СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Потоковое мультимедиа - мультимедиа, которое непрерывно получает пользователь от провайдера потокового вещания. Это понятие применимо как к информации, распространяемой через телекоммуникации, так и к информации, которая изначально распространялась посредством потокового вещания (например, радио, телевидение) или не потоковой (например, книги, видеокассеты, аудио CD).

За последние два десятилетия развитие информационных систем сделали потоковую мультимедийную информацию доступной широкому кругу простых пользователей. Различные интернет-сервисы предлагают пользователям возможность просмотра видео без скачивания.

В технологической основе системы лежит три элемента:

- Генератор видеопотока (либо из списка видео файлов, либо прямой оцифровкой с видеокарты, либо копируя существующий в сети поток) и направляет его серверу.
- Сервер - принимает видеопоток и перенаправляет его копии всем подключённым к серверу клиентам, по сути является репликатором данных.
- Клиент - принимает видеопоток от сервера и преобразует его в видеосигнал, который и видит пользователь.

Серверы могут передавать различающиеся по форматам видеоданные, например, MP4, TS, MKV.

В качестве клиента можно использовать любой видеоплеер, поддерживающий потоковое видео и способный декодировать формат, в котором вещает сервер.

Мультимедиа потоки бывают двух видов: по запросу или живыми. Потоки информации, вызываемой по запросу пользователя, хранятся на серверах продолжительный период времени. Живые потоки доступны короткий период времени, например, при передаче видео со спортивных соревнований.

Примерно в 2002 году интерес к единому унифицированному потоковому формату и широкое распространение Adobe Flash поспособствовали разработке

формата потокового видео через Flash, который использовался в Flash-проигрывателях, которые размещались на многих популярных видеохостингах, таких как YouTube, сегодня потоковые мультимедиа по умолчанию проигрываются в формате HTML5 видео, которые заменили Flash-проигрыватели.

1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

1.1 Анализ существующих аналогов

1.1.1 Программное средство «IceS» [1]

Icecast - свободное ПО для организации потокового цифрового аудио и видеовещания.

Icecast является серверной программой, которая может осуществлять раздачу цифровых потоков различных форматов, таких как MP4, TS, M3U, M3U8.

Передача данных осуществляется по стандартному протоколу HTTP, либо по протоколу SHOUTcast.

Icecast является аналогом программы SHOUTcast компании Nullsoft, однако имеет более развитую функциональность и поддерживает большее количество форматов потоков. При передаче данных Icecast поддерживает теги в UTF-8, что позволяет организовывать трансляцию с русской аннотацией.

Достоинства:

- Сервис поддерживает все распространенные аудиоформаты
- Сервис работает в Debian GNU/Linux, под управлением которых работает большое количество серверов.
- Программное обеспечение распространяется бесплатно и постоянно обновляется.
- Имеется свой протокол передачи данных, что положительно сказывается на качестве трансляций.
- Широкая языковая поддержка интерфейса позволяет «безбаръерно» пользоваться приграммным обеспечением носителям различных языковых категорий.

Недостатки:

- Долгий и кропотливый процесс настройки серверной части стриминговой платформы.
- Отсутствие поддержки операционной системы Windows.
- Необходимость работы с связке с дополнительными программами для обеспечения нормальной работы сервера.
- Необходимо наличие производительного компьютера для ведения трансляции.
- Невозможность ведения трансляции с мобильного устройства.
- Может быть необходимо подключение к удаленному серверу, что ставит под сомнение саму идею трансляции в локальной сети.

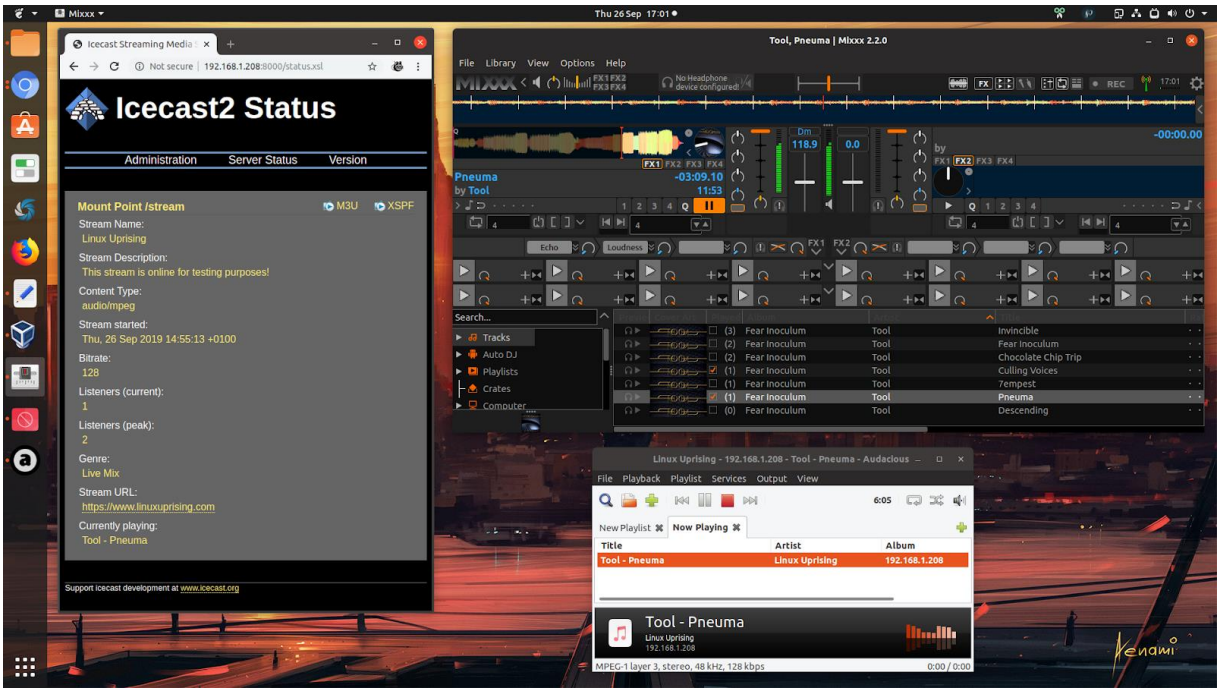


Рисунок 1.1 – Интерфейс программы «Icecast»

1.1.2 Программное средство «Shoutcast» [2]

Shoutcast - кроссплатформенное бесплатное серверное ПО компании Nullsoft. Предназначено для организации потокового вещания цифрового аудио/видео сигнала в формате MP4, MKV, TS, M3U, M3U8 как в локальную сеть, так и в Интернет. Прост в установке, настройке и управлении.

Минимальный набор для организации вещания состоит из собственно сервера Shoutcast (или демона), и источника, в простейшем случае реализуемого программой Winamp или аналогичного программного плеера с соответствующим плагином, кодирующим аудиосигнал с требуемым качеством, и передающего его в потоковом режиме серверу Shoutcast, для последующей передачи оного подключившимся к серверу клиентам, в качестве которых выступает любой программный плеер с поддержкой потокового вещания.

В настоящее время существует большое число интернет-радиостанций, вещающих с использованием данного ПО.

Имеется панель управления на основе Web-интерфейса. Имеется возможность на одном компьютере установить одновременно несколько служб сервера Shoutcast.

Достоинства:

- Прост в установке и использовании.
- Имеется веб-версия приложение.
- Приложение можно установить на многие популярные операционные системы.
- Есть возможность ведения трансляции в локальной сети.
- Поддерживается большое количество форматов кодирования видеофайлов.
- Передача видеопотока ведется по HTTP.

Недостатки:

- Отсутствует мобильное приложение.
- Отсутствие бесплатного варианта сервиса и значительная ограниченность сравнительно «дешевых» вариантов в плане пропускной способности, количества одновременных слушателей и размера серверного хранилища.
- Отсутствует широкая языковая поддержка, что ограничивает пользование клиентов, не владеющих английским на уровне понимания интерфейса программного средства.

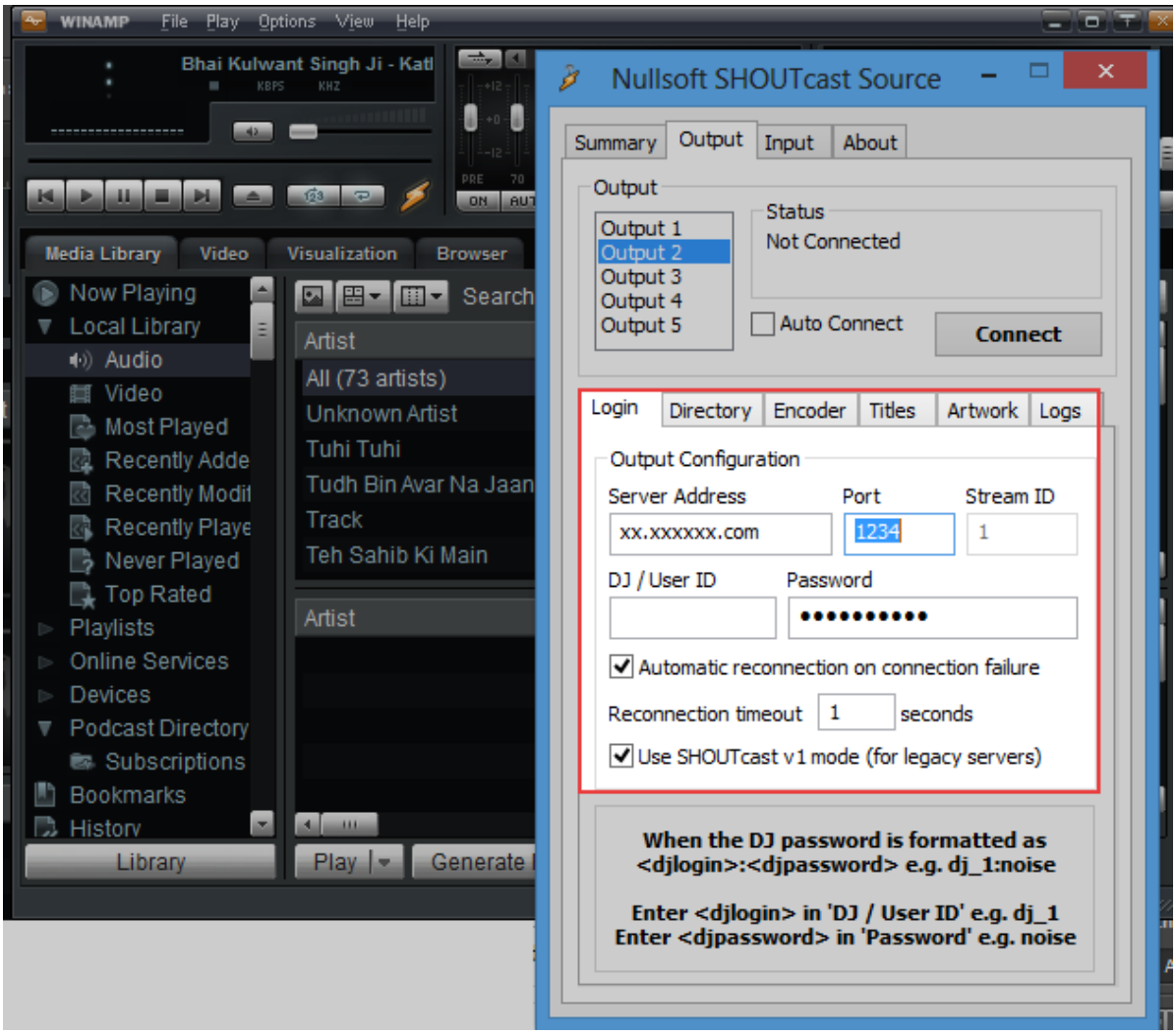


Рисунок 1.2 – Интерфейс программы «Shoutcast»

1.1.3 Программное средство «Nginx» [3]

Nginx - веб-сервер и почтовый прокси-сервер, работающий на Unix-подобных операционных системах (FreeBSD, OpenBSD, Linux, Solaris, macOS, AIX и HP-UX). Также присутствует поддержка Windows, то в экспериментальном режиме.

В nginx рабочие процессы обслуживают одновременно множество соединений, мультиплексируя их вызовами операционной системы select, epoll (Linux) и kqueue (FreeBSD). Рабочие процессы выполняют цикл обработки событий от дескрипторов. Полученные от клиента данные разбираются с помощью конечного автомата. Разобранный запрос последовательно обрабатывается цепочкой модулей, задаваемой конфигурацией. Ответ клиенту формируется в буферах, которые хранят данные либо в памяти, либо указывают на отрезок файла. Буфера объединяются в цепочки, определяющие последовательность, в которой данные будут переданы клиенту. Если операционная система поддерживает эффективные операции ввода-вывода, такие как writev и sendfile, то nginx применяет их по возможности.

Достоинства:

- Прост в установке и использовании.
- Приложение можно установить на многие популярные операционные системы.
- Есть возможность ведения трансляции по протоколам, использующим HTTP и HTTPS.
- Поддерживается большое количество форматов кодирования видеофайлов.
- Полностью бесплатный

Недостатки:

- Отсутствует мобильное приложение.
- Отсутствует графический интерфейс без сторонних модификаций.
- Для гибкой настройки требуется хорошо понимать структуру сервера

1.2 Постановка задачи

После анализа прототипов были сформированы следующие требование к проектированному программному средству:

- Сервер должен иметь возможность передачи данных более чем 1 клиенту
- Возможность быстрого использования программного средства без затрат времени на авторизацию/регистрацию
- Сервер должен предоставлять список возможных к просмотру видео
- Пользователь должен иметь возможность смотреть несколько видеопотоков одновременно

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Транспортный протокол TCP

Transmission Control Protocol (TCP) - это один из основных протоколов, предназначенный для управления передачей данных в сетях и подсетях TCP/IP. Обеспечивает гарантированную доставку как одиночных сообщений, так и потоков данных, сборку фрагментов, квитирование, повторную передачу пакета и установку приоритетов. Выполняет функции протокола транспортного уровня в стеке протоколов TCP/IP. TCP поддерживает только однонаправленную связь и отправку из 1 узла в 1 узел. Это также называется связью «точка-точка»

Бит/Смещение	0 - 3	4 - 9	10 - 15	16 - 31
0	Порт источника			Порт назначения
32	Номер последовательности (порядковый номер)			
64	Номер подтверждения			
96	Длина заголовка	Зарезервировано	Флаги	Размер окна
128	Контрольная сумма			Указатель срочных данных
160	Опции			
160/192+	Данные			

Рисунок 2.1 – Структура пакета TCP

Взаимодействие партнеров с использованием протокола TCP строится в три этапа: установление логического соединения, обмен данными, закрытие соединения.

Перед началом передачи полезных данных, TCP позволяет убедиться в том, что получатель существует, слушает нужный отправителю порт и готов принимать данные для этого устанавливается соединение при помощи механизма трёхстороннего рукопожатия, схема которого приведена ниже.

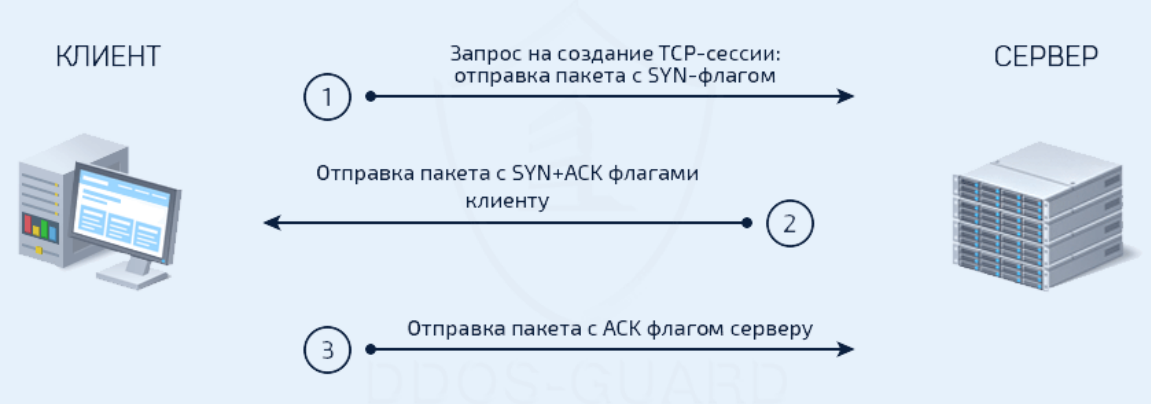


Рисунок 2.2 – Схема TCP соединения

Далее, в рамках соединения передаются пользовательские данные. В большинстве случаев каждый TCP-сегмент пересылается в одной IP-дейтаграмме. Однако при необходимости TCP будет расщеплять сегменты на несколько IP-дейтаграмм, вмещающихся в физические кадры данных, которые используют для передачи информации между компьютерами в сети. Поскольку IP не гарантирует, что дейтаграммы будут получены в той же самой последовательности, в которой они были посланы, TCP осуществляет повторную сборку TCP-сегментов на другом конце маршрута, чтобы образовать непрерывный поток данных.

При пересылке данных используется понятие окна TCP. Окно TCP – это общий объем данных, которые могут быть отправлены и не подтверждены. Размер окна изменяется динамически прямо во время работы и зависит от условий сети. Чем больше размер окна, тем больший объём информации будет передан до получения подтверждения. Для надёжных сетей подтверждения можно присылать редко, чтобы не добавлять трафика, поэтому размер окна в таких сетях автоматически увеличивается. Если же TCP видит, что данные теряются, размер окна автоматически уменьшается.

После завершения передачи соединение закрывается, тем самым получатель извещается о том, что данных больше не будет, а отправитель извещается о том, что получатель извещён.

Отличие упорядоченного закрытия соединения от его разрыва заключается в

том, что при упорядоченном закрытии соединения данные, находящиеся в пути, будут ожидаться пользователем и только потом соединение будет закрыто. При разрыве соединения все недошедшие данные теряются.

2.2 Коммуникационный протокол HLS

HLS (HTTP Live Streaming) - коммуникационный протокол для потоковой передачи медиа на основе HTTP, разработанный компанией Apple как часть программного обеспечения QuickTime, Safari, OS X и iOS. В основе работы лежит принцип разбиения цельного потока на небольшие фрагменты, последовательно скачиваемые по HTTP. Поток непрерывен и теоретически может быть бесконечным. В начале сессии скачивается плей-лист в формате M3U8, содержащий метаданные об имеющихся вложенных потоках.

Серверная часть кодирует и оборачивает входящее медиа в подходящий для доставки формат. Далее материал готовится к распределению путём сегментирования. Медиа сегментируется на фрагменты (чанки, chunks) и индексный файл (плей-лист). Кодировка: видео кодируется в формате H.264 и аудио в MP3, HE-AAC или AC-3. Всё это вкладывается в транспортный поток MPEG-2 для последующей доставки. Сегментирование: контент в MPEG-2 TS разделяется на фрагменты одинаковой длины, записанные в файлы .ts. Также создаётся индексный файл, содержащий ссылки на фрагменты или другие индексные файлы - он сохраняется как файл .m3u8. Работая как стандартный веб-сервер, сервер принимает запросы от клиентов и доставляет всё необходимое для воспроизведения.

Клиент запрашивает и скачивает все файлы, собирая их воедино так, чтобы предоставить пользователю непрерывный поток видео. Клиентское ПО скачивает первый индексный файл через URL и далее несколько доступных файлов медиа. ПО для проигрывания собирает всё в последовательность для воспроизведения.

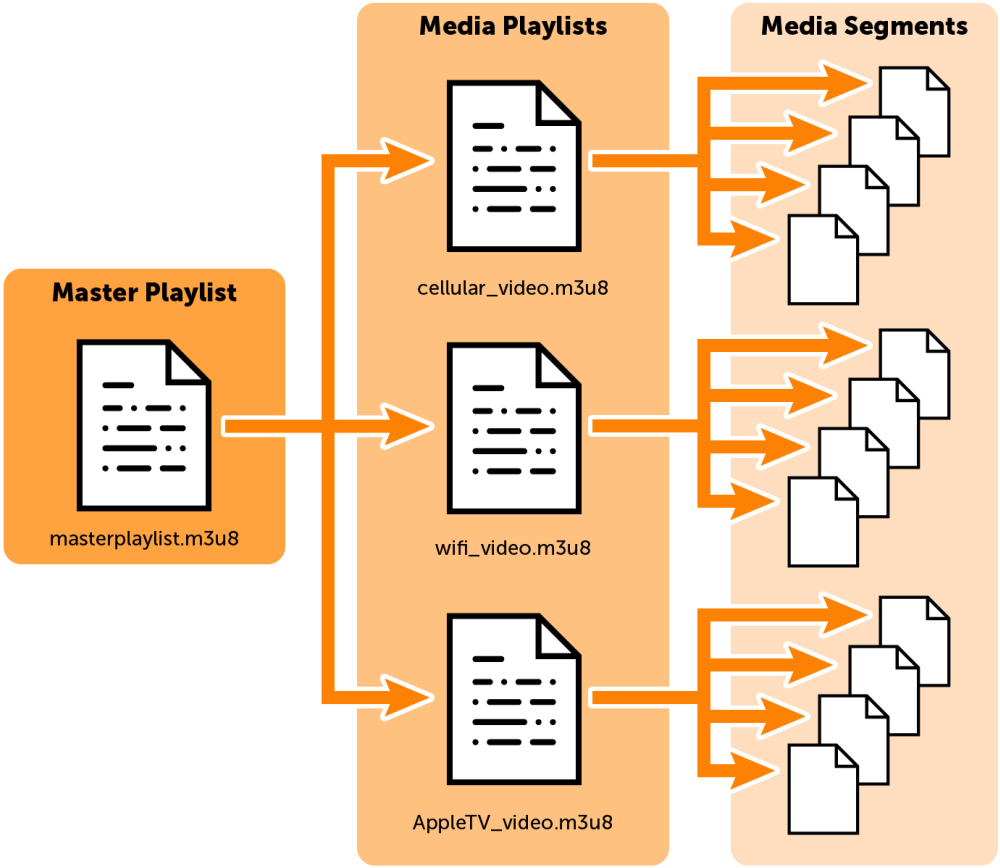


Рисунок 2.3 – Схема работы протокола HLS

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Приложение состоит их 2 частей: серверной и клиентской.

Серверная часть для обработки подключения использует встроенные классы ServerSocket и Socket. Класс FileServerThread используется для считывания HTTP сообщения из сокета с помощью класса HTTPRequest, обработки запроса с помощью класса APIHandler, генерации ответа на запрос с помощью класса HTTPResponse.

Клиентская часть для построения интерфейса и создания запросов использует классы HLSClient и HTTPRequest. Класс Media используется для получения фрагментов видео с сервера.

3.1 Класс ServerSocket [4]

Этот класс реализует сокеты сервера. Серверный сокет ожидает поступления запросов по сети. Он выполняет некоторую операцию на основе этого запроса, а затем, возможно, возвращает результат запрашивающей стороне.

Фактическая работа сокета сервера выполняется экземпляром класса SocketImpl. Приложение может изменить фабрику сокетов, которая создает реализацию сокетов, чтобы настроить себя для создания сокетов, соответствующих локальному брандмауэру.

Конструктор класса ServerSocket создает сокет сервера и привязывает его к указанному номеру локального порта с указанным задним числом. Номер порта 0 означает, что номер порта автоматически выделяется, как правило, из временного диапазона портов. Этот номер порта можно затем получить, вызвав getLocalPort.

Максимальная длина очереди для индикации входящих соединений (запрос на соединение) задается параметром backlog. Если индикация соединения появляется, когда очередь заполнена, соединение отклоняется.

Если приложение указало фабрику сокетов сервера, вызывается метод

createSocketImpl этой фабрики для создания фактической реализации сокета. В противном случае создается «простой» сокет.

Если есть менеджер безопасности, его метод checkListen вызывается с номером порта в качестве аргумента, чтобы убедиться, что операция разрешена. Это может привести к SecurityException. Аргумент backlog представляет собой запрошенное максимальное количество ожидающих соединений в сокете. Его точная семантика зависит от конкретной реализации. В частности, реализация может устанавливать максимальную длину или может игнорировать параметр altogther. Предоставленное значение должно быть больше 0. Если оно меньше или равно 0, то будет использоваться конкретное для реализации значение по умолчанию.

Из «главных» методов класса можно выдлеить методы accept, bind и close.

Метод accept прослушивает соединение с этим сокетом и принимает его. Метод блокируется, пока не будет установлено соединение.

Метод bind связывает объект ServerSocket с определенным адресом (IP-адресом и номером порта).

Аргумент backlog представляет собой запрошенное максимальное количество ожидающих соединений в сокете. Его точная семантика зависит от конкретной реализации. В частности, реализация может устанавливать максимальную длину или может игнорировать параметр altogther. Предоставленное значение должно быть больше 0. Если оно меньше или равно 0, то будет использоваться конкретное для реализации значение по умолчанию.

Метод close закрывает эту сокет. Любой поток, в настоящее время заблокированный в accept(), выдаст исключение SocketException.

3.2 Класс Socket [5]

Этот класс реализует клиентские сокеты (также называемые просто «сокетами»). Сокет является конечной точкой для связи между двумя машинами.

Фактическая работа сокета выполняется экземпляром класса SocketImpl. Приложение, изменяя фабрику сокетов, которая создает реализацию сокетов, может настроить себя на создание сокетов, соответствующих локальному брандмауэру. Создает сокет потока и подключает его к указанному номеру порта на указанном хосте. Если указанный хост имеет значение NULL, это эквивалентно указанию адреса как InetAddress.getByName (NULL). Другими словами, это эквивалентно указанию адреса интерфейса обратной связи. Если приложение указало фабрику сокетов сервера, вызывается метод createSocketImpl этой фабрики для создания фактической реализации сокета. В противном случае создается «простой» сокет.

Если есть менеджер безопасности, его метод checkConnect вызывается с адресом хоста и портом в качестве аргументов. Это может привести к SecurityException.

Рассмотрим метды bind, connect и close.

Метод bind связывает сокет с локальным адресом.

Метод close Закрывает сокет. Любой поток, в настоящее время заблокированный в операции ввода-вывода на этом сокете, вызовет исключение SocketException.

Как только сокет был закрыт, он не доступен для дальнейшего использования в сети (т.е. не может быть повторно подключен или восстановлен).

Закрытие этого сокета также закроет сокет InputStream и OutputStream.

Метод connect подключает этот сокет к серверу с указанным значением времени ожидания. Тайм-аут, равный нулю, интерпретируется как бесконечный тайм-аут. Соединение будет блокироваться до тех пор, пока не будет установлено или не возникнет ошибка.

3.3 Класс Media [6]

Класс Media представляет медиаресурс. Он создается из строковой формы исходного URI. Информация о мультимедиа, такая как длительность, метаданные, дорожки и разрешение видео, может быть получена из экземпляра Media. Медиа-

информация получается асинхронно и поэтому не обязательно доступна сразу после создания экземпляра класса. Однако вся информация должна быть доступна, если экземпляр был связан с MediaPlayer и этот проигрыватель перешел в статус MediaPlayer.Status.READY. Чтобы получать уведомления при добавлении метаданных или треков, наблюдатели могут быть зарегистрированы в коллекциях, возвращаемых getMetadata () и getTracks () соответственно.

Один и тот же объект Media может совместно использоваться несколькими объектами MediaPlayer. Такой общий экземпляр может управлять одной копией исходных данных мультимедиа, которая будет использоваться всеми проигрывателями, или может потребовать отдельной копии данных для каждого проигрывателя.

3.4 Класс FileServerThread

Этот класс реализует основной функционал сервера. В первую очередь принимается информация о сокете, из которого требуется считать HTTP запрос с использованием класса HTTPRequest. После чего используется класс APIHandler для обработки полученного запроса и генерации HTTPResponse ответа. Также конструктор класса принимает путь к папке сервера. Именно в этой папке сервер будет искать запрошенные файлы.

3.5 Класс HTTPRequest

В конструктор класса передается объект класса DataInputStream. Он используется для считывания HTTP запроса их сокета.

HTTP сообщение имеет следующую структуру:

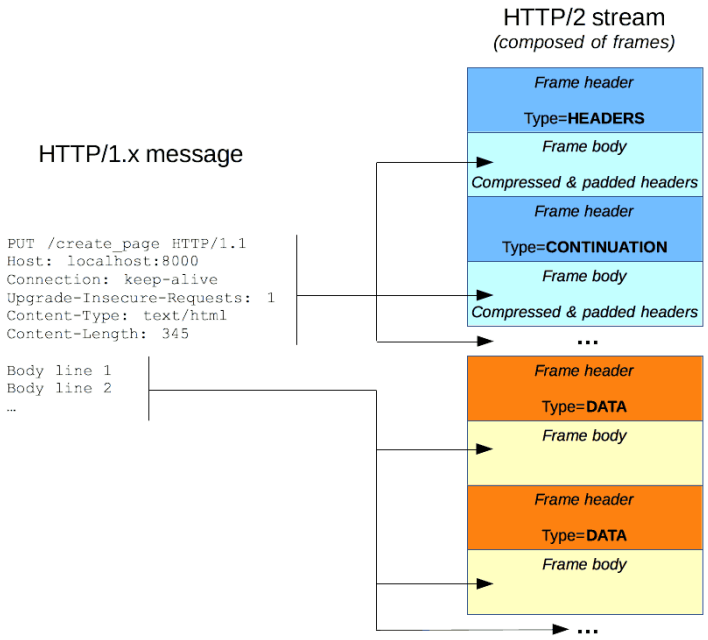


Рисунок 3.1 – Структура HTTP сообщения

Каждая строка заголовка всегда закачивается на \r\n. После всех заголовков идет строка, содержащая только \r\n, то есть пустая.

После пустой строки могут находиться какие-то переданные данные. Чтобы это определить, надо прочитать все заголовки в поисках либо “Content-length:100”, где 100 означает, что требуется прочитать 100 байт данных, либо “Transfer-encoding: chunked”, что означает отправку данных чанками.

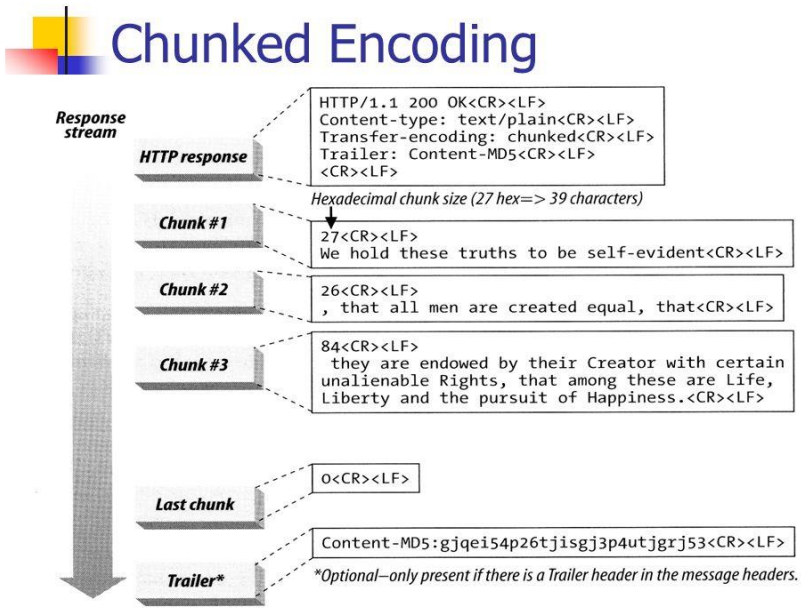


Рисунок 3.2 – Структура данных HTTP сообщения, отправленных чанками

Каждый чанк начинается с числа в 16-ричной системе счисления. Это чисто означает длину чанка. Последний чанк всегда имеет длину 0 и означает конец сообщения.

3.6 Класс HTTPResponse

Этот класс предназначен для формирования ответа на полученный запрос. В конструктор передается код ответа. По этому коду формируется строковый ответ. Например, для 200 – OK. После чего есть возможность записать в ответ какие-то данные. Также на основе этих данных будет автоматически создан заголовок “Content-length” и “Content-type”, если возможно.

Есть возможность самостоятельно создать нужные заголовки. Класс также поддерживает формирование сообщения из DataInputBuffer, что позволяет читать ответ от другого сервера или из какого-то другого источника.

3.7 Класс APIHandler

Данный класс предназначен для обработки полученного сервером запроса. В нем есть возможность определить действия для каждого типа HTTP запроса (GET, POST, PUT, HEAD и так далее). Для HLS сервера требуется определить действия для метода GET для отправки фрагментов выбранного видео. Также должна быть поддержка CORS.

Cross-Origin Resource Sharing (CORS) - механизм, использующий дополнительные HTTP-заголовки, чтобы дать возможность агенту пользователя получать разрешения на доступ к выбранным ресурсам с сервера на источнике (домене), отличном от того, что сайт использует в данный момент. Говорят, что агент пользователя делает запрос с другого источника (cross-origin HTTP request), если источник текущего документа отличается от запрашиваемого ресурса доменом, протоколом или портом.

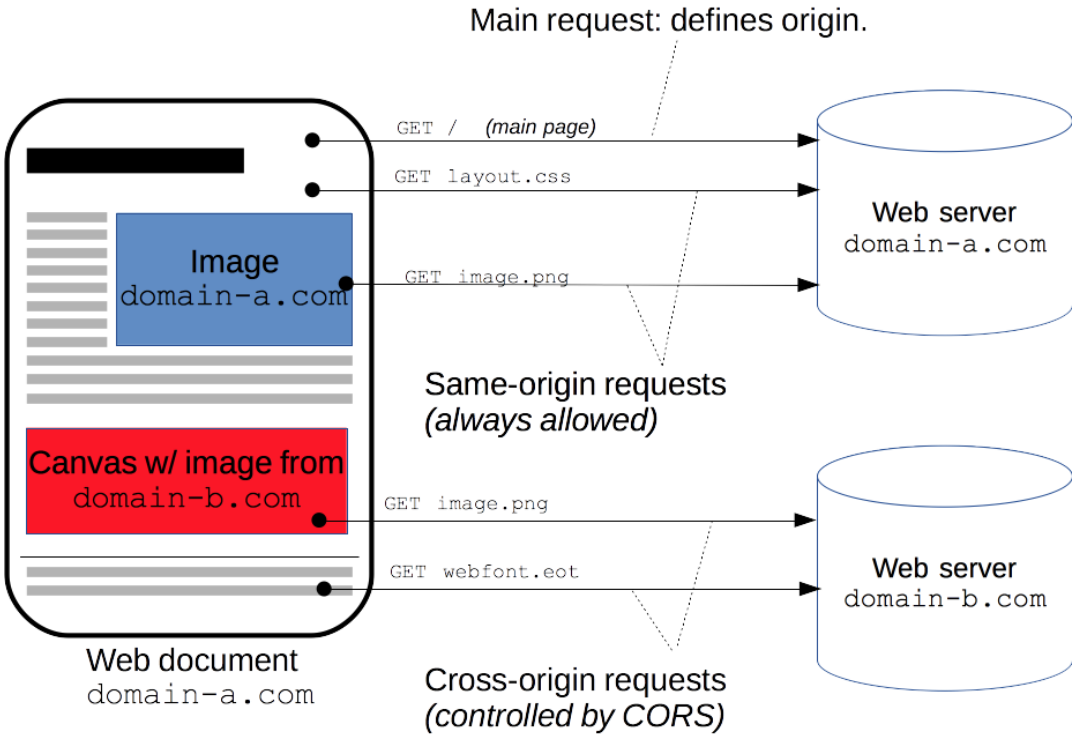


Рисунок 3.3 – Схема работы CORS

Для поддержки сервером CORS добавляем следующие заголовки к каждому ответу:

- Access-Control-Allow-Origin: *
- Access-Control-Allow-Headers: Origin, X-Requested-With, Content-Type, Accept, Authorization
- Access-Control-Allow-Credentials: true
- Access-Control-Allow-Methods: GET

3.8 Класс HLSClient

Данный класс предназначен для регулирования работы клиента с сервером. Конструктор принимает адрес сервера, на котором лежат видео для стриминга. Если подключиться не удалось, то выдается ошибка.

При успешном подключении с сервера запрашивается содержимое папки с видео в формате XML. После получения содержимого применяется XML парсер и создается массив из названий видео с их адресом.

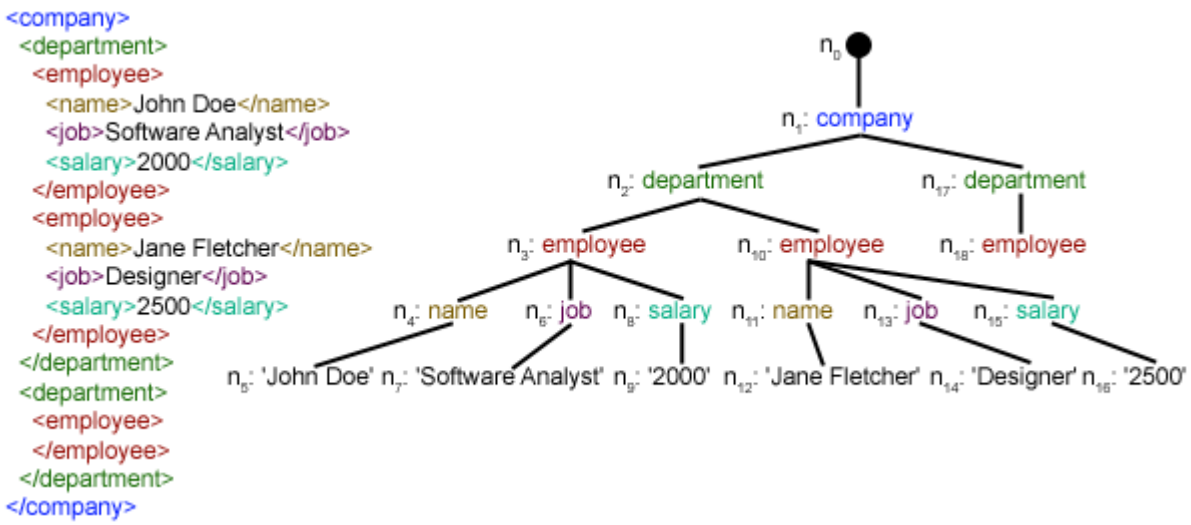


Рисунок 3.4 – Пример XML файла

При обращении к определенному видео файлу HLSClient создает объект Media, что позволяет его использовать для создания видеоплеера. Также перед каждой попыткой просмотра видео проверяется, существует ли файл на сервере, так как возможны ситуации удаления файла с сервера, что не отразится интерфейсом клиента автоматически.

4 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1 Тестирование сервера

Сервер должен уметь отправлять запрашиваемые файлы клиенту и формировать структуру запрошенной папки в XML. Сервер для тестирования запущен по адресу localhost:8080.

4.1.1 Отправка файла

Сервер отправит файл в том случае, если получит GET запрос с путем, указывающим на существующий файл. В папку серверных файлов был изначально помещен файл ffmpeg_script.txt со следующим содержимым: “ffmpeg -i input.mp4 -profile:v baseline -level 3.0 -s 640x360 -start_number 0 -hls_time 10 -hls_list_size 0 -f hls index.m3u8”. Проверим получение данного файла с помощью утилиты curl. Команда для выполнения выглядит следующим образом: “curl -v localhost:8080 ffmpeg_script.txt”. С помощью нее будет отправлен GET запрос на сервер с целью получения файла.

Рисунок 4.1 – Результат работы команды

В ответ был получен файл с указанным содержимым. Тест пройден.

4.1.2 Отправка структуры папки

При попытке обратиться к существующей папке сервер должен прочитать

содержимое папки и сформировать XML документ, отражающий структуру требуемой папки.

С помощью утилиты curl получим содержимое корневой папки сервера. Команда: “curl -v localhost:8080”.

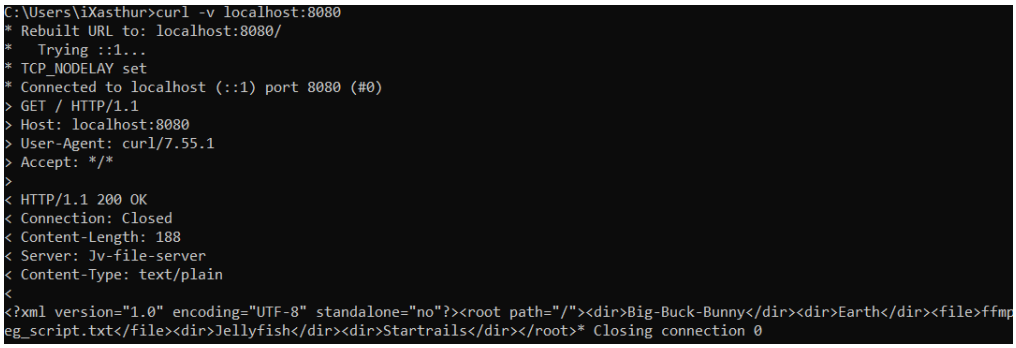


Рисунок 4.2 – Результат работы команды

В ответ была получена структура папки в XML. Тест пройден.

4.1.3 Обращение к несуществующему файлу/папке

При обращении к несуществующему файлу или папке сервер должен вернуть код 404, что позволит понять, что по запрашиваемому пути ничего не существует.

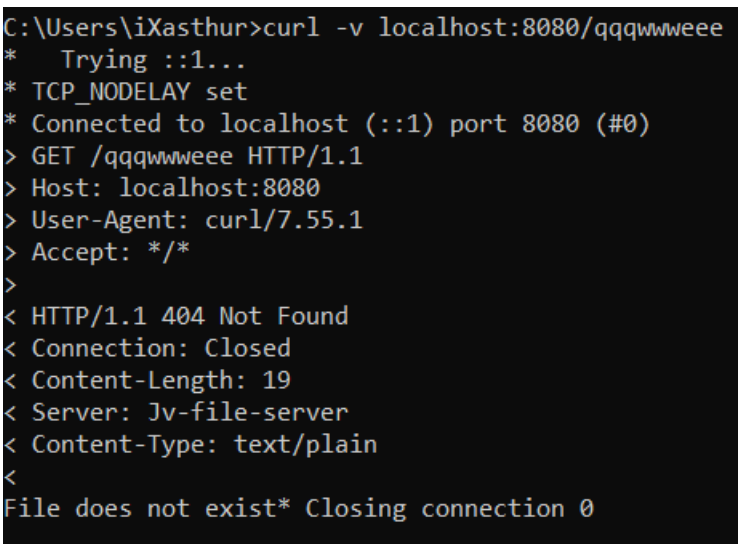


Рисунок 4.3 – Результат работы команды

В ответ был получен код 404. Тест пройден.

4.2 Тестирование клиент

Клиент должен отображать информацию об ошибке при неудачном подключении и обрабатывать ситуацию возможного удаления файла с сервера. Сервер так же запущен по адресу localhost:8080.

4.2.1 Попытка подключения к несуществующему адресу

При попытке подключиться к localhostzzzz:8081 должна быть выведена на экран ошибка о невозможности присоединения.

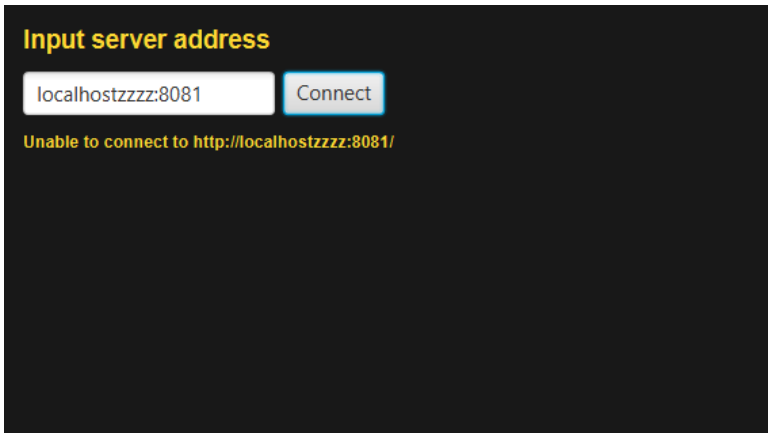


Рисунок 4.4 – Результат неуспешного подключения

Тест пройден.

4.2.2 Подключение к серверу и получение списка видео

При попытке подключения к localhost:8080 клиент должен выдать сообщение об успешном подключении и вывести на экран список доступных на сервере видео.

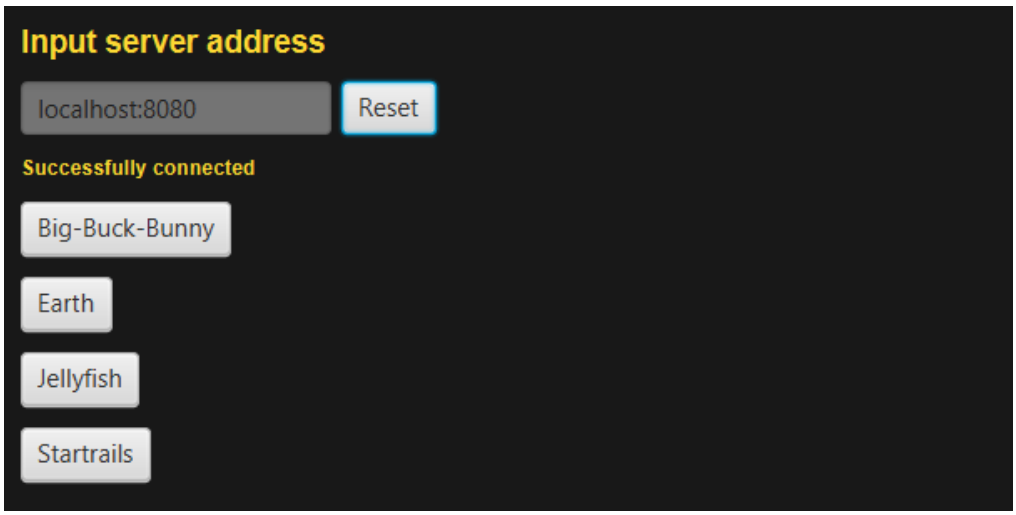


Рисунок 4.5 – Результат успешного подключения

Тест пройден.

4.2.3 Попытка просмотра удаленного видео

После успешного подключения к серверу из серверной папки было удалено видео “Jellyfish”.

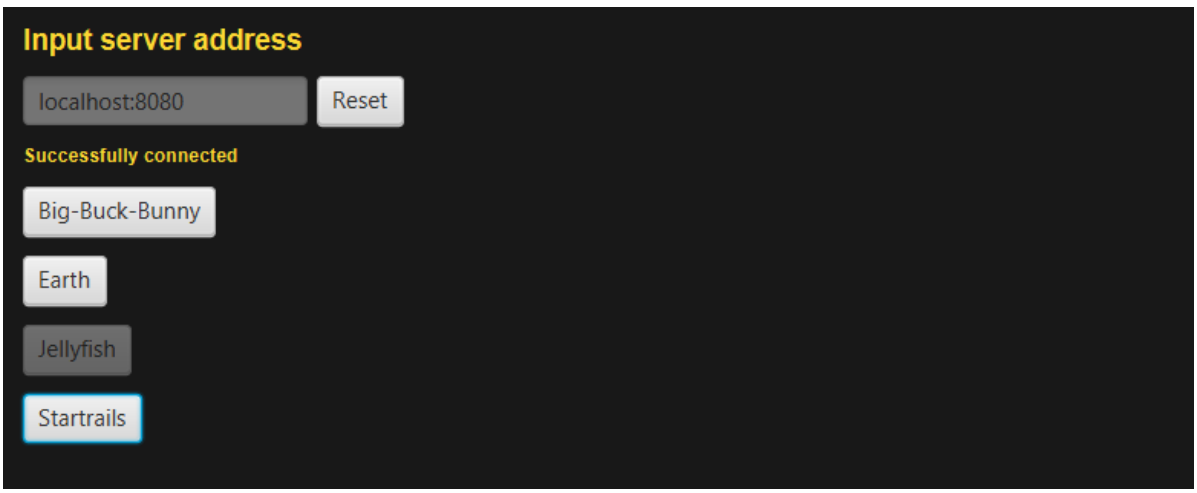


Рисунок 4.6 – Результат попытки открыть удаленный файл

Кнопка стала неактивна и плеер не был открыт. Тест пройден.

4.2.4 Попытка просмотра видео

Для теста будут открыты все видео. Будут открыты сразу все возможные видео. На экране должно появиться 4 окна с идущими видео

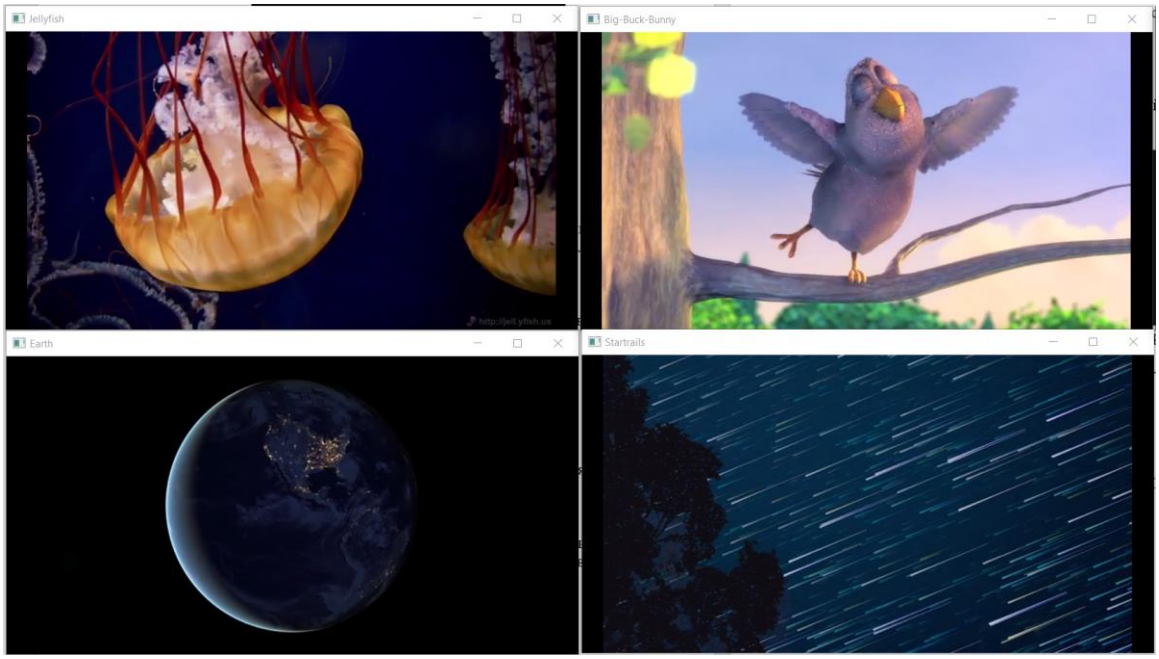


Рисунок 4.7 – Результат попытки открыть все видео

Появилось 4 окна с идущими видео. Тест пройден.

5 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

5.1 Руководство по установке

Для запуска сервера и клиента на компьютере требуется:

1. Установить Java с официального сайта.
2. Установить JavaFX, если скачанная версия Java не содержит данную библиотеку изначально.
3. Доступ к интернету, если требуется подключаться к серверу не из локальной сети, а из сети Интернет.

5.2Руководство по использованию

5.2.1 Запуск сервера

Для запуска сервера требуется запустить исполняемый файл со следующими аргументами: <порт> <серверная папка>

Сервер также будет выводить информацию о приходящих запрсах.

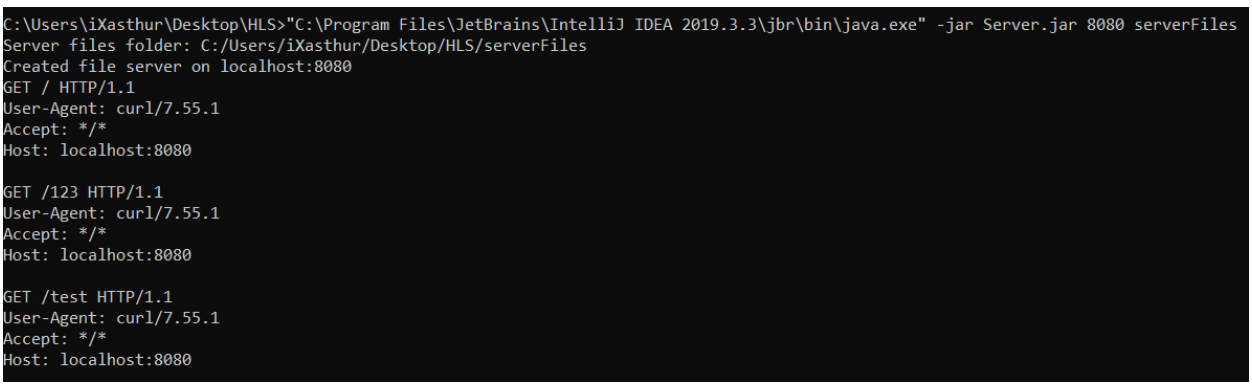


Рисунок 5.1 – Результат работы сервера

5.2.2 Запуск клиента и последующий просмотр видео

Для запуска клиента требуется запустить скомпилированный jar файл. После запуска клиент предложит ввести адрес сервера.

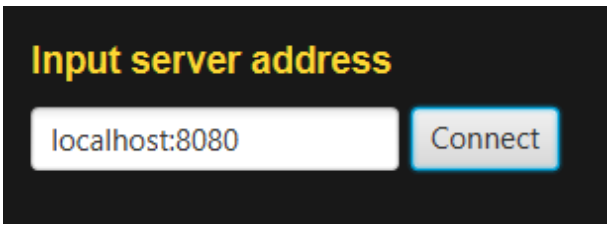


Рисунок 5.2 – Окно ввода адреса

После успешного подключения к серверу у пользователя есть возможность выбрать видео из появившегося списка.

Рисунок 5.3 – Список доступных видео

После появления списка, у пользователя есть возможность выбрать какое-либо видео и начать просмотр.

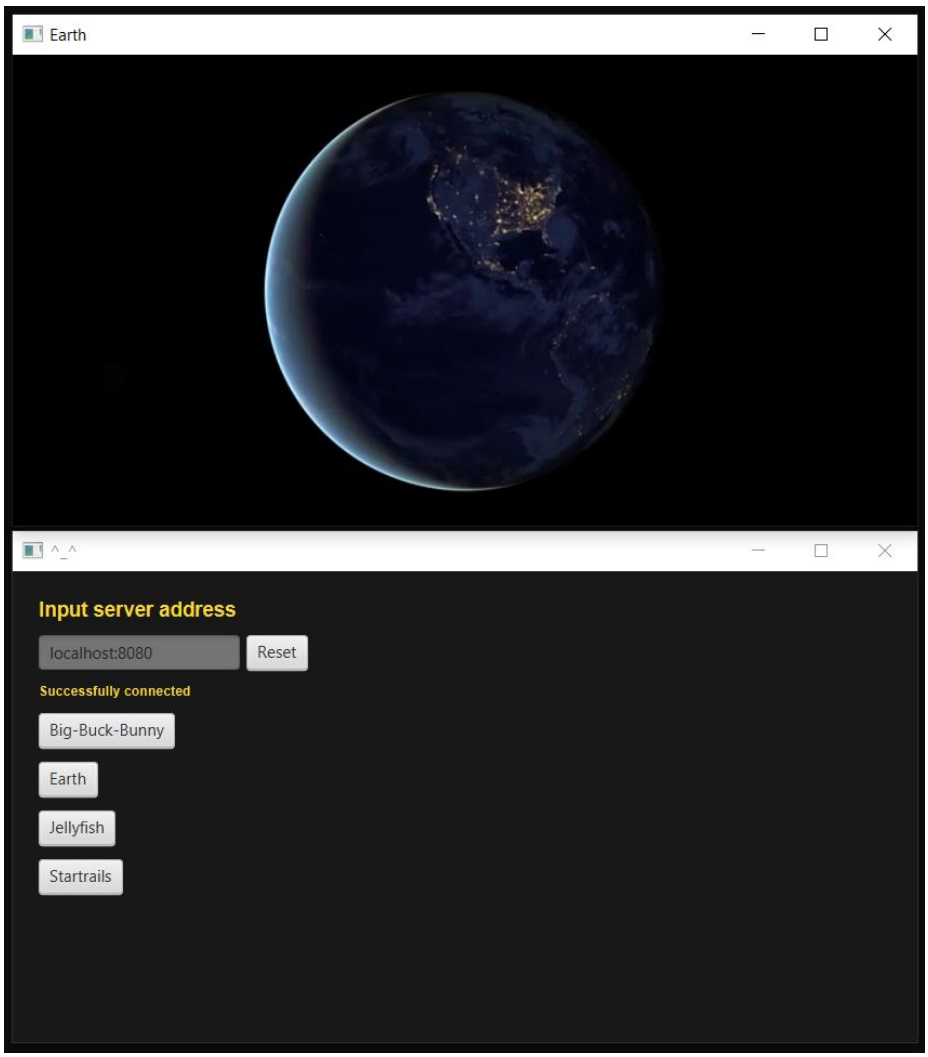


Рисунок 5.4 – Скриншот работы стрима

ЗАКЛЮЧЕНИЕ

Таким образом, в результате проделанной работы был создал сервер, использующий технологию HLS для стриминга видео, которую используют сейчас такие популярные площадки как Twitch и YouTube, а также клиент, позволяющий проигрывать видео, полученное с сервера.

Сервер может быть использован не только с написанным клиентом, а и с любым клиентом, поддерживающим потоковое вещание по протоколу HLS.

В ходе разработки был более глубоко изучен язык программирования Java, протоколы стека TCP/IP, в частности были усовершенствованы знания по протоколам TCP, HTTP, HTTPS, HLS.

Было произведено тестирование серверной и клиентской частей приложения, которое показало полное соответствие разработанного приложения спецификации требований. Разработано руководство пользователя.

Суммируя вышесказанное, можно сказать, что разработанное программное средство обладает всеми заявленными требованиями и представляет полноценный функциональный продукт, доступный для широкого пользователя.

СПИСОК ЛИТЕРАТУРЫ

[1] icecast.org [Электронный портал]. – Электронные данные. – Режим доступа: http://www.icecast.org

[2] shoutcast.com [Электронный портал]. – Электронные данные. – Режим доступа: https://www.shoutcast.com/

[3] niginx.org [Электронный портал]. – Электронные данные. – Режим доступа: http://www.niginx.org

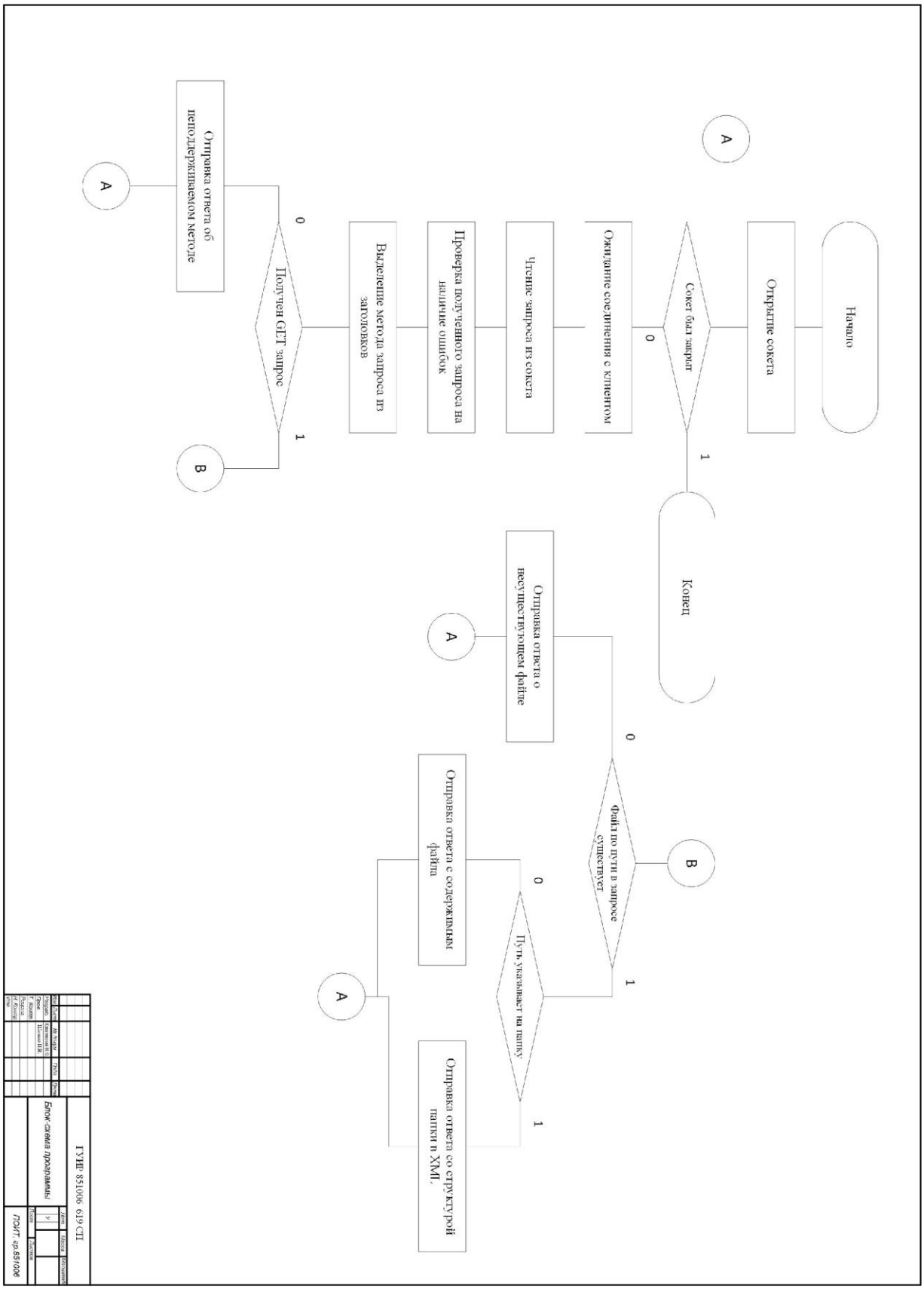
[4] docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html [Электронный портал].–Электронные данные. – Режим доступа: https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html

[5] docs.oracle.com/javase/7/docs/api/java/net/Socket.html [Электронный портал]. – Электронные данные. – Режим доступа: https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

[6] docs.oracle.com/javase/8/javafx/api/javafx/scene/media/Media.html [Электронный портал]. – Электронные данные. – Режим доступа: https://docs.oracle.com/javase/8/javafx/api/javafx/scene/media/Media.html

ПРИЛОЖЕНИЯ

Приложение 1 (схема алгоритма программы на А1)



Приложение 2 (Код программы)

Класс HTTPResponse

```
package jv.http;

import java.io.DataOutputStream;
import java.io.IOException;
import java.util.HashMap;

public class HTTPResponse {

    private static class HTTPStatus {
        private static final HashMap<Integer, String> statuses = new HashMap<>(0);
        static {
            statuses.put(100, "Continue");

            statuses.put(200, "OK");
            statuses.put(201, "Created"); // Created file
            statuses.put(204, "No Content"); // Only file info

            statuses.put(400, "Bad Request");
            statuses.put(403, "Forbidden");
            statuses.put(404, "Not Found"); // File not found

            statuses.put(500, "Internal Server Error");
            statuses.put(501, "Not Implemented");
        }

        public static String getStringFor(int code) {
            String statusString = statuses.get(code);
            if (statusString == null) {
                code = 501;
                statusString = statuses.get(code);
            }
```

```
            return code + " " + statusString;
        }
    }

    private final String responseLine;
    private final HashMap<String, String> headers = new HashMap<>(0);
    private byte[] data = new byte[]{};

    public HTTPResponse(int code) {
        responseLine = "HTTP/1.1 " + HTTPStatus.getStringFor(code);
    }

    public void addHeader(String key, String value) {
        headers.put(key, value);
    }

    public void setData(byte[] data, String contentTypeExtension) {
        // add content length header
        this.data = data;
        addHeader("Content-Length", String.valueOf(data.length));
        addHeader("Content-Type",     HTTPContentType.getContentTypeFor(contentTypeExten-
sion));
    }

    public void send(DataOutputStream outputStream) throws IOException {
        StringBuilder responseString = new StringBuilder(responseLine + "\r\n");
        for(HashMap.Entry<String, String> entry : headers.entrySet()) {
            String key = entry.getKey();
            String value = entry.getValue();

            String headerString = key + ": " + value;
            responseString.append(headerString).append("\r\n");
        }
        responseString.append("\r\n");
```

```
        outputStream.writeBytes(responseString.toString());
        if (data != null) {
            outputStream.write(data);
        }
    }
}
```

Класс HTTPRequest

```
package jv.http;


import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.HashMap;
import java.util.Scanner;
import java.util.Vector;

public class HTTPRequest {

    private final String requestLine;
    private final HashMap<String, String> headers = new HashMap<>(0);
    private byte[] data = new byte[]{};

    public HTTPRequest(DataInputStream inputStream) throws IOException {
        Scanner scanner = new Scanner(inputStream);
        String line;
        if (scanner.hasNextLine()) {
            line = scanner.nextLine();
            requestLine = line;
        } else {
```

```
            throw new IOException("Nothing to read from socket");
        }

        // Read headers
        while (scanner.hasNextLine()) {
            line = scanner.nextLine();
            if (!line.equals("")) {
                int spaceIndex = line.indexOf(" ");
                headers.put(line.substring(0, spaceIndex - 1), line.substring(spaceIndex + 1));
            } else {
                break;
            }
        }

        if (headers.get("Content-Length") != null) {
            int count = Integer.parseInt(headers.get("Content-Length"));
            data = inputStream.readNBytes(count);
        } else if (headers.get("Transfer-Encoding") != null && headers.get("Transfer-Encoding").equals("chunked")) {
            // Read chunked data
            int count = readChunkSize(inputStream);
            while (count > 0) {
                byte[] buffer = inputStream.readNBytes(count);

                ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
                outputStream.write(data);
                outputStream.write(buffer);
                data = outputStream.toByteArray();

                inputStream.readNBytes(2); // skip \r\n
                count = readChunkSize(inputStream);
            }
            inputStream.readNBytes(2);
        }
```

```
}

public String getHeaderValue(String key) {
    return headers.get(key);
}

private int readChunkSize(DataInputStream inputStream) throws IOException {
    Vector<Byte> bytes = new Vector<>(0);
    while (true) {
        byte b = inputStream.readByte();
        if (bytes.size() > 0) {
            if (bytes.lastElement() == 13 && b == 10) {
                bytes.remove(bytes.lastElement());
                break;
            }
        }
        bytes.add(b);
    }
    byte[] buffer = new byte[bytes.size()];
    for (int i = 0; i < bytes.size(); i++) {
        buffer[i] = bytes.elementAt(i);
    }
    return Integer.parseInt(new String(buffer),16);
}

public void outputRequest() {
    System.out.println(requestLine);
    for(HashMap.Entry<String, String> entry : headers.entrySet()) {
        String key = entry.getKey();
        String value = entry.getValue();

        System.out.println(key + ": " + value);
    }
}
```

```
    public String getRequestMethod() {
        if (requestLine != null) {
            String[] split = requestLine.split(" ");
            return split[0];
        }
        return null;
    }

    public String getRelativePath() {
        if (requestLine != null) {
            String[] split = requestLine.split(" ");
            String path = split[1];
            try {
                URL url = new URL(path);
                path = url.getPath();
            } catch (MalformedURLException e) {
                // Do nothing. Path is already relative
            }
            return path;
        }
        return null;
    }

    public byte[] getData() {
        return data;
    }
}
```

Класс HTTPContentType

```
package jv.http;

import java.util.HashMap;
```

```
public class HTTPContentType {
    private static final HashMap<String, String> types = new HashMap<>(0);
    static {
        types.put("pdf", "application/pdf");
        types.put("txt", "text/plain");
        types.put("html", "text/html");
        types.put("exe", "application/octet-stream");
        types.put("zip", "application/zip");
        types.put("doc", "application/msword");
        types.put("xls", "application/vnd.ms-excel");
        types.put("ppt", "application/vnd.ms-powerpoint");
        types.put("gif", "image/gif");
        types.put("png", "image/png");
        types.put("jpeg", "image/jpg");
        types.put("jpg", "image/jpg");
        types.put("php", "text/plain");

        types.put("m3u8", "application/vnd.apple.mpegurl");
        types.put("ts",  "video/mp2t");
    }

    public static String getContentTypeFor(String extension) {
        String contentType = types.get(extension);
        if (contentType == null) {
            contentType = "application/octet-stream";
        }
        return contentType;
    }
}
```

Класс FileServerThread

```
package jv.fileserver;

import jv.http.HTTPRequest;
```

```
import jv.http.HTTPResponse;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;

public class FileServerThread extends Thread {

    private final Socket socket;
    private final String serverFilesFolderPath;

    public FileServerThread(Socket socket, String severFilesFolderPath) {
        this.socket = socket;
        this.serverFilesFolderPath = severFilesFolderPath;
    }

    @Override
    public void run() {
        super.run();

        try (DataInputStream inputStream = new DataInputStream(socket.getInputStream());
             DataOutputStream outputStream = new DataOutputStream(socket.getOutputStream())){

            HTTPRequest request = new HTTPRequest(inputStream);
            request.outputRequest();
            System.out.println();
            HTTPResponse response = new APIHandler(request, serverFilesFolderPath).getResponse();
            response.send(outputStream);

        } catch (IOException e) {
            // Everything will be closed by try-with-resources
```

```
            // e.printStackTrace();
        }

        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

Класс APIHandler

```
package jv.fileserver;

import jv.http.HTTPContentType;
import jv.http.HTTPRequest;
import jv.http.HTTPResponse;
import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
```

```
import java.io.StringWriter;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Comparator;

public class APIHandler {

    private final String serverFilesFolderPath;

    private final HTTPRequest request;
    private HTTPResponse response = new HTTPResponse(501);

    public APIHandler(HTTPRequest request, String serverFilesFolderPath) {
        this.request = request;
        this.serverFilesFolderPath = serverFilesFolderPath;

        parseRequest();
    }

    public String getDirectoryStructureXML(String relativePath) throws IOException, ParserConfigurationException, TransformerException {
        String filePathString = serverFilesFolderPath + request.getRelativePath();
        Path filePath = Paths.get(filePathString);

        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();

        Document document = builder.newDocument();

        Element rootElement = document.createElement("root");
        document.appendChild(rootElement);

        Attr pathAttribute = document.createAttribute("path");
```

```
            pathAttribute.setValue(relativePath);
            rootElement.setAttributeNode(pathAttribute);

            Files.list(filePath)
                    .forEach(path -> {
                        if (Files.isRegularFile(path)) {
                            Element element = document.createElement("file");
                            element.appendChild(document.createTextNode(String.valueOf(path.getFile-
Name())));
                            rootElement.appendChild(element);
                        } else if (Files.isDirectory(path)) {
                            Element element = document.createElement("dir");
                            element.appendChild(document.createTextNode(String.valueOf(path.getFile-
Name())));
                            rootElement.appendChild(element);
                        }
                    });


            TransformerFactory transformerFactory = TransformerFactory.newInstance();
            Transformer transformer = transformerFactory.newTransformer();
            StringWriter writer = new StringWriter();
            transformer.transform(new DOMSource(document), new StreamResult(writer));

            return writer.getBuffer().toString();
        }

        private String extractFileExtension(String fileName) {
            String fileExtension = "";
            if (fileName.contains(".")) {
                int index = fileName.lastIndexOf(".");
                if (index < fileName.length() -1) {
                    fileExtension = fileName.substring(index + 1);
                }
```

```
        }
        return fileExtension;
    }

    synchronized public void parseRequest() {
        String method = request.getRequestMethod();
        String filePathString = serverFilesFolderPath + request.getRelativePath();
        Path filePath = Paths.get(filePathString);

        switch (method) {
            case "GET": {

                // For downloading files
                // + to get available videos

                if (Files.exists(filePath)) {
                    if (Files.isRegularFile(filePath)) {
                        // Send file
                        response = new HTTPResponse(500);

                        try {
                            byte[] bytes = Files.readAllBytes(filePath);
                            String fileName = filePath.getFileName().toString();
                            String fileExtension = extractFileExtension(fileName);

                            response = new HTTPResponse(200);
                            response.setData(bytes, fileExtension);

                            // Headers to allow CORS
                            response.addHeader("Access-Control-Allow-Origin", "*");
                            response.addHeader("Access-Control-Allow-Headers", "Origin, X-Requested-
With, Content-Type, Accept, Authorization");
                            response.addHeader("Access-Control-Allow-Credentials", "true");
                            response.addHeader("Access-Control-Allow-Methods", "GET");
```

```
                    } catch (IOException e) {
//                      e.printStackTrace();
                    }
                } else if (Files.isDirectory(filePath)) {
                    // Send directory structure
                    response = new HTTPResponse(500);
                    try {
                        String xmlString = getDirectoryStructureXML(request.getRelativePath());
                        response = new HTTPResponse(200);
                        response.setData(xmlString.getBytes(), "txt");
                    } catch (IOException | ParserConfigurationException | TransformerException e)
{
//                      e.printStackTrace();
                    }
                }
            } else {
                response = new HTTPResponse(404);
                response.setData("File does not exist".getBytes(), "txt");
            }
            break;
        }
    }
    response.addHeader("Connection", "Closed");
    response.addHeader("Server", "Jv-file-server");
  }

  public HTTPResponse getResponse() {
    return response;
  }

}
```

Класс HLSClient

```
package jv.hlsclient;

import javafx.event.EventHandler;
import javafx.scene.media.Media;
import javafx.scene.media.MediaErrorEvent;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;
import jv.http.HTTPRequest;
import jv.http.HTTPResponse;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import java.io.*;
import java.net.Socket;
import java.net.URL;
import java.util.Vector;

public class HLSClient {

    // Root folder of the server must contain folders with .m3u8 file + .ts files
    // name of folder == name of video
    // every folder in root will be recognized as video folder
    // directory name must not contain spaces

    final public URL domain; // "http://localhost:8080/
                                        // test4/index.m3u8";
    final private String playlistFileName = "index.m3u8";
    final private Vector<String> videoFolderNames = new Vector<>(0);
```

```
public HLSClient(URL domain) throws Exception {
    this.domain = domain;

    try {
        getAvailableVideos();
    } catch (Exception e) {
        throw new Exception(e.getMessage());
    }
}

private void getAvailableVideos() throws Exception {
    videoFolderNames.clear();

    String host = domain.getHost();
    int port = domain.getPort();
    if (port == -1) {
        port = 80;
    }

    Socket socket = null;
    try {
        socket = new Socket(host, port);
    } catch (IOException e) {
        throw new Exception("Unable to connect to " + domain.toString());
    }

    StringBuilder requestStringBuilder = new StringBuilder();
    requestStringBuilder.append("GET");
    requestStringBuilder.append(" ");
    requestStringBuilder.append("/");
    requestStringBuilder.append(" ");
    requestStringBuilder.append("HTTP/1.1");
    requestStringBuilder.append("\r\n");
    requestStringBuilder.append("Host");
```

```
            requestStringBuilder.append(":");
            requestStringBuilder.append(" ");
            requestStringBuilder.append(host);
            requestStringBuilder.append(":");
            requestStringBuilder.append(port);
            requestStringBuilder.append("\r\n");
            requestStringBuilder.append("\r\n");

            InputStream is = new ByteArrayInputStream(requestStringBuilder.toString().getBytes());
            DataInputStream dis = new DataInputStream(is);
            HTTPRequest request = null;
            try {
               request = new HTTPRequest(dis);
            } catch (IOException e) {
               throw new Exception("Unable to create request");
            }

            byte[] data = null;

            try (DataInputStream inputStream = new DataInputStream(socket.getInputStream());
                 DataOutputStream outputStream = new DataOut-
putStream(socket.getOutputStream())){

               request.send(outputStream);
               HTTPResponse response = new HTTPResponse(inputStream);
               data = response.getData();

            } catch (Exception e) {
               throw new Exception("Can't send data to " + host + ":" + port);
            }

            try {
               fillPlaylistVector(data);
            } catch (Exception e) {
```

```
            throw new Exception("Received invalid XML data");
        }
    }

    //  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
    //  <root path="/">
    //    <file>ffmpeg_script.txt</file>
    //    <dir>test1</dir>
    //    <dir>test2</dir>
    //    <dir>test3</dir>
    //    <dir>test4</dir>
    //  </root>
    private void fillPlaylistVector(byte[] data) throws ParserConfigurationException, IOException, SAXException {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(new ByteArrayInputStream(data));

        System.out.println(new String(data));

        Element root = document.getDocumentElement();
        NodeList videoFolders = root.getElementsByTagName("dir");
        for (int i = 0; i < videoFolders.getLength(); i++) {
            videoFolderNames.add(videoFolders.item(i).getTextContent());
        }
    }

    public Vector<String> getVideoFolderNames() {
        return videoFolderNames;
    }

    public HLSMedia getMedia(String playlistDirectory) {
        try {
            getAvailableVideos();
```

```
} catch (Exception e) {
    return null;
}
if (!videoFolderNames.contains(playlistDirectory)) {
    return null;
}

String source = domain + playlistDirectory + "/" + playlistFileName;
Media media;
MediaPlayer mediaPlayer;
MediaView mediaView;
try {
    media = new Media(source);
    if (media.getError() == null) {
        media.setOnError(new Runnable() {
            public void run() {
                // Handle asynchronous error in Media object.
            }
        });
        try {
            mediaPlayer = new MediaPlayer(media);
            if (mediaPlayer.getError() == null) {
                mediaPlayer.setOnError(new Runnable() {
                    public void run() {
                        // Handle asynchronous error in MediaPlayer object.
                    }
                });
                mediaView = new MediaView(mediaPlayer);
                mediaView.setOnError(new EventHandler<MediaErrorEvent>() {
                    public void handle(MediaErrorEvent t) {
                        // Handle asynchronous error in MediaView.
                    }
                });
```

```
                    HLSMedia retMedia = new HLSMedia();
                    retMedia.media = media;
                    retMedia.mediaPlayer = mediaPlayer;
                    retMedia.mediaView = mediaView;

                    return retMedia;
                } else {
                    // Handle synchronous error creating MediaPlayer.
                }
            } catch (Exception mediaPlayerException) {
                // Handle exception in MediaPlayer constructor.
            }
        } else {
            // Handle synchronous error creating Media.
        }
    } catch (Exception mediaException) {
        // Handle exception in Media constructor.
    }

    return null;
  }

}
```

Класс Main клиента

```
import javafx.application.Application;
import javafx.beans.InvalidationListener;
import javafx.beans.Observable;
import javafx.beans.binding.Bindings;
import javafx.beans.property.DoubleProperty;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.event.EventHandler;
import javafx.geometry.Bounds;
```

```
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.*;
import javafx.scene.media.MediaPlayer;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.scene.text.TextAlignment;
import javafx.stage.Stage;
import javafx.stage.WindowEvent;
import jv.hlsclient.HLSClient;
import jv.hlsclient.HLSMedia;

import java.awt.Toolkit;
import java.awt.Dimension;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Vector;

public class Main extends Application {

    private final Color backgroundColor = Color.rgb(24,24,24);
    private final Color textColor = Color.rgb(253,216,53);

    private Group root = null;
    private HLSClient client = null;

    @Override
    public void start(Stage primaryStage) throws Exception{
```

```
            primaryStage.setTitle("^_^");

            final double windowSizeFactor = 1.0f/3.0f;
            Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
            Dimension sceneSize = new Dimension();
            sceneSize.setSize(screenSize.width*windowSizeFactor, screenSize.height*win-
dowSizeFactor);

            System.out.println("Screen size:\n" + screenSize);
            System.out.println("Scene size:\n" + sceneSize);

            root = new Group();
            Scene scene = new Scene(root, sceneSize.getWidth(), sceneSize.getHeight(), background-
Color);
            primaryStage.setMinHeight(scene.getHeight());
            primaryStage.setMinWidth(scene.getWidth());
            primaryStage.setHeight(primaryStage.getMinHeight());
            primaryStage.setWidth(primaryStage.getMinWidth());
            primaryStage.setScene(scene);

            createUI(scene);

            primaryStage.show();
        }

        private void createUI(Scene scene) {
            double sceneWidth = scene.getWidth();
            double sceneHeight = scene.getHeight();
            double fontSize = sceneHeight/26;

            TextField inputAddressTextField = new TextField();
            inputAddressTextField.setPromptText("Address");
            inputAddressTextField.setText("localhost:8080");
```

```
Label connectionResultLabel = new Label("");
connectionResultLabel.setTextFill(textColor);
connectionResultLabel.setFont(Font.font("Arial", FontWeight.BOLD, fontSize/3*2));
connectionResultLabel.setTextAlignment(TextAlignment.CENTER);

Vector<Button> videoButtons = new Vector<>(0);

Button connectButton = new Button("Connect");

HBox hBox = new HBox(inputAddressTextField, connectButton);
hBox.setSpacing(5);

Label inputAddressLabel = new Label("Input server address");
inputAddressLabel.setTextFill(textColor);
inputAddressLabel.setFont(Font.font("Arial", FontWeight.BOLD, fontSize));
VBox vBox = new VBox(inputAddressLabel, hBox, connectionResultLabel);

vBox.setLayoutX(sceneHeight/20);
vBox.setLayoutY(sceneHeight/20);

vBox.setSpacing(10);

root.getChildren().add(vBox);


// Actions
connectButton.setOnAction(action -> {
    if (client == null) {
        try {
            connectionResultLabel.setText("Connecting...");
            connect(inputAddressTextField.getText());
            connectionResultLabel.setText("Successfully connected");
            inputAddressTextField.setDisable(true);
```

```
                connectButton.setText("Reset");

                Vector<String> videoFolders = client.getVideoFolderNames();
                for (int i = 0; i < videoFolders.size(); i++) {
                    Button button = new Button(videoFolders.elementAt(i));
                    videoButtons.add(button);

                    button.setOnAction(actionEvent -> {
                        String videoName = button.getText();
                        HLSMedia hlsMedia = client.getMedia(videoName);
                        if (hlsMedia != null) {
                            Stage videoStage = new Stage();
                            videoStage.setTitle(videoName);
                            VBox videoBox = new VBox(hlsMedia.mediaView);
                            videoBox.setAlignment(Pos.CENTER);
                            videoBox.setBackground(new Background(new Background-
Fill(Color.BLACK, CornerRadii.EMPTY, Insets.EMPTY)));
                            Scene videoScene = new Scene(videoBox, sceneWidth, sceneHeight, back-
groundColor);
                            videoStage.setScene(videoScene);

                            hlsMedia.mediaView.getParent().layoutBoundsProperty().addListener(new
ChangeListener<Bounds>() {
                                @Override
                                public void changed(ObservableValue<? extends Bounds> observable,
Bounds oldValue, Bounds newValue) {
                                    if (hlsMedia.mediaPlayer.getStatus() != MediaPlayer.Status.DIS-
POSED) {
                                        hlsMedia.mediaView.setFitHeight(newValue.getHeight());
                                        hlsMedia.mediaView.setFitWidth(newValue.getWidth());
                                    }
                                }
                            });
```

```
                    hlsMedia.mediaPlayer.setOnEndOfMedia(() -> {
                        hlsMedia.mediaPlayer.dispose();

                        videoBox.setBackground(new Background(new BackgroundFill(back-
groundColor, CornerRadii.EMPTY, Insets.EMPTY)));

                        Label label = new Label("Video ended");
                        label.setTextFill(textColor);
                        label.setFont(Font.font("Arial", FontWeight.BOLD, fontSize*2));

                        videoBox.getChildren().remove(hlsMedia.mediaView);
                        videoBox.getChildren().add(label);
                    });
                    videoStage.setMinWidth(sceneWidth/2.0);
                    videoStage.setMinHeight(sceneHeight/2.0);

                    videoStage.setWidth(sceneWidth);
                    videoStage.setHeight(sceneHeight);

                    videoStage.setOnCloseRequest(new EventHandler<WindowEvent>() {
                                @Override
                                public void handle(WindowEvent windowEvent) {
                                    hlsMedia.mediaPlayer.dispose();
                                }
                            }
                    );

                    videoStage.show();

                    hlsMedia.mediaPlayer.play();
                } else {
                    button.setDisable(true);
                }
            });
```

```
                }

                vBox.getChildren().addAll(videoButtons);
            } catch (Exception e) {
                connectionResultLabel.setText(e.getMessage());
            }
        } else {
            vBox.getChildren().removeAll(videoButtons);
            videoButtons.clear();

            client = null;
            connectButton.setText("Connect");
            connectionResultLabel.setText("");
//              inputAddressTextField.setText("");
            inputAddressTextField.setDisable(false);
        }
    });
  }

  private void connect(String address) throws Exception {
    if (!address.contains("/") && !address.isEmpty()) {
        address = "http://" + address + "/";
        System.out.println("Connecting to " + address);

        URL domain = null;
        try {
            domain = new URL(address);
        } catch (MalformedURLException e) {
            throw new Exception("Invalid address string");
        }

        try {
            client = new HLSClient(domain);
        } catch (Exception e) {
```

```
                throw new Exception(e.getMessage());
            }
        } else {
            throw new Exception("Invalid address string");
        }
    }

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

Класс Main сервера

```
import jv.fileserver.FileServerThread;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

public class Main {

    public static void main(String[] args) {

        // Set 8080 as default value in IDE

        if (args.length != 2) {
            System.out.println("Invalid args. Args: <port> <files directory name>");
            return;
        }

        final int port;
```

```
        try {
            port = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
            System.out.println("Invalid args. Args: <port> <files directory name>");
            return;
        }

        final String serverFilesFolderPathString;
        try {
            serverFilesFolderPathString = new java.io.File(args[1]).getCanonicalPath().replace("\\",
"/");
            Path serverFilesFolderPath = Paths.get(serverFilesFolderPathString);
            if (Files.exists(serverFilesFolderPath)) {
                if (!Files.isDirectory(serverFilesFolderPath)) {
                    throw new Exception(args[1] + " is not directory");
                }
            } else {
                throw new Exception(args[1] + " does not exist");
            }
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Invalid args. Args: <port> <files directory name>");
            return;
        }
        System.out.println("Server files folder: " + serverFilesFolderPathString);

        // The Java runtime automatically closes the input and output streams, the client socket,
        // and the server socket because they have been created in the try-with-resources statement.
        // + JVM/OS will close everything on exit
        try (ServerSocket server = new ServerSocket(port)) {

            System.out.println("Created file server on localhost:" + port);

            while (true) {
```

```
                Socket socket = server.accept();

                // For testing in browser
                // https://www.hlstester.com/
                // http://inspectstream.theoplayer.com/

                FileServerThread thread = new FileServerThread(socket, serverFilesFolder-
PathString);
                thread.start();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```