

Final Project

Web Applications

CT30A3203

Niilo Liimatainen

0523991

06.12.2021



Contents

1. Introduction.....	2
2. Technology choices	2
3. How to use it?.....	3
4. Requirements implemented	4
5. Reasoning for the implemented requirements.....	5
6. Notes.....	10

1. Introduction

This project is done as a final project to the course Web Applications. It is a full-stack application, where users can register, add posts with code snippets, and add comments to other posts. Users can for example ask code-related questions and seek help, or just ask opinions of code that the user has created. The application is fully responsive, so you can use it with any device. I have created a REST API with Node.js and Express. All the content is saved to MongoDB with the help of Mongoose. Frontend is made completely with the Angular framework. Front-end communicates to the backend only through REST API endpoints, which keeps the structure of the project clear. The front-end is inside the client folder and the backend is inside the server folder.

2. Technology choices

Node.js was mandatory to use, so it was a clear choice for the backend. I have used JavaScript a lot, so the familiar syntax was nice to use. MongoDB was also a clear choice for the database. The document data model suited well for this application, and with Mongoose, interacting with the database was efficient and easy. I also used Postman to create backend tests and initializer collection to the app.

For the frontend, I chose the Angular framework. I have worked as a web developer for 8 months now and Angular has been my main working tool, so I am somewhat experienced with it. Although it would have been nice to learn more about React with this project, I just couldn't get myself to do it. Angular has grown upon me and I wanted to do this project also with it. Angular provides a great and clear structure for the app, and two-way data binding and directives offer efficient ways to create a single-page application. Angular provides also a large set of material UI components and breakpoints, so there is no need for external CSS frameworks like Bootstrap. For unit testing, I used Jasmine and Karma, because Angular comes with direct support for them.

3. How to use it?

Comprehensive instructions to run this app can be found from the file README.md. However, I add some screenshots here.

Web Project (Angular/Node.js)

Terms used in this README

- backend - RESTful API implemented with Node.js, Express and MongoDB
- frontend - Single-page application implemented with Angular

Requirements

- Node.js atleast version `v12.17.0`
- npm (Node Package Manager) atleast version `7.14.0`
- Backend must run on localhost in order for frontend to show any content.

Getting started

1. Clone this repository to your own environment [repository](#)
2. Open terminal in the root folder
3. Run `npm run install:client`
4. Run `npm run install:server`
5. Run `npm run dev`
6. Optional: initialize server with Postman collection (instructions are down below)
7. Open browser of your choice and navigate to `http://localhost:4200/`

NPM Scripts

Run these from the root folder of this repository

- `install:client` > download and install all necessary dependencies to frontend
- `install:server` > download and install all necessary dependencies to backend
- `dev` > run backend and frontend concurrently
- `client` > run only frontend
- `server` > run only backend
- `test` > run unit tests for frontend

Tests

Unit tests

Angular unit tests are written with Jasmine and karma. Run tests from root folder with `npm run test` or from client folder with `ng test`

Postman

Server folder holds test folder which have Postman collection and Postman environment inside. These can be used to test the backend and to initialize the app. Postman collection adds users, entities, and comments to the database. If you want to see how the app looks and works with content inside it, you should run this collection.

How to run it:

1. Download Postman from (<https://www.postman.com/downloads/>)
2. Navigate to `server\test\postman` and import two files, `Server init.postman_collection.json` and `Server init.postman_environment.json`
3. Set `Server init.postman_environment.json` as an active environment
4. Run the collection

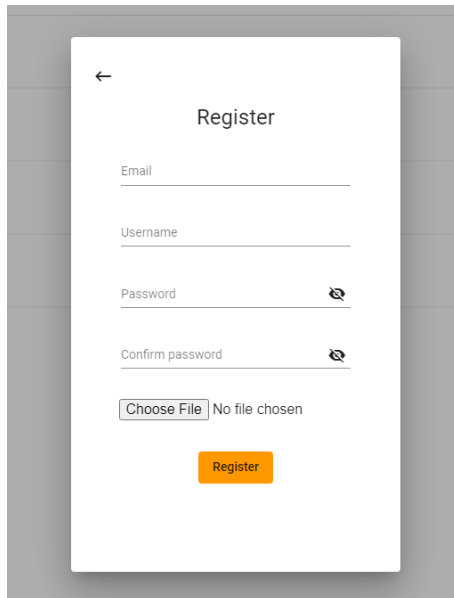
4. Requirements implemented

Feature	Max points
1. Basic features with well written documentation	25
2. Users can edit their own comments/posts	4
3. Utilization of a frontside framework, such as React, but you can also use Angular, Vue or some other	5
4. Use some highlight library for the code snippets, for example https://highlightjs.org/	2
5. Admin account with rights to edit all the post and comments and delete content (if a post is removed, all its comments should be removed too)	3
6. Vote (up or down) posts or comments (only one vote per user)	3
7. User profiles can have images which are show next to posts/comments	3
8. User can click username and see user profile page where name, register date, (user picture) and user bio is listed	2
9. Last edited timestamp is stored and shown with posts/comments	2
10. Create (unit) tests and automate some testing for example with https://www.cypress.io/ (at least 10 cases have to be implemented)	5
11. (Own feature) Loading indicator for user to see when the frontend is awaiting a response from the backend	2
12. (Own feature) Frontend tells user with snack bar if login/registration credentials are invalid	3
13. (Own feature) Postman collection and environment that can be used to test backend and to initialize the app with content	3
Points in total	62

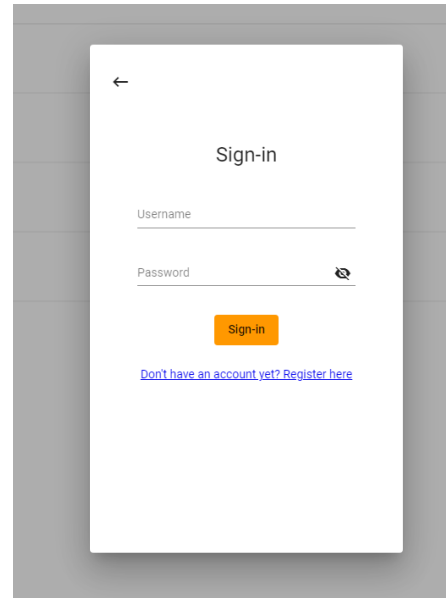
5. Reasoning for the implemented requirements

1. Basic features with well written document

- Backend is implemented with Node.js, Express and MongoDB.
- Users can register and login

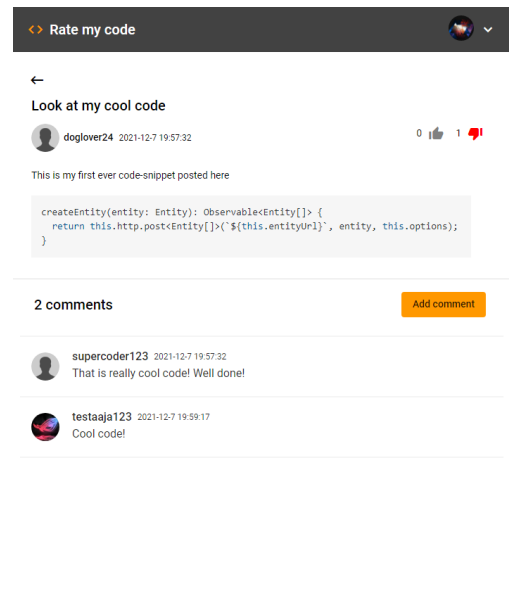
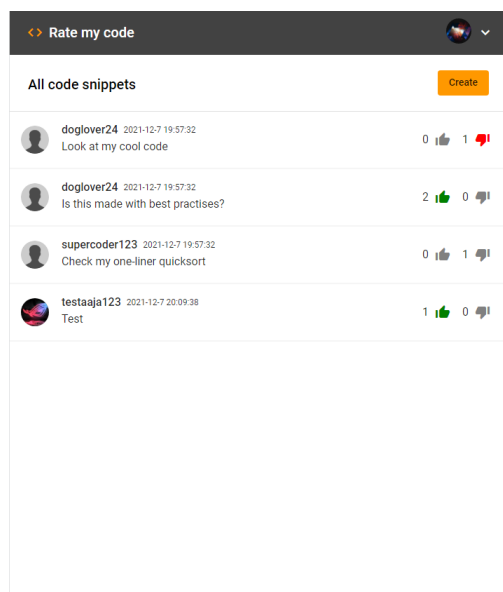


A screenshot of a mobile application's 'Register' screen. It features a white card with a grey header containing a back arrow and the title 'Register'. The form includes input fields for 'Email', 'Username', 'Password', and 'Confirm password', each with a toggle icon for visibility. Below these is a 'Choose File' button and the text 'No file chosen'. At the bottom is an orange 'Register' button.



A screenshot of a mobile application's 'Sign-in' screen. It features a white card with a grey header containing a back arrow and the title 'Sign-in'. The form includes input fields for 'Username' and 'Password', each with a toggle icon for visibility. Below these is an orange 'Sign-in' button and a blue link that says 'Don't have an account yet? Register here'.

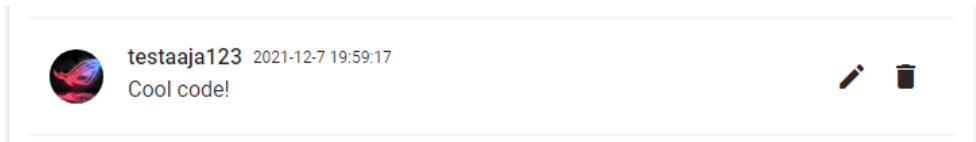
- Authentication is handled with the JWT and the Auth0 Angular SDK. After login jwt-token is stored to local storage and Auth0 adds it to every request automatically.
- Authenticated users can post new code snippets with text content and comment on existing posts (There are dedicated buttons for)
- Non-authenticated users can see all the posts, comments and vote counts
- Front page shows all posts that are created. If one post is clicked, the app opens the post with details and shows all the comments related to it.



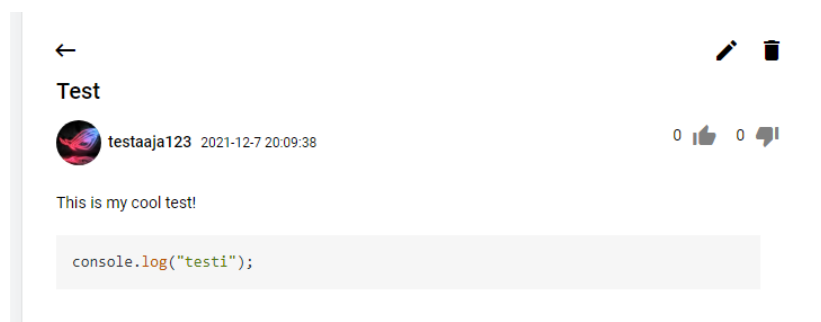
- The app is fully responsive. It can be used with any standard screen sizes and devices.
- Documentation describes reasons behind technology choices and instructions on how to use the app. In addition, README.md has been made comprehensively.

2. Users can edit their own comments/posts

- User can see edit and delete buttons on own comments



- When the user clicks the edit button, a dialog opens where the comment can be edited
- When the user clicks the delete button, opens confirm dialog. If user confirms the action, comment will be deleted
- When user opens their own post, edit and delete buttons will appear in the upper right



3. Utilization of a frontside framework, such as React, but you can also use Angular, Vue or some other

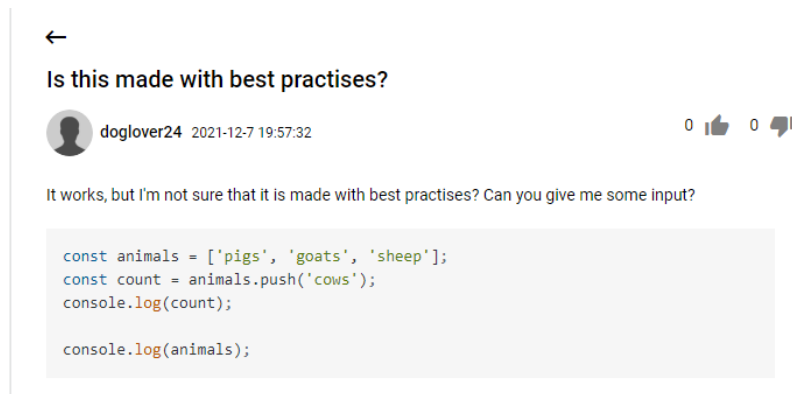
- Frontend is made with the Angular framework

4. Use some highlight library for the code snippets, for example

<https://highlightjs.org/>

- Highlight.js library is used to show the code snippets

- When the user opens a post, the code snippet will be shown in the following way

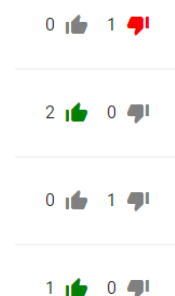


5. Admin account with rights to edit all the post and comments and delete content (if a post is removed, all its comments should be removed too)

- Database is always initialized with one admin account
- Admin account can see edit and delete button in all the posts and comments
- When post is deleted, all the comments under the post will also be deleted (works the same when user deletes their own post)

6. Vote (up or down) posts or comments (only one vote per user)

- User can vote only once one post
- Vote can't be later changed
- User can vote post from the front page or from the opened post
- Thumb up button means like and thumb down button means dislike
- When user has voted, vote will be shown with counter
- Every user can see the vote counts, but only authenticated users can vote
- User can identify own votes from the background color



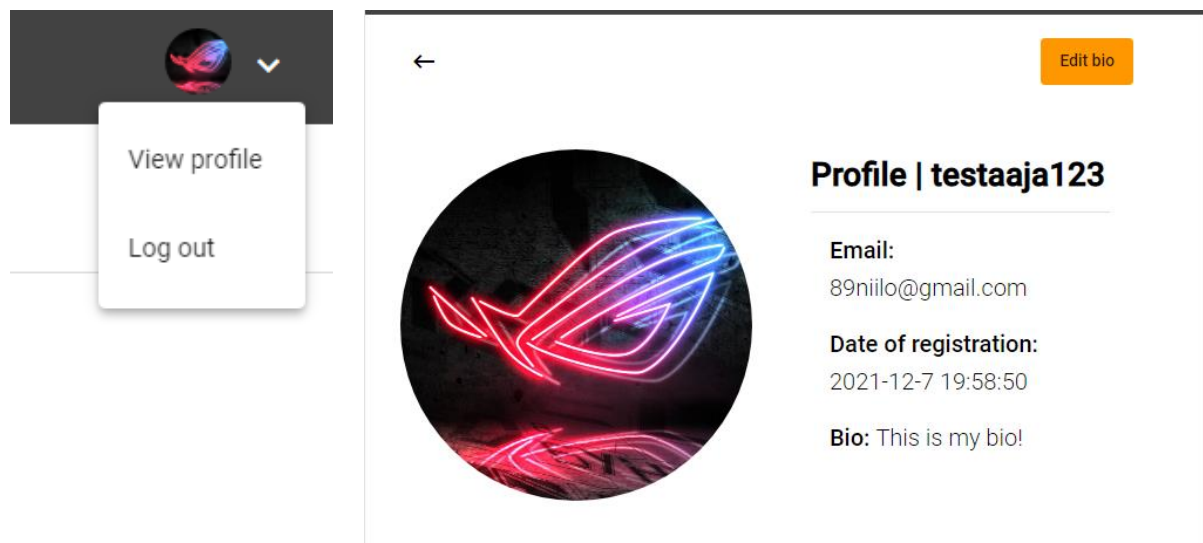
7. User profiles can have images which are show next to posts/comments

- Users can add profile picture upon registration
- If user doesn't have profile picture, the default profile picture will be used
- Profile pictures are shown next to posts and comments

8. User can click username and see user profile page where name, register date, (user picture) and user bio is listed

- Every user can click username or profile picture from post or comment and the app will navigate to the clicked user's profile page
- Profile page shows name register date, profile picture, bio, and email
- Users can view their own profile page from dropdown menu in the upper right corner

- If the user is on their own profile, there will be a button for adding or editing bio



9. Last edited timestamp is stored and shown with posts/comments

- Last edited timestamp is stored with comments and posts
- Timestamp will be updated every time when comment or post is modified
- Timestamp is shown with posts and comments in every view

10. Create (unit) tests and automate some testing for example with

<https://www.cypress.io/> (at least 10 cases have to be implemented)

- Unit tests are created for frontend with Jasmine and Karma
- Every component has at least a unit test to check that the component will compile
- In addition, NavigationComponent is tested thoroughly with unit tests
- NavigationComponent's unit tests can be found from the file 'navigation.component.spec.ts'
- NavigationComponent has a total of 13 unit tests so the requirements will be filled with only that component
- The app has a total of 31 unit tests
- Instructions to run these tests can be found from chapter 3 of this document

Karma v 6.3.7 - connected; test: complete;

Chrome 96.0.4664.45 (Windows 10) is idle

Jasmine 3.8.0

.....

31 specs, 0 failures, randomized with seed 36753

11. (Own feature) Loading indicator for user to see when the frontend is awaiting a response from the backend

- I have made a loading indicator for the app
- It can be found from 'shared/utils/indicator.ts'
- When the frontend is waiting response from the backend, user will see a loading indicator



12. (Own feature) Frontend tells user with snack bar if login/registration credentials are invalid

- App shows user error messages with snackbar upon registration and login
- In addition, forms will change their color to red, if the value inside is invalid
- In registration, user will get an accurate description of the problem. For example, if the email is already in use, it will be told to user
- Password's visibility can also be changed from the user interface

Email
fghfgh

Email already in use

close

13. (Own feature) Postman collection and environment that can be used to test backend and to initialize the app with content

- I have created a Postman collection and environment for testing the backend and initializing the app with content
- Collection and environment can be found from 'server/test/postman'
- Instructions to run these tests are shown in chapter 3 of this document



6. Notes

The code has tried to be written in a descriptive and informative way. Therefore, I have only commented on the more complex parts of the code. The code reviewer should keep in mind that the posts with code snippets are referenced as entities in the code. This is done to avoid any confusion between posts and post operations.