



Lappeenrannan teknillinen yliopisto

Software engineering

Learning diary

CT70A9140 - Software Development Skills: Full-Stack

Name: Marek Mikula

Student number: 003773431

Year: 2026

1 NodeJS

11. 2. 2026

I pretty much quickly went through the first learning video, because I've known NodeJS very well now. For the past 8 years I've worked as a full-stack developer mostly using PHP (Laravel), Vue.js (Nuxt.js) and TypeScript. Along the way I've also worked on some microservices written in NodeJS. I've also went through a NodeJS course on my home country university. There was nothing new in the video for me. But I really support teaching programming using *Traversy Media's* channel, because I've also learned from their videos when I was starting with programming.

2 MongoDB

11. 2. 2026

MongoDB was not new to me also. We've used it on one project to save log files. But I have not really worked with it that much. I haven't used hosted MongoDB instance as in the video. Instead, I've spined up a Docker image on my local machine. I've used the official *mongodb* image with the *latest* version tag. To browse the data and collections I've used a client in my IDE – PhpStorm.

I have never worked with MongoDB in the form of raw queries. I've always used an API to work with the database. So, this was new to me. I kind of like the way it works, it reminds me of JavaScript. All the CRUD methods were easy to use and easy to remember.

I really liked the way the aggregations work in MongoDB. To me it is super clear and easy to use. It feels kind of like shell commands when you pipe the result of one command into the input of another command. I wish MySQL, MariaDB or PostgreSQL had something similar.

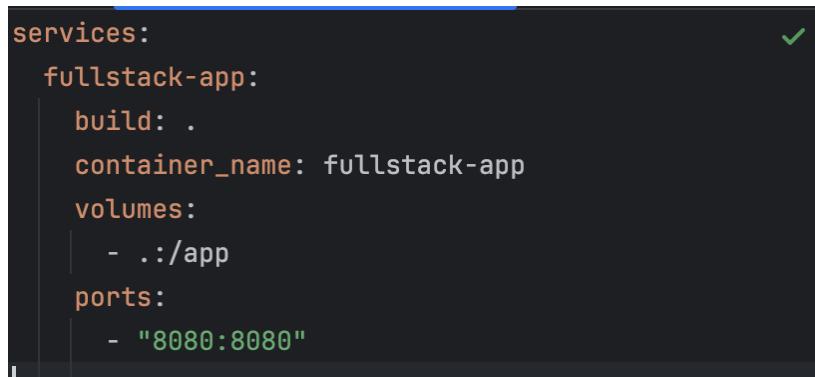
For trying the MongoDB in NodeJS I've also used a Docker image. I've used *node* with version *22-alpine*.

3 ExpressJS

11. 2. 2026

I have no previous experience with ExpressJS. I've always worked with either AdonisJS, which is like Laravel the way it is written, or NestJS.

For my local development I've used Docker again. I've used the *node* image with version *22-alpine* again. I've setup minimal *docker-compose.yaml* file.



```
services:
  fullstack-app:
    build: .
    container_name: fullstack-app
    volumes:
      - .:/app
    ports:
      - "8080:8080"
```

Figure 1 Minimal docker-compose.yaml setup

The basics of routing, getting request params, responding in JSON and so on were straightforward. I did not come to any blockers or anything like that. Since I am using Postman daily in my work, that was not a new thing either.

I really like how the framework is built. It is so different than Laravel, which is heavily opinionated and sometimes feel like magic. ExpressJS looks lightweight.

I went through the crash course without any problem. The only thing I did not really like was the *ejs* templates. I am used to either Vue or Blade templates and they are much better in my opinion. I've learned only the basics of the framework, and I can't wait to try to create my own project in this framework.

I appreciated the way the tutorial is done. Even though the tutorial is for beginners, the author also includes some very important patterns and concepts from programming (controllers, middleware, request-response, error handler, etc.).

4 React

12. 2. 2026

I've already had some experience with react prior to this course. I've taken a React course in my home country university. I've also got big experience with Vue.js, which is a brother to React.js so I know the concepts of these front-end frameworks and meta-frameworks like Nuxt.js/Next.js.

For the React part, I've also used the same Docker setup as for the ExpressJS part. Everything in the video was pretty much known to me. In the course in my home country, we've also used TypeScript for better typing, otherwise everything was like this course.

I did not have any problems when following the video.

5 MERN Stack & final project

14. 2. 2026

I've started my project with a simple docker setup which I copied from previous steps. At first, I've added only the node container. I will ask other containers when I need them. I've talked with Gemini about the standard project structure of a project which uses ExpressJS, and I will follow that as it seems like a best practice. I'm always very serious about projects structure, because it really makes a big difference.

Creating the base of the project and couple of the first goal routes, controllers and one middleware was easy. Then I had to set up the MongoDB. I've added another container to my *docker-compose.yaml* file. I've used the *mongo* image with the *latest* version. One thing that caught me off guard when I was connecting to the database using my PhpStorm client. I had to add the *authSource* parameter to the URL. Otherwise, I would get a bad credentials error. The same thing happened to me when I was trying to connect to the DB from the code. It caught me off guard again. My MongoDB setup was kind of different because I am using Docker. But I managed to create a connection. Once I had the connection up and running, updating my routes to use the DB connection was easy. I found one small bug in the tutorial. In update route, the author uses the *findByIdAndUpdate* method and then he returns the found value. But the returned value is not updated yet. That creates a little bit of confusion and that is not a standard thing in REST. I've used the *findById* method to find the model and then directly on the model the *save* method.

Then I reorganized the project, so it is more "monorepo-ish". I've added workspaces to *package.json*. Then I also added a TypeScript support.

After that I've added new routes for authentication without a problem. I've previously had some experience with JTWs. Creating the middleware was no problem too. I've had some troubles when applying the middleware, but that was

because of Typescript. I've had to inject the User type to default ExpressJS Request type. When I've had the middleware ready, I could finally secure my routes with that middleware.

15. 2. 2026

I haven't used the steps from the video to install the React app. The *create-react-app* way is long time outdated and should not be used. Instead, I've used Vite and installed the React app with Typescript support. Then I installed the Redux library with the help of their documentation. After that, I've installed TailwindCSS, because I am used to it. I did not want to use raw css.

After creating the register and login form, I had to finally setup the store with Redux and connect it to the backend. That was hard. Mostly because of the TypeScript. I had to check the Redux docs on how to correctly type things. Some things were kind of confusing, but I managed to set it up. Then the API call to register the user gave me a little headache because of CORS and the fact that I am using Docker. But at the end I managed to finish the register page.

Login page was quite easy, because now I knew how to do it correctly. I pretty much copied everything from the register page with some changes. After that, I've had working register and login routes.

Securing the dashboard route so unauthenticated user cannot access it was easy. But the solution seems weird. I would rather use a middleware than having it directly in the component. This breaks the rules of separation of concerns from my point of view.

I then slightly redesigned the forms, because I did not like them at all. For that I've used Gemini.

Once I've had the dashboard secured, fetching the goals and deleting them was easy. It was pretty much copy-paste from the previous steps with small changes.

18. 2. 2026

Once I've had the example project done, it was time to think about my own project. I've got an idea to transform the example project into a digital garden.

First, I've transformed the UI, so it looks more "garden-ish". I renamed the app to Flowee. I've added green colors and new logo element. Then I transformed the backend. I replaced goals with flowers. Then I've added a possibility to water the flowers. Also, I've added a progress bar showing the need to water the flower based on its watering cycle. Then I've added new fields to the creation form and the flower card.

Since it is a gardening app, the most important information for gardening is the weather. I've decided to add a card showing current weather in Lappeenranta. I've found a free API with no need for an API key. I designed a weather card using Gemini and I've used the API to pull data about current weather based on latitude and longitude. I've added a cache for the information, so the API won't get called too many times.

Last, I've added new attribute indicating where the flower is placed – a room.

As a final touch, I've updated the readme file, so the installation process is clear. I've also made some changes on how the env file is used in the project. Finally, I've deleted the project folder and tried the installation process myself to check if everything is working correctly.

Even though I've had some previous experience with programming, I consider taking this course a good decision. I've never tried to set up a fullstack project with this programming stack on my own. I take this as a valuable lesson. I am also glad I've decided to enhance my app with TypeScript and Tailwind, because it added a little bit of struggle into the project.

If I could add some advice on how to make the course more up to date, I would maybe consider this:

- Find some more up to date materials, mostly in the MERN stack section. Sometimes there was no context at all in the video. I cannot imagine me being a beginner watching that. I would be super confused.
- I would maybe try to switch the stack a little bit. I would also consider including a frontend framework (Nuxt, Next), because they are popular these days and to be honest, they are much easier to setup than pure React/Vue apps if you want store, routing, etc. Maybe something with GraphQL also?
- Consider adding a TailwindCSS into the course, or even Shadcn.

Anyway, it was fun learning something new.