COMP 4513
Assignment #2:
Mongo/Node + React

Due Sat Dec 11th at midnight (since final exams start on Dec 11, I recommend finishing earlier)
Version 1.0, November 8, any changes will be in yellow

Overview

This assignment is an expansion of your first assignment. This group assignment provides an opportunity to display your front-end, back-end, and dev-ops capabilities. You will be working in a group of 1-3.

Constructing Your Group

Your first step is likely to put together your group. Then choose someone's assignment #1 source code to be the starting point for your group assignment. I'm still a little ways away from marking your first assignments, so don't wait.

You will need to have *"a single source of truth"* when it comes to your source code. That is, there must be a "master" location that contains the definitive version of your source code. Employers want to see that you are able to use GitHub, so I recommend everyone in group try to be active using branches and making frequent pushes.

Please notify me by Fri Nov 12 of the composition of your group by email.

Submitting

You will not be submitting your source code. Instead, I will mark your code from a GitHub repository. I will mark the functionality by visiting some type of server. Thus, you will submit your assignment by emailing me a short message consisting of the group member names, the GitHub repository URL, and the web address where I will be able to find your working assignment.

Grading

The grade for this assignment will be broken down as follows:

Visual Design and Styling	25%
Programming Design and Documentation	5%
Functionality (follows requirements)	70%

Data Files

I will be providing you with a variety of JSON data files that you will import into MongoDB. The github repo for the start files are at:

https://github.com/mru-comp4513-archive/comp4513-f2021-assign2

Requirements

1. In this assignment, you will be creating a contemporary style of web application that is based on your first assignment. Instead of consuming my API, you will implement (and host) an expanded version of the API using Node and MongoDB. While your front-end will still be using React, you will improve its visual look by using a UI library such as Ant Design, React Bootstrap, Material-UI, or Argon.
2. You must host your group's project on one server or *two* live servers. Why two servers? You could host your React front-end on a static site server and your Node back-end on another server.

   Based on experience of previous years, I would recommend hosting your React app file and your Node files on just a single server. In such a case, you would put your React files in the public folder like in the last Node lab and then make sure your Node app serves up static files. This will make login system easier to implement.

   Your MongoDB may potentially also exist on another server: however, I would strongly recommend using a third-party Mongo service such as Mongo Atlas. In such a case, your Node service will simply connect to your MongoDB database that is running on Atlas's servers.

   Regardless of your approach, it will take time to get this working so please don't leave this to the last few days!

   Here are some suggestions for Node hosting:

   - Heroku. It has a free tier and is a popular developer-oriented hosting option that integrates nicely with github. It requires that at least one person register with heroku and install its CLI software on their computer. That person then uses a few command line instructions to pull software from github repo and install to the heroku servers. I'd say 90% of students in the past have used heroku and generally have been happy with its ease of use.
   - Digital Ocean. Similar to Heroku. You can also find free credits via Git Education program.
   - Amazon Web Services (AWS) or Google Cloud Platform (GCP). Will likely be way too expensive over time, but you can get free credits that will last for a few months. Not a bad choice for a group with someone who enjoys devops and wants experience with one of these platforms.
   - Make use of multiple Docker containers (one a LAMP container for react front end, one a Node container for APIs, one a Node container for chat, and possibly one container for running MongoDB), and then deploy containers on any host environment that supports Docker. This could be GCP, AWS, Heroku, Digital Ocean, etc. All the cool kids nowadays are using Docker so you may want to try using it, but can be difficult to set up.

3. **API for Shakespeare and Users.**

In the first assignment, you consumed a web service from my web site. In this assignment, your React front-end will retrieve data instead from an API that you create. I have supplied JSON files that you will use to populate three collections (list, plays, users) in a Mongo database.

Please create the following API routes. You can take one of two strategies in regards to security.

- All of the GET requests require an authenticated user. This will be the easiest approach if you are using a single server. In this case, the Node 3 lab already shows how to implement this functionality using the Passport package.
- All of the requests (except login) must supply a valid API key (passed as a query string). If no valid API key is provided return a JSON-formatted error message. This will require passing this information to the client after login. This would likely require something like JWT. You will only need to follow this strategy if implementing security within React and are using two different servers. A realistic approach but likely too complicated for the time remaining.

`/api/list`	Return list of information of all plays.
`/api/play/id`	Return single play specified by the play id. Return error message if id doesn't exist.
`/api/user/id`	Return single user specified by the user id. Return error message if id doesn't exist. This should return only the following fields: id, details, picture, membership, and email.
`/api/login`	This is a POST request that will be needed if you are implementing your login in react or if front-end and back-end are on separate servers. **If your login is being implemented using NODE+EJS, then you don't need to implement this API.**

If you have your Node API services eventually running on one domain, and your React front-end running on a different domain, you are going to need to enable cross-origin resource sharing (CORS) in your Node API. Using Express, it is easy to add the `Access-Control-Allow-Origin` header to your response.

To help me mark your APIs, provide links/buttons for the first three API urls in your About page.

I would recommend completing the APIs early on!

4. Login System.

The first thing the user must experience with your React front end is a log-in screen if the user is not already logged-in. This login form should contain email and password fields. If you wish to use this as a portfolio piece, you may want to provide sample credential information on the form so that potential employers can log-in.

Given that there is only a month remaining in the course, I would recommend implementing the login as a separate non-React page using Node+EJS as in Node Lab 3. You will then redirect to the React home page of the assignment if user is successful logging in. Do note that this isn't real security, just the illusion of it!

You are going to need some system for "remembering" whether the user is logged in. I recommend using the Passport package in Node to implement your authentication. By default, Passport uses sessions to maintain login status.

If your Node is on a different server from your React app, you will likely need use JWT instead of session state for determining if the user if logged in. I'm afraid I'm not likely to be able to provide you with examples in class … but who knows maybe I will be able to.

In a React application with a "real" authentication system, you would need to add some type of login check to your React Route handlers. But because we are just bolting on authentication via Node+EJS, and not integrating it into our React application, this won't be necessary.

5. Changes to your first assignment React front end.

    a. Redesign the UI using a React UI component library such as Ant Design, React Bootstrap, Material-UI, or Argon. The Ant Design library is amazingly rich, but feel free to use a different one.

    b. The header must contain a way to logout.

    c. You must use your own APIs.

    d. In the About dialog, be sure to indicate which component libraries you are using and which technologies you are using. You want someone who looks at this dialog (e.g., potential employer) to be impressed with your technology use, so be expansive here!

    e. Provide a way for the logged-in user to view their profile information (first name, last name, city, country, picture, date joined) by adding an icon / button to header.