Lappeenrannan teknillinen yliopisto
School of Business and Management

Sofware Development Skills

Lauri Pöhö, 000910844

LEARNING DIARY, Full-stack MODULE

LEARNING DIARY

First day

14.4.2022

I read all of the sections carefully and made sure that I understand the structure and the goals of the course. I decided that I would use VS Code as the editor throughout this course because I already had it set up on my system. I revised the core concepts of Git and created a Git repository for the course. After all of that I did some planning on how I will complete the course alongside my high school studies. Throughout the rest of the day some project ideas already came to my mind.

Node.js

16.4.2022 & 17.4.2022

I started following along the Traversy Media's Node.js crash course. I luckily had Node.js 12 installed already, so I didn't have to worry about that. I did the first commits to the repository that I had made earlier. I didn't struggle at all while following the tutorial. It was pretty straightforward for me to understand the concepts that were introduced in the video as it was not my first time with Node.js. However I had forgotten some things so it was a good mind refreshing. I especially found the events module very interesting and thought that it would be useful in the future.

In the end of each day, I pushed everything to Github.

MongoDB

17.4.2022

Today I watched and followed along the MongoDB crash course.

I learned about

- The difference between relational databases and NoSQL databases and the pros and cons of both
- MongoDB fundamentals
- How to install Mongo and MongoDB Compass
- How to use Mongo shell and the Compass GUI
- Shell commands to add, update, delete, sort, search, find and display MongoDB collections.
- How to setup and connect to the MongoDB Atlas cloud solution through applications and shell.

The video didn’t show the installation process for Linux so I had to figure it out by myself but otherwise everything else went pretty smoothly. I was surprised of how many built-in features and commands MongoDB already has to make our life easier. After watching the video, I thought about what my MongoDB schemas would look like in my project application.

Express
18.4.2022

Today I learned a lot. First I installed Postman desktop client and express and started coding while watching the video. I was able to understand everything I wrote and I'm sure that I will be able to apply everything that I learned. I had an error while implementing express-handlebars but I was able to solve it by a Google search. In the end I did some research and decided that I will use JSON Web tokens in my project application.

I learned about

- The concept of Express and how it can be applied in API development and rendering both static and dynamic pages.
- The advantages of Express (unopinionated, fast, minimalist)
- How to test your API with Postman
- The definiton of middlewares and how they can be used
- Simple CRUD with Express
- HTTP status codes, methods, header and body, requests and responses
- Rendering dynamic pages using templates (express-handlebars)

Angular

20.4. - 24.4.2022

I completed the Angular tutorial in the course of 4 days. In the first day I did the parts 1-3 and in the next 3 days I did one part per day. When I started Angular was a completely new concept to me. The tutorial was comprehensive and I learned a lot of things. I also realized how steep learning curve Angular has.

In the first day I faced a problem. Here's an image for reference and the StackOverflow link that helped me to solve it. The problem wasn't about Angular but I'm happy that I faced this problem because it really taught me how powerful tool Google is for programmers.

https://stackoverflow.com/questions/65300153/error-enospc-system-limit-for-number-of-file-watchers-reached-angular

Otherwise everything else went very well. I took my time to read everything profoundly so that I was able to understand everything.

I learned about

Day 1:
- How to setup your local environment for Angular development
- How to generate Angular application structure using the Angular CLI
- Generating components using the Angular CLI (I find this fascinating)
- Displaying components by adding them to another components and finally to the AppComponent
- Data binding with ngModel
- Using *ngFor to display lists and *ngIf to conditionally display HTML
- Property binding and @Input decorator to introduce the parent component properties of a child and then render a component from the parent

Day 2:
- Angular services and injectables
- How to inject services to any component in the application
- Observables and RxJS. This is a topic that I find complicated but I was able to understand it in some level.

Day 3:
- Angular router and how it’s used to navigate between components
- Using the RouterOutlet directive to turn AppComponent into navigation shell
- Configuring the router and defining routes (redirect and parameterized)
- Using the routerLink attribute in elements to create navigation buttons to different routes

Day 4:
- Using HTTP in Angular
- Making a service to load list from a web API
- Added post(), put() and delete() methods to that service
- Adding, editing and deleting data by components using the methods of a service
- Using an in-memory web API in development
- Learned more about observables

MEAN-stack
24.4. - 4.5.2022

I began with the project the same day I finished the angular doc tutorial. I was kind of scared because this tutorial was so old. But after finishing the tutorial I feel like I've learned so much. I could even notice the little mistakes while watching the tutorial and following along. I will go through my development process using Git commits rather than studying days.

init mean app
https://github.com/poh0/lut-fullstack/commit/27e3f224c0bcfa7b7b6c17dd76dbcf87220106fa

I watched the first two videos where the project was introduced and I managed to create a bare-bones Express application with routes, dependencies and connection to local MongoDB. Nothing special yet.

user model & register
https://github.com/poh0/lut-fullstack/commit/716e88c547a71cce92804f430549f5511735fb6a

I learned to create a basic MongoDB schema for users which I'm sure will be used in my project application. The schema was applied to create basic user registration functionality and some other query functions. Also learned to secure passwords with bcryptjs. I also noticed later on that the application will crash if an empty password is sent within the request body, so later on I would implement a check for that.

API authentication
https://github.com/poh0/lut-fullstack/commit/45c4c00d18ada510bc4da4e35ce599810c51fd92

I learned to create authentication functionality to the API with JSON web tokens and Passport. Honestly I didn't really like the way of how the syntax was in the video because of the nested callbacks but I thought that I will not try to play hero just yet. I also faced an error which I was once again able to solve with the help of Google.

Angular components & routes
https://github.com/poh0/lut-fullstack/commit/81de22993a393609786ddfe29f436f0d08a8c0c2

This time I set up the front-end for the project. I almost instantly got an error. It was related to typescript versioning but it took me some time to solve because there was no spoonfed

solution in Google nor in the source code of the tutorial. Everything else was pretty straightforward because I had already learned most of the stuff in the Angular tutorial. I also had to change the Bootstrap version.

Register component & validation
https://github.com/poh0/lut-fullstack/commit/687a72beea1d387e1ed62cd687e18087d001f4d8

I created the registration page with bootstrap and bound the forms to the correct variables in the component class. I learned how one can validate the form data on the front-end but I thought that the same validation should also be done in the back-end. I also learned to use flash messages with Angular to inform the user about errors and successes.

Auth service & registration
https://github.com/poh0/lut-fullstack/commit/18e2978d7ba0a410782d667c0a964d3afec47bc7

This commit achieved the functionality needed for a user registration from the front-end. I learned more about angular services and I think I will use this as some kind of reference when building my future projects.

Login & logout
https://github.com/poh0/lut-fullstack/commit/2c09e776c8d86883d077999e2636d4ce7209622d

Now we added more stuff to the auth service such as authenticating the user through API and storing the returned JWT token in localstorage. Also the design for the login component. Once again, all of this is will come in handy because these are some things that almost every modern website needs. Also no problems so far.

Protected requests & auth guard
https://github.com/poh0/lut-fullstack/commit/ec85276aa4480adcf69bf7ab3e63fa88ccb24650

Today I learned about guards. I think Angular has implemented guards very nicely because It’s not just an on/off switch but you can actually add functionality to the guard itself. I’m also starting to get how all these guards, services and components work together.

ready for deployment
https://github.com/poh0/lut-fullstack/commit/416ed15e384928f14b3278f019c6bbbcccad3114

Just some little changes to the code so that the application can be deployed. I learned to deploy a fullstack mean application. I will use this part of the tutorial as a reference when deploying my future projects.

Project
5.5. - 5.6.2022

Now this part of the course is all I have been waiting for. I already got an idea on what I will start to create. My project will hopefully be a same kind of question-answer forum as StackOverflow, but mine's will be for mathematics. I think I will call it Mathquery. I know that this will be a challenge for me but at the same time I think that I'll be able to make it happen. I already found an Angular module that can help me with rendering LaTex math formulas.

I ended up having some inconsistency with naming of things so while you're reading this you can assume that a **post** is the same as a **question** and a **comment** is the same as an **answer**

I tried to write this so that every bullet point would represent one commit to the project folder.

1. **API**

- I started making the API first because I'm more familiar with backend than frontend even though it could've been more natural to start with the frontend.

- I decided that I will follow a different design pattern than in the coursework project because I thought that the code would be more readable when the functionalities of the routes were defined elsewhere than in the routes directly.

- First I made the database models for an user and a post. It was pretty straightforward because the models were so simple.

- Then I created controllers for authentication and registration routes. I also set up MongoDB locally for this project. The functionality was almost exactly the same as in the earlier project so I just referenced that.

- Then it was time for the JWT authentication. I really liked the passport-jwt solution so I used that but I ran into a problem which I wasn't able to solve right away. However, when I tried to solve it later, I was able to solve the problem relatively quickly by taking a look into the documentation of passport-jwt. The problem was about the jwt_payload in the Passport middleware.

- I knew that I wanted the ability for users to comment other questions. I was wondering if the comments should have their own model referencing a post which they belong to. I was able to find a MongoDB forum post that helped me a lot.

https://www.mongodb.com/community/forums/t/what-is-the-best-schema-for-a-blog-post-for-storing-the-blog-content-like-share-and-comment/131915

- After figuring that out, I created a controller for post creation. I was able to do that without the help of Google, which I was pretty proud of.

- Now I had to create controllers for getting every post in the database. I knew that those posts would be shown on the dashboard of my site so I wanted to exclude the bodies of the posts from the database query. I once again had to google and I found a quick solution from Mongoose documentation.

- I thought that the dashboard would contain links to every question so I had to create a route for getting a question by the id. This time I would also return the body of the question so that in the frontend we would render a page for that specific question.

This ended up being harder than I expected. I would parse the question id from the request parameters. However, the backend would crash every time an id that wasn't a valid MongoDB objectId was delivered. I was able to solve this with a bit of "hacky" solution. I found a regex for MongoDB objectId from StackOverflow.
https://stackoverflow.com/questions/14940660/whats-mongoose-error-cast-to-objectid-failed-for-value-xxx-at-path-id

- Now the backend was almost ready. Only comment creation functionality was remaining. The problem that I had earlier came in handy when implementing this because the route for comment creation contained the id of the question. I just had to tamper the comment model a little bit. Also some changes had to be made to the getPost controller so that it was able to return also the comments of a question. Everything else was pretty straightforward. You can see the code from the commit called "comment creation".

Now the backend is ready at least for now. I know that I will have to make some changes later on.

2. **Angular**

In this section, I encountered more difficulties. However, I am pleased with the outcome. I also think that it was a good idea to use Bootstrap so that I could focus more on Angular rather than CSS.

- Unlike the earlier project, I decided to use the newer version of Angular CLI. I'm kind of scared to do it.

- I began with initializing the project and then I generated some components with the Angular CLI. I took the navbar for this project from the earlier project. I also defined some routes for the components that I generated.

- Then I designed the dashboard page with some mock content. Then it was time to ask the API for the questions to render them on dashboard. I created a service called “QuestionService” that was used from the dashboard component to get all of the questions. Everything to this point had been pretty straightforward. However I had to google a bit to be able to render the creation dates of the questions. For that I used something called “DatePipe”.

- Now it was time to create the question component. Basically, it’s a page for a specific question where the user can for example answer to the question. The route for the component is “question/:id”. When user clicks a question on the dashboard, the user is redirected to that route. The component then calls the QuestionService for the specific question. For this I spent some time looking into the Angular documentation.

- Next challenge was to render the comments of a specific question. I created a separate component for that called “comment-detail”. The question component would pass the comments of the question to that component and render them with *ngFor. This was something I had learned earlier from the Angular documentation tutorial.

- Then I created the registration component and added some validation to it. I didn’t like how the flash messages looked like in the earlier project so I decided to use another module called “ngx-toaster”. I am happy because I got it working on the first try without any problems.

- Now it was time to add the actual functionality for the registration component. For that I created a service called AuthService. I used the earlier Mean project as a reference. The service itself didn’t have many differences but I got a deprecation warning when I implemented the handling of returned data in the component, so I took the challenge of fixing that warning. https://rxjs.dev/deprecations/subscribe-arguments

- Next up was the login component. It came to be almost identical with the register component. I referenced the earlier Mean project to implement token storing functionality in the AuthService. I also had to make a small modification to the API in order for this to work. Also, the logout function and detecting whether the user is logged in was identical with the earlier project.

- Now it was time to implement the ability for users to create their own questions. For this, I created a new component called “question-form”. First I created the user interface for the form. I found out that Bootstrap has a nice design for the textarea element so I used that. Now it was the time to create the first protected request of this project. To the QuestionService I created a postQuestion function. The QuestionService then accesses the AuthService to add the current JWT token to the request headers. This part went surprisingly well because I spent more time thinking about how I’m going to implement this than actually writing the code.

- At this point I realized that every time I accessed a question component through the dashboard, I would see weird errors in the console of my browser. I spent a long time looking into them. Then I realized that Angular was trying to render the component before the API request was ready. This problem was fixed by simply adding an *ngIf to the element.

- Then I created a component for comment creation, which was very similar to the QuestionForm component. One exception being that this component would be rendered in the question component rather than having it's own route. I wanted the component to render the text "You must be logged in to be able to comment" if the user wasn't logged in. For this I found out by googling that one can use an else-statement inside of *ngIf to render another element instead. I also used this in the question component to render "No comments yet" if the question doesn't have any comments.

- Then I had a full week of exams in school. Knowing that the project was now almost done, I took a break from it.

- Today I touched to the project for the first time in a week. I got straight to work and finished the comment-form component and added a method called postComment to the QuestionService. At this point it didn't take me too long to implement this because it was very similar to other things I had created in this project.

- After I remembered that not anyone should have access to asking questions, I created a guard called "AuthGuard" so that you cant access the question-form component without being logged in. I once again referenced the earlier Mean project to achieve this.

- Now we get to the part I had been waiting for: Turning this ordinary question-answer forum into an actual mathematics forum. Before I began making this project, I tested out this module called "ng-katex" which allows you to render LaTex formulas in Angular applications. The reason why I decided to implement this in the end of the project is that because it's so simple. Basically, instead of an HTML paragraph, you wrap your contents in this "ng-katex-paragraph" angular component. You can see from the commit "add katex rendering" how simple the implementation actually was. Also, it looks pretty decent in my opinion.

- Of course the users should be able to see how their LaTex looks as rendered. For this, I implemented previews for question-form and comment-form. This is better shown than explained so you should take a look into the video.

3. **Summary**

- The front-end part ended up being a bigger challenge for me and I spent more time on that than I spent on the back-end.

- Later on I realized that a lot of the stuff would be the same as in the coursework project.

- I learned a lot because I challenged myself to try new things such as fixing that one deprecation warning and using another module for the flash messages.

- I really liked the idea of my project so I may keep updating it in the future.