COMP 4513
Assignment #2: Mongo/Node/React
(version 1.0)

Due Saturday April 6th at midnight
Version 1.0, March 6

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 or 2: however, you will be collaborating with two or three other groups to create a "super group" project. That is, your mark will be based on your small group's (1-2 of you) work. However, this assignment will require collaborating with a few other groups on a single large project.

Constructing Your Super Group

Your first step is likely to put together your super-group. That is, find 2-3 other groups who want to work together. Then choose one of the small group's assignment #1 source code to be the starting point for your super-group assignment.

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. Each super-group will have a single github master repo within which each small group will work. Each small group will likely create their own branches from this master. Eventually, these branches will have to be merged with the master.

Please notify me by Wed March 13 of the composition of your super-group. This is my first time trying this approach … I may need to make changes to it if it doesn't work out well.

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 live server (not cloud9). 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 several JSON data files that you will import into MongoDB.

Requirements

1. You must host your super-group's project on two/three live servers. Why two/three servers? Your react front-end will be hosted on one server and your node back-end will be hosted on another. Your MongoDB will potentially also exist on another server: however, you may decide to make use of a third-party Mongo service such as mLab (being discontinued in Dec 2019) or Mongo Atlas. In such a case, you simply connect to your MongoDB database to the service which is running on mLab's or Atlas's servers.

   Instead of creating 2-3 live servers, you could create a single virtual server that hosts your react front end with one domain, your node back end as another domain, with MongoDB service running in its own process.

   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:

   - 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. Lots of online instructions available (try searching for "deploy node heroku").
   - 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.

     There are several possibilities for hosting within GCP. One, you make use of GCP's App Engine. This is a PaaS like Heroku, in that it provides a prebuilt server with software already installed. You will likely have to spin up multiple App Engine instances.

     Another is that you create a single virtual server using the Compute Engine. You then will have to use linux commands to install and configure apache, git, node, and mongo. With this approach, you have total control over your machine!
   - 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.

2. In the first assignment, you made use of my images that were in a GCP storage bucket. Now, you will have to create your own GCP storage bucket, and upload/retrieve images from that bucket, and not mine. The starting github repo for the assignment has the images that you will need.

3. 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. It should provide two buttons for the user to login: using the login information in the users JSON file, or using oAuth Facebook and/or Google 9though I might change this oAuth requirement).

   You are going to need some system for "remembering" whether the user is logged in. In a PHP application, you might use PHP session state or cookies to maintain some type of session identifier between requests. Since your back-end API requires authentication, you will need to implement back-end authentication in Node. The authentication status will have to be communicated as well to the client.

   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.

   Login system on the back-end will use Passport and JWT.

4. API for Images. 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 several JSON files that you will use to populate several collections in a Mongoose database.

   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).

   To help me mark your APIs, provide some sample API urls for the first two in your About page.

   Please create the following API services:

`/api/images`	Provides JSON for all the images in the data set. This request must supply a valid API key (these can be found in the users.json dataset), otherwise it returns a JSON-formatted error message.
`/api/image/`*`id`*	Depending on whether it is a GET, POST, or PUT request this will: • GET. Provides JSON for the specified image id. • POST. Add a new image to the database. • PUT. Modify the image in the database. A GET request must supply a valid API key. For POST and PUT, the user must be logged in (that is, there must be a valid authentication token).
`/api/upload`	Uploads an image file (see below).

   Since you will likely 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.

5. Changes to existing React front end.

    a. The image data has been expanded to include user data and some EXIF information about the photo. Your View Single Photo must be updated to display this new data.

    b. Edit Photo Form needs to be updated as well to edit the EXIF data (but not the user data). In the first assignment, you simply modified the JSON data in memory. In this assignment, edits will be permanent in that the underlying MongoDB data source will be altered. To do this, you will use the `/api/image/id` route using a PUT request.

    c. In this assignment, users can only edit their own images.

    d. In this assignment, the logged in user can’t favorite their own images!

6. Provide a mechanism for a logged-in user to add a new image by adding an Upload option to the header. This is essentially the Edit Photo Form component from assignment one with no View / Map links but with an additional File Upload component.

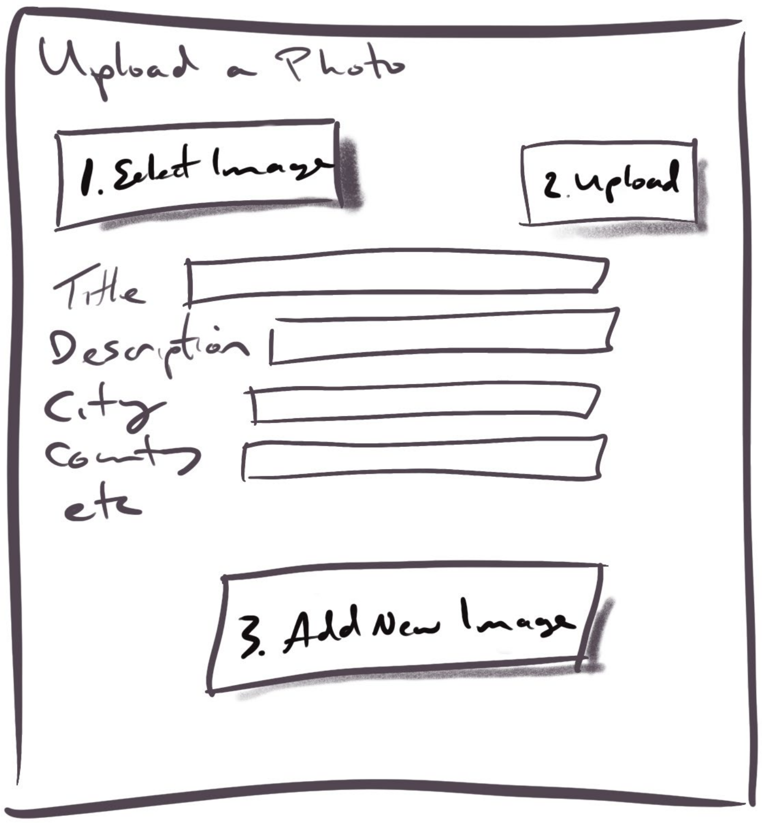
   The sketch below illustrates what is needed. It needs the following:

   a) upload component (buttons 1+2)

      The upload file component will consist of two buttons (1 Select Image and 2 Upload) which are used for selecting the image and uploading the image to the server. There are lots of examples online. If you make use of examples/code found online, be sure to provide credit+link in About page. Uploading the image to the server will make use of the `/api/upload` route.

      On the server, you will have to implement an upload route in your API. This will take the uploaded file and upload it to a GCP storage bucket.

   b) a form+button to enter the image information. The Add New Image button (3) will use the `/api/image/id` route using a POST request which will add the information to the underlying database.



7. A notification system if another user favorites the current user's image. This will be implemented using WebSockets. To test this, let's imagine we have three different computers (or three different browsers), each one in logged into a different user account (say User A, user B, and user C). Now have user A favorite an image belonging to user B. This should push a message to user B: the React SPA for user B should display a notification telling the user the good news!

   I may decide to make this an optional requirement if it ends up being too complicated.

Advice

I'm hoping this is a realistic and achievable assignment. I haven't had an opportunity yet to complete this assignment myself.

I do think the requirements can be worked on in parallel by the different groups. That is, one group could implement the login and authentication stuff, another group could implement the API on Node, while another group could implement the file uploading. Alternately, you could make one group focus on the devops and hosting side, and then let other groups split up the development tasks.