LEARNING DIARY

Full-Stack 2021-22 Module

May 2022

* * *

Noëline MIGEON
Student number : 000739834

Table of Contents

1 INTRODUCTION

I divided my learning diary into 6 parts, corresponding to the 5 tutorials of the course and my project. Before starting the first tutorial, I checked the general information and understood the main focus of the course. I chose the *Full Stack* module because I wanted to be able to create a project from A to Z. I refreshed my mind about the basic git commands and chose **VS Code** as my editor for this course.

2 NODE.JS TUTORIAL (07/04/2022)

This first tutorial enabled me to have a solid basis on Node.js. Below are my notes of what I thought was important about the tutorial.

- Node.js is a Runtime (not a language, nor a framework), we write in JavaScript
- It works on a single thread using non-blocking I/O calls, asynchronous
- Can support tens of thousands of concurrent connections

◊ Initialization

npm init creates the package.json file which contains all the information we need about the project.

◊ Module installation

To install a module, the command is npm install <module> (ex: npm install uuid). It will only install the module on the project folder. To install it globally, add the -g option. To install as a dev dependency, npm install —save-dev <module> or npm install -D <module>.

Once a module is installed, the folder node_modules is created and contains everything we need to make the module work. In the package.json file, the dependencies are also installed.

P.S : If the project needs to be moved somewhere else one day, we don't have to include the nodes_modules. The node_modules folder can be deleted, and if we want it back, we just have to type npm install which will recreate the node_modules folder and install all the dependencies that are in the package.json file.

The file package-lock.json tracks all of our dependencies with the versions.

➔ nodemon: package we need not to have to restart our server every time a modification is made.

◊ Run a file

To run a file : node <file> (ex : node index.js or node index (without the .js).
We can run any JavaScript file we want from the terminal.

• Main points learned

During this tutorial, **I learned about the most important core modules** :

- event
- fs
- http
- os
- path
- url

To import one of the core modules (ex: http), write const http = require('http').

I also learned how to create a server using node.js (without express) as well as to deploy it to Heroku. This first tutorial did not pose problems.

Here is my first server, deployed to Heroku : https://serene-harbor-56548.herokuapp.com.

Notes: The server is written in the index.js file.
Once deployed to Heroku, to open the app directly in the command line, type heroku open.

3 MONGO DB (25/04/2022)

The second tutorial taught me the basics of MongoDB. Below are my notes of the tutorial.

- NoSQL stands for not only SQL
- In MongoDB, we store data in collections of documents (which are very similar to JavaScript objects)
- MongoDB is very flexible → freedom for the structure of data (not a relational database)

I had some trouble opening the mongo shell. For some reason, the commands given in the video and on the MongoDB documentation were not enough. After some research to solve my problem, I found a solution. I need to first run *mongod* in a terminal using line (1) and then open a new terminal and run mongo (2) to open the mongo shell. I am on MacOS so it may be why I had some trouble.

(1) mongod --dbpath ~/db/data
(2) mongo

The contents of the tutorial were:
- installation of mongoDB
- commands
- introduction to compass : GUI

I learned the basic commands to navigate through the db, create collections, get specific items, modify them, etc. I listed below the commands I thought worthwhile. I also created a separate document with concrete examples of how to use the commands (which I did not consider important to add to this learning diary). At the end of the tutorial, I set up Compass, which allows showing everything in a graphical interface. Except from struggling to open the mongo shell at the beginning, the rest of the tutorial did not address me any issue.

Commands:
- show dbs **: shows databases, won't show an existing database if it's empty**
- use <db> **: switch to specified <db> or create it if it doesn't exist**
- show collections **: shows collections**
- db.<collection>.find().pretty() **: show the content of the <collection>**
- db.dropDatabase() : drops database we are on
- db : tells on which db we are
- db.createCollection('<collection_name>')
- db.posts.insert({ <> }) : inserts one document/piece of data/row
- db.posts.insertMany([{first one},{second},{third}])
- db.posts.find().pretty() : displays the data (pretty makes it more readable)
- db.posts.find({category: `News`})
- db.posts.find().sort({title:1}).pretty() : sorts in alphabetical order for title (-1 descending order)
- db.posts.find({category: `News`}).count()
- db.posts.find().limit(2) : only displays two records

- db.posts.find().forEach(function(doc) { print(`Blog Post: ` + doc.title)})
- db.posts.findOne({category:`News`}) : only returns the first row that has `News`as a category
- db.posts.update({filter},{update}) : will update the whole data, upsert option allows to create the data if it doesn't exist. To just modify the data and not overwrite it, add the $set operator. $inc operator allow for incrementing a value. $rename enables to rename a category.
- db.posts.remove({ title: `Post Four`}) : deletes the row that has 'Post Four' as a title
- db.posts.find({views: {$gt:6}}).pretty() : will find the rows that have a views parameter greater than 6 ($gt : greater than, $gte : greater than or equal, $lt : less than, $lte : less than or equal)
- exit : exits the mongo shell

How to do a text search
- db.posts.createIndex({title: 'text' })
- db.posts.find({ $text: { $search:"\"Post O\"" } }) : will search « Post O » in the title categories → will display « Post One »
- db.posts.find({ $text: { $search:"\"Post T\"" } }) : will search « Post T » in the title categories → will display « Post Three » and « Post Two »

4 EXPRESS (27/04/2022)

In this tutorial, I learned what is Express and why we use it (because it is much simpler than setting up a server only with Node.js). I learned the basic syntax of Express (how to set up the server), the basic route handling, middleware, how to create routes, the nodemon package (that I already knew thanks to the first tutorial), rendering HTML, the REST API, the express Router, how to create/update/delete data.

I used Postman for the first time along with this tutorial. This tutorial did not pose me with big problems: if I had an issue, I looked first at the Moodle Page comments and in the comments of the Youtube video, or StackOverflow if I couldn't find the answer. Below are my personal notes.

- Middleware functions are functions that have access to the **request** and **response** object.
- Postman : HTTP client to make requests to a server. Postman can make GET, PUT, POST, DELETE requests.
- npm init -y : allows to create package.json file without answering any question
- res has a method called send() that sends something to the browser (text, whatever)
- When we are visiting a page, it is a get request on /. Slash is the route for the index page. It can't find a route handler for this /. Routes/endpoints.
- npm i -D nodemon : we install nodemon as a dev dependency because it is only for development, we're not using it for production
- The data from POST request is in the request object
- 400 = bad request

5 ANGULAR (08/05/2022)

The Angular tutorial was more complicated to understand than the previous tutorials. As the whole code was already given, I did not have to "write code" a lot, so I focused on understanding how Angular worked. I learned the basics of the Angular framework and the angular CLI : how to create an angular project, components, services. I also learned how components worked (link between HTML file and .ts file) as well as services. I also learned about events, HTTP and routing. As everything was already given in the tutorial (and the tutorial was the Angular documentation), I did not encounter specific problems (no problems of versions for example). However, I struggled to understand everything and I really completely understood Angular during the MEAN tutorial.

Notes:
- ng new <project-name> : creates the angular workspace
- ng serve —open : opens the application
- ng generate component <component-name> (ex: ng generate component hero-detail): the command creates a directory src/app/hero-detail. Inside that directory four files are generated:
    - A CSS file for the component styles.
    - An HTML file for the component template.
    - A TypeScript file with a component class named HeroDetailComponent.
    - A test file for the HeroDetailComponent class.

The command also adds the HeroDetailComponent as a declaration in the @NgModule decorator of the src/app/app.module.ts file.

- ! Every component must be declared in exactly one NgModule.

6 MEAN (10-19/05/2022)

The last tutorial combined all the previous tutorials. The goal of the tutorial was to build a backend express server and an API to let users register, log in, and get a JSON web token when they authenticate. Once they are authenticated, they should be able to log in and see the dashboard and their profile (+any other routes we want to protect with authentication).

The tutorial is divided in 10 videos:

1) Introduction

Nothing to report.

2) Express Setup & Routes

The first video focused on creating the express server. I followed the video and got no problem. I already learned how to create the endpoints in the express tutorial so it was quite easy to understand. I also created the database “meanauth” in MongoDB.

Note: Enabling CORS (Cross Origin & Resource Sharing) : we need to enable CORS because our backend server and our frontend angular app will be on different ports (enabling CORS is quite simple : it is just adding the module at the beginning of the file + using the middleware, 2 lines in total).

➔ GitHub commit: Express Server (10/05/2022)

3) Registration system & password encryption

In this tutorial, I created the User model and the registration system with password encryption. I created the mongoose model User and encapsulated all the DB functionalities within that module file so that everything is centralized into that file (user.js). I also implemented password encryption so that when users register, their password is encrypted. This part did not pose specific issue.

➔ GitHub commit: Registration System + Password Encryption (10/05/2022)

4) API Authentication & Token

In this video, I implemented passport which handles the whole token process along with the JSON web token module. I learned how to authenticate via an API (by comparing passwords) and protect a route

(authorization by token via passport). As stated in the Moodle page, I replaced ExtractJwt.fromAuthHeader() with ExtractJwt.fromAuthHeaderWithScheme("jwt") and got no problem.

➔ GitHub commit: API Authentication + Token (11/05/2022)

5) Angular Components & Routes

In this video, I created the angular workspace and generated all of the components and implemented the router. I also learned about Bootstrap and how to use it within HTML (for the navbar). Despite the recommendation on the Moodle page, I decided not to install the old angular-cli used in the tutorial and preferred to keep my project "updated" to today's technologies. From this video, I worked a lot with the comments below the Youtube video.

➔ GitHub commit: Component & Routes (12/05/2022)

6) Register Component, Validation & Flash Messages

In this video, I created the register form component, created the validation service, and implemented the Angular2 flash messages module. I had a problem with VS Code so I had to change the option "noImplicitReturns" to false in my tsconfig.json file (solution found on StackOverflow).

! Note: Every time we use a service in a component, we need to inject it into the constructor.

➔ GitHub commit: Register Component, Validation & Flash Messages (13/05/2022)

7) Auth Service & User registration

The goal of this video was to create the authentication service and the user registration functionality (using the backend API /users/register endpoint). I had some problems at that point.

First of all, '@angular/http' has now been replaced by '@angular/common/http'. This first issue was quickly solved thanks to the comments below the video. Following this issue, the imports "Http" and "Headers" are now "HttpClient" and "HttpHeaders" so I also had to make the change.

These first changes were not too complicated as I also did the Angular tutorial which used the last version of angular, so I already knew about the existence of HttpClient and HttpHeaders.

In the register component, I had to change the way the function worked by declaring a dataRegister variable to retrieve the data received by the HTML form. I found this solution by myself (doing logs). I also had to delete the .pipe().map() function in the auth.service. I found the solution from the comment of *Nd Haniff* below the video.

➔ GitHub commit: Auth Service & User (15/05/2022)

8) Login & Logout

The goal of this video was to be able to log in (and log out). To do that, we want to take the user object (that we fill in the login form) and submit it (through our auth service) to the backend authenticate route.

In this video, I learned how to fetch tokens and then store them in local storage. I created the login component and authenticate request that gives us a JSON web token and user data that we can then

store in local storage. I also created the logout functionality. I did not have particular problems for this video.

Notes:

- ngModel : links the HTML to the component → we can then access the data thanks to this.data
- localStorage can only store strings, it can't store objects so we have to stringify them

➔ GitHub commit: Login & Logout (15/05/2022)

9) Protected Requests & Auth Guard

The first goal of the video was to be able to access the /profile route only if loggedIn. I first had some problems to access the route (I had Unauthorized despite the token). After some research in the comments (comment from *First Order*), I changed how the headers are declared (instead of appending twice the headers).

New headers:

```
let headers = new HttpHeaders({
    'Content-Type':'application/json',
    'Authorization': this.authToken
  });
```

➔ GitHub commit: Profile (15/05/2022)

The second part of the video focused on creating the service function to make requests to protected routes and adding the authentication guard to protect pages. I also had some problems with this part but found the solutions in the comments. From 12:38 of the video, I followed the instructions of the comment from Sergio Zurita:

« As of May 2020, 'angular2-jwt' is now '@auth0/angular-jwt' and tokenNotExpired() has been replaced with isTokenExpired(). However, there are still more steps to get involved to get it working.
[…] ». Here is what I changed:

- install @auth0/angular-jwt instead of angular2-jwt

- import { JwtHelperService } from '@auth0/angular-jwt'; in auth.service.ts file
- do not pass it into constructor
- modify loggedIn function according to the comment's instructions
- change authService to public in navbar.component.ts
- switch around the "!" symbol when calling authService.loggedIn()

I followed the instructions and everything worked well.

I also encountered a last problem for the ng build. Indeed, in the first video, we had to change the option "outDir" in a certain file "angular-cli.json", but I didn't have this file in my folder so I first changed this outDir option in the wrong file (tsconfig.app.json). At that time, I didn't need it, so it wasn't a problem. However, to make the ng build command, the output directory has to be set correctly and in the correct folder. I did some research and found that the former *angular-cli.json* file was now *angular.json* file and that the « outDir » option became « outputPath » (not that complicated). After that I built the project and everything went well.

➔ GitHub commit: Protected Requests & AuthGuard (19/05/2022)

10) App Deployment to Heroku

The last video's goal was to deploy the app to Heroku. As I had already done that before, I thought the task would be easy.

I first created a new Heroku app (Heroku create) and linked my repository to the app.

However, as I now had many apps in one git repository, I couldn't deploy my app to Heroku (because the express server was already deployed and I had some git conflicts).

I did some research and found this resource: How to Deploy Multiple Apps Under a Single GitHub Repository to Heroku.

I followed the article and created a script to deploy my app to Heroku. In my package.json file, I added a new script:

```
"publishheroku": "cd ../ && git subtree push --prefix mean-auth-app/ mysterious-mountain-06316 main || true"
```

➔ *mysterious-mountain-06316 being the name of my app in Heroku*

To deploy the app to Heroku, I run « npm run publishheroku » in the mean-app folder.

Once deployed, I still had some trouble linking with the database, because the mlab website used in the tutorial is not exactly the same that is used today (MongoDB atlas) but I still managed to make it work after several attempts.

My MEAN app deployed to Heroku can be accessed at this address: https://mysterious-mountain-06316.herokuapp.com

➔ Last GitHub commit for Heroku deployment with everything working on 19/05/2022

7 PROJECT

My project is called « Tasty Recipes » and is based on the last tutorial application. The application allows for :

- account creation (registration)
- login / logout
- access to personal data (via /profile route)

My project is a recipe inventory. The goals are to :

- be able to share recipes (create a recipe that will be seen by everyone)
- see recipes from everyone (including its own recipes)
- see its own recipes
- modify its own recipes
- like recipes

Features:

- New from last tutorial : a user can't register if the username or the email address already exists in the database.
- A recipe is owned by one user. Once a user has created a recipe, only he/she can modify it. In theory, only the owner of the recipe can access the /modifyRecipe route via navigation

(not URL). However, if another user (who is not the owner of the recipe) tries to modify it by accessing directly the route /itemDetail/<recipeId>/modifyRecipe, it will have an error popup message once submitting the new recipe.

- A recipe has a username (owner), a name, ingredients, instructions (recipe), difficulty (from 1 to 5) and time (in minutes). A recipe can be added/modified only if all the fields are filled in and the difficulty is correct (between 1 and 5) and the time is not negative.
- Pre-fill of fields for /modifyRecipe route (the user doesn't need to re-write everything if he/she just want to change the ingredients for example)
- Users can like recipes (though this functionality could be improved, for example, a user shouldn't be able to like its own recipe and users shouldn't be able to like a recipe more than once)

Presentation. In the video, the following points are presented :

- Register (with existing username, and email address)
- Login (wrong and correct password)
- Dashboard with all recipes
- Add a new recipe
- See a specific recipe
- See its own recipes
- Modify Recipe
- Like Recipe

➔ Last GitHub commit for Heroku deployment of the project on 23/05/2022