Lappeenrannan teknillinen yliopisto

School of Business and Management

Software Development Skills

Ali Bitarafan, X100347

LEARNING DIARY, NodeJS Module

**LEARNING DIARY**

15.3.2021

# NodeJS

I started to learn about NodeJS. I learned some interesting facts about it such as it is written in C++ in the lower levels and that it is extremely fast. There are various reasons why Node is used in the industry. One of them is that you can use the same language for front-end and back-end. Additionally, I learned that Node is a non-blocking I/O run time environment where tasks or processes are handled as events leading to high throughput and scalability in apps with many I/O operations. It is important to note that NodeJS is not a ideal solution for CPU intensive calculations that take significant amount of time to complete.

Node’s Event loop is handled on a single thread, and concurrency is supported via events and callbacks. To bind events and listeners, Event Emitters are used.

I managed to create a Node project and practice the following:

* Creating package.json file using npm init
* All the packages used in the project will be added to package.json file.
* If a package is only to be used during development, you can install it by adding -D or --save-dev to the npm install command, e.g., npm install -D nodemon.
* When the application is moved or deployed somewhere else, the node\_modules are not required to be moved in fact, npm install will recreate the node\_modules folder in the project directory.
* All the versioning of the packages are monitored in the package-lock.json

17.3.2021

I continued to learn NodeJS by doing the hands-on exercises as carried out in the video lectures. The following is the list of topics/pacakges covered today.

* Path: accessing directory and base file. Converting Strings into path objects as well as concatenation.
* Fs: Creation of folder, file as well as renaming files. I/O operations such as read write and append.
* Os: Platform and CPU architecture, CPU core info, Free mem, Total mem, Sys uptime.
* url: serilizing url, root domain, hostname, path name, serialized query, parameters addition and looping through them.

19.03.2021

Continuation of NodeJS introduction, with the following topics:

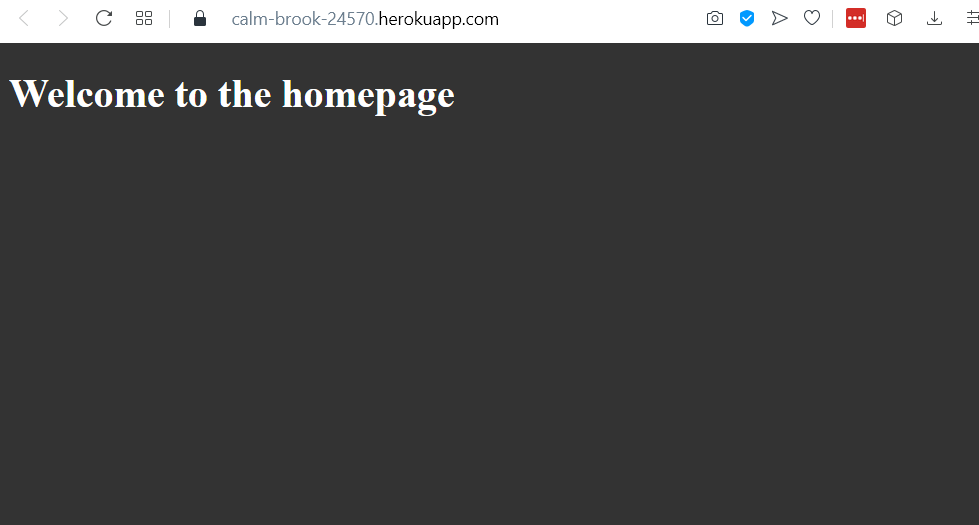
* Event: creating emitter class by extending EventEmitter class. Creating Listener and initializing an event.
* UUID: For generating new unique identifiers.

22.3.2021

NodeJS continuation, Coveting the following topics:

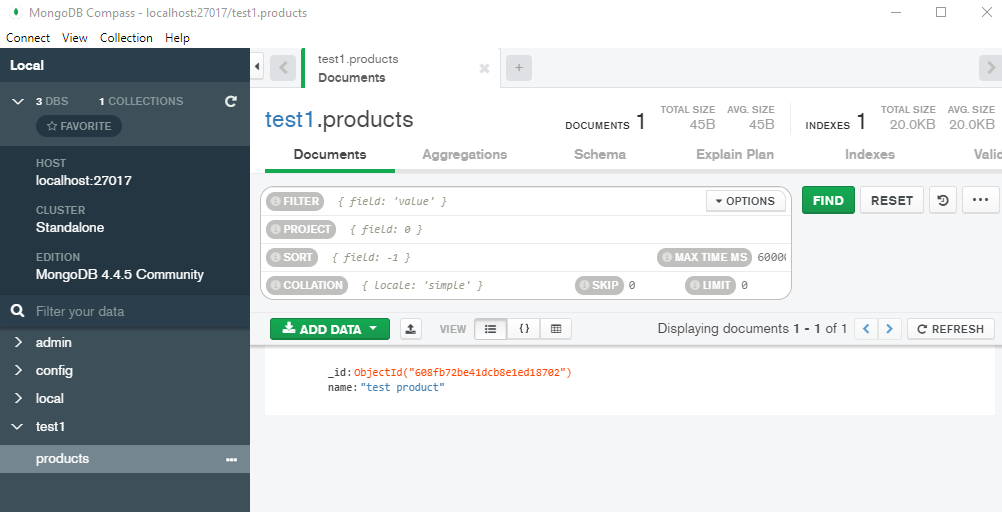
* Creating scripts instead of running “node index”
* Using “nodemon index” instead of “node index” in the script, running the server using “npm run” allows us to update the site automatically whenever we modify the code.
* Handing requests and responses by setting content type in writeHead function and content in the end function.
* Handling response types. Use of switch to set content Type for responses when handling server requests.

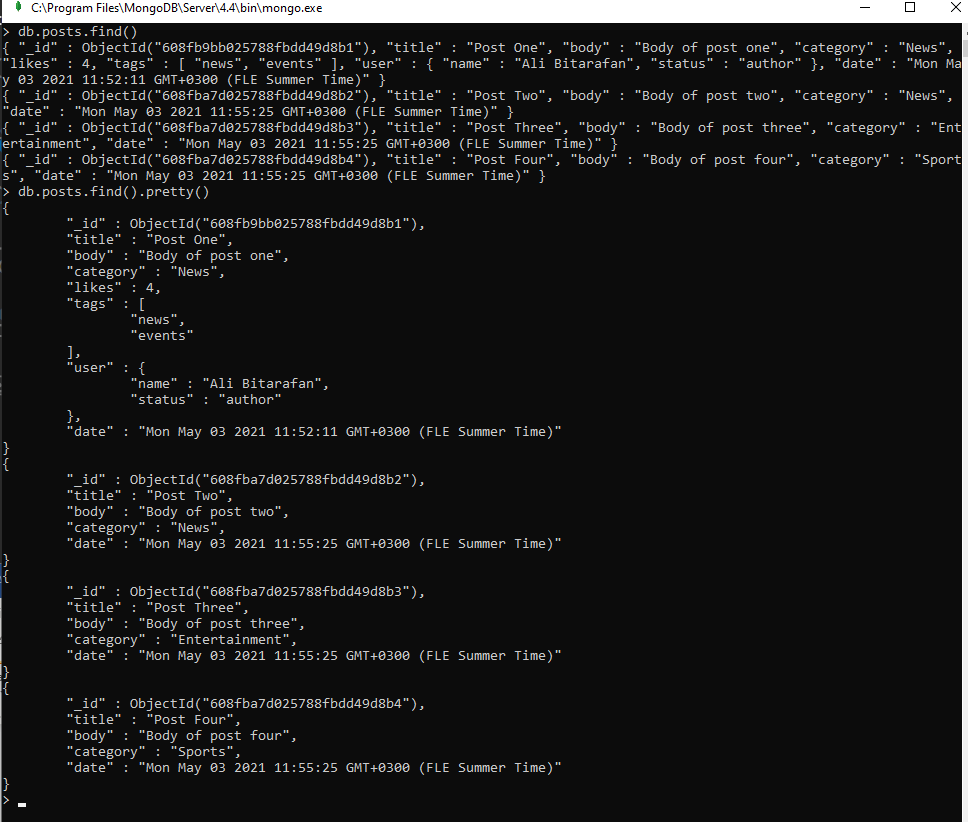
3.5.2021

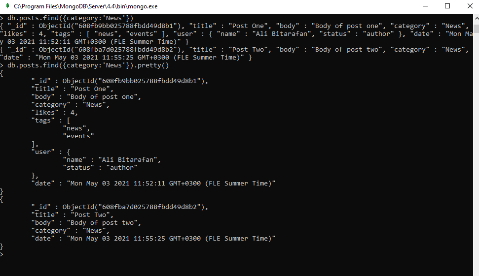
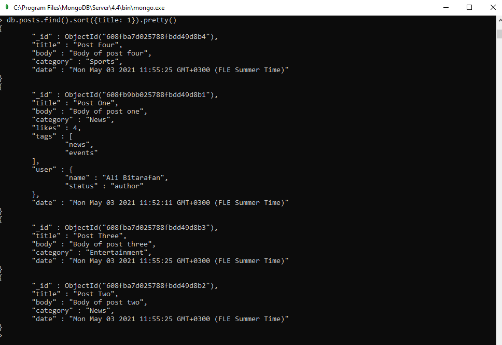
NodeJS continuation, deploying the Welcome to NodeJS app to Heroku,

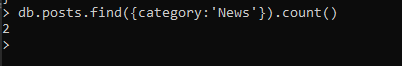
* Using Heroku command line interface I managed to create an app.
* I connected Heroku‘s remote repository (belonging to the newly created application) to the local NodeJS app and then pushed the code to Heroku.
* The application is accessible under the following link: <https://dashboard.heroku.com/apps/calm-brook-24570/deploy/heroku-git>

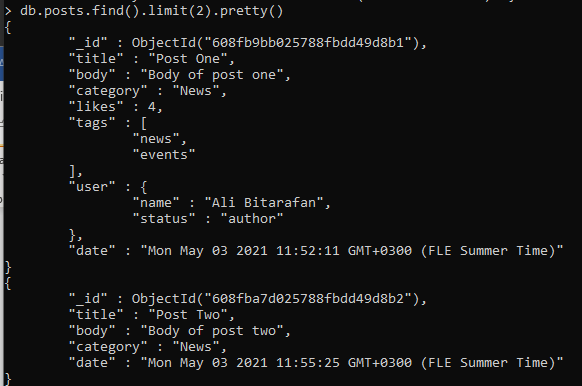
# MangoDB

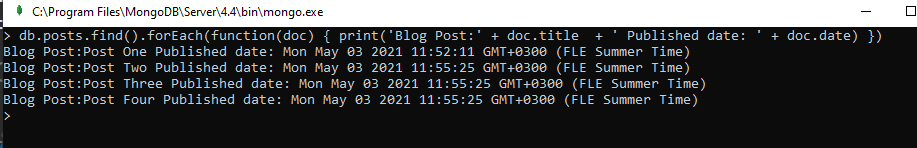
* I downloaded and installed MongoDB and managed to open the MongoDB compass.
* Mongo shell is ideal to interact with the MangoDB and the running databases.
* Posting documents is not limited to single entries, the picture in the bottom shows multiple entries using a single post call and using insertMany built in function.

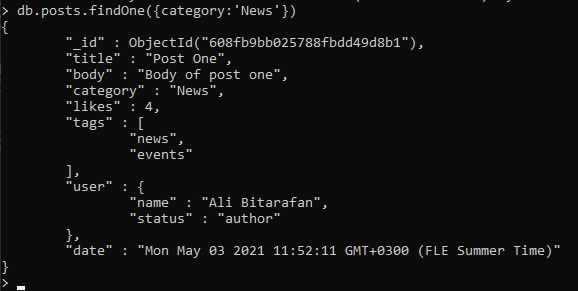
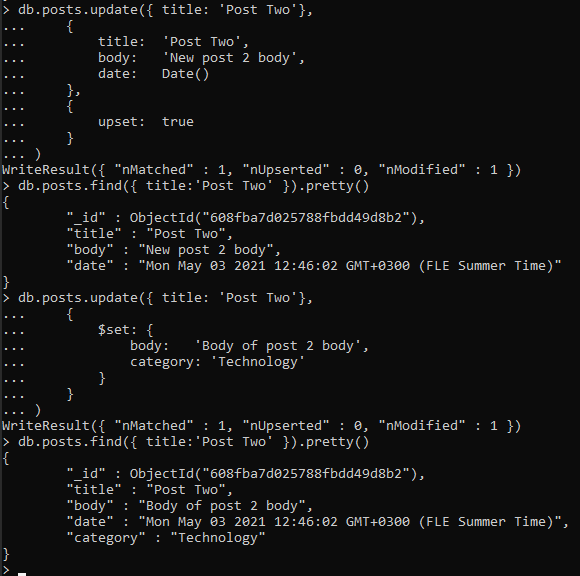
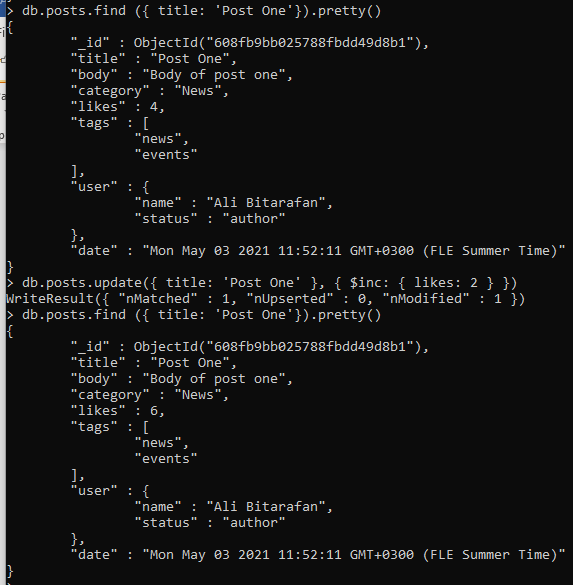


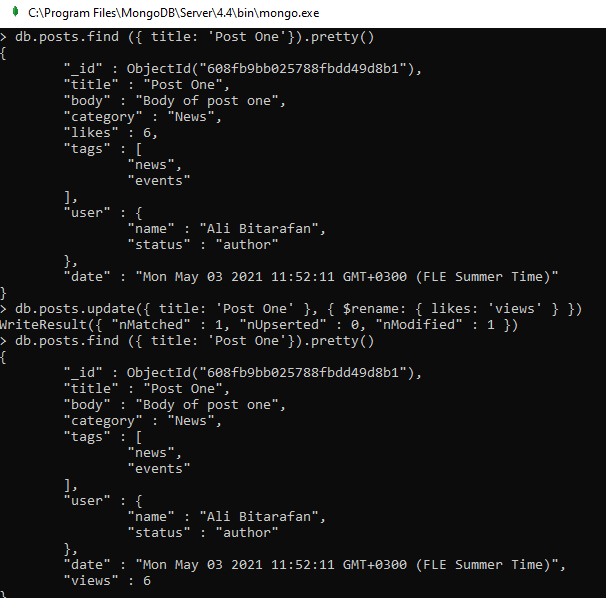
* Querying or finding data is done using “db.posts.find()” command which returns all the posts. In order to make the readable we can use the pretty function.
* To narrow down the results, we can use search terms similar to SQL Where statement.
* Sorting can be done by using the sort fucntion which requires indicating the field and using 1 for ascending or -1 for descending.

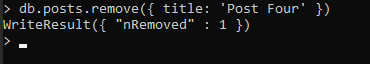
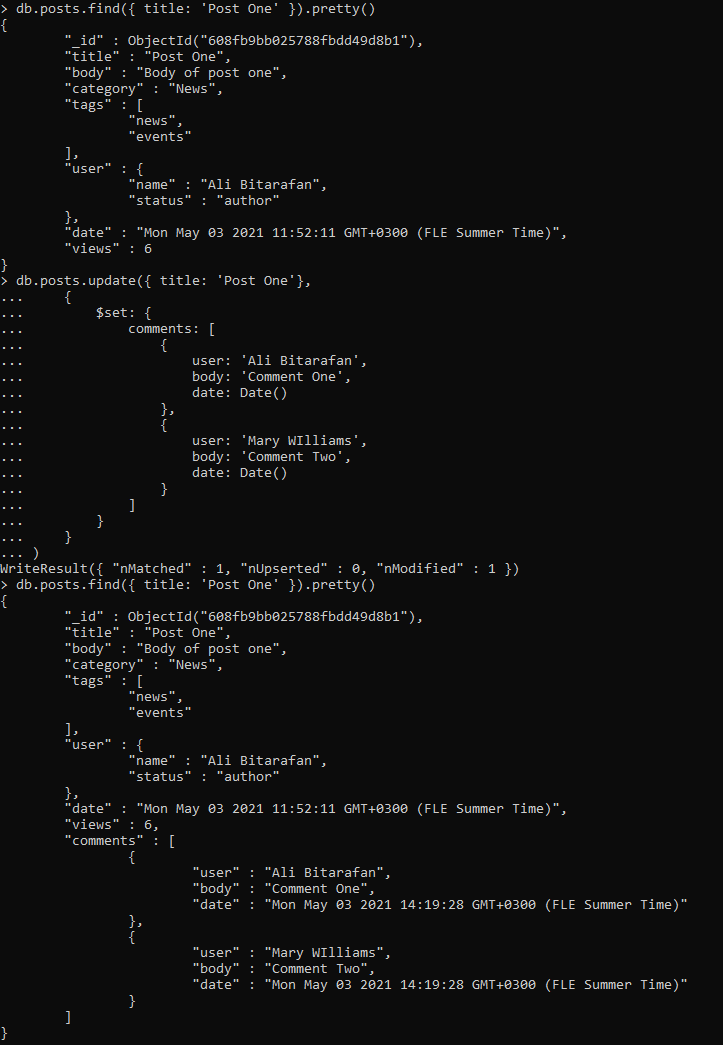
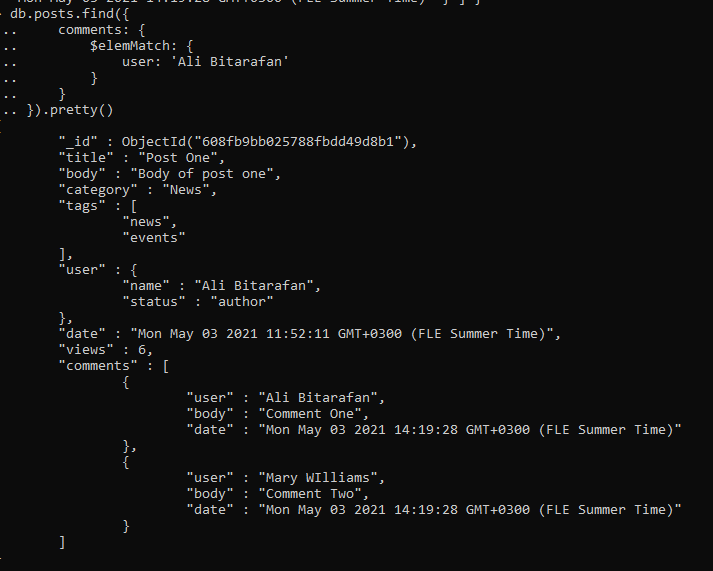


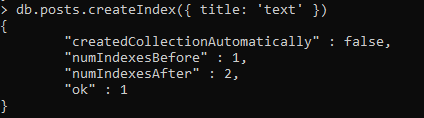
* Counting documents.
* Limiting the returned values.



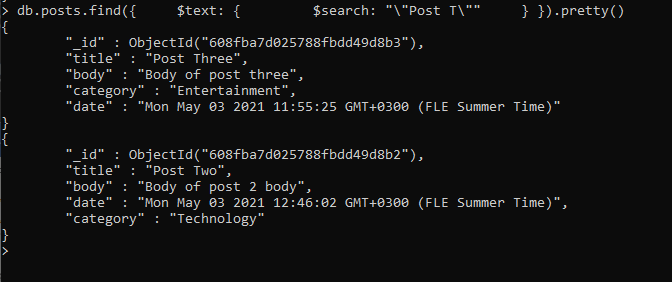
* Using forEach to loop through documents and perform operations.
* Using findOne to find only the first instance found from a query.
* To update a field, we use update function. First parameter defines which document by specifying a unique field and its value. Then the second parameter will be used to update the document.
* To update a document in such a way that only the specified values are updated, and the rest stay intact. We use the $set key.
*  To increment values, we can use $inc as shown in the following picture with incrementing the value of likes.

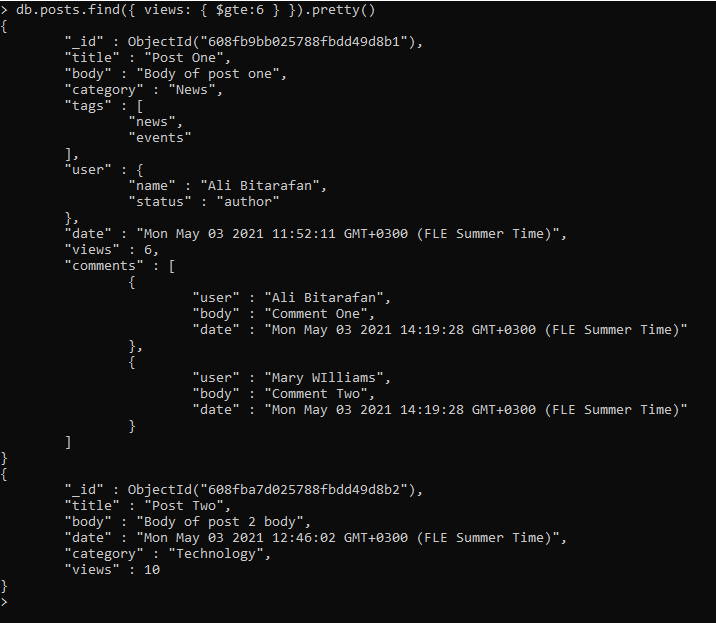


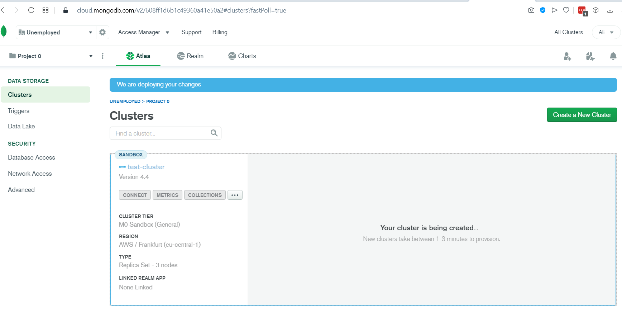
* To rename a field we can use the $rename operator.
* To delete an entry, we would use remove function and pass in a unique identifier as parameter.
* To add sub-documents to an entry we can use $set as shown in the picture to the right.
* To find posts that have a certain value in sub document, we can use $elemMatch. As shown in the following example.

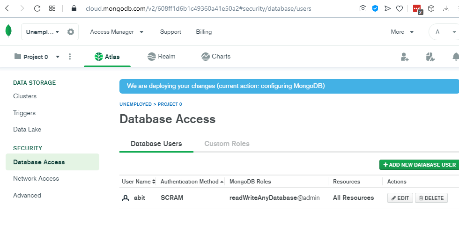
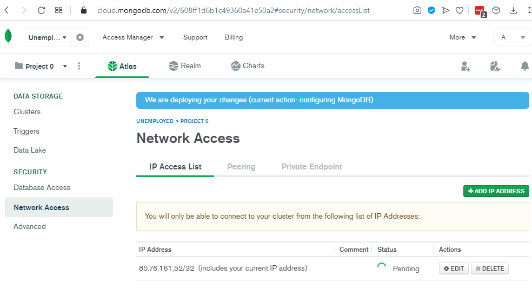


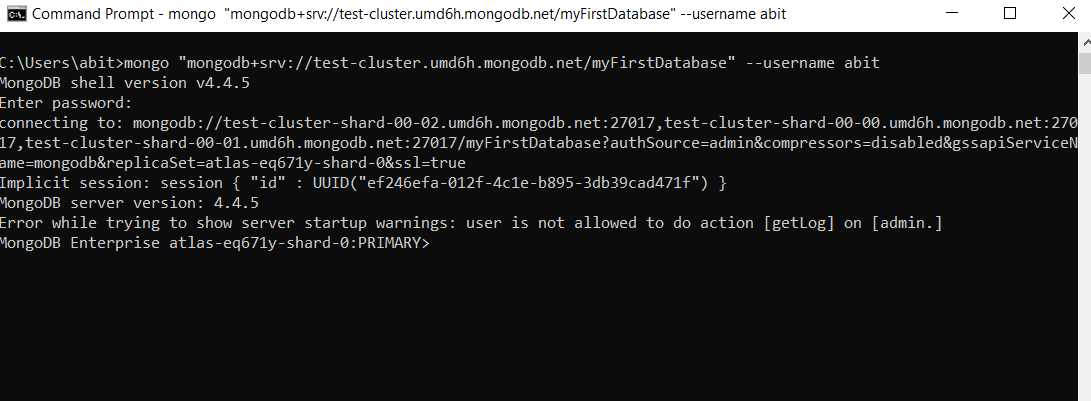
* To search in the database, we need to create index, using the following example.

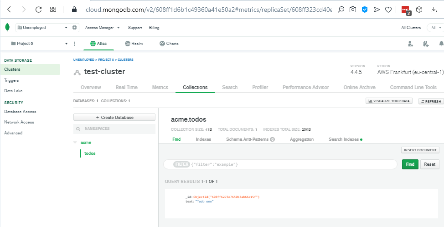
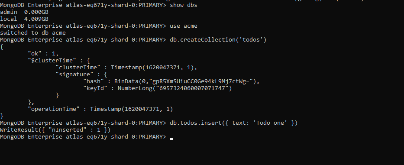


* An example of search using $text operator.
* Searching using greater than conditional operands ($gt or $gte for greater than or equal).



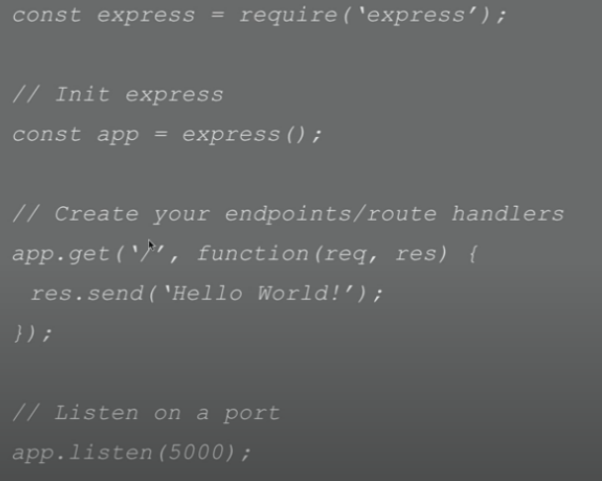
* Created a MongoDB cluster and named it test-cluster.
* Created database user under Password auth method and configure network access.



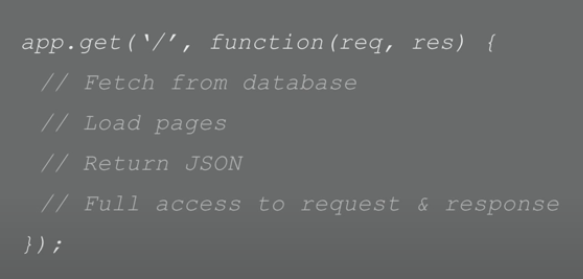
* Connecting to the test-cluster.
* Testing the creation of new collection and inserting data.

**Express JS**

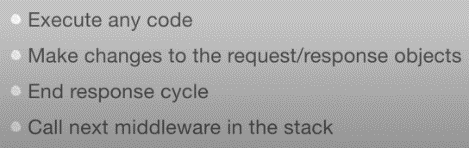
* Express is a flexible and open framework and the most popular one for NodeJS.
* Express makes NodeJS webapps easier by simplifying server rendered apps as well as API/Microservices.
* It is extremely light, fast, free.

Basic Express Server Syntax 🡪

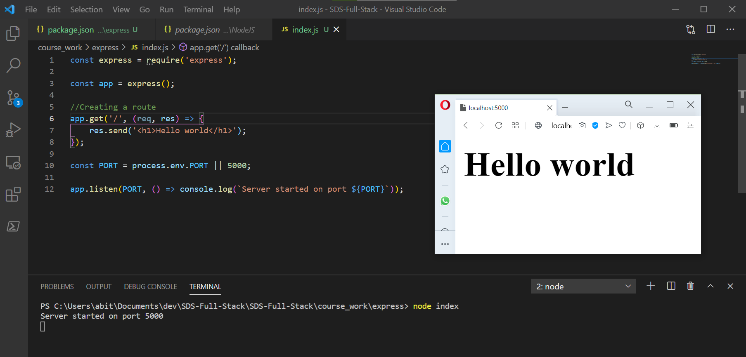
* To initialize express, we assign it to a variable, in this case ‘app’.
* To create endpoints, we define routes. In this case we accept get request to the index route or ‘/’ and then we have a callback function that takes in a request and response.
* To receive API calls we need to listen on port.

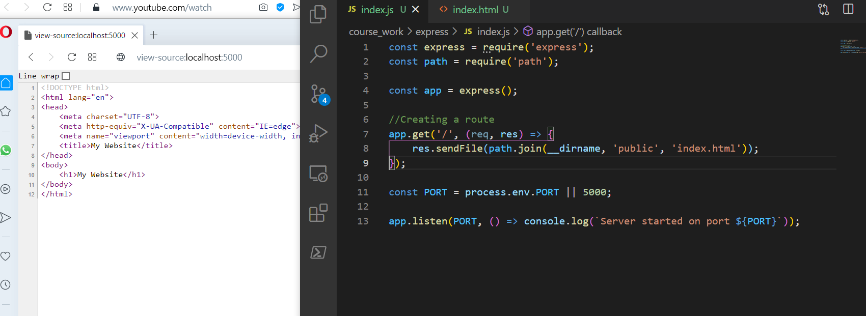
Basic route handling. Within route we can do many things. Usually, we handle requests by sending or receiving data, rendering pages, accessing parameters. The request and response objects are arguably the most important elements of routing. The request represents http request properties such as URL parameters, query strings and any data that is sent within the body, the HTTP headers, etc. The response object represents the HTTP response with which one can send back json data or render a template or even redirection using ‘res.redirect()’ and so on. Express has a router so that we can store them in separate files and export. We can also store incoming data with the Body Parser.

Middleware functions have access to request and response objects. Express has built in middleware but middleware also comes from 3rd party packages as well as custom middleware.

Middlewares possess these characteristics 🡪

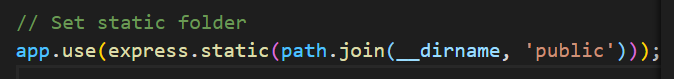
4.5.2021

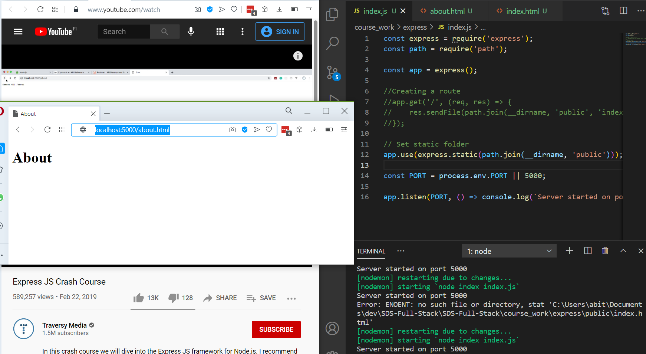
Express Hands-on exercises start by sending a simple “Hello world” response message.

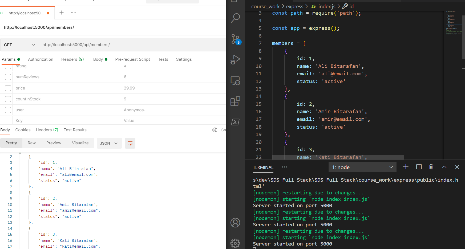
Next step is to create a index.html file and use it as a template for sending response.

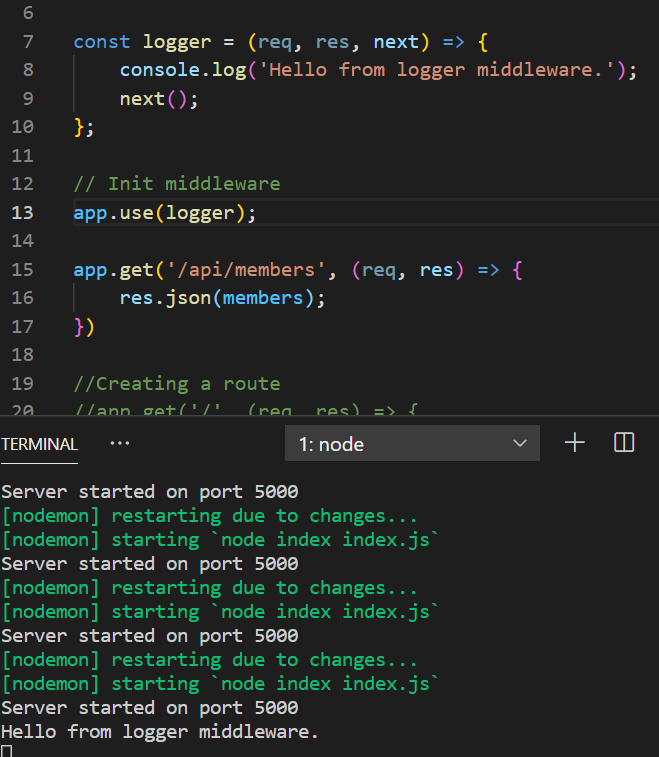
The previously mentioned method requires defining routes manually for every page.

To create a static server for different pages we can use a static folder as shown below.

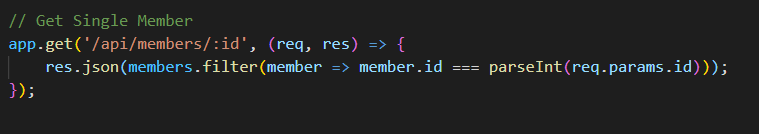
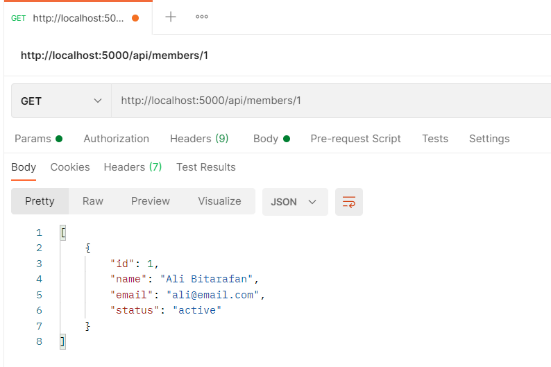


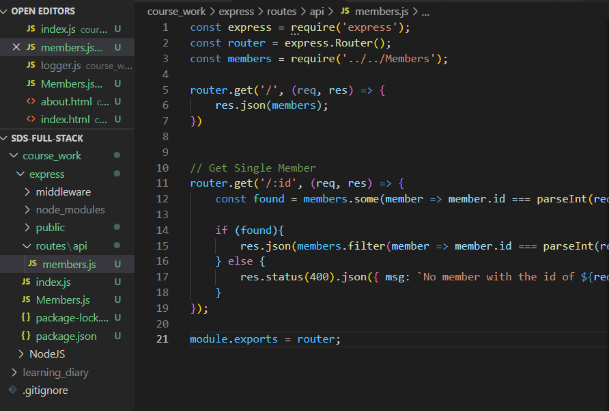
The above code turns the public folder and any html file inside it into a routable static page. Keep in mind that use of ‘html’ file extension is necessary when routing in static folder.

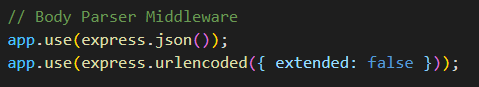
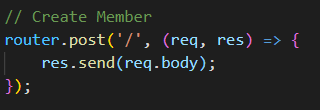
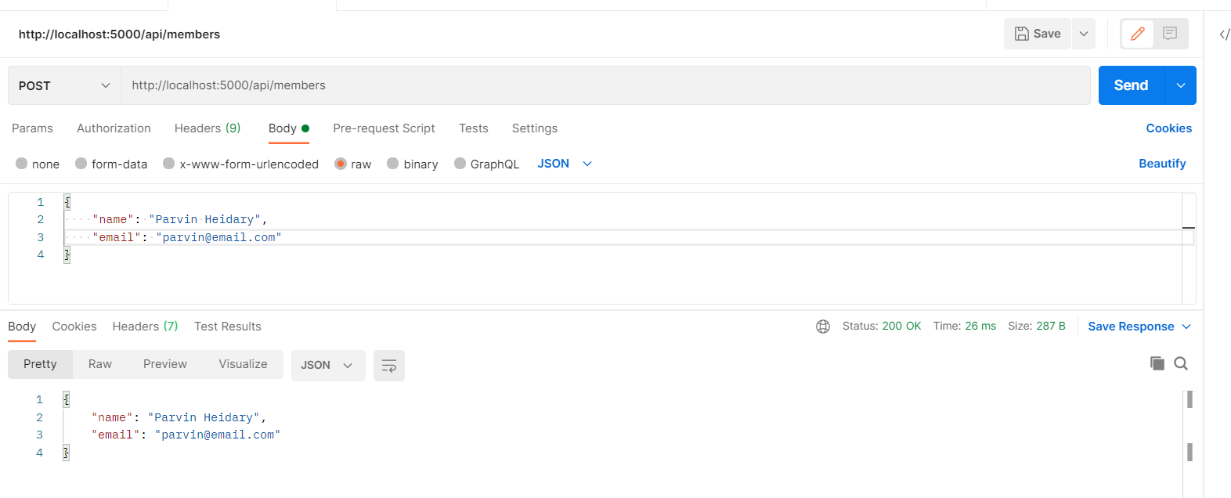
To create dynamic web site and use json document between backend and frontend we can use json function as shown in this picture 🡪.

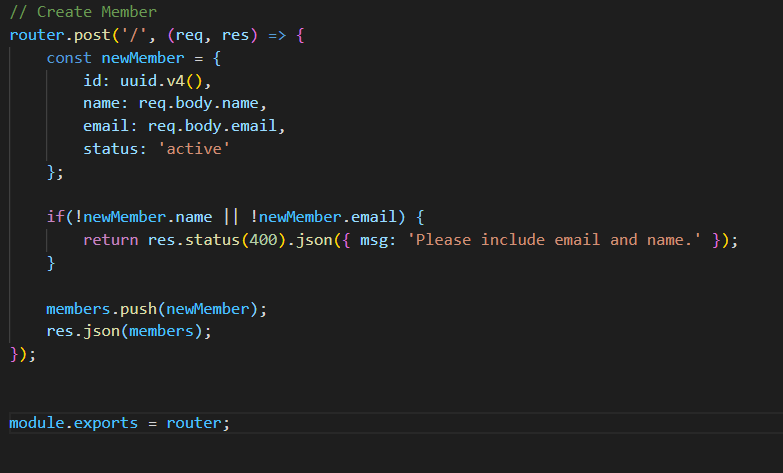
 Additionally, we are going to create a middleware called logger. When creating middleware, it takes req, res and next. In order to initialize it we call the use function on app and pass in the middleware.

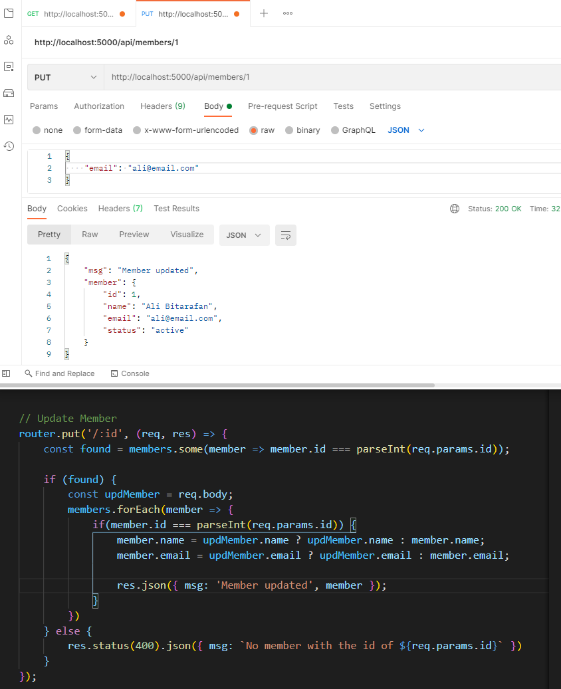
Next, to get a single entry when there is a query to members’ json document.



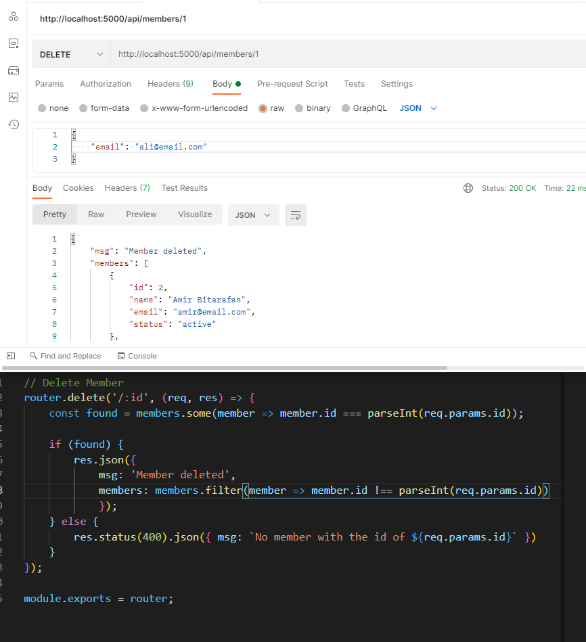
To make the index.js file simpler, we are going to move the routing functions to a new place and us the express Router function to manage the routing of API calls.

Express has built in API call body parsers that can be used to parse data from API calls.

After updating the function to create a new member using API post request the function looks like this 🡪

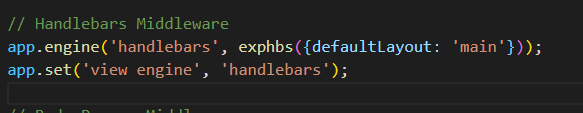


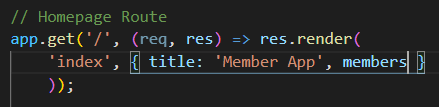
To update a member in the json doc, we use the put method as shown in the following image 🡪

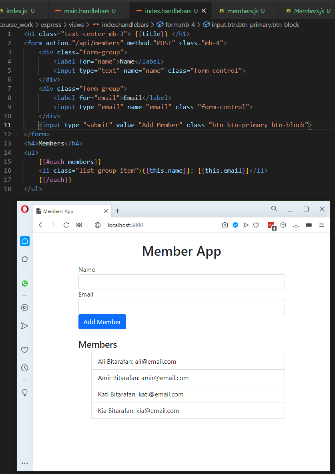
To Delete a member we use the delete method as shown in this image 🡪

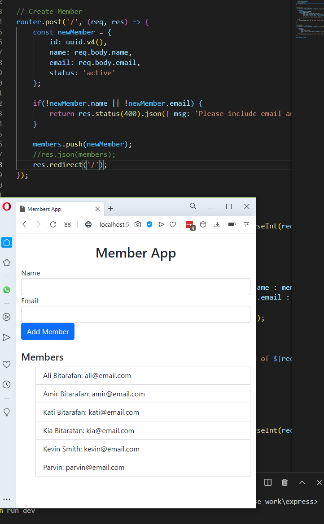
Next we will render pages using template engines. we will make views folder and wrap all the views using res.render()

First we set up handlebars middleware using the following lines of codes.



Then we can use the handlebars template files to create dynamic pages.





Add Memeber button will trigger the POST method from the members.js file in routes/api 🡪

5.5.2021

Starting MEAN stack course exercise

# MEAN Stack project

What is covered?

* REST API – Node.js / Express
* Token Generation & Authentication
* CORS
* Mongoose ODM
* Angular 2 / Angular-CLI
* Angular Router, HTTP Module
* Angular2-JWT
* Auth Guard
* Angular Flash Message Module
* Compile & Deploy

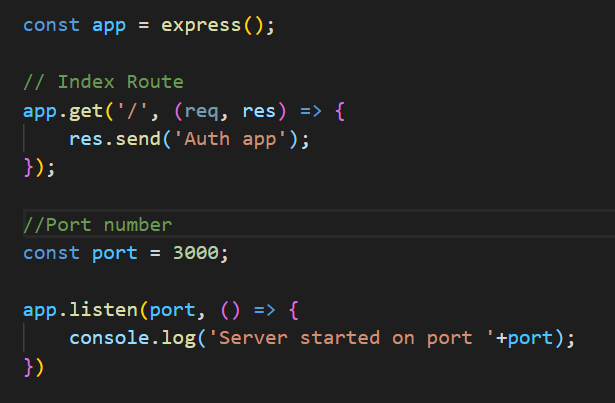
Procedure:



First, we run *npm init* *-y* in the command line and create a *package.json* file and after that we add the packages to it as listed🡪

Then we set up the express server in the *app.js* file and import the required packages:

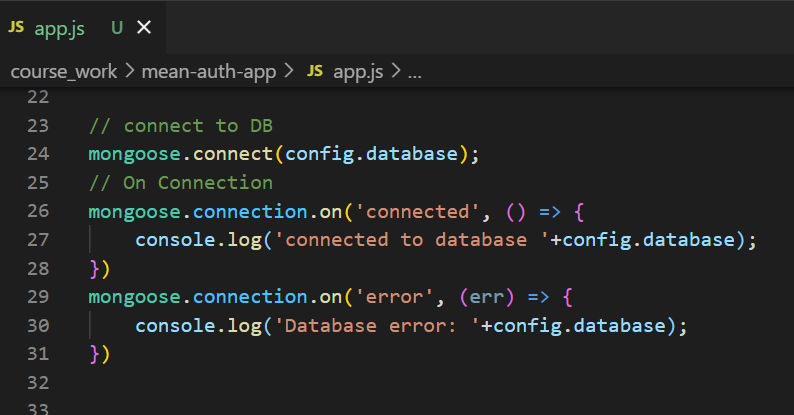




Initialization of express and responding to root url call.

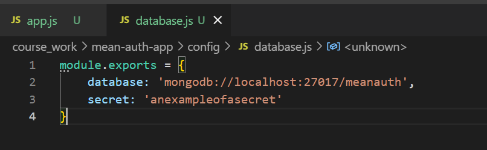
Defining a port to listen for client requests.

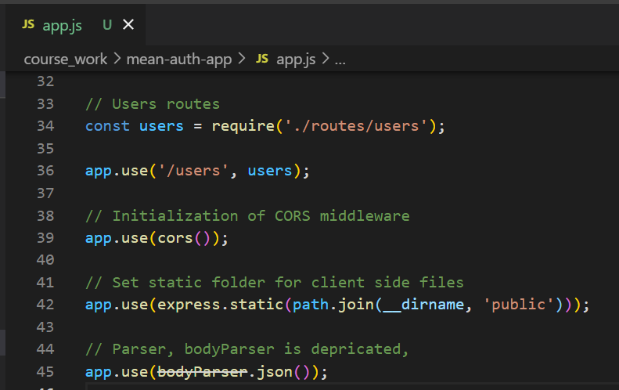
Initializing acceptance of requests using the predefined port value.



For database connection we take advantage of mongoose package and connect using the following lines of codes.

*Mongoose.connect* takes in a database url as parameter which in this case is passed in from a separate file for security reasons.



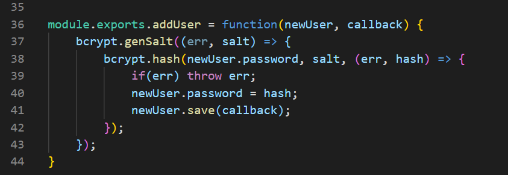
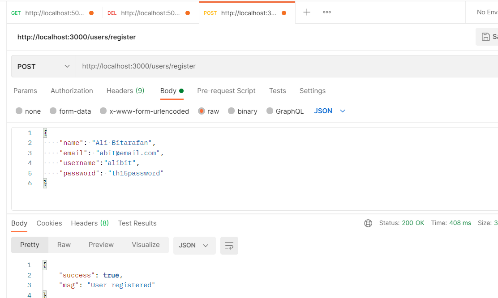
We need to define a custom users schema and pass its route to be able to use it in authentication process.(L34)

We use *cors* middleware (L39)🡪

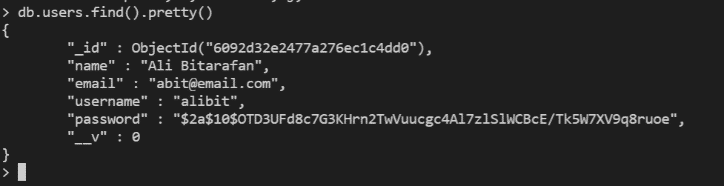
Even though bodyParser is depreciated but we use it for parsing client json docs (L45)🡪

In *users* under *routes* dir, we define the CRUD operations for a user profile such as POST requests to ‘/users/register’ 🡪

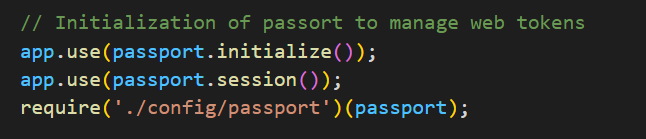
addUser method defined in custom user model.



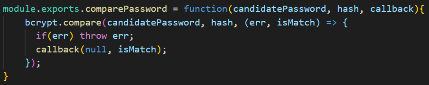
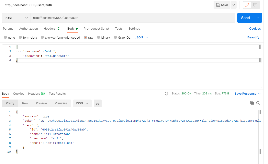
mongodb shell - query of users returns the newly creted user.



postman – result of posting new user.

Then we initialize the *passport* middleware 🡪

Then we apply a strategy,

Passport configuration 🡪

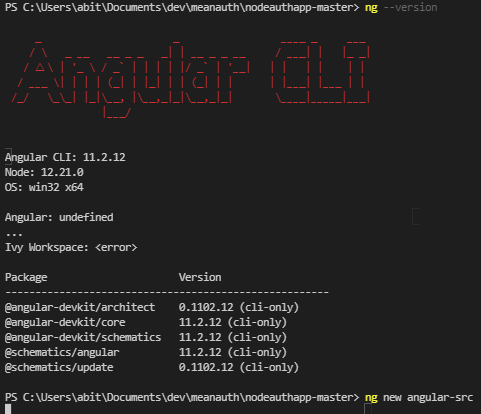
Authentication test

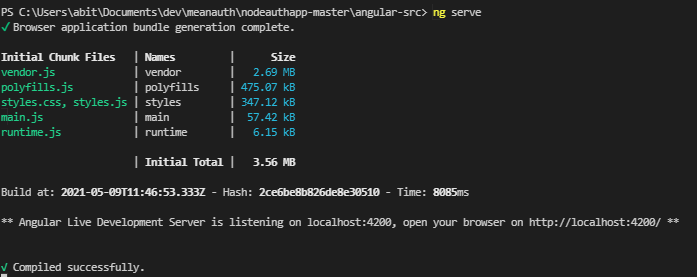
compare password function in user model

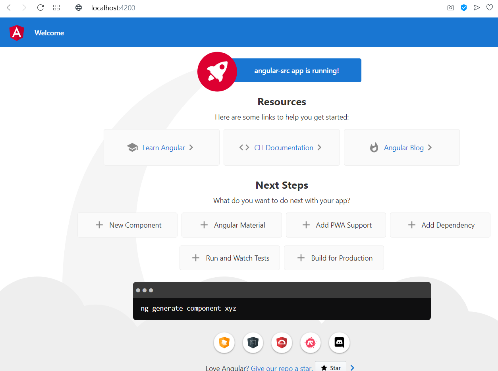
Implementation of passport configurations

Posting user login parameters for authentication.

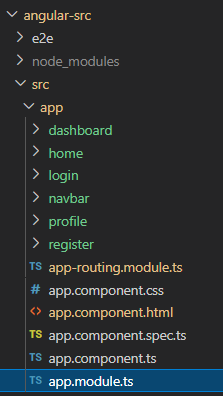
* Next we install angular-cli if haven’t done already using *npm i -G angular-cli command.*



* Using *ng new* {name of angular folder} 🡪
* **Once the Angular project folder is created, we can use ”*ng serve” to run the client application and access it on port 4200.*



10.5.2021 Further development of final exercise.

* Angular exercise poject consists of the following components which are created using ”*ng generate component {comp. Name}”*

*Dashboard*

*Home*

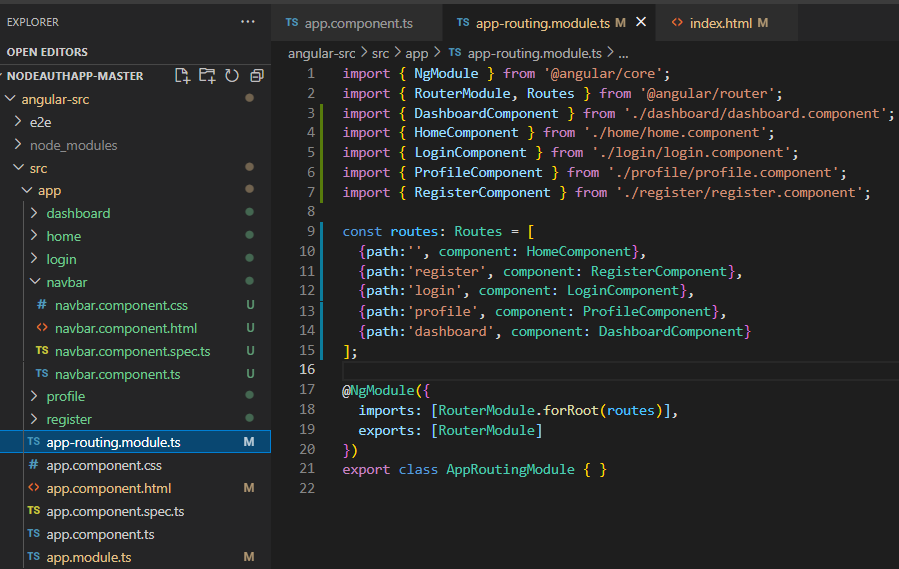
*Login*

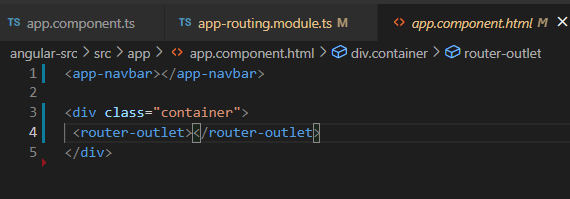
*Navbar*

*Profile*

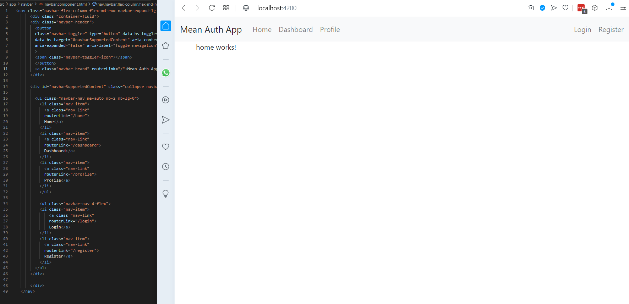
*Register*

*App-routing*

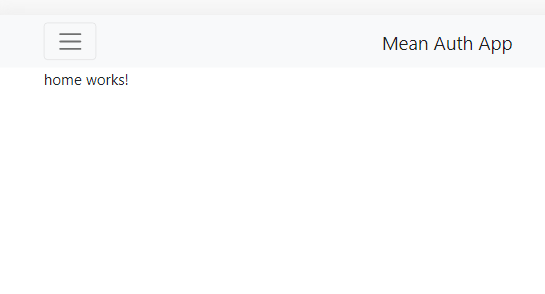
* We will start adding routings and components for each page.
* Routings are defined in *app-routing.module.ts* file.
* An outlet is needed in the app.component.html file to be able to use the configured paths in the angular app.

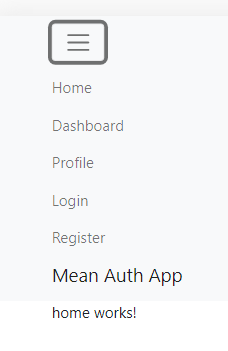


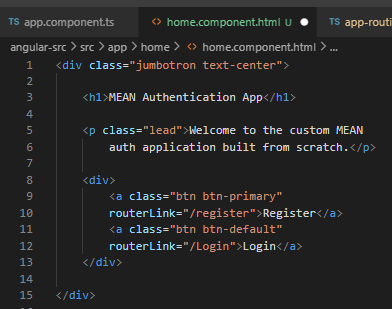
* We are also using app-navbar element in every page, so we include it on top of the template to used in every page in our application.



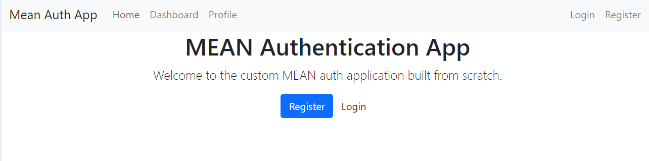
* When starting to add elements to components to build functionalities of each page.

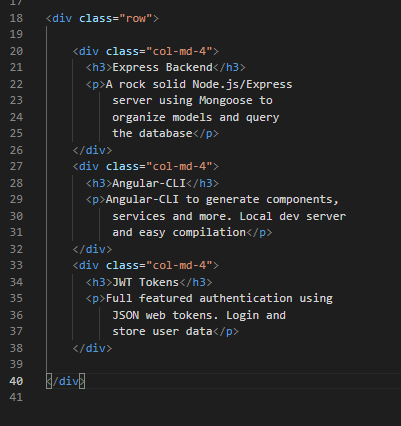


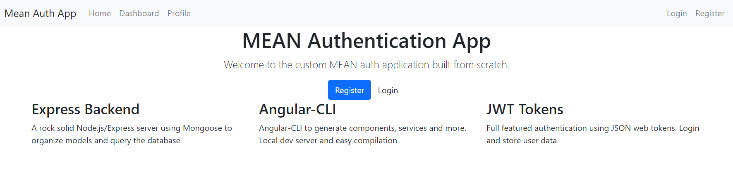


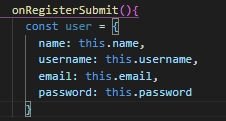
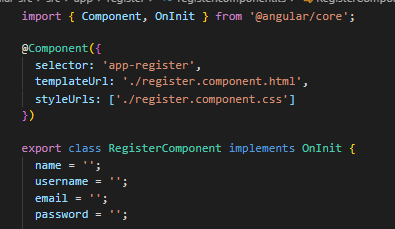


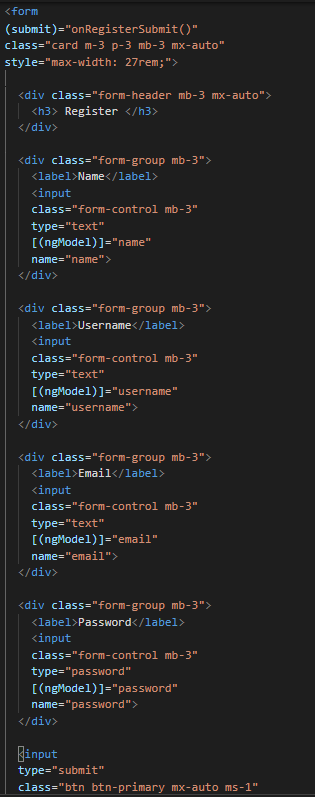
* Next, we add elements of homepage by adding login and register buttons.



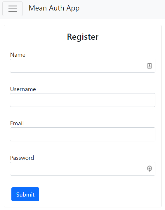
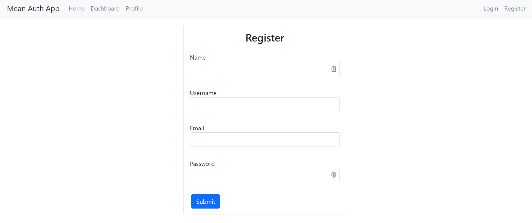
* Furthermore, we add a row with 3 columns to home page.

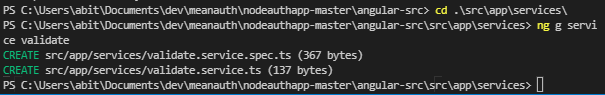


* Next, we add the register form to the relevant component and define its fields.



* Registration form in register.component.html file 🡪



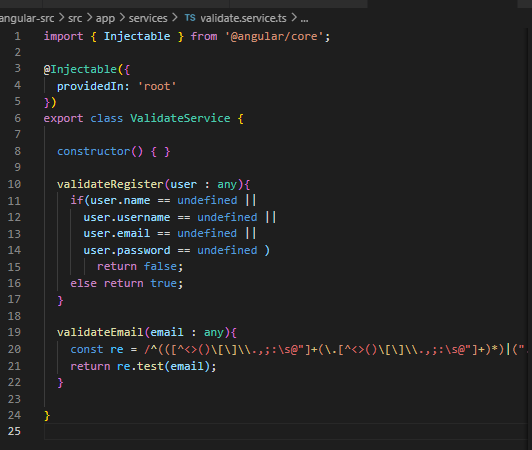
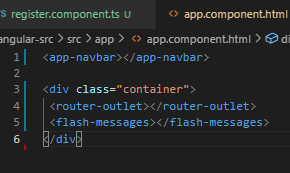
* Next, we generate service validation using “*ng g service validate”*.



* In we add the following lines.

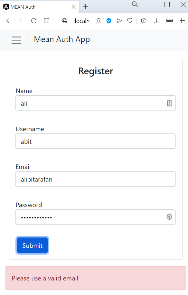


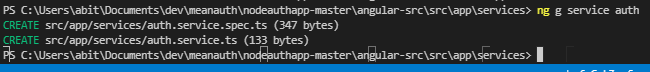
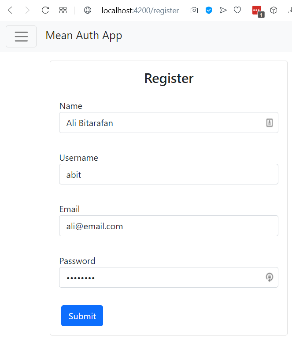


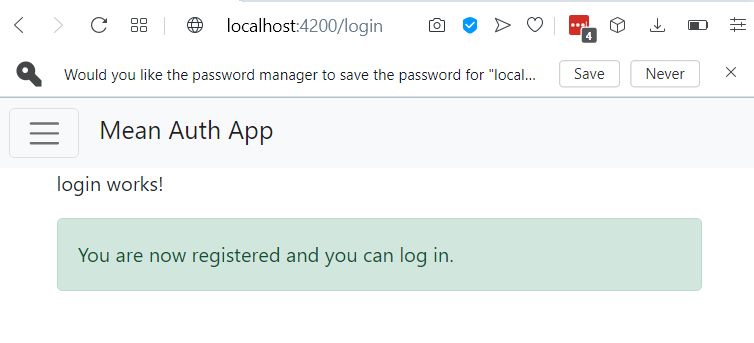
* In *validate.service.ts* we add the validation checks for email and availability of all the fields.
* We are also required to add the flash-messages to the app.component.html to enable flash messages output to the client screen.

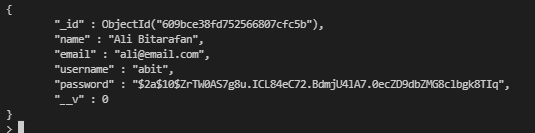
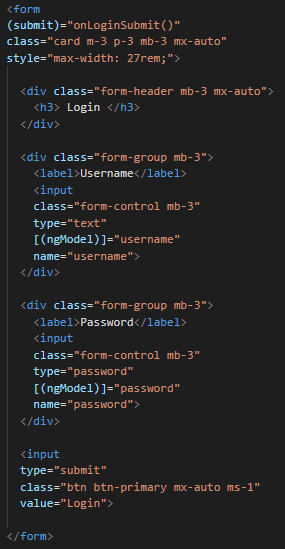
12.05.2021

Continuing coursework exercises with user registration service.



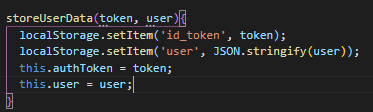
* As result we have the following validation checks and messages 🡪
* Next, we add the auth service by running *“ng g service auth”* inside service folder.
* To authenticate against a backend system, we need to import the HttpClient and HttpHeaders into the *auth.service.ts* file.
* AuthService has a HttpClient passed into its constructor which is used to send user information to the backend for registration.
* Then, we add authservice to *app.module.ts* the same way we did for validate service.
* We define auth service in *register.component.ts* file.

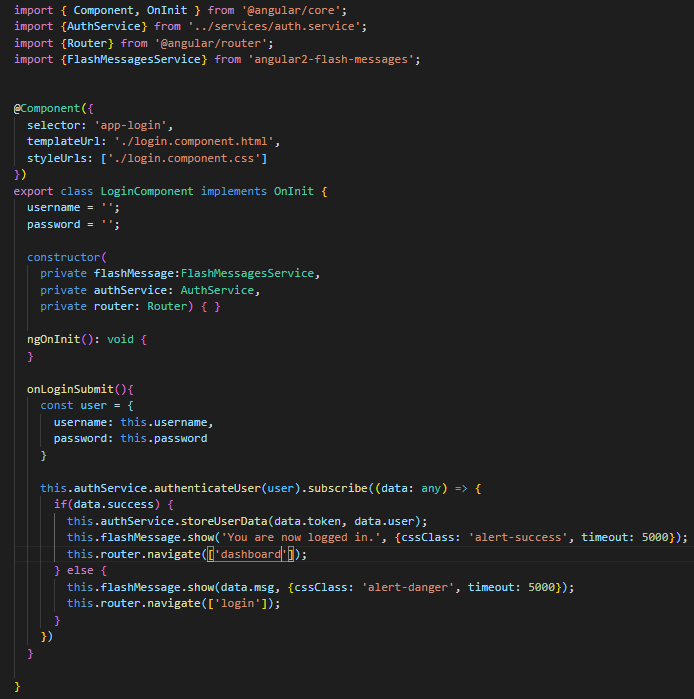


* Next, we check our db to see if the newly created user is listed there.
* Next, we add login components.

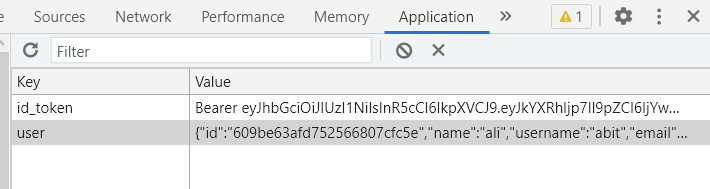
In *login.component.html* file we add the login form 🡪

* In the *login.component.ts* file we import the required packages and add login functionalities to send the user data to backend to against the users in the database.

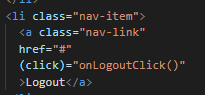


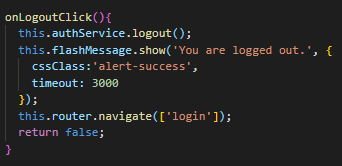


login.component.html

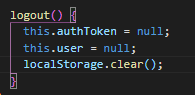


login.component.ts file

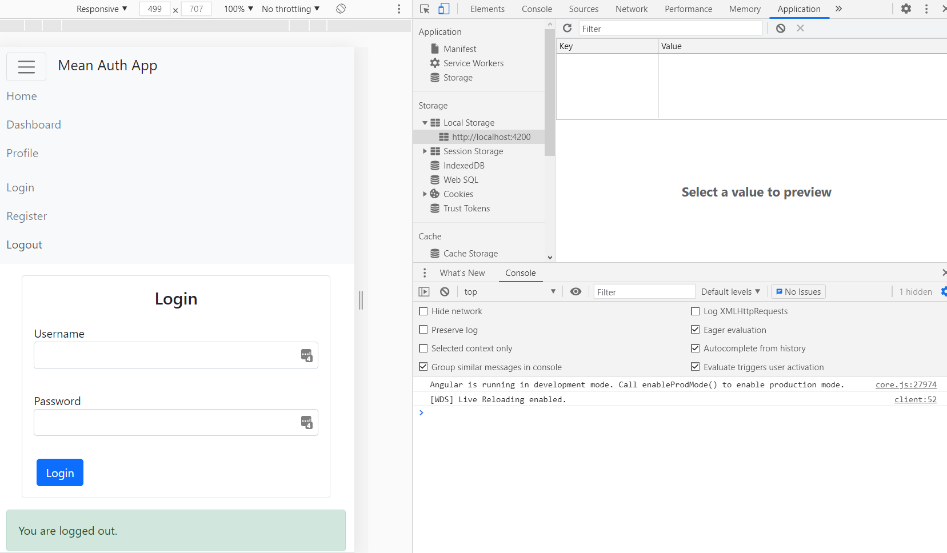
* Next, we implement logout functionality by adding a logout button to navbar and defining a logout function in the auth service to remove user information and token from local storage.

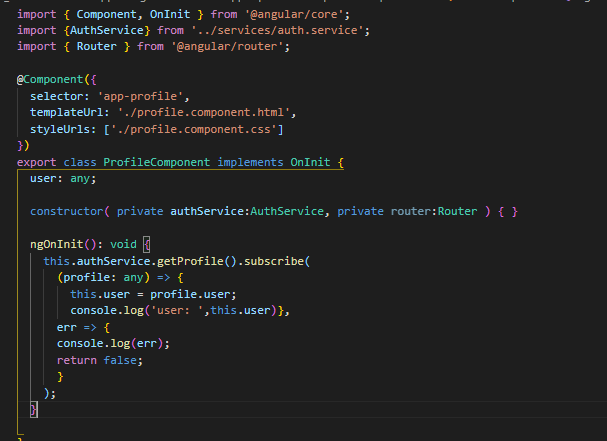
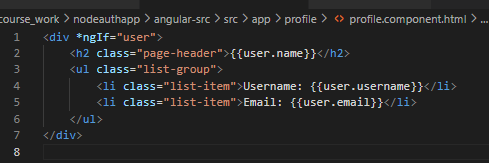


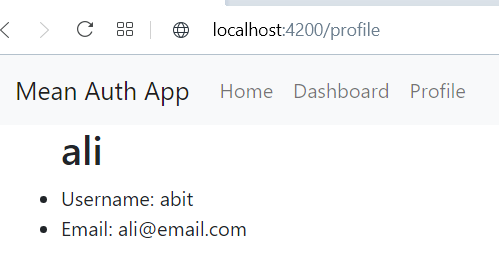
’navbar.component.ts file



auth.service.ts

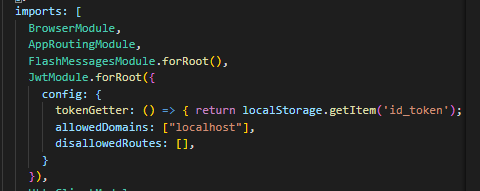
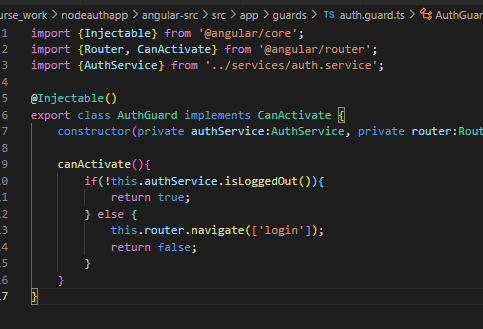


* Implementing profile page where user needs to have authorized access requires checking token against the backend. To achieve this, we implement the following methods in *auth.service.ts* file 🡪
* *profile.component.ts* takes careof calling the *getProfile* function on page load using the following code 🡪
* To demonstrate the functionality, we simply show username and email on the profile page.



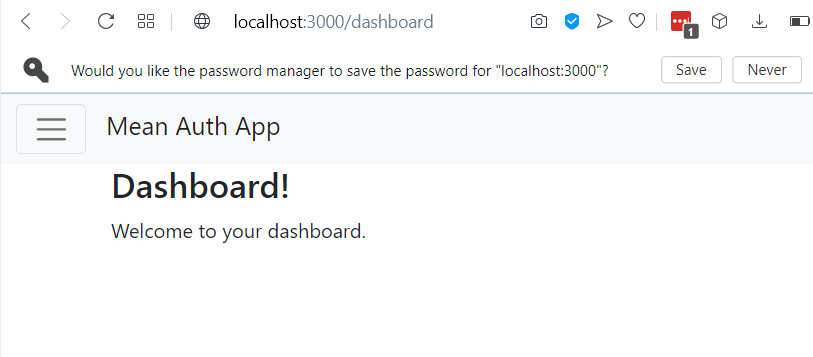
13.5.2021

Continuing coursework with implementation protected requests and authorized access to components.

* First, we add the *JwtModule* to *app.module.ts* file and configure it like the screenshot on the right 🡪
* We are going to use the following package and implement the isTokenExpired function from it.
* Then we write the following code in auth.service.ts file in order to check for the token’s expiration status.
* After that we can simply check for the user status before showing the navbar links by using the following line of code 🡪
* Next, we are going to implement auth guard for safeguarding URLs, so the users are not able to route using typed in url if they are not logged in.

auth.guard.ts file

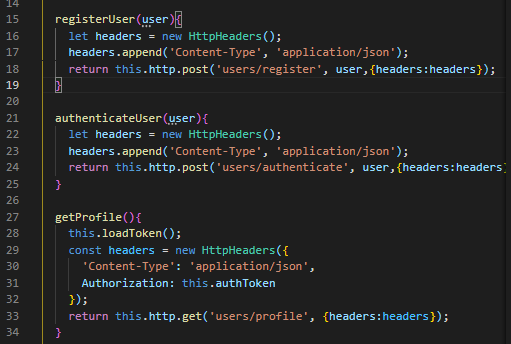
* Next, we import AuthGuard into *app-routing.module.ts* file.
* We can then add the following parameters to the router paths.
* Finally, we run ng build. Then we can access our application from through port 3000 which has express server.



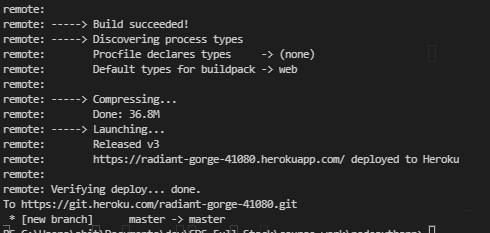
* Next step is deploying the auth app to Heroku. For that purpose, we implement the following changes.



In the app.js file we set the Port as shown in the following screenshot 🡪

Next change is to take the localhost addresses from the *auth.service.ts* file since Heroku will have a different domain for hosting our app 🡪

Then we run ng build once again to bring the changes to the build folder (public).

Lastly, we push the authentication app to Heroku. This version of the app is accessible through this link

<https://radiant-gorge-41080.herokuapp.com/>

* The database is migrated to mongodb atlas.