Mandatory Exercise 3 - Advanced JavaScript with React

The deadline for this exercise is Friday, February 14, 08:59.

For this **mandatory exercise** you should work on **master branch only**.

Preparation

1. Create a new repository on GitHub called **mandatory-advanced-js3**.
2. Follow the instructions that GitHub gives you; Create a local repository and add a remote or clone the newly created repository.

Submission

When you submit the exercise in PingPong, before the deadline, you will enter a link to your repository, such as:

https://github.com/mygithubusername/mandatory-advanced-js3

The teacher will look in the **master branch**. If any commits are done to the branch after the deadline, the teacher will look at the last commit before the deadline.

You will get one of the grades **G** or **IG**.

Instructions

In this exercise you will create a Todo application with user registration and user authentication. A backend server has been provided for the exercise.

With this application a user will be able to create a new user, sign in and manage a list of todo items.

The authentication will be done using **JSON Web Tokens** (JWT).

JSON Web Tokens

JSON Web Tokens is a standard used to issue secure access tokens.

Read more on
https://jwt.io/

JWT can be used to send information securely. A token contains an object (a payload) and is signed using a secret key. This makes it possible for the server to verify if the token is real or not without storing a copy of the token locally.

In this exercise the server will issue a token containing an email address when a user logs in. This token can be sent to the server using the **Authorization** header when making API calls.

The token should be sent like this:

```
Authorization: Bearer <token>
```

The **jsonwebtoken** module can be used to decode tokens.

```
import jwt from 'jsonwebtoken';
const decoded = jwt.decode(token);
console.log(decoded);
```

Backend server

A backend for this exercise is provided at

http://3.120.96.16:3002/

API endpoints

The backend has endpoints for user registration, authentication and managing a personal todo list. The API will accept both JSON and form data. To illustrate how the API is used, examples using **axios** are provided.

POST /register
Used to register a new user. Requires an **email** and a **password**. If no error occurs the API will respond with status code 201.

Example
```
axios.post(API_ROOT + '/register', { email, password });
```

POST /auth
Used to sign in. Requires an **email** and a **password**. If the email and password are valid the API will respond with status code 200 and issue a JWT token which expires in 1 hour.

Example
```
axios.post(API_ROOT + '/auth', { email, password });
```

GET /todos
Used to fetch a list of todos for a user. A JWT token must be sent with the request.

Example
```
axios.get(API_ROOT + '/todos', options);
```

POST /todos
Used to create a new todo. Requires an object with a "content" property.

Example
```
axios.post(API_ROOT + '/todos', { content: 'Water the plants' },
options);
```

DELETE /todos/:id
Used to delete a todo.

Example
```
axios.delete(API_ROOT + '/todos/' + id, options);
```

Views

The application should contain at least three different pages:

- A "registration page" with a form used to register a new user
- A "login page" with a form used to sign in
- A "todos page" with a list of todos and form to add a new todo

Every page should share a header which shows the email address of the currently logged in user. The email should be extracted from JWT token.

The header should also contain links to the login page, registration page and a button to sign out if the user is already signed in.

Registration page

The registration page should contain a form with input fields for email and password. When the form is submitted a new user should be added using the API.

An error message should be displayed if the API responds with an error.

Login page

The login page should contain a form with input fields for email and password. When the form is submitted the client receives a JWT token. Save the token in **localStorage** so the user is still signed in if the page is refreshed.

An error message should be displayed if the API responds with an error.

Todos page

The todo page should fetch a list of todos from the server and display them in a list. It should be possible to delete items from the list.

This page should also contain a form with a text input field used to add new todos.

Display error messages if the API returns an error.

Requirements

- The application should be an SPA written using React
- It should implement correct routing
- The user should stay signed in if the page is refreshed
- It should contain at least three views
    - Registration page
    - Login page
    - Todos page
- The email shown in the header should be extracted from the JWT token. Save only the token in localStorage

Tips

- Use **react-router** for routing
- Try to break the application into smaller, simpler parts
- Implement every page as a small application with its own state
- Try to help each other and ask questions if you get stuck
- Good luck!