

Министерство науки и высшего образования Российской Федерации
Пензенский государственный университет
Кафедра «Вычислительной техники»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по курсу «Разработка кроссплатформенных приложений»
на тему: «Разработка многомодульного приложения на языке Java»

Выполнил:

Студенты группы 23BBB2

Стрельцов А.П.

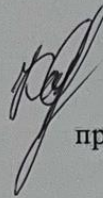
Приняла:

Юрова О.В.

25.05.26
отлично
ф.и.о.

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет Вычислительной техники

Кафедра "Вычислительная техника"



"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

профессор М. А. Митрохин

«___» _____ 20__ г

ЗАДАНИЕ

на курсовое проектирование по курсу

Разработка кроссплатформенных приложений
Студенту Стрелькову Александру Павловичу Группа 23ВВ42
Тема проекта MP-3-проигрыватель
Разработка многомодульного приложения на языке Java

Исходные данные (технические требования) на проектирование

Язык программирования Java, среда разработки
NetBeans
Приложение должно обладать графическим интер-
фейсом и использовать следующие технологии:
1. Java Collections Framework.
2. Механизм обработки исключительных ситуаций
3. Java Stream API
4. Java Multithreading
5. Сетевое взаимодействие

Объем работы по курсу

1. Расчетная часть

2. Графическая часть

Диаграммы языка UML:

1) диаграмма классов	5) диаграмма вариантов использования
2) диаграмма деятельности	
3) диаграмма развертывания	
4) диаграмма последовательности	

3. Экспериментальная часть

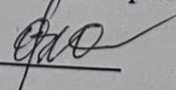
Тестирование многофункционального приложения

Срок выполнения проекта по разделам

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8

Дата выдачи задания "19" 02. 2026

Дата защиты проекта " " "

Руководитель Нурова С.В. 

Задание получил "19" 02

Студент Стрельцов С.П.  2026г.

Содержание

Содержание	2
Введение.....	5
1. Постановка задачи	7
2. Выбор решения.....	8
3. Описание программы.....	12
3.1. Язык и среда разработки	12
3.2. Интерфейс приложения	12
3.3. Необходимые входные данные.....	14
3.4. Работа приложения	14
4.Описание способа организации пользовательского интерфейса	16
5.Описание результатов работы программы	18
5.1. Руководство пользователя	18
5.1.1. Запуск и первичное взаимодействие	18
5.1.2. Управление сетевым подключением и работа с серверной медиатекой	19
5.1.3. Администрирование сервера.....	19
5.1.4. Управление локальной медиатекой и плейлистом	20
5.1.5. Воспроизведение и навигация. Режимы воспроизведения.	20
5.1.6. Работа с радио.....	21
5.2. Ожидаемые результаты работы программы	23
Заключение	25
Список Литературы.....	26
Приложение А. Листинг программы.....	27
Приложение Б. UML-диаграммы.....	66

Введение

Современные технологии разработки программного обеспечения уделяют особое внимание кроссплатформенности и сетевому взаимодействию, позволяя создавать приложения, работающие на различных операционных системах и эффективно использующие распределённые ресурсы. Язык программирования Java, реализующий принцип «написано однажды — работает везде», предоставляет разработчикам универсальную среду для создания надёжных и масштабируемых решений. В условиях растущего разнообразия пользовательских устройств и развития облачных мультимедийных сервисов разработка сетевого кроссплатформенного аудиопроигрывателя приобретает высокую практическую значимость.

Аудиопроигрыватели стали неотъемлемой частью повседневной цифровой экосистемы, используясь как в личных, так и в профессиональных целях. Формат MP3, благодаря эффективному алгоритму сжатия аудиоданных при сохранении приемлемого качества звучания, остаётся одним из самых распространённых стандартов хранения и передачи музыки. Графический интерфейс современного плеера должен быть интуитивно понятным, отзывчивым и адаптивным, что требует применения современных UI-фреймворков и грамотной клиент-серверной архитектуры.

Кроссплатформенная разработка на базе Java активно опирается на специализированные инструменты и системы сборки. JavaFX представляет собой современную платформу для создания богатых клиентских и серверных приложений, поддерживающую работу с мультимедиа, анимацией и стилизацией элементов управления. Система управления проектами Maven автоматизирует процессы компиляции, тестирования и упаковки приложения, обеспечивая удобное управление внешними библиотеками и зависимостями. Дополнительно в проекте задействованы стандартные API сетевого взаимодействия (java.net) и многопоточности (java.util.concurrent) для асинхронной обработки запросов и неблокирующего обмена данными.

Использование связки Java, JavaFX и Maven позволяет не только

гарантировать кроссплатформенность готового продукта, но и применять современные принципы объектно-ориентированного проектирования, модульной архитектуры и многопоточной обработки данных. Реализация клиент-серверной модели в учебном проекте способствует закреплению навыков работы с мультимедийными API, организации сетевых протоколов передачи файлов, обработке событий пользовательского интерфейса и оптимизации потребления системных ресурсов.

Для разработки приложения была выбрана интегрированная среда разработки Apache NetBeans, а в качестве основы проекта использован тип Java with Maven с архетипом `simple javafx maven archetype`. Программирование ведётся на языке Java с применением фреймворка JavaFX для реализации графического интерфейса клиента, панели управления сервером и воспроизведения аудио. Взаимодействие между клиентом и сервером организовано через пользовательский бинарный протокол на базе TCP-сокетов.

Целью данной курсовой работы является разработка кроссплатформенного MP3-проигрывателя с графическим пользовательским интерфейсом, поддерживающего базовые функции воспроизведения аудиозаписей, управления списком воспроизведения, настройки параметров звучания, а также реализацию клиент-серверного взаимодействия для удалённого хранения, загрузки и управления музыкальными файлами.

1. Постановка задачи

Необходимо разработать многомодульное клиент-серверное приложение на языке программирования Java в среде разработки Apache NetBeans. Приложение должно представлять собой сетевой MP3-проигрыватель с интуитивно понятным графическим интерфейсом для клиентской части и панелью администрирования для серверной части.

При разработке необходимо использовать следующие технологии:

1. Java Collections Framework
2. Механизм обработки исключительных ситуаций
3. Java Stream API
4. Java Multithreading
5. Сетевое взаимодействие.

Полный перечень возможных вариантов использования программы показан на диаграмме рис.10.

2. Выбор решения

Для реализации многомодульного клиент-серверного приложения используется среда разработки Apache NetBeans, а в качестве основы проекта использован тип Java with Maven с архетипом Simple JavaFX Maven Archetype. Особенности выбора данного архетипа обусловлены возможностью автоматически сгенерировать корректную структуру проекта, настроить pom.xml для подключения необходимых модулей JavaFX и исключить ручное управление внешними JAR-библиотеками. Архитектура приложения расширена до распределённой клиент-серверной модели, что потребовало внедрения сетевого взаимодействия на базе TCP-сокетов, многопоточной обработки подключений, кастомного бинарного протокола обмена данными и асинхронного обновления интерфейса.

Для обеспечения чистоты кода и удобства приложение разделено на логические программные модули, реализующие принцип разделения ответственности:

1) Модуль пользовательского интерфейса

Классы: UIComponents.java, DiscAnimation.java

Содержит всю логику построения графического интерфейса на JavaFX. Отвечает за создание и компоновку визуальных элементов: панели управления, списка плейлиста, переключателей режимов, ползунков громкости и прогресса, анимированного диска с обложкой. Обрабатывает пользовательские события: клики по кнопкам, выбор треков, поиск, переименование, сохранение плейлиста. Реализует адаптивное масштабирование интерфейса при изменении размера окна, динамическое переключение между режимами «Плейлист» и «Радио», визуальную подсветку активных элементов и кэширование обложек для оптимизации отображения.

2) Модуль медиа-движка

Классы: AudioPlayer.java

Инкапсулирует работу с классами javafx.scene.media.Media и MediaPlayer. Обеспечивает инициализацию аудиоисточников как из локальных MP3-файлов,

так и из интернет-радиопотоков. Управляет циклом воспроизведения: запуск, пауза, возобновление, остановка, перемотка. Реализует подписку на события медиаплеера: окончание трека, ошибки воспроизведения, готовность к работе. Поддерживает разделение логики для файловых треков и потокового радио (перемотка недоступна для радио). Предоставляет методы установки громкости с защитой от выхода за допустимые границы.

3) Модуль управления плейлистом

Классы: `PlaylistManager.java`, `RadioStation.java`

Отвечает за хранение и манипуляцию коллекцией треков: добавление, удаление, переименование, получение путей к файлам. Поддерживает два параллельных списка — отображаемые имена и оригинальные пути — для гибкого управления данными. Реализует логику режимов воспроизведения: последовательный, циклический (повтор трека), случайный (перемешивание) с взаимным исключением режимов. Предоставляет методы получения индексов следующей и предыдущей песни с учётом активного режима. Класс `RadioStation` служит моделью данных для радиостанций, храня название, URL потока и заглушку для иконки.

4) Модуль работы с файловой системой

Классы: `FileUtils.java`

Содержит вспомогательные методы для взаимодействия с файловой системой. Отвечает за открытие диалогов выбора файлов и папок, фильтрацию по расширению `.mp3`, обработку дубликатов при добавлении треков (предлагает пользователю: отменить, переименовать копию или заменить существующую запись). Реализует безопасное копирование файлов при сохранении плейлиста в новую папку с созданием директории при необходимости. Все операции сопровождаются обработкой исключений и информированием пользователя через диалоговые окна.

5) Модуль визуализации и анимации

Классы: `DiscAnimation.java`

Отвечает за графическое отображение вращающегося диска с обложкой

альбома или иконкой радиостанции. Реализует алгоритм обрезки изображения по кругу для создания эффекта виниловой пластинки, добавление декоративных элементов: центрального отверстия, рамки, бликов. Управляет анимацией вращения через RotateTransition, предоставляет методы загрузки обложек из ID3-тегов MP3-файлов с использованием библиотеки mp3agic. При отсутствии обложки или ошибке чтения автоматически подставляет изображение диска по умолчанию в стиле приложения.

6) Модуль сетевого клиента

Классы: NetworkClient.java

Реализует клиентскую часть сетевого взаимодействия. Отвечает за установку и поддержание TCP-соединения с сервером, отправку команд в формате пользовательского бинарного протокола (тип-длина-полезная нагрузка), асинхронную загрузку списков файлов, скачивание аудиофайлов в локальный кэш (./cache) и загрузку локальных треков на сервер. Использует выделенный фоновый поток (listenLoop) для непрерывного прослушивания ответов сервера и делегирует обновление UI через callback-интерфейсы (Consumer), обеспечивая неблокирующую работу приложения и корректную обработку разрывов соединения.

Модуль сервера

Классы: Mp3Server.java

Представляет собой самостоятельное JavaFX-приложение для администрирования медиатеки. Обеспечивает приём входящих подключений через ServerSocket, обработку запросов в пуле потоков (ExecutorService), хранение файлов в директории ./server_music, ведение журнала сетевых событий и управление подключенными клиентами. Поддерживает команды: просмотр списка, скачивание, удаление и загрузку файлов. Графическая панель сервера отображает логи в реальном времени, список активных клиентов и файлов, а также предоставляет инструменты локального управления медиатекой (загрузка/удаление через диалоги).

8) Модуль конфигурации и точки входа

Классы: App.java

Содержит точку входа в клиентское приложение и логику первоначальной инициализации. Отвечает за создание экземпляров менеджера плейлиста, аудиоплеера, сетевого клиента и пользовательского интерфейса, их связывание через зависимости, настройку сцены и стадии JavaFX. Применяет цветовую тему ко всему интерфейсу, запускает механизм масштабирования, отображает главное окно. Гарантирует правильный порядок инициализации. После запуска автоматически пытается установить подключение к локальному серверу в фоновом потоке, обеспечивая прозрачную работу в сетевом режиме.

Полный функционал классов изображен на рис.8.

3. Описание программы

Программа представляет собой распределённое клиент-серверное приложение для воспроизведения и управления аудиозаписями в формате MP3. Система состоит из двух независимых компонентов: клиентского плеера (App.java) и сервера медиатеки (Mp3Server.java), взаимодействующих через TCP-сокеты с использованием пользовательского бинарного протокола обмена данными.

3.1. Язык и среда разработки

Программа разработана на языке программирования Java с использованием графического фреймворка JavaFX. В качестве среды разработки и управления зависимостями используются Apache NetBeans и Apache Maven (архетип simple-javafx-maven). Для работы с метаданными MP3-файлов (ID3-теги и обложки альбомов) применяется сторонняя библиотека mp3agic.

Сетевое взаимодействие реализовано через стандартные API java.net (классы Socket, ServerSocket, DataInputStream, DataOutputStream), а многопоточная обработка подключений и фоновых задач организована с помощью java.util.concurrent (ExecutorService). Архитектура приложения построена по принципам модульного разделения ответственности, что обеспечивает кроссплатформенность и стабильную работу в операционных системах, поддерживающих Java Runtime Environment версии 11 и выше.

3.2. Интерфейс приложения

Графический интерфейс системы реализован в виде двух независимых модулей: клиентского приложения и серверной панели администрирования. Каждый модуль обладает собственной структурой, оптимизированной под выполняемые задачи, и выполнен в едином визуальном стиле.

Клиентский интерфейс построен по принципу двухпанельной горизонтальной компоновки с использованием контейнера HBox.

Левая панель предназначена для навигации и управления контентом. В её верхней части расположен переключатель режимов работы («Плейлист» и

«Радио») и текстовое поле для фильтрации списка воспроизведения. Основную область занимает перечень треков или радиостанций в зависимости от выбранного режима. Ниже размещена группа кнопок для работы с локальными файлами (добавление, удаление, переименование, сохранение плейлиста). В нижней части панели выделен раздел сетевого взаимодействия, содержащий инструменты подключения к удалённому серверу, управления сетевым списком файлов и их загрузки в локальный кэш.

Правая панель обеспечивает визуализацию процесса воспроизведения и контроль параметров аудио. В верхней части отображается анимированный графический элемент (вращающийся диск с обложкой альбома или иконкой радиостанции). Ниже расположены информационная метка с названием текущего трека, таймлайн с индикаторами времени, основная группа кнопок управления воспроизведением, переключатели режимов последовательности и регулятор громкости с цифровым отображением уровня.

Серверный интерфейс реализован в отдельном окне приложения и имеет трёхзональную вертикальную структуру. Верхняя зона содержит элементы управления состоянием сервера (запуск/остановка прослушивания порта), текстовую индикацию текущего статуса и номер используемого порта. Центральная зона разделена на два блока: перечень активных клиентских подключений и список аудиофайлов, находящихся в директории медиатеки. Рядом со списком файлов расположены кнопки для локальной загрузки новых треков и удаления существующих. Нижняя зона отведена под многострочное текстовое поле, предназначенное для вывода журнала сетевых событий и системных уведомлений в реальном времени.

Оба интерфейса формируются программными средствами JavaFX без применения FXML-разметки, что обеспечивает полный контроль над жизненным циклом компонентов и динамическое обновление состояния. Визуальное оформление унифицировано: применяется тёмная цветовая схема с фиолетовыми акцентами, стилизация элементов осуществляется через inline-CSS

свойства. Реализован механизм пропорционального масштабирования, позволяющий адаптировать размеры шрифтов, отступов и графических элементов под разрешение окна без нарушения композиции.

3.3. Необходимые входные данные

Для полноценной работы приложения требуются следующие данные и ресурсы:

- Локальные медиафайлы: MP3-аудиозаписи для добавления в локальный плейлист. Файлы должны содержать корректные ID3-теги для извлечения обложек.
- Сетевая конфигурация: IP-адрес сервера (по умолчанию 127.0.0.1) и порт (8080). Сервер использует директорию `./server_music` в качестве общего хранилища, а клиент — папку `./cache` для сохранения загруженных по сети треков.
- Протокол обмена: Клиент и сервер обмениваются сообщениями в формате «тип команды — длина полезных данных — полезная нагрузка». Типы данных включают текстовые команды (LIST, DOWNLOAD, UPLOAD, DELETE), бинарные потоки аудио и статусные ответы.
- Радиопотоки: Встроенные URL-адреса интернет-радиостанций, прописанные в коде клиента.

Приложение не требует настройки внешних баз данных. Все метаданные и состояния хранятся в оперативной памяти, а аудиофайлы обрабатываются напрямую из файловой системы или сетевых потоков.

3.4. Работа приложения

При запуске клиент автоматически пытается установить фоновое TCP-соединение с сервером на localhost. В случае успеха запрашивается список доступных файлов, который отображается в сетевой панели. Пользователь может выбрать трек из списка сервера и инициировать его загрузку в локальный кэш;

по завершении скачивания файл автоматически добавляется в плейлист и готов к воспроизведению.

Серверная часть запускается отдельно, после чего администратор нажимает «Запустить сервер». Приложение открывает порт, создаёт пул потоков для обработки подключений и начинает прослушивание входящих запросов. В журнале фиксируются все события: подключения клиентов, запросы списков, передачи файлов, ошибки протокола. Администратор может вручную управлять медиатекой через диалоги выбора файлов.

Воспроизведение аудио поддерживает локальные файлы, загруженные из кэша треки и интернет-радио. Переключение между режимами происходит мгновенно: при выборе радио скрываются элементы плейлиста и таймлайна, активируется сетевой поток. Все сетевые операции (прослушивание сокетов, чтение/запись файлов, сериализация протокола) выполняются в выделенных рабочих потоках, что гарантирует отсутствие блокировок графического интерфейса. Обновление UI происходит реактивно через Platform.runLater. При разрыве соединения или ошибке сети клиент корректно обрабатывает исключения, очищает временные ресурсы и предоставляет пользователю возможность повторного подключения.

4. Описание способа организации пользовательского интерфейса

Графический интерфейс приложения реализован на базе фреймворка JavaFX методом программной генерации компонентов без использования FXML-разметки. Корневым контейнером выступает `BorderPane`, центральная область которого занята горизонтальным блоком `HBox`, разделяющим окно на две независимые вертикальные панели `VBox`. Левая панель отвечает за навигацию, управление плейлистом и сетевое взаимодействие, правая — за визуализацию воспроизведения, таймлайн и регулировку параметров аудио. Такая модульная компоновка обеспечивает чёткое разделение функциональных зон и упрощает сопровождение кода.

Все элементы управления создаются динамически и группируются по назначению. Списки треков, радиостанций и серверных файлов реализованы через `ListView` с кастомными ячейками `ListCell`, содержащими `ImageView` и `Label` для отображения обложек и названий. Визуальное оформление выполнено посредством `inline-CSS` с использованием единой тёмной цветовой палитры и акцентных оттенков. Стили кнопок и переключателей изменяются реактивно в зависимости от состояния (наведение, активность, отключённость), что гарантирует единообразие интерфейса без внешних таблиц стилей.

Интерфейс поддерживает пропорциональное масштабирование через слушатели свойств `widthProperty` и `heightProperty`. Коэффициент масштабирования ограничивается диапазоном от 0,8 до 1,5, после чего пересчитываются размеры шрифтов, отступов и графических элементов. Обработка пользовательских действий реализована через лямбда-выражения и привязки свойств JavaFX. Поле поиска связано с `FilteredList`, что позволяет фильтровать данные в реальном времени без модификации исходной коллекции. Таймлайн и регулятор громкости синхронизированы с медиаплеером через реактивные привязки, блокирующие автообновление во время ручного взаимодействия.

Управление режимами работы осуществляется без перезагрузки сцены. При

переключении между плейлистом, радио и сетевым списком применяются методы `setVisible()` и `setManaged()`, мгновенно изменяющие видимость и занимаемое пространство контейнеров. Состояния кнопок воспроизведения, анимация диска и текстовые метки обновляются асинхронно через `Platform.runLater()`, что предотвращает блокировку UI-потока при сетевых операциях или работе с файловой системой. Архитектура визуального слоя полностью изолирована от бизнес-логики, обеспечивая слабую связанность компонентов и высокую отзывчивость приложения.

5. Описание результатов работы программы

Графический интерфейс приложения реализован на основе библиотеки JavaFX методом программной генерации компонентов без использования FXML-разметки. Такой подход обеспечивает полный контроль над жизненным циклом элементов, упрощает динамическое изменение интерфейса и позволяет тесно интегрировать визуальный слой с бизнес-логикой. Организация интерфейса построена по принципу модульной контейнерной иерархии с чётким разделением функциональных зон.

5.1. Руководство пользователя

5.1.1. Запуск и первичное взаимодействие

При запуске клиентского приложения (App.java) автоматически инициируется фоновое подключение к локальному серверу по адресу 127.0.0.1 на порт 8080. В случае успешного соединения запрашивается список доступных файлов, после чего активируется сетевая панель в левой части интерфейса. Для работы распределённой системы необходимо предварительно запустить серверное приложение (Mp3Server.java) в отдельном процессе.

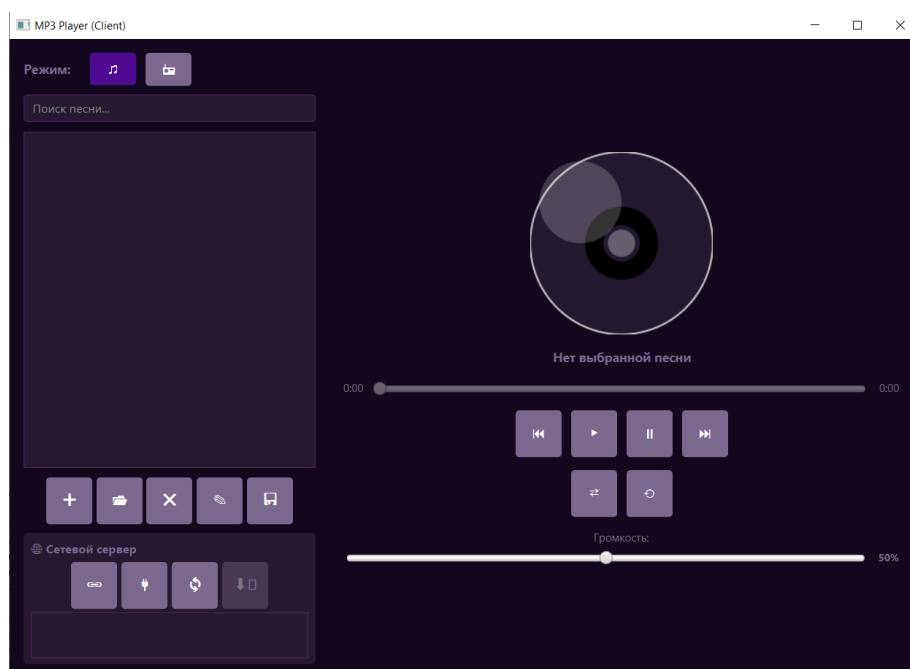


Рисунок 1 – Клиентское приложение. Режим “Плейлист”

5.1.2. Управление сетевым подключением и работа с серверной медиатекой

В нижней части левой панели расположен раздел *Сетевой сервер*. Кнопки *Подключиться* и *Отключиться* управляют состоянием TCP-сокета.

При активном соединении кнопки переключения в серверный режим обновляют своё состояние, а список удалённых файлов загружается автоматически. В случае разрыва связи интерфейс деактивирует сетевые элементы управления и возвращает систему в локальный режим без потери текущего плейлиста.

Список файлов, размещённых на сервере, отображается в отдельном ListView. Выбор записи активирует кнопку *Скачать в плейлист*.

При нажатии инициируется асинхронная загрузка аудиофайла в локальную директорию `./cache` по пользовательскому бинарному протоколу. По завершении передачи трек автоматически добавляется в основной плейлист и становится доступен для воспроизведения.

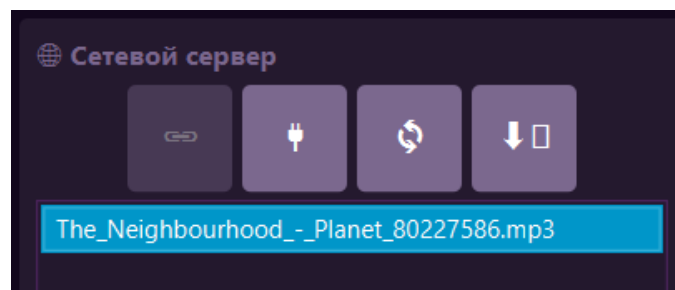


Рисунок 2 – Клиентское приложение. Панель “Сетевой сервер”.

5.1.3. Администрирование сервера

Серверное приложение предоставляет графическую панель управления. Кнопка *Запустить сервер* открывает порт и создаёт пул потоков для обработки подключений. Администратор может просматривать список активных клиентов, загружать новые MP3-файлы с локального диска или удалять существующие из директории `./server_music`. Все события фиксируются в журнале (*TextArea*) в реальном времени.

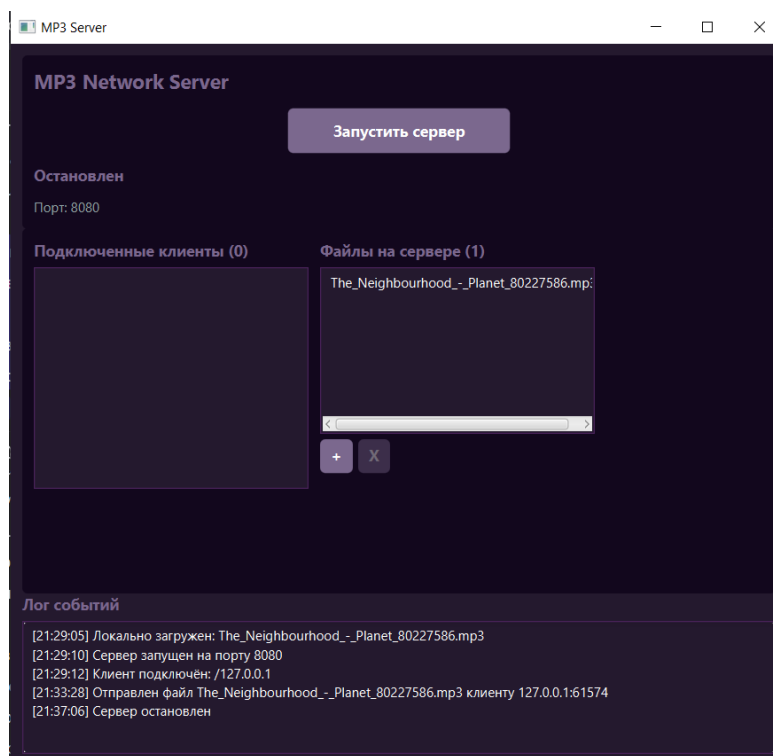


Рисунок 3 –Серверное приложение.

5.1.4. Управление локальной медиатекой и плейлистом

Для работы с локальной коллекцией аудиофайлов интерфейс предоставляет набор инструментов (выбрать файл, выбрать папку с музыкой, удалить из списка, изменить имя файла, сохранить плейлист в папку), обеспечивающих полный цикл управления содержимым плейлиста. Все операции выполняются синхронно, с предварительной проверкой допустимости действий и блокировкой в сетевом режиме.



Рисунок 4 – Клиентское приложение, инструменты управления локальной медиатекой и плейлистом.

5.1.5. Воспроизведение и навигация. Режимы воспроизведения.

Запуск трека осуществляется двойным кликом по элементу списка или нажатием кнопки *воспроизвести*. Управление циклом выполняется стандартными элементами: *предыдущая песня*, *пауза*, *возобновление*, *следующая*

песня. Перемотка выполняется перетаскиванием ползунка таймлайна. Громкость регулируется нижним слайдером с выводом текущего значения в процентах. При переключении между локальными файлами и сетевыми треками состояние плеера сбрасывается корректно, без прерывания работы приложения.

Кнопки *случайный порядок* и *повтор трека* взаимно исключают друг друга: активация одного режима автоматически сбрасывает другой, что отражается изменением цвета элементов.

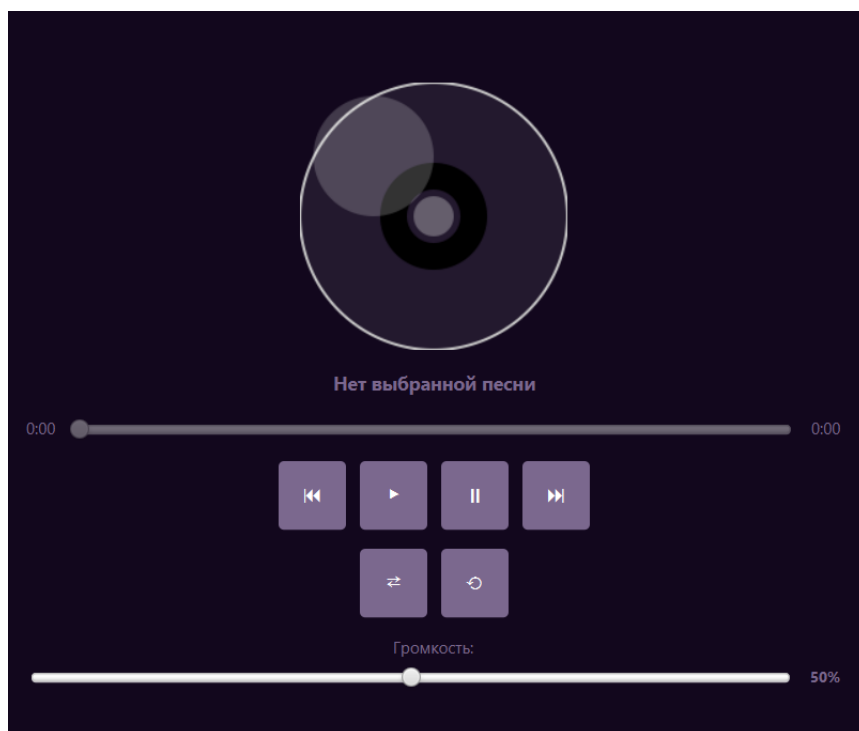


Рисунок 5 – Клиентское приложение. Панель воспроизведения и регулировки параметров аудио.

Ввод текста в поле *поиск песни* мгновенно фильтрует отображаемый список без изменения исходного порядка данных.



Рисунок 6 – Клиентское приложение. Поле поиска.

5.1.6. Работа с радио.

Для прослушивания потокового вещания интерфейс предоставляет выделенный режим «Радио». Переключение осуществляется нажатием соответствующей кнопки в заголовке левой панели. При активации данного режима скрываются элементы управления локальным плейлистом и таймлайном,

а в центральной области отображается предустановленный список интернет-радиостанций.

Выбор станции из списка автоматически инициирует сетевое подключение к заданному аудиопотоку и запускает воспроизведение. Графический элемент визуализации (вращающийся диск) заменяет обложку альбома на тематическую иконку радиостанции.

Навигация между доступными потоками выполняется стандартными кнопками перехода к предыдущему или следующему элементу. Управление циклом воспроизведения (старт, пауза, возобновление) доступно в полном объеме, однако функция перемотки по времени отключена в соответствии с природой потоковой передачи данных. Регулировка громкости и визуальная анимация остаются активными и работают синхронно с основным интерфейсом плеера.

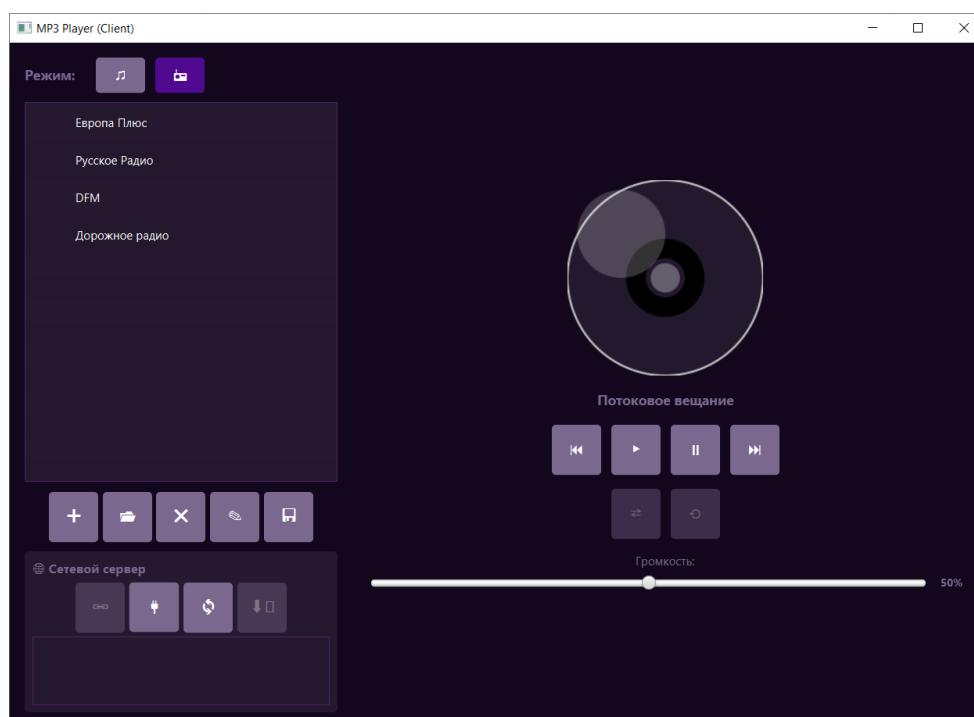


Рисунок 7 – Клиентское приложение. Режим “Радио”.

5.2. Ожидаемые результаты работы программы

Ниже представлен перечень ключевых сценариев взаимодействия пользователя с приложением и соответствующая реакция системы.

Таблица 1 – Ожидаемые результаты работы программы

Действие пользователя	Ожидаемый результат
Запуск клиентского приложения	Программа автоматически пытается установить связь с локальным сервером. При успешном подключении активируется панель сетевого взаимодействия и загружается список доступных аудиофайлов.
Подключение к серверу	Устанавливается сетевое соединение, интерфейс разблокирует инструменты работы с удалённой медиатекой, фоновый процесс запрашивает и отображает актуальный перечень файлов.
Скачивание трека из сетевого списка	Выбранная аудиозапись передаётся по сети, сохраняется в локальную папку кэша, после завершения загрузки автоматически добавляется в основной плейлист и становится доступна для воспроизведения.
Добавление локальных файлов или папки	Открывается системное окно выбора данных. Программа проверяет файлы на наличие дубликатов, отбирает только аудиоформаты и добавляет их в конец списка с автоматической загрузкой обложек.
Запуск воспроизведения композиции	Аудиопоток активируется, графический элемент визуализации начинает вращение, на экране отображаются название трека, текущее время звучания и общая длительность.

Продолжение таблицы 1

Действие пользователя	Ожидаемый результат
Переключение в режим интернет-радио	Элементы локального плейлиста скрываются, вместо них отображается список радиостанций. Выбор станции мгновенно запускает потоковое вещание, функция перемотки времени деактивируется.
Запуск сервера и загрузка музыки	Серверная часть открывает порт для входящих подключений, выбранный с компьютера файл копируется в общую директорию хранения и сразу становится видимым в списке доступных треков для всех подключённых клиентов.

Заключение

В ходе выполнения курсовой работы было разработано распределённое клиент-серверное приложение для воспроизведения и управления MP3-аудиофайлами. Система реализует полный цикл взаимодействия: от локального воспроизведения и работы с интернет-радио до удалённого хранения, загрузки и синхронизации медиатеки через пользовательский бинарный протокол поверх TCP-сокетов.

В процессе проектирования закреплены навыки построения модульной архитектуры на Java, применения многопоточности (ExecutorService) для асинхронной обработки сетевых подключений и передачи данных, а также программирования реактивного графического интерфейса на JavaFX без FXML-разметки. Реализованы механизмы локального кэширования загруженных файлов, корректной обработки сетевых разрывов и безопасного освобождения медиа-ресурсов. Чёткое разделение ответственности между клиентским интерфейсом, медиа-движком, сетевым модулем и сервером администрирования обеспечило стабильную работу приложения при одновременном обслуживании нескольких клиентов.

Созданная программа успешно выполняет заявленные функции, демонстрирует кроссплатформенность и отказоустойчивость в штатных и аварийных сетевых сценариях. Полученные результаты подтверждают эффективность применения современных подходов к десктопной и сетевой разработке. В перспективе система может быть дополнена механизмами авторизации пользователей, поддержкой дополнительных аудиоформатов, веб-панелью удалённого администрирования и интеграцией с облачными хранилищами медиаконтента.

Список Литературы

1. Шилдт, Г. Java: полное руководство / Г. Шилдт ; пер. с англ. - 11-е изд. - Москва : Вильямс, 2023. - 1360 с. : ил.
2. Хорстманн, К. С. Java. Библиотека профессионала. Том 1. Основы / К. С. Хорстманн. - 12-е изд. - Москва : Вильямс, 2022. - 1056 с. : ил.
3. Андерсон, К. JavaFX в действии: создание современных десктопных приложений / К. Андерсон ; пер. с англ. - Москва : ДМК Пресс, 2019. - 384 с. : ил.
4. Машнин, Т. Многопоточное программирование в Java / Т. Машнин. - Москва : Издательские решения (Ridero), 2024. - 209 с.
5. Дубаков, А. А. Сетевое программирование: клиент-серверные приложения на Java / А. А. Дубаков. - Санкт-Петербург : БХВ-Петербург, 2019. - 320 с. : ил.

Приложение А. Листинг программы

App.java

```
package com.mycompany.mp3player;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class App extends Application {
    private PlaylistManager playlistManager;
    private AudioPlayer audioPlayer;
    private UIComponents uiComponents;
    private NetworkClient networkClient;

    @Override
    public void start(Stage primaryStage) {
        playlistManager = new PlaylistManager();
        audioPlayer = new AudioPlayer(playlistManager);

        // Инициализация сетевого клиента
        networkClient = new NetworkClient();

        // Создаём UIComponents с сетевым клиентом
        uiComponents = new UIComponents(playlistManager, audioPlayer, primaryStage, networkClient);

        primaryStage.setTitle("MP3 Player (Client)");
        Scene scene = new Scene(uiComponents.getRootPane(), 1000, 700);
        scene.getRoot().setStyle("-fx-background-color: " + UIComponents.COLOR_BG_MAIN + ";");
        uiComponents.setupScaling(scene, primaryStage);
        primaryStage.setScene(scene);
        primaryStage.show();

        // Автоподключение к локальному серверу
        new Thread(() -> {
            try { Thread.sleep(500); } catch (InterruptedException ignored) {}
            if (networkClient.connect("127.0.0.1", 8080)) {
                System.out.println("□ Подключено к серверу");
                networkClient.requestFileList();
            } else {
                System.out.println("□ Сервер не найден (запустите Mp3Server отдельно)");
            }
        }).start();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

AudioPlayer.java

```
package com.mycompany.mp3player;

import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.util.Duration;

import java.io.File;
import java.util.function.Consumer;
```

```

// Управляет воспроизведением аудио
// Отвечает за запуск паузу перемотку и радио
public class AudioPlayer {

    private MediaPlayer mediaPlayer;
    private final PlaylistManager playlistManager;
    private Runnable onEndOfMedia;
    private Consumer<MediaPlayer> onPlayerReady;
    private boolean isPaused;
    private boolean isRadioMode;

    public AudioPlayer(PlaylistManager playlistManager) {
        this.playlistManager = playlistManager;
        this.isPaused = false;
        this.isRadioMode = false;
    }

    // Устанавливает обработчик готовности плеера
    public void setOnPlayerReady(Consumer<MediaPlayer> handler) {
        this.onPlayerReady = handler;
    }

    // Устанавливает обработчик конца трека
    public void setOnEndOfMedia(Runnable handler) {
        this.onEndOfMedia = handler;
    }

    // Возвращает текущий режим файл или радио
    public boolean isRadioMode() {
        return isRadioMode;
    }

    // Запускает текущую песню из плейлиста
    public void playCurrentSong() {
        isRadioMode = false;
        String filePath = playlistManager.getCurrentFilePath();
        if (filePath == null) return;

        File currentFile = new File(filePath);
        if (!currentFile.exists()) {
            System.err.println("Файл не найден: " + filePath);
            return;
        }

        try {
            stopCurrentPlayback();
            Media media = new Media(currentFile.toURI().toString());
            mediaPlayer = new MediaPlayer(media);
            setupMediaPlayerHandlers();
            mediaPlayer.play();
            isPaused = false;
        } catch (Exception e) {
            System.err.println("Ошибка при запуске: " + e.getMessage());
        }
    }

    // Запускает интернет-радио по ссылке потока
    public void playRadioStream(String streamUrl) {
        isRadioMode = true;
        try {
            stopCurrentPlayback();
            Media streamMedia = new Media(streamUrl);

```

```

mediaPlayer = new MediaPlayer(streamMedia);

// Обработчик ошибок для радиопотока
mediaPlayer.setOnError() ->
    System.err.println("Ошибка потока радио: проверьте URL или подключение")
);

if (onPlayerReady != null) {
    onPlayerReady.accept(mediaPlayer);
}

mediaPlayer.play();
isPaused = false;
} catch (Exception e) {
    System.err.println("Ошибка запуска радио: " + e.getMessage());
}
}

// Останавливает текущее воспроизведение и освобождает ресурсы
private void stopCurrentPlayback() {
    if (mediaPlayer != null) {
        mediaPlayer.stop();
        mediaPlayer.dispose();
        mediaPlayer = null;
    }
    isPaused = false;
}

// Настраивает обработчики событий для медиаплеера
private void setupMediaPlayerHandlers() {
    // Обработчик конца трека
    mediaPlayer.setOnEndOfMedia() -> {
        if (playlistManager.isRepeat()) {
            mediaPlayer.seek(Duration.ZERO);
            mediaPlayer.play();
        } else if (onEndOfMedia != null) {
            onEndOfMedia.run();
        }
    }
});

// Обработчик ошибок воспроизведения
mediaPlayer.setOnError() ->
    System.err.println("Ошибка воспроизведения: " +
        (mediaPlayer.getError() != null ? mediaPlayer.getError().getMessage() : "Неизвестная ошибка"))
);

// Уведомляем о готовности плеера
if (onPlayerReady != null) {
    onPlayerReady.accept(mediaPlayer);
}
}

// Ставит воспроизведение на паузу
public void pause() {
    if (mediaPlayer != null) {
        mediaPlayer.pause();
        isPaused = true;
    }
}

// Возобновляет воспроизведение с места паузы
public void resume() {

```

```

        if (mediaPlayer != null && isPaused) {
            mediaPlayer.play();
            isPaused = false;
        }
    }

    // Полностью останавливает воспроизведение
    public void stop() {
        stopCurrentPlayback();
    }

    public boolean isPaused() {
        return isPaused;
    }

    public MediaPlayer getMediaPlayer() {
        return mediaPlayer;
    }

    // Устанавливает громкость от 0.0 до 1.0 с защитой от выхода за границы
    public void setVolume(double volume) {
        if (mediaPlayer != null) {
            mediaPlayer.setVolume(Math.max(0, Math.min(1, volume)));
        }
    }

    // Перемотка на указанную позицию
    // Не работает для радиопотоков
    public void seek(Duration duration) {
        if (mediaPlayer != null && duration != null && !isRadioMode) {
            Duration total = mediaPlayer.getTotalDuration();
            if (total != null && duration.lessThan(total)) {
                mediaPlayer.seek(duration);
            }
        }
    }
}

```

DiscAnimation.java

```
package com.mycompany.mp3player;
```

```

import javafx.animation.RotateTransition;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.image.*;
import javafx.scene.paint.Color;
import javafx.util.Duration;

```

```

import com.mpatric.mp3agic.ID3v2;
import com.mpatric.mp3agic.Mp3File;

```

```

import java.io.ByteArrayInputStream;
import java.io.File;

```

```

// Отображает вращающийся диск с обложкой альбома или иконкой радио
public class DiscAnimation {

```

```

    private static final int DISK_SIZE = 200;
    private static final Color BG_COLOR = Color.rgb(18, 7, 29);
    private static final Color DISK_COLOR = Color.rgb(35, 25, 46);

```

```

    private final ImageView discImageView;

```

```

private final RotateTransition rotation;

public DiscAnimation() {
    discImageView = new ImageView();
    discImageView.setFitWidth(DISK_SIZE);
    discImageView.setFitHeight(DISK_SIZE);
    discImageView.setPreserveRatio(true);

    // Настраиваем анимацию вращения
    rotation = new RotateTransition(Duration.seconds(2), discImageView);
    rotation.setByAngle(360);
    rotation.setCycleCount(RotateTransition.INDEFINITE);

    // Устанавливаем изображение диска по умолчанию
    discImageView.setImage(createDefaultDiscImage());
}

public ImageView getDiscImageView() {
    return discImageView;
}

public void startRotation() {
    rotation.play();
}

public void stopRotation() {
    rotation.pause();
}

// Загружает обложку из MP3-файла
// Если обложка не найдена использует изображение по умолчанию
public void loadAlbumArt(File mp3File) {
    if (mp3File == null || !mp3File.exists()) {
        discImageView.setImage(createDefaultDiscImage());
        return;
    }

    try {
        Mp3File mp3file = new Mp3File(mp3File);
        if (mp3file.hasId3v2Tag()) {
            byte[] data = mp3file.getId3v2Tag().getAlbumImage();
            if (data != null && data.length > 0) {
                Image albumArt = new Image(new ByteArrayInputStream(data), DISK_SIZE, DISK_SIZE, true, true);
                discImageView.setImage(createDiscWithAlbumArt(albumArt));
                return;
            }
        }
    } catch (Exception e) {
        System.err.println("Ошибка загрузки обложки: " + e.getMessage());
    }

    discImageView.setImage(createDefaultDiscImage());
}

// Загружает иконку радиостанции на диск
public void loadRadioIcon(Image radioIcon) {
    if (radioIcon == null) {
        discImageView.setImage(createDefaultDiscImage());
        return;
    }
    try {
        discImageView.setImage(createDiscWithAlbumArt(radioIcon));
    }
}

```

```

    } catch (Exception e) {
        System.err.println("Ошибка загрузки иконки радио: " + e.getMessage());
        discImageView.setImage(createDefaultDiscImage());
    }
}

// Создаёт изображение диска с обрезанной по кругу обложкой
// Добавляет декоративные элементы центр и рамку
private Image createDiscWithAlbumArt(Image albumArt) {
    WritableImage result = new WritableImage(DISK_SIZE, DISK_SIZE);
    PixelWriter pw = result.getPixelWriter();
    PixelReader pr = albumArt.getPixelReader();

    int cx = DISK_SIZE / 2;
    int cy = DISK_SIZE / 2;
    int radius = DISK_SIZE / 2;

    // Копируем пиксели обложки в круглую область
    for (int y = 0; y < DISK_SIZE; y++) {
        for (int x = 0; x < DISK_SIZE; x++) {
            double dist = Math.sqrt(Math.pow(x - cx, 2) + Math.pow(y - cy, 2));
            if (dist <= radius) {
                int sx = Math.min((int)((double)x / DISK_SIZE * albumArt.getWidth()), (int)albumArt.getWidth() - 1);
                int sy = Math.min((int)((double)y / DISK_SIZE * albumArt.getHeight()), (int)albumArt.getHeight() - 1);
                pw.setColor(x, y, pr.getColor(sx, sy));
            } else {
                pw.setColor(x, y, BG_COLOR);
            }
        }
    }

    // Добавляем декоративные элементы на диск
    return decorateDisc(result);
}

// Добавляет декоративные элементы на изображение диска
// Центр рамка блики для визуального эффекта
private Image decorateDisc(WritableImage baseImage) {
    Canvas canvas = new Canvas(DISK_SIZE, DISK_SIZE);
    GraphicsContext gc = canvas.getGraphicsContext2D();

    gc.drawImage(baseImage, 0, 0);

    // Полупрозрачное затемнение
    gc.setFill(Color.rgb(0, 0, 0, 0.2));
    gc.fillOval(0, 0, DISK_SIZE, DISK_SIZE);

    // Белая рамка
    gc.setStroke(Color.rgb(255, 255, 255, 0.6));
    gc.setLineWidth(2);
    gc.strokeOval(0, 0, DISK_SIZE, DISK_SIZE);

    // Центр диска
    gc.setFill(Color.BLACK);
    gc.fillOval(60, 60, 80, 80);
    gc.setFill(BG_COLOR);
    gc.fillOval(80, 80, 40, 40);
    gc.setFill(Color.rgb(255, 255, 255, 0.3));
    gc.fillOval(85, 85, 30, 30);

    return canvas.snapshot(null, null);
}

```

```

// Создаёт стандартный диск без обложки
// Использует цвета темы приложения
private Image createDefaultDiscImage() {
    Canvas canvas = new Canvas(DISK_SIZE, DISK_SIZE);
    GraphicsContext gc = canvas.getGraphicsContext2D();

    // Фон
    gc.setFill(BG_COLOR);
    gc.fillRect(0, 0, DISK_SIZE, DISK_SIZE);

    // Основной круг диска
    gc.setFill(DISK_COLOR);
    gc.fillOval(0, 0, DISK_SIZE, DISK_SIZE);

    // Рамка
    gc.setStroke(Color.rgb(200, 200, 200));
    gc.setLineWidth(2);
    gc.strokeOval(0, 0, DISK_SIZE, DISK_SIZE);

    // Центр диска
    gc.setFill(Color.BLACK);
    gc.fillOval(60, 60, 80, 80);
    gc.setFill(DISK_COLOR);
    gc.fillOval(80, 80, 40, 40);
    gc.setFill(Color.rgb(255, 255, 255, 0.3));
    gc.fillOval(85, 85, 30, 30);

    // Декоративное кольцо
    gc.setFill(Color.rgb(255, 255, 255, 0.2));
    gc.fillOval(10, 10, 90, 90);

    return canvas.snapshot(null, null);
}
}

```

FileUtils.java

```

package com.mycompany.mp3player;

import javafx.scene.control.*;
import javafx.stage.DirectoryChooser;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.StandardCopyOption;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

// Вспомогательный класс для работы с файловой системой
// Отвечает за выбор файлов папок и обработку дубликатов
public class FileUtils {

    // Открывает диалог выбора одного MP3-файла
    // Возвращает файл или null если выбор отменён
    public static File chooseMp3File(Stage stage) {
        try {
            FileChooser fc = new FileChooser();
            fc.setTitle("Выберите MP3 файл");

```

```

        fc.getExtensionFilters().add(new FileChooser.ExtensionFilter("MP3 файлы", "*.mp3"));
        return fc.showOpenDialog(stage);
    } catch (Exception e) {
        System.err.println("Ошибка выбора файла: " + e.getMessage());
        return null;
    }
}

// Открывает диалог выбора папки с музыкой
// Возвращает папку или null если выбор отменён
public static File chooseDirectory(Stage stage) {
    try {
        DirectoryChooser dc = new DirectoryChooser();
        dc.setTitle("Выберите папку");
        return dc.showDialog(stage);
    } catch (Exception e) {
        System.err.println("Ошибка выбора папки: " + e.getMessage());
        return null;
    }
}

// Добавляет один файл в плейлист с проверкой на дубликаты
// При дубликate предлагает пользователю выбор действия
// Возвращает добавленный файл или null
public static File addFileWithDuplicateHandling(Stage stage, PlaylistManager manager) {
    File file = chooseMp3File(stage);
    if (file == null) return null;
    return handleFileAddition(stage, manager, file);
}

// Добавляет все MP3-файлы из папки в плейлист
// Для каждого файла проверяет дубликаты
// Возвращает список успешно добавленных файлов
public static List<File> addFolderWithDuplicateHandling(Stage stage, PlaylistManager manager) {
    File directory = chooseDirectory(stage);
    if (directory == null) return new ArrayList<>();

    File[] files = getMp3FilesFromDirectory(directory);
    if (files == null || files.length == 0) return new ArrayList<>();

    List<File> addedFiles = new ArrayList<>();
    int skippedCount = 0;

    for (File file : files) {
        File result = handleFileAddition(stage, manager, file);
        if (result != null) {
            addedFiles.add(result);
        } else {
            skippedCount++;
        }
    }

    // Показываем статистику если были пропуски
    if (skippedCount > 0) {
        showAlert(stage, "Результат добавления",
            "Добавлено: " + addedFiles.size() + "\n" +
            "Пропущено уже есть в списке: " + skippedCount);
    }

    return addedFiles;
}

```

```

// Универсальный метод обработки добавления одного файла
// Проверяет дубликаты и вызывает нужный обработчик
// Возвращает файл если он был добавлен или null
private static File handleFileAddition(Stage stage, PlaylistManager manager, File file) {
    int existingIndex = findFileIndexStrict(manager, file);

    if (existingIndex != -1) {
        return handleDuplicateFile(stage, file, manager, existingIndex);
    } else {
        manager.addFile(file.getName(), file.getAbsolutePath());
        return file;
    }
}

// Ищет файл в плейлисте по пути или имени
// Использует игнорирование регистра для надёжности
// Возвращает индекс файла или -1 если не найден
private static int findFileIndexStrict(PlaylistManager manager, File targetFile) {
    String targetPath = targetFile.getAbsolutePath();
    String targetName = targetFile.getName().toLowerCase();

    for (int i = 0; i < manager.size(); i++) {
        String existingPath = manager.getFilePath(i);
        if (existingPath != null) {
            File existingFile = new File(existingPath);

            // Сравнение по полному пути
            if (targetPath.equalsIgnoreCase(existingFile.getAbsolutePath())) {
                return i;
            }

            // Дополнительная проверка по имени файла
            if (targetName.equals(existingFile.getName().toLowerCase())) {
                return i;
            }
        }
    }
    return -1;
}

// Показывает диалог при обнаружении дубликата
// Предлагает отменить переименовать или заменить файл
// Возвращает файл если он был добавлен или null
private static File handleDuplicateFile(Stage stage, File file, PlaylistManager manager, int existingIndex) {
    Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
    alert.setTitle("Файл уже существует");
    alert.setHeaderText("Файл " + file.getName() + " уже есть в плейлисте");
    alert.setContentText("Выберите действие:");

    ButtonType buttonCancel = new ButtonType("Отменить", ButtonBar.ButtonData.CANCEL_CLOSE);
    ButtonType buttonRename = new ButtonType("Переименовать", ButtonBar.ButtonData.OK_DONE);
    ButtonType buttonReplace = new ButtonType("Заменить", ButtonBar.ButtonData.YES);

    alert.getButtonTypes().setAll(buttonCancel, buttonRename, buttonReplace);

    Optional<ButtonType> result = alert.showAndWait();

    if (result.isPresent()) {
        if (result.get() == buttonReplace) {
            // Заменяем существующую запись новой
            manager.removeFile(existingIndex);
            manager.addFile(file.getName(), file.getAbsolutePath());
        }
    }
}

```

```

        return file;
    } else if (result.get() == buttonRename) {
        // Создаём копию с новым именем
        return handleRenameDialog(stage, file, manager);
    }
}
return null;
}

// Открывает диалог для ввода нового имени файла
// Создаёт копию оригинального файла с новым именем
// Возвращает новый файл или null при ошибке
private static File handleRenameDialog(Stage stage, File originalFile, PlaylistManager manager) {
    TextInputDialog textDialog = new TextInputDialog(originalFile.getName().replace(".mp3", "") + "_copy");
    textDialog.setTitle("Переименовать файл");
    textDialog.setHeaderText("Введите новое имя для копии файла:");
    textDialog.setContentText("Имя:");

    Optional<String> renameResult = textDialog.showAndWait();
    if (renameResult.isPresent()) {
        String newName = renameResult.get().trim();
        if (!newName.isEmpty()) {
            if (!newName.toLowerCase().endsWith(".mp3")) newName += ".mp3";
            try {
                File newFile = new File(originalFile.getParent(), newName);
                Files.copy(originalFile.toPath(), newFile.toPath(), StandardCopyOption.REPLACE_EXISTING);
                manager.addFile(newName, newFile.getAbsolutePath());
                return newFile;
            } catch (IOException e) {
                showAlert(stage, "Ошибка", "Не удалось создать копию: " + e.getMessage());
            }
        }
    }
    return null;
}

// Показывает информационное окно с сообщением
// Блокирует интерфейс до закрытия окна
private static void showAlert(Stage stage, String title, String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

// Возвращает массив всех MP3-файлов в указанной папке
// Игнорирует файлы с другими расширениями
// Возвращает пустой массив если папка не существует
public static File[] getMp3FilesFromDirectory(File dir) {
    if (dir == null || !dir.isDirectory()) return new File[0];
    try {
        return dir.listFiles((d, name) -> name.toLowerCase().endsWith(".mp3"));
    } catch (Exception e) {
        return new File[0];
    }
}
}
}

```

Mp3Server.java

package com.mycompany.mp3player;

```

import java.io.*;
import java.net.*;
import java.nio.charset.StandardCharsets;
import java.nio.file.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import javafx.application.Application;
import javafx.application.Platform;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public class Mp3Server extends Application {
    private static final int DEFAULT_PORT = 8080;
    private static final String MUSIC_DIR = "./server_music";

    // UI КОМПОНЕНТЫ
    private Button startStopButton;
    private Button uploadLocalBtn;
    private Button deleteLocalBtn;
    private Label statusLabel;
    private Label portLabel;
    private Label clientsLabel;
    private Label filesLabel;
    private ListView<String> clientsListView;
    private ListView<String> serverFilesView;
    private TextArea logArea;

    // Сетевые компоненты
    private ServerSocket serverSocket;
    private volatile boolean serverRunning = false;
    private ExecutorService pool;
    private List<String> connectedClients = new ArrayList<>();

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        try {
            Files.createDirectories(Paths.get(MUSIC_DIR));
            primaryStage.setTitle("MP3 Server");
            primaryStage.setMinWidth(700);
            primaryStage.setMinHeight(650);

            BorderPane root = new BorderPane();
            root.setPadding(new Insets(10));
            root.setStyle("-fx-background-color: #24192E;");

            root.setTop(createTopPanel());
            root.setCenter(createCenterPanel());
            root.setBottom(createBottomPanel());

            Scene scene = new Scene(root, 700, 650);

```

```

primaryStage.setScene(scene);
primaryStage.show();

primaryStage.setOnCloseRequest(e -> {
    stopServer();
    Platform.exit();
});

refreshServerFilesView();

} catch (Exception e) {
    e.printStackTrace();
}
}

private VBox createTopPanel() {
    VBox panel = new VBox(10);
    panel.setStyle("-fx-background-color: #12071D; -fx-padding: 10; -fx-border-radius: 5; -fx-background-radius:
5;");

    Label title = new Label("☐ MP3 Network Server");
    title.setStyle("-fx-font-size: 18px; -fx-font-weight: bold; -fx-text-fill: #7B688E;");
    title.setAlignment(Pos.CENTER);

    HBox buttonBox = new HBox(10);
    buttonBox.setAlignment(Pos.CENTER);

    startStopButton = new Button("▶ Запустить сервер");
    startStopButton.setStyle(getButtonStyle("#7B688E"));
    startStopButton.setPrefWidth(200);
    startStopButton.setPrefHeight(40);
    startStopButton.setOnAction(e -> toggleServer());

    buttonBox.getChildren().add(startStopButton);

    statusLabel = new Label("● Остановлен");
    statusLabel.setStyle("-fx-font-size: 14px; -fx-text-fill: #7B688E; -fx-font-weight: bold;");
    statusLabel.setAlignment(Pos.CENTER);

    portLabel = new Label("Порт: " + DEFAULT_PORT);
    portLabel.setStyle("-fx-font-size: 12px; -fx-text-fill: #95a5a6;");
    portLabel.setAlignment(Pos.CENTER);

    panel.getChildren().addAll(title, buttonBox, statusLabel, portLabel);
    return panel;
}

private HBox createCenterPanel() {
    HBox mainBox = new HBox(10);
    mainBox.setStyle("-fx-background-color: #12071D; -fx-padding: 10; -fx-border-radius: 5; -fx-background-radius:
5;");

    // Панель клиентов
    VBox clientBox = new VBox(5);
    clientsLabel = new Label("☐ Подключенные клиенты (0)");
    clientsLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; -fx-text-fill: #7B688E;");

    clientsListView = new ListView<>();
    clientsListView.setStyle("-fx-background-color: #24192E; -fx-text-fill: #FFFFFF; " +
        "-fx-border-color: #4A235A; -fx-border-width: 1; " +
        "-fx-border-radius: 4; -fx-background-radius: 4; " +
        "-fx-control-inner-background: #24192E; " +

```

```

        "-fx-control-inner-background-alt: #24192E;");
clientsListView.setPrefHeight(200);

clientsListView.setCellFactory(lv -> createDarkCell());

clientBox.getChildren().addAll(clientsLabel, clientsListView);
VBox.setVgrow(clientBox, Priority.ALWAYS);

// Панель файлов на сервере
VBox fileBox = new VBox(5);
filesLabel = new Label("□ Файлы на сервере (0)");
filesLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; -fx-text-fill: #7B688E;");

serverFilesView = new ListView<>();
serverFilesView.setStyle("-fx-background-color: #24192E; -fx-text-fill: #FFFFFF; " +
    "-fx-border-color: #4A235A; -fx-border-width: 1; " +
    "-fx-border-radius: 4; -fx-background-radius: 4; " +
    "-fx-control-inner-background: #24192E; " +
    "-fx-control-inner-background-alt: #24192E;");
serverFilesView.setPrefHeight(150);

serverFilesView.setCellFactory(lv -> createDarkCell());

// Кнопки управления файлами
uploadLocalBtn = createSmallButton("↑ □", "Загрузить файл с ПК");
uploadLocalBtn.setStyle(getButtonStyle("#7B688E"));
uploadLocalBtn.setOnAction(e -> uploadLocalFile());

deleteLocalBtn = createSmallButton("□", "Удалить выбранный файл");
deleteLocalBtn.setStyle(getButtonStyle("#7B688E"));
deleteLocalBtn.setDisable(true);
deleteLocalBtn.setOnAction(e -> deleteLocalFile());

// Активация кнопки удаления при выборе файла
serverFilesView.getSelectionModel().selectedItemProperty().addListener((obs, oldVal, newVal) -> {
    deleteLocalBtn.setDisable(newVal == null);
});

HBox fileActions = new HBox(5, uploadLocalBtn, deleteLocalBtn);
fileBox.getChildren().addAll(filesLabel, serverFilesView, fileActions);
VBox.setVgrow(fileBox, Priority.ALWAYS);

mainBox.getChildren().addAll(clientBox, fileBox);
return mainBox;
}

private ListCell<String> createDarkCell() {
    return new ListCell<String>() {
        @Override
        protected void updateItem(String item, boolean empty) {
            super.updateItem(item, empty);
            if (empty || item == null) {
                setText(null);
                setStyle("-fx-background-color: #24192E;");
            } else {
                setText(item);
                setStyle("-fx-background-color: #24192E; -fx-text-fill: #FFFFFF;");
            }
        }
    };
}
}

```

```

private Button createSmallButton(String text, String tooltip) {
    Button btn = new Button(text);
    btn.setTooltip(new Tooltip(tooltip));
    btn.setStyle("-fx-background-color: #4A235A; -fx-text-fill: white; " +
        "-fx-font-weight: bold; -fx-border-radius: 4; " +
        "-fx-background-radius: 4; -fx-cursor: hand;");
    return btn;
}

private VBox createBottomPanel() {
    VBox panel = new VBox(5);
    panel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; -fx-text-fill: #7B688E;");

    Label logLabel = new Label("□ Лог событий");
    logLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; -fx-text-fill: #7B688E;");

    logArea = new TextArea();
    logArea.setEditable(false);
    logArea.setStyle("-fx-background-color: #24192E; -fx-text-fill: #FFFFFF; " +
        "-fx-border-color: #4A235A; -fx-border-width: 1; " +
        "-fx-border-radius: 4; -fx-background-radius: 4; " +
        "-fx-control-inner-background: #24192E; " +
        "-fx-control-inner-background-alt: #24192E;");
    logArea.setPrefHeight(120);
    logArea.setWrapText(true);

    panel.getChildren().addAll(logLabel, logArea);
    VBox.setVgrow(logArea, Priority.ALWAYS);

    return panel;
}

private String getButtonStyle(String color) {
    return "-fx-background-color: #7B688E;" +
        "-fx-text-fill: white; " +
        "-fx-font-size: 14px; " +
        "-fx-font-weight: bold; " +
        "-fx-border-radius: 5; " +
        "-fx-background-radius: 5; " +
        "-fx-cursor: hand;";
}

private void toggleServer() {
    if (serverRunning) {
        stopServer();
    } else {
        startServer();
    }
}

private void startServer() {
    try {
        serverSocket = new ServerSocket(DEFAULT_PORT);
        serverRunning = true;
        pool = Executors.newCachedThreadPool();

        Platform.runLater() -> {
            startStopButton.setText("□ Остановить сервер");
            startStopButton.setStyle(getButtonStyle("#500a91"));
            statusLabel.setText("● Работает");
            statusLabel.setStyle("-fx-font-size: 14px; -fx-text-fill: #500a91; -fx-font-weight: bold;");
        }
    }
}

```

```

        log("Сервер запущен на порту " + DEFAULT_PORT);
        log("Папка с музыкой: " + new File(MUSIC_DIR).getAbsolutePath());
    });

    new Thread(this::acceptConnections).start();

} catch (IOException e) {
    Platform.runLater() -> {
        log("Ошибка запуска сервера: " + e.getMessage());
        showAlert("Ошибка", "Не удалось запустить сервер:\n" + e.getMessage());
    });
}
}

private void stopServer() {
    serverRunning = false;

    try {
        if (serverSocket != null && !serverSocket.isClosed()) {
            serverSocket.close();
        }
        if (pool != null) {
            pool.shutdown();
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    Platform.runLater() -> {
        connectedClients.clear();
        clientsListView.getItems().clear();
        updateClientsCount();

        startStopButton.setText("▶ Запустить сервер");
        startStopButton.setStyle(getButtonStyle("#7B688E"));
        statusLabel.setText("● Остановлен");
        statusLabel.setStyle("-fx-font-size: 14px; -fx-text-fill: #7B688E; -fx-font-weight: bold;");
        log("Сервер остановлен");
    });
}

private void acceptConnections() {
    while (serverRunning) {
        try {
            Socket client = serverSocket.accept();
            log("Клиент подключён: " + client.getInetAddress());

            String clientInfo = client.getInetAddress().getHostAddress() + ":" + client.getPort();
            Platform.runLater() -> {
                connectedClients.add(clientInfo);
                clientsListView.getItems().add(clientInfo);
                updateClientsCount();
            });

            pool.submit() -> handleClient(client, clientInfo);

        } catch (IOException e) {
            if (serverRunning) {
                log("Ошибка принятия подключения: " + e.getMessage());
            }
        }
    }
}
}

```

```

}

private void handleClient(Socket client, String clientInfo) {
    FileOutputStream uploadStream = null;
    String uploadFileName = null;

    try (DataInputStream in = new DataInputStream(client.getInputStream());
        DataOutputStream out = new DataOutputStream(client.getOutputStream())) {

        while (client.isConnected() && !client.isClosed() && serverRunning) {
            byte type = in.readByte();
            int len = in.readInt();

            if (len < 0 || len > 10 * 1024 * 1024) {
                log("❑ Некорректная длина от " + clientInfo + ": " + len);
                break;
            }

            byte[] payload = new byte[len];
            in.readFully(payload);

            if (type == 1) { // КОМАНДА
                String cmd = new String(payload, StandardCharsets.UTF_8);

                if (cmd.equals("LIST")) {
                    sendResponse(out, listFiles());
                }

                else if (cmd.startsWith("DOWNLOAD:")) {
                    String file = cmd.substring(9);
                    File target = new File(MUSIC_DIR, file);
                    if (target.exists()) {
                        sendResponse(out, "START_DOWNLOAD:" + file);
                        streamFile(out, target);
                        sendResponse(out, "END_DOWNLOAD");
                        log("❑ Отправлен файл " + file + " клиенту " + clientInfo);
                    } else {
                        sendResponse(out, "ERROR:File not found");
                    }
                }

                else if (cmd.startsWith("DELETE:")) {
                    String fileName = cmd.substring(7);
                    File target = new File(MUSIC_DIR, fileName);
                    if (target.exists()) {
                        if (target.delete()) {
                            sendResponse(out, "OK");
                            log("❑ Удалён файл: " + fileName + " (клиент " + clientInfo + ")");
                            refreshServerFilesView();
                        } else {
                            sendResponse(out, "ERROR:Deletion failed");
                            log("❑ Ошибка удаления: " + fileName);
                        }
                    } else {
                        sendResponse(out, "ERROR:File not found");
                    }
                }

                else if (cmd.startsWith("UPLOAD_START:")) {
                    uploadFileName = cmd.substring(13);
                    File dest = new File(MUSIC_DIR, uploadFileName);
                    uploadStream = new FileOutputStream(dest);
                }
            }
        }
    }
}

```

```

        log("↑ □ Начало загрузки от " + clientInfo + ": " + uploadFileName);
        sendResponse(out, "OK");
    }

    else if (cmd.equals("UPLOAD_END")) {
        if (uploadStream != null) {
            uploadStream.close();
            uploadStream = null;
        }
        log("□ Загрузка завершена: " + uploadFileName);
        sendResponse(out, "OK");
        uploadFileName = null;
        refreshServerFilesView();
    }

    } else if (type == 2 && uploadStream != null) { // ДАННЫЕ ЗАГРУЗКИ
        uploadStream.write(payload, 0, len);
        uploadStream.flush();
    }
}
} catch (Exception e) {
    log("□ Клиент отключился: " + clientInfo);
} finally {
    if (uploadStream != null) {
        try { uploadStream.close(); } catch (IOException ignored) {}
    }
    try { client.close(); } catch (IOException ignored) {}

    Platform.runLater() -> {
        connectedClients.remove(clientInfo);
        clientsListView.getItems().remove(clientInfo);
        updateClientsCount();
    });
}
}

private String listFiles() {
    File dir = new File(MUSIC_DIR);
    File[] files = dir.listFiles((d, n) -> n.toLowerCase().endsWith(".mp3"));
    StringBuilder sb = new StringBuilder("FILES:");
    if (files != null) {
        Arrays.stream(files).sorted().forEach(f -> sb.append(f.getName()).append("\n"));
    }
    return sb.toString();
}

private void streamFile(DataOutputStream out, File file) throws IOException {
    try (FileInputStream fis = new FileInputStream(file)) {
        byte[] buf = new byte[8192];
        int n;
        while ((n = fis.read(buf)) != -1) {
            out.writeByte(2);
            out.writeInt(n);
            out.write(buf, 0, n);
            out.flush();
        }
    }
}

private void sendResponse(DataOutputStream out, String msg) throws IOException {
    byte[] b = msg.getBytes(StandardCharsets.UTF_8);
    out.writeByte(3);
}

```

```

        out.writeInt(b.length);
        out.write(b);
        out.flush();
    }

    private void updateClientsCount() {
        clientsLabel.setText("□ Подключенные клиенты (" + connectedClients.size() + ")");
    }

    private void log(String message) {
        Platform.runLater(() -> {
            logArea.appendText("[ " + getTime() + " ] " + message + "\n");
        });
    }

    private String getTime() {
        return java.time.format.DateTimeFormatter.ofPattern("HH:mm:ss")
            .format(java.time.LocalDateTime.now());
    }

    private void refreshServerFilesView() {
        Platform.runLater(() -> {
            serverFilesView.getItems().clear();
            File dir = new File(MUSIC_DIR);
            File[] files = dir.listFiles((d, n) -> n.toLowerCase().endsWith(".mp3"));
            if (files != null) {
                Arrays.stream(files).sorted().forEach(f -> serverFilesView.getItems().add(f.getName()));
            }
            filesLabel.setText("□ Файлы на сервере (" + serverFilesView.getItems().size() + ")");
        });
    }

    private void uploadLocalFile() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Выберите MP3 файл для загрузки на сервер");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("MP3 Files", "*.mp3"));

        File selectedFile = fileChooser.showOpenDialog(null);
        if (selectedFile != null) {
            try {
                File destFile = new File(MUSIC_DIR, selectedFile.getName());
                Files.copy(selectedFile.toPath(), destFile.toPath(), StandardCopyOption.REPLACE_EXISTING);
                log("□ Локально загружен: " + selectedFile.getName());
                refreshServerFilesView();
                showAlert("Успех", "Файл " + selectedFile.getName() + " успешно загружен на сервер");
            } catch (IOException e) {
                log("□ Ошибка локальной загрузки: " + e.getMessage());
                showAlert("Ошибка", "Не удалось загрузить файл:\n" + e.getMessage());
            }
        }
    }

    private void deleteLocalFile() {
        String selectedFile = serverFilesView.getSelectionModel().getSelectedItem();
        if (selectedFile != null) {
            Alert confirm = new Alert(Alert.AlertType.CONFIRMATION);
            confirm.setTitle("Подтверждение удаления");
            confirm.setHeaderText(null);
            confirm.setContentText("Удалить файл " + selectedFile + " с сервера?");

            confirm.showAndWait().ifPresent(response -> {

```

```

        if (response == ButtonType.OK) {
            File fileToDelete = new File(MUSIC_DIR, selectedFile);
            if (fileToDelete.delete()) {
                log("□ Локально удалён: " + selectedFile);
                refreshServerFilesView();
                showAlert("Успех", "Файл успешно удалён");
            } else {
                log("□ Ошибка удаления: " + selectedFile);
                showAlert("Ошибка", "Не удалось удалить файл");
            }
        }
    });
}
}

private void showAlert(String title, String message) {
    Platform.runLater() -> {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    });
}
}

```

NetworkClient.java

```
package com.mycompany.mp3player;
```

```

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;

public class NetworkClient {
    private Socket socket;
    private DataInputStream in;
    private DataOutputStream out;
    private volatile boolean connected = false;
    private final String cacheDir = "./cache";

    private Consumer<List<String>> onListUpdated;
    private Consumer<String[]> onFileDownloaded;
    private Consumer<Boolean> onConnectionChanged;

    public boolean connect(String host, int port) {
        try {
            Files.createDirectories(Paths.get(cacheDir));
            socket = new Socket(host, port);
            in = new DataInputStream(socket.getInputStream());
            out = new DataOutputStream(socket.getOutputStream());
            connected = true;
            if (onConnectionChanged != null) onConnectionChanged.accept(true);

```

```

        new Thread(this::listenLoop).start();
        return true;
    } catch (IOException e) { return false; }
}

public void disconnect() {
    connected = false;
    try { if (socket != null) socket.close(); } catch (IOException ignored) {}
    if (onConnectionChanged != null) onConnectionChanged.accept(false);
}

public boolean isConnected() { return connected; }

public void setOnListUpdated(Consumer<List<String>> c) { onListUpdated = c; }
public void setOnFileDownloaded(Consumer<String[]> c) { onFileDownloaded = c; }
public void setOnConnectionChanged(Consumer<Boolean> c) { onConnectionChanged = c; }

public void requestFileList() { sendCommandFrame("LIST"); }
public void downloadFile(String filename) { sendCommandFrame("DOWNLOAD:" + filename); }

    public void deleteFile(String filename) {
        sendCommandFrame("DELETE:" + filename);
        // Автоматическое обновление списка через полсекунды
        new Thread() -> {
            try { Thread.sleep(500); } catch (InterruptedException e) {}
            requestFileList();
        }.start();
    }

public void uploadFile(File file) {
    if (!connected || !file.exists()) return;
    new Thread() -> {
        try {
            sendCommandFrame("UPLOAD_START:" + file.getName());
            byte[] buf = new byte[8192];
            try (FileInputStream fis = new FileInputStream(file)) {
                int n; while ((n = fis.read(buf)) != -1) sendAudioFrame(buf, n);
            }
            sendCommandFrame("UPLOAD_END");
            requestFileList();
        } catch (IOException e) { e.printStackTrace(); }
    }.start();
}

private FileOutputStream downloadStream = null;
private String downloadingFile = null;

private void listenLoop() {
    try {
        while (connected && !socket.isClosed()) {
            byte type = in.readByte();
            int len = in.readInt();
            if (len < 0 || len > 10 * 1024 * 1024) break;

            byte[] payload = new byte[len];
            in.readFully(payload);

            if (type == 3) {
                String resp = new String(payload, StandardCharsets.UTF_8);
                if (resp.startsWith("FILES:")) {
                    List<String> list = new ArrayList<>();

```

```

        String[] lines = resp.substring(6).split("\n");
        for (String l : lines) if (!l.isEmpty()) list.add(l);
        if (onListUpdated != null) onListUpdated.accept(list);
    } else if (resp.startsWith("START_DOWNLOAD:")) {
        downloadingFile = resp.substring(15);
        downloadStream = new FileOutputStream(new File(cacheDir, downloadingFile));
    } else if (resp.equals("END_DOWNLOAD")) {
        if (downloadStream != null) { try { downloadStream.close(); } catch (IOException ignored) {}
downloadStream = null; }
        if (downloadingFile != null) {
            String path = new File(cacheDir, downloadingFile).getAbsolutePath();
            if (onFileDownloaded != null) onFileDownloaded.accept(new String[]{path, downloadingFile});
        }
        downloadingFile = null;
    }
    } else if (type == 2 && downloadStream != null) {
        downloadStream.write(payload, 0, len);
    }
    }
    } catch (IOException e) {
        if (connected) System.err.println("❑ Сеть разорвана: " + e.getMessage());
        if (downloadStream != null) { try { downloadStream.close(); } catch (IOException ignored) {} downloadStream
= null; }
        disconnect();
    }
    }

private void sendCommandFrame(String command) {
    try {
        if (out == null) return;
        byte[] data = command.getBytes(StandardCharsets.UTF_8);
        out.writeByte(1); out.writeInt(data.length); out.write(data); out.flush();
    } catch (IOException e) { System.err.println("❑ Ошибка отправки: " + e.getMessage()); }
}

private void sendAudioFrame(byte[] data, int len) {
    try {
        if (out == null) return;
        out.writeByte(2); out.writeInt(len); out.write(data, 0, len); out.flush();
    } catch (IOException e) { System.err.println("❑ Ошибка данных: " + e.getMessage()); }
}
}

```

PlaylistManager.java

```
package com.mycompany.mp3player;
```

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import java.io.*;
import java.nio.file.*;
import java.util.*;
```

```
public class PlaylistManager {
```

```

    private final ObservableList<String> playlist;
    private final List<String> originalPlaylist;
    private int currentIndex;
    private boolean isShuffle;
    private boolean isRepeat;
    private boolean isNetworkPlaylist;
    private NetworkClient networkClient;
```

```

public PlaylistManager() {
    this.playlist = FXCollections.observableArrayList();
    this.originalPlaylist = new ArrayList<>();
    this.currentIndex = 0;
    this.isShuffle = false;
    this.isRepeat = false;
    this.isNetworkPlaylist = false;
}

public void setNetworkClient(NetworkClient c) { this.networkClient = c; }
public boolean isNetworkPlaylist() { return isNetworkPlaylist; }
public ObservableList<String> getPlaylist() { return FXCollections.unmodifiableObservableList(playlist); }
public int getCurrentIndex() { return currentIndex; }
public void setCurrentIndex(int i) { this.currentIndex = i; }
public boolean isShuffle() { return isShuffle; }
public void setShuffle(boolean s) { this.isShuffle = s; }
public boolean isRepeat() { return isRepeat; }
public void setRepeat(boolean r) { this.isRepeat = r; }
public int size() { return playlist.size(); }

public String getCurrentFilePath() {
    if (currentIndex >= 0 && currentIndex < originalPlaylist.size()) return originalPlaylist.get(currentIndex);
    return null;
}

public String getFilePath(int i) {
    if (i >= 0 && i < originalPlaylist.size()) return originalPlaylist.get(i);
    return null;
}

public synchronized void addFile(String name, String path) {
    if (name != null && path != null && !playlist.contains(name)) {
        playlist.add(name);
        originalPlaylist.add(path);
    }
}

public synchronized boolean removeFile(int i) {
    if (isNetworkPlaylist) return false;
    if (i < 0 || i >= playlist.size()) return false;
    boolean was = (i == currentIndex);
    playlist.remove(i);
    originalPlaylist.remove(i);
    if (currentIndex == i) currentIndex = 0;
    else if (currentIndex > i) currentIndex--;
    return was;
}

public synchronized void renameFile(int i, String newName) {
    if (isNetworkPlaylist || i < 0 || i >= originalPlaylist.size()) return;
    String old = originalPlaylist.get(i);
    if (old.startsWith("network://")) return;
    File of = new File(old);
    File nf = new File(of.getParent(), newName);
    try {
        if (of.renameTo(nf)) {
            originalPlaylist.set(i, nf.getAbsolutePath());
        }
    }
}

```

```

        playlist.set(i, newName);
    }
} catch (Exception e) {
    System.err.println("Ошибка переименования: " + e.getMessage());
}
}

public String savePlaylistWithFiles(String name, String baseDir) throws IOException {
    if (isNetworkPlaylist) throw new IOException("Сетевой плейлист не сохраняется локально");
    if (name == null || baseDir == null) throw new IllegalArgumentException("Параметры пустые");
    File folder = new File(baseDir, name);
    if (!folder.exists()) folder.mkdirs();
    for (int i = 0; i < originalPlaylist.size(); i++) {
        String p = originalPlaylist.get(i);
        if (p.startsWith("network://")) continue;
        File src = new File(p);
        File dst = new File(folder, playlist.get(i));
        if (src.exists()) Files.copy(src.toPath(), dst.toPath(), StandardCopyOption.REPLACE_EXISTING);
    }
    return folder.getAbsolutePath();
}

public int getNextIndex() {
    if (playlist.isEmpty()) return -1;
    return isShuffle ? (int)(Math.random() * playlist.size()) : (currentIndex + 1) % playlist.size();
}

public int getPreviousIndex() {
    if (playlist.isEmpty()) return -1;
    return currentIndex > 0 ? currentIndex - 1 : playlist.size() - 1;
}

public synchronized void clear() {
    playlist.clear();
    originalPlaylist.clear();
    currentIndex = 0;
    isNetworkPlaylist = false;
}
}

```

RadioStation.java

```
package com.mycompany.mp3player;
```

```
import javafx.scene.image.ImageView;
```

```
// Класс для хранения данных радиостанции
```

```
// Исправление: добавлен конструктор с иконкой
```

```
public class RadioStation {
```

```
    private final String name;
    private final String streamUrl;
    private final ImageView iconView;
```

```
    public RadioStation(String name, String streamUrl) {
        this(name, streamUrl, null);
    }
```

```
// Конструктор с возможностью задать иконку
```

```
    public RadioStation(String name, String streamUrl, ImageView icon) {
        this.name = name;
        this.streamUrl = streamUrl;
        this.iconView = icon != null ? icon : createDefaultIcon();
    }

```

```

        this.iconView.setFitWidth(32);
        this.iconView.setFitHeight(32);
    }

    // Создаёт иконку по умолчанию для радиостанции
    private ImageView createDefaultIcon() {
        ImageView iv = new ImageView();
        // Здесь можно добавить создание изображения по умолчанию
        return iv;
    }

    public String getName() {
        return name;
    }

    public String getStreamUrl() {
        return streamUrl;
    }

    public ImageView getIconView() {
        return iconView;
    }

    @Override
    public String toString() {
        return name;
    }
}

```

UIComponents.java

```

package com.mycompany.mp3player;

import com.mpatric.mp3agic.ID3v2;
import javafx.application.Platform;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.collections.transformation.FilteredList;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.*;
import javafx.scene.media.MediaPlayer;
import javafx.scene.paint.Color;
import javafx.stage.Stage;
import javafx.util.Duration;
import com.mpatric.mp3agic.Mp3File;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Consumer;

public class UIComponents {
    public static final String COLOR_BG_MAIN = "#12071D";
    public static final String COLOR_BG_LIST = "#24192E";
    public static final String COLOR_TEXT = "#FFFFFF";
}

```

```

public static final String COLOR_BTN = "#7B688E";
public static final String COLOR_BTN_HOVER = "#755C8E";
public static final String COLOR_ACCENT = "#500a91";

private final PlaylistManager playlistManager;
private final AudioPlayer audioPlayer;
private final DiscAnimation discAnimation;
private final Stage stage;
private final NetworkClient networkClient;

private ListView<PlaylistItem> playlistView;
private Label currentSongLabel;
private Slider timeSlider;
private Slider volumeSlider;
private Label currentTimeLabel;
private Label totalTimeLabel;
private Label volumePercentLabel;
private TextField searchField;
private ToggleButton playlistModeToggle;
private ToggleButton radioModeToggle;
private ToggleGroup modeToggleGroup;
private ListView<RadioStation> radioListView;
private HBox timeBox;

private Button addFileButton;
private Button addFolderButton;
private Button removeButton;
private Button renameButton;
private Button saveButton;
private Button previousButton;
private Button playButton;
private Button pauseButton;
private Button nextButton;
private ToggleButton shuffleButton;
private ToggleButton repeatButton;

private Button btnConnect;
private Button btnDisconnect;
private Button btnRefreshServer;
private Button btnDownload;
private ListView<String> serverFilesListView;

private boolean isUserDraggingTimeSlider;
private final Map<String, Image> albumIconCache;
private double currentScale;
private boolean isPlaying;
private int currentRadioIndex;
private boolean isRadioModeActive;
private ObservableList<PlaylistItem> masterPlaylistData;
private ObservableList<RadioStation> radioStations;

public UIComponents(PlaylistManager playlistManager, AudioPlayer audioPlayer, Stage stage, NetworkClient
networkClient) {
    this.playlistManager = playlistManager;
    this.audioPlayer = audioPlayer;
    this.discAnimation = new DiscAnimation();
    this.stage = stage;
    this.networkClient = networkClient;
    this.isUserDraggingTimeSlider = false;
    this.albumIconCache = new HashMap<>();
    this.currentScale = 1.0;

```

```

this.isPlaying = false;
this.currentRadioIndex = 0;
this.isRadioModeActive = false;
this.masterPlaylistData = FXCollections.observableArrayList();
this.radioStations = FXCollections.observableArrayList();

radioStations.add(new RadioStation("Европа Плюс", "http://ep128.hostingradio.ru:8030/ep128"));
radioStations.add(new RadioStation("Русское Радио", "https://rusradio.hostingradio.ru/rusradio128.mp3"));
radioStations.add(new RadioStation("ДФМ", "https://dfm.hostingradio.ru/dfm128.mp3"));
radioStations.add(new RadioStation("Дорожное радио", "http://dorognoe.hostingradio.ru:8000/dorognoe"));

setupNetworkCallbacks();
}

private void setupNetworkCallbacks() {
    if (networkClient == null) return;

    networkClient.setOnConnectionChanged(connected ->
        Platform.runLater() -> {
            if (btnConnect != null) btnConnect.setDisable(connected);
            if (btnDisconnect != null) btnDisconnect.setDisable(!connected);
            if (btnRefreshServer != null) btnRefreshServer.setDisable(!connected);
            if (btnDownload != null) btnDownload.setDisable(!connected ||
                (serverFilesListView != null && serverFilesListView.getSelectionModel().getSelectedItem() == null));
        });

    networkClient.setOnListUpdated(files ->
        Platform.runLater() -> {
            if (serverFilesListView != null) {
                serverFilesListView.getItems().setAll(files);
                if (btnDownload != null) {
                    btnDownload.setDisable(serverFilesListView.getSelectionModel().getSelectedItem() == null);
                }
            }
        });

    networkClient.setOnFileDownloaded(data ->
        Platform.runLater() -> {
            String path = data[0];
            String name = data[1];
            playlistManager.addFile(name, path);
            loadAlbumArtFromMp3(new File(path));
            refreshPlaylist();
            showAlert("Готово", "Трек " + name + " скачан и добавлен в плейлист");
        });
    }

    public BorderPane getRootPane() {
        BorderPane root = new BorderPane();
        root.setPadding(new Insets(15));
        HBox mainContent = new HBox(20);
        VBox leftPanel = createLeftPanel();
        VBox rightPanel = createRightPanel();
        HBox.setHgrow(rightPanel, Priority.ALWAYS);
        mainContent.getChildren().addAll(leftPanel, rightPanel);
        root.setCenter(mainContent);
        return root;
    }

    public VBox createLeftPanel() {
        VBox panel = new VBox(10);
        panel.setPrefWidth(320);
    }

```

```

panel.setMinWidth(280);
panel.setStyle("-fx-background-color: transparent;");
HBox.setHgrow(panel, Priority.NEVER);

HBox headerBox = createModeToggleBox();
searchField = createSearchField();
playlistView = createPlaylistView();
radioListView = createRadioListView();
HBox buttonBox = createFileButtonsBox();
VBox networkPanel = createNetworkPanel();

panel.getChildren().addAll(headerBox, searchField, playlistView, radioListView, buttonBox, networkPanel);
refreshPlaylist();
return panel;
}

private VBox createNetworkPanel() {
    VBox p = new VBox(5);
    p.setStyle("-fx-background-color: " + COLOR_BG_LIST + "; -fx-padding: 8; -fx-border-radius: 4; -fx-
background-radius: 4;");
    Label title = new Label("☐ Сетевой сервер");
    title.setStyle("-fx-text-fill: " + COLOR_BTN + "; -fx-font-weight: bold; -fx-font-size: 13px;");

    btnConnect = createButton("☐", "Подключиться");
    btnDisconnect = createButton("☐", "Отключиться");
    btnRefreshServer = createButton("☐", "Обновить список");
    btnDownload = createButton("↓☐", "Скачать в плейлист");

    applyButtonStyle(btnConnect);
    applyButtonStyle(btnDisconnect);
    applyButtonStyle(btnRefreshServer);
    applyButtonStyle(btnDownload);

    HBox buttonRow = new HBox(5, btnConnect, btnDisconnect, btnRefreshServer, btnDownload);
    buttonRow.setAlignment(Pos.CENTER);

    serverFilesListView = new ListView<>();
    serverFilesListView.setStyle(getListStyle());
    serverFilesListView.setPrefHeight(80);

    btnConnect.setOnAction(e -> {
        if(networkClient.connect("127.0.0.1", 8080))
            networkClient.requestFileList();
    });
    btnDisconnect.setOnAction(e -> networkClient.disconnect());
    btnRefreshServer.setOnAction(e -> networkClient.requestFileList());
    btnDownload.setOnAction(e -> {
        String sel = serverFilesListView.getSelectionModel().getSelectedItem();
        if(sel != null) networkClient.downloadFile(sel);
    });

    serverFilesListView.getSelectionModel().selectedItemProperty()
        .addListener((o,ov,nv) -> {
            if (btnDownload != null) btnDownload.setDisable(nv==null);
        });
    btnDownload.setDisable(true);

    p.getChildren().addAll(title, buttonRow, serverFilesListView);
    return p;
}

```

```

private HBox createModeToggleBox() {
    HBox headerBox = new HBox(10);
    headerBox.setAlignment(Pos.CENTER_LEFT);
    Label modeLabel = new Label("Режим:");
    modeLabel.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; -fx-text-fill: " + COLOR_BTN + ";");
    modeToggleGroup = new ToggleGroup();
    playlistModeToggle = createModeToggleButton("🎵", "Плейлист", true);
    radioModeToggle = createModeToggleButton("📻", "Радио", false);
    modeToggleGroup.selectedToggleProperty().addListener((obs, oldToggle, newToggle) -> {
        if (newToggle == playlistModeToggle) toggleRadioMode(false);
        else if (newToggle == radioModeToggle) toggleRadioMode(true);
    });
    headerBox.getChildren().addAll(modeLabel, new Region(), playlistModeToggle, radioModeToggle);
    HBox.setHgrow(headerBox, Priority.ALWAYS);
    return headerBox;
}

private ToggleButton createModeToggleButton(String text, String tooltip, boolean selected) {
    ToggleButton btn = new ToggleButton(text);
    btn.setToggleGroup(modeToggleGroup);
    btn.setSelected(selected);
    applyToggleStyle(btn, selected);
    btn.setTooltip(new Tooltip(tooltip));
    return btn;
}

private TextField createSearchField() {
    TextField field = new TextField();
    field.setPromptText("Поиск песни...");
    field.setStyle("-fx-background-color: " + COLOR_BG_LIST + "; -fx-text-fill: " + COLOR_TEXT +
        "; -fx-prompt-text-fill: #888888; -fx-border-color: #4A235A; -fx-border-width: 1; " +
        "-fx-border-radius: 4; -fx-background-radius: 4; -fx-font-size: 13px;");
    return field;
}

private ListView<PlaylistItem> createPlaylistView() {
    FilteredList<PlaylistItem> filteredData = new FilteredList<>(masterPlaylistData, p -> true);
    searchField.textProperty().addListener((obs, oldVal, newVal) -> {
        filteredData.setPredicate(item -> {
            if (newVal == null || newVal.isEmpty()) return true;
            return item.getName().toLowerCase().contains(newVal.toLowerCase());
        });
    });
    ListView<PlaylistItem> view = new ListView<>(filteredData);
    view.setCellFactory(lv -> createPlaylistCell());
    view.setStyle(getListStyle());
    VBox.setVgrow(view, Priority.ALWAYS);
    view.setOnMouseClicked(event -> {
        if (event.getClickCount() == 2) {
            PlaylistItem selectedItem = view.getSelectionModel().getSelectedItem();
            if (selectedItem != null) {
                int selectedIndex = masterPlaylistData.indexOf(selectedItem);
                if (selectedIndex != -1) {
                    playSongAtIndex(selectedIndex);
                }
            }
        }
    });
    return view;
}

private ListCell<PlaylistItem> createPlaylistCell() {

```

```

return new ListCell<PlaylistItem>() {
    @Override
    protected void updateItem(PlaylistItem item, boolean empty) {
        super.updateItem(item, empty);
        if (item == null || empty) {
            setText(null);
            setGraphic(null);
        } else {
            HBox hbox = new HBox(10);
            hbox.setAlignment(Pos.CENTER_LEFT);
            ImageView icon = item.getIcon();
            icon.setFitWidth(32);
            icon.setFitHeight(32);
            icon.setPreserveRatio(true);
            Label nameLabel = new Label(item.getName());
            nameLabel.setTextFill(Color.WHITE);
            hbox.getChildren().addAll(icon, nameLabel);
            setGraphic(hbox);
            setText(null);
            setStyle(isSelected() ? "-fx-background-color: " + COLOR_BTN + ";" : "");
        }
    }
};
}

private ListView<RadioStation> createRadioListView() {
    ListView<RadioStation> view = new ListView<>(radioStations);
    view.setCellFactory(lv -> createRadioCell());
    view.setStyle(getListStyle());
    view.setVisible(false);
    view.setManaged(false);
    VBox.setVgrow(view, Priority.ALWAYS);
    view.getSelectionModel().selectedItemProperty().addListener((obs, oldVal, newVal) -> {
        if (newVal != null) {
            currentRadioIndex = radioStations.indexOf(newVal);
            handleRadioStationSelection(newVal);
        }
    });
    return view;
}

private ListCell<RadioStation> createRadioCell() {
    return new ListCell<RadioStation>() {
        @Override
        protected void updateItem(RadioStation item, boolean empty) {
            super.updateItem(item, empty);
            if (item == null || empty) {
                setText(null);
                setGraphic(null);
            } else {
                HBox hbox = new HBox(10);
                hbox.setAlignment(Pos.CENTER_LEFT);
                ImageView icon = item.getIconView();
                Label nameLabel = new Label(item.getName());
                nameLabel.setTextFill(Color.WHITE);
                hbox.getChildren().addAll(icon, nameLabel);
                setGraphic(hbox);
                setText(null);
                setStyle(isSelected() ? "-fx-background-color: " + COLOR_ACCENT + ";" : "");
            }
        }
    };
}

```

```

}

private String getListStyle() {
    return "-fx-background-color: " + COLOR_BG_LIST + "; -fx-control-inner-background: " + COLOR_BG_LIST +
        "; -fx-text-fill: " + COLOR_TEXT + "; -fx-border-color: #4A235A; -fx-border-width: 1;";
}

private HBox createFileButtonsBox() {
    HBox buttonBox = new HBox(5);
    buttonBox.setAlignment(Pos.CENTER);
    buttonBox.setMaxWidth(Double.MAX_VALUE);
    addFileButton = createButton("□", "Выбрать файл");
    addFolderButton = createButton("□", "Выбрать папку с музыкой");
    removeButton = createButton("□", "Удалить из списка");
    renameButton = createButton("✎", "Изменить имя файла");
    saveButton = createButton("□", "Сохранить плейлист в папку");
    applyButtonStyle(addFileButton);
    applyButtonStyle(addFolderButton);
    applyButtonStyle(removeButton);
    applyButtonStyle(renameButton);
    applyButtonStyle(saveButton);
    addFileButton.setOnAction(e -> addFileToPlaylist());
    addFolderButton.setOnAction(e -> addFolderToPlaylist());
    removeButton.setOnAction(e -> removeFromPlaylist());
    renameButton.setOnAction(e -> renameSelectedSong());
    saveButton.setOnAction(e -> savePlaylist());
    for (Button btn : List.of(addFileButton, addFolderButton, removeButton, renameButton, saveButton)) {
        HBox.setHgrow(btn, Priority.ALWAYS);
    }
    buttonBox.getChildren().addAll(addFileButton, addFolderButton, removeButton, renameButton, saveButton);
    return buttonBox;
}

public VBox createRightPanel() {
    VBox panel = new VBox(15);
    panel.setAlignment(Pos.CENTER);
    panel.setPadding(new Insets(10));
    panel.setStyle("-fx-background-color: transparent;");
    panel.setFillWidth(true);
    discAnimation.getDiscImageView().setStyle("-fx-background-color: transparent;");
    panel.getChildren().add(discAnimation.getDiscImageView());
    currentSongLabel = createSongLabel();
    panel.getChildren().add(currentSongLabel);
    timeBox = createTimeBox();
    panel.getChildren().add(timeBox);
    panel.getChildren().add(createPlaybackControls());
    panel.getChildren().add(createModeControls());
    panel.getChildren().add(createVolumeControl());
    setupTimelineUpdates();
    return panel;
}

private Label createSongLabel() {
    Label label = new Label("Нет выбранной песни");
    label.setStyle("-fx-font-size: 14px; -fx-font-weight: bold; -fx-text-fill: " + COLOR_BTN + ";");
    label.setWrapText(true);
    label.setAlignment(Pos.CENTER);
    label.setMaxWidth(600);
    return label;
}

private HBox createTimeBox() {

```

```

timeSlider = new Slider(0, 100, 0);
timeSlider.setPrefWidth(300);
timeSlider.setMinWidth(200);
timeSlider.setDisable(true);
timeSlider.setStyle("-fx-accent: " + COLOR_BTN + ";");
timeSlider.setOnMousePressed(e -> isUserDraggingTimeSlider = true);
timeSlider.setOnMouseDragged(e -> isUserDraggingTimeSlider = true);
timeSlider.setOnMouseReleased(e -> {
    isUserDraggingTimeSlider = false;
    seekToPosition();
});
currentTimeLabel = createTimeLabel("0:00");
totalTimeLabel = createTimeLabel("0:00");
HBox timeBox = new HBox(10);
timeBox.setAlignment(Pos.CENTER);
HBox.setHgrow(timeSlider, Priority.ALWAYS);
timeBox.getChildren().addAll(currentTimeLabel, timeSlider, totalTimeLabel);
return timeBox;
}

private Label createTimeLabel(String text) {
    Label label = new Label(text);
    label.setStyle("-fx-text-fill: " + COLOR_BTN + ";");
    return label;
}

private HBox createPlaybackControls() {
    HBox controlBox = new HBox(10);
    controlBox.setAlignment(Pos.CENTER);
    previousButton = createButton("◀", "Предыдущая");
    playButton = createButton("▶", "Воспроизвести");
    pauseButton = createButton("⏸", "Пауза");
    nextButton = createButton("⏭", "Следующая");
    applyButtonStyle(previousButton);
    applyButtonStyle(playButton);
    applyButtonStyle(pauseButton);
    applyButtonStyle(nextButton);
    previousButton.setOnAction(e -> playPrevious());
    playButton.setOnAction(e -> playCurrent());
    pauseButton.setOnAction(e -> pause());
    nextButton.setOnAction(e -> playNext());
    controlBox.getChildren().addAll(previousButton, playButton, pauseButton, nextButton);
    return controlBox;
}

private HBox createModeControls() {
    HBox modeBox = new HBox(10);
    modeBox.setAlignment(Pos.CENTER);

    shuffleButton = new ToggleButton("⇄");
    shuffleButton.setTooltip(new Tooltip("Случайный порядок"));
    shuffleButton.setSelected(false);
    updateModeButtonStyle(shuffleButton, false, 1.0);
    shuffleButton.setOnAction(e -> toggleShuffle());

    repeatButton = new ToggleButton("🔄");
    repeatButton.setTooltip(new Tooltip("Повторять песню"));
    repeatButton.setSelected(false);
    updateModeButtonStyle(repeatButton, false, 1.0);
    repeatButton.setOnAction(e -> toggleRepeat());

    modeBox.getChildren().addAll(shuffleButton, repeatButton);
}

```

```

    return modeBox;
}

private VBox createVolumeControl() {
    VBox volumeBox = new VBox(5);
    volumeBox.setAlignment(Pos.CENTER);
    volumeBox.setFillWidth(true);
    Label volumeLabel = new Label("Громкость:");
    volumeLabel.setStyle("-fx-text-fill: " + COLOR_BTN + ";");
    volumeSlider = new Slider(0, 100, 50);
    volumeSlider.setPrefWidth(250);
    volumeSlider.setMinWidth(150);
    volumeSlider.setStyle("-fx-accent: " + COLOR_BTN + ";");
    volumePercentLabel = new Label("50%");
    volumePercentLabel.setStyle("-fx-text-fill: " + COLOR_BTN + "; -fx-font-weight: bold;");
    volumeSlider.valueProperty().addListener((obs, oldVal, newVal) -> {
        audioPlayer.setVolume(newVal.doubleValue() / 100.0);
        volumePercentLabel.setText(Math.round(newVal.doubleValue()) + "%");
    });
    HBox volumeControls = new HBox(10);
    volumeControls.setAlignment(Pos.CENTER);
    HBox.setHgrow(volumeSlider, Priority.ALWAYS);
    volumeControls.getChildren().addAll(volumeSlider, volumePercentLabel);
    volumeBox.getChildren().addAll(volumeLabel, volumeControls);
    return volumeBox;
}

private void applyToggleStyle(ToggleButton btn, boolean isSelected) {
    String bgColor = isSelected ? COLOR_ACCENT : COLOR_BTN;
    String textColor = isSelected ? "white" : COLOR_TEXT;
    btn.setStyle("-fx-background-color: " + bgColor + "; -fx-text-fill: " + textColor +
        "; -fx-font-size: 14px; -fx-font-weight: bold; -fx-border-radius: 4; -fx-background-radius: 4; " +
        "-fx-min-width: 50px; -fx-pref-width: 50px; -fx-min-height: 35px; -fx-pref-height: 35px;");
}

private void toggleRadioMode(boolean isRadio) {
    isRadioModeActive = isRadio;
    audioPlayer.stop();
    discAnimation.stopRotation();
    discAnimation.loadAlbumArt(null);
    setPlayingState(false);
    currentTimeLabel.setText("0:00");
    totalTimeLabel.setText("0:00");
    timeSlider.setValue(0);
    currentSongLabel.setText(isRadio ? "Потоковое вещание" : "Нет выбранной песни");
    playlistView.setVisible(!isRadio);
    playlistView.setManaged(!isRadio);
    searchField.setVisible(!isRadio);
    searchField.setManaged(!isRadio);
    radioListView.setVisible(isRadio);
    radioListView.setManaged(isRadio);
    timeBox.setVisible(!isRadio);
    timeBox.setManaged(!isRadio);
    shuffleButton.setDisable(isRadio);
    repeatButton.setDisable(isRadio);
    applyToggleStyle(playlistModeToggle, !isRadio);
    applyToggleStyle(radioModeToggle, isRadio);
}

private void handleRadioStationSelection(RadioStation station) {
    if (station != null) {
        audioPlayer.playRadioStream(station.getStreamUrl());
    }
}

```

```

        currentSongLabel.setText(station.getName());
        discAnimation.loadRadioIcon(station.getIconView().getImage());
        discAnimation.startRotation();
        setPlayingState(true);
    }
}

private Button createButton(String text, String tooltipText) {
    Button btn = new Button(text);
    Tooltip tooltip = new Tooltip(tooltipText);
    tooltip.setStyle("-fx-background-color: #2b2b2b; -fx-text-fill: #ffffff; -fx-font-size: 12px; " +
        "-fx-padding: 8px; -fx-background-radius: 4px;");
    btn.setTooltip(tooltip);
    return btn;
}

private void applyButtonStyle(Button btn) {
    String baseStyle = "-fx-background-color: " + COLOR_BTN + "; -fx-text-fill: " + COLOR_TEXT +
        "; -fx-border-radius: 4; -fx-background-radius: 4; -fx-font-weight: bold; -fx-font-size: 16px; " +
        "-fx-min-width: 50px; -fx-pref-width: 50px; -fx-min-height: 50px; -fx-pref-height: 50px;";
    btn.setStyle(baseStyle);
    btn.setOnMouseEntered(e -> {
        if (!btn.getStyle().contains(COLOR_ACCENT)) {
            String current = btn.getStyle();
            String toReplace = "-fx-background-color: " + COLOR_BTN + ";";
            if (current.contains(toReplace)) {
                btn.setStyle(current.replace(toReplace, "-fx-background-color: " + COLOR_BTN_HOVER + ";"));
            }
        }
    });
    btn.setOnMouseExited(e -> {
        if (!btn.getStyle().contains(COLOR_ACCENT)) {
            String current = btn.getStyle();
            String toReplace = "-fx-background-color: " + COLOR_BTN_HOVER + ";";
            if (current.contains(toReplace)) {
                btn.setStyle(current.replace(toReplace, "-fx-background-color: " + COLOR_BTN + ";"));
            }
        }
    });
}

private void addFileToPlaylist() {
    File addedFile = FileUtils.addFileWithDuplicateHandling(stage, playlistManager);
    if (addedFile != null) {
        loadAlbumArtFromMp3(addedFile);
        refreshPlaylist();
    }
}

private void addFolderToPlaylist() {
    List<File> addedFiles = FileUtils.addFolderWithDuplicateHandling(stage, playlistManager);
    if (!addedFiles.isEmpty()) {
        for (File file : addedFiles) loadAlbumArtFromMp3(file);
        refreshPlaylist();
    }
}

private void removeFromPlaylist() {
    int index = playlistView.getSelectionModel().getSelectedIndex();
    if (index != -1) {
        boolean wasCurrent = playlistManager.removeFile(index);
        if (wasCurrent) {

```

```

        audioPlayer.stop();
        discAnimation.stopRotation();
        discAnimation.loadAlbumArt(null);
        currentSongLabel.setText("Нет выбранной песни");
        setPlayingState(false);
    }
    refreshPlaylist();
}
}

private void renameSelectedSong() {
    int index = playlistView.getSelectionModel().getSelectedIndex();
    if (index != -1) {
        String currentName = playlistManager.getPlaylist().get(index);
        TextInputDialog dialog = new TextInputDialog(currentName);
        dialog.setTitle("Переименовать");
        dialog.setHeaderText("Введите новое имя:");
        dialog.showAndWait().ifPresent(newName -> {
            if (!newName.isEmpty()) {
                if (!newName.toLowerCase().endsWith(".mp3")) newName += ".mp3";
                playlistManager.renameFile(index, newName);
                refreshPlaylist();
            }
        });
    }
}

private void savePlaylist() {
    if (playlistManager.getPlaylist().isEmpty()) {
        showAlert("Ошибка", "Плейлист пуст!");
        return;
    }
    TextInputDialog dialog = new TextInputDialog("MyPlaylist");
    dialog.setTitle("Сохранить плейлист");
    dialog.setHeaderText("Введите имя для папки плейлиста:");
    dialog.showAndWait().ifPresent(playlistName -> {
        if (!playlistName.isEmpty()) {
            try {
                File directory = FileUtils.chooseDirectory(stage);
                if (directory != null) {
                    String path = playlistManager.savePlaylistWithFiles(playlistName, directory.getAbsolutePath());
                    showAlert("Успех", "Плейлист сохранён в:\n" + path);
                }
            } catch (Exception e) {
                showAlert("Ошибка", "Не удалось сохранить: " + e.getMessage());
            }
        }
    });
}

private void playCurrent() {
    if (isRadioModeActive) {
        if (audioPlayer.isRadioMode() && audioPlayer.isPaused()) {
            audioPlayer.resume();
            discAnimation.startRotation();
            setPlayingState(true);
        } else if (!radioStations.isEmpty()) {
            handleRadioStationSelection(radioStations.get(currentRadioIndex));
        }
    } else {
        if (audioPlayer.getMediaPlayer() != null && audioPlayer.isPaused()) {
            audioPlayer.resume();
        }
    }
}

```

```

        discAnimation.startRotation();
        setPlayingState(true);
    } else {
        if (playlistManager.getCurrentFilePath() == null && !masterPlaylistData.isEmpty()) {
            playlistManager.setCurrentIndex(0);
        }
        String path = playlistManager.getCurrentFilePath();
        if (path != null) {
            discAnimation.loadAlbumArt(new File(path));
            audioPlayer.playCurrentSong();
            discAnimation.startRotation();
            updateCurrentSongLabel();
            highlightCurrentSongInList();
            setPlayingState(true);
        }
    }
}

private void pause() {
    audioPlayer.pause();
    discAnimation.stopRotation();
    setPlayingState(false);
}

private void playNext() {
    if (isRadioModeActive) {
        if (radioStations.isEmpty()) return;
        currentRadioIndex = (currentRadioIndex + 1) % radioStations.size();
        radioListView.getSelectionModel().select(currentRadioIndex);
        handleRadioStationSelection(radioStations.get(currentRadioIndex));
    } else {
        int nextIndex = playlistManager.getNextIndex();
        if (nextIndex != -1) {
            playlistManager.setCurrentIndex(nextIndex);
            String path = playlistManager.getCurrentFilePath();
            if (path != null) discAnimation.loadAlbumArt(new File(path));
            audioPlayer.playCurrentSong();
            discAnimation.startRotation();
            updateCurrentSongLabel();
            highlightCurrentSongInList();
            setPlayingState(true);
        }
    }
}

private void playPrevious() {
    if (isRadioModeActive) {
        if (radioStations.isEmpty()) return;
        currentRadioIndex = (currentRadioIndex - 1 + radioStations.size()) % radioStations.size();
        radioListView.getSelectionModel().select(currentRadioIndex);
        handleRadioStationSelection(radioStations.get(currentRadioIndex));
    } else {
        int prevIndex = playlistManager.getPreviousIndex();
        if (prevIndex != -1) {
            playlistManager.setCurrentIndex(prevIndex);
            String path = playlistManager.getCurrentFilePath();
            if (path != null) discAnimation.loadAlbumArt(new File(path));
            audioPlayer.playCurrentSong();
            discAnimation.startRotation();
            updateCurrentSongLabel();
            highlightCurrentSongInList();
        }
    }
}

```

```

        setPlayingState(true);
    }
}

private void playSongAtIndex(int index) {
    playlistManager.setCurrentIndex(index);
    audioPlayer.playCurrentSong();
    discAnimation.loadAlbumArt(new File(playlistManager.getCurrentFilePath()));
    discAnimation.startRotation();
    updateCurrentSongLabel();
    setPlayingState(true);
}

private void setPlayingState(boolean playing) {
    isPlaying = playing;
    if (playing) {
        updateButtonScale(playButton, currentScale);
        updateButtonScale(pauseButton, currentScale);
    } else {
        updateButtonScale(pauseButton, currentScale);
        updateButtonScale(playButton, currentScale);
    }
}

private void toggleShuffle() {
    if (isRadioModeActive) return;
    boolean newState = !playlistManager.isShuffle();
    playlistManager.setShuffle(newState);

    if (newState && playlistManager.isRepeat()) {
        playlistManager.setRepeat(false);
        repeatButton.setSelected(false);
        updateModeButtonStyle(repeatButton, false, currentScale);
    }

    shuffleButton.setSelected(newState);
    updateModeButtonStyle(shuffleButton, newState, currentScale);
}

private void toggleRepeat() {
    if (isRadioModeActive) return;
    boolean newState = !playlistManager.isRepeat();
    playlistManager.setRepeat(newState);

    if (newState && playlistManager.isShuffle()) {
        playlistManager.setShuffle(false);
        shuffleButton.setSelected(false);
        updateModeButtonStyle(shuffleButton, false, currentScale);
    }

    repeatButton.setSelected(newState);
    updateModeButtonStyle(repeatButton, newState, currentScale);
}

private void updateModeButtonStyle(ToggleButton btn, boolean isActive, double scale) {
    double size = 50 * scale;
    double fontSize = 16 * scale;
    String bgColor = isActive ? COLOR_ACCENT : COLOR_BTN;

    btn.setMinSize(size, size);
    btn.setPrefSize(size, size);
}

```

```

btn.setMaxSize(size, size);

btn.setStyle("-fx-background-color: " + bgColor + "; -fx-text-fill: white; " +
    "-fx-border-radius: 4; -fx-background-radius: 4; -fx-font-weight: bold; -fx-font-size: " +
    fontSize + "px;");

btn.setOnMouseEntered(null);
btn.setOnMouseExited(null);
}

private void seekToPosition() {
    MediaPlayer player = audioPlayer.getMediaPlayer();
    if (player != null && player.getTotalDuration() != null) {
        player.seek(Duration.millis(timeSlider.getValue()));
    }
}

private void setupTimelineUpdates() {
    audioPlayer.setOnEndOfMedia(() -> {
        if (!playlistManager.isRepeat() && !audioPlayer.isRadioMode()) {
            playNext();
        }
    });
    audioPlayer.setOnPlayerReady(player -> {
        if (player != null && !audioPlayer.isRadioMode()) {
            player.totalDurationProperty().addListener((obs, oldDur, newDur) -> {
                if (newDur != null) {
                    timeSlider.setMin(0);
                    timeSlider.setMax(newDur.toMillis());
                    timeSlider.setDisable(false);
                    totalTimeLabel.setText(formatTime(newDur));
                }
            });
            player.currentTimeProperty().addListener((obs, oldTime, newTime) -> {
                if (!isUserDraggingTimeSlider && player.getTotalDuration() != null) {
                    timeSlider.setValue(newTime.toMillis());
                    currentTimeLabel.setText(formatTime(newTime));
                }
            });
        }
    });
}

private void updateCurrentSongLabel() {
    String path = playlistManager.getCurrentFilePath();
    if (path != null) currentSongLabel.setText(new File(path).getName());
}

private String formatTime(Duration duration) {
    if (duration == null) return "0:00";
    int minutes = (int) duration.toMinutes();
    int seconds = (int) (duration.toSeconds() % 60);
    return String.format("%d:%02d", minutes, seconds);
}

private void showAlert(String title, String message) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle(title);
    alert.setHeaderText(null);
    alert.setContentText(message);
    alert.showAndWait();
}

```

```

private Image createDefaultIconImage() {
    Canvas canvas = new Canvas(32, 32);
    GraphicsContext gc = canvas.getGraphicsContext2D();
    gc.setFill(Color.rgb(50, 50, 70));
    gc.fillRect(0, 0, 32, 32);
    return canvas.snapshot(null, null);
}

private Image loadAlbumArtFromMp3(File mp3File) {
    if (mp3File == null || !mp3File.exists()) return createDefaultIconImage();
    try {
        Mp3File mp3file = new Mp3File(mp3File);
        if (mp3file.hasId3v2Tag()) {
            ID3v2 tag = mp3file.getId3v2Tag();
            byte[] data = tag.getAlbumImage();
            if (data != null && data.length > 0) {
                return new Image(new ByteArrayInputStream(data), 32, 32, true, true);
            }
        }
    } catch (Exception e) {
    }
    return createDefaultIconImage();
}

private void refreshPlaylist() {
    masterPlaylistData.clear();
    for (int i = 0; i < playlistManager.getPlaylist().size(); i++) {
        String fileName = playlistManager.getPlaylist().get(i);
        String filePath = playlistManager.getFilePath(i);
        if (filePath == null) continue;
        File file = new File(filePath);
        Image iconImage = albumIconCache.computeIfAbsent(filePath, k -> loadAlbumArtFromMp3(file));
        masterPlaylistData.add(new PlaylistItem(fileName, new ImageView(iconImage)));
    }
}

private void highlightCurrentSongInList() {
    int index = playlistManager.getCurrentIndex();
    if (index >= 0 && index < masterPlaylistData.size()) {
        playlistView.getSelectionModel().select(index);
        playlistView.scrollTo(index);
    }
}

public void setupScaling(javafx.scene.Scene scene, Stage stage) {
    scene.widthProperty().addListener((obs, oldVal, newVal) ->
        scaleUI(newVal.doubleValue(), scene.getHeight()));
    scene.heightProperty().addListener((obs, oldVal, newVal) ->
        scaleUI(scene.getWidth(), newVal.doubleValue()));
    scaleUI(scene.getWidth() > 0 ? scene.getWidth() : 1000,
        scene.getHeight() > 0 ? scene.getHeight() : 700);
}

private void scaleUI(double width, double height) {
    double scale = Math.min(width / 1000.0, height / 700.0);
    scale = Math.max(0.8, Math.min(1.5, scale));
    currentScale = scale;

    double discSize = 200 * scale;
    discAnimation.getDiscImageView().setFitWidth(discSize);
    discAnimation.getDiscImageView().setFitHeight(discSize);
}

```

```

        double fontSize = 14 * scale;
        currentSongLabel.setStyle("-fx-font-size: " + fontSize + "px; -fx-font-weight: bold; -fx-text-fill: " + COLOR_BTN
+ ";");
        currentSongLabel.setMaxWidth(width * 0.5);
        volumePercentLabel.setStyle("-fx-text-fill: " + COLOR_BTN + "; -fx-font-weight: bold; -fx-font-size: " +
(fontSize * 0.8) + "px;");

        updateButtonScale(addFileButton, scale);
        updateButtonScale(addFolderButton, scale);
        updateButtonScale(removeButton, scale);
        updateButtonScale(renameButton, scale);
        updateButtonScale(saveButton, scale);
        updateButtonScale(previousButton, scale);
        updateButtonScale(playButton, scale);
        updateButtonScale(pauseButton, scale);
        updateButtonScale(nextButton, scale);

        updateModeButtonStyle(shuffleButton, playlistManager.isShuffle(), scale);
        updateModeButtonStyle(repeatButton, playlistManager.isRepeat(), scale);

        if (isPlaying) setPlayingState(true);
    }

    private void updateButtonScale(Button btn, double scale) {
        if (btn == null) return;
        double size = 50 * scale;
        double fontSize = 16 * scale;
        String currentStyle = btn.getStyle();

        String bgColor = COLOR_BTN;
        String textColor = COLOR_TEXT;
        if (currentStyle.contains(COLOR_ACCENT)) {
            bgColor = COLOR_ACCENT;
            textColor = "white";
        } else if (currentStyle.contains(COLOR_BTN_HOVER)) {
            bgColor = COLOR_BTN_HOVER;
        }

        btn.setMinSize(size, size);
        btn.setPrefSize(size, size);
        btn.setMaxSize(size, size);

        btn.setStyle("-fx-background-color: " + bgColor + "; -fx-text-fill: " + textColor +
            "; -fx-border-radius: 4; -fx-background-radius: 4; -fx-font-weight: bold; -fx-font-size: " +
            fontSize + "px;");
    }

    private static class PlaylistItem {
        private final ImageView icon;
        private final String name;
        public PlaylistItem(String name, ImageView icon) {
            this.name = name;
            this.icon = icon;
        }
        public ImageView getIcon() { return icon; }
        public String getName() { return name; }
    }
}

```

Приложение Б. UML-диаграммы

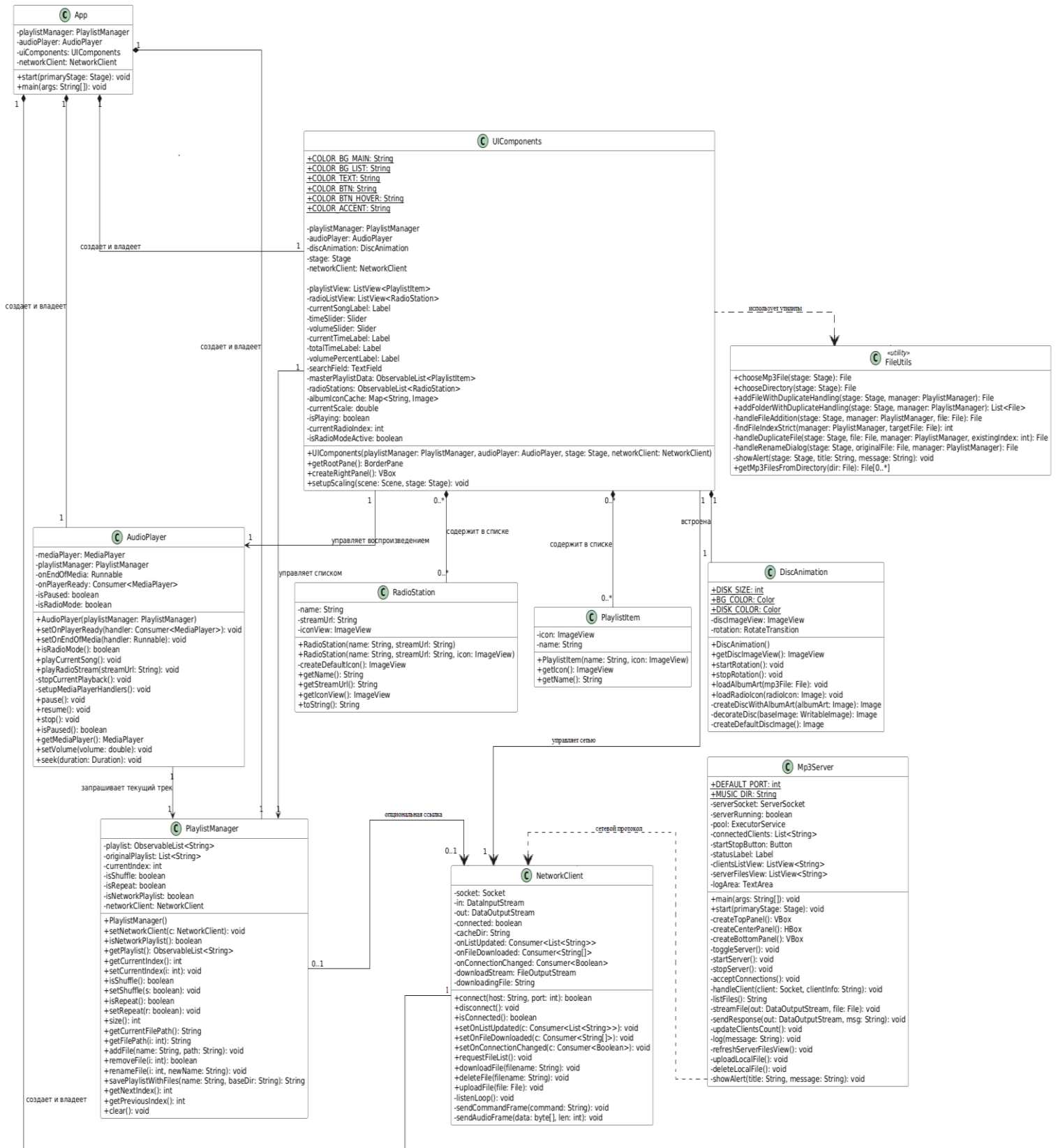


Рисунок 8 – диаграмма классов

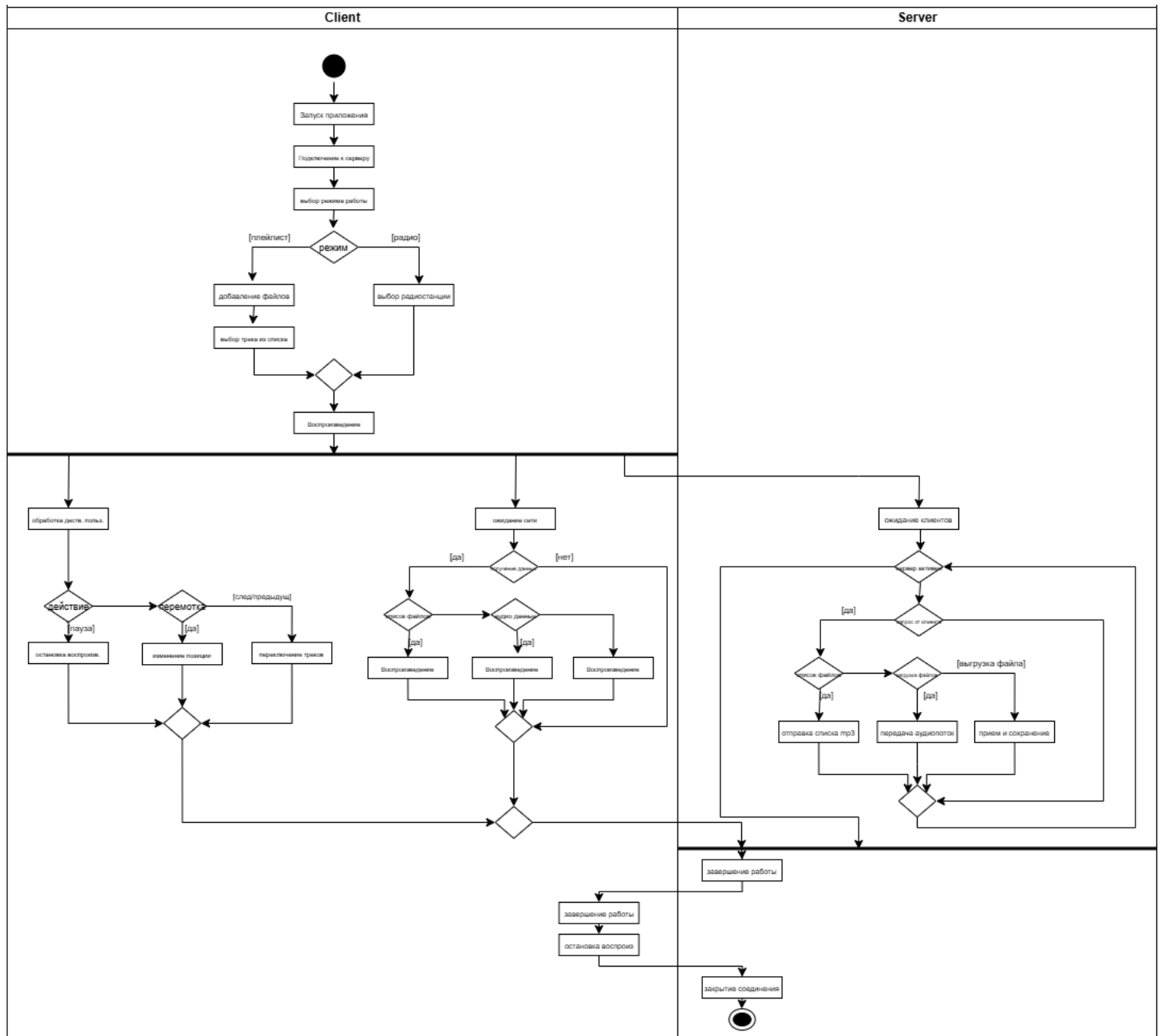


Рисунок 9 – диаграмма деятельности



Рисунок 10 – диаграмма вариантов использования

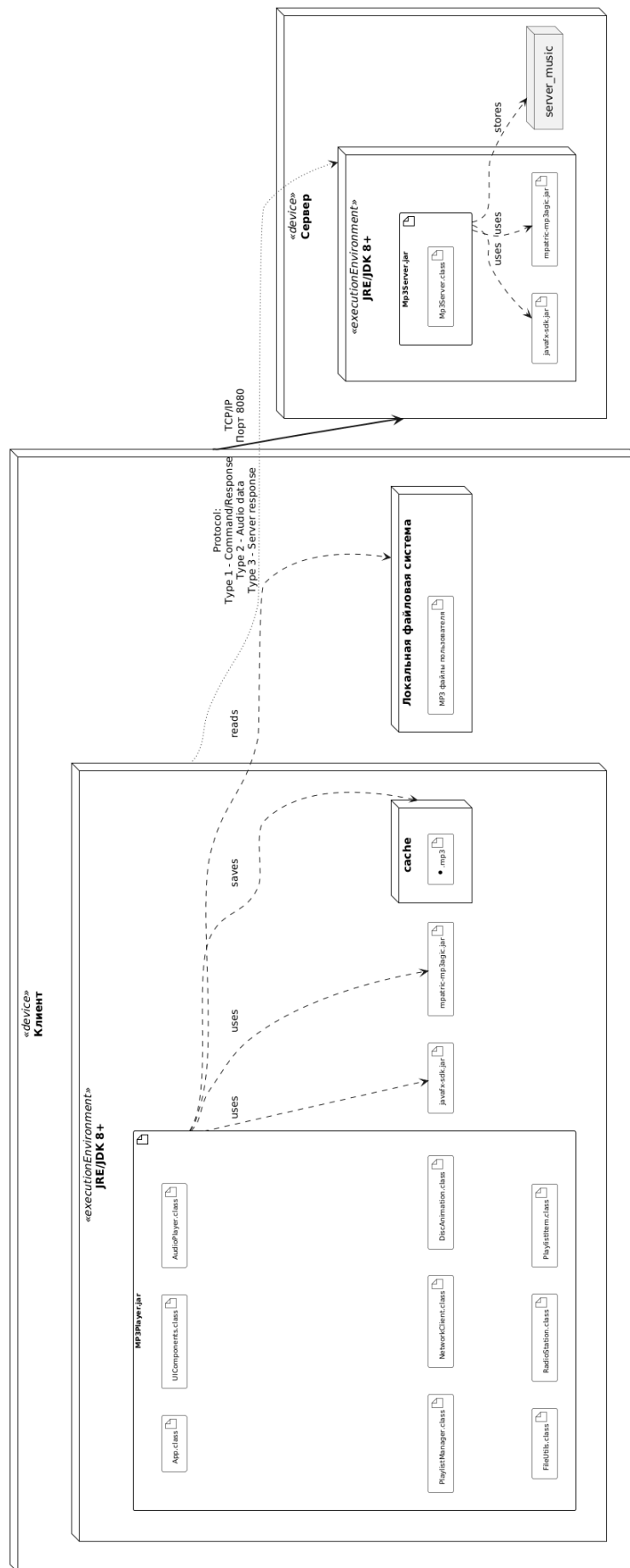
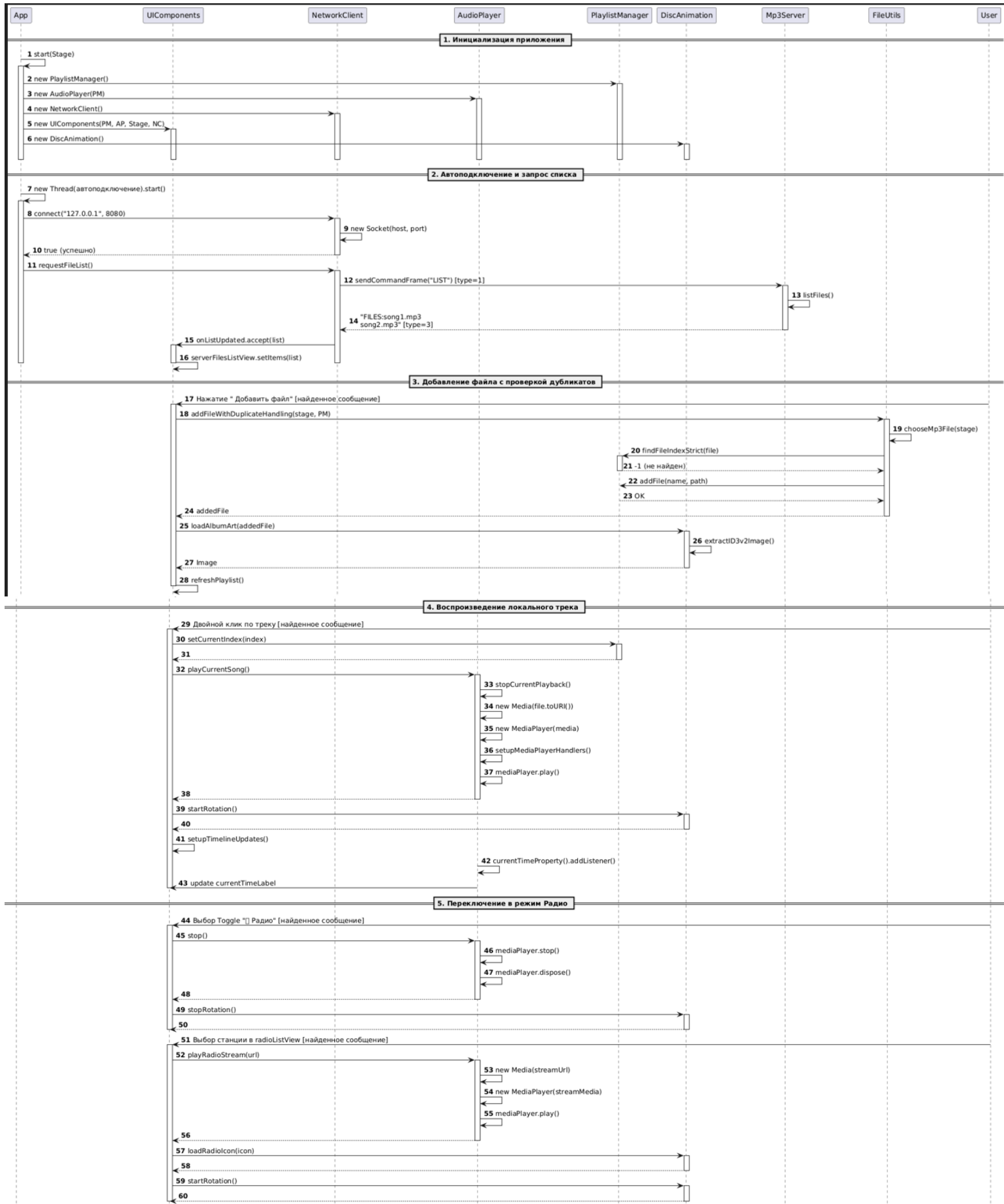


Рисунок 11 – диаграмма развёртывания



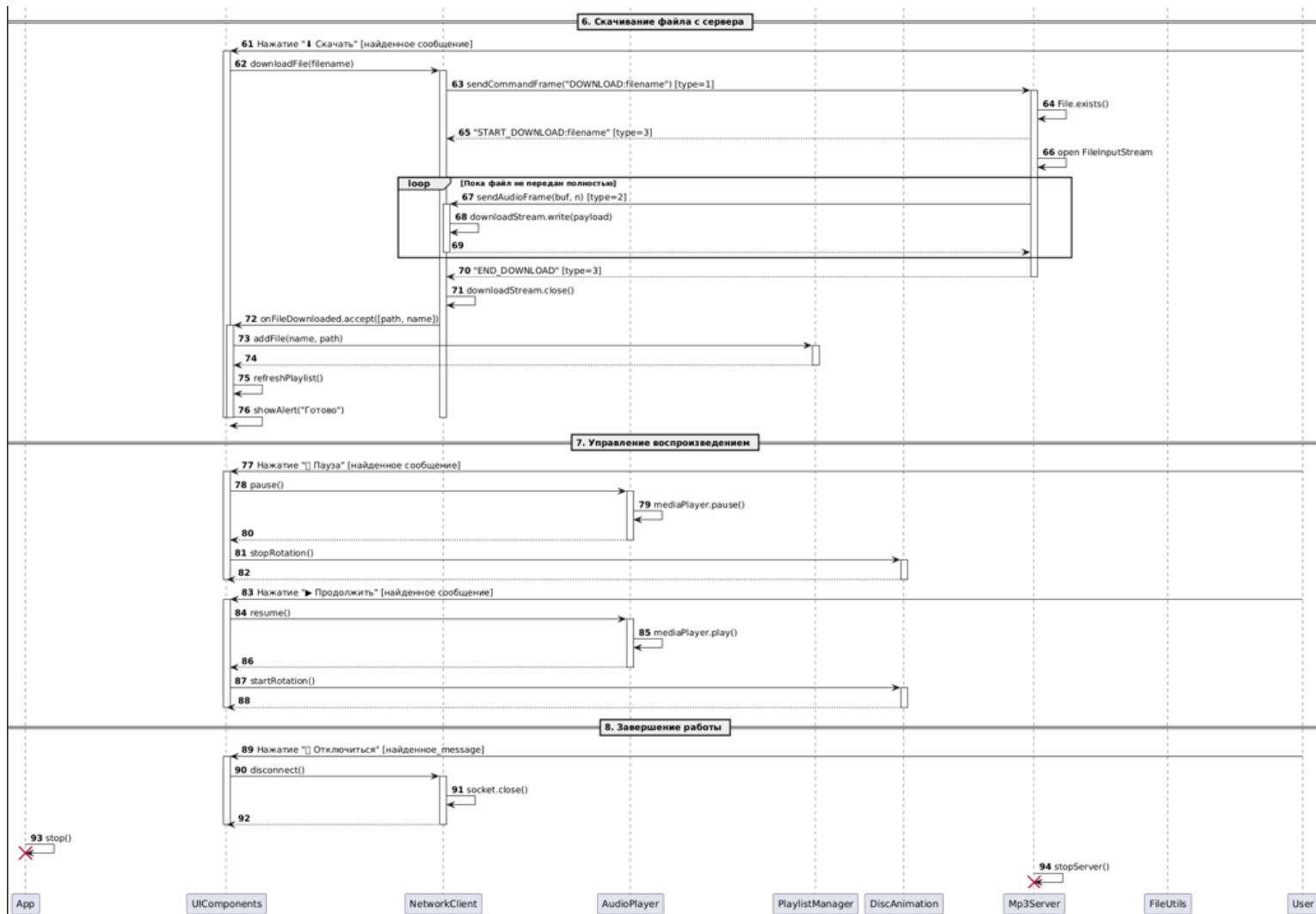


Рисунок 12 – диаграмма последовательности