**ВВЕДЕНИЕ**

Современный уровень развития вычислительной техники предоставляет много возможностей ее использования для различного рода занятий. Компьютерные программы становятся все более функциональными и удобными для пользователя, а их количество постоянно растет.

Каждый день человек выполняет свои повседневные задачи, движется к поставленным целям, работает над собой. Задумываясь о том, почему в один день получилось сделать огромный кусок работы, будь то постройка дома, разработка программного обеспечения или что-либо еще, а через несколько дней не удалось выполнить и половины задач; в голову приходит понятие продуктивности. Продуктивность – это способность человека создавать за определенный промежуток времени некое количество чего-либо или совершать определенное количество действий. Не имеет значения, где работает человек, чем выше продуктивность, тем выше и скорость достижения целей, решения задач и, как следствие, покорения новых вершин.

О продуктивности знают многие, однако не так много людей знают о том, что её можно повысить, причем сделать это достаточно просто. Существует множество различных методик повышения продуктивности, которым действительно стоит уделить внимание. Но это вещи, которые требуют определенных изменений в повседневной жизни, поэтому речь пойдет не о них. Самый простой способ повысить продуктивность: время от времени слушать любимую музыку. Нет определенных указаний по поводу того, когда лучше слушать музыку – это зависит от конкретного человека.

Бесчисленное количество сервисов предлагает Вам прослушивание музыки онлайн. Большинство из них требует регистрацию, а некоторые еще и являются платными.

Тем самым стала ясна задача: программное средство должно давать доступ к музыке офлайн, использовать базу музыки из Вконтакте, иметь простой дизайн и быть мультиплатформенным и простым в использовании.

В сети трудно найти программное средство, решающее все эти проблемы в совокупности, в основном данные ПС решают поставленную задачу лишь частично. Поэтому было решено сделать такое ПС самому.

Таким образом, целью данной курсовой работы является написание приложения PlusAcc, которое предоставит выше указанные возможности и будет служить помощником в повышении продуктивности.

В данной курсовой работе будут представлены следующие разделы.

Первый раздел – аналитический обзор литературы и существующих аналогов. Рассматриваются аналоги данного программного средства, их достоинства и недостатки. Происходит постановка задачи.

Второй раздел - разработка алгоритма. Описывается работа с аудиоданными в языке Java и разработка чата.

Третий раздел – разработка программного средства. В данном разделе описываются иерархия инструментов, присутствующих в программе, вспомогательные классы. Рассматриваются инструменты для разработки пользовательского интерфейса.

Четвёртый раздел – обоснование технических приёмов программирования. Обосновывается выбор языка программирования и протоколов передачи данных.

Пятый раздел – экспериментальные исследования программного средства. Проверяется работоспособность программного средства.

Шестой раздел – руководство пользователя.

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

* 1. **Обзор аналогов**

Аналоги данного приложения достаточно сложно найти. Существует множество плагинов для браузеров, которые требуют выполненного входа в учетную запись непосредственно в самом браузере, что не всегда удобно. Так же было найдено приложение VKMusic 4.

VKMusic 4 – универсальная программа для скачивания музыки и видео из Вконтакте, Youtube, Rutube (Рисунок 1.1).

Основные возможности и особенности:

* Скачивание музыки и видео из Вконтакте, Youtube, Rutube;
* Клиент Вконтакте;

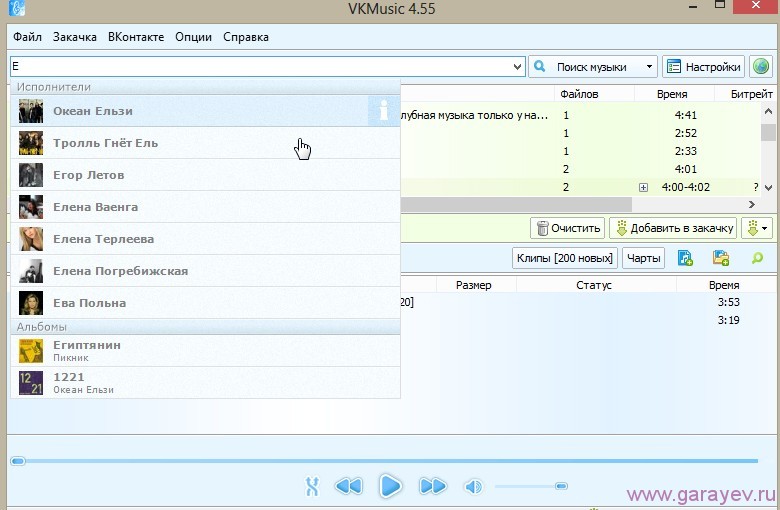


Рисунок 1.1 – Главное окно VKMusic

Основными достоинствами VKMusic 4 можно назвать: бесплатное распространение программного средства и возможность загружать видео.

VKSaver – плагин для скачивания музыки из Вконтакте (Рисунок 1.2).

Основные возможности и особенности:

* Скачивание музыки из Вконтакте;
* Работа из браузера;



Рисунок 1.2 – Принцип работы VKSaver

Основными достоинствами VKSaver можно назвать: бесплатное распространение программного средства и работа из браузера.

* 1. **Постановка задачи**

Исходя из анализа предметной области и аналогов программного средства, можно поставить следующую задачу для курсового проекта: разработать программное средство, которое будет выполнять следующие функции:

* Быстрая и корректная работа приложения;
* Простота использования для конечного пользователя;
* Возможность загрузки сразу всего списка песен пользователя;
* Использование минимальных системных ресурсов;

При разработке программного средства использовался объектно-ориентированный язык программирования Java и интегрированную среду разработки программного обеспечения IntelliJ IDEA.

1. **РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**
   1. **Работа с аудиоданными**

Музыкальные файлы проделывают следующий путь: сначала они отсылаются сервером в виде списка, который хранит в себе всю основную информацию о песнях, включая ссылки на сами песню, затем клиент выводит информацию о песнях, получает аудиозаписи по ссылке и может загрузить их на свою машину.

Запись производится с помощью стандартного класса VKMusic, представляющего собой поток. Для его создания нам необходимо в предложенном окне войти в учетную запись.

После входа создается рабочая площадка для пользователя, на которой он имеет возможность выбрать песни и загрузить их все, за что отвечает класс Downloader.

При загрузке аудиозаписи выводится строка состояния, за что так же отвечает класс VKMusic.

* 1. **Реализация QueryToVK**

Данный класс отвечает за запросы на сервер Вконтакте. Все запросы отправляются getPage, в качестве аргумента которого определяется запрос в виде String. Запросы формируются в различных методах в зависимости от того, требуется ли уникальный токен клиента для запроса.

При формировании запросов учитывается уникальный код приложения, токен клиента, а также сам запрос вида users.get.

Метод getPage использует стандартные компоненты URL и URLConnection, которые отвечают за отправку запросов.

Сервер возвращает ответы в виде JSON объектов, информацию из которых получает класс JSONParser.

* 1. **Реализация JSONParser**

При создании объекта класса, аргументом конструктора является строка, которую нужно распарсить. Парсинг происходит при помощи подключаемых классов org.json.simple, которые имеют в себе стандартные средства для выполнения требуемых задач.



Рисунок 3.1 – структура объекта JSON

**2.4 Использование MVC**

Для написания данного программного средства использовалась схема MVC, так как этот подход позволяет решить задачу проще и позволяет в дальнейшем легко улучшать данное приложение.

Model-view-controller (MVC, «модель-представление-контроллер», «модель-вид-контроллер») — схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. Данная схема проектирования часто используется для построения архитектурного каркаса.

Ниже представлено наглядное пояснение принципа MVC.

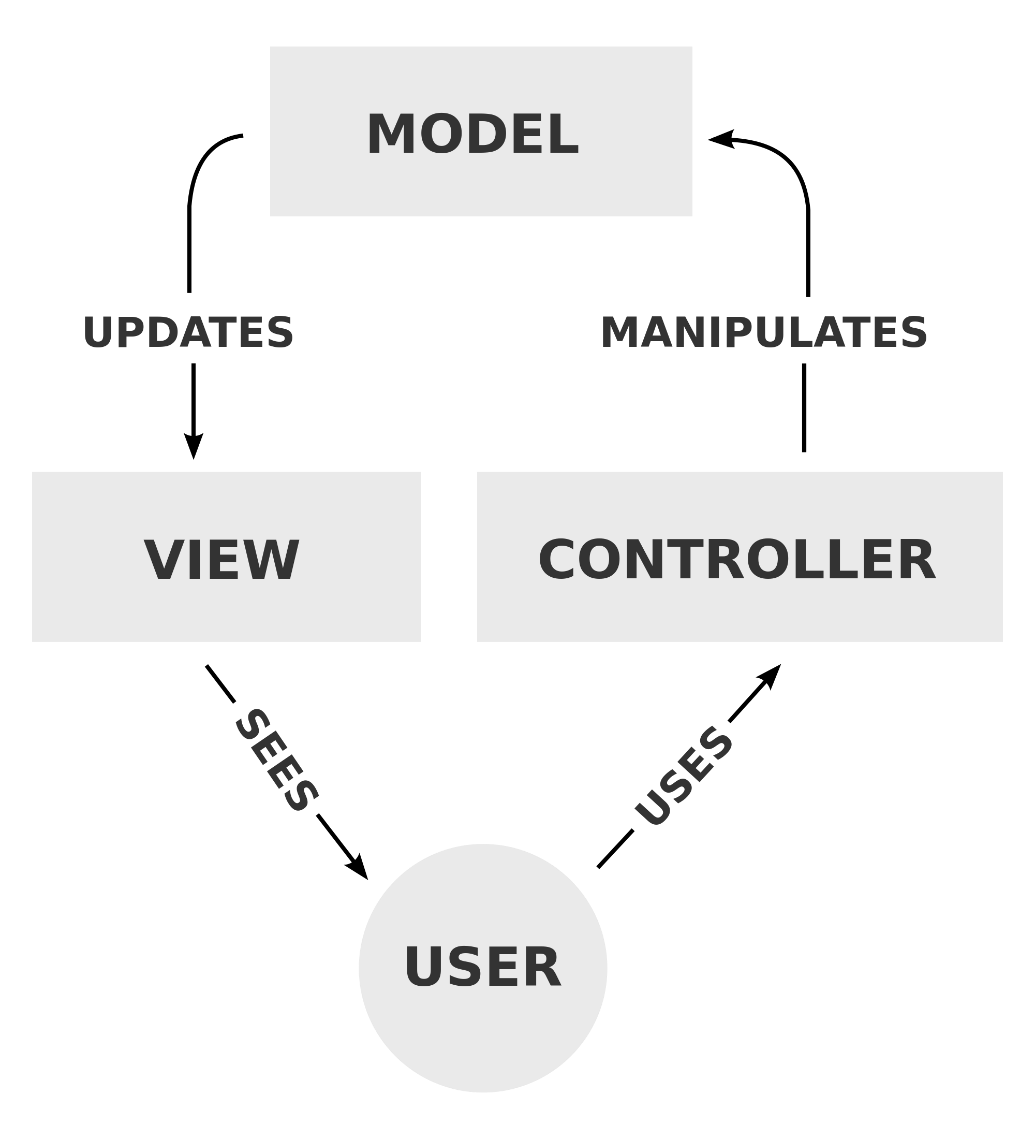


Рисунок 3.1 – Принцип MVC

**2.5 Описание вспомогательных классов**

В данном разделе хотелось бы рассказать про два вспомогательных класса AudioUtil и MessageUtil. Они решают вопрос бинарной сериализации объекта, то есть преобразовывают объект к массиву байт и обратно.

Для целей сериализации используются два потока ObjectInputStream и ObjectOutputStream. Изначально в их конструкторы планировалось передавать InputStream и OutputStream взятые у сокета. Однако возникала проблема: при создании объектных потоков программа останавливалась (deadlock). Поэтому было решено создать ObjectInputStream и ObjectOutputStream на основе FileInputStream и FileOutputStream соответственно. Это замедлит работу приложения, но решает проблему остановки программы.

**2.6 Создание графического пользовательского интерфейса**

Используя язык Java создавать графические пользовательские интерфейсы сложнее, чем в том же C# или Delphi. Однако было два варианта (библиотеки) – Swing и JavaFX. [1]

Swing – библиотека для создания графических пользовательских интерфейсов для программ на языке программирования Java. Была разработана компанией Sun Microsystems в 1998 г. Содержит ряд графических компонентов, таких как кнопки, поля ввода и т.д.

JavaFX – платформа для создания насыщенных интернет-приложений, позваоляет строить унифицированные приложения с насыщенным графическим интерфейсом пользователя для непосредственного запуска из-под операционной системы, работы в браузере и на мобильных телефонах. Первая версия появилась в 2009 году, разработана компанией Oracle.

Выбор остановился на библиотеке JavaFX. Данная платформа более новая, на десять лет младше Swing. Позволяет строить более гибкие и расширяемые интерфейсы, поддерживает стилизацию CSS (Cascade Style Sheets), что даёт возможность украсить свои приложения, как угодно. JavaFX является более удобной с позиции написания кода. [1]

Интерфейс, созданный с помощью JavaFX, сохраняется в файле с расширением fxml – своеобразный XML. Так как написание интерфейса среднего по наполнению окна может занять около 500 строк fxml-кода, что займёт много времени, рекомендуется использовать графические конструкторы интерфейсов, которые сами генерируют fxml-файл. Для данного проекта использовался программный продукт Scene Builder (рисунок 3.2).

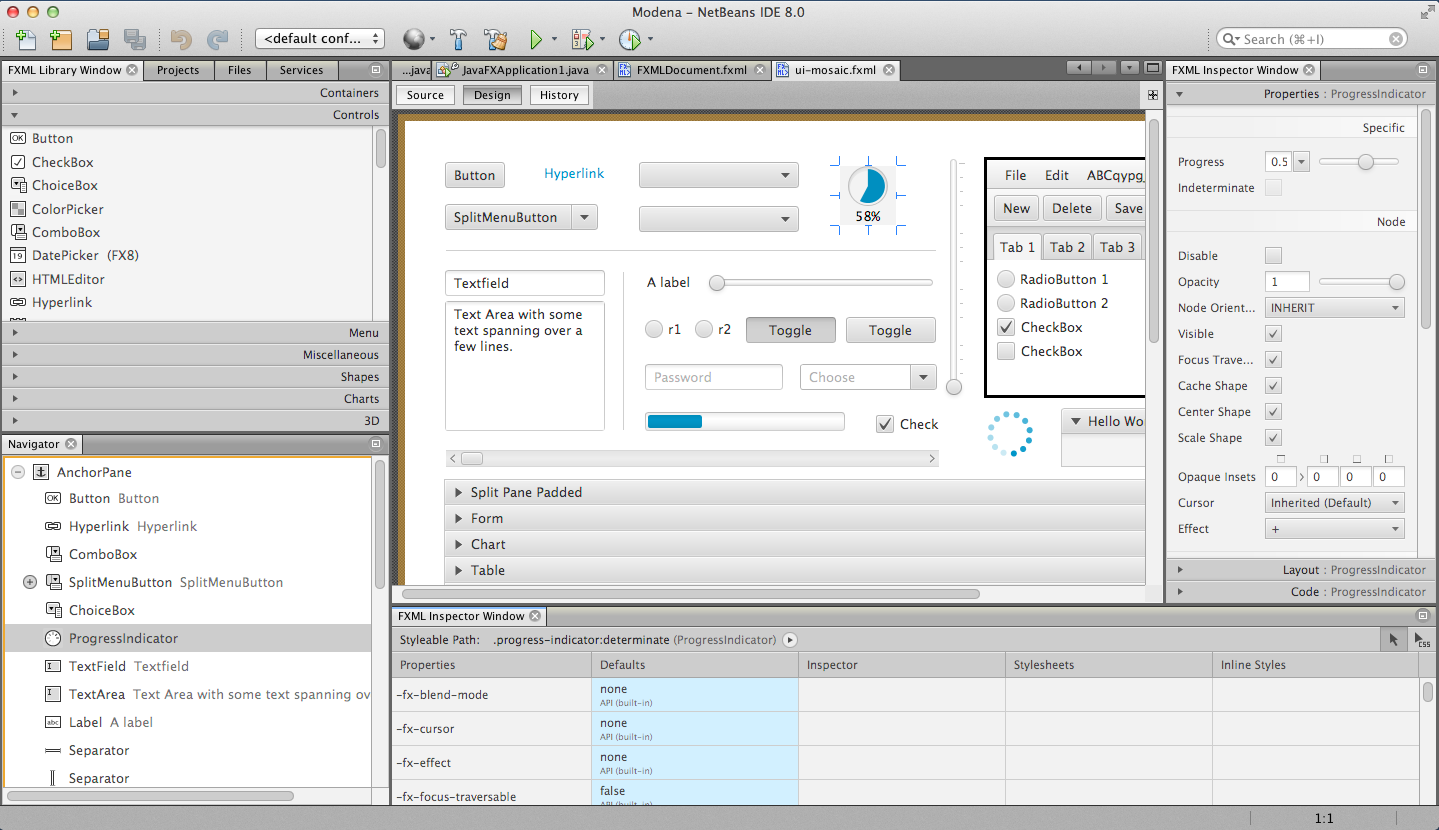


Рисунок 3.2 – Интерфейс GUI-конструктора Scene Builder

1. **ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЁМОВ ПРОГРАММИРОВАНИЯ**
   1. **Выбор языка**

Язык программирования Java реализует множество парадигм программирования, например, объектно-ориентированную, которая сыграла важную роль в реализации данного проекта.

Также приложения, написанные на языке Java, не зависят от платформы, что тоже важно. Язык Java подойдёт всем, кто пользуется не только операционной системой Windows, т.к. все необходимые для разработки инструменты есть для Mac OS и Linux. Так как я поклонник операционных систем ядра UNIX, это было одним из решающих позиций при выборе языка.

Язык Java имеет синтаксис похожий на синтаксис языка Си, поэтому программистам, знающим язык Си, будет легко разобраться в проекте.

Разрабатывать программы без ошибок гораздо легче, чем на каком-либо другом языке, т.к. любая операция, которая может бросить Exception, должна быть заключена в блок try, а в блоке catch эта исключительная ситуация должна быть обработана.

Также в виртуальной машине Java, которая обрабатывает байт-код, встроен автоматический сборщик мусора. Это значит, что нам не надо следить за освобождением памяти, т.к. любой не использующийся объект будет удалён сборщиком мусора.

* 1. **Выбор протоколов передачи данных**

В данной работе используются два протокола передачи данных: UDP и TCP. Это протоколы транспортного уровня сетевой модели стека сетевых протоколов OSI.

Транспортный уровень модели предназначен для обеспечения надёжной передачи данных от отправителя к получателю. При этом уровень надёжности может варьироваться в широких пределах. Существует множество классов протоколов транспортного уровня, начиная от протоколов, предоставляющих только основные транспортные функции (например, функции передачи данных без подтверждения приема), и заканчивая протоколами, которые гарантируют доставку в пункт назначения нескольких пакетов данных в надлежащей последовательности, мультиплексируют несколько потоков данных, обеспечивают механизм управления потоками данных и гарантируют достоверность принятых данных.

Механизм TCP предоставляет [поток данных](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) с предварительной установкой соединения, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым, в отличие от [UDP](https://ru.wikipedia.org/wiki/UDP), целостность передаваемых данных и уведомление отправителя о результатах передачи. [4]

В языке Java протокол TCP реализован с помощью класса Socket и ServerSocket.

ServerSocket – класс, использующийся на серверных приложениях. В конструктор данного класса передаётся порт, за которым будет закреплён данный сокет и ip-адрес машины, на которой запущен соект сервера. Имеется метод accept (), который ожидает подключения клиентов, выполнение программы приостанавливается до момента подключения. После данный метод возвращает объект типа Socket, который используется для общения с клиентом.

Для передачи данных между сокетами используются потоки InputStream и OutputStream, которые можно достать как на стороне сервера, так и стороне клиента, с помощью методов getInputStream (для записи) и getOutputStream (для считывания данных).

Класс Socket используется для подключения к ServerSocket. В конструктор передаётся порт и ip-адрес сервера, за которым закреплён ServerSocket. Из объекта типа Socket, также можно достать потоки чтения и записи.

UDP использует простую модель передачи, без установления связи для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, UDP предоставляет ненадёжный сервис, и дейтаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты, что может оказаться невозможным в [системах реального времени](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%80%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B2%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%B8).

Данный протокол отличается высокой скоростью передачи, что обуславливается ненадобностью создания подключения и проверки доставки данных, что позволяет его использовать в системах реального времени.

UDP протокол предоставляет возможность групповой рассылки, что необходимо в программе курсового проекта.

UDP в языке Java реализуется с помощью следующих классов: DatagramSocket, DatagramPacket, MulticastSocket.

DatagramSocket используется для передачи и отправки дейтаграмм, пакетов типа DatagramPacket. Имеется метод send, который передаёт UDP-пакеты. В этом пакете указывается ip-адрес и порт конечной точки, куда мы хотим отправить данные.

MulticastSocket используется для группой передачи на стороне клиента, данный сокет закрепляется за определённым портом и присоединяется к сетевой группе методом joinGroup, которому передаётся адрес группы. На платформе Java для адресов групп возможно использование адресов в промежутке 224.0.0.1 – 239.255.255.255. Для получения дейтаграмм используется метод receive, как и в классе DatagramSocket.

Из дейтаграммы DatagramPacket данные можно извлечь с помощью метода getData, он возвращает массив байт.

Так как нам необходимо передавать графические объекты и сообщения с высокой надёжностью и в правильном порядке, то для их передачи будем использовать протокол TCP.

Для передачи же звука используется протокол UDP, т.к. необходима возможность быстрой передачи пакетов (возможно без полной сохранности), а также групповая рассылка.

1. **ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ ПРОГРАММНОГО СРЕДСТВА**
   1. **Проверка входа в учетную запись в приложении**

Для данной проверки следует выполнить следующую последовательность действий:

1. Запустить приложение.
2. Дождаться открытия главного окна входа.
3. Ввести учетные данные.
4. Дождаться входа. При неудаче будет выдана ошибка (Рисунок 4.1), при этом будет возможно повторить вход.
5. Когда будут введены корректные данные, приложение откроет свою рабочую область. После 3 попыток неправильного входа, приложение закроется.

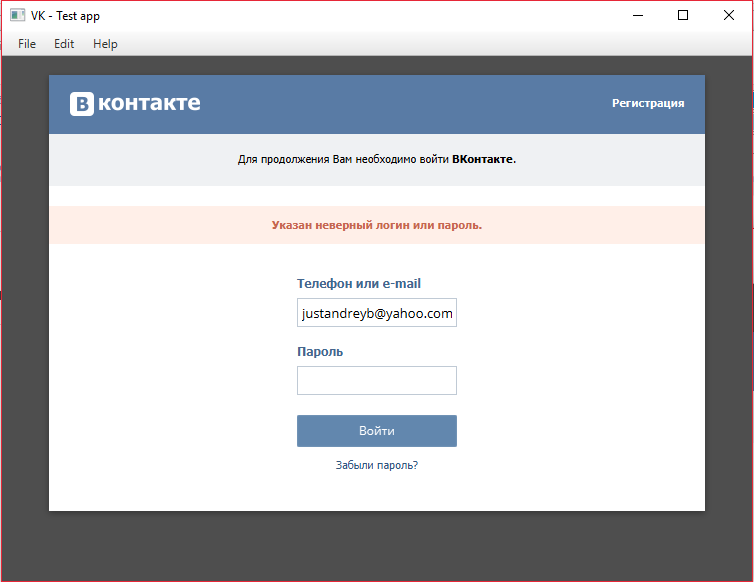


Рисунок 4.1 – Окно ошибки при подключении

При выполнении данных действий и при правильных настройках соединение между клиентом и сервером устанавливается успешно.

* 1. **Проверка загрузки аудиозаписей**

Для проверки загрузки аудиозаписей необходимо произвести подключение, как это сделать описано в разделе 4.1. Далее нам необходимо нажать кнопку Download (Рисунок 4.2) и произойдет загрузка. В противном случае приложение уведомит об ошибке.

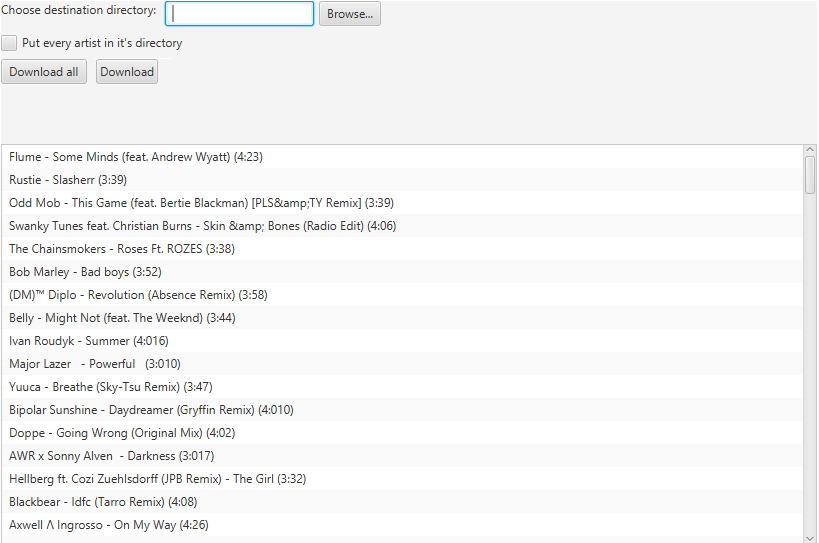


Рисунок 4.2 – Окно ошибки при подключении

При выполнении данных действий и при правильных настройках соединение между клиентом и сервером устанавливается успешно

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**
   1. **Руководство пользователя**
      1. **Включение приложения**

Нужно запустить приложение, кликнув по его ярлыку. Появится окно как на рисунке 5.1.

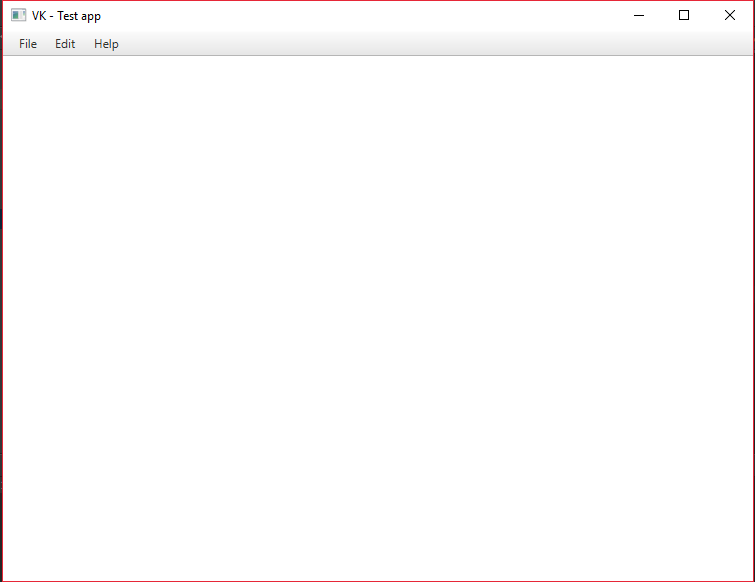


Рисунок 5.1 – Окно запуска

Server IP предстваляет собой сетевой адрес машины, на которой будет работать сервер. Чтобы его узнать необходимо в консоли ввести следующую команду ifconfig – для Linux (ipconfig – для Windows). Появится информация о всех ваших сетевых подключениях, там вы найдёте необходимый адрес.

Group IP – это сетевой адрес группы, к которой будут подключаться клиенты для получения аудиоданных. Её значение может быть получено следующим образом: первые три числа из Server IP остаются, а четвёртое – 255. Пример, если Server IP равен 192.168.43.186, то Group IP должен быть 192.168.43.255.

* + 1. **Использование приложения**

После включения нужно дождаться окна появления ввода логина и пароля (Рисунок 6.2).

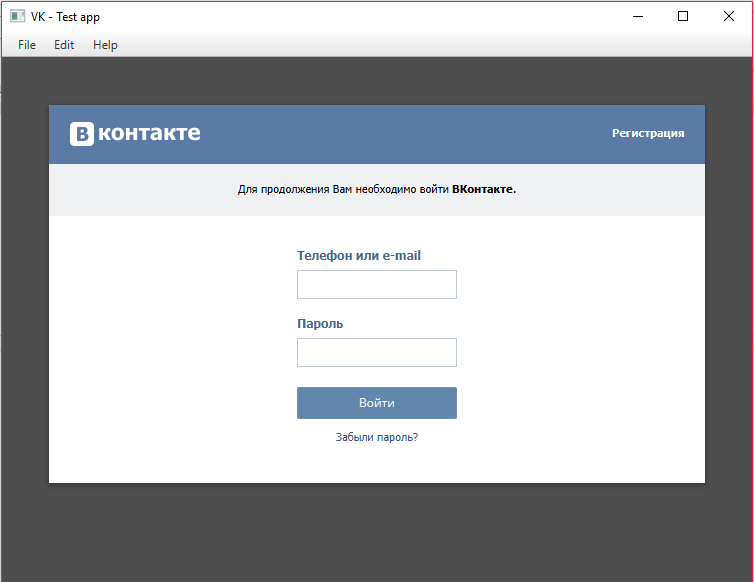


Рисунок 6.2 – Главное окно входа

Ввести логин и пароль от учетной записи и дождаться открытия главного окна (Рисунок 6.3). На данном окне присутствуют кнопки Download и Download All. Данные кнопки предоставляют пользователю доступ к загрузке аудиозаписей.

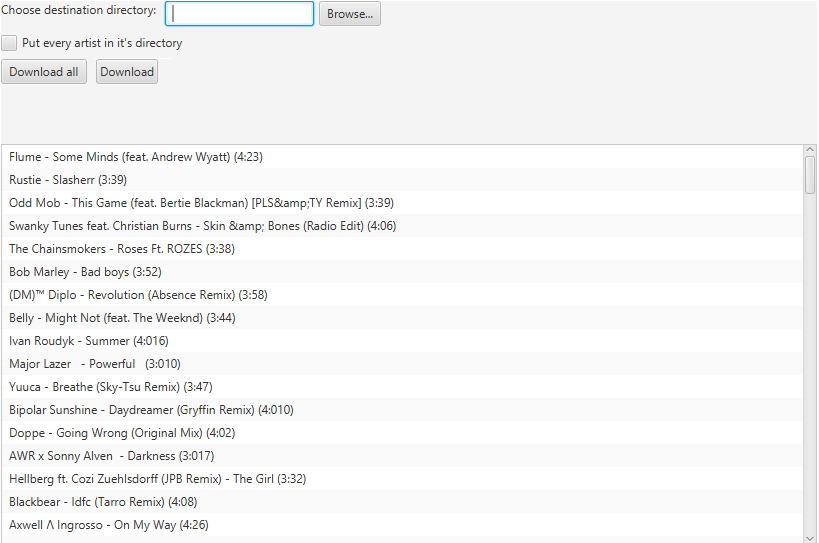


Рисунок 6.3 – Главное окно приложения

Затем для трансляции звука необходимо включить микрофон, для этого необходимо нажать на кнопку с иконкой микрофона (если микрофон зачёркнут, значит микрофон отключён, если не зачёркнут – работает).

Окно сервера разбито на области (рисунок 6.2):

1. Панель инструментов, здесь находятся все графические компоненты, регулятор ширины кисти, панель для выбора цвета кисти, кнопка очистки экрана, кнопка включения/отключения микрофона и кнопка для начала/останова трансляции.
2. Область изменений на экране (рисунок 6.3), здесь в виде таблицы отображаются все добавления на канвас (упорядочены по времени добавления, последние добавленные внизу). Для удаления какого-либо графического элемента необходимо в таблице выбрать строку, которая отвечает за удаляемый элемент и нажать комбинацию клавиш Alt + Delete.
3. Область рисования, здесь происходит отрисовка графичеких объектов.
4. Область чата. Разделена на три части. На верхней области отображаюся исходящие и входящие сообщения. В нижней области с надписью-подсказкой «Enter your message...» вводятся сообщения для отправки. Кнопка Send отвечает за отправку введённых сообщений.

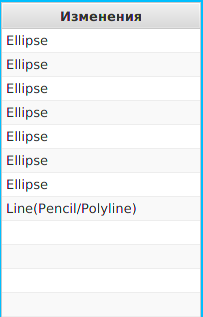


Рисунок 6.4 - Изменения на области рисования

Для создания графического элемента необходимо выбрать один из инструментов: Pencil, Polyline, Rectangle, Ellipse или Text.

Для рисования интрументом карандаш необходимо на области рисования зажать левую кнопку мыши и перетаскивать её, при освобождении ЛКМ создастся кривая и отправится на клиенты.

Для рисования инструментов полилиния необходимо кликать левой кнопкой мыши по области рисования тем самым добавляя опорные точки, через которые будет проходить полилиния. Для создания полилинии необходимо два раза кликнуть на последней опорной точке.

Принципы рисования инструментами прямоугольник и эллипс одинаковы: сначала вы одним кликом левой кнопки мыши задаёте опорную точку

Для изменения ширины кисти необходимо изменить положение регулятор (рисунок 6.4).

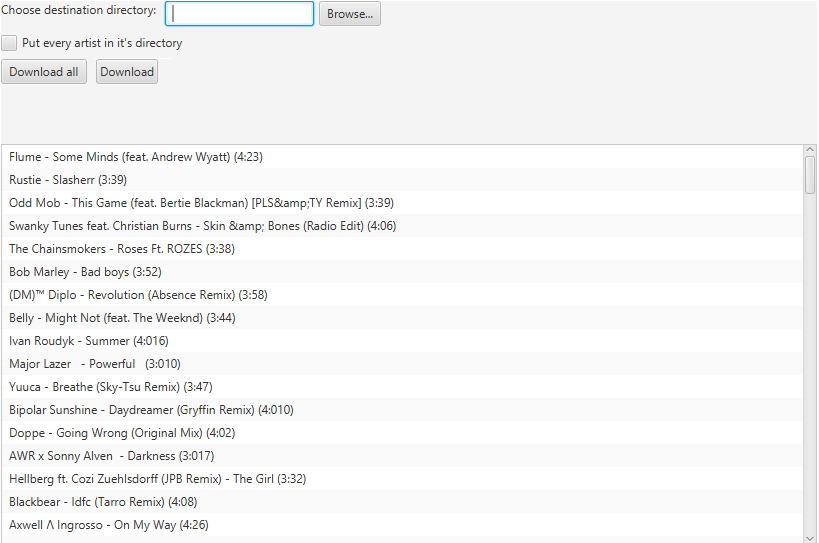


Рисунок 6.6 – Компонент для выбора цвета кисти

* 1. **Руководство пользователя загрузки**
     1. **Настройка загрузчика**

Для входа в настройки клиента необходимо нажать кнопку File, затем Settings. Окно настроек клиента (риснок 6.6) похоже на окно настроек сервера. Однако там есть дополнительное поле для имени сервера. Это имя будет отображаться в чате при отправке сообщений. Также комбобокс для для выбора считывающего устройства заменён на комбобокс для выбора воспроизводящего устройства.

Поля Server IP и Group IP заполняются значениями, которые вычисляются по правилам, описанным в разделе 6.1.1. Значения портов и сетевых адресов должны совпадать со значениями портов и адресов, на которых работает сервер, к которому вы хотите подключиться.

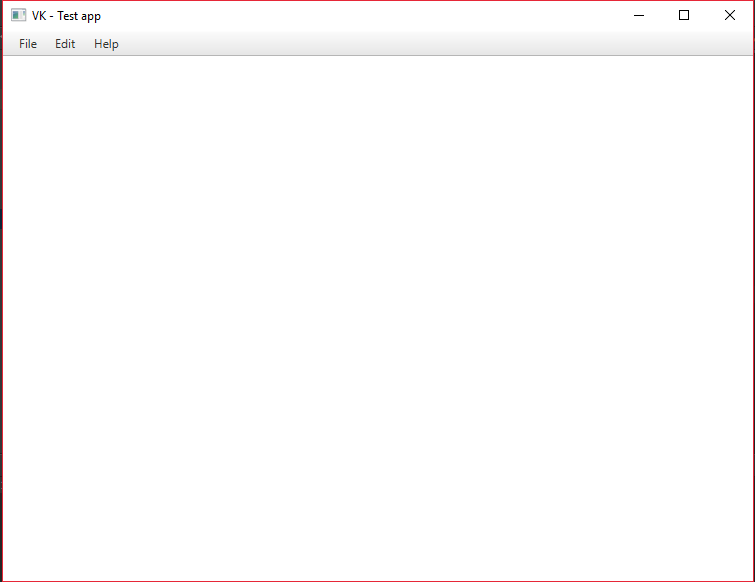


Рисунок 6.6 – Окно настроек клиента

* + 1. **Использование загрузчика**

Окно клиента разбито на три части:

1. Полотно, на котором будут отображаться присланные с сервера графические элементы.
2. Поле настроек имеет включатель/отключатель воспроизводящего устройства и кнопку для соединения/отсоединения.
3. Область чата. Представляет собой такой же интерфейс, как и на сервере, для обмена сообщениями.

Работа за клиентом очень проста, для соедиения с сервером необходимо нажать кнопку Connect. Для отправки сообщения введённого в поле с подсказкой «Enter your text...» осуществляется нажатием кнопки Send.

Настройки клиента во время сеанса с сервером недоступны во избежание изменения в настройках во время онлайн лекции.

**ЗАКЛЮЧЕНИЕ**

В ходе данной курсовой работы были проведены анализ поставленной задачи, исследование существующих аналогов, работа с литературой, разработка алгоритмов, экспериментальные исследования созданного приложения, составление руководства пользователя.

Все возможности программы, которые обсуждались при постановке задачи, предоставляются пользователям. При поиске решения программной реализации были использованы наиболее подходящие алгоритмы, структуры данных и методы программирования.

Были проведены многочисленные тесты, в ходе которых явные неисправности обнаружены не были. Количество тестов было достаточным, чтобы утверждать, что приложение будет стабильно работать при любых возможных входных данных.

В ходе разработке приложения было получено много знаний и опыта по работе с языком программирования Java, средой программирования на данном языке IntelliJ IDEA, конструктором интерфейсов Scene Builder. Закреплены знания по работе с библиотекой Java FX, попутно для украшения были изучены некоторые области CSS.

Так как в данном программном средстве данного курсового проекта было много работы с сетевыми технологиями, были углублены знания в области системного программирования.

Также надо отдать должное объектно-ориентированному методу программирования, который облегчил труд над программой, помог структурировать проект.

**СПИСОК ЛИТЕРАТУРЫ**

[1] JavaFX Documentation [Электронный ресурс] – 2009. - Режим доступа: http://docs.oracle.com/javafx/2/. – Дата доступа: 15.02.2016.

[2] Java Platform Standart Edition 7 Documentation [Электронный ресурс] – 2011. – Режим доступа: <https://docs.oracle.com/javase/7/docs/> . – Дата доступа: 15.02.2016.

[3] UDP - Википедия [Электронный ресурс] – 2015. - Режим доступа: https://ru.wikipedia.org/wiki/UDP. – Дата доступа: 10.04.2016.

[4] TCP - Википедия [Электронный ресурс] – 2015. - Режим доступа: https://ru.wikipedia.org/wiki/TCP. – Дата доступа: 10.04.2016.

[5] Elliote Rusty Harold, Java Network Programming / Elliote Rusty Harold. – 4-ое изд. – O’Reilly Media, 2013. – 502 c.

[6] Сериализация в Java [Электронный ресурс] – 2009. – Режим доступа: <https://habrahabr.ru/post/60317/> . – Дата доступа: 25.03.2016.

# ПРИЛОЖЕНИЕ А.

# ЛИСТИНГ ПРОГРАММЫ

**package** client;  
  
**import** fx.Controller;  
  
**import** javax.sound.sampled.AudioFormat;  
**import** java.io.IOException;  
**import** java.net.InetAddress;  
**import** java.net.SocketException;  
**import** java.util.ArrayList;  
**import** java.util.Arrays;  
**import** java.util.regex.Matcher;  
**import** java.util.regex.Pattern;  
**import** java.util.regex.PatternSyntaxException;  
  
**public class** ClientNetworkManager {  
 **static final int *CHAT\_PORT*** = 61320;  
 **public static final int *CHECK\_INSTANCE\_PORT*** = 61309;  
 **static final int *RECORD\_VOICE\_SIZE*** = 4 \* 500;  
 **static final int *FORMAT\_TYPE\_NUM*** = 1;  
 **public static final** String ***USER\_EXIT\_MSG*** = **"VTCPUSEREXIT"**;  
 **public static final** String ***USER\_UPDATE\_MSG*** = **"VTCPUPDATELIST"**;  
 **private final** Controller **controller**;  
  
 **private** VoiceSender **voiceSender** = **null**;  
 **private** VoiceReceiver **voiceReceiver** = **null**;  
 **private** TextTransmitter **textTransmitter** = **null**;  
 **private boolean isInitialized** = **false**;  
 **private** ArrayList<String> **users**;  
 **private** String **nickname**;  
  
 **public** ClientNetworkManager(Controller controller) {  
 **this**.**controller** = controller;  
 }  
  
 **public boolean** initializeConnection(InetAddress address, String nickname) **throws** IOException {  
 **textTransmitter** = **new** TextTransmitter(address, **controller**, **this**);  
 **textTransmitter**.getClientSocket().setSoTimeout(5000);  
 **textTransmitter**.sendTextMessage(**"VTCPHELO "** + nickname.trim());  
 **this**.**nickname** = nickname;  
 String message = **textTransmitter**.receiveTextMessage();  
 String args[] = message.split(**" "**);  
 **int** sendPort, receivePort;  
 **if**(args.**length** == 3) {  
 **if** (args[0].equals(**"VTCPHELOCLIENT"**)) {  
 **try** {  
 receivePort = Integer.*parseInt*(args[1]);  
 sendPort = Integer.*parseInt*(args[2]);  
 } **catch** (NumberFormatException e) {  
 **return isInitialized** = **false**;  
 }  
 } **else** {  
 **return isInitialized** = **false**;  
 }  
 } **else** {  
 **return isInitialized** = **false**;  
 }  
 **textTransmitter**.sendTextMessage(**"VTCPUSERLIST"**);  
 message = **textTransmitter**.receiveTextMessage();  
 args = message.split(**" "**);  
 **if** (args.**length** > 0) {  
 **users** = **new** ArrayList<>();  
 **users**.add(nickname);  
 **if** (args[0].trim().equals(**"USERS:"**)) {  
 **users**.addAll(Arrays.*asList*(args).subList(1, args.**length**));  
 **voiceSender** = **new** VoiceSender(address, sendPort);  
 **voiceReceiver** = **new** VoiceReceiver(receivePort);  
 printUserList();  
 **controller**.**chatTextArea**.appendText(**"Connected to server\n"**);  
 **textTransmitter**.getClientSocket().setSoTimeout(0);  
 **isInitialized** = **true**;  
 } **else** {  
 **isInitialized** = **false**;  
 }  
 } **else** {  
 **isInitialized** = **false**;  
 }  
 **return isInitialized**;  
 }  
  
  
 **public void** setVolumeLevel(**float** level) **throws** ClientNotInitializedException {  
 **if** (!**isInitialized**) {  
 **throw new** ClientNotInitializedException();  
 }  
 **voiceReceiver**.setVolumeLevel(level);  
 }  
  
 **public void** setMuted(**boolean** state) **throws** ClientNotInitializedException {  
 **if** (!**isInitialized**) {  
 **throw new** ClientNotInitializedException();  
 }  
 **voiceSender**.setMuted(state);  
 }  
  
 **public void** startTransfer() **throws** ClientNotInitializedException {  
 **if** (!**isInitialized**) {  
 **throw new** ClientNotInitializedException();  
 }  
 **voiceSender**.start();  
 **voiceReceiver**.start();  
 **textTransmitter**.start();  
 }  
  
 **public void** sendTextMessage(String message) **throws** IOException, ClientNotInitializedException {  
 **if** (!**isInitialized**) {  
 **throw new** ClientNotInitializedException();  
 }  
 **textTransmitter**.sendTextMessage(**this**.**nickname** + **": "** + message);  
 }  
  
 **public void** stopTransfer() {  
 **if** ((**voiceSender** != **null**) && (**voiceReceiver** != **null**) && (**textTransmitter** != **null**)) {  
 **try** {  
 **textTransmitter**.sendTextMessage(***USER\_EXIT\_MSG***);  
 } **catch** (SocketException ignored) {  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **voiceSender**.stopThread();  
 **voiceReceiver**.stopThread();  
 **textTransmitter**.stopThread();  
 }  
 }  
  
 **private void** printUserList() {  
 **controller**.**usersTextArea**.clear();  
 **if** (**users** != **null**) {  
 **for** (String user : **users**) {  
 **controller**.**usersTextArea**.appendText(user + **"\n"**);  
 }  
 }  
 }  
  
 **boolean** isUpdateUserList(String message) {  
 String args[] = message.split(**" "**);  
 **if** (args.**length** > 0) {  
 **users** = **new** ArrayList<>();  
 **if** (args[0].trim().equals(***USER\_UPDATE\_MSG***)) {  
 **users**.addAll(Arrays.*asList*(args).subList(1, args.**length**));  
 printUserList();  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 **static** AudioFormat getAudioFormat(**int** num) {  
 **switch** (num){  
 **case** 0: **return new** AudioFormat(AudioFormat.Encoding.***ULAW***, 8000.0f, 8, 1, 1, 8000.0f, **false**);  
 **case** 1: **return new** AudioFormat(AudioFormat.Encoding.***PCM\_SIGNED***, 22050.0f, 16, 1, 2, 22050.0f, **true**);  
 **case** 2: **return new** AudioFormat(AudioFormat.Encoding.***PCM\_SIGNED***, 44100.0f, 16, 1, 2, 44100.0f, **true**);  
 }  
 **return new** AudioFormat(AudioFormat.Encoding.***PCM\_SIGNED***, 22050.0f, 16, 1, 2, 22050.0f, **true**);  
 }  
  
 **public static boolean** isValidHost(String ip) {  
 **try** {  
 **if** (ip == **null** || ip.isEmpty()) {  
 **return false**;  
 }  
  
 String[] parts = ip.split(**"\\."**);  
 **if** (parts.**length** != 4) {  
 **return false**;  
 }  
  
 **for** (String s : parts) {  
 **int** i = Integer.*parseInt*(s);  
 **if** ((i < 0) || (i > 255)) {  
 **return false**;  
 }  
 }  
 **return** !ip.endsWith(**"."**);  
 } **catch** (NumberFormatException nfe) {  
 **return false**;  
 }  
 }  
  
 **public static boolean** isValidNickname(String name) {  
 **try** {  
 Pattern pattern = Pattern.*compile*(**"^[a-zA-Z0-9\_]+$"**);  
 Matcher matcher = pattern.matcher(name);  
 **return** matcher.matches();  
 } **catch** (PatternSyntaxException ex) {  
 **return false**;  
 }  
 }  
  
  
 **public** String getNickname() {  
 **return nickname**;  
 }  
}

**package** server;  
  
**import** fx.Controller;  
  
**import** java.io.IOException;  
**import** java.net.\*;  
**import** java.util.ArrayList;  
**import** java.util.Enumeration;  
  
**public class** ServerNetworkManager {  
  
 **static final int *CHAT\_PORT*** = 61320;  
 **private static final int *RECEIVE\_VOICE\_PORTS***[] = {61310, 61311, 61312, 61313, 61314};  
 **private static final int *SEND\_VOICE\_PORTS***[] = {61315, 61316, 61317, 61318, 61319};  
 **static final int *RECORD\_VOICE\_SIZE*** = 4 \* 500;  
 **static final** String ***USER\_EXIT\_MSG*** = **"VTCPUSEREXIT"**;  
 **static final** String ***USER\_UPDATE\_MSG*** = **"VTCPUPDATELIST"**;  
  
 **private boolean**[] **sendAllowPorts** = {**true**, **true**, **true**, **true**, **true**};  
 **private boolean**[] **receiveAllowPorts** = {**true**, **true**, **true**, **true**, **true**};  
  
 **private** ArrayList<UserConnection> **users** = **new** ArrayList<>();  
 **private** NewConnectionListener **listener**;  
  
 **public** ServerNetworkManager(Controller controller) **throws** IOException {  
 **listener** = **new** NewConnectionListener(controller, **users**, **this**);  
 }  
  
 **public void** startServer() {  
 **listener**.start();  
 **users** = **new** ArrayList<>();  
 }  
  
 **public void** stopServer() {  
 **listener**.stopThread();  
 **listener**.interrupt();  
 **for** (UserConnection user: **users**) {  
 user.stopThread();  
 user.interrupt();  
 }  
 }  
  
 **public static** String getServerAddress() {  
 Enumeration e = **null**;  
 **try** {  
 e = NetworkInterface.*getNetworkInterfaces*();  
 } **catch** (SocketException e1) {  
 e1.printStackTrace();  
 }  
 **if** (e != **null**) {  
 **while**(e.hasMoreElements())  
 {  
 NetworkInterface n = (NetworkInterface) e.nextElement();  
 Enumeration ee = n.getInetAddresses();  
 **while** (ee.hasMoreElements())  
 {  
 InetAddress i = (InetAddress) ee.nextElement();  
 **if**(*isValidHost*(i.getHostAddress()) && !(i.getHostAddress().equals(**"127.0.0.1"**))) {  
 **return** i.getHostAddress();  
 }  
 }  
 }  
 }  
 **return "127.0.0.1"**;  
 }  
  
 **private static boolean** isValidHost(String ip) {  
 **try** {  
 **if** (ip == **null** || ip.isEmpty()) {  
 **return false**;  
 }  
  
 String[] parts = ip.split(**"\\."**);  
 **if** (parts.**length** != 4) {  
 **return false**;  
 }  
  
 **for** (String s : parts) {  
 **int** i = Integer.*parseInt*(s);  
 **if** ((i < 0) || (i > 255)) {  
 **return false**;  
 }  
 }  
 **return** !ip.endsWith(**"."**);  
 } **catch** (NumberFormatException nfe) {  
 **return false**;  
 }  
 }  
  
 **int** getFreeSendPort() {  
 **for** (**int** i = 0; i < ***SEND\_VOICE\_PORTS***.**length**; i++) {  
 **if**(**sendAllowPorts**[i]) {  
 **sendAllowPorts**[i] = **false**;  
 **return *SEND\_VOICE\_PORTS***[i];  
 }  
 }  
 **return** -1;  
 }  
  
 **void** freePort(**int** port) {  
 **for** (**int** i = 0; i < ***SEND\_VOICE\_PORTS***.**length**; i++) {  
 **if**(***SEND\_VOICE\_PORTS***[i] == port) {  
 **sendAllowPorts**[i] = **true**;  
 **return**;  
 }  
 }  
 **for** (**int** i = 0; i < ***RECEIVE\_VOICE\_PORTS***.**length**; i++) {  
 **if**(***RECEIVE\_VOICE\_PORTS***[i] == port) {  
 **receiveAllowPorts**[i] = **true**;  
 **return**;  
 }  
 }  
 }  
  
 **int** getFreeReceivePort() {  
 **for** (**int** i = 0; i < ***RECEIVE\_VOICE\_PORTS***.**length**; i++) {  
 **if**(**receiveAllowPorts**[i]) {  
 **receiveAllowPorts**[i] = **false**;  
 **return *SEND\_VOICE\_PORTS***[i];  
 }  
 }  
 **return** -1;  
 }  
}

**package** server;  
  
**import** fx.Controller;  
  
**import** java.io.DataInputStream;  
**import** java.io.DataOutputStream;  
**import** java.io.EOFException;  
**import** java.io.IOException;  
**import** java.net.InetAddress;  
**import** java.net.Socket;  
**import** java.net.SocketException;  
**import** java.util.ArrayList;  
  
**import static** server.ServerNetworkManager.***USER\_EXIT\_MSG***;  
**import static** server.ServerNetworkManager.***USER\_UPDATE\_MSG***;  
  
**class** UserConnection **extends** Thread {  
  
  
 **private final** Socket **clientSocket**;  
 **private** DataOutputStream **outputStream**;  
 **private** DataInputStream **inputStream**;  
 **private** Controller **controller**;  
 **private final int id**;  
 **private** String **nickname**;  
 **private boolean isEnabled** = **true**;  
 **private** ArrayList<UserConnection> **users**;  
 **private** VoiceConnection **voiceConnection**;  
 **private int sendPort**;  
 **private int receivePort**;  
 **private** ServerNetworkManager **manager**;  
  
 UserConnection(**int** id, Controller controller, Socket socket, ArrayList<UserConnection> users, ServerNetworkManager manager) {  
 **this**.**manager** = manager;  
 **this**.**users** = users;  
 **this**.**controller** = controller;  
 **this**.**id** = id;  
 **this**.**clientSocket** = socket;  
 **try** {  
 **outputStream** = **new** DataOutputStream(socket.getOutputStream());  
 **inputStream** = **new** DataInputStream(socket.getInputStream());  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 setDaemon(**true**);  
 }  
  
 **boolean** initializeConnection() **throws** IOException {  
 String message = receiveMessage();  
 String args[] = message.split(**" "**);  
 **if** (args.**length** == 2) {  
 **if** (args[0].equals(**"VTCPHELO"**)) {  
 **nickname** = args[1].trim();  
 **sendPort** = **manager**.getFreeSendPort();  
 **receivePort** = **manager**.getFreeReceivePort();  
 **if**(**receivePort** == -1 || **sendPort** == -1) {  
 **return false**;  
 }  
 sendMessage(**"VTCPHELOCLIENT "** + **sendPort** + **" "** + **receivePort**);  
 message = receiveMessage();  
 **if** (message.equals(**"VTCPUSERLIST"**)) {  
 sendUserList(**false**);  
 **this**.**voiceConnection** = **new** VoiceConnection(**receivePort**, **id**, **users**);  
 **voiceConnection**.start();  
 **return true**;  
 }  
 }  
 }  
 **return false**;  
 }  
  
 @Override  
 **public void** run() {  
 **while** (**isEnabled**) {  
 **try** {  
 String message = receiveMessage();  
 **if**(message.equals(***USER\_EXIT\_MSG***)) {  
 **controller**.**trayList**.appendText(**"User \""** + **nickname** + **"\" disconnected\n"**);  
 **users**.remove(**this**);  
 **controller**.**clientsList**.clear();  
 **for** (UserConnection user : **users**) {  
 **controller**.**clientsList**.appendText(user.getNickname()+ **":"** + user.getID() + **"\n"**);  
 user.sendMessage(**"User \""** + **nickname** + **"\" disconnected\n"**);  
 user.sendUserList(**true**);  
 }  
 **this**.stopThread();  
 **return**;  
 }  
 **controller**.**trayList**.appendText(**"MSG> "** + message.trim() + **"\n"**);  
 **for** (UserConnection user : **users**) {  
 user.sendMessage(message);  
 }  
 } **catch** (EOFException e) {  
 **users**.remove(**this**);  
 **this**.stopThread();  
 e.printStackTrace();  
 **this**.stopThread();  
 } **catch** (SocketException e) {  
 **controller**.**trayList**.appendText(**"Lost connection with user \""** + **nickname** + **"\"\n"**);  
 **users**.remove(**this**);  
 **this**.stopThread();  
 } **catch** (IOException e) {  
 **users**.remove(**this**);  
 **this**.stopThread();  
 e.printStackTrace();  
 }  
 }  
 }  
  
 String getNickname() {  
 **return nickname**;  
 }  
  
 **int** getID() {  
 **return id**;  
 }  
  
 **synchronized void** stopThread() {  
 **isEnabled** = **false**;  
 **voiceConnection**.stopThread();  
 **manager**.freePort(**sendPort**);  
 **manager**.freePort(**receivePort**);  
 **try** {  
 **outputStream**.close();  
 **inputStream**.close();  
 **clientSocket**.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private void** sendMessage(String message) **throws** IOException {  
 **outputStream**.writeUTF(message);  
 }  
  
 **private** String receiveMessage() **throws** IOException {  
 **return inputStream**.readUTF();  
 }  
  
 **void** sendUserList(**boolean** isUpdate) **throws** IOException {  
 StringBuilder output;  
 **if**(isUpdate) {  
 output = **new** StringBuilder(***USER\_UPDATE\_MSG***);  
 } **else** {  
 output = **new** StringBuilder(**"USERS:"**);  
 }  
  
 **for** (UserConnection user : **users**) {  
 output.append(**" "**).append(user.getNickname().trim());  
 }  
 sendMessage(output.toString());  
 }  
  
 InetAddress getAddress() {  
 **return clientSocket**.getInetAddress();  
 }  
  
 **public int** getSendPort() {  
 **return sendPort**;  
 }  
}

package com.Model.VK.audio;  
  
import org.json.simple.JSONArray;  
import org.json.simple.JSONObject;  
  
import java.util.ArrayList;  
  
public class TrackList extends ArrayList <Track>{  
  
 public TrackList(JSONArray jsonArray) {  
 for (Object trackObject : jsonArray) {  
 this.add(new Track((JSONObject)trackObject));  
 }  
 }  
}

package com;  
  
import com.Controller.WorkplaceController;  
import com.Model.VK.Downloader;  
import com.Model.VK.QueryToVK;  
import com.Model.VK.VkApi;  
import com.Model.VK.audio.AudioApi;  
import com.Model.VK.audio.Track;  
import com.Model.VK.audio.TrackList;  
import javafx.application.Application;  
import javafx.application.Platform;  
import javafx.beans.property.ReadOnlyBooleanProperty;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.concurrent.Worker;  
import javafx.fxml.FXMLLoader;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.Node;  
import javafx.scene.Scene;  
import javafx.scene.control.\*;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.layout.\*;  
import javafx.scene.web.WebEngine;  
import javafx.scene.web.WebView;  
import javafx.stage.DirectoryChooser;  
import javafx.stage.Stage;  
import org.json.simple.parser.ParseException;  
  
import java.io.File;  
import java.io.IOException;  
import java.io.UnsupportedEncodingException;  
import java.net.MalformedURLException;  
import java.net.URL;  
import java.net.URLDecoder;  
import java.util.HashMap;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class VKNotifer extends Application {  
  
 public static final String *LOGIN\_SUCCESS\_PAGE* = "blank.html#", *LOGIN\_FAILURE\_PAGE* = "blank.html#error";  
 private volatile boolean loginSuccess = false, loginFailure = false;  
 private String formData = null;  
  
 private Stage primaryStage;  
 private BorderPane rootLayout;  
 private BorderPane workplaceLayout;  
  
 HashMap<String, URL> fxmlPaths = new HashMap<>();  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
 @Override  
 public void start(Stage primaryStage) throws Exception {  
 this.primaryStage = primaryStage;  
 this.primaryStage.setTitle("VK - Test app");  
  
 setFXMLPaths();  
  
 initRootLayout();  
 initVKConnection();  
  
 }  
  
 public void initRootLayout() {  
 try {  
  
 FXMLLoader loader = new FXMLLoader();  
 loader.setLocation(getFXMLPath("RootPage"));  
 rootLayout = loader.load();  
  
 // Show the scene containing the root layout.  
 Scene scene = new Scene(rootLayout);  
 primaryStage.setScene(scene);  
 primaryStage.show();  
  
// initWorkplaceLayout();  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
  
 /\* public void initWorkplaceLayout() throws IOException, ParseException {  
  
 FXMLLoader loaderWorkplace = new FXMLLoader();  
 loaderWorkplace.setLocation(getFXMLPath("Workplace"));  
 workplaceLayout = loaderWorkplace.load();  
  
 loaderWorkplace.setController(workplaceController = new WorkplaceController(formData));  
  
 }\*/  
  
 public void initVKConnection() throws Exception {  
  
 VkApi VK = new VkApi();  
 final WebView webView = new WebView();  
 final WebEngine webEngine = webView.getEngine();  
 webEngine.load(VK.getAuthRequest());  
 primaryStage.show();  
 webEngine.getLoadWorker().stateProperty().addListener(  
 (ov, oldState, newState) -> {  
 if (newState == Worker.State.*SUCCEEDED*) {  
 changeState(webEngine.getLocation());  
 }  
 });  
  
 rootLayout.setCenter(webView);  
 primaryStage.show();  
  
 new Thread(() -> {  
  
 while (!loginSuccess && !loginFailure && primaryStage.isShowing()) ;  
  
 if (loginFailure || (!primaryStage.isShowing())) {  
 Platform.*exit*();  
 } else {  
 try {  
 AudioApi api = new AudioApi(formData);  
 final TrackList tracks = api.getTrackList();  
 final ListView<Track> list = new ListView<>();  
 ObservableList<Track> items = FXCollections.*observableArrayList*(tracks);  
 list.setItems(items);  
 Label destLabel = new Label("Choose destination directory: ");  
 final TextField dest = new TextField();  
 Button browse = new Button("Browse...");  
 browse.setOnAction(t -> {  
 DirectoryChooser fc = new DirectoryChooser();  
  
 File dir = fc.showDialog(primaryStage);  
 if (dir != null) {  
 dest.setText(dir.toString());  
 }  
 });  
 HBox browseLayout = new HBox(5);  
 browseLayout.getChildren().addAll(destLabel, dest, browse);  
 final VBox mainLayout = new VBox(8);  
 Button downloadAll = new Button("Download all");  
 final Button cancel = new Button("Cancel");  
 final HBox downloadLayout = new HBox(5);  
 downloadLayout.getChildren().add(downloadAll);  
 downloadLayout.getChildren().add(cancel);  
 cancel.setVisible(false);  
 final ProgressBar current = new ProgressBar();  
 final ProgressBar global = new ProgressBar();  
 current.setPrefWidth(primaryStage.getWidth());  
 global.setPrefWidth(primaryStage.getWidth());  
 current.setVisible(false);  
 global.setVisible(false);  
 final CheckBox artistDir = new CheckBox("Put every artist in it's directory");  
 mainLayout.getChildren().addAll(browseLayout, artistDir, downloadLayout,  
 current, global, list);  
 final Downloader downloader = new Downloader(tracks, current, global);  
 primaryStage.setOnCloseRequest(t -> downloader.stop());  
 downloadAll.setOnAction(t -> {  
 downloader.setDir(new File(dest.getText()));  
 downloader.setArtistDir(artistDir.isSelected());  
 new Thread(downloader).start();  
 });  
 cancel.setOnAction(t -> downloader.stop());  
 Platform.*runLater*(() -> primaryStage.setScene(new Scene(mainLayout)));  
 ControlsChecker checker = new ControlsChecker(downloader, primaryStage);  
 checker.setCancelButton(cancel);  
 checker.setDownloadButton(downloadAll);  
 checker.setPrimaryStage(primaryStage);  
 checker.setCurrentProgressBar(current);  
 checker.setGlobalProgressBar(global);  
 new Thread(checker).start();  
 } catch (Exception ex) {  
 new Exception(ex);  
 }  
 }  
 }).start();  
 }  
  
 protected void changeState(String Url) {  
 if (Url.contains(*LOGIN\_FAILURE\_PAGE*)) {  
 loginFailure = true;  
 } else if (Url.contains(*LOGIN\_SUCCESS\_PAGE*)) {  
 loginSuccess = true;  
 try {  
 this.formData = URLDecoder.*decode*(Url.substring(Url.indexOf(*LOGIN\_SUCCESS\_PAGE*) + *LOGIN\_SUCCESS\_PAGE*.length()), "UTF-8");  
 } catch (UnsupportedEncodingException ex) {  
 Logger.*getLogger*(VKNotifer.class.getName()).log(Level.*SEVERE*, null, ex);  
 }  
 }  
 }  
  
 public URL getPathURL(String targetFile) throws MalformedURLException {  
 String defaultPath = "file:" +  
 System.*getProperty*("user.dir").replaceAll("[\\\\]", "/") +  
 "/src/com/";  
 return new URL(defaultPath + targetFile);  
 }  
  
 private void setFXMLPaths() throws MalformedURLException {  
  
 this.fxmlPaths.clear();  
 addFXMLPath("RootPage");  
 addFXMLPath("Workplace");  
 addFXMLPath("Audio");  
 addFXMLPath("Photos");  
 addFXMLPath("Wall");  
 }  
  
 public void addFXMLPath(String key) throws MalformedURLException {  
  
 this.fxmlPaths.put(key, getPathURL("View/" + key + ".fxml"));  
  
 }  
  
 private URL getFXMLPath(String targetFile) {  
 return this.fxmlPaths.get(targetFile);  
 }  
  
 private class ControlsChecker implements Runnable {  
  
 volatile Downloader downloader;  
 volatile Stage primaryStage = null;  
 Button cancelButton = null;  
 Button downloadButton = null;  
 ProgressBar current = null;  
 ProgressBar global = null;  
  
 public ControlsChecker(Downloader downloader, Stage primaryStage) {  
 this.downloader = downloader;  
 this.primaryStage = primaryStage;  
 }  
  
 public void setCancelButton(Button cancelButton) {  
 this.cancelButton = cancelButton;  
 }  
  
 public void setDownloadButton(Button downloadButton) {  
 this.downloadButton = downloadButton;  
 }  
  
 public void setPrimaryStage(Stage primaryStage) {  
 this.primaryStage = primaryStage;  
 }  
  
 public void setCurrentProgressBar(ProgressBar current) {  
 this.current = current;  
 }  
  
 public void setGlobalProgressBar(ProgressBar global) {  
 this.global = global;  
 }  
  
 @Override  
 public void run() {  
 boolean running = downloader.isRunning();  
 ReadOnlyBooleanProperty stageShowing = primaryStage.showingProperty();  
 while (stageShowing.getValue()) {  
 while (running == downloader.isRunning() && stageShowing.getValue());  
 running = downloader.isRunning();  
 if (running) {  
 setVisible(cancelButton, true);  
 setVisible(downloadButton, false);  
 setVisible(current, true);  
 setVisible(global, true);  
 } else {  
 setVisible(cancelButton, false);  
 setVisible(downloadButton, true);  
 setVisible(current, false);  
 }  
 }  
 }  
  
 public void setVisible(final Node node, final boolean visible) {  
 if (node != null) {  
 Platform.*runLater*(() -> node.setVisible(visible));  
 }  
 }  
 }  
  
}