WEB422 Assignment 5

Submission Deadline:

Friday, July 31st 2020 @ 11:59pm

Assessment Weight:

9% of your final course Grade

Objective:

The main objective in this assignment is to make an incremental update to the solution that you created for your Assignment 4 (ie: the "Blog" app). For this assignment, we will publish a back-end API to manage the blog posts in your existing MongoDB Atlas account as well as wire up our application to use a single service to manage the data. We will also enable users to view more than a single blog post by clicking on it from the main "blog" page. A sample solution is provided here: https://limitless-forest-06256.herokuapp.com/

NOTE: If you are unable to start this assignment because there was an issue with your Assignment 4, please email your professor for a fresh (working) copy.

Specification:

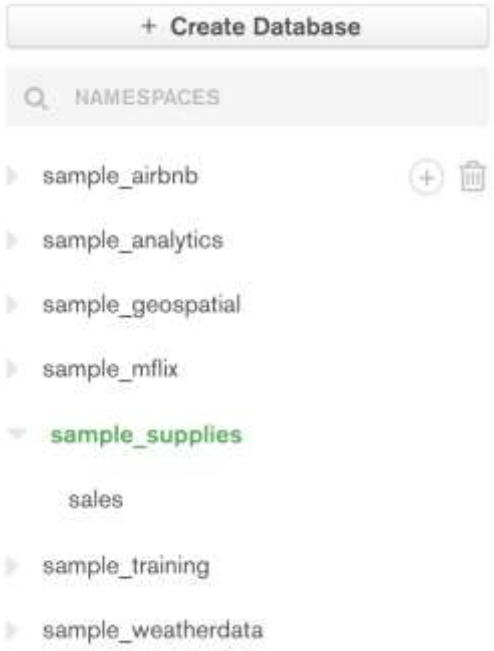
Step 1 – The Database

It's time that we move our application data out of a .json file and onto MongoDB Atlas. Fortunately, everyone should have a working database instance from back in Assignment 1. For this step, we simply need to remove all unused databases, add a new "blog" database with a "posts" collection, and populate it with data.

Removing Unused Databases

Log in to your MongoDB Atlas account and locate your "sample_supplies" database. It can be found by clicking the "collections" button for one of your clusters (if you have multiple):

Once you have located it, you should see a number of other databases, ie: "sample_airbnb", "sample_analytics", etc, etc.):



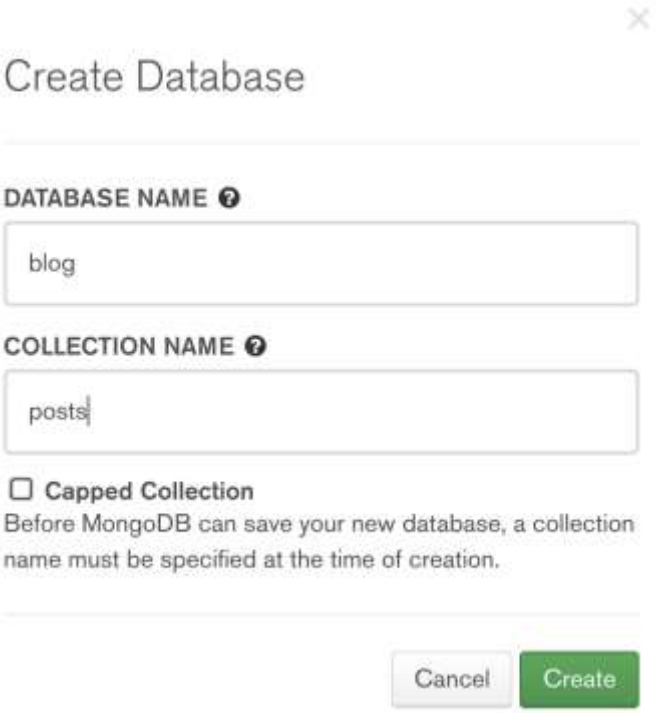
Since we will not be using any of these other "sample" datasets this semester, you can go ahead and drop them to save space. This can be done by hovering over each database and clicking the "garbage can" icon:

NOTE: Be sure **not** to drop "sample_supplies" as this will wipe out your "sales" data and your earlier assignments will cease to function.

Once this is complete, you should be left with a single "sample_supplies" database in the list:

Adding the "blog" Database

We can now go ahead and add in our new "blog" database by clicking the "+ Create Database" button. Go ahead and enter "blog" for the database name and "posts" for the collection name:



With this complete, we should now have two databases in our cluster: "blog" and "sample_supplies".

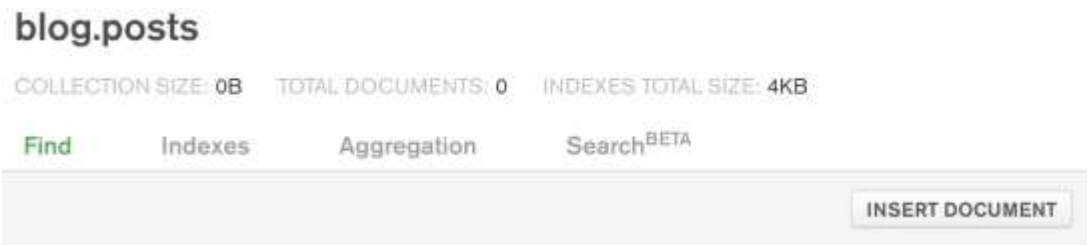
Inserting Sample Posts

Next, to ensure that our database has some content to work with, we must manually insert the same collection of "posts" from assignment 4, however without the "_id" attributes. To save you the trouble of manipulating the .json file yourself, we have provided a clean version here:

https://ict.senecacollege.ca/~patrick.crawford/shared/winter-2020/web422/A5/blogData.json

Note: Keep that page open in your browser for now, as we will have to select all the text and copy it into our new "posts" collection.

To initiate this process, first click on your "posts" collection. You should see that it has no documents yet. To insert documents manually, click on the "insert document" button towards the right edge of the screen:



This should spawn a modal window titled: "Insert to Collection".

Before we can copy & paste all of our blogData (from the link above), we first have to switch the "view". This can be achieved by toggling the button next to the text "VIEW" in the modal window. This should change your view to look like this (enabling us to paste the full "blogData" array):

Finally, delete lines 4 - 8 in the text box (effectively removing the suggested document) and proceed to copy the full content of the blogData.json file (from above) and paste it in to the window beneath the comment

With this done, we can now click "insert" to add the data!

The Connection String

The connection string for this database should be identical to your previous connection string, except with one change:

The text "sample_supplies" should be replaced with "blog", for example, if your connection string looks like the below example:

mongodb+srv://yourUserName:yourPassword@cluster0-3abc.mongodb.net/**sample_supplies**?retryWrites=true&w=majority"

it should be changed to:

mongodb+srv://yourUserName:yourPassword@cluster0-3abc.mongodb.net/**blog**?retryWrites=true&w=majority"

Be sure to copy this value, as we will need it in the next step: "Configuring / Publishing the Blog API"

Step 2: Configuring / Publishing the Blog API

For this next step, we must publish an API (similar to what was done in assignment 1). However for this assignment, rather than creating the API from scratch we can use a pre-configured "blogAPI". You can download a copy of it here:

https://ict.senecacollege.ca/~patrick.crawford/shared/winter-2020/web422/A5/blog-API.zip

Once you have downloaded and extracted the above file, open the folder in Visual Studio Code and execute the command "npm install" to fetch the dependencies.

Finally, before you can test the API locally, you must update line 1 of "server.js" to include your new MongoDB Connection String (obtained above) for the "blog" database, ie:

mongodb+srv://yourUserName:yourPassword@cluster0-3abc.mongodb.net/**blog**?retryWrites=true&w=majority"

Once you have saved the changes, start the server with the command "node server.js". With the server running, access the route:

http://localhost:8080/api/posts?page=1&perPage=10

You should now see your data!

Please feel free to play around with the API at this point, you will find that you can access posts by id, as well as filter posts by category (ie: http://localhost:8080/api/posts?page=1&perPage=10&category=Adventure) or tag (ie: http://localhost:8080/api/posts?page=1&perPage=10&tag=lovedit) .

Pushing to Heroku

Now that our API is correctly configured, please push it to Heroku and save the link, as we will be using it throughout this assignment.

Step 3: Adding a "PostService" to our App

Before we can start working on the Blog App for Assignment 5, we must first make a copy of our completed Assignment 4 App (once again: If you are unable to start this assignment because there was an issue with your Assignment 4, please email your professor for a fresh (working) copy).

Open the folder containing the copy of Assignment 4 (now Assignment 5) in Visual Studio code and execute the "ng" command in the terminal to create a service with the class **PostService** (defined in post.service.ts).

This service will be responsible for working with our blog API (now on Heroku) using the HttpClient. Therefore, we must make a few updates to our existing code, ie:

- In **comment.ts** remove the _id property (the data in MongoDB does not contain one)
- Import and add the "HttpClientModule" to the "imports" array in app.module.ts

With the above configuration complete, we can now implement the following functions within our PostService:

NOTE: Set a constant variable called "perPage" at the top of the file and set its value to 6. This will be our maximum results "perPage"

getPosts(page, tag, category): Observable<BlogPost[]>

This method will use HttpClient to return all of the posts available in the blogAPI for a specific page using the path: /api/posts?page=***page***&perPage=***perPage*** where ***page*** is the first parameter to the function and ***perPage*** is the constant defined (above).

Additionally, if the "tag" parameter is not null / undefined, then add &tag=***tag*** to the path.

IMPORTANT NOTE: The API will not permit you to use "#" in the "tag" parameter, ie: "tag=#scary" will not function, but "tag=scary" will. Therefore, never invoke this function with a tag value that includes "#".

Finally, if the "category" parameter is not null /undefined, then add &category=***category*** to the path

getPostbyId(id): Observable<BlogPost>

This method will use HttpClient to return a single post available in the blogAPI for a specific page using the path: /api/posts/***id*** where ***id*** is the single parameter to the function.

getCategories(): Observable<any>

This method will use HttpClient to return an array of "Categories" in the format: {cat: ***string***, num: ***number***} using the path /api/categories (Note: You should recognize this format, as it was the one used in assignment 4 for the "CategoriesComponent" data.)

getTags: Observable<string[]>

This method will use HttpClient to return an array of "Tags" (represented as strings) using the path /api/tags.

Step 4: Refactoring the "Blog" view (BlogComponent)

With all of our PostService methods in place, we can now start refactoring our code to start using it.

The best place to start is with the "BlogComponent", since this is the primary interface for interacting with the blog.

To begin, open up blog.component.ts and make the following updates:

- Remove the import line for the "blogData.json" file (we will no longer be requiring it)
- Import & inject the "PostService" service
- Import & inject the "ActivatedRoute" service
- Make sure that you do not initialize the "blogPosts" property to anything by default (we'll be using our service instead)
- Add the following properties to the component
    - page (type: number, initial value: 1)
    - tag (type: string, initial value: null)
    - category (type: string, initial value: null)
    - querySub (type: any, [no initial value])
- In the ngOnInit() function, add the following logic

```
this.querySub = this.route.queryParams.subscribe(params => {

 if(params['tag']){
  this.tag = params['tag'];
  this.category = null;
 }else{
  this.tag = null;
 }

 if(params['category']){
  this.category = params['category'];
  this.tag = null;
 }else{
  this.category = null;
 }

 this.getPage(+params['page'] || 1);

});
```

- Add a new getPage(num) function with the following logic:
    - Get all of the blog posts using the values of num, this.tag and this.category.
    - If the length of the data coming back > 0, set this.blogPosts with the data and this.page with num
- In the ngOnDestroy() function, add the following logic:
    - if(this.querySub) this.querySub.unsubscribe();

Step 5: The PagingComponent

With our "blog" page now showing live data from our database, it's time to develop a new component to help with our pagination tasks within the BlogComponent. To begin, create a new component called "PagingComponent" with the following specification:

- The following HTML can be used as its starting template:

```
<nav aria-label="Page navigation">
  <ul class="pagination pagination-template d-flex justify-content-center">
    <li class="page-item"><a class="page-link"> <i class="fa fa-angle-left"></i></a></li>
    <li class="page-item"><a class="page-link">1</a></li>
    <li class="page-item"><a class="page-link"> <i class="fa fa-angle-right"></i></a></li>
  </ul>
</nav>
```

- This component will accept a single property "**page**" (type: number)
- It will emit (output) the event "**newPage**" which will contain the new page number obtained by clicking on one of the "left" or "right" "page-item" buttons (details below)
- If the first "page-item" button is clicked (ie: "<") , execute the following logic (placed in a separate "callback" function):
    - If the value of "**page**" is greater than 1 emit the "**newPage**" event with a value of "**page**" - 1
- If the last "page-item" button is clicked (ie: ">") , execute the following logic (placed in a separate "callback" function):
    - emit the "**newPage**" event with a value of "**page**" + 1

Once completed, the component must be placed beneath the <app-post-card></app-post-card> element in blog.component.html using the html:

** <app-paging></app-paging> **

Note: Do not forget to provide <app-paging> with a "[page]" input parameter (set to the value of "page" in the blog component), as well as wire up the emitted "(newPage)" event to execute "getPage($event)" – declared above.

Step 6: The CategoriesComponent

Now that we're working with live blog data, we must refactor our CategoriesComponent to use it as well. Start with updating the class "CategoriesComponent" (categories.component.ts) according to the following specification:

- Remove the hardcoded data assigned to the "categories" array
- Import & inject the "PostService" service
- In the ngOnInit() function, add logic to populate the "categories" array using the "PostService" getCategories() function.
- Update the categories.component.html **template** to use the following "routerLink" attribute (assuming that "cat" is the current "category":
    - [routerLink]="['/blog']" [queryParams]="{ category: cat.cat }"

Step 7: The TagsComponent

We must refactor our TagsComponent to use our new service as well. Start with updating the class "TagsComponent" (tags.component.ts) according to the following:

- Remove the hardcoded data assigned to the "tags" array
- Import & inject the "PostService" service
- In the ngOnInit() function, add logic to populate the "tags" array using the "PostService" getTags() function.
- Update the tags.component.html **template** to use the following "routerLink" attribute (assuming that "tag" is the current "tag". Note: we eliminate the leading "#" by using substring(1)):
    - [routerLink]="['/blog']" [queryParams]="{ tag: tag.substring(1) }"

Step 8: The LatestPostsComponent

Up until now our "LatestPostsComponent" has been relying on a "posts" property for its data. However, now that we are using an external data source, we can refactor this component to fetch the data itself. First begin by opening "latest-posts.component.ts" and making the following changes:

- Remove the @Input() decorator from "posts"
- Import & inject the "PostService" service
- In the ngOnInit() function, add logic to populate the "posts" array using the "PostService" getPosts(1, null, null) function. However, make sure that you only grab the first 3 results. This can be done by once again using the .slice(0,3) method on the returned array

Next, locate both instances of the <app-latest-posts> component (blog.component.html & post.component.html), it should look something like:

- <app-latest-posts [posts]="blogPosts.slice(0,3)"></app-latest-posts>.

We simply want to remove the [posts] property, therefore it should be changed to

- **<app-latest-posts></app-latest-posts>**

Finally, in the template (latest-posts.component.html), update the **routerLink** attribute to link to the correct "post", ie: [routerLink]="['/post', post._id]"

Step 9: The PostCardComponent & Routing Update

With our main "Blog" page in fairly good working order, we must finally update our PostCardComponent so that we can click on a specific blog post from within the "Blog" page and access it directly. The main task here is to open the template (post-card.component.html) and make the following key change:

- Both **routerLink="/post"** properties within the template need to be changed to the below (assuming "post" is the current "post" object: **[routerLink]="['/post', post._id]"**

Lastly, for this route to function properly, we need to update our "app-routing.module.ts" file to ensure that "post" has an "id" route parameter, ie "**post/:id**".

Step 10: The PostDataComponent

The last major component to update for this assignment is the "PostDataComponent". To begin, open the post-data.component.ts file and make the following changes:

- Add the following property to the component
    - querySub (type: any, [no initial value])
- Remove the @Input() decorator from the "post" property as well as update any instance of the component that uses the [post] property binding, ie:

  <app-post-data [post]="blogPosts[0]"></app-post-data>

  Should be changed to:

  <app-post-data></app-post-data>

- Next, Import & inject the "PostService" service
- Import & Inject the "ActivatedRoute" service
- In the ngOnInit() function, add the following logic assuming that "route" references the ActivatedRoute service (NOTE: be sure to fill in the "TODO" with your own code)

```
this.querySub = this.route.params.subscribe(params =>{
  //TODO: Get post by Id params['id'] and store the result in this.post
})
```

- In the ngOnDestroy() function, add the following logic:
    - if(this.querySub) this.querySub.unsubscribe();
- Finally, open up the template (post-data.component.html) and locate the element. Inside, you will find an empty "routerLink" that needs to be refactored to link to the "blog" page with the correct tag filter, ie:
    - [routerLink]="['/blog']" [queryParams]="{ tag: tag.substring(1) }"

Step 11: New FooterPostsComponent

We're almost done with the major components for the Blog and Page views, however we are still showing static (unrelated) blog posts in the footer of our blog. To fix this, let's add a new component: "FooterPostsComponent" (Note: this component is very closely related to our "LatestPostsComponent", so we will actually be recycling the code from that class)

Once you have crated your "FooterPostsComponent", open the "footer-posts.component.ts" file and replace the code within the class with exact copy of the code within your "LatestPostsComponent" class (this will include the "posts" property, the constructor, and the ngOnInit() function).

Note: "BlogPost" and "PostService" will need to be imported once this is complete.

Updating the Template

The "static" template for this component exists near the bottom of your "footer.component.html" file. You should see a … . Delete this entire element.

Now, instead of rendering the full element in the footer.component.html file, we will use the new "FooterPostsComponent", ie:

<app-footer-posts></app-footer-posts>

Using "posts" in the Template (footer-posts.component.html)

For this template, you can use the following HTML for a single post (Note: You will have to repeat each <div class="post" for each "post")

```
<div class="latest-posts">
  <div class="post">
    <a routerLink>
      <div class="post d-flex align-items-center">
        <div class="image"><img src="featuredImage" alt="..." class="img-fluid"></div>
        <div class="title"><strong>title</strong><span class="date last-meta">postDate</span></div>
      </div>
    </a>
  </div>
</div>
```

- The "routerLink" attribute should be changed to link to the correct post (using post._id)
- The <img> should use the "featuredImage" for the post
- The "title" should be the "title" for the post
- The "postDate should be the "postDate" for the post

Step 12: Angular "Date" Pipe

Finally, as a last step to clean our output, we will use Angular's date pipe functionality. The way this works is that whenever a date is rendered, ie {{post.postDate}}, you will pipe the output to date: "mediumDate", ie:

{{post.postDate}} becomes **{{post.postDate | date:'mediumDate'}}**

Note: Do not forget to "pipe" the date for your post comments as well!

Extra Challenge!

When running our app, you might have noticed something slightly annoying from a usability perspective. When we change routes, our scroll position on the page remains the same! Ideally, we should be scrolled back to the top after every route change. As an extra challenge, try researching this topic for yourself and implementing it within your application.

Assignment Submission:

- Add the following declaration at the top of your app.component.ts file:

```
/*********************************************************************************
* WEB422 – Assignment 05
* 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 web sites) or
* distributed to other students.
*
* Name: _____ Student ID: _____ Date: _____
*
*********************************************************************************/
```

- Compress (.zip) the all files in your Visual Studio code folder **EXCEPT the node_modules folder** (this will just make your submission unnecessarily large, and all your module dependencies should be in your package.json file anyway).
- Submit your compressed file (without the node_modules folder) to My.Seneca under **Assignments** -> **Assignment 5**

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.