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

Table of Contents

Overview

Individually, you will be creating a rich web application that uses a backend web API **of your own design**. You may use external APIs **in addition to your own**, although you are not required to. The app must be engaging and aesthetically pleasing in addition to being functional. The expectation is that you will be producing portfolio-quality work. Your project should be scoped for a few weeks, meaning it will have a fairly slim feature set that is well implemented.

Application Description

The actual application is fairly open-ended. Your goal is to create a rich web app that uses backend (server) data served up by the Node API that you will also create. The Node server should send this data as JSON to the client. It should also serve up static files like HTML, CSS, Client JS, images, etc.

Project Ideas

Some projects that have been made in the past include task trackers, workout planners, pokemon team builders, non-realtime games, chat applications, etc.

When determining the concept for your project, there are a few questions to ask yourself.

1) How long will it take me to implement the server for this application? If you are trying to work with a large dataset, you will need to write a lot of backend code to handle all that data.

   For example, students in the past have tried to implement a TTRPG Character Sheet for systems like D&D 5e, only to realize that the 5e character sheet has a LOT of data. These projects have almost always turned out underwhelming because the students cannot complete them in time.

2) How long will it take me to polish the front end? Remember that not only does your app need to work, but it should also feature a well-implemented user experience. A non-technical recruiter should open your app and be able to tell what it does. It should have a well-thought-out and implemented CSS stylesheet that presents the page well. It should have clear user interactions. All of this development will take time.

3) Is this a project I am passionate about making? Oftentimes you will not be given the opportunity to work on something you want to make. If you don't have an idea you are passionate about, you can certainly fall back on something, but having a project you *want* to make will make the whole process more enjoyable.

Below are a few professional applications that could be accomplished (with a limited feature set) for this project.

Trello - https://trello.com/
Trello is a card-based task tracking system for teams. An implementation of something similar to Trello for this project might be a basic task tracker with cards that can be moved between a "Backlog", "Doing", and "Completed" column. Keep in mind we will not be covering user accounts and login systems before this project is due.

Idiots Win - https://idiots.win/

A simple game powered by an API. Players try and guess the most searched google result based on the starting of a search. An implementation of something similar for this project might include a series of questions and answers on the backend. The client could then request a new question which would be displayed to the user. Then the users guess is sent to the server for validation.

Google Forms - https://www.google.com/form
Creating a basic form application in which users can create their own forms is actually fairly straightforward for this project. Your client could submit some questions to the server, which then stores them and gives the client a unique link back. If someone goes to that link, the server retrieves those questions. Then the client fills out their answers, which the server also stores.

Project Requirements

The following are all the criteria your project must meet.

- Provides a rich and engaging user experience.
- The application is designed and implemented to professional standards.
- Application has stable performance and is not throttled due to hardcoding, etc.
- Any static files (HTML, CSS, Client JS, images, videos, etc) are sent by the server. Note that you may use an external API to retrieve resources like images or videos to compliment your own.
- Information calls (getting/posting data, etc) should be done through either the fetch() or ajax APIs.
- Direct calls to GET requests should work correctly from both within the app and through direct calls to the endpoint in the browser.
- Server API must use and support the following status codes:
    - 200
    - 201
    - 204
    - 400
    - 404
    - 500 (if necessary)
- Server API must support the following methods:
    - GET
    - HEAD (Client does not need to use HEAD, but the server should be able to accept HEAD requests to all GET endpoints that send back JSON data).
    - POST
- At least one GET request must support query parameters. A reminder that you should not add data to your API using GET requests (that is the job of POST). Instead, your query parameters should be used for things like filtering the results of a query, etc.
- The client must send the accept header. Note that your server only needs to support JSON, although you may add support for things like XML.
- The client must submit a body in a POST request that adds or updates data in the API. Users must be able to add data to the API that is then persisted by the API for retrieval in the future. For example, a user could add a task to your task tracker. Note that after 30 minutes of inactivity, Heroku will restart your app and your data will be lost. That is okay. You are not expected to persist the data beyond that restart.

Functional Requirements

- Uses Git for version control in a repo that the professor can access.
- Uses ESLint with the Airbnb spec.
- Uses CircleCI for build testing.
- Uses a cloud service (such as Heroku) for deployment.
- Borrowed code and code fragments must be credited in code comments and in the 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, utilizing a CSS framework to increase the professional look and feel of your frontend application, integrating an external API that interacts with your own API to create a more full-featured user experience.

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.

Deliverables

There are two deliverables for this project. The first is an ungraded prototype, and the second is the final graded submission. They are due on the date listed on their respective MyCourses dropboxes.

Prototype Submission

As mentioned above, the prototype submission is not graded. That being said, it is critical to your success in the project. The prototype will be used by the professor to give you feedback on your current progress and to give suggestions regarding what you should do to improve the application.

Your prototype should include the core experience of your application, with a focus on the backend API. The full API might not be ready, and the app might look like an unstyled HTML page, but the main part of the experience should be complete and testable.

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.
- A CircleCI link showing a passing build of your most recent commit.
- A zip file of your code (without the node_modules folder) submitted to the MyCourses dropbox.
- Documentation submitted to the MyCourses dropbox along with the .zip file (it should not be inside of the .zip). See below for details.

Documentation

You must submit a documentation document with your code and link submissions. This should be in a .doc, .docx, .pdf, or .txt. In the documentation, answer the following questions and provide endpoint documentation.

Questions

Answer the following questions in full sentences. Please break out your answers into individual paragraphs so that they are easier to parse.

- What is the intended purpose of your application?
- What data will be stored and delivered by the API?

- What work has been completed for this prototype?
- 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?

Endpoint Documentation

You must document each data endpoint (those that do not serve static files) with the following information:

- The URL of the endpoint
- The supported methods of the endpoint (GET, HEAD, POST, etc)
- Any query/body parameters that the endpoint accepts/requires.
- A brief description of what the endpoint does and returns.

Example:

URL: /getDogByName
Supported Methods: GET, HEAD
Query Params: dogName (the name of the dog being searched for)
Description: Retrieves information about a specific dog.
Return Type(s): JSON

Final Submission

The final submission is graded, and worth 20% of the final grade for the course. 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.
- A CircleCI link showing a passing build of your most recent commit.
- A zip file of your code (without the node_modules folder) submitted to the MyCourses dropbox.
- Documentation submitted to the MyCourses dropbox along with the .zip file (it should not be inside of the .zip). See below for details.

Questions

Answer the following questions in full sentences. Please break out your answers into individual paragraphs so that they are easier to parse.

- What is the purpose of your application?
- What data is being stored and delivered by the API?
- 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 were to continue, what would you do to improve your application?
- 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?

Endpoint Documentation

You must document each data endpoint (those that do not serve static files) with the following information:

- The URL of the endpoint
- The supported methods of the endpoint (GET, HEAD, POST, etc)
- Any query/body parameters that the endpoint accepts/requires.
- A brief description of what the endpoint does and returns.

Example:

URL: /getDogByName
Supported Methods: GET, HEAD
Query Params: dogName (the name of the dog being searched for)
Description: Retrieves information about a specific dog.
Return Type(s): JSON

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 does not provide a rich and engaging user experience.	-5 to -20
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
Static files (HTML, CSS, Client JS, images, videos, etc) required for the app to function are not served by the server. Note that you may use an external API to deliver media such as images and video in addition to static files delivered by your own server.	-5 to -10
Client does not use either the fetch or ajax API for making data requests to the server.	-10
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 a POST request, or that data is not persisted by the server. Note that your heroku app will restart after 30 minutes of inactivity, and you will lose your data. That is allowable, as long as the data exists until the server restarts.	-10
Server does not support the following status codes: 200, 201, 204, 400, 404, 500 (if necessary).	-3 per code
Server API does not support the following methods: GET, HEAD, POST	-5 per method
No GET requests support query parameters.	-5
Client does not send the accept header with data requests.	-5

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.	-20
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 CircleCI.	-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.