Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Пояснительная записка к курсовой работе**

по курсу «Программирование на языке Java»

на тему «Разработка многомодульного приложения»

Выполнил:

Студент группы 19ВВ3

Табрисов С. А.

Приняла:

к.т.н. Юрова О. В.

Пенза 2022

**Содержание**

[Введение 4](#_Toc105144008)

[1 Постановка задачи 5](#_Toc105144009)

[2 Выбор решения 6](#_Toc105144010)

[3 Описание программы 7](#_Toc105144011)

[3.1 Серверная часть 7](#_Toc105144012)

[3.2 Клиентская часть 8](#_Toc105144013)

[4 Описание способа организации пользовательского интерфейса 11](#_Toc105144014)

[5 Описание результатов работы программы 13](#_Toc105144015)

[Заключение 20](#_Toc105144016)

[Список используемых источников 21](#_Toc105144017)

[Приложение 22](#_Toc105144018)

[Приложение А «Листинг клиентской части приложения» 22](#_Toc105144019)

[MainController.java 22](#_Toc105144020)

[Client.java 26](#_Toc105144021)

[Приложение Б «Листинг серверной части приложения» 28](#_Toc105144022)

[Server. java 28](#_Toc105144023)

[ConnectDB.java 30](#_Toc105144024)

[Приложение В «UML диаграммы» 32](#_Toc105144025)

[Диаграмма вариантов использования 32](#_Toc105144026)

[Диаграммы классов 33](#_Toc105144027)

[Диаграмма деятельности 35](#_Toc105144028)

[Диаграмма развёртывания 36](#_Toc105144029)

[Диаграмма последовательности 37](#_Toc105144030)

Введение

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Первая версия языка появилась еще в 1996 году в недрах компании Sun Microsystems, впоследствии поглощенной компанией Oracle. Java задумывался как универсальный язык программирования, который можно применять для различного рода задач. И к настоящему времени язык Java проделал большой путь, было издано множество различных версий. Текущей версией является Java 18, которая вышла 22 марта 2022 года. А Java превратилась из просто универсального языка в целую платформу и экосистему, которая объединяет различные технологии, используемые для целого ряда задач: от создания десктопных приложений до написания крупных веб-порталов и сервисов. Кроме того, язык Java активно применяется для создания программного обеспечения для множества устройств: обычных ПК, планшетов, смартфонов и мобильных телефонов и даже бытовой техники.

После изучения основ данного языка, чтобы закрепить полученный материал была выбрана разработка многомодульного клиент-серверного приложения, а именно музыкальныный сервис. Идейным вдохновением послужили такие стриминговые сервисы, как “Spotify”, “Apple Music”, “Яндекс музыка”.

# Постановка задачи

Разработать приложение клиент-серверной архитектуры, которое позволяет прослушивать музыку пользователю.

Функции сервера: прием и передача информации от пользователя, авторизация и регистрация пользователя, отправка запросов к базе данных.

Функции клиента: графический интерфейс пользователя, сетевое взаимодействие с сервером.

Операционная система: Windows;

Среда разработки: IntelliJ IDEA 2022;

Язык программирования Java.

Используемые технологии:

* Java Collections Framework
* JDBC
* Механизм обработки исключительных ситуаций
* Java Stream API
* Java Multithreading
* Сетевое взаимодействие

# Выбор решения

В данном курсовом проекте соединение клиента с сервером реализовано с помощью классов DatagramPacket и DatagramSocket(протокол UDP).

UDP - это протокол связи без установления соединения, то есть во время передачи данных отправитель и получатель данных не устанавливают логическое соединение. То есть, когда один компьютер отправляет данные на другой компьютер, отправляющая сторона не будет подтверждать, существует ли принимающая сторона, и будет отправлять данные. Также, когда принимающая сторона получает данные, она не будет передавать отправляющей стороне, следует ли получать.

Обычно приложение клиент-сервер, состоит из двух разных приложений - клиента, инициирующего соединение с целью (сервером), и сервера, ожидающего соединения от клиента.

На стороне клиента приложение имеет свой адрес и номер порта. Перед отправкой сообщения серверу, клиент создает datagramsocket, используемый для получения пакетов данных, и datagrampacket, используемый для инкапсуляции данных, в том числе адреса и порта. Отправляя запрос на соединение, клиент пытается установить соединение с сервером:

Если события развиваются удачно, при условии, что сервер запущен прежде, чем клиент попытался с ним соединиться, сервер получает пакет данных от клиента, при этом извлекая адрес и порт. После этого происходит взаимодействие между клиентом и сервером через сокет.

Пользовательский интерфейс реализован с помощью JavaFX.

# Описание программы

Данная программа состоит из клиентской и серверной части. Первым запускается сервер. На остальных компьютерах локальной сети - клиент. Сервер представлен в виде консоли, клиентская часть - оконное приложение.

Программа предоставляет пользователю поиск, добавление и прослушивание аудиофайлов.

## Серверная часть

**Server.java** – главный файл серверного процесса. Здесь реализуется функция Main. Задается порт и объявляется сокет. Происходит ожидание клиента для создания с ним связи. В методе main реализована обрабока сообщений от клиента.

**User.java** - класс, в котором хранится информация о пользователе. Здесь присутствуют два метода CompareUser(), RegistrationUser().

CompareUser() – Метод сравнивает введенный пароль с паролем, хранящемся в базе данных.

RegistrationUser() – При регистрации пользователя проверяет введенное имя на уникальность. Если все проходит успешно, то данные о новом пользователе заносятся в базу данных.

**ConnectDB.java** - класс, в котором происходит подключение к базе данных.

getDBConnect() – Устанавливает соединение с базой данных.

InsertUser(String userName, String userMail, String userPassword) – Добавляет нового пользователя в базу данных.

SelectUser(String userName) – Выполняет запрос на получение пользователя из базы данных.

searchMusicList(String s) – Выполняет поиск треков в базе данных.

InsertMusicInPlaylistDB(String name, String musicName) – Добавляет в базу данных трек, который пользователь добавил к себе в плейлист.

getUserPlayList(String name) – Загрузка плейлиста пользователя.

**ConstValues.java** – класс, в котором прописаны константы для работы

**Config.java** - класс, хранящий параметры базы данных(порт, хост, логин, пароль, название).

## Клиентская часть

**Main.java** – файл, в котором происходит загрузка окна входа, запуск приложения, подключение к серверу.

**User.java** – класс, в котором хранятся данные пользователя.

**Music.java** - класс, для создания аудиофайла.

**Client.java** - класс, в котором происходит инкапсуляции всего, что связано с соединением с сервером.

Client() – конструктор по умолчанию, который создает сокет и адрес.

SendMessage(int param, String outMessage) – Инкапсулирует запрос от клиента и отправляет на сервер.

SendMessage(String outMessage) – Отправляет запрос на подключение к серверу.

ReceiveMessage() – Получает ответ от сервера.

ReceiveMessage() – Получает ответ от сервера в виде сериализованного списка.

**RegController.java** - класс, который отвечает за регистрацию пользователя.

CompleteRegistration(ActionEvent actionEvent) – Проверяет поля ввода, если регистрация прошла успешно, то пользователь переходит в главное меню.

CheckNameUser(User user) – Отправляет данные серверу для проверки и возвращает ответ.

**LoginController.java** - класс, отвечающий за вход пользователя в приложение.

SignUpBtn(ActionEvent e) - Проверяет поля ввода, если авторизация прошла успешно, то пользователь переходит в главное меню.

GoToRegistration(ActionEvent actionEvent) – Обрабатывает событие нажатия на кнопку “Регистрация”. Загружает окно регистрации.

LoginUser(String name, String password) – Отправляет введенные данные серверу для проверки и возвращает ответ.

**MainController.java** - класс, описывающий логику клиентского приложения.

Initialize(URL url, ResourceBundle resourceBundle) – метод, который инициализирует объекты @FXML. В нем происходит обработка событий, связанных с графическим интерфейсом, загрузка плейлиста пользователя.

Данный метод содержит обработку событий:

userPlayList.getSelectionModel().selectedItemProperty().addListener – отслеживание выбранного трека в плейлисте пользователя и его воспроизведение.

volumeSlider.valueProperty().addListener – отслеживание изменения громкости.

search.textProperty().addListener – отслеживание набранного текста в поисковой строке. Во время ввода происходит отправка текста на сервер, далее происходит ожидание результата и внесение найденных треков в список.

searchList.getSelectionModel().selectedItemProperty().addListener – отслеживание выбранного трека по результатам поиска и его воспроизведение.

addTrack.setOnMouseClicked – обработка события добавления трека в плейлист. Отправление резульата на сервер для последующего добавления в базу данных.

playSong(Music music) – воспроизведение аудиофайла. Также данный метод отслеживает события:

mediaPlayer.setOnReady – в новом потоке устанавливает текущее значение слайдера.

timeSlider.setOnMouseClicked – обработка перемотки.

mediaPlayer.currentTimeProperty().addListener – устанавливает текущее время аудиофайла.

getTimeString(double millis) – преобразует миллисикунды в минуты и секунды.

beginTimer() – устанавливает таймер для аудиофайла.

cancelTimer() – закрывает таймер.

playMusic(MouseEvent actionEvent) – обрабатывает событие нажатия на кнопку паузы.

prevMusic(MouseEvent actionEvent) – запускает следующий трек в плейлисте пользователя.

nextMusic(MouseEvent actionEvent) – запускает предыдущий трек в плейлисте пользователя.

setMain(ActionEvent actionEvent), setSearch(ActionEvent actionEvent), setCollect(ActionEvent actionEvent) – преключение между вкладками Главная, Поиск, Ваша коллекция.

SetVolume(MouseEvent mouseEvent), AddVolume(MouseEvent mouseEvent) – устанавливает громкость трека.

enterArray(ArrayList<String> b) – преобразует массив строк в массив объектов Music/

goToLogin(ActionEvent actionEvent) – выход из приложения. Метод открывает окно авторизации.

# Описание способа организации пользовательского интерфейса

Средой разработки была выбрана IntelliJ IDEA. Для дополнительного редактирования графической части приложения использовалось дополнение в виде приложения Scene Builder. Оно помогает редактировать fxml-файлы с непосредственным графическим представлением сцены.

Клиентская часть приложения содержит 3 сцены: авторизации, регистрации и основную, в которой реализовано взаимодействие с аудиофайлами.

Сцена авторизации включает в себя два поля Textfield для ввода логина и пароля и две кнопки Button: вход, регистрация. Кнопки осуществлют переход между сценами.

Сцена регистрации пользователя имеет также поля данных ввода и кнопку завершения регистрации.

После входа или регистрации пользователь попадает в основное меню приложения. В меню с помощью кнопок реализовано переключение между вкладками Главная, Поиск, Ваша коллекция.

Вкладка поиска содержит в себе два компонента: Textfield – для поиска трека и список ListView, в котором отображается результат поиска. Список хранит в себе объекты класса Music, который содержит в себе название трека и аудиофайл в формате .mp3. Для пользователя в списке отоброжаются только названия треков.

Вкладка с плейлистом пользователя реализована аналогично поиску, за исключением поля поиска.

Для реализации управления воспроизведения трека, снизу была создана небольшая панель AnchorPane, которая хранит несколько элементов Label, ImageView и Slider. Label используется для отображения названия трека и времени его воспроизведения. Изображения ImageView выступают в роли кнопок управления (играть/пауза, следующий/предыдущий трек). С помощью Slider реализовано управление перемоткой и изменение громкости.

Для настройки отдельных элементов сцены в виде текста, списка, кнопок и фона, используются css стили.

# Описание результатов работы программы

Для начала необходимо запустить сервер (рис.1).

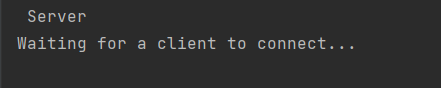


Рисунок 1 – Запуск сервера.

Далее запускаем клиент (рис.2).

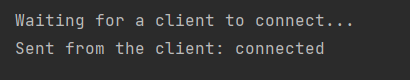


Рисунок 2 – Установка соединения.

Если вы уже зарегистрированы, то можете войти в приложение (рис.3).



Рисунок 3 – Окно авторизации.

Если вы не зарегистрированы, то можете пройти регистрацию (рис.4).

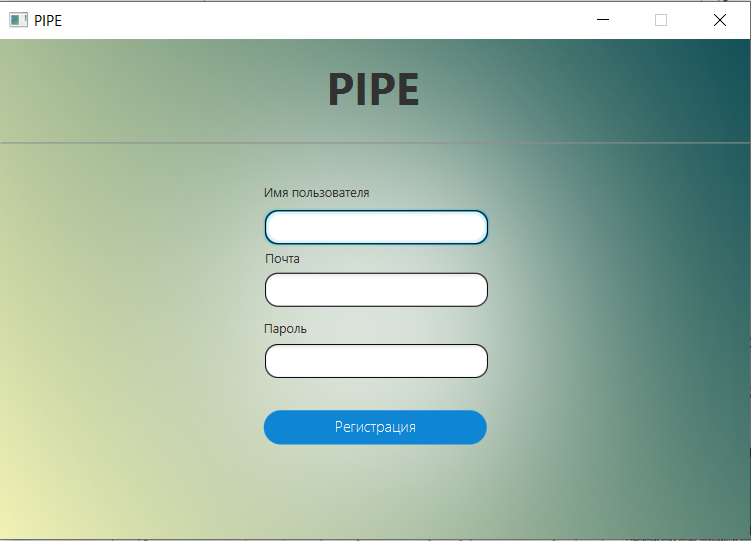


Рисунок 4 – Окно регистрации.

После регистрации или входа, пользоваетель попадает в основное меню приложения(рис.5).

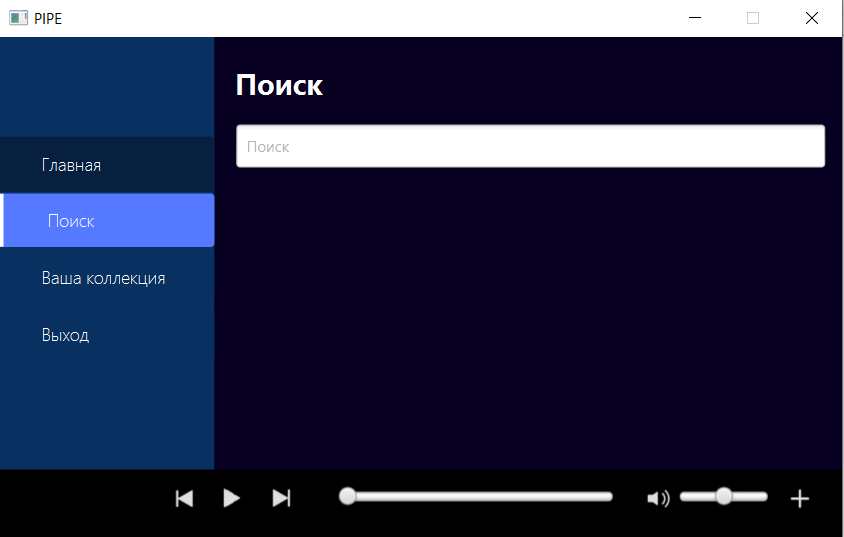


Рисунок 5 – Главное меню.

Далее пользователь может искать, добавлять и прослушивать аудиофайлы(рис.6,7,8,9,10)

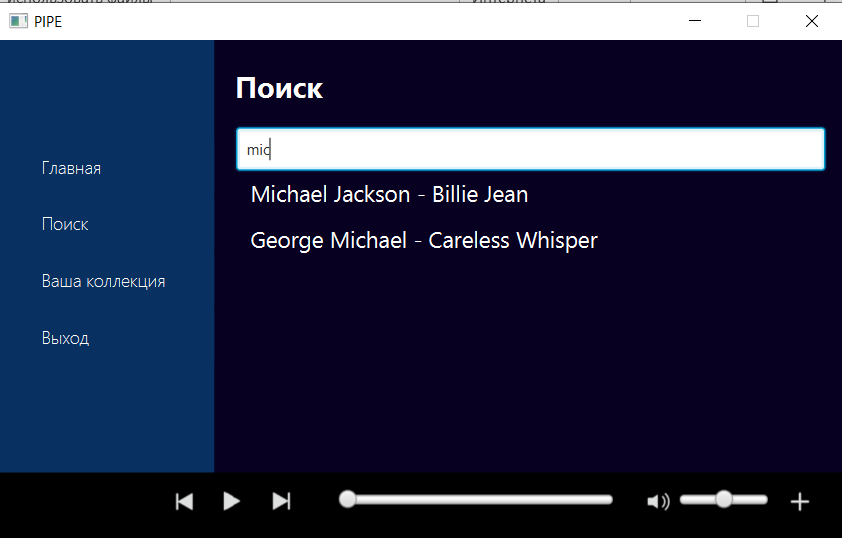


Рисунок 6 – Поиск трека.

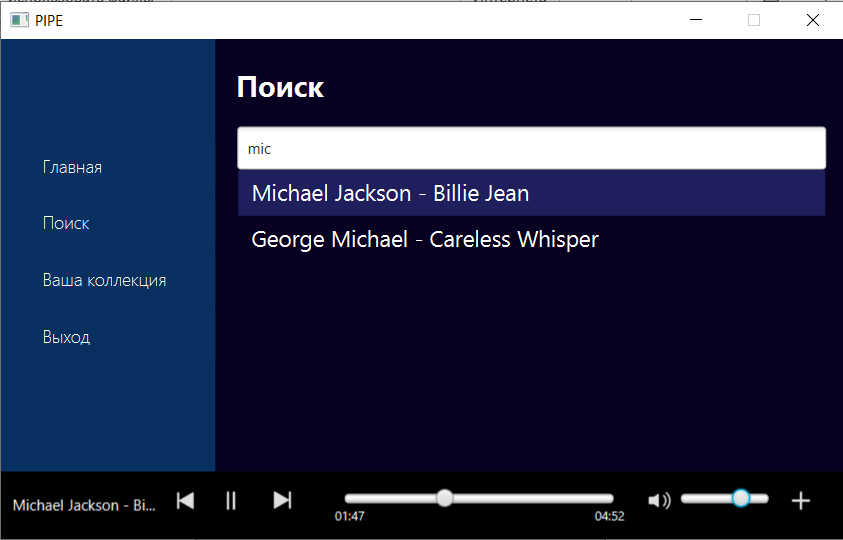


Рисунок 7 – Воспроизведение.

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описание

Рисунок 8 – Добавление в плейлист.

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описание

Рисунок 9 – Воспроизведение предыдущего трека в плейлисте.

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описание

Рисунок 10 – Воспроизведение следующего трека в плейлисте.

Плейлист пользователя сохраняется в базе данных и при следующем входе в систему плейлист автоматически загружается (рис.12).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – Сообщение о неккоректных данных.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Загрузка плейлиста при входе.

Все аудиофайлы, данные о пользователе хранятся в базе данных(рис.13,14,15).

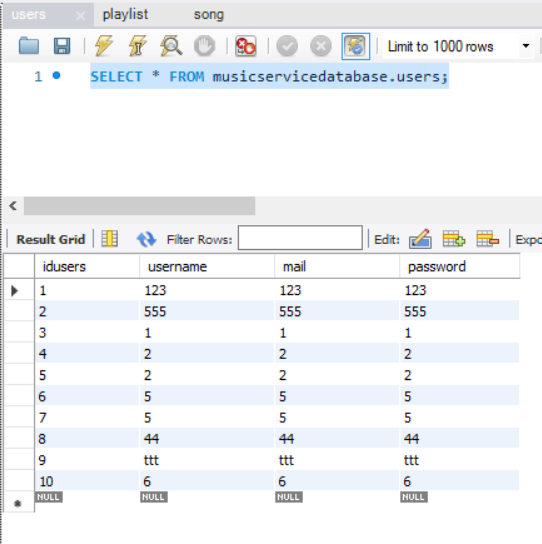


Рисунок 13 – Таблица, содержащая данные пользователей.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14 – Таблица, содержащая данные о плейлистах пользователей.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 15 – Таблица, содержащая аудиофайлы.

# Заключение

При выполнении данной курсовой работы были получены навыки разработки программ на языке высокого уровня. Были освоены приемы создания графического интерфейса. Усвоены механизмы реализации меню. Были закреплены знания сетевого взаимодействия в Jave и приобретены практические навыки работы с базой данных в многомодульных приложениях. Изучены основные возможности среды программирования IntelliJ IDEA.

В рамках выполнения курсовой работы был написан музыкальный сервис.

Не все сделано идеально и поэтому программу можно улучшить, оптимизировав код для более рационального использования графических ресурсов, оперативной памяти. Можно добавить подборку песен и плейлистов для пользователя, историю прослушанных песен, возможность создания нескольких плейлистов и тп.

# Список используемых источников

1. https://www.youtube.com/
2. https://www.cyberforum.ru/
3. https://stackoverflow.com/
4. https://habr.com/
5. https://javarush.ru/

Приложение

Приложение А «Листинг клиентской части приложения»

MainController.java

package com.example.alfa;  
  
import javafx.beans.InvalidationListener;  
import javafx.beans.Observable;  
import javafx.beans.value.ChangeListener;  
import javafx.beans.value.ObservableValue;  
import javafx.collections.FXCollections;  
import javafx.collections.ObservableList;  
import javafx.event.ActionEvent;  
import javafx.event.EventHandler;  
import javafx.event.EventType;  
import javafx.fxml.FXMLLoader;  
import javafx.fxml.Initializable;  
import javafx.scene.Node;  
import javafx.scene.Parent;  
import javafx.scene.control.\*;  
import javafx.scene.image.ImageView;  
import javafx.scene.input.MouseEvent;  
import javafx.scene.layout.AnchorPane;  
import javafx.stage.Stage;  
import javafx.scene.Scene;  
  
import java.io.ByteArrayInputStream;  
import java.io.IOException;  
import java.io.ObjectInputStream;  
import java.net.SocketException;  
import java.net.URL;  
import java.net.UnknownHostException;  
import java.util.ArrayList;  
import java.util.ResourceBundle;  
import java.util.Timer;  
import java.util.TimerTask;  
  
import javafx.scene.media.Media;  
import javafx.scene.media.MediaPlayer;  
import javafx.util.Duration;  
  
public class MainController implements Initializable {  
 public Label mainLabel;  
 public AnchorPane searchMenu;  
 public AnchorPane userCollection;  
 public ListView<Music> userPlayList;  
 public Slider volumeSlider;  
 public Label nameTrackOnPane;  
 public Slider timeSlider;  
 public Label currTime;  
 public Label timeTrack;  
 public ImageView previousTrack;  
 public ImageView playTrack;  
 public ImageView nextTrack;  
 public ImageView stopTrack;  
 public ImageView volume1;  
 public ImageView volume2;  
 public TextField search;  
 public ListView<Music> searchList;  
 public ImageView addTrack;  
  
 private Stage stage;  
 private Scene scene;  
 private Parent root;  
  
 boolean play = false;  
 public Media media;  
 public MediaPlayer mediaPlayer;  
 public Timer timer;  
 public TimerTask timerTask;  
 int currentTrackIndex=0;  
 boolean b = false;  
 ObservableList<Music> musicListObservableList = FXCollections.observableArrayList();  
 ObservableList<Music> searchMusicListObservableList = FXCollections.observableArrayList();  
 public ArrayList <Music> musicArrayList = new ArrayList<Music>();  
 public ArrayList <Music> searchMusicArrayList = new ArrayList<Music>();  
 public ArrayList <String> searchMusicArrayListS = new ArrayList<String>();  
 public ArrayList <String> userPlayListS = new ArrayList<String>();  
 public ArrayList <Music> musicPlaylist = new ArrayList<Music>();  
 Music curr;  
  
 @Override  
 public void initialize(URL url, ResourceBundle resourceBundle) {  
 Client client = null;  
 try {  
 client = new Client();  
 client.SendMessage(5, "playlist");  
 ByteArrayInputStream baos = new ByteArrayInputStream(client.ReceiveMes());  
 ObjectInputStream oos = new ObjectInputStream(baos);  
 userPlayListS = (ArrayList<String>) oos.readObject();  
 musicPlaylist = enterArray(userPlayListS);  
 musicListObservableList.addAll(musicPlaylist);  
 userPlayList.setItems(musicListObservableList);  
 } catch (SocketException e) {  
 e.printStackTrace();  
 } catch (UnknownHostException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 userPlayList.setCellFactory(param -> new ListCell<Music>() {  
 @Override  
 protected void updateItem(Music item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty || item == null || item.name == null) {  
 setText(null);  
 } else {  
 setText(item.name);  
 }  
 }  
 });  
  
 userPlayList.getSelectionModel().selectedItemProperty().addListener(new ChangeListener<Music>(){  
 public void changed(ObservableValue<? extends Music> changed, Music oldValue, Music newValue){  
 addTrack.setVisible(false);  
 currentTrackIndex = userPlayList.getSelectionModel().getSelectedIndex();  
 playSong(userPlayList.getSelectionModel().getSelectedItem());  
 }  
 });  
  
 volumeSlider.valueProperty().addListener(new ChangeListener<Number>() {  
 @Override  
 public void changed(ObservableValue<? extends Number> observableValue, Number number, Number t1) {  
 if(volumeSlider.getValue() == 0){  
 volume2.setVisible(true);  
 }else{  
 volume2.setVisible(false);  
 }  
 mediaPlayer.setVolume(volumeSlider.getValue() \* 0.01);  
 }  
 });  
  
 Client finalClient = client;  
 search.textProperty().addListener(new ChangeListener<String>() {  
 @Override  
 public void changed(ObservableValue<? extends String> observableValue, String oldV, String newV) {  
 searchMusicListObservableList.clear();  
 try {  
 finalClient.SendMessage(3, newV);  
 System.out.println(newV);  
 ByteArrayInputStream baos = new ByteArrayInputStream(finalClient.ReceiveMes());  
 ObjectInputStream oos = new ObjectInputStream(baos);  
 searchMusicArrayListS = (ArrayList<String>) oos.readObject();  
 searchMusicArrayList = enterArray(searchMusicArrayListS);  
 searchMusicListObservableList.addAll(searchMusicArrayList);  
 searchList.setItems(searchMusicListObservableList);  
 } catch (IOException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 searchList.setCellFactory(param -> new ListCell<Music>() {  
 @Override  
 protected void updateItem(Music item, boolean empty) {  
 super.updateItem(item, empty);  
 if (empty || item == null || item.name == null) {  
 setText(null);  
 } else {  
 setText(item.name);  
 }  
 }  
 });  
  
 searchList.getSelectionModel().selectedItemProperty().addListener(new ChangeListener<Music>(){  
 public void changed(ObservableValue<? extends Music> changed, Music oldValue, Music newValue){  
 addTrack.setVisible(true);  
 currentTrackIndex = searchList.getSelectionModel().getSelectedIndex();  
 if(currentTrackIndex!=-1){  
 playSong(searchList.getSelectionModel().getSelectedItem());  
 }  
 }  
 });  
  
 addTrack.setOnMouseClicked(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent mouseEvent) {  
 if(!musicArrayList.contains(searchList.getSelectionModel().getSelectedItem())){  
 musicArrayList.add(searchList.getSelectionModel().getSelectedItem());  
 musicListObservableList.add(searchList.getSelectionModel().getSelectedItem());  
 userPlayList.setItems(musicListObservableList);  
 try {  
 finalClient.SendMessage(4, searchList.getSelectionModel().getSelectedItem().name);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 });  
 }  
  
 public void playSong(Music music){  
 if(play){  
 mediaPlayer.stop();  
 cancelTimer();  
 }  
 stopTrack.setVisible(true);  
 media = new Media(music.file.toURI().toString());  
 nameTrackOnPane.setText(music.name);  
 mediaPlayer = new MediaPlayer(media);  
 mediaPlayer.play();  
 play = true;  
 beginTimer();  
 mediaPlayer.setOnReady(new Runnable() {  
 @Override  
 public void run() {  
 timeSlider.setMax(mediaPlayer.getTotalDuration().toSeconds());  
 timeTrack.setText(getTimeString(mediaPlayer.getTotalDuration().toMillis()));  
 double curr = mediaPlayer.getCurrentTime().toSeconds();  
 timeSlider.setValue(curr);  
 }  
 });  
  
 timeSlider.setOnMouseReleased(mouseEvent -> mediaPlayer.seek(Duration.seconds(timeSlider.getValue())));  
  
 timeSlider.setOnMouseReleased(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent mouseEvent) {  
 mediaPlayer.pause();  
 mediaPlayer.seek(Duration.seconds(timeSlider.getValue()));  
 mediaPlayer.play();  
 }  
 });  
  
 timeSlider.setOnMouseClicked(new EventHandler<MouseEvent>() {  
 @Override  
 public void handle(MouseEvent mouseEvent) {  
 mediaPlayer.pause();  
 mediaPlayer.seek(Duration.seconds(timeSlider.getValue()));  
 mediaPlayer.play();  
 }  
 });  
  
 mediaPlayer.currentTimeProperty().addListener(new ChangeListener<Duration>() {  
 @Override  
 public void changed(ObservableValue<? extends Duration> observableValue, Duration duration, Duration t1) {  
 currTime.setText(getTimeString(t1.toMillis()));  
 }  
 });  
 }  
  
 public static String getTimeString(double millis) {  
 String s = formatTime((millis / 1000) % 60);  
 String m = formatTime((millis / 1000) / 60);  
 return m + ":" + s;  
 }  
  
 public static String formatTime(double time) {  
 int t = (int)time;  
 if (t > 9) { return String.valueOf(t); }  
 return "0" + t;  
 }  
  
 public void beginTimer(){  
 timer = new Timer();  
 timerTask = new TimerTask() {  
 @Override  
 public void run() {  
 double curr = mediaPlayer.getCurrentTime().toSeconds();  
 double end = media.getDuration().toSeconds();  
 timeSlider.setValue(curr);  
 if(curr / end == 1) cancelTimer();;  
 }  
 };  
 timer.scheduleAtFixedRate(timerTask, 1000, 1000);  
 }  
  
 public void cancelTimer(){  
 timer.cancel();  
 }  
  
 public void playMusic(MouseEvent actionEvent) {  
 if(play){  
 stopTrack.setVisible(false);  
 mediaPlayer.pause();  
 play = false;  
 }  
 else{  
 stopTrack.setVisible(true);  
 mediaPlayer.play();  
 play = true;  
 }  
 }  
  
 public void prevMusic(MouseEvent actionEvent) {  
 cancelTimer();  
 if(currentTrackIndex > 0){  
 currentTrackIndex--;  
 }else{  
  
 currentTrackIndex = userPlayList.getItems().size() - 1;  
 }  
 userPlayList.getSelectionModel().select(currentTrackIndex);  
 playSong(userPlayList.getItems().get(currentTrackIndex));  
 }  
  
 public void nextMusic(MouseEvent actionEvent) {  
 cancelTimer();  
 if(currentTrackIndex < userPlayList.getItems().size() - 1){  
 currentTrackIndex++;  
 }else{  
 currentTrackIndex = 0;  
 }  
 userPlayList.getSelectionModel().select(currentTrackIndex);  
 playSong(userPlayList.getItems().get(currentTrackIndex));  
 }  
  
 public void setMain(ActionEvent actionEvent){  
 mainLabel.setText("Главная");  
 searchMenu.setVisible(false);  
 userCollection.setVisible(false);  
 }  
  
 public void setSearch(ActionEvent actionEvent) {  
 mainLabel.setText("Поиск");  
 searchMenu.setVisible(true);  
 userCollection.setVisible(false);  
 }  
  
 public void setCollect(ActionEvent actionEvent) {  
 mainLabel.setText("Ваша коллекция");  
 searchMenu.setVisible(false);  
 userCollection.setVisible(true);  
 }  
  
 public void SetVolume(MouseEvent mouseEvent) {  
 volumeSlider.setValue(0);  
 volume2.setVisible(true);  
 }  
  
 public void AddVolume(MouseEvent mouseEvent) {  
 volumeSlider.setValue(50);  
 volume2.setVisible(false);  
 }  
  
 public ArrayList<Music> enterArray(ArrayList<String> b){  
 ArrayList<Music> temp = new ArrayList<Music>();  
 for(int i = 0; i < b.size(); i++){  
 temp.add(i, new Music(b.get(i)));  
 }  
 return temp;  
 }  
  
 public void goToLogin(ActionEvent actionEvent) throws IOException {  
 mediaPlayer.stop();  
 Parent root = FXMLLoader.load(getClass().getResource("Login.fxml"));  
 stage = (Stage) ((Node) actionEvent.getSource()).getScene().getWindow();  
 scene = new Scene(root);  
 stage.setScene(scene);  
 stage.show();  
 }  
}

Client.java

package com.example.alfa;  
  
import java.io.IOException;  
import java.net.\*;  
import java.util.Arrays;  
  
public class Client {  
 public final static int SERVER\_PORT = 5000;  
 DatagramSocket clientSocket;  
 InetAddress IPAddress;  
 byte[] receivingDataBuffer = new byte[128];  
 byte[] sendingDataBuffer = new byte[128];  
  
 public Client() throws SocketException, UnknownHostException {  
 clientSocket = new DatagramSocket();  
 IPAddress = InetAddress.getByName("localhost");  
 }  
  
 public void SendMessage(int param, String outMessage) throws IOException {  
 String res = new String(outMessage!="" ? String.valueOf(param) + "/" + outMessage :String.valueOf(param) + "/" + "0");  
 sendingDataBuffer = res.getBytes();  
 DatagramPacket sendingPacket = new DatagramPacket(sendingDataBuffer,sendingDataBuffer.length, IPAddress, SERVER\_PORT);  
 clientSocket.send(sendingPacket);  
 Arrays.fill(sendingDataBuffer, (byte) 0);  
  
 }  
  
 public void SendMessage(String outMessage) throws IOException {  
 String res = outMessage;  
 sendingDataBuffer = res.getBytes();  
 DatagramPacket sendingPacket = new DatagramPacket(sendingDataBuffer,sendingDataBuffer.length, IPAddress, SERVER\_PORT);  
 clientSocket.send(sendingPacket);  
 Arrays.fill(sendingDataBuffer, (byte) 0);  
 }  
  
 public DatagramPacket ReceiveMessage() throws IOException {  
 DatagramPacket inputPacket = new DatagramPacket(receivingDataBuffer, receivingDataBuffer.length);  
 clientSocket.receive(inputPacket);  
 return inputPacket;  
 }  
  
 public byte[] ReceiveMes() throws IOException {  
 DatagramPacket inputPacket = new DatagramPacket(receivingDataBuffer, receivingDataBuffer.length);  
 clientSocket.receive(inputPacket);  
  
 int len=0;  
 for (int i = 0; i < 4; ++i) {  
 len |= (receivingDataBuffer[3-i] & 0xff) << (i << 3);  
 }  
  
 Arrays.fill(receivingDataBuffer, (byte) 0);  
 receivingDataBuffer = new byte [len];  
  
 inputPacket = new DatagramPacket(receivingDataBuffer, len);  
 clientSocket.receive(inputPacket);  
 return receivingDataBuffer;  
 }  
}

Приложение Б «Листинг серверной части приложения»

Server. java

import java.io.ByteArrayOutputStream;  
import java.io.IOException;  
import java.io.ObjectOutputStream;  
import java.net.DatagramPacket;  
import java.net.DatagramSocket;  
import java.net.InetAddress;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.Arrays;  
  
public class Server {  
  
 public final static int SERVER\_PORT = 5000;  
  
 public static void main(String[] args) throws SQLException, ClassNotFoundException, IOException {  
 ConnectDB v = new ConnectDB();  
 ArrayList <String> searchMusicArrayList = new ArrayList<String>();  
 ArrayList <String> userPlayList = new ArrayList<String>();  
 DatagramSocket serverSocket = new DatagramSocket(SERVER\_PORT);  
 System.out.println("Waiting for a client to connect...");  
 byte[] receivingDataBuffer = new byte[256];  
 byte[] sendingDataBuffer = new byte[256];  
 String newMsg = "";  
 User user = new User();  
 int param = 0;  
 String[] parseMsg;  
 boolean bAnswer;  
  
 DatagramPacket inputPacket = new DatagramPacket(receivingDataBuffer, receivingDataBuffer.length);  
 serverSocket.receive(inputPacket);  
  
 String receivedData = new String(inputPacket.getData());  
 System.out.println("Sent from the client: " + receivedData.trim());  
  
 Arrays.fill(receivingDataBuffer, (byte) 0);  
  
 InetAddress senderAddress;  
 int senderPort;  
 DatagramPacket outputPacket;  
  
 while(true){  
 serverSocket.receive(inputPacket);  
 newMsg = new String(inputPacket.getData()).trim();  
 Arrays.fill(receivingDataBuffer, (byte) 0);  
 parseMsg = (newMsg.split("/"));  
 param = Integer.parseInt(parseMsg[0]);  
 System.out.println(parseMsg[0] + parseMsg[1]);  
 switch(param){  
 case 1:  
 user = new User(parseMsg[1], parseMsg[2]);  
 bAnswer = user.CompareUser();  
 sendingDataBuffer = String.valueOf(bAnswer).getBytes();  
 senderAddress = inputPacket.getAddress();  
 senderPort = inputPacket.getPort();  
 outputPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, senderAddress,senderPort);  
 serverSocket.send(outputPacket);  
 System.out.println(bAnswer);  
 break;  
 case 2:  
 user = new User(parseMsg[1], parseMsg[2], parseMsg[3]);  
 bAnswer = user.RegistrationUser();  
 sendingDataBuffer = String.valueOf(bAnswer).getBytes();  
 senderAddress = inputPacket.getAddress();  
 senderPort = inputPacket.getPort();  
 outputPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, senderAddress,senderPort);  
 serverSocket.send(outputPacket);  
 System.out.println(bAnswer);  
 break;  
 case 3:  
 System.out.println(parseMsg[1]);  
 searchMusicArrayList = v.searchMusicList(parseMsg[1]);  
  
 ByteArrayOutputStream baos = new ByteArrayOutputStream();  
 ObjectOutputStream oos = new ObjectOutputStream(baos);  
 oos.writeObject(searchMusicArrayList);  
 oos.flush();  
 sendingDataBuffer = baos.toByteArray();  
  
 int len = sendingDataBuffer.length;  
 byte[] data = new byte[4];  
  
 for (int i = 0; i < 4; ++i) {  
 int shift = i << 3; // i \* 8  
 data[3-i] = (byte)((len & (0xff << shift)) >>> shift);  
 }  
  
 senderAddress = inputPacket.getAddress();  
 senderPort = inputPacket.getPort();  
  
 outputPacket = new DatagramPacket(data, data.length, senderAddress, senderPort);  
 serverSocket.send(outputPacket);  
  
 senderAddress = inputPacket.getAddress();  
 senderPort = inputPacket.getPort();  
  
 outputPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, senderAddress, senderPort);  
 serverSocket.send(outputPacket);  
 break;  
 case 4:  
 v.InsertMusicInPlaylistDB(user.name, parseMsg[1]);  
 break;  
 case 5:  
 userPlayList = v.getUserPlayList(user.name);  
  
 baos = new ByteArrayOutputStream();  
 oos = new ObjectOutputStream(baos);  
 oos.writeObject(userPlayList);  
 oos.flush();  
 sendingDataBuffer = baos.toByteArray();  
  
 len = sendingDataBuffer.length;  
 data = new byte[4];  
  
 for (int i = 0; i < 4; ++i) {  
 int shift = i << 3; // i \* 8  
 data[3-i] = (byte)((len & (0xff << shift)) >>> shift);  
 }  
  
 senderAddress = inputPacket.getAddress();  
 senderPort = inputPacket.getPort();  
  
 outputPacket = new DatagramPacket(data, data.length, senderAddress, senderPort);  
 serverSocket.send(outputPacket);  
  
 senderAddress = inputPacket.getAddress();  
 senderPort = inputPacket.getPort();  
  
 outputPacket = new DatagramPacket(sendingDataBuffer, sendingDataBuffer.length, senderAddress, senderPort);  
 serverSocket.send(outputPacket);  
 default:  
 }  
 }  
 }  
}

ConnectDB.java

import java.sql.\*;  
import java.util.ArrayList;  
  
public class ConnectDB extends Config {  
 Connection dbConnect;  
  
 public Connection getDbConnect() throws ClassNotFoundException, SQLException {  
 String ConnectionStr = "jdbc:mysql://" + dbHost + ":" + dbPort + "/" + dbName;  
 Class.forName("com.mysql.cj.jdbc.Driver");  
 dbConnect = DriverManager.getConnection(ConnectionStr, dbUser, dbPassword);  
 return dbConnect;  
 }  
  
 public void InsertUser(String userName, String userMail, String userPassword) {  
 String InsertAt = "INSERT INTO " + ConstValues.USER\_TABLE +  
 "(" + ConstValues.USER\_NAME + "," + ConstValues.USER\_MAIl + "," + ConstValues.USER\_PASSWORD + ")" + "VALUES(?,?,?)";  
 try {  
 PreparedStatement preparedStatement = getDbConnect().prepareStatement(InsertAt);  
 preparedStatement.setString(1, userName);  
 preparedStatement.setString(2, userMail);  
 preparedStatement.setString(3, userPassword);  
 preparedStatement.execute();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public ResultSet SelectUser(String userName) {  
 ResultSet res = null;  
 String select = "SELECT \* FROM " + ConstValues.USER\_TABLE + " WHERE " + ConstValues.USER\_NAME + "=?";  
 try {  
 PreparedStatement preparedStatement = getDbConnect().prepareStatement(select);  
 preparedStatement.setString(1, userName);  
 res = preparedStatement.executeQuery();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return res;  
 }  
  
 public ArrayList<String> searchMusicList(String s) {  
 ArrayList<String> paths = new ArrayList<String>();  
 if (s.equals("")) {  
 return paths;  
 }  
 String select = "SELECT \* FROM musicservicedatabase.song WHERE name LIKE ?";  
 try {  
 PreparedStatement preparedStatement = getDbConnect().prepareStatement(select);  
 preparedStatement.setString(1, "%" + s + "%");  
 ResultSet rs = preparedStatement.executeQuery();  
 while (rs.next()) {  
 paths.add(rs.getNString(3));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return paths;  
 }  
  
 public void InsertMusicInPlaylistDB(String name, String musicName){  
 String InsertAt = "INSERT INTO musicservicedatabase.playlist (song, username) VALUES(?,?)";  
 try {  
 PreparedStatement preparedStatement = getDbConnect().prepareStatement(InsertAt);  
 preparedStatement.setString(1, musicName);  
 preparedStatement.setString(2, name);  
 preparedStatement.execute();  
 } catch (SQLException | ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public ArrayList<String> getUserPlayList(String name){  
 ArrayList<String> musicList = new ArrayList<String>();  
 String select = "SELECT \* FROM musicservicedatabase.playlist WHERE username =?";  
 try {  
 PreparedStatement preparedStatement = getDbConnect().prepareStatement(select);  
 preparedStatement.setString(1, name);  
 ResultSet rs = preparedStatement.executeQuery();  
 while (rs.next()) {  
 musicList.add(rs.getNString(2));  
 }  
 } catch (SQLException e) {  
 e.printStackTrace();  
 } catch (ClassNotFoundException e) {  
 e.printStackTrace();  
 }  
 return musicList;  
 }  
}

Приложение В «UML диаграммы»

Диаграмма вариантов использования

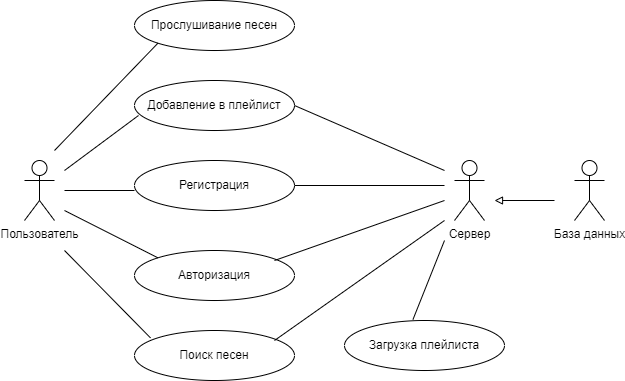


Рисунок В1 – UML диаграмма вариантов использования.

Диаграммы классов

Изображение выглядит как стол

Автоматически созданное описание

Рисунок В2 – UML диаграмма классов Клиента*.*

Изображение выглядит как текст

Автоматически созданное описание

Рисунок В3 – UML диаграмма классов Сервера.

Диаграмма деятельности

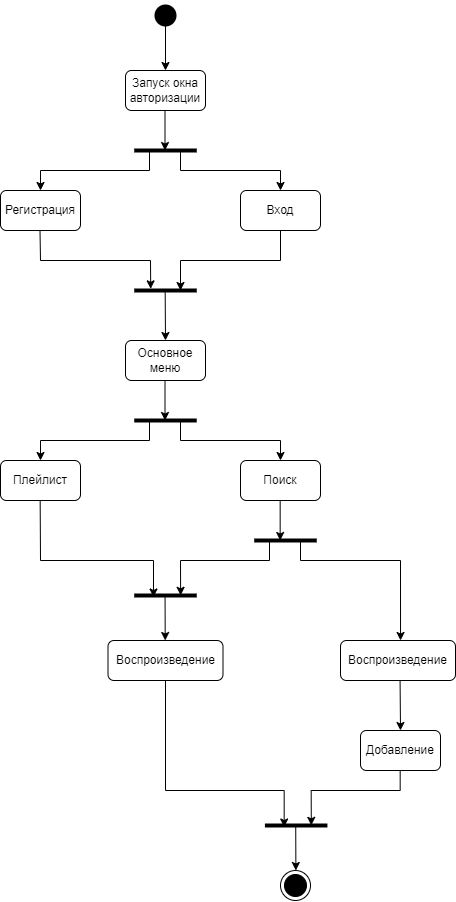


Рисунок В4 – UML диаграмма деятельности.

Диаграмма развёртывания

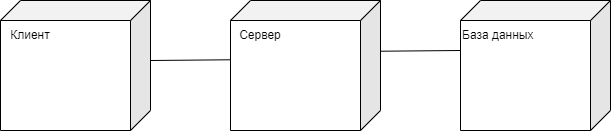


Рисунок В5 – UML диаграмма развертывания.

Диаграмма последовательности

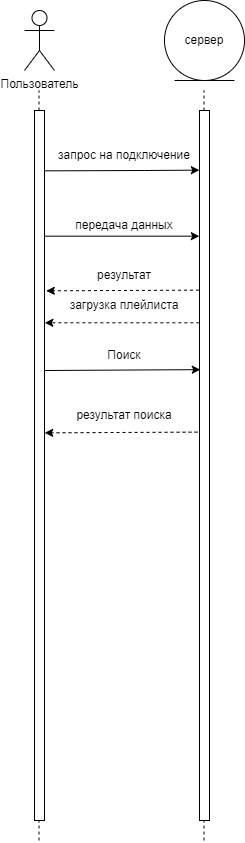


Рисунок В6 – UML диаграмма последовательности.