CPSC 2600 Individual Final Project

The purpose of this project is for you to have a chance practice the techniques we have learned in this class. Your job is to create a fully-functional single-page web application with database, server-side and client-side components.

The project can be about *almost* whatever you want - choose a topic that interests you! There are a few restrictions:

- You must create a single-page application. (Not a page-based one.)
- You may *not* use any of the following from this list of common projects:
    - Hotel booking application
    - Note or ticket editor/manager
    - To-do list
    - Twitter clone
    - Book or library website
    - "E-commerce" site

You *really really* want to do one of the above, please talk to me first. The project is partly graded on originality, so we'll have to come up with a way to make it your own.

Generally, I strongly recommend that you run your idea by me before starting. I can help you manage the scope of the work and make sure the idea will work well given the technologies we've covered in the class.

Here are the technical requirements:

- Must use a **MongoDB database** with **Mongoose**.
    - Must use at least **two schemas/models** with some kind of relationship between two types of documents (such as **embedded subdocuments** or **references**.)
- Use **Node.js and Express** on the backend.
    - Your server-side application must be organized with **routing** and **controller** logic separated. Use **middleware** where appropriate - don't write all logic in one file!
    - Create an **API** that will be used by your client-side application:
        - Don't worry about authorization or authentication;
        - Data submitted through the API must be validated and sanitized at the controller level, before Mongoose's validation/sanitization. This means that your application must have two levels of validation/sanitization on the back end!
        - Your API must have GET and POST routes;
        - Use REST conventions when designing the API. Don't worry about more advanced REST features like HATEOAS, caching, asynchronous responses, etc. but do implement the following:
            - responses in JSON format
            - URL versioning

        - resource-based endpoint names
        - sensible response codes
        - filter parameters for large collections
- You must have a client-side, single-page application created with React:
    - Your React application must use **state**
    - Must have at least one form, using the **controlled component** pattern
    - The application must be updated asynchronously.
- You may ***not*** use a project (or any code from a project) that you created in another course. All work must be original and unique to this course. All the work must be created by you alone; this is ***not*** a group project.
- You may ***not*** use a project you created from an online tutorial. (Although it's fine to consult online tutorials for reference.)
- Use **environment variables** for credentials. Use **Webpack** to bundle your front-end code. Be sure to style your front end using CSS.
- When you've almost completed your project, **meet with me** (during our regular lab time or office hours) to review it together. Note that our last lab session of the semester will probably be extremely busy and I may not have time to meet with everyone, so take this into account when managing your time.
- Your project must be deployed to the web using some online hosting service. I will provide instructions for a particular service later (TBD), but you may use another service if you prefer.

Some other notes:

- The challenge level and originality of your project will be factors in your grade. I'd be happy to review a project proposal if that would be helpful.
- The user interface should be relatively attractive and easy to use:
    - The result of user actions should clearly be shown (maybe with a success message).
    - Error messages should be rendered in the client and visible to the user, including validation errors; use conditional rendering in your React components to do this and figure out how to access the error message from the server in your client-side application.
    - Raw JSON responses should NOT be shown to the user. Find a way to render the data in your application.
- Your application should be well designed and engineered from a software design perspective. Ensure that your code is appropriately modularized so that each part does only the thing it's responsible for, and apply abstraction appropriately. (For example, the client-side code should not require any special knowledge of how the database works.) Format your code (you can use an auto-formatter) and provide comments where necessary. Delete code that has been commented out. Use helper functions where necessary to avoid code duplication.

- Test your application thoroughly! Get your friends to use it. Test it after you've uploaded to the web. Make sure it's impossible to break the application, since I will intentionally be trying to break it! Use Postman to test your API endpoints.
- Include references to resources you've consulted on the web. Include the following:
    - A hyperlink to the source;
    - The file name and line numbers of the code you used.

      If you intend to heavily use a particular resource, please consult me first.

Hand In

Please follow this procedure:

- Place a clickable hyperlink to your project in the comment of your Brightspace submission. Also include instructions for building and running your project.
- Rename your project folder to **project-firstname-lastname** and delete the node_modules folder. Make sure all necessary modules are saved in package.json.
- Include in package.json scripts that are necessary to build and start the application.
- Include a **readme** file in the root of the project folder (in md, html, or txt format) with basic documentation, including the following:
    - A clickable hyperlink to your project;
    - Instructions for building and running the project;
    - A brief, one-or-two sentence description of your application;
    - A list of features you're proud of that you would like me to consider when grading;
    - Instructions on how to use your application if it's not obvious;
    - References, including links to the sources and file names and line numbers within your project;
    - API documentation, including the following:
        - Endpoints and methods with a brief description of each;
        - Response format;
        - Expected POST body format;
        - Examples on how to use each endpoint;
- Zip your folder and submit it to Brightspace.
- Feel free to invite me to view your Github or other repository.

Grading:

- **<u>Technical requirements:</u>**
    - [3 marks] API using REST conventions; filter parameters implemented
    - [2 marks] Database with relationship between entities and schemas
    - [1 marks] Data validation implemented
    - [2 marks] Back-end application is well designed
    - [3 marks] single-page client-side React application with state and at least one form
    - [2 marks] Application is easy to use
    - [2 marks] Review meeting with me has been completed when project is nearing completion

      Total: 15

- **<u>Marks will be deducted for the following:</u>**
    - [-5 marks] Significantly similar to one of our assignments OR low challenge level, similar to online tutorial, etc.
    - [-3 marks] Significant bugs, crashes, errors, etc.
    - [-3 marks] Missing or incomplete documentation
    - [-2 marks] Not uploaded to web
    - [-100%] External sources used with no references.

As the above criteria are somewhat subjective, please feel free to ask me to provide feedback on your project before submitting.