COMPSCI 326

Milestone 2: Front-end JavaScript | Due: **4/16/22**

Overview

This project will give your team an opportunity to present an innovative web application that your team will be developing over the latter half of this semester. In the last milestone, we asked you to visualize your user interface using wireframes and a combination of HTML and CSS. **In this milestone, you will be implementing most of the changes, updates and animations the user would expect for the client UI.** As before, we expect your thoughts and ideas for your project to change during the course of the semester leading to changes in the data your application will support and the interface your users will expect. This and all future projects will be building blocks leading to the final project submission at the end of the semester. Please read the Final Project Specification to understand the requirements for the final submission.

Application Structure

In order to assist you with planning your application, we are going to walk through the structure and format of an example application below. Imagine an application whose goal is to manage an inventory of books which can be borrowed by users for a library checkout system. Let's first define the data-flow as we did in the previous milestone.

- Book Objects - each has 4 fields: ISBN, Title, Author, and Quantity
- User Objects - each has 4 fields: ID, Name, Email, Password, List of Checkouts
- Checkouts - Is a relationship between 1 Book & 1 User and belongs to User. Each checkout has the following fields: checkoutID, userID, bookISBN, & date created

For our API, we will need CRUD operations for each of these 3 objects. Some example endpoints might look like:

- **`/book/new`** which allows for a book to be added to the library when a request is sent to this endpoint containing the title, author and quantity of the book
- **`/book/ISBN`** a view endpoint which returns all 4 fields for a book as a JSON object
- **`/user/new`** or **`/register`** which allows for a new user to sign up
- **`/login`** which allows for a user to login
- **`/user/id/checkout/new?isbn=1234`** which creates a checkout object for the given user provided a book ISBN, and also decrements the quantity. This should only work if the quantity is > 0 for the given book.
- **`/user/id/checkout/view`** which returns all of the checkouts for a given user
- **`/user/id/checkout/delete?id=12`** which deletes a given checkout (i.e. when the user returns the book) and updates the quantity

Your file structure to implement this might look something like this:

1. **`server.js`** - a server that handles parsing the incoming requests with their URLs and parameters and sends that information to the correct functions. There should

be a handler as well as functions for all of the CRUD operations that perform the requisite checks as well as call the required database functions
2. **`database.js`** - a file that *only* implements database logic, should have/wrap functions such as `insert()`, `find()` and `findAndUpdate()` and handle any errors that may occur
3. **`bookview.html`** - a file that renders HTML to view any 1 book
4. **`bookview.js`** - a client-side JS file which pings the endpoint via `fetch` for **`book/isbn/cover-page-url`** and then uses the returned JSON to inject HTML into the **`bookview.html`** file so that the book can be viewed

Part 0: Project API Planning

Before you write any code, you will need to think of how the client should obtain information from the back-end server. Whether you use RESTful APIs or using other types of APIs, you will need to have different endpoints to provide different functionalities for the users. You will need to write out all APIs the server will provide with a brief description for each. You can also include examples of input and output.
Alternatively, you can draw out a flow-chart with the information.
Put all of the details in **docs/milestone2.md**

Examples of good API Documentations:
- https://developer.github.com/v3/guides/getting-started/
- https://developer.spotify.com/documentation/web-api/

Examples of some API flowcharts
- https://developer.spotify.com/documentation/general/guides/authorization-guide/#authorization-code-flow
- https://developer.accuweather.com/api-flow-diagram

Part 1: Back-end Skeleton Code

Your team will implement a *dummy server* that responds to all client requests you specified in Part 0. Note that for this milestone, you are not required nor expected to implement all logic for your back-end server such as processing requests or database operations.

What your server will need to do is to simply send back **fake data** (use faker-js to make this easier and more interesting!)back to the client in some format (we recommend JSON) that you can parse on the client side. You don't need to write anything for this part in **docs/milestone2.md**. Once you are done, make sure you test your API (using Postman) and make sure it is returning the data you are expecting.

Here are a few instructions for writing the dummy server:

- You are required to use JavaScript to write the server side.
- The server must be deployed on Heroku; see **Part 3** for details.
- We strongly encourage you to use Express.js which we will be talking about soon in lectures.

- You must serve your front-end files (e.g. HTML, client side js/ts) from your server. We recommend using Basic Routing in Express to serve the files.
- You are required to return at least some data to the client whenever you are processing a request.

Part 2: Front-end Implementation

Your team created a polished web interface with HTML and CSS code in the last milestone. Now, you will need to glue them together. In this part, your team will need to implement all functions using JavaScript which the client side uses to perform CRUD operations on the server side, and render it on the HTML page with the fetched data. Once you are done, please take **four** screenshots with a brief description for your user interface to illustrate each of the Create, Read, Update, and Delete operations and include them in your **docs/milestone2.md**.

Part 3: Deployment

Include the link to the hosted application in your **docs/milestone2.md**.

For the most part. it should be as simple as linking your existing GitHub repository to the Heroku Auto-Deploy feature as described here: GitHub Integration (Heroku GitHub Deploys).

If you need additional instructions or help, you can refer to the Documentation from Heroku for more details.

Submission

You are required to write a **docs/setup.md** with all the steps required to build your project. Remember to include all the source files including all JavaScript in your repo.

You will be creating a Markdown file called **milestone2.md** in the **docs** folder of your github repository. It should contain:

1. ✓ A brief and precise representation of APIs for you application
2. At least one set of four screenshots of your client interface with descriptions
3. The URL of your Heroku Application

In addition, your **milestone2.md** file must contain a **breakdown of the division of labor** for each team member — that is, saying who did what. Remember that *everyone is expected to contribute roughly equally* to each phase of the project. We expect to see similar numbers and kinds of GitHub commits by each student.

Update your github repository by the due date and then create a “release” for Milestone 2 tagged `milestone-2`. There is no need to “submit” anything. A reminder: for instructions on how to create a release, see:

- https://help.github.com/en/github/administering-a-repository/managing-releases-in-a-repository#creating-a-release

As before, you should use the COMPSCI 326 Slack for discussions while you are working on your project.

NOTES

client-side static files:
You should have all of your client-side static files in one folder. This folder will contain HTML & CSS files for the UI as well as JavaScript for the calls to the API server and any animations or dynamic things you need.

You should serve this folder by using the line you mentioned from the example.
this.server.use('/', express.static('yourfoldername'));

What this line of code does is it tells express to serve the contents of yourfoldername at the base URL of your website (so www.yourwebsite.com/ as opposed to www.yourwebsite.com/something/)

By default your main page should be called index.html because a web-browser will load index.html from / as long as index.html is in the folder you specified.

MongoDB and authentication: *(this is if you have decided to work ahead beyond this milestone)*
If you create a MongoDB account for your final project. YOU SHOULD NOT COMMIT YOUR CONNECTION URI TO GITHUB!

The connection URI contains a pre-authentication that allows anyone with access to that URI to connect to your database and read, write and even DELETE ANYTHING. It's a huge security flaw.

What you need to do instead is either store the URI in some kind of secrets.json file and then read it in (BUT DO NOT COMMIT THIS FILE TO GITHUB! ADD IT TO .gitignore!!!!) OR use Environment Variables as Heroku instructs.

If you are using Atlas then use an ignored secrets.json file for development and an environment variable for production (you can set env variables using the Heroku web GUI on the settings page, which looks like this:
https://raw.githubusercontent.com/wiki/fuseumass/dashboard/images/ses.png)

Note: If you have already committed a file containing the URI, we would highly recommend you just create a new DB instance with a new URI and delete the old one. However, if you cannot for some reason, reach out to the TAs via DM and they will help you remove it from your git history. It is not enough to simply make a new commit and delete it from your file as it will remain in the history of your repository.