YouGo Pics or it didn’t happen

(Part 0)

Software Design Document

Version (1.00)

Created (Apr-25-2022)

Updated (Apr-29-2022)

Table of Contents:

Introduction:

This project is intended to successfully complete the requirements of the YouGo Pics or it didn't happen (part 0) Backend Project located here.

Summary:

The project requirements are to design and deploy a service that exposes 5 endpoints over a HTTP(S) connection that allows a user to send an image, does a manipulation to that image and then returns the manipulation to the user. The supported manipulations are:

- Image Resizing
- Grayscaling
- Transformation
- 2 other manipulations supported by a chosen 3rd party package

What is meant to be delivered:

- A swagger spec for the API
- The completed and launched API
- A general README for the project
- A document describing the design decisions made during the project and their reasonings

*Note: This document *DOES NOT* constitute the previously mentioned design decisions document that needs to be included.*

Goal:

Completing this project is meant to be educational, simulating being given requirements in a professional setting and building a Backend service based on those requirements while making it as professional as possible. This document is also meant to play a part in that goal and will serve as a place for potential designs to be discussed and 'agreed upon'.

Intended Implementation:

This section will focus on the intended implementation of the Backend service.

Technologies Used:

- **Node.js / Express.js:** I am a Front-End Engineer by training and have extensive knowledge of JavaScript so I felt development would go faster using Node which I had some experience with than also learning a new language to complete this project
- **TypeScript:** Provides type-checking, is a standard choice for any new professional project and being able to write ES6 JavaScript and having it compiled down to ES5 JavaScript made the development experience a much more enjoyable one while using LTS Node.
- **Sharp:** Listed as a suggested package for image manipulation in the original project requirements doc. Is a go-to package for image manipulation so I went with it.
- **Jest / Supertest:** Jest is the go-to package for testing JavaScript applications and Supertest was a recommended package for testing APIs.
- **Winston:** Winston is a highly recommended package for doing logging in JavaScript.
- **openAPI 3:** The standard for documenting web APIs
- **Swagger UI Express:** A package for implementing a Swagger UI page that allows the user to learn about the API and make test API requests.

Endpoint Design:

The endpoints have been split so that each action has its own endpoint. This was done for an easier user experience, requiring less parameters to apply a manipulation and having an expectation of the result through the endpoint name. It also is an easier developer experience for myself since I could make a controller for each endpoint. The endpoints would be accessed by /image/*manipulation action* using the HTTP POST method.

The endpoints are:

- /image/resize
- /image/grayscale
- /image/transformation
- /image/negate
- /image/blur

If an endpoint is accessed with anything other than a POST to the above endpoints, the server will reply with a ‘not implemented’ error.

Request Format:

The endpoints will accept a POST request with JSON as the message format with all the parameters in the JSON object. The image is sent over as a base64 string of the format: “data:*Mime type of image*;base64,*Base64 image string*”.

Response Format:

Responses are sent in JSON format with a message object containing either the manipulated image in the previously stated base64 format or sending an error message that resulted from attempting to complete the manipulation.

Request Validation:

The user request is validated by a custom written validation by each endpoint. Each endpoint validates that the base64 string exists and is in the format that is expected and then if needed validates specific parameters for the endpoint.

API Documentation:

openAPI is used to document the endpoints. The file with the openAPI specification can be found here. Note that while there are example base64 images in the examples, they do not work and will throw an internal server error if used. They are only there to represent that a base64 image is needed.

Swagger UI Express is used to automatically generate the Swagger UI web page to view the openAPI specification in a user-friendly way as well as test the API from that page.

Testing:

Each endpoint is tested for the successful completion of each image manipulation and then for each path that would violate the specific validation rules for each endpoint. Any utility functions written will also be tested that they operate correctly.

Logging:

Each request from the user and each response to the user is logged. In the handling of the response, notable events will also be logged such as the successful conversion of a base64 string to an image is logged, the image being successfully manipulated and then the manipulated image being reencoded. Errors will also be logged, separately from the general log.

Deployment:

The completed API is hosted on Heroku initially with plans to migrate to another platform in the future. Heroku’s free tier does not provide the ability to write data to the filesystem so log files will not be available to the application. Heroku however saves messages sent to the console which can be utilized for logging.

Retrospective:

Links to completed artifacts:

Live application: https://you-go-nodejs-project.herokuapp.com/
Live Swagger UI: https://you-go-nodejs-project.herokuapp.com/api-docs/
Code Repository: https://github.com/chadstewart/you-go-backend-project
Design Decisions Document: https://github.com/chadstewart/you-go-backend-project/design-decisions/

Reflections:

This project took me 3 months to complete, working on it off and on. The hardest part was actually utilizing base64 images as it was a technique I hadn't done before. I believe that it took me a few weeks to implement the initial version of that feature.

I learned a fair amount about buffers in working with this project. When I initially implemented the base64 feature, I purposefully had the image written to disk and read the image from disk. This was because this was what I was used to working with. I decided that for the sake of performance, I'd try to implement the project moving the buffer around rather than writing to disk. This allowed me to utilize the asynchronicity of JavaScript by writing only the result to disk while finishing the result to return to the user. That code is currently not in use but it will be as the project is expanded upon.

Ironically, the initial implementation of writing the image to disk would not work deployed on Heroku.

Testing an API was also new to me, initial tests would often fail because the server teardown at the end of the test would not run correctly resulting in the test hanging. Some research led me to the pattern of describing the server in one file and then actually starting it with another which ultimately solved that issue.

What would I do differently:

To be quite honest, I don't think there's much I would change. A lot of the hardships of delivering on this project taught me a lot about JavaScript, Node and TypeScript. Any change in the way I went through the application would potentially change what I learned which was quite valuable. I do have two things I would consider changing though:

1. I wish I started researching the base64 decoding earlier. The major reason it took so long was because I was depending on outside help but was unable to meet with them. I ultimately was able to deliver on the feature on my own.
2. I deployed to Heroku because I have prior experience there but I wish I chose to deploy on AWS. I have an interest in learning more about AWS and want to take the opportunity to deploy a project onto the service to learn more about using it.