Lappeenrannan teknillinen yliopisto

School of Business and Management

Software Development Skills

Akseli Aula, 0545267

LEARNING DIARY, FULL-STACK MODULE

**LEARNING DIARY**

**22.6.2021 – NodeJS Module**

I chose the full-stack course as when searching for a summer job, seemed like experience in full-stack development could have been beneficial. I started the course by reading through the course Moodle page, so that I could get a general overview on how to complete the course. At first the module task list seemed a bit confusing, as there were no clear tasks other than those links to different learning materials. From what I understood the coding tasks are inside the videos or other material – I guess I will find that out later.

**23.6.2021 – NodeJS Module**

Today I set up my GitHub repository for the course, there was nothing special as I was already familiar with version control and especially GitHub. I also downloaded VS Code as I did not have that installed on my laptop. I did not add any extensions, as they seemed unnecessary for now.

After that I started getting familiar with the NodeJS-module by watching the start of the introduction video (2019 version). I learned some basic information about Node.js such as that it is a back-end environment used to execute JavaScript on the server, and what it can be used for etc.

At the end I also installed the current version of node (v.14.17.1) even though as stated it might cause some errors later, but I am ready to find alternative ways to solve the errors later.

**6.7.2021 – NodeJS Module**

I continued the first video I had started earlier and started getting more familiar with Node. The first task was to create a json package file, which can be done using “npm init” terminal command. That json package file lists used to describe the project module versions, name, used dependencies and main file among other information. It can be used to install modules (as the needed dependencies are listed in the json) if they are lost are used on a different machine by typing “npm install”.

I also learned about different dependencies, such as uuid and nodemon, which is so called dev dependency. Nodemon is used to save persistence so that the developer does not have to restart the server every time a change of code occurs.

During the video, the instructor introduced some basic modules, such as path, fs, os, url, events and logger provided with some coding examples of how to use them. While following through the coding examples, I tried my best to learn any new syntax used. At first the use of gravis (`) and function arrow (=>) felt weird, but I did some googling and found out when to use them.

When the instructor was about to move to the HTTP module demo, I decided to call it a day and made my first actual code commit to GitHub but before that I decided to install my first VS Code extension called “Prettier” which is used for code formatting.

**7.7.2021 – NodeJS Module**

Today I learned some basic skills on how to deploy a simple http server using Node.js which could be accessed with localhost followed by the port.

I learned how to load different kinds of files to the server, such as html, css or json. However, on my node.js version it did not quite work as expected – the html file contents were on the site as they are in the file. To display the file correctly, I had to use an alternative way to change the “Content-Type” in the header.

I also learned that the “script” part of the package.json file could be altered for example to run it in “dev” mode using nodemon to keep the server running while doing changes to the source code.

In the end of the node.js introduction video when Heroku was introduced I faced some difficulties. As I already had a GitHub repository, I did not want to create a new one for Heroku, so I tried to pair my existing repository with Heroku app. However, as my node.js module files were not in the root of the repository I could not deploy the app as there was some other files such as this diary which caused errors. Finally, after some time googling, I found a way to run apps from subdirectories using custom Heroku buildpacks.

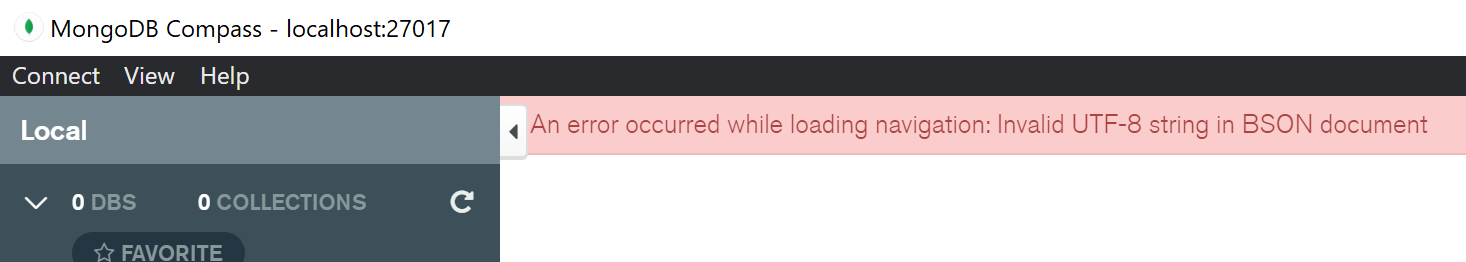
For that I used this guide: <https://stackoverflow.com/a/53221996>

After that I had finished the NodeJS Module.

**8.7.2021 – MongoDb**

I started the mongoDb module. I learned that mongoDb is a NoSQL database, which means that there is no need to map out any data structures before setting up the database, i.e., they store data differently than relational databases.

In the beginning of the video after installing mongoDb community server I came across some problems related to mongoDb compass, which I could not solve even after reinstallations and googling. However, the mongoDb itself was working as intended, so I continued with the introduction video as compass is just a more user-friendly alternative to the command line version.



The error in the upper right corner, I still could create dbs, but they were not displayed in the compass. On the other hand, show dbs command displayed all of them.

**9.7.2021 – MongoDb**

The main thing I learned during the end of this module was the basic usage of mongoDb:

**Insertions:** I learned that one could insert datatypes such as arrays, embedded objects, or even embedded documents. Also, there can be rows that do not have all the same attributes.

**Queries:** I found out that the queries work similar to SQL. However, the video did not cover how to use “find” command to find rows with arrays or embedded objects.



I found out that the following commands can be used for that (“value” is replaced with actual value):

“db.posts.find({tags: “value”})”

and

“db.posts.find({user: {name: “value”, status: “value”}})

For finding from embedded documents $elemMatch could be used:

db.posts.find({comments: {$elemMatch: { user: ‘Mary Williams’ }}})

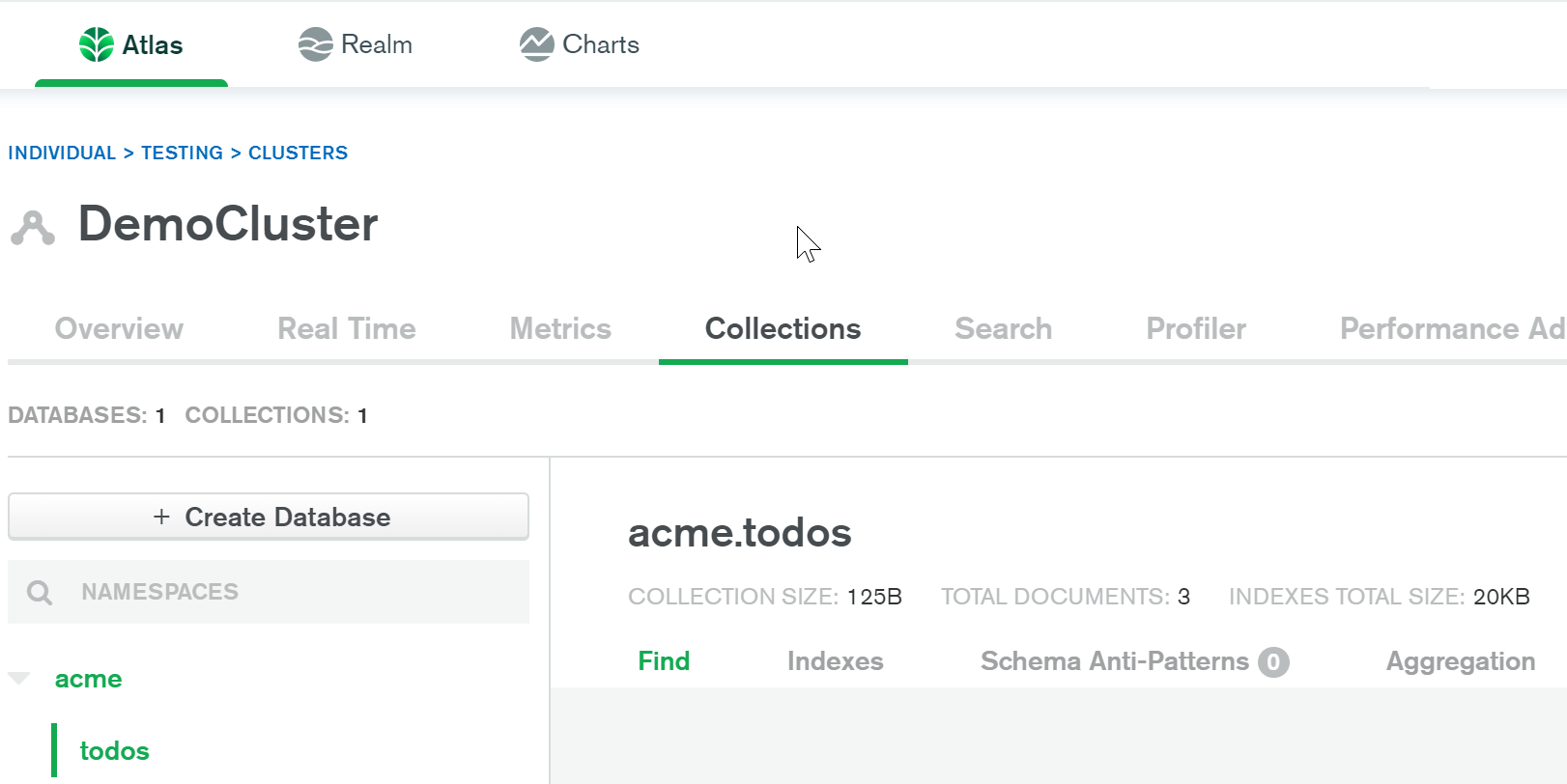
For searching rows (after indexing, finds posts starting with “Post O…” – Similar to SQL command …like “Post O%”):

db.posts.find({ $text: { $search: “\”Post O\””} })

Other query commands were pretty similar to SQL ones (sort, count, findOne) except the “forEach()” function, which could be used for example to concatenate the results with strings.

**Updates:** I learned that generally the update function alters the row so that it only has those values that are updated. To update only certain attributes “$set” must be used, so that the other attributes are not removed.

At the end of this introduction video, I also created a mongoDb atlas cluster database and collection and populated it with some rows.



End of MongoDb Module.

**12.7.2021 – ExpressJS**

In the beginning of the video, I learned that ExpressJS is a back-end web framework for Node. It can be used with various front-end frameworks to create full-stack applications. Also, the video introduced postman, which was later used during the module. For example, Postman can be used for creating http requests and to examine the results.

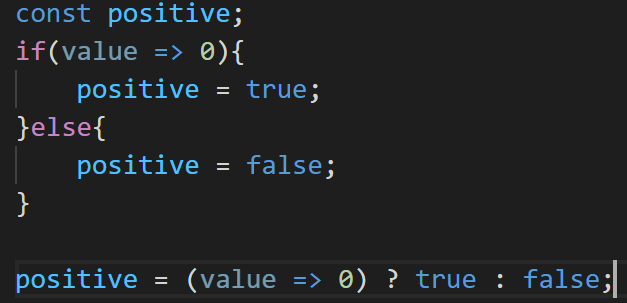
The coding part started the same way as in NodeJS module, first the project was initialized using “npm init” and later express and nodemon modules were installed using “npm install” command. In the beginning the coding examples were just basically redoing the tasks done in nodeJS module, but with expressJS which made the tasks much easier. For example, with expressJS there is no need to mess with the http header manually (at least not yet), also the file loading can be done easily with static folders so that for example when “<http://url/site.html>” is entered, the “site.html” file is loaded automatically.

I also learned to create middleware and routers. Middleware can be used to execute tasks automatically for example, when a request is made. Router can be used to execute different middleware or routes, when a particular condition is met (request type, url, etc.).

**13.7.2021 – ExpressJS**

Today I learned more about expressJS framework. During the introduction video I learned to do different http requests, such as GET, POST and DELETE. Postman application was a handy tool for creating appropriate requests to test the newly written code for handling those requests. I used those requests to create, update and delete members, however the changes obviously did not affect the members json that was hardcoded.

I also learned to use the ternary operator (? :), which can be used to shorten conditional statements. Example usage, if-else replaced with ternary operator:



In the end of this video, I learned to use a template engine to load static html files to the website, which are populated with values. That is when it got a bit confusing, as I do not really have experiences of html. I followed through the video and tried to learn some more. At least now I know how to iterate using <ul>, <li> and #each tags and do some basic html.

End of ExpressJS Module.

**16.7.2021 – Angular**

I started the module by getting some basic information about angular, such as what is it and what is it used for. Angular is a front-end framework used for building web applications and it uses typescript.

Next I started the angular.io tour of heroes tutorial introduction, from which I learned how to set up the development environment (installed the CLI, initial workspace and a running sample app to check that everything is working). After creating a tour of heroes application skeleton, I typed “code . && ng open --serve”, which opened up vs code and the compiled app in browser. To my understanding the command ng serve is similar to nodemon, so that there is no need to restart the server between changes. Later I only did some tweaks to the html, css and ts files to change the appearance.

**19.7.2021 – Angular**

Today I finished parts 1-4 of the tour of heroes tutorial.

In the **1st part** I learned to create components using the angular CLI. I was using VS code terminal (CMD instead of PowerShell) for the angular commands. To create components a command “ng generate component component-name” is used. When generating new components, the CLI automatically declares that newly generated component to the app.module file. I learned to display different components by adding the element selectors to template files (e.g. app.component). There are also some angular built-in pipes, which can be used to format the displayed data (for example. | uppercase). Lastly, I learned to use angular two-way data binding, so that data is linked in a way that if one is modified so is the other. [(ngModel)] is used for two-way data binding.

In the **2nd part** i learned to iterate a list of objects using \*ngFor to display list objects on the html page. I also learned to style individual objects using CSS files. After displaying the heroes, it was time to implement click event handlers, which was pretty easy but didn’t quite work as expected as “name” was undefined. Turned out that \*ngIf can be used for conditional statements in html, and in this case, it fixed the problem.

In the **3rd part** I learned to separate a component into subcomponents. This can make the application easier to maintain as tasks and workflows are separated to subcomponents. To display data from the external component, @input() decorator must be used.

In the **4th part** I learned to add services by using command “ng generate service name”. Services are injected to the component constructors. Angular’s HttpClient returns RxJS observable data, in which case “of()” function is used.

**20.7.2021 – Angular**

In the **5th part** first, I learned to generate new modules using the CLI, with generated imports to app.module file and without generating a new folder in the src/app folder by using flags “--module=app --flat".

I learned to use Angular router to navigate between components and to display the components by implementing <router-outlet> to the app.component.html file. I also learned to create “routerLink” anchor elements for navigation between the components. Angulars’ “ActivatedRoute” can be used for extracting the hero id from the component that is loaded (when a hero is clicked). Also, for navigating between the pages “Location” service can be used.

During the **final part** (6th) I learned to fetch data using HTTP requests, to create new heroes, edit and delete them over http (so that the changes are persistent) and to search heroes by their names.

For simulating the HTTP requests “HttpClientModule” was used together with “In-memory Web API”-module, which mimics a server by altering in-memory data store instead of a server and returns http responses to the “HttpClientModule”. For that purpose, I learned to set up the environment for the above mentioned.

For getting the heroes from the in-memory data storage using Http GET request is made. For example, when getting an individual hero (id=1) the following line of code can be used “this.http.get<Hero>(‘api/heroes/1’)” which returns an Observable variable.

For updating, creating and deleting heroes, I needed to alter the html files of the responsible components by creating buttons, input fields etc. The requests itself were done similar to the GET request, but the request type was changed and also delete, post, put requests needed a second parameter – the http options in which the header content type was set to application/json.

Lastly I learned to implement a search function, which did “live” searching so that when a term is typed the data is searched at certain intervals.

For all of the above mentioned there was also error handling for any errors.

After finishing the introduction, I did some minor tweaks to the site appearance by editing the css files. So far this was the most challenging module, but after all I think that this was very educational.

End of Angular module.

**22.7.2021 – MEAN Stack project**

Today I started working towards the course project, which is a MEAN stack app consisting of the previously learned frameworks (MongoDb, ExpressJS, Angular, NodeJS). The initial application is going to be built according to the MEAN Stack project video series, but the application will be expanded and refined further.

I started by watching the two first MEAN stack tutorial videos. The first video mostly covered the basic information of the project, such as used frameworks etc. The following technologies will be used:

|  |  |
| --- | --- |
| **Back-end** | **Front-end** |
| NodeJs, ExpressJs (Restful API) | Angular 2 / Angular CLI |
| passportJs, JWT (Authentication methods) | Angular Router Module |
| CORS (Cross origin resource sharing, This is needed as front-end, and back-end are on different ports) | Angular2-JWT |
| Mongoose ODM (Modeling the app data) | Auth Guard |
|  | Angular Flash message module |
|  | Heroku |

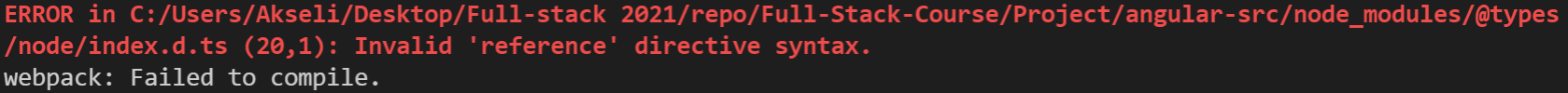
During the second video, I mostly set up the environment, initialized the application and its dependencies manually and using npm commands. I first used body-parser module as it was used in the video, but after some research turned out that express has body-parses as a built-in module from version 4.16+, so I uninstalled body-parser module using npm. I also learned to connect mongoose database to the application and to create a config file for that purpose.

**23.7.2021 – MEAN Stack project**

In the part 3 I created a user schema for the database which I had set up yesterday. I also created my first successful database “row”, I was surprised of how easy it is to use mongoDb together with those other frameworks especially compared to SQL for example. I also learned to use bcrypt for hashing passwords with salt. I have previous experiences of using sha256+salt and compared to that, bcrypt seemed easier to use, though I am not sure about the security aspect.

Part 4 mostly covered the authentication from which I learned the usage of passport and JsonWebToken modules. There were some errors in some lines of code, which I had to fix. For example, I had to examine the JWT payload used in the passport module used for authentication, as it was not working. I also noticed some deprecation warnings concerning mongo (url parser and server discover & monitoring engine), which I got rid of by altering the mongoose.connect() parameters.

I began the part 5 by reinstalling Angular-CLI as guided in Moodle.

After creating the angular skeleton app, I tried running it, but It couldn’t compile:  


I got that fixed by following this [stackoverflow](https://stackoverflow.com/a/63977287) answer.

This part was mostly creating components and setting up routes for them. During this part I learned more about the usage of bootstrap themes and how to implement a navbar. Turned out I couldn’t use newer than bootstrap 3 version “themes” for this project as they wouldn’t display correctly.

During the sixth part I created the layout for registration component, consisting of input fields for different user attributes and a submit button. During that I used ngModel to map out the fields, so that they can be used on the typescript side (this.email for example). In addition to that I created a validation service, which checks that all fields are filled, and that email field is valid. Email is checked with the help of test()-function and regular expressions. In case the required fields are not filled, or email is invalid I learned to use FlashMessagesModule to show popup messages/warnings for example.

I will implement a check for password later so that the password matches the minimum requirements to the validateService.

**26.7.2021 – MEAN stack project**

During part 7 and 8 of the project video series, I learned to “connect” back-end with front-end. It was done by using a service (called “auth” in this case), which sends http requests to the running back-end server. For example, when registering an account, the front-end side creates a user, which is then sent as a http request parameter to the running back-end server (<http://localhost:port/users/register>). When the back-end server receives the request, it completes the request accordingly, in this case creates a new user and adds it to the database. Back-end then returns a status (success: true/false in this case) which can be compared on the front-end side to react accordingly.

By learning to connect back-end and front-end I could create functioning registration and login screens in addition to logout.