MAY AI Hackathon

Submitter: Michael Jefferson, Pinpoint Global Communications

Project: Build a web site that lets users upload mp4 files, and generates text transcription, closed caption files, and creates language specific versions of the uploaded video.

Live URL: https://medicare2.uat4.pinpointglobal.com/AIMediaPlayer/

Source Repos: Windows Service: https://github.com/MichaelPJefferson/MPAuditExtractV6

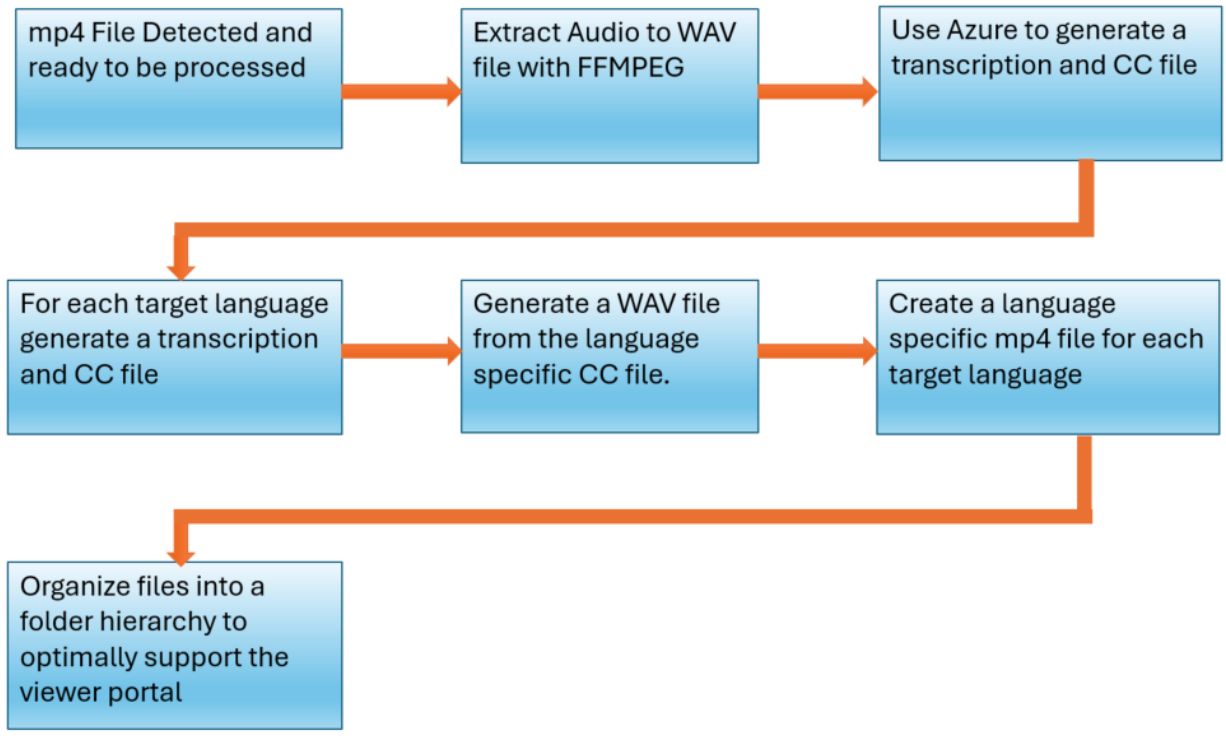
Front End Viewer: https://github.com/MichaelPJefferson/MediaPlayerV7

Approach: Being brand new to AI, I decided to start with a project that borrows a bit from a project we've already implemented to some degree, rebuilt It from scratch and expanded it in some different ways.

Project Idea: Pinpoint has clients that require multilingual support, and I wanted to build a tool that would assist in media content creation. I designed and built two components, a front-end web site that would let users upload a video and view resulting jobs, and a back-end Windows service that would leverage FFMPEG in a variety of ways to generate language specific artifacts that could be easily leveraged.

Tools and Methodology: After researching some of the tools, I initially wanted to use Bolt.New to generate projects, but I was not pleased with the lack of integration into Visual Studio. I felt that I was wasting too much time translating the recommendations from Bolt into my Visual Studio 2022 solution, and I was quickly running out of daily token allocation. I pivoted to using GitHub Copilot with Visual Studio. I found ongoing lengthy and very granular interactions with Copilot to be rewarding and very productive. Frequent interactions for focused, targeted ideas coupled with other best practices of testing frequently, periodic check-ins for establishing new baselines, and making most functions small and granular in purpose, helped move the project along quickly.

Transcoder Workflow



Challenges: This approach helped make pivotal design decisions along the way (for example, for the transcription, I initially used a component called VOSK primarily because it was royalty free. The initial model had too many transcription errors, and the more robust model was more accurate but lacked punctuation which made the phrasing awkward at best. Consulting with GitHub Copilot led me to change course and pursue Azure Cognitive Speech as a back-end transcription service. While there is cost associated with its usage, it seemed like a far more robust solution.

My initial hope was that I could generate an mp3 file and use as an alternative audio track for playback within the portal. The HTML 5 media player would not support this, and even though its possible for embedding multiple audio tracks into a single media file, it seemed like all but highly specialized playback engines would only load and play the first embedded audio track. Accordingly, I decided to generate language specific mp4 files and to do that, I needed to generate language specific audio files, then use FFMPEG to copy the video stream from the original mp4 file, and the audio stream from the translated language file, to create a language specific mp4 file. In order to maintain “lip sync” as best as possible, I ended up using the closed caption file, with its phrasing and its timecode indicators, to build the best possible language specific audio track.

Initially the closed captioning was not appearing. Using developer tools I determined the VTT file was not loading, despite having he correct filename and location. Ultimately I figured out that the VTT Mime Type was not configured either locally or on the IIS server that I hosted to solution on, and I used Copilot to help me identify the correct way to resolve in the two different use cases.

Identified and resolved a reentrancy issue. As I pivoted to different solutions, I ended up needing to generate language specific mp4 files, and the final steps of the encoder (organizing and moving files into their final folder structure) found that files were locked as in use. These files were unable to be moved to their final folder hierarchy destination. I found that FFMPEG processes were still running when my service completed, and realized that the file watcher designed to monitor for new mp4 files being uploaded to the service, was picking up on the language specific artifacts I was generating, and started to process those files as if they were new files to operate on. Once I excluded those language specific variants, I was able to resolve the locked file issue. Along the way, I built an option for the windows service to run as a console application to facilitate debugging.

Extensibility: Adding additional languages is trivial. I refactored the code so that specifying new languages is as easy as adding new configuration values in the appsettings.json file. As an experiment, I added Greek in addition to the primary languages for

Possible Next Steps:

1. Build monitoring for Azure Cognitive Speech utilization.
2. Tighter integration between the front end and back end to show the user when the windows service is processing files.
3. Expose language selection capabilities into the front end to let the user select which languages would be desired
4. Monetization: Add a shopping cart implementation to charge the end user on a per upload basis for the language transcription and media repackaging tasks.