RESTful Movie List

This workshop guides you in building a dynamic web application that interacts with a server using **AJAX (Asynchronous JavaScript and XML)** via the modern **Fetch API**. You will use **JSON Server** to simulate a backend and practice the core **CRUD** (Create, Read, Update, Delete) operations, including a client-side search feature.

---

1. Backend Simulation: Setting up the JSON Server

The JSON Server acts as a simple, local REST API for your application.

1.1 Installation and Database Creation

Install JSON Server globally (terminal command):

Bash
```
npm install -g json-server
```

Create the Database File: Create a file named `movies.json`.

JSON
```
{
  "movies": [
    { "id": 1, "title": "Inception", "genre": "Sci-Fi", "year": 2010 },
    { "id": 2, "title": "Pulp Fiction", "genre": "Crime", "year": 1994 },
    { "id": 3, "title": "Dune", "genre": "Sci-Fi", "year": 2021 }
  ]
}
```

1.2 Launching the Server

Execute this command in your project directory:

Bash
```
json-server --watch movies.json
```

- **API Endpoint:** `http://localhost:3000/movies` (The base URL for all requests).

---

2. Frontend Structure: HTML (`index.html`)

This file provides the structure for the movie list, the new movie form, and the search input.

HTML
```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Movie Manager</title>
</head>
<body>
  <h1>Movie Collection Manager</h1>

  <form id="add-movie-form">
    <h3>Add New Movie</h3>
    <input type="text" id="title" placeholder="Title" required>
    <input type="text" id="genre" placeholder="Genre">
    <input type="number" id="year" placeholder="Release Year" required>
    <button type="submit">Add Movie</button>
  </form>

  <hr>

  <h2>Movie List</h2>

  <input type="text" id="search-input" placeholder="Search movies by title or genre">

  <div id="movie-list">
    </div>

  <script src="script.js"></script>
</body>
</html>
```

---

3. Application Logic: JavaScript (`script.js`)

This file manages the communication between the browser and the JSON Server using the **Fetch API**.

3.1 Setup and READ (GET Method)

JavaScript
```javascript
const API_URL = 'http://localhost:3000/movies';
const movieListDiv = document.getElementById('movie-list');
const searchInput = document.getElementById('search-input');
const form = document.getElementById('add-movie-form');

let allMovies = []; // Stores the full, unfiltered list of movies

// Function to dynamically render movies to the HTML
function renderMovies(moviesToDisplay) {
   movieListDiv.innerHTML = '';
   if (moviesToDisplay.length === 0) {
      movieListDiv.innerHTML = '<p>No movies found matching your criteria.</p>';
      return;
   }

   moviesToDisplay.forEach(movie => {
      const movieElement = document.createElement('div');
      movieElement.classList.add('movie-item');
      movieElement.innerHTML = `
         <p><strong>${movie.title}</strong> (${movie.year}) - ${movie.genre}</p>
         <button onclick="editMoviePrompt(${movie.id}, '${movie.title}', ${movie.year},
'${movie.genre}')">Edit</button>
         <button onclick="deleteMovie(${movie.id})">Delete</button>
      `;
      movieListDiv.appendChild(movieElement);
   });
}

// Function to fetch all movies and store them (READ)
function fetchMovies() {
   fetch(API_URL)
      .then(response => response.json())
      .then(movies => {
         allMovies = movies; // Store the full list
         renderMovies(allMovies); // Display the full list
      })
      .catch(error => console.error('Error fetching movies:', error));
}
fetchMovies(); // Initial load
```

3.2 Search Functionality

JavaScript

```
searchInput.addEventListener('input', function() {
   const searchTerm = searchInput.value.toLowerCase();
```

```
    // Filter the global 'allMovies' array based on title or genre match
    const filteredMovies = allMovies.filter(movie => {
        const titleMatch = movie.title.toLowerCase().includes(searchTerm);
        const genreMatch = movie.genre.toLowerCase().includes(searchTerm);

        return titleMatch || genreMatch;
    });

    renderMovies(filteredMovies); // Display the filtered results
});
```

3.3 CREATE Operation (POST Method)

JavaScript

```
form.addEventListener('submit', function(event) {
    event.preventDefault();

    const newMovie = {
        title: document.getElementById('title').value,
        genre: document.getElementById('genre').value,
        year: parseInt(document.getElementById('year').value)
    };

    fetch(API_URL, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(newMovie),
    })
    .then(response => {
        if (!response.ok) throw new Error('Failed to add movie');
        return response.json();
    })
    .then(() => {
        this.reset();
        fetchMovies(); // Refresh the list
    })
    .catch(error => console.error('Error adding movie:', error));
});
```

3.4 UPDATE Operation (PUT Method)

JavaScript

```
// Function to collect new data
function editMoviePrompt(id, currentTitle, currentYear, currentGenre) {
    const newTitle = prompt('Enter new Title:', currentTitle);
    const newYearStr = prompt('Enter new Year:', currentYear);
    const newGenre = prompt('Enter new Genre:', currentGenre);

    if (newTitle && newYearStr && newGenre) {
        const updatedMovie = {
            id: id,
            title: newTitle,
            year: parseInt(newYearStr),
            genre: newGenre
        };
        updateMovie(id, updatedMovie);
    }
}

// Function to send PUT request
function updateMovie(movieId, updatedMovieData) {
    fetch(`${API_URL}/${movieId}`, { // Target the specific resource by ID
        method: 'PUT',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify(updatedMovieData),
    })
    .then(response => {
        if (!response.ok) throw new Error('Failed to update movie');
        return response.json();
    })
    .then(() => {
        fetchMovies(); // Refresh list
    })
    .catch(error => console.error('Error updating movie:', error));
}
```

3.5 DELETE Operation (DELETE Method)

JavaScript

```
function deleteMovie(movieId) {
    fetch(`${API_URL}/${movieId}`, { // Target the specific resource by ID
        method: 'DELETE',
    })
    .then(response => {
        if (!response.ok) throw new Error('Failed to delete movie');
        fetchMovies(); // Refresh list
    })
    .catch(error => console.error('Error deleting movie:', error));
}
```

---

4. Testing and Verification

Step 5: Final Check

1. **Server Status:** Ensure `json-server --watch movies.json` is running.
2. **Access the App:** Open your **`index.html`** file in your browser.
3. **Verify All Operations:** Test the **Create (POST)**, **Read (GET)**, **Update (PUT)**, and **Delete (DELETE)** buttons/forms, confirming that the list updates on the page and the `movies.json` file changes on the server.
4. **Verify Search:** Use the search bar to filter the list instantly.