

Exercise: Single Page Application

Problems for exercises and homework for the ["JavaScript Apps" course @ SoftUni.](#)

Working with Remote Data

For the solution of some of the following tasks, you will need to use an up-to-date version of the **local REST service**, provided in the lesson's resources archive. You can [read the documentation here](#).

1. SoftForum

This task is to write functionality for a public forum. This app doesn't need any registration. Every user is able to read and write in the forum, and by creating a post or topic, the user should write also his/her own name.

SoftUni Forum
Home

New Topic

Title *

Username *

Post *

CancelPost

Angular 10

Date: 2020-10-10T12:08:28.451Z
Username: David

JS Advanced course

Date: 2020-04-10T15:08:28.431Z
Username: Johnny

This site is designed to be used for training purposes at SoftUni.
You can find the real SoftUni forum at this link:

Creating topic with post

On the home page there are all topics visible. Also there is a form for creating a new topic with a title, username and post (comment) in it. The button "Post" should create the new topic if all the fields are correctly filled up (and cleans up the input fields after that). The button "Cancel" only clears all the input fields, without sending any request to the server.

Use this URL to create topics: <http://localhost:3030/jsonstore/collections/myboard/posts>

The form is titled "New Topic" and is contained within a rounded rectangle. It has a blue header bar with the title "New Topic" in white. Below the header, there are three input fields: "Title *" (a single-line text input), "Username *" (a single-line text input), and "Post *" (a multi-line text area). At the bottom of the form, there are two buttons: "Cancel" and "Post", both with a blue gradient and white text.

Hint:

You can add an attribute with unique id to each post when creating it.

Comment on Topic

When the user selects a topic by clicking on it, the app should redirect to a page with all posts (comments) for the selected topic and a form for creating new post (comment).

The structure for the post should be as the following:

```
<div class="comment">
  <div class="header">
    
    <p><span>David</span> posted on <time>2020-10-10 12:08:28</time></p>

    <p class="post-content">Lorem ipsum dolor sit amet consectetur adipisicing elit. Iure facere sint
    dolorem quam,
    accusantium ipsa veniam laudantium inventore aut, tenetur quibusdam doloribus. Incidunt odio
    nostrum facilis ipsum dolorem deserunt illum?</p>
  </div>
</div>
```

After typing in the comment and sending the POST request to the server the page should render updated view with the new post.

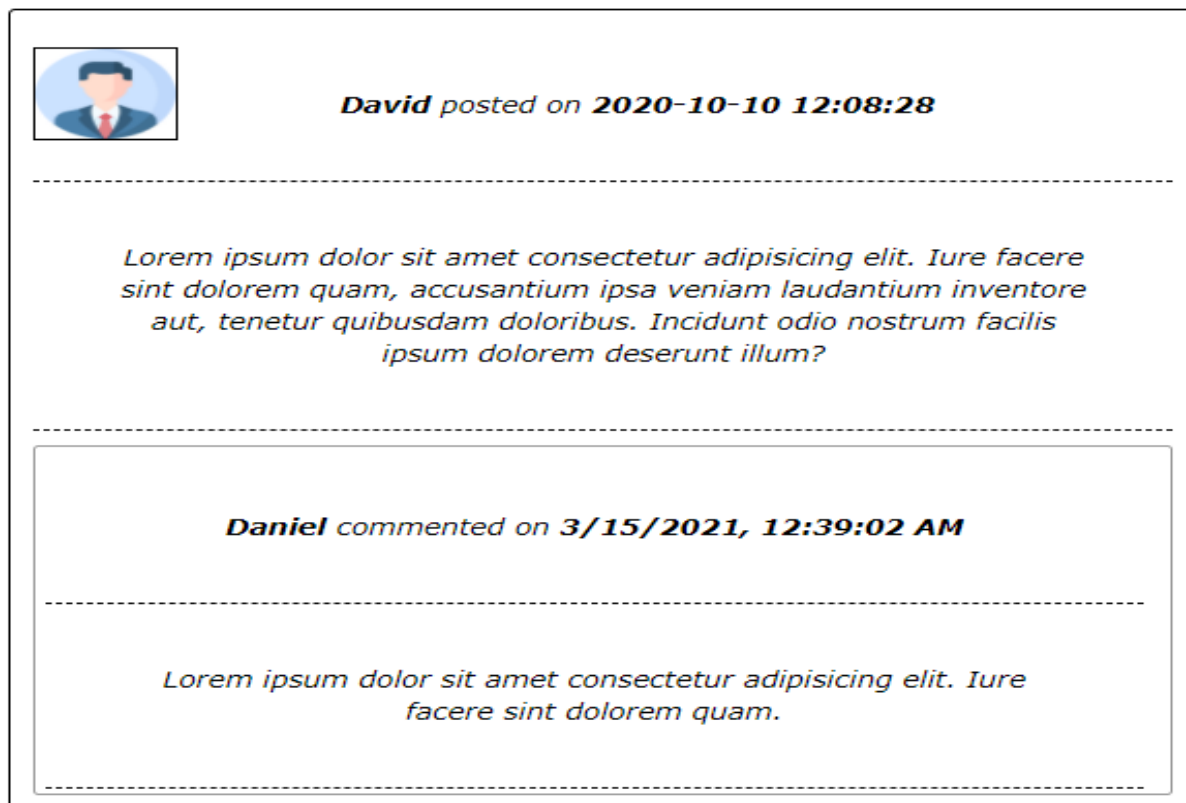
```
<div class="comment">
  <div class="header">
    
    <p><span>David</span> posted on <time>2020-10-10 12:08:28</time></p>

    <p class="post-content">Lorem ipsum dolor sit amet consectetur adipisicing elit. Iure facere sint
      dolorem quam,
      accusantium ipsa veniam laudantium inventore aut, tenetur quibusdam doloribus. Incidunt odio
      nostrum facilis ipsum dolorem deserunt illum?</p>
  </div>

  <div id="user-comment">
    <div class="topic-name-wrapper">
      <div class="topic-name">
        <p><strong>Daniel</strong> commented on <time>3/15/2021, 12:39:02 AM</time></p>
        <div class="post-content">
          <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Iure facere sint
            dolorem quam.</p>
        </div>
      </div>
    </div>
  </div>
</div>
```

You can use this URL: <http://localhost:3030/jsonstore/collections/myboard/comments>

Angular 10



currentUser comment:

Username *

Post

2. Movies – SPA

You are assigned to implement a **Single Page Web Application**. The app keeps **users** and **movies**. Logged-in users should be able to view **add movies** and **like movies**. Logged-in users should also be able to **edit** or **delete** the movies **they have added**.

REST Service

You can use the provided server to store your data. Use the following endpoints:

- Get all movies: `/data/movies` (GET)
- Create movie: `/data/movies` (POST)
- Update movie: `/data/movies/:id` (PUT)
- Delete movie: `/data/movies/:id` (DELETE)
- Get number of likes for a movie:
`/data/likes?where=movieId%3D%22{movieId}%22&distinct= ownerId&count` (GET)
- Get like for a movie from specific user:
`/data/likes?where=movieId%3D%22{movieId}%22%20and%20 ownerId%3D%22{userId}%22` (GET)
- Add a like: `/data/likes` (POST)
- Revoke a like: `/data/likes/:id` (DELETE)

Note that the provided service automatically limits the number of results from every request to 10. You may need to use pagination options, to get all entries.

HTML and CSS

You have been given the web design of the application as **HTML + CSS** files. Initially all views and forms are shown by the HTML. Your application may **hide/show elements** by CSS (**display: none**) or **delete/reattach** from and to the DOM all unneeded elements, or just display the views it needs to display.

Important: Don't change the elements' **class names** and **ids**. Don't rename form fields/link/ids. You are **allowed** to add **data attributes** to any elements. You may modify **href attributes** of links and add **action/method attributes** to **forms**.

Navigation Bar

Navigation links should correctly change the current page (view).

- Clicking on the links in the **NavBar** should display the view behind the navigation link.
- Your application may **hide/show elements** by CSS (**display: none**) or **delete/reattach** from and to the DOM all unneeded elements, or just display the views it needs to display.
- **The Logged-in** user navbar should contain the following elements: **[Movies]** a link to the **Home** page, the user caption ("Welcome, {email}"), **[Logout]**. Each link navigates to the named page.



- The guest users navbar should contain the following elements: **[Movies]** which is a link to the **Home** page and **[Login]**, **[Register]**.



Register User

By given **email** and **password**, the app should register a new user in the system.

- The following validations should be made:

- The **email** input must be **filled**
- The **password** should be **at least 6 characters** long
- The **repeat password** should be **equal to the password**
- After a **successful registration** the app should **redirect** to the **home page with the right navbar**.
- In case of **error** (eg. invalid username/password), an appropriate error **message** should be displayed, and the user should be able to **try** to register again.
- Keep the user data in the browser's **session or local storage**.
- After a **successful registration** redirect to **Home page**.

Movies Login Register

Email

Email

Password

Password

Repeat Password

Repeat-Password

Register

© 2020 JS Applications

Login User

By given **username** and **password**, the app should login an existing user.

- After a **successful login** the user **home page should be displayed**.
- In case of **error**, an appropriate error **message** should be displayed and the user should be able to fill in the login form again.
- Keep the user data in the browser's **session or locale storage**.
- After a **successful login** redirect to **Home page**.

Movies Login Register

Email

Email

Password

Password

Login

© 2020 JS Applications

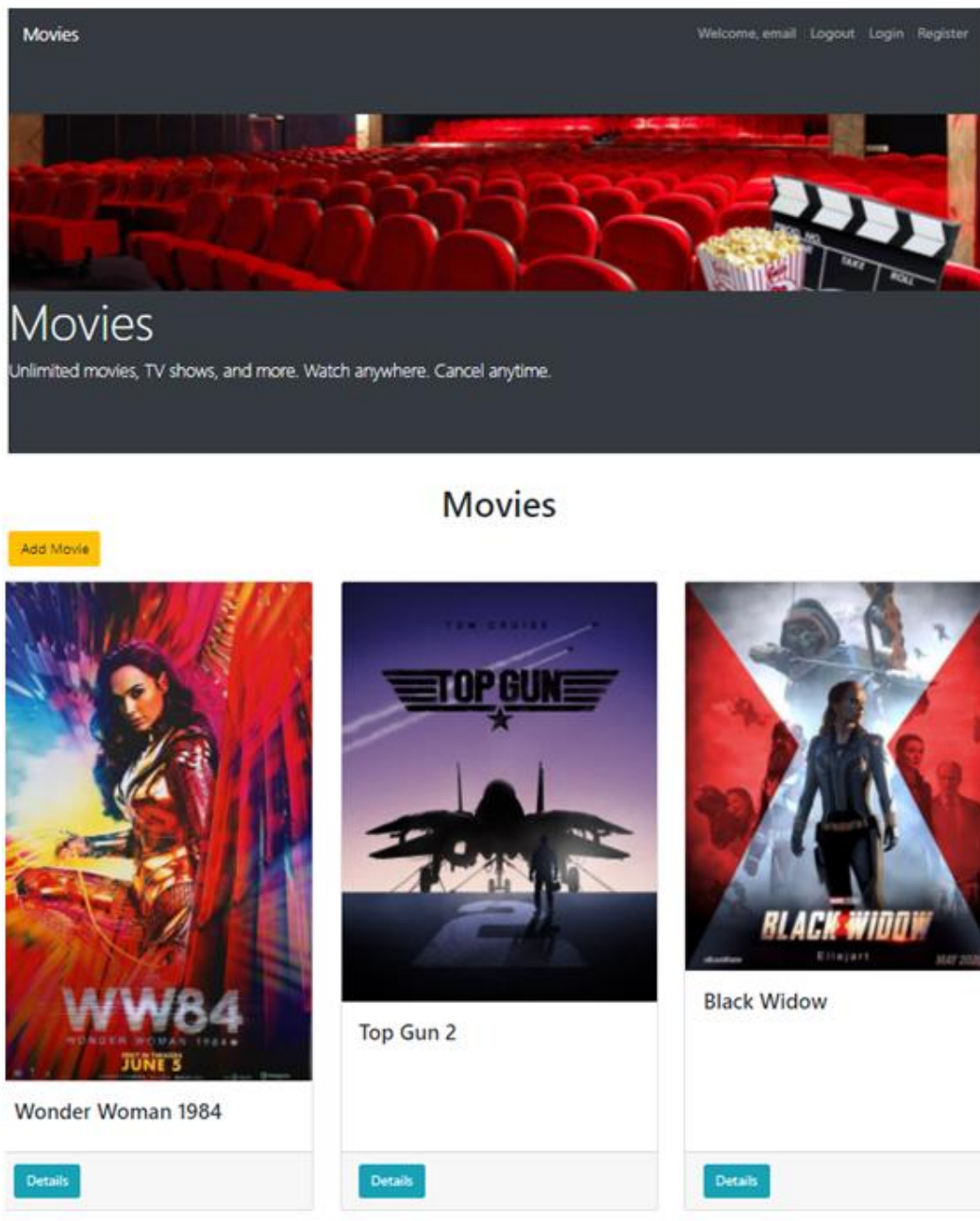
Logout

Successfully logged in users should be able to **logout** from the app.

- After a **successful** logout the **anonymous screen** should be shown
- The "**logout**" **REST service** at the back-end **must** be called at logout
- All local information in the browser (**user session data**) about the current user should be deleted
- After a **successful logout** redirect to **Login page**.

Home Page

Users should be welcomed by the **Home page**. They should be able to see all added movies:



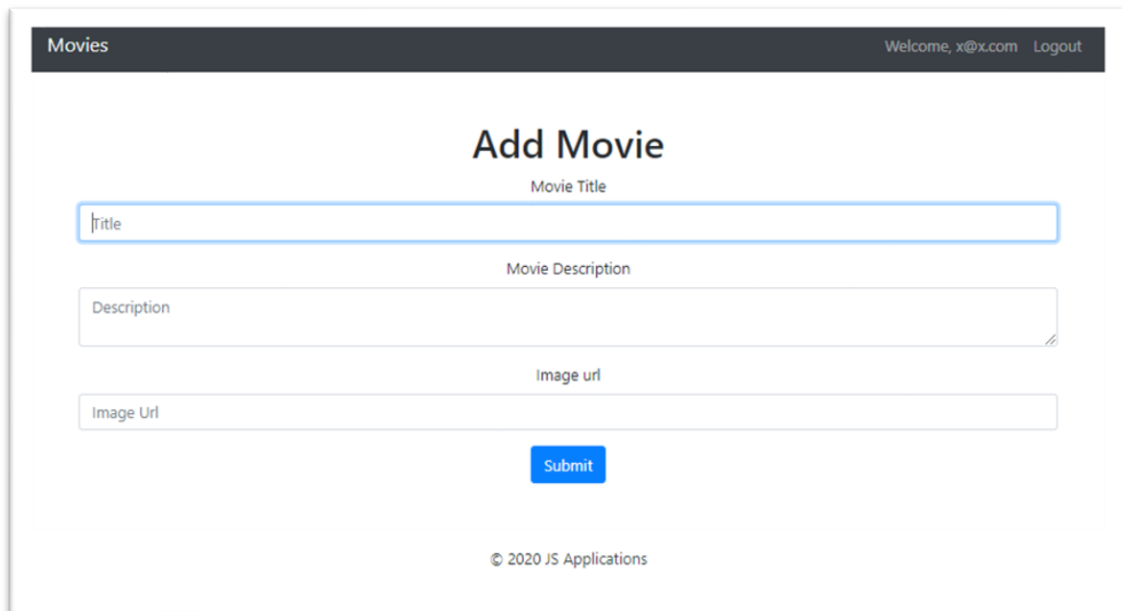
[Add Movie] button should refer to the **add movie form**. It should only be visible if there is a logged-in user.

Add Movie

Logged-in users should be able to **add movie**.

Clicking the **[Add Movie]** button in the **Home** page should **display** the **Add Movie** page.

- The form should contain the following validations:
 - The **title, description and image** shouldn't be **empty**.
 - After a **successful** movie adding the **Home** page should be shown.
- The newly added movie should be stored in the database collection "**movies**".



The screenshot shows a web application interface for adding a movie. At the top, there is a dark header bar with the word "Movies" on the left and "Welcome, x@x.com Logout" on the right. Below the header, the main content area has a title "Add Movie" in the center. Underneath the title, there are three input fields: "Movie Title" (with a placeholder "Title"), "Movie Description" (with a placeholder "Description"), and "Image url" (with a placeholder "Image Url"). Each input field has a light blue border. Below the input fields is a blue "Submit" button. At the bottom of the form, there is a copyright notice: "© 2020 JS Applications".

Trying to submit empty fields should **NOT** be possible.


Movie Details

Logged-in users should be able to **view details** about movies.

Clicking the **[Details]** button of a **particular movie** should **display** the **Movie Details** page.

- If the currently logged-in user is the creator of the movie, the **[Delete]** and **[Edit]** buttons should be shown, otherwise show the **[Like]** button.
- When displaying the number of likes a movies has, make sure to only count each user once!

Movie title: Black Widow



Movie Description


Natasha Romanoff aka Black Widow confronts the darker parts of her ledger when a dangerous conspiracy with ties to her past arises. Comes on the screens 2020.

Delete
Edit
Like
Liked 1

Movies

Welcome, xox@x.com Logout

Movie title: Black Widow



Movie Description

Natasha Romanoff aka Black Widow confronts the darker parts of her ledger when a dangerous conspiracy with ties to her past arises.

Like

© 2020 JS Applications

Edit Movie

Logged-in users should be able to **edit** movies, added by them.

Clicking the **[Edit]** button of a **particular movie** on the **Movie Details** page should **display** the **Edit Movie** page inserting the additional information of the movie in the input fields:

Movies

Welcome, x@x.com Logout

Edit Movie

Movie Title

Black Widow

Movie Description

Natasha Romanoff aka Black Widow confronts the darker parts of her ledger when a dangerous conspiracy with ties to her past arises.

Image url

https://miro.medium.com/max/735/1*akkAa2CcbKqHsvqVusF3-w.jpeg

Submit

© 2020 JS Applications

- After a **successful edit** the user should be redirected to the current movie **Details** page.

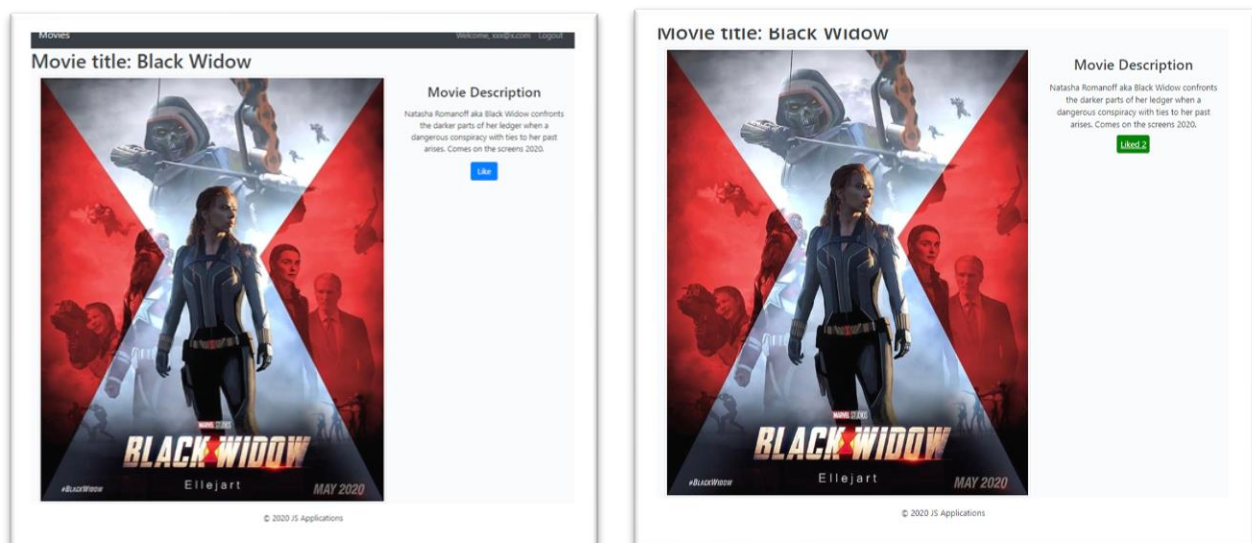
Like Movie

Logged-in users should be able to **like** movie, added by **other user**.

NOTE: A user should **NOT** be able to like a **movie**, created by **himself**.

Clicking the [**Like**] button of an **movie** (on the **Movie Details** page) should **make a POST request to the following URL:** <http://localhost:3030/data/likes> with body { movieId: id } (id – this is the id to the current movie) . After **successfully like**:

- [**Like**] button changes to [**Liked {number of likes}**] span so users can't like a **movie** multiple times.



Delete Movie

Logged-in users should be able to **delete their** movies.

Clicking the [**Delete**] button of an **movie** (on the **Movie Details** page) should **delete the movie**.

After **successful movie delete** the **Home** page should be shown