Lappeenrannan teknillinen yliopisto

LUT UNIVERSITY (School of Technology)

Software Development Skills Full stack, Online course

Ekong Rejoice, 001494262

LEARNING DIARY, FULLSTACK MODULE

**LEARNING DIARY (22.5.2024 - 29.6.2024)**

**COURSE OVERVIEW**

22.5.2024  
  
I reviewed the general material and grasped the course's main goal. I chose the FULLSTACK module since it would further my knowledge in building applications. I read everything, which is the course info and everything we should install and all that needs to be done. Honestly, I already had everything I needed for this course installed and ready for my use. Then I started working on my learning diary and moved on to the first video of the course.

**NODEJS**

23.5.2024

I started watching the video and I followed smoothly. I learnt the definition of Node.js as a javascript runtime that allows JavaScript to run outside the browser, it utilizes the V8 JavaScript engine, it is mostly written in C++ for performance and it enables JavaScript as a server-side language. I also noted that Node.js handles asynchronous operations using a single thread. Manages and optimizes throughput and scalability by handling events and callbacks.It is great for REST APIs, microservices, real-time services, CRUD apps and non-CPU intensive toos. It also works with any database like MongoDB which is popular due to its speed and scalabilithy. I also learnt about NPM(Node Package Manager) and Node.js installation includes it and use it uses NPM to install third party packages. I learnt that Node.js comes with a set of core modules and we can create custom modules and also export them. We could also import modules.I learnt how to use the different path modules for like base name, directory name, file extension, path object and join paths as well. It was not really clear to me how the join paths worked initially but after looking at another article, I found out how it works with better understanding. I understood how to use the FS module for creating, reading and writing folders with their different command statements. We learnt how to run javascript in Node.js, which included the REPL for executing javascript dierectly in the console and also we can execute files using the terminal using the node filename command.I learnt encoding in file reading which is by using the UTF-8 to return that the data is returned as a string and not a raw buffer. The OS module provides information about operating system such as platform, architecture , CPU info, memory info, home directory, and uptime.We also touched on the URL module which was kind of confusing at first and I finally got to understand after repeating that part several times that it can parse and manipulate URLs, extracting components like host, hostname, pathname, query parameters, and dynamically adding parameters. I understood the EventEmitter class which creates custom event emitters by extending the EventEmitter class by listening for custom events. Implementing a logger class that emits events with messages and IDs helps in organizing logging functionality. I understood the reading and serving HTML files from the filesystem using the fs.readfile method. I learnt Using res.writeHead to set the status code and content type of HTTP responses. We touched public directory for static files by storing static files like HTML in a public directory for easy access and organization. We Used nodemon to automatically restart the server on code changes, improving development workflow. I learnt Setting up servers to use environment variables for port numbers, making them adaptable to different deployment environments. We started Instantiating and configuring an HTTP server to handle incoming requests and send appropriate responses. I learnt how to Respond with basic HTML content directly in the code for simple responses. I also Checked request URLs to determine which content to serve, useful for routing. Implementing error handling when reading files to ensure the server can respond appropriately in case of issues. Combining multiple core Node.js modules (fs, path, http, events, os, url) to build a functional web server with logging and dynamic content serving capabilities. The video was very long and I learnt a lot of things and they were How to create a basic server using Node.js,The importance of serving different content types such as HTML, CSS, and JSON, How to handle dynamic file paths based on the request URL, Usage of the path module to manipulate file paths, The significance of setting proper content types for different file extensions, How to read files asynchronously using the fs module, Handling specific errors, such as page not found (404), sending appropriate HTTP status codes (200 for success, 404 for not found, etc.), How to include CSS files and handle static assets, Understanding the basics of HTTP response handling, An overview of deploying a Node.js app to Heroku, using git for version control and deployment, Managing dependencies with a .gitignore file, Pushing code to a remote repository, Accessing the deployed application via a unique Heroku domain. Then I moved to the next section of the course.

**MONGODB**

26.5.2024

This tutorial started with the introduction to MongoDB and its compass and atlas which I understood to be a graphical user interface for MongoDB and a cloud-based implementation of MongoDB respectively. I followed the tutorial and installed MongoDB for windows. I was led to know that MongoDB is a NoSQL database that uses collections and documents instead of tables and rows and its documents in MongoDB are similar to JavaScript objects or JSON. one of the major advantages of using this interface is because of its scalability and flexibility. I followed on to downloading and installing MongoDB server and Compass. and then I used the MongoDB shell for commands like inserting, finding, updating, and deleting data. At one point, it was quite hard for me to follow along because my systems was crashing due to another software but I later fixed this issue and then I moved on to creating databases and collections and inserting documents and understanding their structure. I used operators like $set, $inc, and $rename for updates. querying data with various methods was also easy to follow along as we used find, findOne, and find.pretty(). I got to understand the embedding sub-documents (e.g. comments within posts) and indexing and searching and lastly getting to know certain operators for queries. I went on to continue viewing what was done in the tutorial and it was about managing data using MongoDB Compass and editing documents directly in compass. Lastly, I understood how to create an account and setting up a cluster on MongoDB atlas, selecting a cloud provider of which I chose Azure and of course I chose the student tier for this. Overall, the tutorial was quite educative and not very easy to follow but at the end of the day I did follow all that was given in it thanks to it being not too long.

**EXPRESSJS**

27.5.2024

In this tutorial I learnt all about the ExpressJS and all its components. I started understanding what it is as a backend framework for Node.js, often used with frontend frameworks like React, Angular, or Vue to build full-stack web applications. It also helps simplifies building web servers and APIs by providing minimal setup and extensive middleware support. Express makes building Node.js web apps easier with less code and It is flexible and gives developers control over how they handle server requests and responses. I learnt how to install express and since I already had NodeJS and then I was able to Create a basic server setup by requiring Express, initializing an Express application, and defining routes. I then created a simple server responding with "Hello World" using app.get('/', (req, res) => res.send('Hello World')). I went further in understanding creating routes and handling JSON which define routes to handle different endpoints and HTTP methods (GET, POST, etc.). I then started working with middleware and its's functions which have access to request and response objects and can modify them or terminate the request-response cycle. some example middleware for logging request URLs and timestamps using the Moment.js library. I learnt how to serve static files like HTML, CSS, and other static assets without manually creating routes for each file. I also understood development tools like NodeMon. The walkthrough provided using postman to test API endpoints by sending different types of requests and checking the responses. I was also able to access route parameters in Express using req.params to extract specific values from the URL. I used the JavaScript filter method to filter an array based on a condition. The need to convert route parameters from strings to numbers using parseInt to ensure proper comparison when filtering data was also done to my understanding. Then I moved on to handle requests for non-existent resources by checking if an item exists and responding with appropriate HTTP status codes and messages. The tutorial the importance of using the correct HTTP status codes (e.g., 200 for OK, 400 for Bad Request) to indicate the outcome of API requests. I generated unique IDs for new resources using the uid package, while creating, adding, updating and deleting data using the POST, PUT, DELETE request respectively. Organizing routes was not a problem for me because of the good explanation I got from the tutorial. I created and configured a template engine (e.g., Handlebars) in an Express application to render HTML templates, then I used layout templates in Handlebars to wrap and render content consistently across different views. I understood creating both a JSON API and server-rendered templates using Handlebars, showing how to serve data for different front-end needs, Integration of Bootstrap via CDN to style the application and make it look decent quickly with minimal custom CSS. we then passed data to the main view which was the handlebars templates from routes using res.render(view, data). Initially, I did not understand the looping through data in templates but after some time I was following Using Handlebars {{#each}} to loop through an array of data and render each item in a list. I then redirected users after form submission to a different route, rather than showing JSON responses directly in a browser. I got exposed to using external authentication libraries which I found very fascinating.

**ANGULAR**

28.5.2024

This course was so long that I forgot to keep track of everything that I learnt, and I just went along with all of it. So I will try to mention all that I had learned. I learnt that Angular is a front-end JavaScript framework for building single-page applications. Angular 2 and onwards are part of the same framework, while AngularJS (Angular 1) is a different framework. Angular Offers routing, HTTP client services with dependency injection, and more. I have good knowledge of JavaScript fundamentals. it Consist of template (HTML), logic (JavaScript/TypeScript), and styling (CSS). I used a Command-line interface for setting up, running a local dev server, and building Angular applications. Its services are Modular, reusable code for specific tasks like data fetching and input validation. Angular modules are used to organize and configure Angular applications. it has Functions in components that execute at specific points in a component's lifecycle. Observables are Used in Angular with RxJS for efficient asynchronous programming. Angular is test-friendly, providing tools for unit tests and end-to-end tests. Angular directives are instructions in the DOM that tell Angular how to modify the DOM. A design pattern is used to make components more modular and easier to test. HTML files with Angular-specific syntax are used to define the view of a component. CSS files for styling Angular components and applications. Angular allows handling of DOM events like clicks using event binding. Angular components and services are designed for reusability across applications. ngular components can emit custom events using the EventEmitter class for communication between parent and child components. I also learnt that JSON Server is a tool to create a fake REST API locally. I can create a file named db.json to simulate a database for JSON Server. Angular uses TypeScript, a superset of JavaScript. In Angular, you create interfaces to define the structure of your data models. Interfaces in TypeScript are like classes but are mainly used for defining shapes of objects. Interfaces can have optional properties denoted by a question mark. To create a new Angular component, you use the Angular CLI command ng generate component. Angular components consist of TypeScript, HTML, CSS, and a spec file. Angular uses directives like \*ngFor for looping over arrays in templates. You can create child components in Angular using the CLI command ng generate component. In Angular, you use input properties to pass data from parent to child components. Angular provides styling options using CSS files for each component. I also use third-party packages like Angular Font Awesome for icons in Angular applications. Services in Angular are used for encapsulating logic and data that can be shared across components. Angular services can be generated using the Angular CLI command ng generate service. The Angular HTTP client module is used for making HTTP requests to servers. I could import the HTTP client module in Angular's app module to use it throughout the application. In Angular services, you can use the HTTP client to make GET requests to fetch data from an API. JSON Server allows you to simulate a REST API locally without the need for a real backend. JSON Server enables CRUD (Create, Read, Update, Delete) operations on mock data stored in a JSON file. I was then broadening on the knowledge that HTTP requests in Angular can be made to a backend server to fetch data. The HTTP client in Angular returns an observable, which simplifies asynchronous operations. In Angular, event handling can be implemented using directives like (click) for button clicks. Components in Angular can emit custom events using @Output and EventEmitter. Event emissions in Angular are often used to communicate between child and parent components. Angular's ngClass directive dynamically adds or removes CSS classes based on conditions. Forms in Angular can be created using the FormsModule, allowing for two-way data binding. Angular's ngSubmit directive can handle form submissions without needing to manually prevent the default action. Basic form validation in Angular can be achieved using conditional statements in the component class.

Objects in Angular can be structured to hold form data before submission. Angular's @Output and EventEmitter are commonly used to notify parent components about form submissions. Clearing form fields after submission helps improve user experience and maintain data integrity. Angular components can interact with backend services to perform CRUD operations. HTTP requests in Angular are commonly used to send data to a backend server for storage or processing. Angular's HTTP client simplifies the process of making requests to a backend API by handling headers and other configurations. Angular components can use filters to manipulate or display data based on specific criteria. Understanding TypeScript helps in writing more robust and error-free Angular applications. The Angular framework provides various built-in directives and modules to simplify common development tasks. I then learnt How to use event emitters to communicate between components in Angular. Understanding Angular template syntax, including directives like \*ngIf and \*ngFor. Creating and using Angular services to manage application state and logic. Making HTTP requests in Angular using the HttpClient module. Implementing routing in Angular to navigate between different components/pages. Creating UI services to manage UI-related functionality across multiple components. I also understood Handling component interaction using services rather than passing data through props. I then used Angular CLI commands to generate components, services, and modules. Understood and managed Angular modules to organize the application. Building template-driven forms in Angular to handle user input. I then was Using reactive programming concepts like observables and subjects in Angular. Implementing conditional rendering in Angular templates using \*ngIf directives. Utilizing Angular component lifecycle hooks like ngOnInit. Debugging Angular applications using console.log statements and browser developer tools. Implementing routing guards to control access to certain routes in Angular applications. Applying CSS styling to Angular components and templates. The tutorial then had an Introduction to RxJS concepts like observables and operators. Organizing components, services, and modules within an Angular application.

**MEAN-STACK AND PROJECT DEVELOPMENT**

30.5.2024 - 29.6.2024

I started this project while I started watching the mean-stack tutorial because I knew that I was going to make some changes to the tutorial and use the same concept that the tutorial used because we were allowed to use the example code in our own project. So this what I learnt alongside doing my project as well. In the first video, I was just introduced to how the final products of the tutorial series will look and the things that we will be doing in reference to the project structure and how all this will be done and also how we will deploy all of it at the end of the series and it was very understandable how in the backend we used a server 3000 and in the deployment we used 4200 and they were all the same thing and the same way of work. Using npm install, I added necessary packages like Express for handling web requests, Mongoose for connecting to MongoDB, and other modules like bcrypt for encryption, CORS for allowing cross-origin requests, and JSON Web Token (JWT) for secure authentication.I learned about using JWT (JSON Web Tokens) and Passport.js for implementing authentication. This secures API endpoints by verifying tokens and ensuring only authorized users can access protected routes. Express was configured to serve static files (like HTML, CSS, and JavaScript files) using express.static, pointing to a public folder where frontend files will reside. I used Express Router to manage different API endpoints (/register, /authenticate, /profile). Each endpoint handles specific functionalities like user registration, authentication, and user profile access.Middleware functions like CORS and body-parser were added to handle cross-origin requests and parse incoming data from requests, respectively. We created a model for our users using something called Mongoose. This model defines what each user should look like in our database, like their name, email, username, and password.Security-wise, before storing passwords, we hashed them using bcrypt.js. This makes sure even if someone gets into our database, they can't see the actual passwords.When someone registers on our app, we handle their data using POST requests. This means we receive their info, hash their password, and save it all to MongoDB.Mongoose comes in handy here again because we use its methods to find users by their ID or username. This helps us with tasks like retrieving user data when needed.If something goes wrong, like an error happens, we've set up ways to catch these errors. This ensures our app doesn't crash and burn when things don't go as planned.To test if everything works, we used Postman. It's a tool where you can send different kinds of requests to your server and see how it responds. It helped us see if our registration process was working smoothly.MongoDB is cool because it's flexible. Unlike MySQL, where you have to plan out your database structure beforehand, MongoDB creates databases and collections on the go. This makes development more flexible and agile. We added passport.initialize() and passport.session() middleware in our Express app to handle authentication. Created a JWT (JSON Web Token) strategy in Passport to authenticate users based on tokens sent in the authorization header.Set up a passport.js file to configure our JWT strategy, defining how tokens are extracted and verified.Added a comparePassword() function in the user model to securely compare entered passwords with hashed passwords stored in the database. Implemented error handling for token validation failures and unauthorized access attempts.Used Postman to test the /authenticate route, verifying that tokens are correctly issued upon successful login and that protected routes return user data when authenticated. need to install Angular CLI using npm install -g @angular/cli to start creating Angular applications.Use ng new angular-source to generate a new Angular project called angular-source.This file acts as the main meeting place for all components, services, and modules. Components and services need to be imported and declared here.Angular uses RouterModule and Routes to handle navigation between different components/pages. Routes are defined in app.module.ts using RouterModule.forRoot(appRoutes).Navigation links in Angular are managed using [routerLink] directive. To highlight the active link, use [routerLinkActive] and [routerLinkActiveOptions]="{ exact: true }".  
After this tutorial I started getting so many errors that I tried contacting the teacher about but I did not get a any good response from anyone atall. It has been one month and still no reply so I decided to watch other mean stack tutorials to complete my project by myself.