Tech Design

This document is to determine the technical design of the Soapbox coding assessment. All decisions described within are subject to change as more information is discovered.

# Technology Stack

The following technology stack has been decided upon:

* Elixir
* ReactJS
* JSON for response data.

Elixir will deliver the SPA ReactJS application in order to simplify this example. In a real world situation it would be better to have the ReactJS application served separately from the Elixir application because it would allow for independent updates to the application(s). The API responses will be in a JSON format because JavaScript and ReactJS both handle JSON natively. This will allow most applications to handle the response data, not just my own and JSON is a commonly accepted response format. The use of Elixir is based on the job being applied to.

# Required Endpoints

The following generic endpoints would be necessary. This is not an exhaustive list of endpoints available within the application:

| Type | URL | Purpose | Success Code |
| --- | --- | --- | --- |
| GET | / | Deliver the ReactJS Application | 200 |
| POST | /api/auth | Authenticate user for login purposes. | 200 |
| GET | /api/users | Retrieve the list of users | 200 |
| POST | /api/users | Create a user | 201 |
| PATCH | /api/users/:id | Update a user | 204 |
| DELETE | /api/users/:id | Delete a user | 204 |
| GET | /api/users/:id/videos | Get a list of the user’s videos | 200 |
| GET | /api/videos | Get the list of videos | 200 |
| POST | /api/videos | Create a video | 200 |
| GET | /api/videos/:id | Get the video content for a specific video | 200 |
| PATCH | /api/videos/:id | Update a video | 204 |
| DELETE | /api/videos/:id | Delete a video | 204 |
| GET | /api/videos/:id/assets | Get the list of assets for a video | 200 |
| POST | /api/videos/:id/assets | Create a new asset | 201 |
| GET | /api/assets/:id | Get the asset with a list of edits | 200 |
| PATCH | /api/assets/:id | Update an asset and its edits | 204 |
| DELETE | /api/asset/:id | Delete an asset | 204 |

The default response codes are as follows:

| Code | Meaning |
| --- | --- |
| 403 | The user is not authorized to access. Verify that the auth token is valid and provided. |
| 404 | You are trying to hit an endpoint that is not found. |
| 422 | The JSON data provided is invalid for some reason (response will include the exact reason) |

# Application Design

The following models will be used within the system. The Primary Key is marked with (PK), the Foreign Key is marked with (FK) and required fields are marked by ®. Default values are marked with => and some fields will be updated automatically by the API after running tooling on them (such as the duration of a video).

| Model | Variables | Methods |
| --- | --- | --- |
| User | id (PK) username ®  password ® | create(username, password) auth(username, password)  delete(id) |
| Video | id (PK)  name ®  user\_id ® | create(name, user\_id)  update(id, data)  delete(id) |
| Asset | id (PK)  video\_id (FK)  src ®  name is\_original => false  duration | create(name, video\_id, src, edits)  update(id, data)  delete(id) |
| Edit | id (PK)  asset\_id (FK)  type ®  start\_time  end\_time  scale | create(asset\_id, type, data) delete(id) |

You will not access the edits directly but instead apply a series of edits to an asset. When you upload a video a source asset with NO EDITS will be created. No edits will be allowed on the original source asset. At this time, for simplicity, edits cannot be modified after saved however an asset will control its edits, creating and deleting edits as needed. Certain edits will be limited based on the type of edit. For example, a single cut will be allowed per asset. An asset will have a null source until it has gone through processing.

# Response Agreement

The response will be in JSON format with the following specified formats.

| Model | Response Format |
| --- | --- |
| User | {  id: <number>,  username: <number>  } |
| Video | {  id: <number>,  name: <string>,  owner: <number, user\_id>  } |
| Asset | {  id: <number>,  videoId: <number>,  src: <string, url>,  name: <string>,  isOriginal: <boolean>,  duration: <number in seconds>,  edits: <array of edits>  } |
| Edit | {  type: <string>,  startTime: <number in seconds>,  endTime: <number in seconds>,  scale: <number in seconds>  } |

Any “null” values on the edit list on assets will be stripped out.

# External Dependencies

The following external dependencies are required to run this application:

* ffmpeg
* Pheonix framework
* NodeJS
* bcrypt

We are using ffmpeg to convert videos because this is a free command that we can use and integrates nicely within Elixir. We are using the Pheonix framework for Elixir to handle the framework for the API and static asset delivery. This is simply due to personal comfort with the framework when working with Elixir in order to lower development timelines. NodeJS is used for installation and development of the ReactJS webpage.

Bcrypt will be used for generating secure password hashes to lock down user accounts. Ideally I would have no way of knowing what a user’s password is and could at best reset the password. This is for security reasons though a possible future feature would be an “act as” concept on user accounts so that I can see what issues users are having (would be mostly useful for permissions based things).

# UPDATES

The following has been updated as I have learned more:

* I am combining Guardian and BCrypt for authentication using a JWT Token based authentication.
* I have updated users to have a “role” so that there can be different scenarios of users. Any new user will be a standard “user” role but a user may be updated to have an “admin” role which allows them to see any/all videos within the system and not just their own. A future improvement here would be to add a “Company” concept with another role around managing the company. The company user(s) could then adjust users up to their role (“user”, “supervisor”, etc) but not “admin”. Only “admin” users would be allowed to make users admin.
* Forgot to add a dependency above. I am using react-router-dom to handle frontend displaying of data. I am not guaranteeing that data will be saved at this point as I would like to finish the application as soon as possible (11/5/2020).
* Scaling as suggested was found to be more complex than anticipated. Trimming the video length and rotating the video are implemented but scaling would take further research as I am not as familiar with FFmpeg as I would like for the scaling.

# Planned Improvements

There are several improvements that are planned at the time of writing this:

1. Allow viewing and editing of any asset. Currently only the last asset can be viewed or edited (chosen due to time constraints).
2. Allow uploading of larger files via progressive upload. Currently only video files up to 1GB can be uploaded. A progressive upload API would be needed for videos that are much larger as otherwise the upload would time out.
3. Scaling the video. This was originally planned as an integral piece however my inexperience with FFmpeg caused me to leave this off in favor of other video transformations.
4. Separate pieces out further into reusable components. Due to time constraints I found myself scrambling to piece things together as fast as possible. This lead to some pieces not being as compartmentalized as I would have liked. It would also be nice to increase the number of reusable components drastically, such as the video container with the custom configurations.
5. Implement user admin screen for admin users (currently not possible to create an admin user via the UI so should not be an issue).

# Lessons Learned

* The biggest thing I learned during this project is how to use FFmpeg. While I have used FFmpeg based utilities before, I never had to deal with FFmpeg directly. With this project I had to as the Elixir tool FFmpex talks directly to FFmpeg.
* Video editors are hard to make. Honestly, I figured this would be much simpler or at the very least that I could pull in a pre-made version to make use of. This was definitely not the case and I ended up rolling my own.
* HTML5 video components are relatively simple to recreate if you have enough time and patience. This means generating a custom player based on a user profile is completely within reason.
* CSS Gradients to generate a progress bar while playing the video. I had never used CSS gradients before this but found it rather complex to get the desired result. That being said, I understand what is needed to make things work better for the CSS and would definitely consider rewriting portions of it if in a more complex environment.
* Sometimes giving the bare minimum functionality just to get something completed is the best way to move forward. I tried giving a complete application for this assignment but ended up feeling burned out when I was informed that I had to speed up my development process drastically (was told Wednesday that I had to have something completed by Monday if I wanted to still be within consideration). This lead to both stress and corners being cut that would have otherwise not been cut if I had spent the time determining the bare minimum and developing towards that with nice to haves being added later. I have gotten so used to “we want these nice to haves tomorrow” with my recent roles that I simply forgot to take a step back and think about what a “nice to have” actually is in reference to this project.