Mandatory Exercise 2 - Advanced JavaScript with React

The deadline for this exercise is Friday, January 24, 08:59.

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

Preparation

1. Create a new repository on GitHub called **mandatory-advanced-js2**.
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-js2

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 movie directory application. The application should be written as a Single Page Application using React and communicates with a JSON REST API that has been provided.

REST API

The REST API is available at

3.120.96.16:3001

The API is used to add, remove, edit and list movie objects. A **movie object** has the following structure.

```
{
  “id”: “99a7d7ba-8660-4011-b98c-c927c5f0d34c”,
  “title”: “A title”,
  “description”: “A description”,
  “director”: “A director”,
  “rating”: 3.0,
}
```

The server does some validation on the objects

- The **title** must be between 1 and 40 characters
- The **description** must be between 1 and 300 characters
- The name of the **director** must be between 1 and 40 characters
- The **rating** must be a number between 0.0 and 5.0
- The **id** is created by the server and cannot be modified.

The following operations are supported

GET /movies
Returns a list of movies and responds with the status code 200.

GET /movies/:id
Returns a movie with a specific **id** and responds with the status code 200.

Will respond with a 404 status code if a movie with the supplied id does not exist.

POST /movies
Adds a new movie. The payload should be a JSON-encoded **movie object** and the correct Content-Type header must be used.

Will respond with the status code 400 if the object is invalid. If the object is successfully added it will respond with the status code 201.

PUT /movies/:id
Updates a movie with a specific **id**. The payload should be a JSON-encoded **movie object** and the correct Content-Type header must be used.

Will respond with a 404 status code if a movie with the supplied id does not exist. Will respond with a 400 status code if the object is invalid.

If the object is successfully updated it will respond with the status code 201.

DELETE /movies/:id
Deletes a movie with a specific **id**.

Will respond with a 404 status code if a movie with the supplied id does not exist. Will respond with a 204 status code if the object is successfully deleted.

Views

The application should contain four views:

- A **"main page"** that shows a table of movies
- An **"add page"** with a form that lets the user add a new movie
- An **"edit page"** with a form that lets the user edit a movie
- A **"details page"** that shows information about a movie

All pages should have a shared navigation with links to the "main page" and the "add page".

Main page

This page should display a table with all movies. The number of movies is limited to 20 so pagination is not necessary.

The table should display the **title**, **director** and **rating** for the movies in separate columns. Do not display the description in this table.

Each row in the table should have three buttons/links

- a button to delete the movie
- a link to the "edit page"
- a link to the "details page"

The main page should contain a text input that can be used to search for movies by **title or director**.

The filtering must be done on the client. The API does not support filtering.

Add page

This page is used to add new movies. The page should contain a form with the following inputs

- A text input for title
- A textarea for description
- A text input for director
- A number/range input for rating (some other input like a rating star input is also acceptable)

When the user submits the form one of two things can happen.

- If there is a validation error or some other error a suitable error message should be displayed
- If the object is successfully added, the user should be redirected to the “main page”.

Edit page

This page is used to edit movies. The page should work exactly like the “add page” but the form should be automatically populated with the current data.

Details page

This page shows information about a movie. It should display the **title**, **description**, **director** and **rating**.

The page should also contain a link to the “edit page”.

Routing

The application must implement correct routing and should contain at least four routes, one of every view.

The web browser history and refreshing the page must work correctly.

You must also dynamically change the title when the user navigates to a new page.

Requirements

- The application should be an SPA written using React
- It should implement correct routing
- It should implement four views
    - Main page
    - Add page
    - Edit page
    - Details page

- It should handle errors from the REST API correctly

Tips

- Use the **react-router** library for routing. It is very popular and has a great documentation
- Use the **react-helmet** library to dynamically change the title.
- Try to break the application into smaller, simple parts
- Implement every page as a small application with its own state
- Try to help each other and asks questions if you get stuck
- Good luck!