Lappeenrannan teknillinen yliopisto
School of Engineering Sciences

Software Development Skills Full-Stack, Online course

Kanak Chakma, 001955895

LEARNING DIARY, Full Stack MODULE

LEARNING DIARY

(NODE JS CRASH)

25.03.2024

Previously I had node.js and npm installed. But those were very old versions. I couldn't update the npm because node version was outdated. I installed the latest node by downloading from the website and updated the npm afterwhile. Now everything is in order.

I've learnt how to export and import functions from one file to another file in 'commonjs' type, which is default.

Next, I learned how to export and import in 'ES module' type. Here in need to use .js file extension in order to import properly. Default exporting can be done by using the word default where {} need not be used to import. For other non-default imports {} need to be used.

Started with the server concept. I've learnt how to create a server and send a response to the client. 'Content-Type' is used to specify what type of text/file is to be sent as response. It can be either html, plain or JSON. Also, I've learnt how to write status code for a message using 'writeHead' function or statusCode funtion. After creating the server it needs to execute listen function to wait for the response.

I've learnt how to add NPM scripts to ease the server running using npm.

03.04.2025

I started reviewing what I've done in the previous time. Practiced the concepts like server creaton, handling request responses in the server. Distinguished the methods to be used in case of commonjs and ES modules.

05.04.2025

Learned concepts like gitignore that ignores file names when pushing the code into github. Also learned about .env file that is used to store the secret API keys and some commonly used variables and how to access these variables in the files. Installed nodemon that helps in reloading the server every time I make changes in the server code.

06.04.2025

Previously I heard about postman but never used it. I learned how postman extension is used to create http requests easily. In this way we do not need to use the web browser to check the console or the properties of the webpage. Using postman I've done simple routing using try catch method. Here I did the routing by using the url and the http method that are accessed by request variable of the server, Using the response variable, the output type and status code are updated.

07.04.2025

I practiced the simple routing again to review the codes. This time I created a simple API that sends GET request using url and a POST request to add data in the array of Users that was manually created in the file. To get the specific user with its ID and name regular expression was used. I worked with regexp in my bachelors studies. So it was kind of a recap for me.

08.04.2025

Today I've taken a boost by covering quite a lot of materials.

I started with the concept of middleware and the handler inside the server. A middleware is used inside the server function that can manipulate the request, respnose object of the server. This is also used to write cleaner code and avoid redundancy. The handler also works almost the same way. The diffence is middleware has a next() function that tells what to do next and the handler has no next() function but it executes just like a function does. We send the req and res object when using the handlers and middlewares.

Then I started the file system module. Here I learned how callbacks, fs/promises and async await works. I used readFile() function from all three of these ways to handle asynchronous operation is Javascript. Then for writeFile() and appendFile() I've used only the async await version as this is the most commonly used way. I had some issues here regarding append and readFile() function. After appending the readFile wasn't printing the appended data. But I fixed it using all 3 functions inside another async function and running that.

Started the path module. Here I learned functions to get the base Filename, directory name, file extension name. Also path.parse() function to get the full details of the path. I found some previous concepts of 2 variables __filename and __dirname that are available for commonjs module by default. Also i revised join and resolve method that are used to create a path by adding path values.

I started OS module. Here I've learned userInfo() , totalmem(), freemem() and cpus() functions.

I started url module. Here I learned an URL object returns array of url data. Url.format(urlObj) returns the url in string form. Search parameters can be returned by using URLSearchParams();

I started crypto module. This was an interesting module that provides cryptographic features in node. I learned how to encrypt and decrypt a message using algorithm, key and iv.
Lastly, I did learn about eventEmitter method and process module. Eventemitter is used to emit events as well as registering functions to specific events. Process module is used for some certain tasks like getting memory usage, process ID, title, access environment variables etc.

(EXPRESS CRASH)

10.04.2025
Started the express.js crash course. Its an unopinionated web framework and so we have different ways to do the same thing.
First, I initialized the package.json file with commonjs module. Then I installed express. Created a server. Here we can directly create routes with req.method name. Res.send() , this can parse json object, html, plain text to show it in the console.
Created a new script so that I dont need to start the server again when I make changes. Previously, I've used nodemon, a third party module. Now node has added a new feature to do the same with script.
To get the staticFile in return I've used res.sendFile and inside it I used path.join to create the path. But it has a drawback because we have to send a separate file for each of the page which is messy. To solve this problem I've added a static folder for the server and used it as a middleware. But now in the url I had to use file extension to load the page.

Now I've started to deal with JSON api. To test it I've used postman. Now I dont need to leave the vsCode to testt the APIs. Now I've adde a .env file where i can store my important API keys/ secret keys. I can access it using process.env.VarName. But to access it a little change is required in the package.json file which is "node --watch --env-file=.env server".
Then I created a route to get a single element using the element ID.
Now http://localhost:8000/api/posts?limit=2&sort=desc here ?limit=2 limit the returned element into 2. and sort=desc sorts them in descending order. We can access these values with req.query. In real life we need to be careful with these value handling because someone can eject SQL query and change the database.
To send or change status code write res.status(code).json(rest of the code);
Now there was no error handling till now. So now I started error handling. To send 404 status I checked the ID of the post manually and returned the fuction if it was not found. The return helped to reduce the code.

11.04.2025

I shifted to the es modules from commonjs module because ES module is more user friendly I guess.

I've created a route folder with a file to handle routes inside it. Now the server.js file is cleaner. Inside the router we need to write router.get instead of app.get and router file needs to be exported to use it in the server as a middleware.

Now for the POST request with a body in it we need to have bodyparser middleware to use the postman.

app.use(express.json()); lets to parse json data for post request.

app.use(express.urlencoded({extended: false})); this helps to parse data from URL form body.

I've created a new route for POST request with a body. New post's title can be accessed using req.body.title;

Then I've created a route for PUT request. Here I accessed the id from req.params.id and fetched the post using the find() function.

A route for delete request is done with the same functionality but here I used filter() function to get all posts except the one to be deleted.

I've created a logger middleware to show the method and the URL. To use the middleware in the routes it just needs to be added in the arguments before (req, res).

For using the middleware in all routes the best is to create a separate middleware folder and create middleware files inside it. Then import it and use it in the server.

12.04.2025

Next, I've added Error handling functions by creating a middleware and using it in the server. In this way I've reduced the code and used conventionally used error handling features.

Then I've installed and added colors package to use colors for different request methods.

In this way it is easy to detect the methods that is used.

To further clean the code, I've added a separate file called postController.js to store all the functions of the routes and exporting them to the router's file. This creates a structure of the total code which is easy to navigate through.

19.04.2025

This day I learned how to make request from the front end to the backend. I've created simple button to get the posts and a form to add posts using html and javascript. Here to fetch data

async await function is used. I just revised some of the html contents that I learned in Front end development course. Overall, It was fun to connect the front end and backend using the api.

20.04.2025

I explored ‘EJS’ template engine. Using this template I can fetch data without using api. Its useful for simple websites.

We need to install EJS along with express. Then We need to set view Engine and the vies folder where our ejs files will be stored with file extension .ejs. But inside the file html is written. To access the data from the app.js file EJS uses some syntax similar to PHP.

(REACT CRASH)

21.04.2025

I started react crash course. The course that was in the moodle was a bit outdated where “create react app” was used. Now react has evolved and I took a latest crash course from the same channel. It was 3 hours long.

Here, react app is created using VITE. I learned how to use vite to create a react app. It was easy to setup a react app. However, there was a problem here while installing tailwind. So, I had to troubleshoot the issue and fix the problem.

I learned here that the things that are developed are called components. It is easier than normal vanilla javascript because components can be reused anywhere in other components and pages.

Then I explored props. Props can be passed in a component while using it. Inside a prop a default value can be assigned. Props are usually used to pass values or functions while calling or embedding a component.

I forgot to write about JSX format of html. It is similar to html but is can be written in a way how programming language works. We need to use {} to use the variables and it has some variations in the words that is used. I learned the basic syntax of writting JSX.

Here, in the react crash course a Job posting site is built using react. At first the job data was used from a json file. Then the data was fetched from a mock backend called json server. I learned how to use that.

Here the thing that makes react efficient is ‘a component’. It can be used in building pages or other components just by calling. Diferrent layouts components can be used as well. As all the components and pages are in separate folders, the main app file looks very clean making it easy to navigate. The components tree can be seen while inspecting the page.

I’ve learned about an extension called ES7-React-Redux-snippet which makes it very easy to write functions with just keywords.

I learned about using routes using React-Router-Dom. After creating the router, I only need to embed the pages inside the router. Props can be passed there making it clean. I've created a layout file having the navbar that is used in all the pages.
I've learned how to use tailwind inside the components. Also I learned how to use react icons inside the components.
I learned about react hooks. It's a state management feature of react. I used useState hook to show or collapse the job descriptions using a button. UseState hook takes the name of the state and the function that changes the state. A default state is assigned at first.

22.04.2025
Today I learned how to use NAVLINK to change the background color of the buttons in the navbar when switching the pages.

23.04.2025
Today I learned how to use a mock backend. I used json server that is created using the json file for job data. The data is fetched using an API.
I learned about react-spinner package. Used a spinner from this package while loading the page.
Most importantly, today I learned about useEffect hook. It takes in a function and an array. The effect takes place when the value inside the array is changed. I used this hook to fetch the data using the api when the app is rendered.
I learned how to create a dataloader and use it in different components to load the data. Its easy to use similar to the components. It is passed just like a prop.

11.05.2025 - 13.05.2025
I took a gap before this period because I was busy with my thesis work. Till 13th of may I practiced so far what I learned about react. Then immediately after that I started the rest of the course.
I learned how to pass a function as a prop. It is a very unique feature of react. I finished implementing POST request to add a job using form.

14.05.2025
Today is the day I finished my react course.
I learned how to use react-toast to show a notification of success or failure of posting a job or deleting a job or updating a job. But unfortunately I couldn't fix a problem here. The problem

is the toast was vanishing immediately after the action was done. I used the same jobloader to load the job description and update the job.

(PROJECT)

16.05.2025 - 17.05.2025

I started MERN project following the assigned tutorials. With the backend I was quite familiar because I've already done those in the crash courses. The project is a goalSetter app where a user can add, delete, edit and view its goals.
At first the CRUD routes were added inside the server.js file. Then the CRUD functions were shifted to controller that made the server.js file cleaner. Also, I fixed a bug where req.body was returning undefined. I added an error checking that solved that issue.
Next the routes were added using express router.
For handling async operation express-async-handler was used. I added an error middleware to handle error messages.
MongoDB was connected to the backend. I have previous experience on using mongoDB on other projects, so it was easy to apply. But I learned a new thing on how to build a schema. I created 2 model schemas for this project.
For testing all the CRUD operations Postman extension for vscode was used. I was familiar with it because I learned those on the express crash course.
I finished all the CRUD operations for goals and tested using the postman extension.

18.05.2025 - 19.05.2025
Today I started adding authentication. It was totally a new thing for me because here JWT tokens were used for user authentication.
I learned what JWT token is and how to use it for authentication. I learned new concepts like hashing passwords to secure it.
So here at first routes were set up for login, registration and getting user data. I tested those using postman.
Then schema for users was created.
I learned how to generate token and how this token can be used to authentication during registration and login. For generating token a JWT_SECRET variable was used to sign the token for a specific user ID. The user ID is actually inside the token and it can be extracted from the token using the JWT_SECRET variable.

For hashing and verifying passwords brcypt js library was used. The syntax was not complex at all, and it was easy to learn with the flow.
Then a middleware called authMiddleware was added to protect the routes using the token. Here the token is verified using the JWT_SECRET env variable. This authMiddleware was added to the goalroutes. So far till this point everything was smooth as I was learning and going with the flow.

20.05.2025 - 22.05.2025
Today I started the front-end part of the project. However here I had to take a different approach to create a react application. Because the tool in the tutorial was depreciated an was no longer suggested.
Instead of using create-react-app, I used vite to create the react app. Here, Initially I had to troubleshoot some problems regarding file structure and config files. Then, i made the initial commit.
At first I added the header with login and registration button. React icons package was used here.
I started adding pages and components for login, registration and the dashboard. It was quite easy because I recently used those syntax in the react crash course.
Routes were added using react router dom. Till the UI was done it was quite smooth.
When I started redux it started to become a bit complicated. I had to go through the redux twice to catch the concepts. I studied some definitions and terms for redux. Those are reduces, store, usedispatch, useSelector, asyncThunk etc. I was not familiar with these terms because it didn't resemble their applications. So I also studied about their operations and where to use them. Then it became clear.
Also for redux setup in the front end there needs to be some dedicated folders which I created manually. In the main.jsx file some files were imported from those folders such as store. A provider tag was used to provide the store for the main App file. The store file stores all the states for certain operations. All these were done by troubleshooting and these were not available in the tutorials.
Basically the redux manages the states of operations. Redux toolkit helps to create the slice and asyncThunk operations.
Here, we generally create a slice file. In this file we have thunk functions for each of the operations. Inside the thunk function we call the service function from service file where http request is made. For http request I've used axios. After calling this service function the states

are handled by extrareducers which are functions to handle states if operations are in pending, fullfilled or rejected. All the errors are handled inside the thunk function.
Inside goalSlice I have createGoal, updateGoal, deleteGoal , getGoals thunk functions. All these thunk functions handle states of the goals and the users.
The updateGoal thunk function was added by me as an extra feature for this project.
In the UI side useState, useEffect hooks were used to handle responsiveness. I had a critical bug for update functionality. I spent a whole day to find the bug. Finally, it was fixed and project was completed.
Unfortunately, in the deployment part I failed to deploy it on heroku. Because the tutorial was 3 years old and some of the things are outdated. I tried to fix those issues, but I failed. Maybe I'll try to deploy it in other platforms in the upcoming days.

In conclusion, I'd say that I have cleared a lot of concepts regarding a development of fullstack App. I learnt its efficient and easy to code if the codebase is divided into folders and files. Previously, I had questions regarding how a user state is handled when logged in. Using redux made this concept clear. I also had question about how the front end and the backend was connected. By going through this course now I have a clear concept. I hope to develop more full stack apps in the future and this course will be the base of my full stack developer journey.

Thank you.