COMP 4513
Assignment #2:
Mongo/Node + React/Static-Generator

Due Tuesday April 7th at noon
Version 1.1, March 13, Changes 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 2-5.

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.

Please notify me by Wed March 11 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	15%
Programming Design	15%
Functionality (follows requirements)	70%

Data Files

I will be providing you with two 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-w2020-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.

   You can instead host 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, you may decide to make use of 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.
   - Digital Ocean. Similar to Heroku. You can also find free credits via Git Education program.
   - Google Cloud Platform (GCP). Will be likely too expensive over time, but you can get free credits (I think I have already done that for you) that will last for a few months. To use GCP, at least one person will have to install the gcloud CLI tools on their computer.
   - Amazon Web Services (AWS). Similar to Google's offerings. Will be likely too expensive over time, but you can get free credits that will last for a few months.
   - 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.

   Here are some suggestions for Front-End hosting:

   - GitHub Pages, FireBase Hosting, Netlify, Render, and Surge

3. **API for Movies.**

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 two JSON files that you will use to populate two collections in a Mongo database. You will have to create

If you are deciding to take an AWS- or GCP-oriented approach with your assignment, you might instead decide to use their noSQL options (DynamoDB for AWS, Datastore for GCP) instead of Mongo. That's fine with me, but you'll need to use a replacement for Mongoose (Dynamoose for DynamoDB, gstore-node for GCP).

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

- 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.
- 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.

`/api/movies`	Return all the movies in the data set.
`/api/movies/`*`id`*	Return single movie specified by the movie id.
`/api/brief`	Return a brief version of all the movies in the data set.
`/api/find/title/`*`substring`*	Return all the movies whose title contains substring.
`/api/find/year/`*`y1`*`/`*`y2`*	Return all the movies whose year is between y1 and y2.
`/api/find/rating/`*`r1`*`/`*`r2`*	Return all the movies whose average rating is between r1 and r2.
`/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.
`/api/favorites`	This will differ depending on the HTTP method. • For GET, return a list of favorited movies for the current user • For POST, add a new movie to the current user's list of favorites • For DELETE, remove a movie from the current user's list of favorites

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 five 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.

To implement the login in React, instead of relying on <form action=???>, you would instead use fetch to make an asynchronous post request to your node login route.

You can also implement your login as a separate non-react page using Node+pug/ejs as in Node Lab 3. You would 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, but given the craziness of this March, doing things easy is okay!

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.

Your React application needs to add some type of login check to your React Route handlers. Luckily, this functionality is already built-in to React Routes via the onEnter event:

e.g. <Route path="…" component="…" onEnter="{authCheck}" />

The main trickiness is that this check has to be added to each route and that your authCheck has to have access to the current login status via the application state tree.

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. For the filters, instead of doing the filtering logic yourself, you will use your new find APIs. This means the three filters (title, year, rating) must now be mutually exclusive. There will be no need to use local storage anymore.

    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 to be impressed with your technology use!

    e. In this assignment, the favorites will need to be persisted to your database. This means when a user logs in, your front-end will have to retrieve that user's saved favorites. When a movie is removed from the favorites, you will have to update the database.

    f. Provide a way for the logged-in user to view their profile information (first name, last name, city, country, picture, date joined).

6. To improve the performance of your application, and for extra credit, you will make use of a static site generator such as Next.js, or Hugo, and host that on a static site server provider or on your Node site within the static folder. Static site generators are very hot right now, and we will talk about them in class. By default, create-react-app does client-side rendering; server-side rendering tries to dramatically improve the performance of React on the client by having the JavaScript render as much of the DOM as possible on the server. Next.js is probably the most popular of these systems and it assumes you will be using Node to serve the front-end. It does require you to create your application with their own skeleton program (and not create-react-app). This is a totally optional requirement for extra marks.