1.1 Sample REST API application

Here we give an overview for how you might satisfy the criteria for assessment item 1 using a video application. Note that many other types of applications are possible. This is a sketch of a basic application showing the core functionality related to the criteria for the assessment item. There are many ways that you could modify or extend this functionality to improve the app and achieve a higher grade.

You can ignore this if you like.

Core criteria

Here we have omitted the criteria related to Docker and deployment as they are more or less independent of the application functionality.

- **CPU intensive process:** video transcoding
- **Method for generating load:** videos can be uploaded ahead of time and requests to transcode to a new format made via the web client or an HTTP request
- **Data:** video files, user ownership of videos, session tokens
- **REST-based API:** the app is implemented with API endpoints for logging in, uploading a file, requesting a video be transcoded, downloading a file, and listing a user's files.
- **User login:** User authentication is handled via a simple hard-coded list of users and passwords. A JWT is returned which is used by other calls to the API to identify the user.

If competently implemented, the above is sufficient for a 5.

Implementation sketch

There are many possible ways of implementing this sketch. Below are some reasonable starting points.

- NodeJS and Express are used to implement the app
- User sessions managed using the **jsonwebtoken** **(https://www.npmjs.com/package/jsonwebtoken)** or **express-session** **(https://www.npmjs.com/package/express-session)** npm module
- **ffmpeg** **(https://www.npmjs.com/package/ffmpeg)** or **fluent-ffmpeg** **(https://www.npmjs.com/package/fluent-ffmpeg)** npm module used for transcoding. Note that you need to separately install `ffmpeg` itself with `sudo apt install ffmpeg` or use **ffmpeg-installer** **(https://www.npmjs.com/package/@ffmpeg-installer/ffmpeg)** to automate the installation.
- **express-fileupload** **(https://www.npmjs.com/package/express-fileupload)** npm module used for video file upload.
- Video files downloaded to the client using Express's built in **res.download** **(https://expressjs.com/en/api.html#res.download)** function.
- Video files are stored locally with a unique identifier. Video metadata is stored locally using a simple JSON based database like **lowdb** **(https://www.npmjs.com/package/lowdb)** , or a local SQL database like **SQLite** **(https://www.npmjs.com/package/sqlite)** .
- The client-side application can be implemented using plain HTML, JavaScript and CSS.

Additional criteria

There are many ways of extending the basic sketch to satisfy additional criteria. We give some examples here based on the idea of extending the basic sketch to a YouTube type video sharing application.

- **External API:** Query another source (eg. YouTube) for content related to a particular video, showing links, thumbnails, and titles in the video's page (similar to how YouTube shows other videos on the right when playing a video.)
- **Infrastructure as code:** Docker compose used to deploy the app along with a standard MariaDB container for storing metadata. Environment variables used for configuring ports, passwords and other details. Note that MariaDB is also available as a AWS managed service so this will be easy to migrate later on.

If the above, and the core criteria, are competently implemented then this is sufficient for a 7. Other things are possible, these are just examples.