University Of West Attica

**Multimedia and Multimedia Communications 22-23**

Report

**By**: Ahmad Hijazi

**Contents**

[**Description of the System and Their Functions………..……….3**](file:///C:\Users\asus\Documents\Fundamentals%20of%20Information%20Systems%20Coursework%20Report%20template-1.docx#_gjdgxs)

**Presentation of the Final Implemented System……………………13**

**Presentation of the Problems Encountered and Their Solutions…17**

**Description of the System and Their Functions:**

* 1. **Introduction:**

The application is designed to convert video files to different resolutions and formats using the FFmpeg library. It provides a convenient way to process multiple video files in a specified directory and generate additional files with varying resolutions and formats.

These files are filtered based on the Client Wi-Fi speed and his choice of video format.

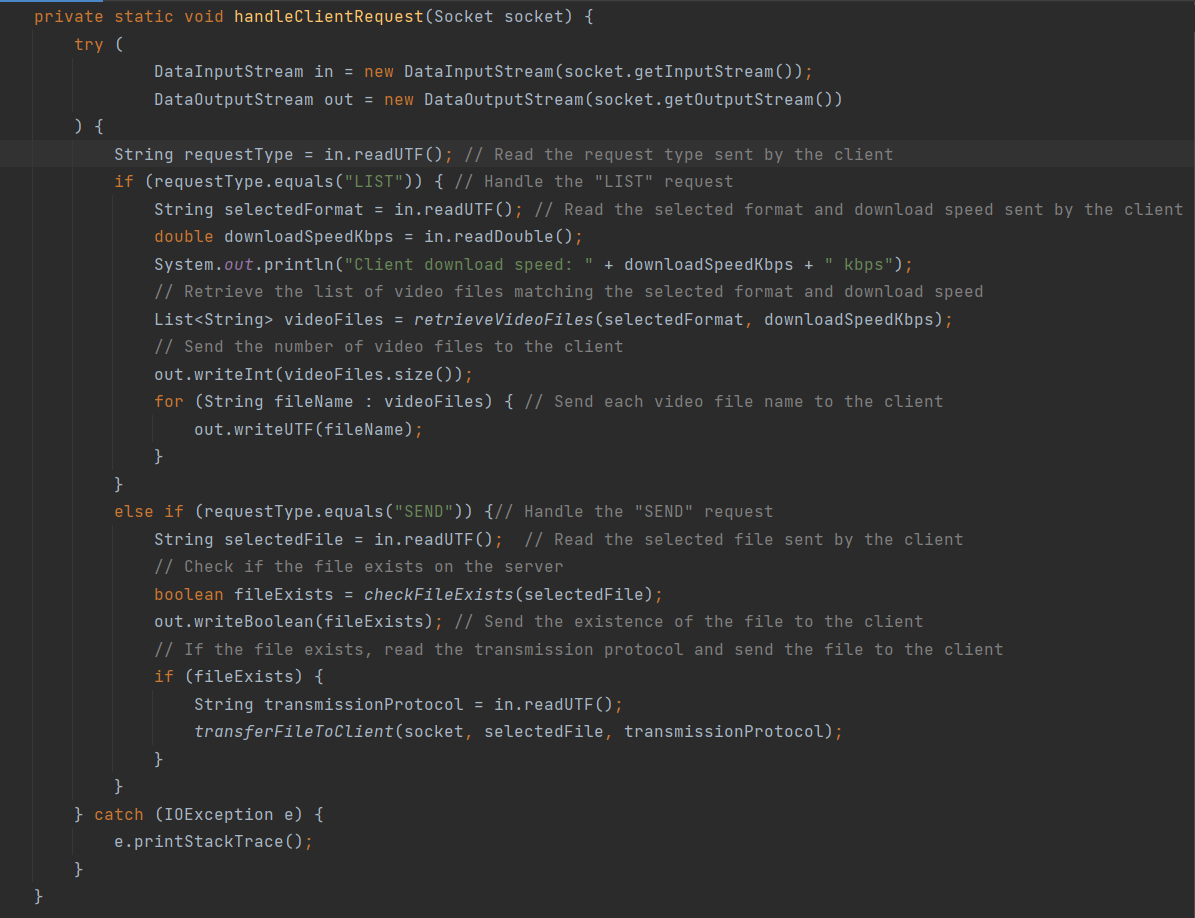
After that, the video will be transmitted and played by the client.

* 1. **System Overview**:

The system consists of two main components:

* **Streaming Server:** A Java application responsible for managing a collection of available video files for streaming, converting files into multiple formats and resolutions, handling client communication, and transmitting video files to clients.
* **Streaming Client:** A Java application that runs on the end-user side and facilitates communication with the Streaming Server, enables downloading of video files, and provides a user interface for playing the videos.

**Streaming Server Functions:**

****

The method receives a Socket object representing the connection with the client.

It creates DataInputStream and DataOutputStream objects to facilitate communication with the client.

It reads the requestType sent by the client using in.readUTF(). The requestType indicates the type of operation the client is requesting.

If the requestType is "LIST", it proceeds to handle the request for a list of available video files.

* It reads the selectedFormat and downloadSpeedKbps values sent by the client using in.readUTF() and in.readDouble(), respectively.
* It prints the client's download speed to the console.
* It calls the retrieveVideoFiles method to retrieve a list of video files that match the selected format and are suitable for the client's download speed.
* It sends the size of the videoFiles list to the client using out.writeInt(videoFiles.size()). e. It iterates over the videoFiles list and sends each video file name to the client using out.writeUTF(fileName).

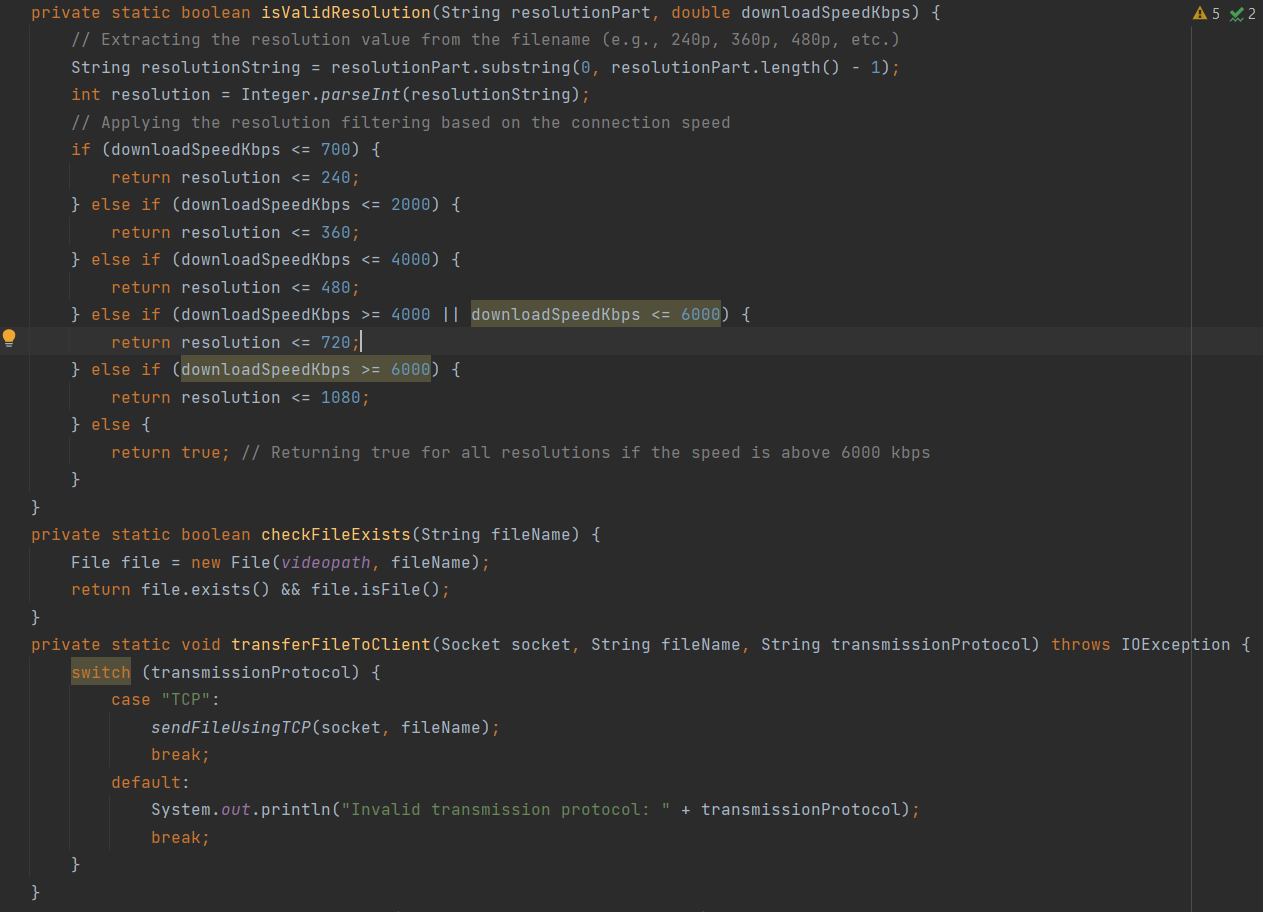
If the requestType is "SEND", it handles the request to send a specific video file to the client.

If an IOException occurs during the process, it prints the stack trace to the console.

A screen shot of a computer screen

Description automatically generated with low confidence

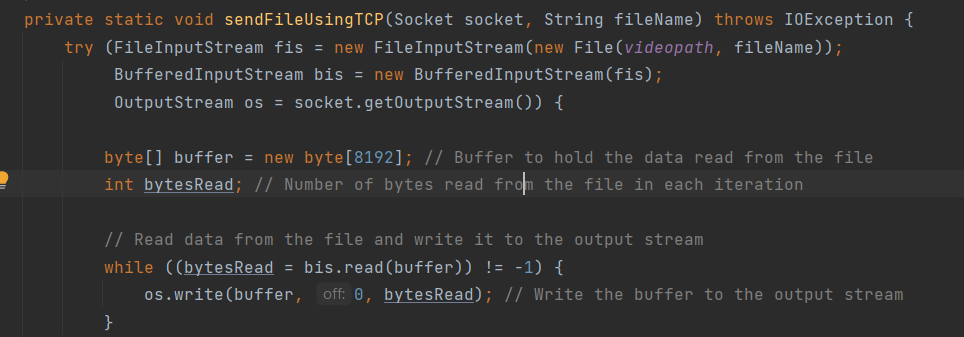
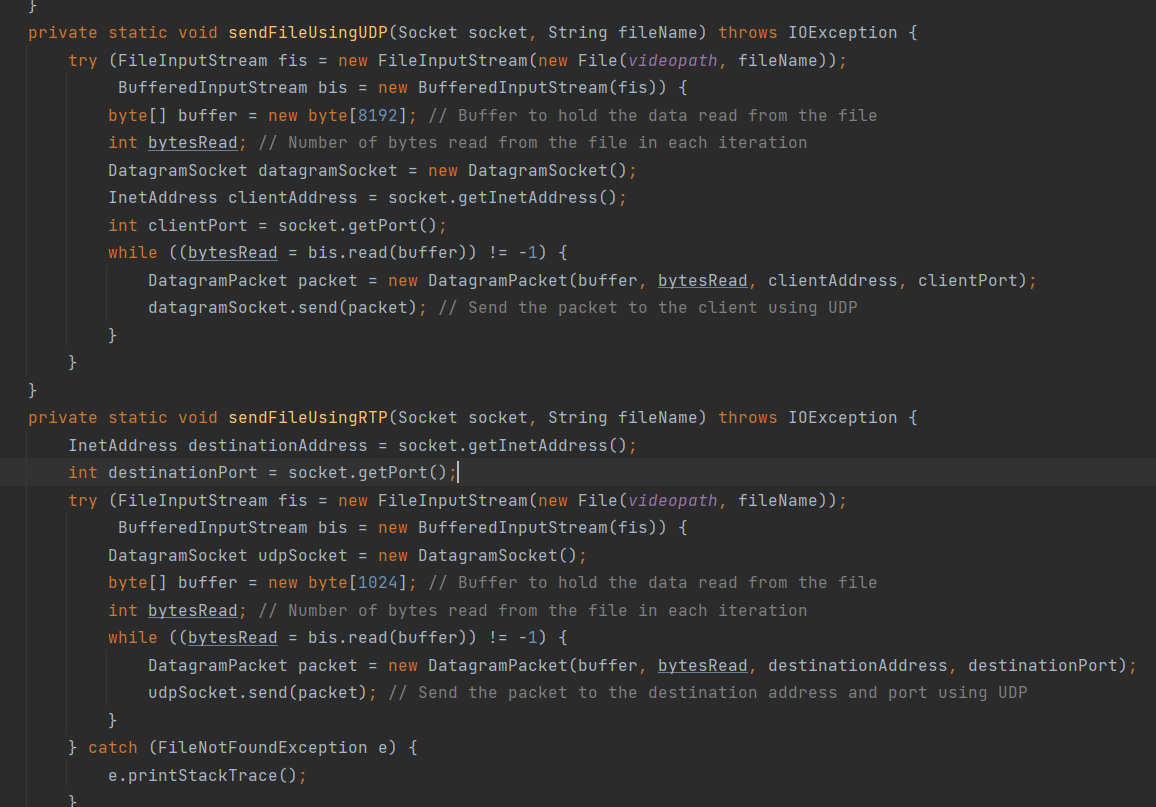
The code iterates through each file in the specified directory, checks if it is a regular file and its name ends with the specified format. If the file name contains at least two parts (name and resolution), it extracts the resolution part and checks if it is valid for the given download speed. If the resolution is valid, the file name is added to the list of video files. Finally, the method returns the list of video files.



The isValidResolution checks wifi speed and applies the resolution filter, after extracting it from the file name.

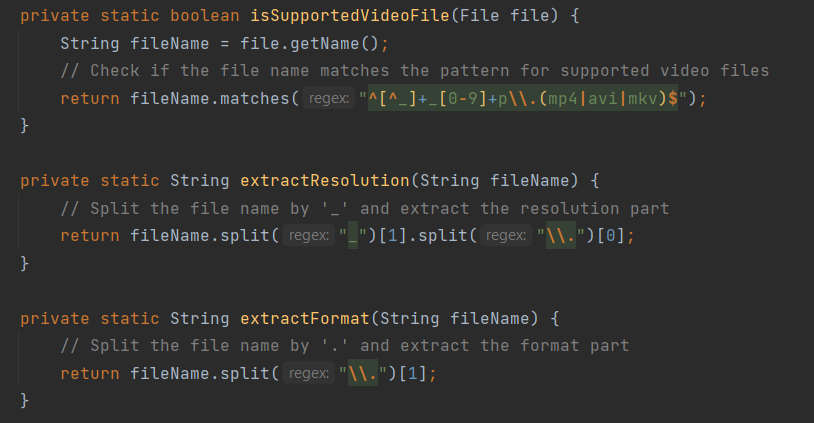
The checkFileExists method checks if a file with the given fileName exists in the specified videopath

The transferFileToClient method handles the transfer of a file to the client using the specified transmission protocol. Only TCP exists for now.

3 method each is to send a file using a 1 of the transmission protocol

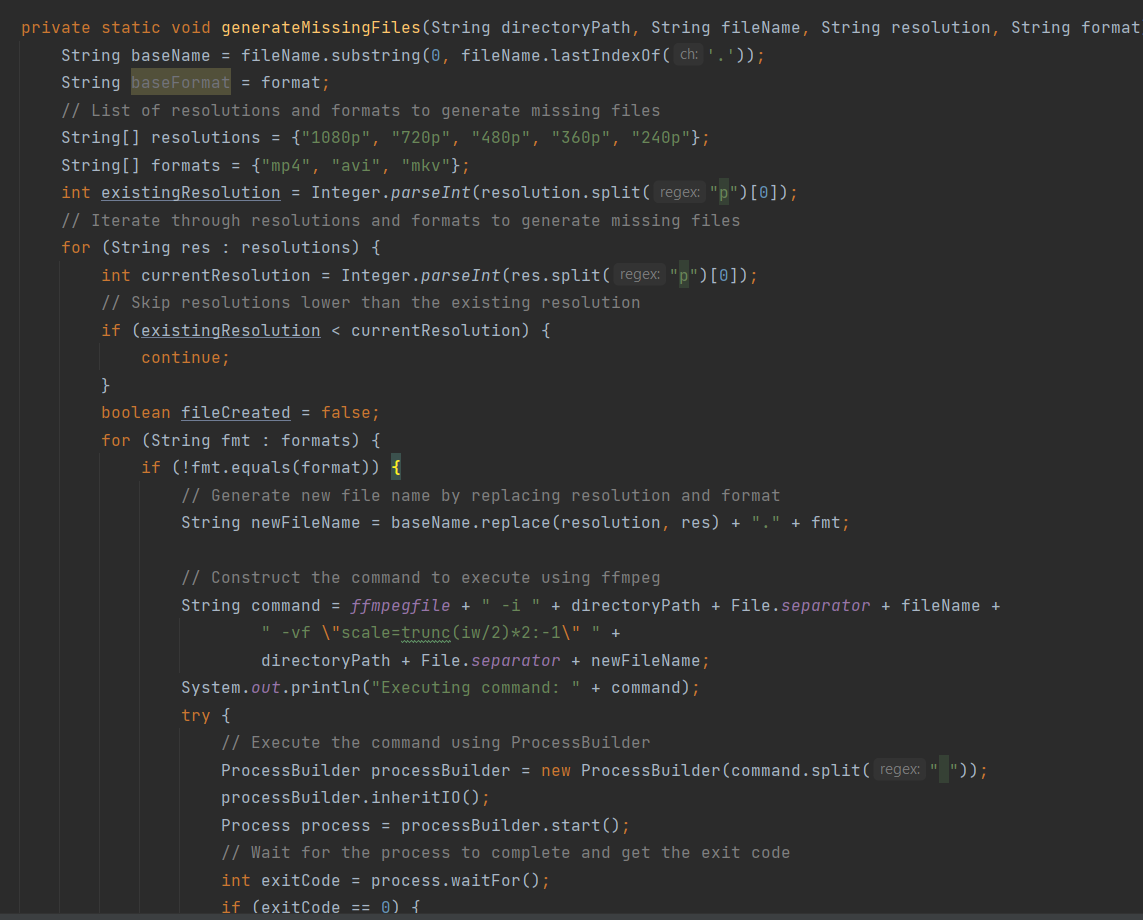
In this case the file generation using ffmpeg happen in a class videoconverter which is called in streamingserver

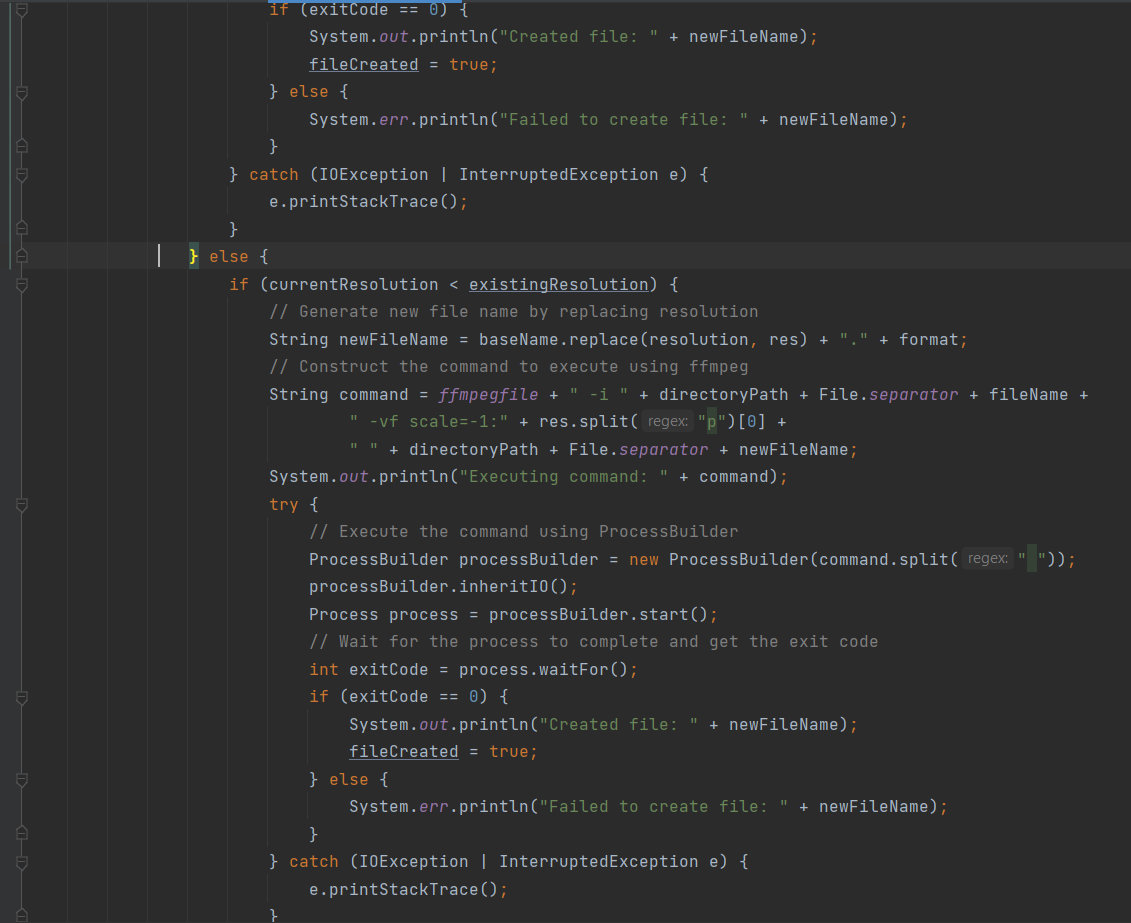


isSupportedVideoFile checks if the given File object represents a supported video file by matching its name against a regular expression pattern.

extractResolution takes a file name as input, splits it by '\_' and extracts the resolution part (e.g., "720p").

extractFormat takes a file name as input, splits it by '.' and extracts the format part (e.g., "mp4").





generateMissingFiles generates missing files with different resolutions and formats based on the existing resolution and format of the original file.

The method iterates through the list of resolutions and formats to generate missing files.

For each resolution, it checks if the current resolution is greater than or equal to the existing resolution.

If it is, it proceeds to iterate through the formats.

If the format is different from the original format, it constructs a command to execute using ffmpeg for scaling the original file to the new resolution and format.

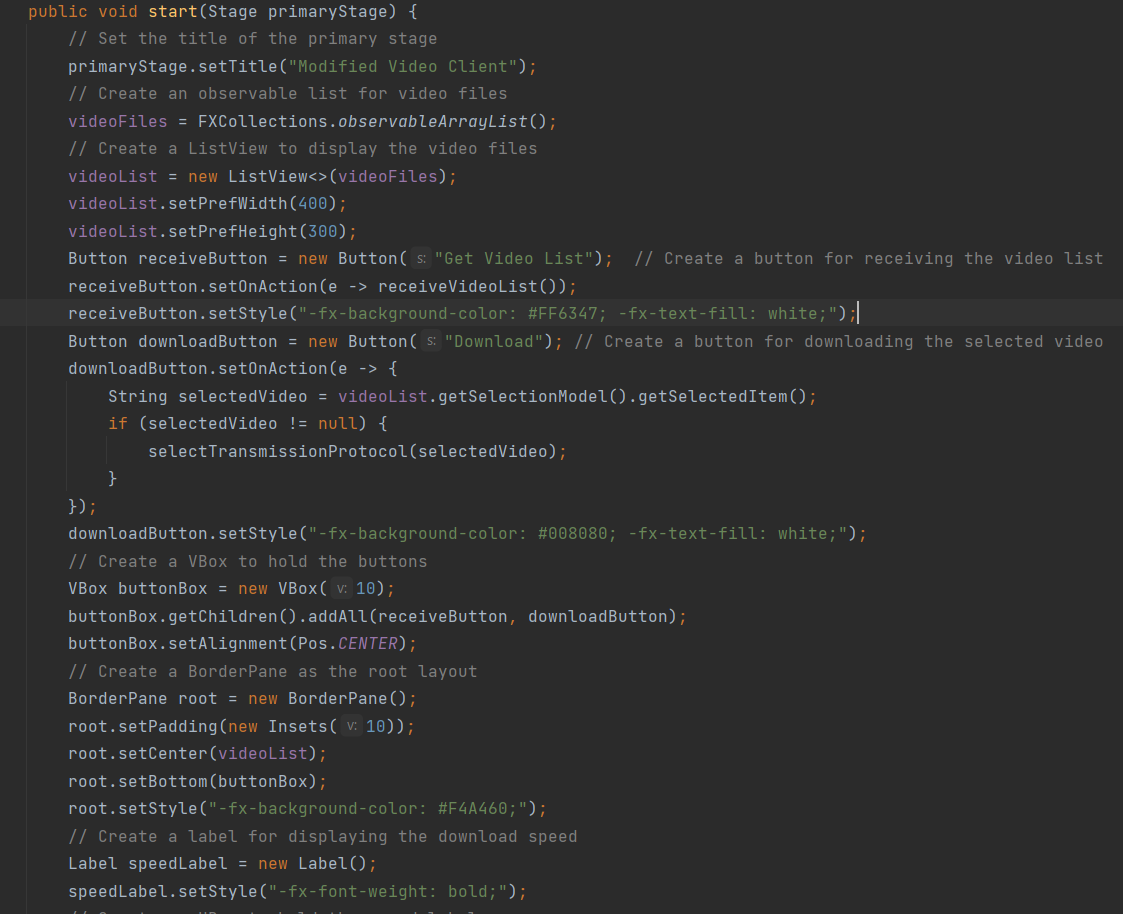
If the format is the same as the original format and the current resolution is lower than the existing resolution, it constructs a command to execute using ffmpeg for scaling the original file to a lower resolution.

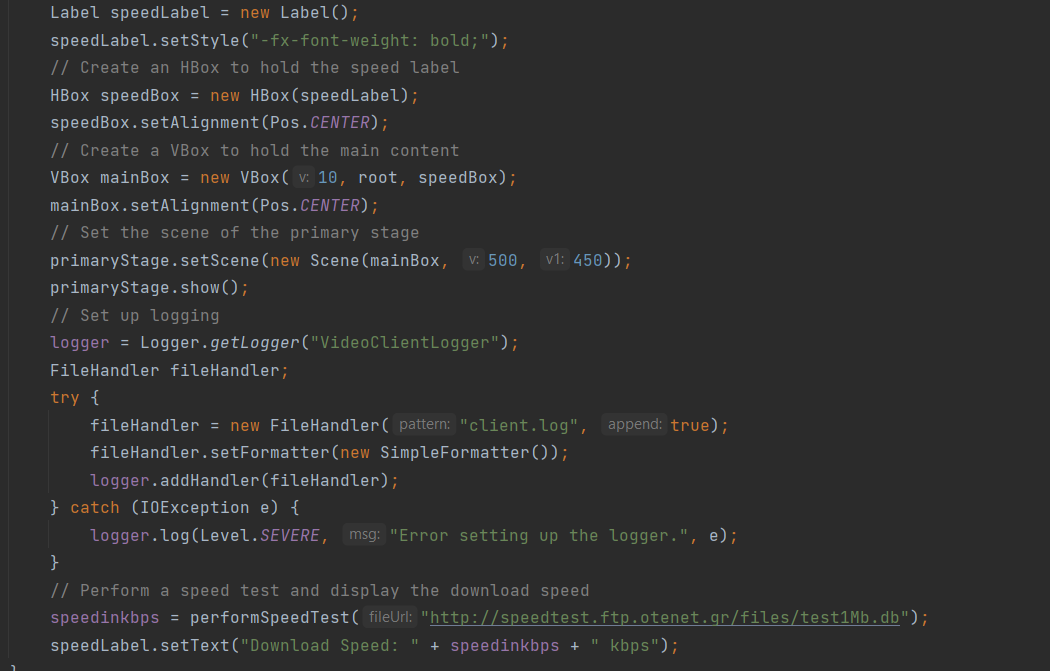
The commands are executed using ProcessBuilder, and the output and error streams are redirected to the console.

The method waits for the process to complete and checks the exit code to determine if the file was successfully created or not.

If a file is successfully created, it updates the existing resolution.

**Streaming Client Functions:**

****

****

The start method is the entry point for launching the JavaFX application.

It sets up the UI components, event handlers, styling, logging, and performs a speed test.

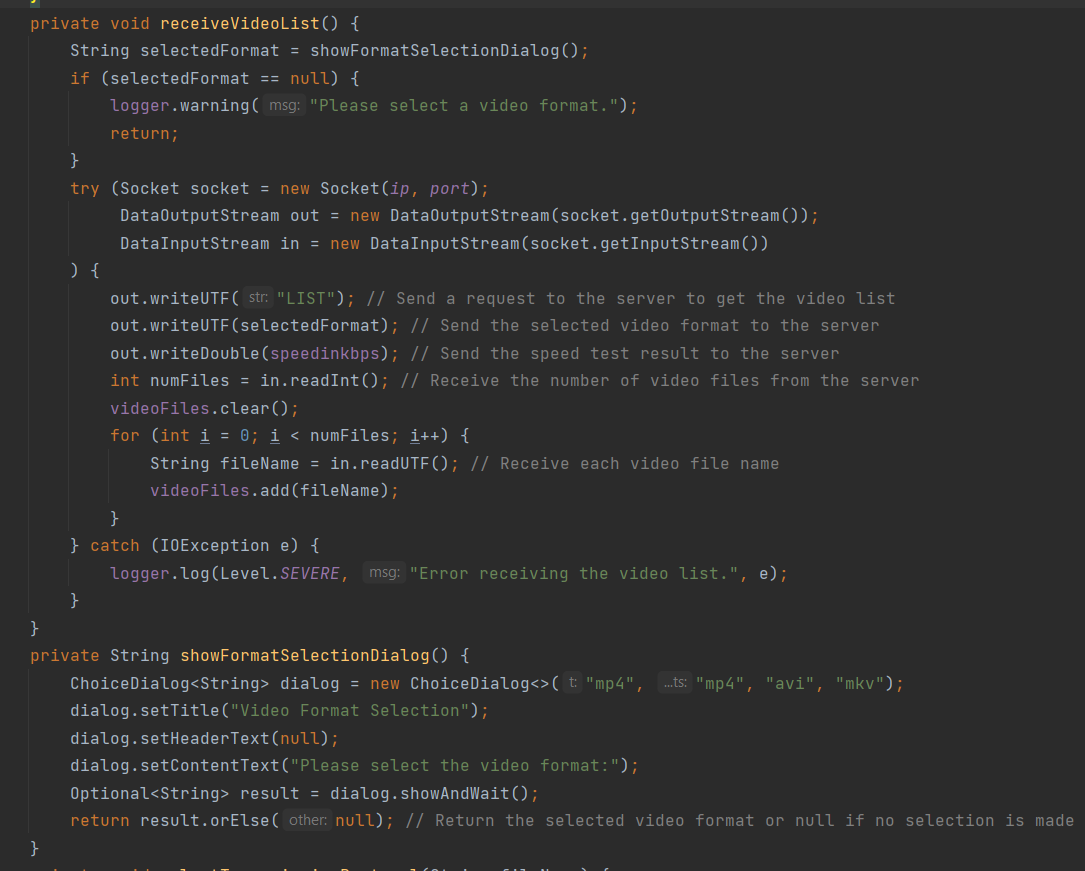


performSpeedTest method measures the download speed of a file from the given URL.

It opens a connection to the URL and reads the data without doing anything.

The duration of the download and the size of the file are used to calculate the speed in kilobits per second (kbps).

getFileSize method retrieves the size of the file in bytes by sending a HEAD request to the URL.



receiveVideoList method is called when the "Get Video List" button is clicked.

It retrieves the selected video format from the user using a choice dialog.

It establishes a socket connection to the specified IP address and port.

It sends a request to the server to get the video list, along with the selected format and the speed test result.

It receives the number of video files from the server and clears the existing list.

It receives each video file name and adds it to the videoFiles list.

showFormatSelectionDialog method displays a choice dialog for selecting the video format.

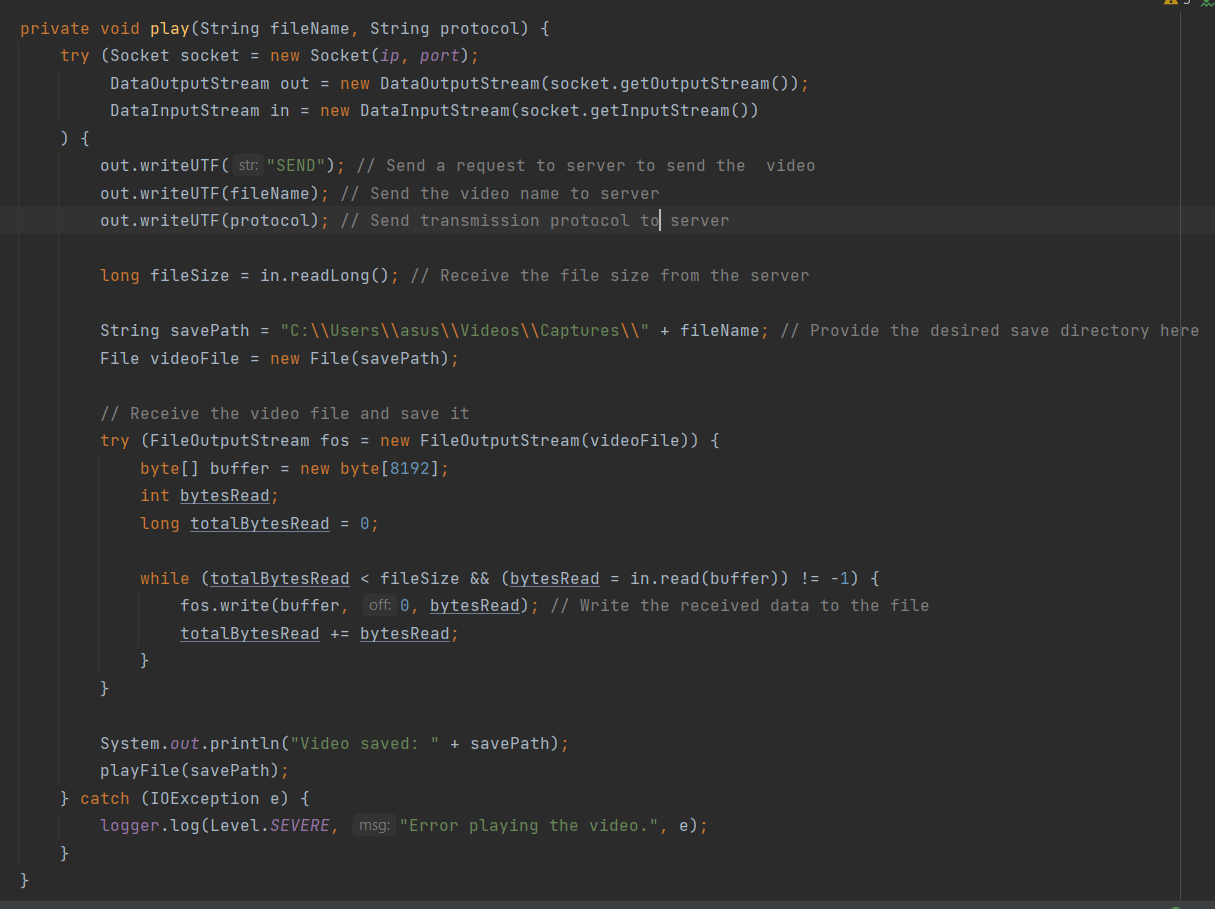
It presents the dialog with options "mp4", "avi", and "mkv" and returns the selected format or null if no selection is made.

A picture containing text, screenshot, font

Description automatically generated

selectTransmissionProtocol method displays a choice dialog for selecting the transmission protocol.

It presents the dialog with options "TCP", "UDP", and "RDP" and calls the playVideo method with the selected protocol if any.



The play method establishes a connection to the server by creating a socket and providing the server's IP address and port number.

Next, it sends a request to the server, asking it to send the selected video. The request includes the file name and the desired transmission protocol.

The server responds by sending the file size back to the client.

A File object is created using the savePath to represent the location where the video will be stored on the client's machine.

The video file is received from the server and saved on the client-side using a FileOutputStream.

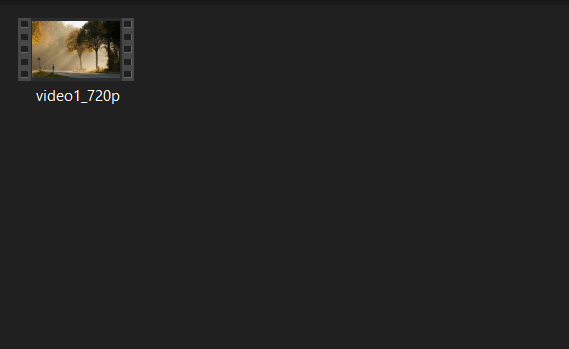
The received data is read into a buffer and then written to the file, ensuring that the entire file is received by keeping track of the total number of bytes read.

Once the file is successfully saved, a message is displayed, indicating the path where the video has been saved.

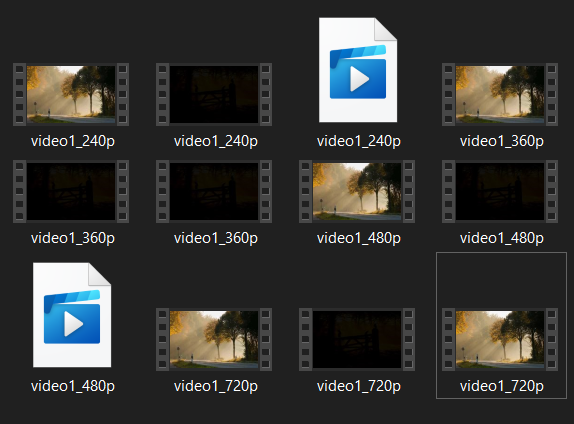
In case any errors occur during this process, such as network issues or file handling problems, an error message is logged to provide information about the issue.

**Presentation of the Final Implemented System:**

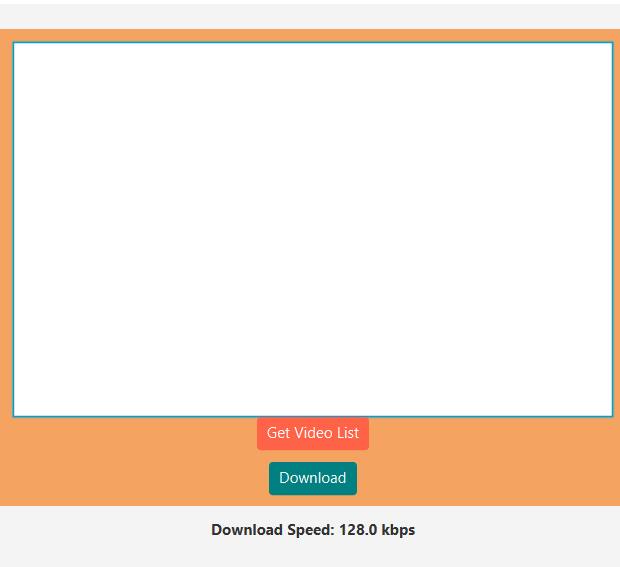
First we run the server code, the initial video file name should be in format **“videoname\_resolution.filetype”**



Automatically the video converter gets to work in creating all the copies.



In this case the original file is video1\_720p.mp4, we can see 12 total files which is 3 for every resolution below 720p.

After we run the client, and it connects and a GUI pops up. 

We can see the Download speed at the bottom of the GUI window, which will afect what videos we get when we click “ Get Video List”, after clicking it we see A screenshot of a video format

Description automatically generated

With a dropdown of video formats possible.

After choosing mp4 we get the desired format with the resolution based on the download speed. A screenshot of a computer

Description automatically generated with medium confidence

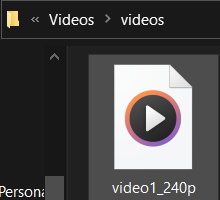
Then we click on the video and “download” we can choose the transmission type

A screenshot of a computer

Description automatically generated

After clicking ok the video is played using preferred media player

( 2 of the mp4 files generated are corrupted so it will show an error when played)

The rest of the files will play using media player normally. 

And it will download in the specified videos directory.

**Presentation of the Problems Encountered and Their Solutions:**

During the implementation of the streaming server code, certain challenges were encountered, and appropriate solutions were devised to address them.

**Video Corruption Issue:** Initially, video corruption was observed during the video conversion process. Specifically, the method responsible for generating missing files was not functioning correctly, resulting in the generation of only one file while corrupting the remaining files if more were generated.

Solution: To tackle this issue and minimize video corruption, improvements were made to the video conversion process, particularly to the "generateMissingFiles" method.

Solution:

* Refactoring the Code: The code responsible for generating missing files was carefully examined, and potential errors were identified. The code logic was modified to ensure accurate file generation and prevent corruption issues.
* Error Handling and Logging: Error handling mechanisms were implemented to capture and log any errors encountered during the file generation process. This allowed for the identification of specific issues causing corruption and facilitated troubleshooting.

The modified "generateMissingFiles" method now generates all the required files accurately.

Another problem encountered is,

**Video List:**

The video list from the specified directory wasn’t showing when you ran the program, despite the existence of video files in the specified directory, the server failed to retrieve and present the video list to the client.

Solution:

* After I inspected the code, I found that the problem was the retrievevideofiles and receivevideolist methods, so I tried implementing the method in a different approach.
* Extra error handling has been added so the code will display a message when an error happens.

**Video files generated:**

I ran into another problem with the video generator where it started generating extra videos than what is needed.

Solution:

* There was an error when generating missing files where it was not filtering what to generate properly, I fixed it by adding the filter where it does not generate files above the current resolution.

**2 mp4 files are always corrupt:**

When I generated the video, you can see in the screenshot provided that there is 2 mp4 files corrupt, this error followed throughout the project, and I could not find the reason why it happens. So, this error I left open.

**TCP, UDP, RTP/UDP:**

I have implemented some method for these transmission protocols, but I can’t get them to transmit the video properly and play it. This error is still open.