Final Project

DUE DATE: December 3 at 11:59:59

The final project for this class is a culmination of everything you've done in this class. It gives you free reign as someone who *engineers* web applications. So we won't be holding your hand, but your TAs and I will provide you support wherever you need (as long as it's timely). If you're past Week 9 and haven't started, don't expect too much help because you're about a week away from submission and we all need some amount of vacation.

Get started early so we can provide you support without having to stress ourselves. We are also people, so the earlier you get in the more we can help you.

A little pitch for why you should try on this project. If you want employers to notice you for internships or maybe your first full-time job, having a solid project is really attractive. Most of the interns I've hired have been able to create applications like these. Additionally, if you build something you're passionate about, it's a great talking point when being interviewed.

The Prompt

The prompt for your final project is very open-ended. Build a web application which has the following requirements:

1. Has a server which runs both a backend (logic on a server) and serves a frontend (UI to interact with)
2. The frontend has to have interactions which cause some changes to the data stored on the server.
3. The frontend must retrieve some amount of data from the backend and display it on the web page dynamically.
4. The frontend must have MULTIPLE pages that are linked together.
5. The frontend must handle network errors in data transfer.
6. The backend must expose a RESTful API so that we can interact with your server from the command line.
7. The backend must store data in a manner such that if the program exits, the data is recoverable.

8. The backend must similarly store statistics on how people use the server and what responses are given

Groups & Grading

This is a group project with the maximum group size being 4. If you want to have groups of less than that size, that's cool as well. For groups of 3 or 4, I've labeled additional smaller things in each of the example project. They all entail going slightly above and beyond, using some sort of other API or concept. If you're in a group of 3, you can also make your page mobile responsive using media-queries and that will also fulfill. Your grade for this project is divided into two sections:

1. Project Score - 200 points (everyone gets the same score)
2. Individual Score - 50 points (everyone scored individually)

You can also use this project to pad your class grade. If you add extra functionality to your application (**as long as we've discussed it before hand**) I'll give you up to 50 extra points in the class. Speak to me individually if you plan on pursuing this.

Project Score

The project score is based on how closely you fulfill each of the 8 requirements:

1. [20 points] Has a server which runs both a backend (logic on a server) and serves a frontend (UI to interact with)
2. [10 points] The frontend has to have interactions which cause some changes to the data stored on the server.
3. [20 points] The frontend must retrieve some amount of data from the backend and display it on the web page dynamically.
4. [10 points] The frontend must have MULTIPLE pages that are linked together.
5. [10 points] The frontend must handle network errors in data transfer.
6. [30 points] The backend must expose a RESTful API so that we can interact with your server from the command line.
    a. This includes having proper status codes and error handling
7. [20 points] The backend must store data in a manner such that if the program exits, the data is recoverable.

8. [10 points] The backend must similarly store statistics on how people use the server and what responses are given
9. [10 points] For groups of 3 and 4, you must complete an extra task on each project that scales with your group size. If you have your own project, we'll work it out.

Individual Score

The individual score is made up of a report and a survey.

The report is a reflection on the project that you completed as well as what challenges and things you don't understand. The questions for the report can be found in the appendices:

1. [15 points] Teammate survey (depending on team size)
2. [35 - 45 points] Answer questions in the appendices

Submission

Each group will submit their project once. Each team member will have a second individual submission as well.

The individual submission should be a single PDF file answering the questions listed in the appendix based on your group size. The group submission should be a single zip file with at least:

1. A server.js file that we can invoke as node server.js --port=[NUM]
    a. For example, if your project runs on port 8080, we should be able to reach the following routes
        i. localhost:8080/api - Your API
        ii. localhost:8080/stats - Statistics for your server
        iii. localhost:8080/index.html - the main page
2. A readme file for how to interact with your api
    a. Put example requests for EACH of the endpoints of your API.
3. A /static directory which has all your static assets

Appendix A: Example Project Ideas

If none of these speak to you, I have a range of possible applications to build. The most important part is that you should be fulfilling all the requirements for grading. If you're a senior and want to get a head start on your Senior Design, that's also a really good thing to do, **as long as you fulfill all the requirements for grading.** Did I mention that you should fulfill all the requirements for grading?

For each of the applications, I'll give you some ideas on how to approach them as well as how they fulfill each part of the grading. Some of them have some pretty good ways to expand on them that I've also listed.

Most of these projects have some form of data input, some way to retrieve a bunch of information, and some way to retrieve information for a single user.

COVID-19 Contact Tracer

A COVID-19 Contact tracing application would be pretty dope right now for sure. It's pretty relevant to everything that SCU will be going through in Winter and Spring.

The Frontend

The UI for this project would have 3 pages:

1. A front page to show where you have been and what you've answered
2. A form to submit your schedule and whether you have symptoms, tested positive, or neither
3. A page to show people who you've been near
4. A static page that shows information on COVID-19

The API

1. GET endpoint for a single user to see their history
2. GET endpoint for a single user to see if they've been near someone
3. POST endpoint for inputting form data (JSON or form data)
4. DELETE endpoint to delete all data on a single user
5. GET request for usage statistics

The Backend

1. Some sort of data store (file, database, cache)
2. Log of data for usage statistics, periodically committed
3. How do you pull up which users were nearby a certain location?
4. How do you show people if they were near someone who was infected?

Possible Extensions

1. Use Open Street Map to plot nearby users.
2. [3] Use the HTML Geolocation API so people can "check in" easily to the web application
3. [4] Use SSE and HTTP/2 to push notifications to users if they might have been in the area of an infected individual.

Collaborative YouTube Playlist

Because Netflix shouldn't be the only thing that you can watch party. Let's make collaborative viewing a application for Youtube.

The UI

1. One page to create a new room (for only you and your friends)
2. One page to join a room
3. One page to play videos and add new videos into the queue
    a. Long poll on this page so people play at ~the same time

The API

1. GET endpoint for videos in a room
2. POST endpoint to create a room
3. POST endpoint to add videos to a queue

The Backend

1. Support normal CRUD operations.
2. How do you keep people in sync?

Possible Extensions

1. Add a queue of videos that is pausable for a group of users
2. [3] Add upvoting and downvoting (or a threshold to be played)
3. [4] Use Websockets!

Queue Manager for TAs

True to the adage that every Software Engineer's biggest competitor is an Excel Workbook that just works, it's time to replace your TAs spreadsheet for questions.

The Frontend

1. One page for TAs and other students to upload + helpful resources
2. One page to show and submit all the questions
3. One page for showing questions from each of the labs.

The API

1. A GET endpoint for all questions
2. A GET endpoint for questions from old labs
3. A POST endpoint for creating new questions
4. A POST endpoint for TAs and students to drop helpful links

The Backend

1. Normal CRUD operations and managing a database
2. How do you prioritize the most requested questions?
3. How do you make sure that no student is starved of TA help (they have a niche question)

Possible Extensions

1. [3] File Upload: Allow TAs to upload files directly into the website
2. [4] Authentication + Authorization: Make links require TA approval

Composer

Probably my favorite idea on here, composer is an application that plays midi files to create music collaboratively. This is probably one of the harder projects you can do (but I think really rewarding)

The UI

1. One page which is a sequencer that plays midi files
2. One page to find published songs sequences

The API

1. GET request to get a sequence by ID
2. GET request for a bunch of sequences (for the second page)
3. POST request to save a sequence
4. PUT request to make a sequence publicly available

The Backend

1. Normal CRUD operations
2. How do you allow people to share sequences with their friends?
3. How do you line up two sequences that don't match in length?

Possible Expansions

1. [3] Record your own voice over midi files: Look into the getUserMedia API
2. [4] Sequence battle: Make a freestyle rap sort of composition and allow people to vote for what's best.

Your Own Blogging Engine

One way to attract employers is just to write, write, and write more. So how about we build a blogging engine (similar to Wordpress). The only requirement here is that you have to create your own templating engine (which seems daunting, but it's really just parsing strings).

The UI

1. A page to navigate through an archive of blog entries
2. A page to view a single blog entry
3. A way to add comments onto a blog post.

The API

1. A GET endpoint to list all entries with some filters)
2. A GET endpoint to render a single blog post + templated data
3. A POST endpoint for comments
4. A GET endpoint for comments

The Backend

1. Normal CRUD Operations
2. How do you translate a user's input to match the template
3. How do you store the information for each blog post + comments

Possible Extensions

1. [3] Tagging System: Add tags for each blog post
2. [4] Add WYSIWYG (wi-see-wig) editing, what you see is what you get

Appendix B: Individual Report Questions

1. What were you directly responsible for in this project?
2. What possible ethical issues do you face as a developer of this application and how might you deal with them?
3. What challenges did you face in developing the API, the UI, and the backend?
4. [Only for Groups < 4 people] How does your application handle network errors? What sorts of error scenarios did you handle?
5. [Only for Groups < 3 people] What are the next steps in your mind for a project like this? What would you improve?