RhythmRipper

Team 9: Arnav Doshi, Sidharth Krishnaswamy, Owen Hall

Proposal

Arnav worked on researching Youtube's API documentation and finding out how to download Youtube videos from their server, converting it to an mp3/mp4, and packaging this script into a simple executable to run in the Java program.

Sidharth worked on researching elements for the GUI and the audio and video playback. This included researching JavaFX documentation, finding Java libraries that supported running .exe's, and creating a video playback bar using Java's native FX libraries.

Owen worked on researching the class design and the JavaFX/FXML GUI. This included finding a simple design pattern to follow, finding OOP concepts to implement, and further researching any JavaFX documentation to include.

Presentation

Arnav worked on creating the problem and solution slides, and other concepts slides, in order to explain to the audience the aspects of the presentation.

Sidharth worked on the demo for the presentation, and added relevant images to the slides. This included giving a live demonstration of the working code.

Owen worked on creating the class diagram and explaining to the class. He also talked about the user interface and the multiple versions of RhythmRipper.

Code/Report

Arnav worked on creating the Youtube to mp3 and mp4 downloader and converter in Python, packaging it into an .exe and .unix file, and creating the Mac version with the visual player. This included researching python libraries yt-dlp and ffmpeg and implementing them, and then using programs such as pyinstaller and cx_freeze in order to package the script and all the dependencies into a singular .exe or .unix file. Due to the differences between Mac and Windows, Arnav also had to create a Mac compatible version of RhythmRipper which uses .unix files and uses Mac video encoding (yuv420p pixel format) in order to display the mp4 on Mac.

Sidharth worked on the main Windows GUI and the audio and video playback and combining the Java and Python code. This included creating the main logic for the program, using ProcessBuilder and JFX to properly create the UI and buttons, and utilizing relative paths in order to get the users' proper downloads. This also included designing a process that invoked a thread to combine the python script and the Java, and creating a video player for Windows that opened the .mp4 program which was saved on the Desktop by the script.

Owen worked on JavaFX/FXML, class design, and design and aesthetics. This included creating the different Java controller classes and FXML scenes, focusing on implementing OOP principles such as abstraction, inheritance, and encapsulation and creating the general design for our program and all the state changes in the MVC-designed application. Owen utilized various methods to achieve an effective modern design such as gradient (shadow) effects and event handlers. To design the fxml scenes, he utilized Scene Builder, and implemented many layout controls such as Anchor Panes, Stack Panes, HBoxes, and VBoxes.

Problem/Issues

Many users face significant challenges when trying to download YouTube audio and video content without a YouTube Premium subscription. Online YouTube to MP3/MP4 converters often come with numerous intrusive ads, creating a frustrating and cluttered experience. Additionally, these converters frequently have untrustworthy appearances, raising concerns about security and privacy. The user interfaces are often poorly designed, making the tools difficult to navigate and use efficiently. These issues collectively hinder a smooth and reliable way to download and enjoy YouTube content offline.

Previous works

https://ytmp3.cc/3kRz/

- Has Ads
- Website and clunky interface
- Does not have a built in player
- Lots of text on the screen
- Cannot tell what is going on and what we are downloading

Assumptions/OS/Intended Usage

Assumptions: NO foreign letter / special character videos allowed, youtube video exists (error handling exists for non-youtube links)

OS can be Windows or Mac, provided that the user downloads the correct version off the github.

Intended usage: The user can input a valid youtube link or playlist and listen and watch Youtube videos on their own device using our program. They can also see everything that happens behind the scenes.

Diagrams

Class Diagram:

https://github.com/arnav-doshi/CS151-RhythmRipper/blob/main/diagrams/final_class_diagram.png

Sequence Diagram:

https://github.com/arnav-doshi/CS151-RhythmRipper/blob/main/diagrams/final_sequence_diagram.png

State Diagram:
https://github.com/arnav-doshi/CS151-RhythmRipper/blob/main/diagrams/final_state_diagram.png

Use Case Diagram:

https://github.com/arnav-doshi/CS151-RhythmRipper/blob/main/diagrams/final_useCaseDiagram.png

Functionality

Our solution addresses the issues of downloading YouTube audio and video without requiring a YouTube Premium subscription by providing a native desktop application with a clean, ad-free interface. The application ensures transparency by clearly communicating its processes to the user. It allows for seamless downloading and

playback of both individual videos and entire playlists, offering a user-friendly experience without the clutter and risks associated with other online converters.

Operations

1. Users search up Youtube video (or playlist) they would like to download. User copies the link address.
2. User inputs the link address into the RhythmRipper text box. The user then clicks the “Convert” button.
3. The user waits for the program to finish converting. The user can then click on either “Play/pause” or “Open Video”, depending on whether they want to hear the audio (mp3) or see the video (mp4)
4. The user can also play or pause the audio, and skip through if they want to.
5. The user can enter another youtube link if desired and repeat the process.

Solution

The provided solution addresses the problem of downloading YouTube audio and video without a YouTube Premium subscription through a Java-based native desktop application using JavaFX for the user interface and Python for backend processing. The application named Rhythm Ripper, features a clean, ad-free interface that ensures a user-friendly experience. The UI is built using JavaFX, offering a modern and responsive design with components such as a TextField for entering YouTube links, buttons for initiating conversions and playing media, a TextArea for displaying logs and feedback, and progress bars for indicating download, conversion, and playback status.

After entering a YouTube link and clicking the convert button, the application verifies the link's validity using an HTTP GET request. If the link is valid, the application invokes a Python script using ProcessBuilder, which handles downloading and converting YouTube content to MP3 and MP4 formats. The Java application captures real-time output from the Python script, updating the UI to reflect the conversion status. Once the audio file (MP3) is downloaded, it uses JavaFX's Media and MediaPlayer classes to enable playback within the application, with a synchronized progress bar showing playback status and allowing users to seek within the audio.

For video files (MP4), for the Mac version, the application converts them to a QuickTime-compatible format using ffmpeg, invoked through another ProcessBuilder process. The UI provides feedback during this conversion, and the “Open Video” button becomes available upon completion. Error handling is included, with pop-up alerts informing users of invalid links or conversion issues, ensuring transparency. The

application is designed to extend functionality for downloading entire playlists by sequentially processing each link.

By integrating JavaFX for the interface and using Python (using ffmpeg and yt-dlp) for backend processing, Rhythm Ripper offers a seamless, efficient solution for downloading and playing YouTube content while maintaining transparency and usability.

Steps to run code

1. Download the correct version for your OS (demo = Windows or MACdemo = Mac)
2. If Windows, then run MainController.java, otherwise run HelloApplication.java for Mac
3. Copy a *valid* Youtube link or playlist link from Youtube. Please make sure it has NO special characters (such as foreign letters) in the title.
4. Paste it into the text box and click convert.
5. Wait for the video to download, and click on play audio or play video, depending on what you'd like to do
6. Fast forward or pause the audio as desired
7. Check if mp3 and mp4 are downloaded on your Desktop. On mac there are 2 mp4 files, one encoded one non encoded.
8. Paste in another link and repeat steps above if desired.

Problems we faced while developing

Some issues that we had encountered (but fixed!) included:

- Youtube's API blocking youtube downloads, we had to rely on another library in python named yt-dlp
- Packaging all python dependencies into one script. Researching and finding that we needed to use cx_freeze and pyinstaller to create an exe took a long time
- Local pathing issues existed from system to system. We wanted to make sure that any user could use our program with ease. We had to rely on a lot of relative paths.
- Integrating java and python into the same program. We settled on creating a .exe with the python and calling that in the java program.
- Working with deprecation issues. Lots of JavaFX libraries were deprecated and we had to find workarounds. We wanted to include a native video player into our application, but since the player was deprecated, we had to settle on opening the downloaded video on the user's home system.

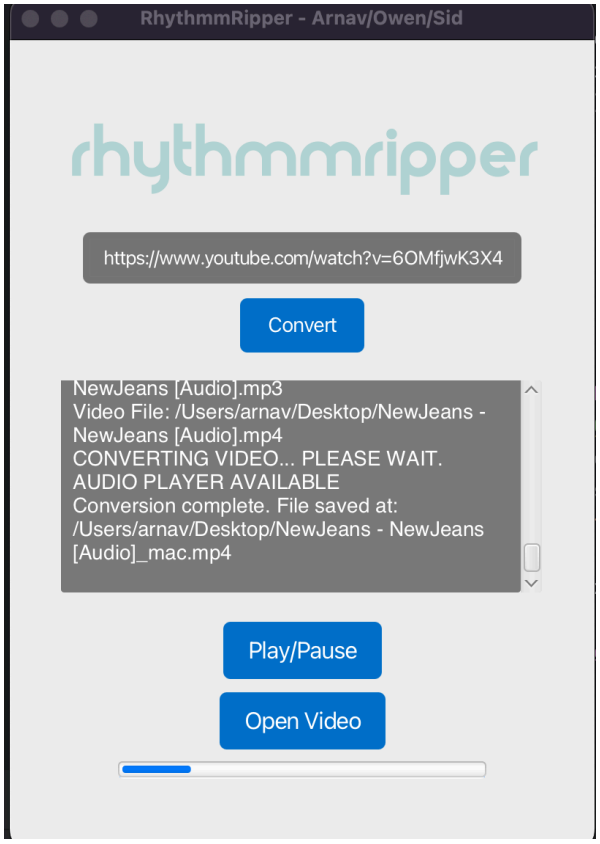
- Windows versus Mac proved to be a big issue within our project. The python script, at first, was a .exe, and that was not runnable on MacOS. Instead, we had to create a UNIX executable file, and create a separate program for Mac. Additionally, when using a .mp4, Mac requires additional encoding, so we had to use a library to “yuv420p” encode the downloaded video, so it was playable.

Some issues we need to fix:

- Foreign characters do not work on videos. This should be fixable using Unicode encoding, but we ran out of time.
- Fully verifying a youtube video’s existence. We can only verify that a link exists for the video, not that the video exists itself.

Snapshots

Mac/lightweight version:



Windows version: