Lappeenrannan teknillinen yliopisto

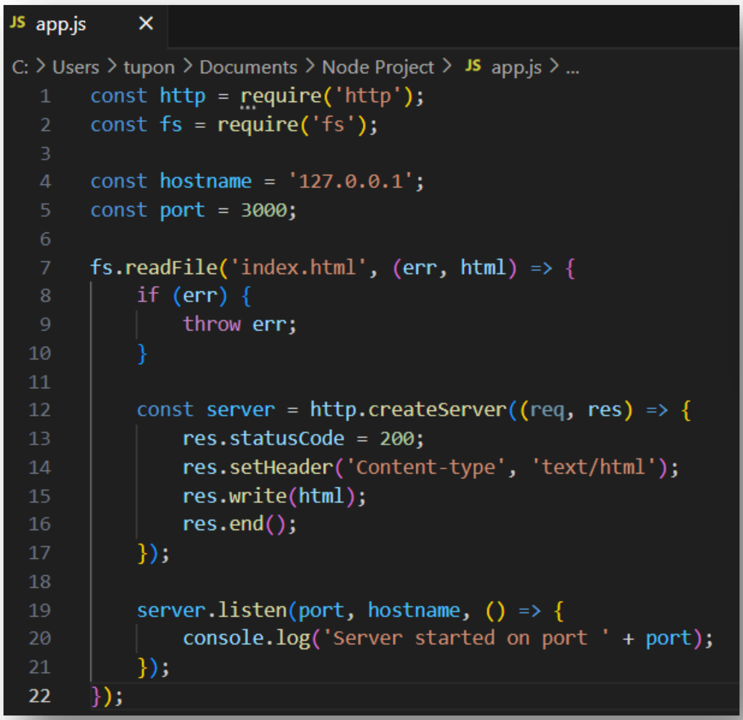
Software Development Skills: Full-Stack 2023-24, Online course

Tommi Uponen, 001134917

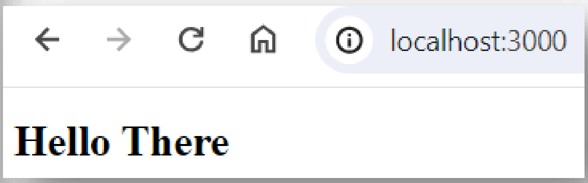
Learning Diary

5.3.2024

Today, I started the online course by looking at the introductory videos for NodeJS and MongoDB. I have used NodeJS before, but a refresher on the topic was quite useful. Along the NodeJS video, I learned a lot about the features that come with NodeJS such as non-blocking I/O calls which makes the apps efficient and fast performance-wise. After watching the theory part of the video, I started the installation of NodeJS version 12. I created a project folder in which I created a simple HTTP website according to the tutorial video. I used VS Code as the code editor, and I got the website working. Below is the code used for the server and website, and the results:

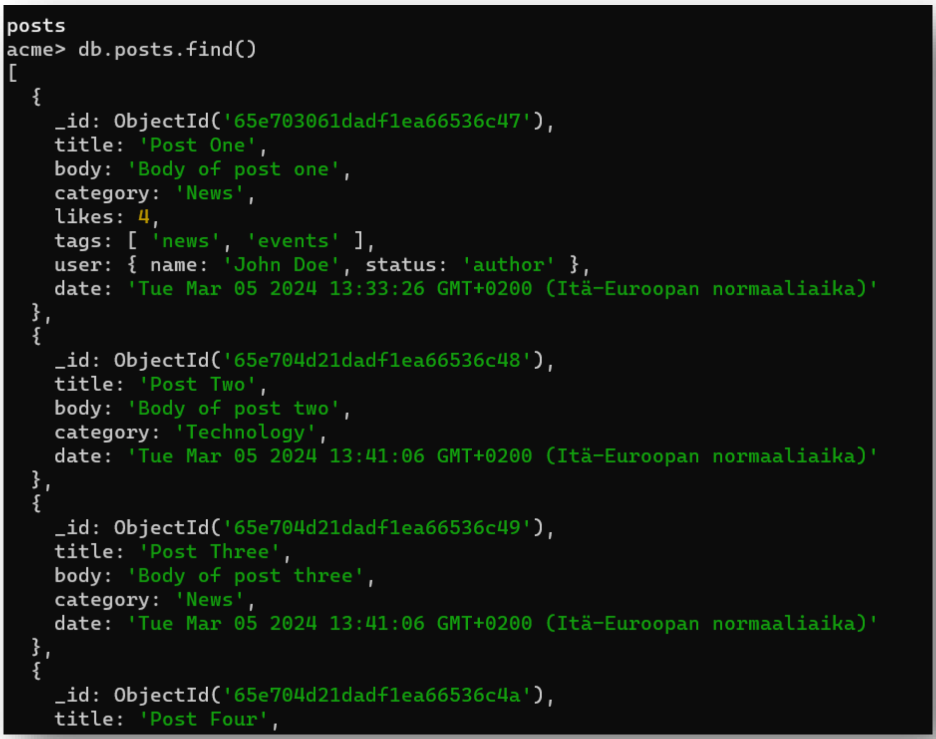

app.js
C: > Users > tupon > Documents > Node Project > app.js > ...

```javascript
const http = require('http');
const fs = require('fs');

const hostname = '127.0.0.1';
const port = 3000;

fs.readFile('index.html', (err, html) => {
    if (err) {
        throw err;
    }

    const server = http.createServer((req, res) => {
        res.statusCode = 200;
        res.setHeader('Content-type', 'text/html');
        res.write(html);
        res.end();
    });

    server.listen(port, hostname, () => {
        console.log('Server started on port ' + port);
    });
});
```



Hello There

The other thing I wanted to go through today is the MongoDB introduction and installation. While downloading MongoDB, I encountered some issues with the Mongo Shell not working properly. I went to the troubleshooting tab on the MongoDB homepage to look for a solution and I found that you can install Mongo Shell separately, which comes with the exe file "mongosh.exe". All in all, I got the shell running and tested out creating a collection, inserting a few queries to the collection, and called it a day. Below is a picture of the progress so far in my database named "acme":

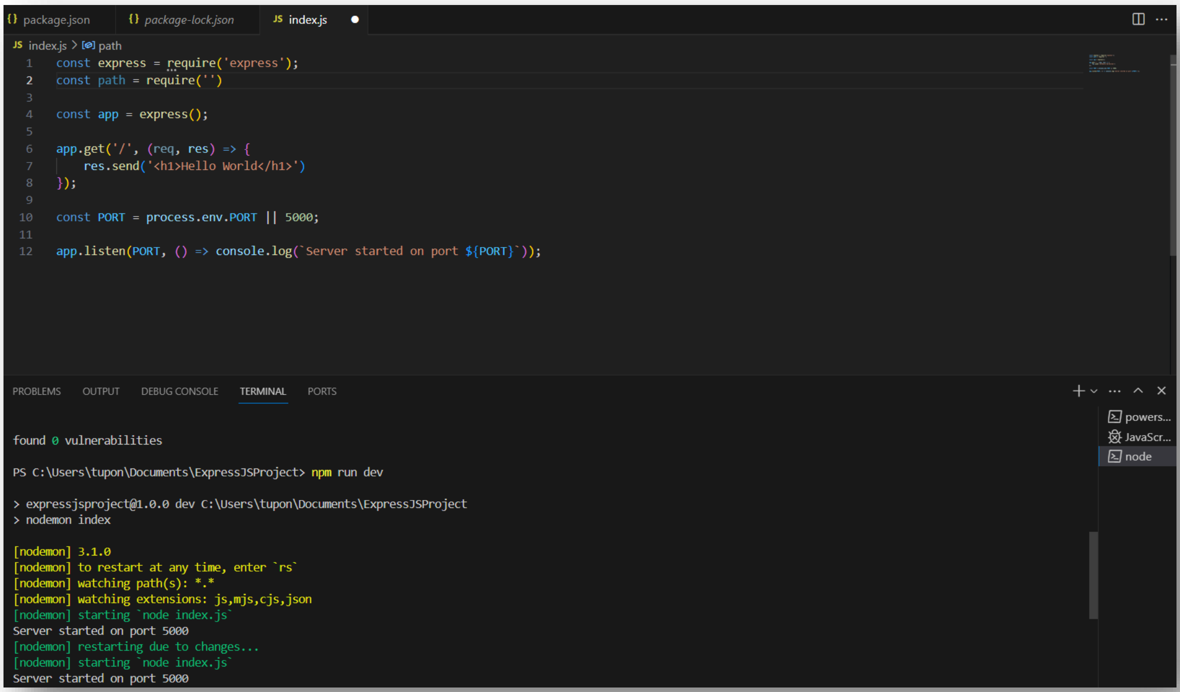


```
posts
acme> db.posts.find()
[
  {
    _id: ObjectId('65e703061dadf1ea66536c47'),
    title: 'Post One',
    body: 'Body of post one',
    category: 'News',
    likes: 4,
    tags: [ 'news', 'events' ],
    user: { name: 'John Doe', status: 'author' },
    date: 'Tue Mar 05 2024 13:33:26 GMT+0200 (Itä-Euroopan normaaliaika)'
  },
  {
    _id: ObjectId('65e704d21dadf1ea66536c48'),
    title: 'Post Two',
    body: 'Body of post two',
    category: 'Technology',
    date: 'Tue Mar 05 2024 13:41:06 GMT+0200 (Itä-Euroopan normaaliaika)'
  },
  {
    _id: ObjectId('65e704d21dadf1ea66536c49'),
    title: 'Post Three',
    body: 'Body of post three',
    category: 'News',
    date: 'Tue Mar 05 2024 13:41:06 GMT+0200 (Itä-Euroopan normaaliaika)'
  },
  {
    _id: ObjectId('65e704d21dadf1ea66536c4a'),
    title: 'Post Four',
```

11.3.2024

Today, I continued my work and started delving straight into the video to learn other features of MongoDB. I have been watching the older MongoDB video and I found out that the update function is deprecated nowadays. However, when I tried to use the original update function with '$set', I got the function working even though it was warning the user of deprecation. Other things I got to try in this video were incrementing values, renaming posts, deleting posts, and searching posts by first creating an index and then searching the text. All in all, I think I learned a lot of the default operators and functions in MongoDB, and I found the compass app very useful for utilizing the database.

Next, I moved on to the ExpressJS crash course video. The start of the video explained the usefulness of ExpressJS as a node framework, and I learned that it is quite fast, effective, and easy to use especially when building web applications with NodeJS. I am already familiar with the syntax and basic route handling, so I continued to the environment installation part and initialized the Express JS folder. Here is the result for today:



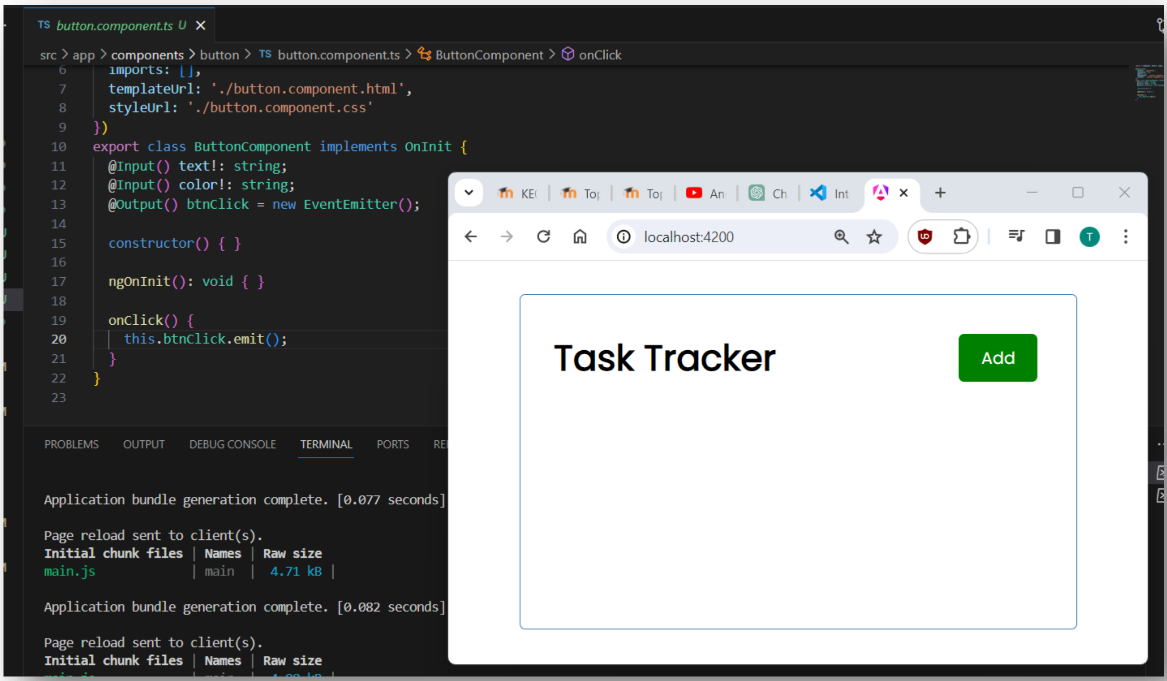
```
const express = require('express');
const path = require('')

const app = express();

app.get('/', (req, res) => {
    res.send('<h1>Hello World</h1>')
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Server started on port ${PORT}`));
```

```
found 0 vulnerabilities

PS C:\Users\tupon\Documents\ExpressJSProject> npm run dev

> expressjsproject@1.0.0 dev C:\Users\tupon\Documents\ExpressJSProject
> nodemon index

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Server started on port 5000
[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Server started on port 5000
```

18.3.2024

Today, I continued the express JS video. I learned a bit about static folders and middleware, which seem to be quite useful in simplifying some of the processes. After learning to use middleware and static folders, I followed the rest of the video through creating, updating, and deleting members. I ran into some trouble regarding the express handlebars in the new video, but it seems the latest library was not compatible with some of the code because it threw an error saying ‘node: path not found’ even though it worked with an older version. However, I continued my work by proceeding to the Angular tutorial.

At the start, I was confused about the properties and components of Angular, but after further testing, it became more clear to me how everything works. I encountered some problems regarding the initialization of variables but that's because typescript requires you to initialize your variables, so I got it working by putting an exclamation mark on the variables (text and color).



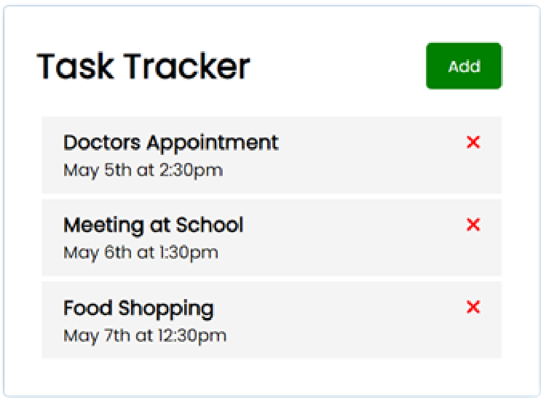
```
  imports: [],
  templateUrl: './button.component.html',
  styleUrl: './button.component.css'
})
export class ButtonComponent implements OnInit {
  @Input() text!: string;
  @Input() color!: string;
  @Output() btnClick = new EventEmitter();

  constructor() { }

  ngOnInit(): void { }

  onClick() {
    this.btnClick.emit();
  }
}
```

```
Application bundle generation complete. [0.077 seconds]

Page reload sent to client(s).
Initial chunk files | Names | Raw size
main.js             | main  |  4.71 kB |

Application bundle generation complete. [0.082 seconds]

Page reload sent to client(s).
Initial chunk files | Names | Raw size
```

All in all, today I learned the basics of ExpressJS and I got to a good start with Angular. Next time I will finish the Angular crash course tutorial and probably proceed to the MEAN-stack tutorial, which will be my final course project after I add my own modifications to it.

27.3.2024

I decided to continue my Angular crash course video today, starting with the mock-tasks. I encountered some problems along the way with my Angular version, as I had to import my common modules manually for the components to work (for example, when using '*ngFor'). However, I got my code working by manually importing the correct modules and the outcome is this.


Task Tracker

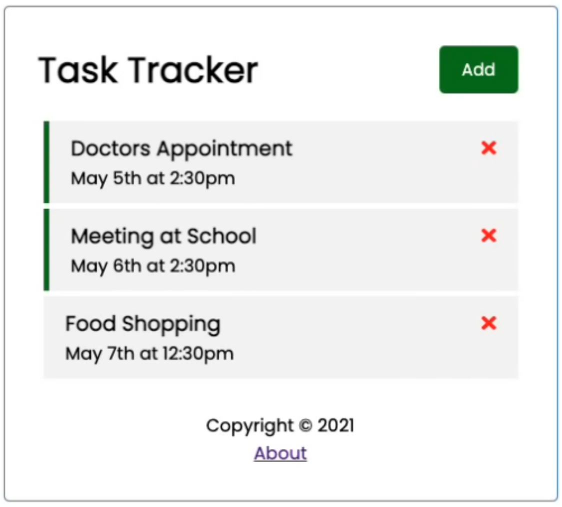
Add

Doctors Appointment ×
May 5th at 2:30pm

Meeting at School ×
May 6th at 1:30pm

Food Shopping ×
May 7th at 12:30pm

The next thing on the list was to set up the JSON server. At first, I had some problems with the tasks not appearing and I don't know what change fixed that, but suddenly it started working. After that, I continued with the other features as well as routing without any difficulties. All in all, I gained a slightly better understanding of how Angular works and how I can modify my webpage with components. Next time, I will continue with the tutorial series of MEAN-stack, while making my own changes to it at the same time. Here is today's result:


Task Tracker

Add

Doctors Appointment ×
May 5th at 2:30pm

Meeting at School ×
May 6th at 2:30pm

Food Shopping ×
May 7th at 12:30pm

Copyright © 2021
About

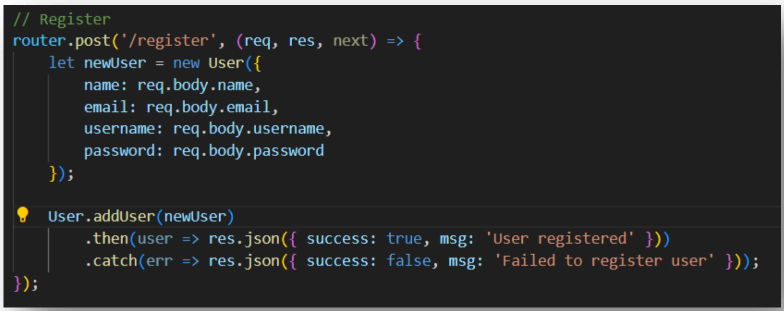
7.4.2024

My goal for today is to start the MEAN-Stack project and get it to a good start. I watched the first two videos today and while they were quite simple and the setup was easy, the videos are quite old and there were some better methods such as adding the dependencies with 'npm install'. I encountered some problems with node version 12, as I couldn't get the other dependencies to work with it as the available versions required a higher node version. I decided to use the latest available node version and got everything working properly (NOTE: I reverted my node version to 12 later in this diary as I couldn't progress with the latest version. I found the exact versions from Brad Traversy's source code in GitHub)

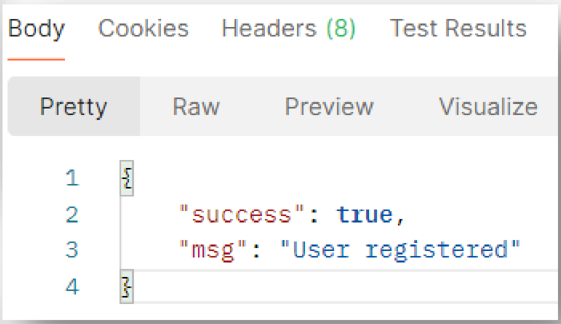
All in all, I got the dependencies working without problems. I will continue the tutorial series next time and see how far I can go now that I have a working setup.

9.4.2024

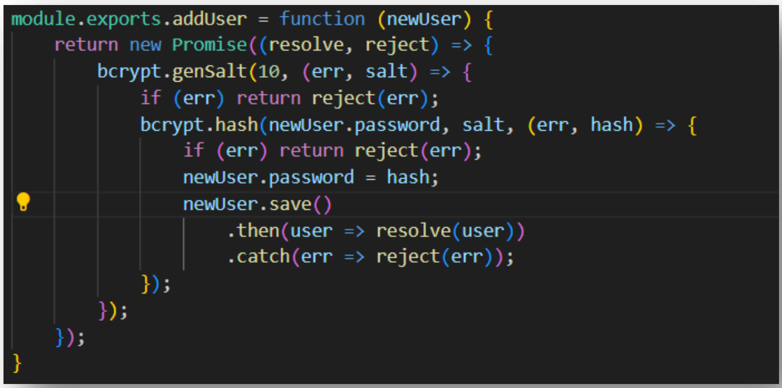
Today I continued my work with the user models and with the function to add a user. Everything in the video was clear to me except the last process with the post requests. This part of the video was the biggest challenge today because I encountered problems regarding mongoose no longer accepting callbacks mentioned in the video. However, I fixed this problem by modifying the 'addUser' function to work with the latest version of mongoose and here is the result:



```
// Register
router.post('/register', (req, res, next) => {
    let newUser = new User({
        name: req.body.name,
        email: req.body.email,
        username: req.body.username,
        password: req.body.password
    });

    User.addUser(newUser)
        .then(user => res.json({ success: true, msg: 'User registered' }))
        .catch(err => res.json({ success: false, msg: 'Failed to register user' }));
});
```

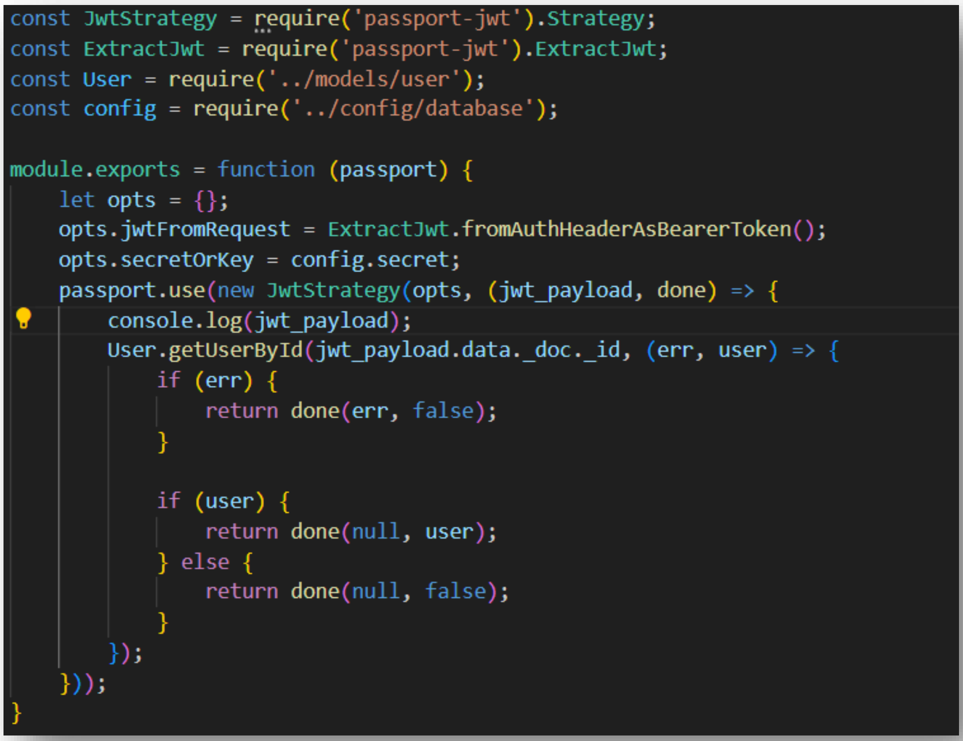


```
{
    "success": true,
    "msg": "User registered"
}
```



```
module.exports.addUser = function (newUser) {
    return new Promise((resolve, reject) => {
        bcrypt.genSalt(10, (err, salt) => {
            if (err) return reject(err);
            bcrypt.hash(newUser.password, salt, (err, hash) => {
                if (err) return reject(err);
                newUser.password = hash;
                newUser.save()
                    .then(user => resolve(user))
                    .catch(err => reject(err));
            });
        });
    });
}
```

The next thing was to proceed to the user authentication and token. This part also had major changes and the callback was no longer accepted, so I modified the code by replacing ‘fromAuthHeader’ with ‘fromAuthHeaderAsBearerToken’:



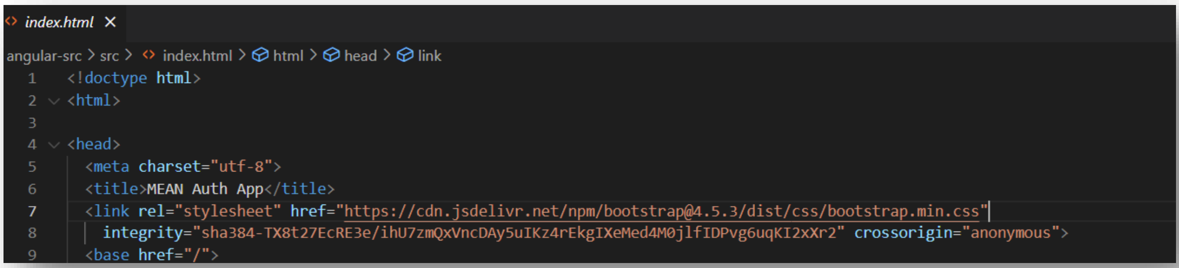
```
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const User = require('../models/user');
const config = require('../config/database');

module.exports = function (passport) {
    let opts = {};
    opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
    opts.secretOrKey = config.secret;
    passport.use(new JwtStrategy(opts, (jwt_payload, done) => {
        console.log(jwt_payload);
        User.getUserById(jwt_payload.data._doc._id, (err, user) => {
            if (err) {
                return done(err, false);
            }

            if (user) {
                return done(null, user);
            } else {
                return done(null, false);
            }
        });
    }));
}
```

The following changes fixed most of the code, but still I didn’t get the profile protection step correct because it still says ‘unauthorized’ even though I have followed the video in every step and looked at solutions in the comment section. I will hopefully get that problem fixed next time but I learned a lot about passports today and I got everything else to work fine at least.

10.4.2024

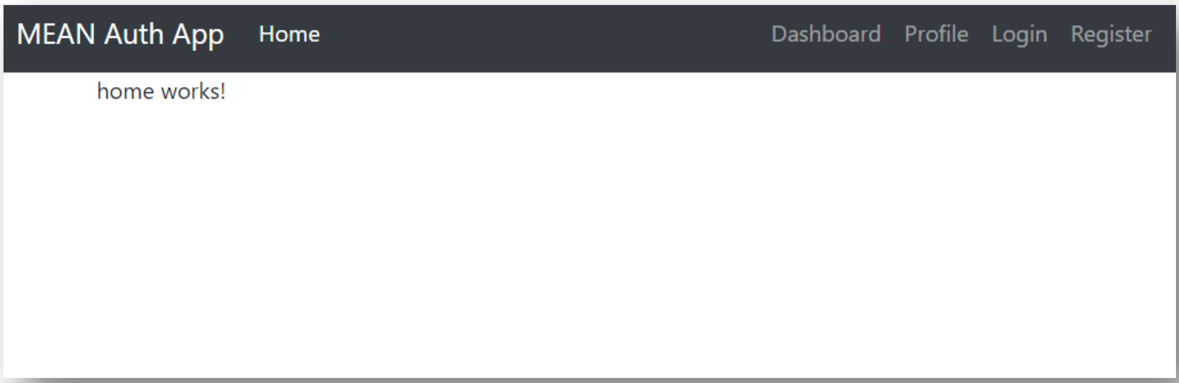
When proceeding to the next video about Angular, I found out that it became very difficult with the latest node version, so I decided to redo my work with node version 12. I got everything to work with some minor difficulties and differences in the code but now I got the angular-cli working with the exact version as in the video. I also tested the user register, authentication, and profile requests, and I also solved the profile request by using ‘fromAuthHeaderAsScheme’ mentioned in the notes of the course page.

After all the changes, I continued my work with the angular-cli by adding the components and trying out the new style used in the webpage. I found out that the old style in the video was deprecated and no longer available, so I decided to use a different style from the same bootstrap page:



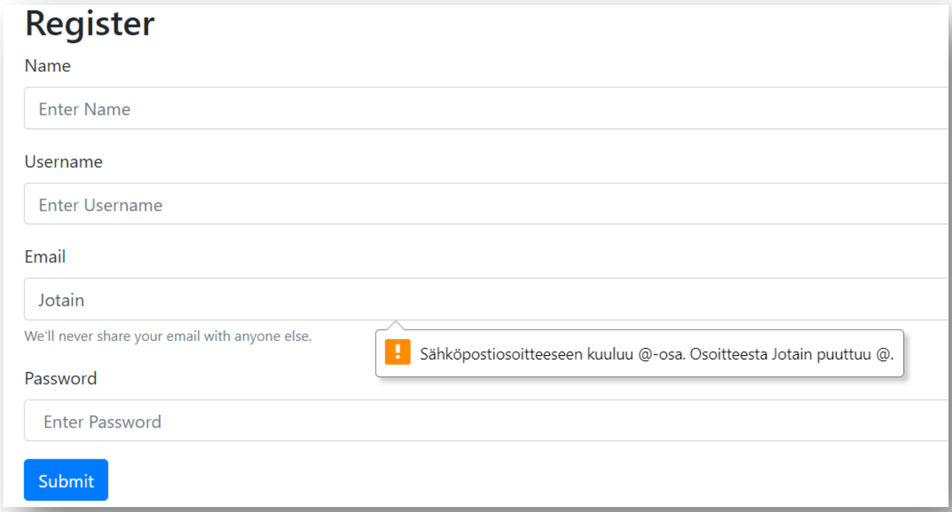
```html
<!doctype html>
<html>

<head>
  <meta charset="utf-8">
  <title>MEAN Auth App</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.5.3/dist/css/bootstrap.min.css"
    integrity="sha384-TX8t27EcRE3e/ihU7zmQxVncDAy5uIKz4rEkgIXeMed4M0jlfIDPvg6uqKI2xXr2" crossorigin="anonymous">
  <base href="/">
```

Overall, I learned a lot today as I had to redo my work and debug several problems, but this worked as a good practice. Here is the progress of the webpage made with angular components and my new style so far:



21.4.2024

Today, I decided to finish this tutorial series. I started by working on the home and register sections. I added some content for the home and register sections for the pages to look a bit better. My own selected style differs a bit from the one used in the tutorial because that style was deprecated. After stylizing both sections, I added the flash messages for registration, which activates if the user doesn't fill in all the fields or the email type isn't valid. However, I didn't like the email validation flash message, so I found a function in the email field called 'emailHelp', which notices the user that it must contain an '@' symbol to be a valid email. The register section looks like this with my changes:


Register

Name

Enter Name

Username

Enter Username

Email

Jotain

We'll never share your email with anyone else.

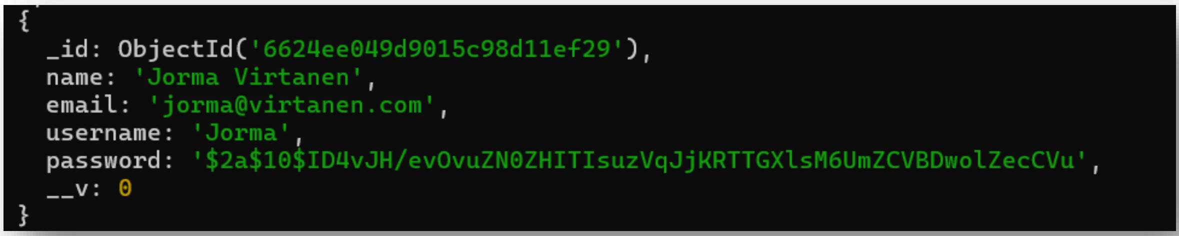
Sähköpostiosoitteeseen kuuluu @-osa. Osoitteesta Jotain puuttuu @.

Password

Enter Password

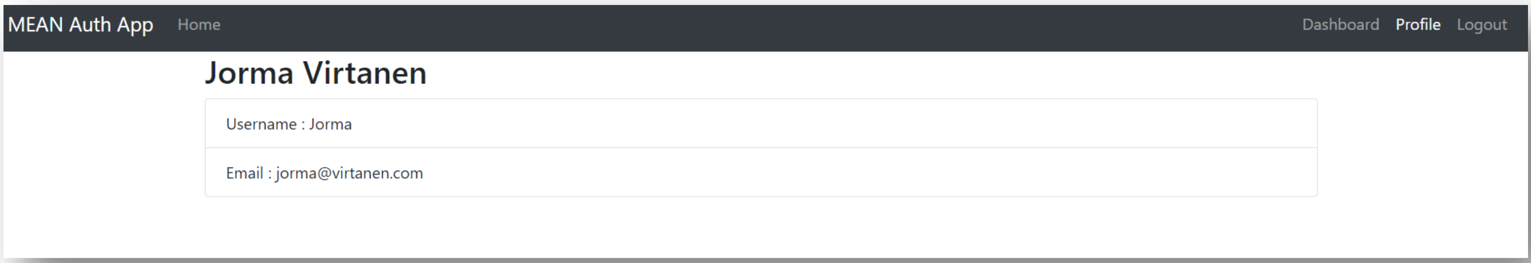
Submit

Next thing was the user registration functionality, which worked like a charm on the first try. Once the user has registered, the user is stored correctly in the database:


```
{
  _id: ObjectId('6624ee049d9015c98d11ef29'),
  name: 'Jorma Virtanen',
  email: 'jorma@virtanen.com',
  username: 'Jorma',
  password: '$2a$10$ID4vJH/evOvuZN0ZHITIsuzVqJjKRTTGXlsM6UmZCVBDwolZecCVu',
  __v: 0
}
```

After completing the user registration part, I continued the tutorial by creating the login and logout functions. The login should receive the token when I enter the correct user credentials and put up a flash message saying that you are logged in. This feature also worked in one go and I had zero problems with it. As an extra, a login page usually has a remember me – checkbox for the webpage to remember the user by adding a cookie, so I decided to add that checkbox. However, I noticed that the checkbox was also in the source code but not in the tutorial series, so this feature seems to be missing from the tutorial.

The logout feature was as simple as the login feature, so I continued the tutorial by hiding some of the webpage sections from the user if the user has/hasn't logged in. Next was to make protection, which blocks the user from going to the dashboard and profile sections by '/dashboard' or '/profile' if the user hasn't logged in. Finally, I added some content and style to the dashboard and built the project with angular 'ng build'. Now I have a working project with all the features implemented and in a different style. I learned a lot about the relationship between the backend and frontend through this tutorial series and next time, I will add some personal modifications to this project, to make it represent my skills better. Here is a picture of the project running through 'nodemon' localhost at the profile section, when I have logged in as the registered user 'Jorma Virtanen':



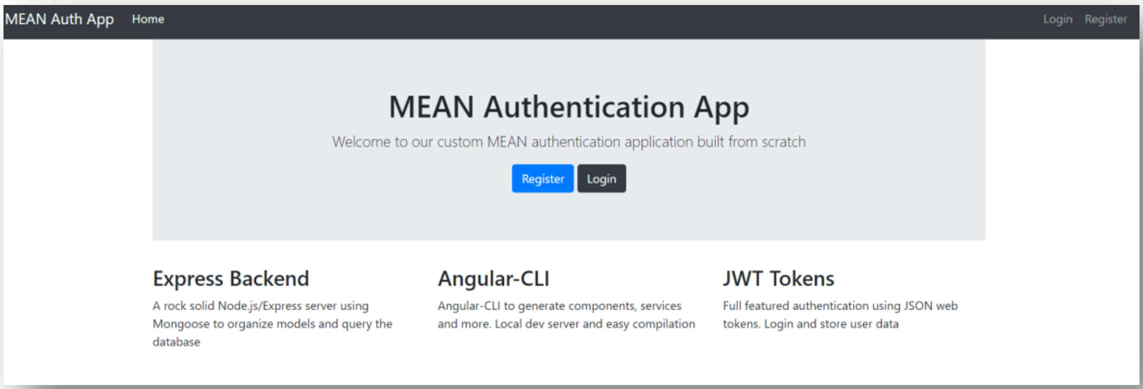
Jorma Virtanen

Username : Jorma

Email : jorma@virtanen.com

23.4.2024

Today I decided to make my own modifications to the project to make it look and feel better. I wanted to keep the style as it was minimalistic and provided good clarity for the user. However, I wanted to chase that same style as on the home page, which looks like this:



MEAN Authentication App

Welcome to our custom MEAN authentication application built from scratch

Express Backend

A rock solid Node.js/Express server using Mongoose to organize models and query the database

Angular-CLI

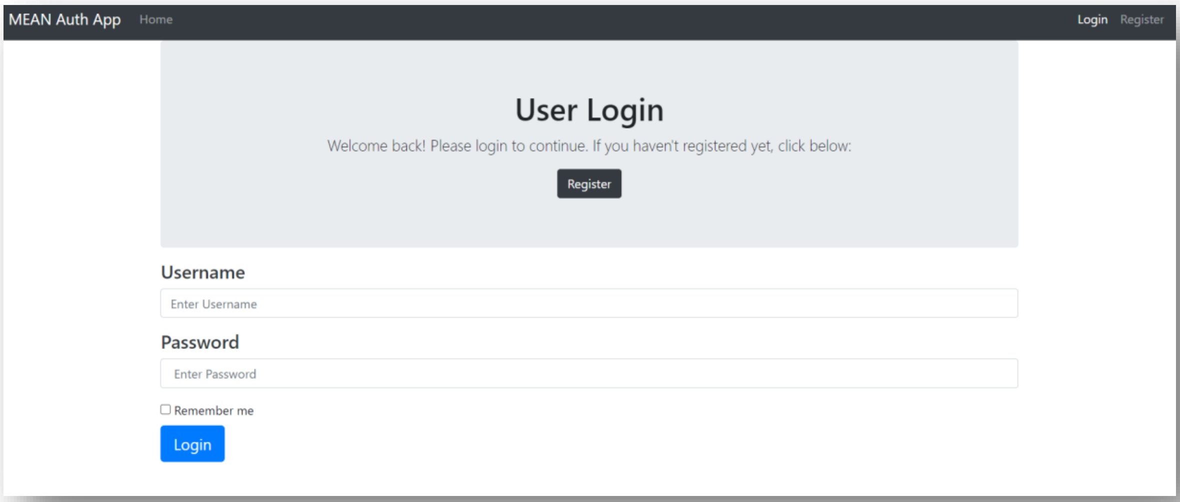
Angular-CLI to generate components, services and more. Local dev server and easy compilation

JWT Tokens

Full featured authentication using JSON web tokens. Login and store user data

To make that happen, I modified all of the components in Angular to look similar to the same style. I started with the login and register sections as they were barebones. I changed the heading for the fields in login and register as well as added a similar lead dark theme on top to match the style. However, I wasn't done with that and decided to add the typical buttons for navigation, which will guide the user to the register section if he/she hasn't yet registered a user and vice versa. Here is the result of the login and registration sections:

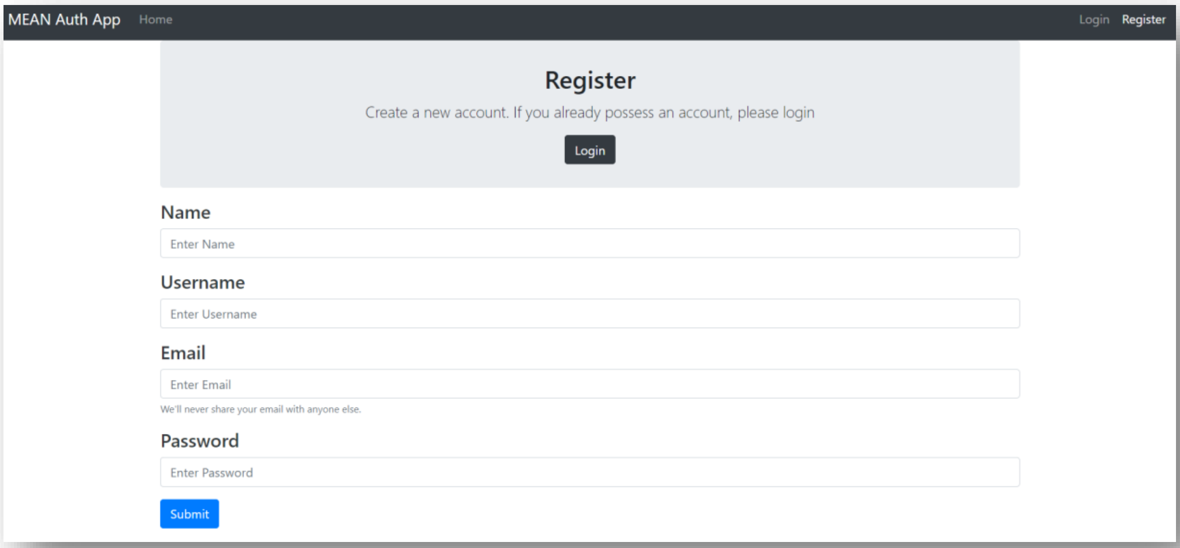
User Login:



User Login

Welcome back! Please login to continue. If you haven't registered yet, click below:

User Registration:

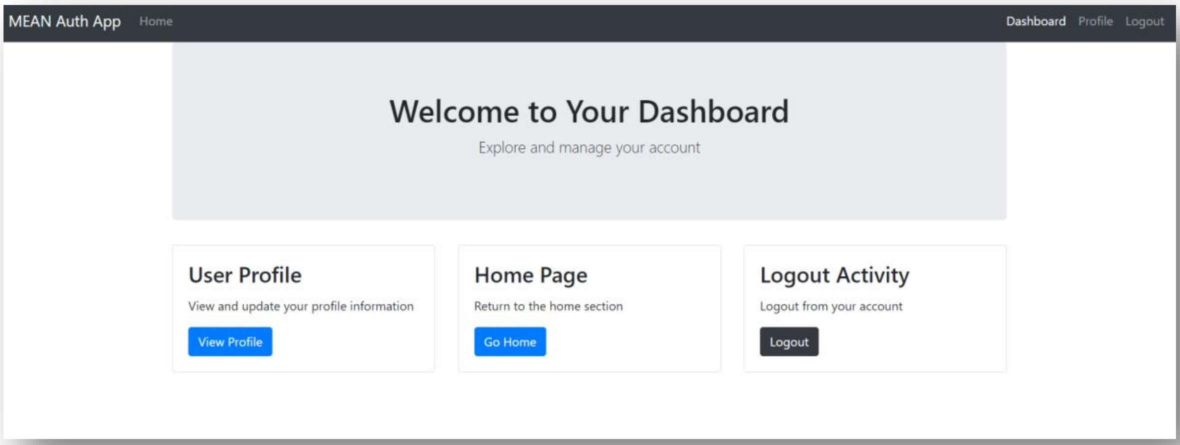


Register

Create a new account. If you already possess an account, please login

We'll never share your email with anyone else.

Next, I continued with the additions in the profile and dashboard sections. I disliked the dashboard section as it didn't have any content, so I decided to add some navigation for the user who has just logged in to his/her account. I added three cards, which include the navigation to the user profile and home page, as well as a button for logging out of the account. Here is the result with a similar style after the user logins in:



Welcome to Your Dashboard

Explore and manage your account

User Profile

View and update your profile information

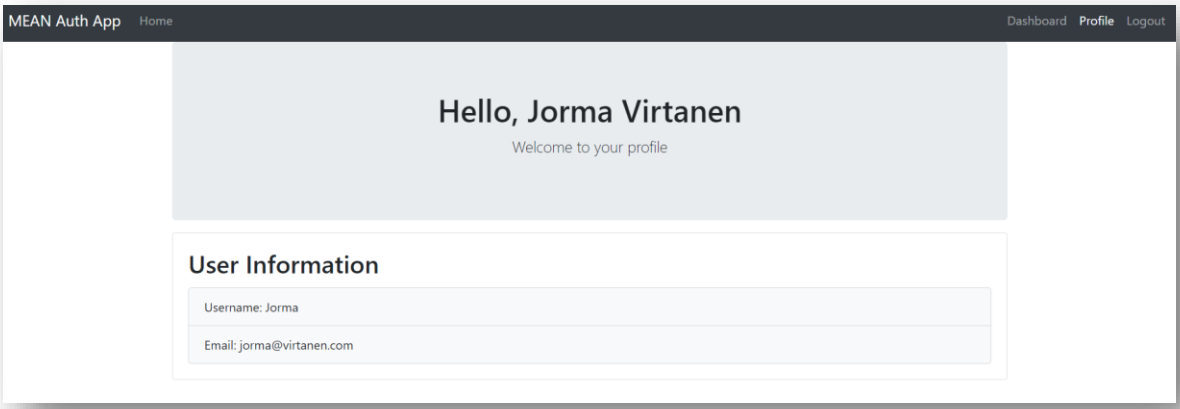
Home Page

Return to the home section

Logout Activity

Logout from your account

Lastly, the profile section needed some work. I wanted to keep it as a section, that includes only the user information but with a more welcoming message, so I modified the style and added a greeting:



Hello, Jorma Virtanen

Welcome to your profile

User Information

Username: Jorma

Email: jorma@virtanen.com

Overall, I modified all the sections to match the same style as on the homepage. I added more content to each section and added some typical navigation buttons to some of the components even though the top right corner navigation also works. All in all, I learned a lot about the relationship between the backend and frontend in this course. I got to combine lots of different environments to create a functional MEAN-stack project, which further improved my knowledge of how the environments work together. I encountered lots of problems but in the end, I gained a better understanding of the code and reasoning due to the problems. In my opinion, this project now represents my skills, and I’m sure that this will work as solid proof of my skills at the moment.