IGME 430: Rich Media Web App Dev II
Project 1 Overview

Table of Contents

Overview

Web APIs are ubiquitous in the modern world as a means of accessing and interacting with a dataset. Oftentimes companies provide APIs to provide limited or complete access to the data they have and manage. Many of these APIs are publicly available such as the Google Maps API, Spotify API, PokeAPI, and many others. They allow other developers to build rich web applications and experiences that are augmented by the data they get. Private Web APIs are also used by nearly every company to pass data between various internal systems and tools.

For this project, you will not be utilizing one of these preexisting APIs. Instead, you will take a JSON dataset and create a Web API that allows others to access, search, filter, and edit your dataset.

Project Description

Individually, you will be building a Node.js web server that both hosts static files such as HTML, and hosts a backend web API **of your own creation**. The server will host a simple webpage that allows people to interact with your data without writing code, as well as a page that documents the endpoints of the API for developers to use.

It is important to note that the web API is something that you are developing. That is to say: you will not be leveraging an external API but rather building one from scratch by yourself. It is not uncommon for people to make the mistake of leveraging an external API rather than using one they built.

Usually APIs will store data in databases, but since we have not learned how to use a database yet your server will read data from a file at startup and use/manage that data. This means any data added by users while your server is running will disappear when the server restarts. You are not expected to persist data beyond the runtime of the program.

You may choose one of the datasets available on MyCourses, or find one on your own (see below). In addition to building the API, you will create comprehensive documentation for it for other developers to use.

API Datasets

Several API datasets have been provided on MyCourses for your API to use. These datasets are contained in .json files. Upon startup, your Node server will have to read these files into memory and parse them into JavaScript objects for runtime use. You are **highly encouraged to use one of the provided datasets**. You may, however, use one that you find or make. However, there are a few things you must confirm before you can use it.

1. The data must be in proper .json format.
2. The dataset must have some amount of data nesting. That means there must be an object or array nested inside of the individual data objects.
3. You **must** understand how that data is organized and be able to properly parse it with JavaScript. If you are not able to figure out how to parse your own dataset, you cannot use it.

Feel free to speak with your professor if you want to know if a dataset is acceptable.

Project Requirements

The following are all the criteria your project must meet.

Node Server / API

- Data from .json files must be loaded into memory and parsed at server startup. All future access to data should be from the resulting JavaScript object.
- Must use all of the following status codes in the appropriate places:
  - 200, 201, 204, 400, 404
  - Additional status codes should be used if/where applicable
- Must have <u>at least</u> 4 GET endpoints to retrieve different data, along with support for HEAD requests to those same endpoints.
- At least one GET endpoint must support query parameters for filtering/limiting the data returned by the API. If query parameters are applicable for more than one endpoint, they should also be used there.
  - Data filtering should be done by the API, not by the client webpage.
- Must have <u>at least</u> 2 POST endpoints for adding and editing data.
  - POST request endpoints should accept incoming body data in both JSON and x-www-form-urlencoded formats, and parse them based on the Content-Type of the POST request.
  - Note that GET/HEAD requests should never add, modify, or remove data from the server.
  - Be aware that we have not covered data persistence, which means data added to your API will disappear when the server restarts (every 30 minutes on Heroku). You are not required to persist data permanently, but it should stick around between requests until the server restarts.
- Endpoints must have proper error handling for invalid / bad requests, etc.
- Must return a 404 response if the user goes to a non-existent endpoint.
- Must set the following response headers for all responses:
  - Content-Type
  - Content-Length
- All data endpoints must support and default to JSON responses. Support for other response types is optional, but should be controlled by the Accept header.
- Any static files needed for the client facing website (HTML, CSS, Client JS, images, videos, etc) must be served by your server.

(Requirements continue on next page)

Client Facing Website

- The main page must display a simple web interface for viewing and adding data using your API. The user should not have to write any code for this to work. Imagine interfaces like the ones we have used for past assignments.
- Client forms must utilize fetch() to make API requests to the server, and must cancel the default form actions.
- All requests made by the client website (including from the API interface on the main page) should send the Accept and Content-Type headers where applicable.
- A documentation page containing formatted endpoint documentation for all API endpoints (all endpoints that return JSON data, not static pages/files).
    - For each endpoint, provide the following information:
        - Relative URL
        - Supported Methods
        - Query Params (where applicable)
            - Param name, type, required/optional for each.
        - Body Params (where applicable)
            - Param name, type, required/optional for each.
        - What the endpoint returns and its format.
        - Examples of usage

Functional Requirements

- Uses Git for version control in a repo that the professor can access.
- Uses ESLint with the Airbnb spec for all server code.
- Uses GitHub Actions for build testing.
- Uses a proper .eslintrc file and proper .gitignore file placed correctly in the repo.
- Uses a cloud service (such as Heroku) for deployment.
- Borrowed code and code fragments must be credited in code comments and in the written documentation for the project.
- Separation of Concerns: your code should be appropriately broken up into files and functions based on the main functionality of those pieces of code.
- D.R.Y. - Don’t Repeat Yourself. If you have multiple nearly identical blocks of code, those should be factored out into separate functions.
- Code must be well commented. You don’t need to comment on every line. Have a comment for each function, and comments for confusing lines of code. Ideally, code should be “self-documenting”, meaning the variable and function names explicitly state what they are and what they do.
- Code must be free of runtime and ESLint errors.

Above and Beyond Work

As detailed in the grading rubric below, completing all the above requirements will yield a grade of 90% overall. The last 10% is reserved for above and beyond work. That is to say, to get above a 90% on this project you must go beyond the requirements.

Some examples of above and beyond work would include: utilizing a package from npm to aid in the development of your server/API, styling your client facing web pages using a CSS framework for a more polished final product, integrating an external API that interacts with your own API to create a more full-featured user experience, etc.

Points for above and beyond work will be distributed based on the size and quality of the implementations. Something that takes a line of code and a few minutes of work will net a few points. A larger scale integration that drastically improves the application will yield a maximum of 10 points.

Be aware that there are some NPM packages that would trivialize this project, such as one of the numerous web server frameworks. We will be covering those after this project, and so they are not allowed for this project in any capacity. If you are unsure of if you could use a package or not, speak with the professor.

Deliverables

There are two deliverables for this project. The first is milestone submission worth 30% of the total project grade, and the second is the final submission worth 70% of the total project grade. As a whole the first project is worth 20% of the course grade. The milestone and final submission are due at the time listed on their respective MyCourses assignment dropboxes.

Project Milestone

The milestone submission is worth 30% of the total grade (6% of your final course grade). It is also critical to the success of your project, as it will be used by your professor to provide feedback on your current progress and where you can improve.

The milestone submission should include the core API experience, meaning all endpoints should be implemented in your Node server. You are not required to have the client pages set up, although you are welcome to surpass the expectations. Your milestone submission does not need to be perfect. If it has errors, crashes at times, etc. that is all okay. It is simply meant as a check-in to ensure that you have been utilizing the time to work on the project and that you are on the right path.

Milestone Submission

You must submit the following things for the prototype submission.

- A live Heroku link to your application.
- A git repo containing the code for your project. The most recent commit should ideally be passing CI on GitHub Actions.
- A zip file of your code (without the node_modules folder) submitted to the MyCourses dropbox.
- Written documentation submitted to the MyCourses dropbox along with the .zip file (it should not be inside of the .zip). See below for details.

Along with your code zip file, Heroku link, and GitHub link, you will also submit written documentation as a .txt, .docx, or .pdf file to the MyCourses dropbox **as a separate file**. In that documentation, you must answer the following questions in full sentences.

- What dataset are you using?
    - If not one of the provided ones, where did you get it?
- What work has been completed for this milestone?

- What work is left, and how do you plan to complete it?
- Do you have a plan for going above and beyond? If so, what is it?
- If you used any borrowed code or code fragments, where did you get them from? What do the code fragments do? Where are they in your code?

Milestone Grading
Because there are no exact expectations beyond that your endpoints are setup, grading for the milestones follows a simple rubric, as shown below. Be aware that there are no above and beyond points allocated for the milestone.

Progress	Grade
Shows sufficient progress considering when the project was assigned.	100%
Shows some progress, but not all endpoints are fleshed out.	70%
Little to no progress made on the project.	0%

Note: 10% will be deducted from any grade if *complete* written documentation and project links are not submitted correctly to MyCourses.

Final Submission

The final submission is worth 70% of the project 1 grade (14% of the course grade). As per the syllabus, each day a project late is a 10% deduction. Additionally, no late project can receive a grade higher than 85%. For the final submission, provide the following through the dropbox on MyCourses.

You must submit the following things for the final submission.

- A live Heroku link to your application.
- A git repo containing the code for your project. The most recent commit should be passing CI on GitHub Actions.
- A zip file of your code (without the node_modules folder) submitted to the MyCourses dropbox.
- Written documentation submitted to the MyCourses dropbox along with the .zip file (it should not be inside of the .zip). See below for details.

Along with your code zip file, Heroku link, and GitHub link, you will also submit written documentation as a .txt, .docx, or .pdf file to the MyCourses dropbox **as a separate file**. In that documentation, you must answer the following questions in full sentences.

- What dataset are you using?
  - If not one of the provided ones, where did you get it?
- What went right in the development of this project?
- What went wrong in the development of this project?
- What did you learn while developing this project?
- If you went above and beyond, how did you do so?
- If you used any borrowed code or code fragments, where did you get them from? What do the code fragments do? Where are they in your code?

<u>Common Mistakes</u>

- There must be a client facing test page **and** a client facing API documentation page. These are separate pages that must both link to one another and be visible on your Heroku application.
- The API documentation page is **separate** from the submitted documentation questions listed above.

Final Submission Grading Rubric / Checklist

The grading process for projects is deductive. Each project starts with a base grade of 90%. Then, based on the table below, points are either reduced or increased.

Description	Value (%)
Above and beyond work.	+1 to +10
App is not designed and implemented to professional standards.	-5 to -20
App does not have stable performance / is throttled due to hardcoding, etc.	-5 to -20
Data from the .json file is not loaded at startup, and/or is accessed from the file every time an API request is made.	-10 to -20
Static files (HTML, CSS, Client JS etc) required for the app to function are not delivered by the server.	-5 to -10
Server does not support the following status codes: 200, 201, 204, 400, 404, others (if necessary).	-3 per code
Server API does not support the following methods for appropriate endpoints: GET, HEAD, POST	-5 per method
API does not have at least 4 GET request endpoints. This number does not include endpoints serving static files.	-5 per missing endpoint
API GET endpoints do not also support HEAD requests.	-5 per endpoint
API does not have at least 2 POST request endpoints.	-5 per missing endpoint
No GET requests support query parameters.	-5
Direct calls to GET requests do not work from either the app, the browser, or both.	-5 to -10
Users cannot add data to the API through POST requests, or the server does not persist that data until restart. Note that data persistence beyond restart is not expected.	-10
Server responses do not set Content-Type and Content-Length headers.	-3 per offense

Server does not return a 404 response for invalid URLs.	-5
POST requests do not accept both JSON and x-www-form-urlencoded formats for the body (based on accept header)	-5
Data endpoint responses do not default to JSON.	-5
Client does not use the fetch API for data requests to the server.	-10
Client does not send the accept header with data requests.	-5
API Documentation is missing information as defined in the project requirements section above.	-5 per endpoint
Project does not use Git for version control or the professor cannot access the code repository on GitHub.	-20
Project does not use ESLint with the Airbnb spec defined in .eslintrc file.	-10
Airbnb spec has been modified, or ESLint errors have been manually silenced.	-5 per instance
Code is not free of ESLint errors.	-5 per error
Code is not free of runtime errors.	-5 to -50 based on severity
Project does not use an appropriate .gitignore file.	-10
Project does not use GitHub Actions with the proper configuration.	-10
Project does not use Heroku or other cloud platform.	-20
Borrowed code or code fragments are not commented in the code, and/or the documentation.	-100
Code is not well organized and commented.	-5 to -20
Missing proper documentation submitted separately from the .zip file.	-5

A Note on Git Usage

While it will not count directly against you, there should be a substantial number of commits in your project repository by the time of the final submission. Proper utilization of version control is essential in the industry.

If you only push your entire project to your repo after it is complete, that is a failing on your part to understand why git exists. Git is only useful if you regularly commit your code. Without doing so, rollback points are not created and you cannot undo mistakes you have made without manual code editing.