WEB322 Assignment 3

Submission Deadline:

Friday, June 17th, 2022 @ 11:59 PM

Assessment Weight:

9% of your final course Grade

Objective:

Build upon the foundation established in Assignment 2 by providing new routes / views to support adding new posts and querying the data.

NOTE: If you are unable to start this assignment because Assignment 2 was incomplete - email your professor for a clean version of the Assignment 2 files to start from (effectively removing any custom CSS or text added to your solution).

Specification:

For this assignment, we will be enhancing the functionality of Assignment 2 to include new routes & logic to handle the code to add new posts. We will also add new routes & functionality to execute more focused queries for data (i.e.: fetch a post by id, all posts by a category, etc.)

Part 1: Adding / Updating Static .html & Directories

Step 1: Modifying about.html

- Open the about.html file from within the "views" folder
- Add the following entry to the **<ul class="nav navbar-nav navbar-right">** element:
    - Add Post

- Your "About" page should now have a menu bar that looks like the following:

Step 2: Adding a routes in server.js to support the new view

- Inside your server.js file add the following route (HINT: do not forget __dirname & path.join):
    - GET /posts/add
        - This route simply sends the file "/views/addPost.html"

Step 3: Adding new file: addPost.html

- Create a new file in your "views" directory called "addPost.html" and open it for editing
- Copy the contents of "about.html" and paste it in as a starting point.
- Ensure that the "Add Post" item in the ** ** element is the **only** with the class "active" (this will make sure the correct navigation element is "highlighted")
- Remove all html code **inside** the ** ** … ** **
- Inside the (now empty) ** ** … ** ** element, use the html from the sample solution (https://web322-a3-sample.herokuapp.com/posts/add) to reconstruct the "Add Post" form (HINT: You can right-click the page to "view source" - the html you want is within the ** … ** element)

Part 2: Adding Routes / Middleware to Support Adding Posts

Before we begin adding logic to our server, we must first register an account with an image hosting service. This is required since the file hosting on Heroku is "ephemeral", meaning that any files that we upload (such as featured images for our Blog posts), will not be permanently stored on the Heroku file system. Therefore, instead of relying on Heroku to store our images, we will instead use Cloudinary.

- Sign up for a free account here: https://cloudinary.com/users/register/free (Choose "Programmable Media for image and video API" as your "product")
- Validate your email address once Cloudinary sends you a "Welcome" email
- Log in to Cloudinary and navigate to the "**Dashboard**"
- Record your "**Cloud Name**", "**API Key**" and "**API Secret**" values (we will need them later).

Once you have successfully created your Cloudinary account and obtained the required information, we can proceed to update our code:

Step 1: Adding multer, cloudinary and streamifier

- Use npm to install the following modules:
  - "multer"
  - "cloudinary"
  - "streamifier"
- Inside your server.js file "require" the libraries:
  - const multer = require("multer");
  - const cloudinary = require('cloudinary').v2
  - const streamifier = require('streamifier')

- Set the cloudinary config to use your "**Cloud Name**", "**API Key**" and "**API Secret**" values, ie:

```
cloudinary.config({
  cloud_name: 'Cloud Name',
  api_key: 'API Key',
  api_secret: 'API Secret',
  secure: true
});
```

- Finally, create an "upload" variable without any disk storage, ie:
    - const upload = multer(); // no { storage: storage } since we are not using disk storage

Step 2: Adding the "Post" route

- Add the following route: POST **/posts/add**
    - This route uses the middleware: **upload.single("featureImage")**
    - Inside the route, add the following code (from: the Cloudinary Documentation)

```
let streamUpload = (req) => {
    return new Promise((resolve, reject) => {
        let stream = cloudinary.uploader.upload_stream(
            (error, result) => {
            if (result) {
                resolve(result);
            } else {
                reject(error);
            }
            }
        );


        streamifier.createReadStream(req.file.buffer).pipe(stream);
    });
};
```

```
async function upload(req) {
    let result = await streamUpload(req);
    console.log(result);
    return result;
}



upload(req).then((uploaded)=> {
    req.body.featureImage = uploaded.url;



    // TODO: Process the req.body and add it as a new Blog Post before redirecting to /posts



});
```

Step 3: Adding an "addPost" function within blog-service.js

- Create the function "addPost(postData)" within blog-service.js according to the following specification: (**HINT**: do not forget to add it to module.exports)
    - Like all functions within blog-service.js, this function must return a Promise
    - If **postData.published** is undefined, explicitly set it to **false**, otherwise set it to **true** (this gets around the issue of the checkbox not sending "false" if it is unchecked)
    - Explicitly set the **id** property of **postData** to be the **length of the "posts"** array **plus one (1)**. This will have the effect of setting the first new post's id to: 31, and so on.
    - **Push** the updated **PostData** object onto the **"posts"** array and **resolve** the promise with the updated **postData** value (ie: the newly added blog post).

- Once this is complete, make use of the new "addPost(postData)" function within POST **/posts/add** route to correctly add the new blog post before redirecting the user to the **/posts** route

Step 4: Verify your Solution

At this point, you should now be able to add new blog posts using the "/posts/add" route and see the full posts listing on the "/posts" route.

IMPORTANT NOTE:

At the moment, we are not persisting our newly created Blog Posts (they simply exist in memory), however any images that we add are being stored within Cloudinary. This means that once our server restarts, the new blog posts will be gone, but the featureImage link will still be valid. To cut down on your storage usage on Cloudinary, please remember to delete these images once you have completed your testing (see the "Media Library" tab in Cloudinary)

Part 3: Adding New Routes to query "Posts"

Step 1: Update the "/posts" route

- In addition to providing all the posts, this route must now also support the following optional filters (via the query string)

  NOTE: We ***do not*** have to support the possibility of having both "category" and "minDate" queries present at the same time in the URL.
  - /posts?category=***value***
    - return a JSON string consisting of all posts whose category property equals ***value*** - where ***value*** could be one of 1,2,3,4 or 5 (there are currently 5 categories in the dataset). This can be accomplished by calling the **getPostsByCategory(category)** function of your blog-service (defined below)

      Sample: https://web322-a3-

sample.herokuapp.com/posts?category=5

- /posts?minDate=***value***
  - return a JSON string consisting of all posts whose postDate property is equal or greater than ***value*** **-** where ***value*** is a date string in the format YYYY-MM-DD (ie: 2020-12-01 would only show posts **8** and **9**). This can be accomplished by calling the **getPostsByMinDate(minDateStr)** function of your blog-service (defined below)

    Sample: https://web322-a3-sample.herokuapp.com/posts?minDate=2020-12-01
- /posts
  - return a JSON string consisting of all posts without any filter (existing functionality)

Step 2: Add the "/post/***value***" route

- This route will return a JSON formatted string containing a single post whose **id** matches the ***value***. This can be accomplished by calling the **getPostById(id)** function of your blog-service (defined below).

  Sample: https://web322-a3-sample.herokuapp.com/post/3

Part 4: Updating "blog-service.js" to support the new "Post" routes

Note: All the functions below must return a **promise** (continuing with the pattern from the rest of the blog-service.js module)

Step 1: Add the getPostsByCategory(category) Function

- This function will provide an array of "post" objects whose **category** property matches the ***category*** parameter (ie: if ***category*** is 5 then the array will consist of only posts who have a "category" property value of 5) using the **resolve** method of the returned promise.
- If for some reason, the length of the array is 0 (no results returned), this function must invoke the **reject** method and pass a meaningful message, ie: "no results returned".

Step 2: Add the getPostsByMinDate(minDateStr) Function

- This function will provide an array of "post" objects whose **postDate** property represents a **Date** value that is greater than, or equal to the ***minDateStr*** parameter **Date** value. For example, if minDateString is "2020-12-01", then all "post" objects returned will have a **postDate** property that represents a *larger* **Date**.

  Note: Date strings in this format can be compared by creating new Date objects and comparing them directly, ie:

  if(new Date(somePostObj.postDate) >= new Date(minDateStr)){

```
  console.log("The postDate value is greater than minDateStr")
}
```

If for some reason, the length of the array is 0 (no results returned), this function must invoke the **reject** method and pass a meaningful message, ie: "no results returned".

Step 3: Add the getPostById(id) Function

- This function will provide a single "post" object whose **id** property matches the ***id*** parameter using the **resolve** method of the returned promise.
- If for some reason, the post cannot be found, this function must invoke the **reject** method and pass a meaningful message, ie: "no result returned".

Part 5: Pushing to Heroku

Once you are satisfied with your application, deploy it to Heroku:

- Ensure that you have checked in your latest code using **git** (from within Visual Studio Code)
- Open the integrated terminal in Visual Studio Code

- Log in to your Heroku account using the command **heroku login**
- Create a new app on Heroku using the command **heroku create**
- Push your code to Heroku using the command **git push heroku master**
- **IMPORTANT NOTE:** Since we are using an "**unverified" free** account on Heroku, we are limited to only **5 apps**, so if you have been experimenting on Heroku and have created 5 apps already, you must delete one (or verify your account with a credit card). Once you have received a grade for Assignment 1, it is safe to delete this app (login to the Heroku website, click on your app and then click the **Delete app…** button under "**Settings**").

Testing: Sample Solution

To see a completed version of this app running, visit: https://web322-a3-sample.herokuapp.com

Pushing code to a private GitHub repository

Once you have pushed your code from local to Heroku, **you must also push to a remote repository:**

- Add this assignment and the rest of the assignments to the **web322-app repo. (Add all assignments to the root of the project.**
- Share this repository by going to your **web322-app** repository in GitHub -> Settings -> Collaborators -> Add People -> Enter "at-seneca" and **Invite Collaborator**

Assignment Submission:

- Next, Add the following declaration at the top of your **server.js** file:

```
/*********************************************************************************
 * WEB322 – Assignment 03
 * I declare that this assignment is my own work in accordance with Seneca Academic Policy. No part
```

* Of this assignment has been copied manually or electronically from any other source
* (including 3rd party web sites) or distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
*
* Online (Heroku) Link: _____
*
**/

- Compress (.zip) your web322-app folder and submit the .zip file to My.Seneca under
  Assignments -> **Assignment** 3

Important Note:

- **NO LATE SUBMISSIONS** for assignments. Late assignment submissions will not be accepted and will receive a **grade of zero (0)**.
- After the end (11:59PM) of the due date, the assignment submission link on My.Seneca will no longer be available.
- Submitted assignments must run locally, ie: start up errors causing the assignment/app to fail on startup will result in a **grade of zero (0)** for the assignment.