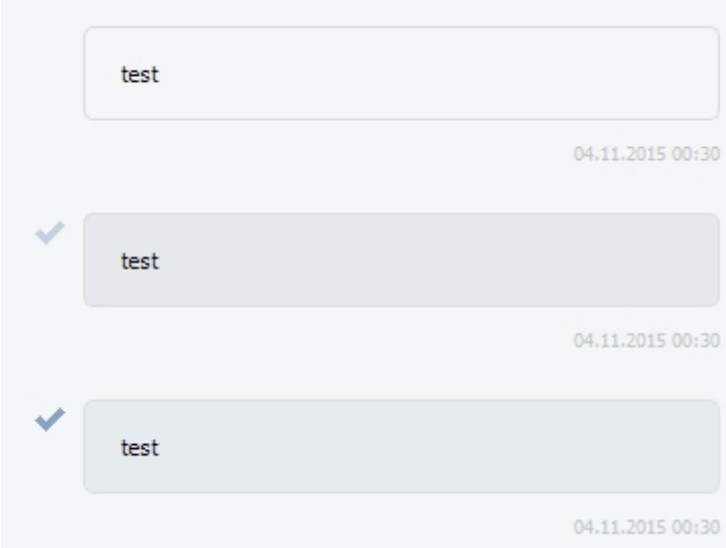

Рисунок 3.3 – Виджет для отображения сообщения в разных состояниях

Полученные виджеты удобно добавлять в любой контейнер наподобие QGridLayout для отображения истории сообщений.

Аналогичный подход используется и для отображения списка недавних диалогов. Каждый диалог представляется пунктом меню, содержащим имя собеседника и его изображение профиля. Класс VDialogView реализует необходимый виджет, используя вспомогательный класс VAvatarWidget, используемый для отображения изображения пользователя в характерном для «ВКонтакте» стиле. Готовый виджет, отображающий тестовые данные, приведён на рисунке 3.4.

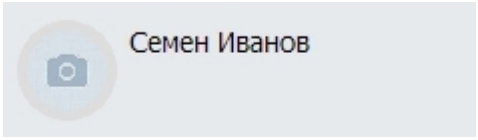

Рисунок 3.4 – Виджет для отображения информации о диалоге

Для отображения списка диалогов, как и в случае с историей сообщений, удобно воспользоваться компонентом QGridLayout.

Для хранения необходимых для графического интерфейса файлов изображений и QSS стилей используется система ресурсов Qt, описанная в подразделе **??**. Такой подход позволяет сэкономить время на сборку программы и упростить оформление графического интерфейса.

Стоит отметить, что тонкая настройка внешнего вида с помощью CSS доступна и для созданных описанным способом пользовательских виджетов, так как они являются объединением уже имеющихся стандартных органов управления.

Такой способ отображения сообщений и списка диалогов отличается гибкостью и расширяемостью. При добавлении новых типов отображаемых в сообщениях вложений можно будет сконцентрироваться на реализации нового функционала без необходимости внесения существенных изменений в существующий код.

3.6 Кэширование с использованием базы данных SQLite

В подразделе 2.3 описываются достоинства базы данных SQLite, позволяющие использовать её в качестве хранилища кэша.

Qt содержит все необходимые классы для работы с различными базами данных. Основным таким классом является QSqlDatabase, служащий для представления связи с базой данных. В связи с тем, что низкоуровневые механизмы работы с различными базами данных могут существенно отличаться, QSqlDatabase исользует класс QSqlDriver. Данный класс уже зависит от используемой базы данных, однако вручную адаптировать его придётся только в случае использования экзотических баз данных, драйверы для которых ещё не реализованы. Стандартной библиотекой Qt поддерживаются базы данных следующих типов:

- IBM DB2;
- Borland InterBase Driver;
- MySQL;
- Oracle Call Interface Driver;
- ODBC;
- PostgreSQL;
- SQLite;
- Sybase Adaptive Server.

Для работы с базой данных в программе используется пользователь-

ский класс VStorageDatabase, инкапсулирующий всю работу с базой для загрузки или сохранения кэша.

Для подключения к существующему файлу базы данных используется статический метод addDatabase класса QSqlDatabase, одна из версий которого принимает в качестве параметра строку, указывающую тип базы данных, для выбора соответствующего драйвера. Реализация метода openDB класса VStorageDatabase для подключения к базе данных приведена в листинге 3.6.

Листинг 3.6 – Реализация метода openDB

```
bool VStorageDatabase::openDB()
{
	db = QSqlDatabase::addDatabase("QSQLITE");
	db.setDatabaseName(DATABASE_FILE_PATH);
	if (!db.open())
		return false;
	return this->createDB();
}
```

Константа DATABASE_FILE_PATH содержит путь к файлу с базой данных. В случае ошибки при открытии существующей базы происходит вызов метода createDB, который с нуля задает структуру базы и выполняет все запросы для создания необходимых таблиц. Метод возвращает переменную логического типа, отражающую успех выполненной операции.

Для выполнения SQL-запросов и работы с полученными данными применяются встроенные классы QSqlQuery и QSqlRecord.

Для выполнения запроса необходимо создать новый объект класса QSqlQuery и задать необходимую команду с помощью методов prepare и bindValue. При выполнении запросов возможно использовать специальный синтаксис для указания мест, которые в дальнейшем будет заполнены с помощью методов bindValue. Такой способ позволяет уменьшить громоздкость строки построения запроса и обеспечивает большую гибкость. В листинге 3.7 приведён фрагмент реализации метода insertUser, выполняющего добавление информации о пользователе в базу данных.

Метод принимает единственным параметром объект VUser, служащий для внутреннего представления типов данных API «ВКонтакте». Далее формируется строка SQL-запроса, содержащая специально оформленные метки, куда впоследствии будут помещаться аргументы. После этого метод bindValue вызывается необходимое количество раз для помещения в предварительно помеченные места фактических параметров запроса.

Листинг 3.7 – Фрагмент реализации метода insertUser

```
...
QSqlQuery query;

query.prepare( "INSERT INTO `users` (`user_id`, `first_name`,
              `last_name`, `sex`, `bdate`, `photo_50`, `photo_100`)
              VALUES (:user_id, :first_name, :last_name, :sex,
              :bdate, :photo_50, :photo_100);");
query.bindValue(":user_id", QVariant((qulonglong)user->userId));
query.bindValue(":first_name", QVariant(user->firstName));
query.bindValue(":last_name", QVariant(user->lastName));
...
return query.exec();
```

Для выбора данных из таблицы используется тот же класс QSqlQuery, у которого для получения результатов запроса имеется метод record, возвращающий объект класса QSqlRecord. Объект класса QSqlRecord поддерживает итерацию по списку возвращенных объектов через использование метода next. Для получения значений необходимого поля используются методы indexOf и value классов QSqlRecord и QSqlQuery, соответственно. Метод indexOf возвращает целочисленное значение, идентифицирующее поле по имени поля, а value в свою очередь получает в формате QVariant значение данного поля. Пример выбора значений из результатов SQL-запроса в методе loadUsers приведён в листинге 3.8.

Листинг 3.8 – Получение информации о пользователе из результатов SQL-запроса

```
user->userId = query.value(rec.indexOf("user_id")).toULongLong();
user->firstName = query.value(rec.indexOf("first_name")).toString();
user->lastName = query.value(rec.indexOf("last_name")).toString();
```

В приведённом коде происходит обращение к нужному полю результата запроса и преобразование типа QVariant к типу соответствующему данных.

3.7 Обработка ошибок

Ошибки, возникающие в процессе работы с API «ВКонтакте»можно условно разделить на два типа: ошибки авторизации и ошибки выполнения запросов к API, связанные с необходимостью ввода капчи.

Проблемы авторизации (в том числе необходимость двухфакторной авторизации) обнаруживаются в методе onAuthError класса VApiWrapper. Данный метод вызывается при завершении выполнения запроса авторизации. В зависимости от типа проблемы метод onAuthError инициирует сиг-

нал об изменении данных авторизации, который обрабатывается соответствующей формой. В форме реакция на запрос авторизации выполняется в методе onAuthDataChanged, который управляет выводом информации об ошибках или информационных сообщениях о ходе двухфакторной авторизации.

Ошибки выполнения запросов к API, связанные с необходимостью ввода капчи. также обрабатываются в классе VApiWrapper, но уже в методе onError. Данный метод является обработчиком сигнала error, который может инициироваться при разборе JSON-ответа в классе VNetworkReply. При обнаружении такой ошибки демонстрируется форма для ввода капчи, которую требует сервер. После ввода происходит повторная отправка запроса, при обработке которого возникла данная ошибка, к которому присоединяется введенный пользователем код с предложенной картинки. Повторная ошибка такого же типа может возникнуть в случае неверно введенного кода. В этом случае произойдет повторная демонстрация формы ввода кода. Таким образом сигнал, соответствующий успешному выполнению запроса, не будет инициирован, пока введенный код не окажется корректным.

Такой подход позволяет обрабатывать ошибки, возможные при работе с API «ВКонтакте», без существенного влияния на основную логику обработки запросов.

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Разработанное программное средство обладает интуитивно понятным интерфейсом и привычными органами управления, характерными для большинства программ мгновенного обмена сообщениями. При этом цветовая гамма и стиль элементов управления выдержан в соответствии с оформлением социальной сети «ВКонтакте».

При запуске программы осуществляется проверка наличия сохраненных данных об авторизации. При отсутствии таких данных будет продемонстрировано окно для проведения авторизации, изображенное на рисунке 4.1.

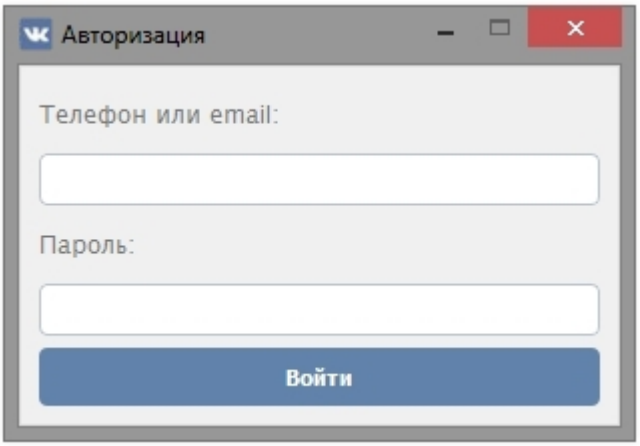

Рисунок 4.1 – Окно авторизации пользователя

В случае ошибки авторизации отобразится соответствующая информация. Пример сообщения об ошибке предствлен на рисунке 4.2.

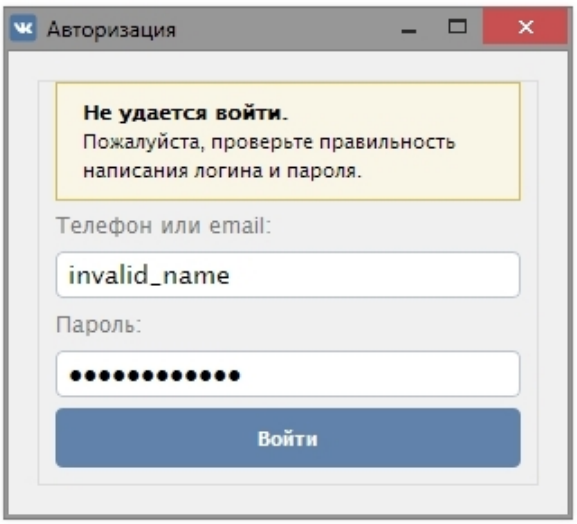

Рисунок 4.2 – Сообщение об ошибке авторизации

После успешной авторизации произойдет открытие главного окна про-

граммы (рисунок 4.3) и автоматическая синхронизация истории сообщений с данными учётной записи «ВКонтакте».

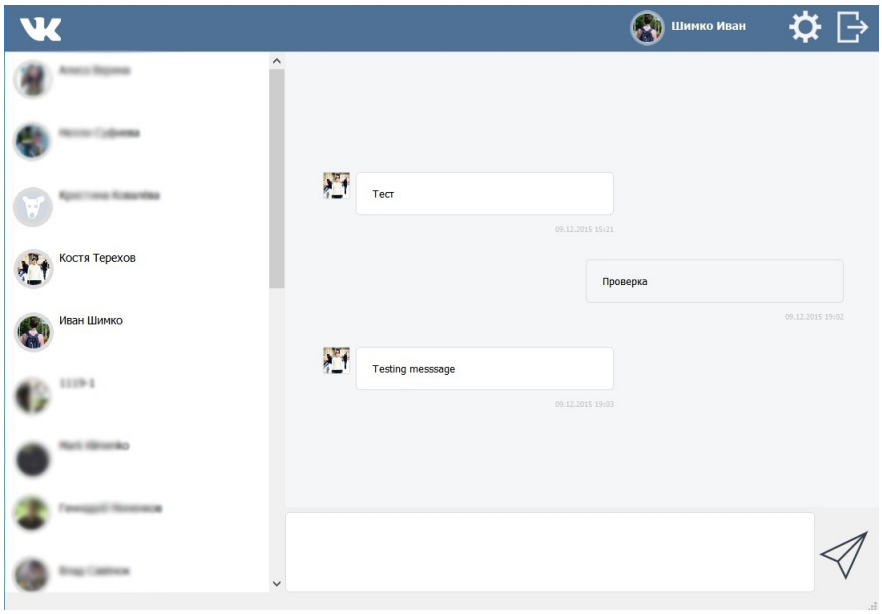

Рисунок 4.3 – Главное окно Vk App for Desktop

В верхней панели программы находится информация об активной учётной записи, кнопка для вызова меню настроек и кнопка выхода из приложения.

В левой части окна находится список активных диалогов пользователя, отсортированный в порядке даты последнего сообщения. Выбор диалога осуществляется простым кликом по соответствующему пункту списка.

После выбора нового диалога произойдёт запуск синхронизации данных кэша с данными учётной записи.

В случае подозрительной активности со стороны учетной записи, серверы «ВКонтакте»могут потребовать ввода кода с картинки. Пример окна для ввода данного кода приведено на рисунке 4.4.

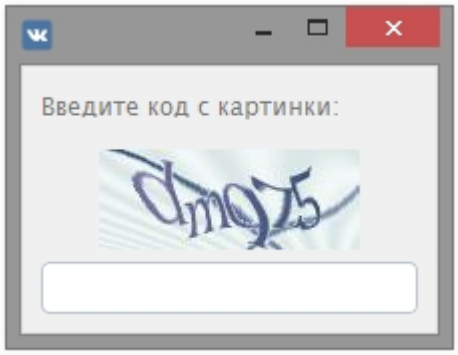

Рисунок 4.4 – Окно ввода капчи

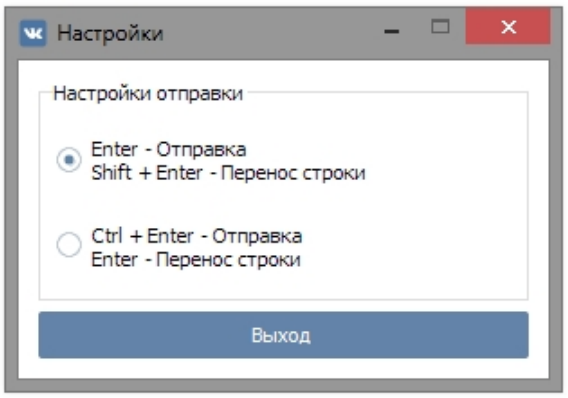

Рисунок 4.5 – Меню настроек

Для отправки сообщения пользователю достаточно ввести сообщение в поле, находящееся под историей сообщений и воспользоваться кнопкой с соответствующей пиктограммой или сочетанием клавиш, выбранным в меню настроек.

Догрузка старых сообщений происходит автоматически при прокрутке истории диалога. Внешний вид меню настроек программы представлен на рисунке 4.5.

Из данного меню возможно переключить настройки сочетаний клавиш для отправки сообщения и переноса строки, а также осуществить выход из текущего аккаунта.

Выход из аккаунта приведет к очистке локального кэша и демонстрации окна для повторного прохождения авторизации.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы было создано пррограммное средство Vk App for Desktop, предназначенное для мгновенного обмена сообщениями через социальную сеть «ВКонтакте». Исходный код программного средства без изменений может быть скомпилирован под такие операционные системы, как Windows, Mac OS и большинство дистрибутивов Linux.

Модули для кэширования, работой с базой данных и взаимодействия с API «ВКонтакте» были полностью разработаны Константином Тереховым. Классы, предназначенные для осуществления сетевых запросов, внутрипрограммного представления объектов социальной сети, виджеты для отображения сообщений и диалогов, логика графического интерфейса — Иваном Шимко и Владиславом Савёнком. Проектирование общей архитектуры приложения и периодическая инспекция кода осуществлялись совместно.

В процессе разработке были приобретены навыки по работе с сетью, расширены знания фреймворка Qt, языка программирования C++, паттернов объектно-ориентированного проектирования и концепции Model-view-controller. Разработка модуля для сохранения кэша потребовала дополнительного изучения реляционных баз данных и языка SQL. При разработке модуля для взаимодействия с «ВКонтакте» был подробно изучен формат обмена данными JSON, используемый API социальной сети. Коллективная разработка данного программного средства позволила приобрести опыт работы с системой контроля версий git.

В связи со сложностью и объёмностью данного проекта, на что косвенно указывает практически полное отсутствие неофициальных клиентов для данной социальной сети для десктопных ОС, не весь изначально запланированный функционал был реализован в полной мере.

Программное средство имеет большой потенциал для развития и совершенствования, однако текущая версия содержит необходимый каркас, который обеспечивает достаточную гибкость для расширения поддержки функций социальной сети «ВКонтакте».

В будущих версиях планируется расширить перечень отображаемых вложений в личные сообщения, усовершенствовать поддержку групповых диалогов, добавить настройки пометки сообщений прочитанными и обеспечить возможность настройки уведомлений о новых сообщениях. Также возможно добавление встроенного плеера для воспроизведения прикрепленных к личным сообщениям аудиозаписей и взаимодействия с хранилищем аудиозаписей «ВКонтакте».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Koltin Documentation [Электронный ресурс]. – Электронные данные. – Режим доступа: https://kotlinlang.org/docs/reference/ ;

2. Аналитические технологии для прогнозирования и анализа данных [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.neuroproject.ru/forecasting_tutorial.php ;

3. Анализ временных рядов и прогнозирование /Н. А. Садовникова, Р. А. Шмойлова — М.: Московский финансово-промышленный университет «Синергия», 2016 — 152 с.

4. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — Питер, 2007 — 366 с.