1. MovieService: Service-Based State Management

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root',
})
export class MovieService {
  private movies: any[] = [];

  constructor() {
    // Initialize movies from local storage
    this.getMovies();
  }

  getMovies() {
    const storedMovies = localStorage.getItem('movies');
    this.movies = storedMovies ? JSON.parse(storedMovies) : [];
    return this.movies;
  }

  addMovie(movie: any) {
    this.movies.push(movie);
    this.saveToLocalStorage();
  }

  updateMovie(id: number, updatedMovie: any) {
    const index = this.movies.findIndex((movie) => movie.id === id);
    if (index !== -1) {
      this.movies[index] = { ...this.movies[index], ...updatedMovie };
      this.saveToLocalStorage();
    }
  }

  deleteMovie(id: number) {
    this.movies = this.movies.filter((movie) => movie.id !== id);
    this.saveToLocalStorage();
```

```
  }

  private saveToLocalStorage() {
    localStorage.setItem('movies', JSON.stringify(this.movies));
  }
}
```

---

2. MovieListComponent: Display and Manage Favorite Movies

Template (HTML)

```
<div>
  <h2>Favorite Movies</h2>
  <ul>
    <li *ngFor="let movie of movies">
      <span>{{ movie.name }}</span>
      <button (click)="deleteMovie(movie.id)">Delete</button>
    </li>
  </ul>

  <h3>Add a Movie</h3>
  <form (ngSubmit)="addMovie()">
    <input
      type="text"
      placeholder="Movie name"
      [(ngModel)]="newMovieName"
      name="movieName"
      required
    />
    <button type="submit">Add</button>
  </form>
</div>
```

Component (TypeScript)

```
import { Component, OnInit } from '@angular/core';
import { MovieService } from './movie.service';
```

```
@Component({
  selector: 'app-movie-list',
  templateUrl: './movie-list.component.html',
  styleUrls: ['./movie-list.component.css'],
})
export class MovieListComponent implements OnInit {
  movies: any[] = [];
  newMovieName: string = '';

  constructor(private movieService: MovieService) {}

  ngOnInit() {
    this.movies = this.movieService.getMovies();
  }

  addMovie() {
    const newMovie = {
      id: Date.now(), // Unique ID for the movie
      name: this.newMovieName,
    };
    this.movieService.addMovie(newMovie);
    this.newMovieName = ''; // Reset input field
    this.movies = this.movieService.getMovies(); // Refresh movie list
  }

  deleteMovie(id: number) {
    this.movieService.deleteMovie(id);
    this.movies = this.movieService.getMovies(); // Refresh movie list
  }
}
```

---

3. Module Updates

Add FormsModule for Two-Way Binding

Ensure `FormsModule` is imported in your `app.module.ts`:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { MovieListComponent } from
'./movie-list/movie-list.component';

@NgModule({
  declarations: [AppComponent, MovieListComponent],
  imports: [BrowserModule, FormsModule],
  providers: [],
  bootstrap: [AppComponent],
})
export class AppModule {}
```

---

4. Running the Application

When you run this app:

1. You can add movies to your favorite list via the input form.
2. The list of movies will persist in **local storage**, even after refreshing the page.
3. You can delete movies, and the local storage will update automatically.

Overview

We built an Angular app to manage a list of favorite movies. The app allows you to:

1. Add movies to a list.
2. See the list of movies displayed on the screen.
3. Delete movies from the list.
4. Save the list to the browser's **local storage** so it stays even after you refresh the page.

This example also introduces the concept of **state management** using a **service**, which helps keep our app organized by separating data (state) logic from the component that displays it.

Step-by-Step Breakdown

1. Created a Service to Manage the Movie Data

A service in Angular is like a "helper" that stores and manages data and logic. This is useful because:

- It keeps all the movie-related functionality (e.g., adding, deleting, updating) in one place.
- It allows us to share the same movie data across multiple components if needed.

Here's what we did in the service:

1. **Stored the Movie Data:**
    - We used an array `movies` to store the list of movies.
    - This array starts empty when the app runs, but we load any saved movies from **local storage**.
2. **Added Methods to Manage the Movies:**
    - `getMovies()`: Gets the list of movies (loads from local storage if available).
    - `addMovie(movie)`: Adds a new movie to the list.
    - `deleteMovie(id)`: Removes a movie by its unique `id`.
    - `saveToLocalStorage()`: Saves the current list of movies to the browser's **local storage**.
3. **Local Storage for Persistence:**
    - **What is local storage?**
        - A small, built-in storage space in your browser where you can save data.
        - It doesn't get erased when you refresh the page or close the browser (unless you clear it manually).
    - **Why use it?**
        - We used local storage to ensure that the favorite movies list is saved across sessions. If the user adds movies and refreshes the page, the list remains the same.

Here’s an example from the service:

```
getMovies() {
  const storedMovies = localStorage.getItem('movies'); // Get saved
movies from local storage
  this.movies = storedMovies ? JSON.parse(storedMovies) : []; // Parse
the JSON string or use an empty array
  return this.movies; // Return the movies
}
```

This checks if there’s already a list of movies saved and loads it when the app starts.

---

2. Created a Component to Display and Interact with the Movies

The component is where we show the list of movies and allow users to interact with it (e.g., adding and deleting movies).

1. **Getting Movies from the Service:**
    - When the component loads, it calls the `getMovies()` method from the service to get the list of movies and display them.
2. **Adding a New Movie:**
    - The user types the movie's name in a text input field and clicks the "Add" button.
    - The component calls the `addMovie()` method in the service to add the new movie.
    - After adding, it updates the list to show the new movie on the screen.
3. **Deleting a Movie:**
    - When the user clicks the "Delete" button for a movie, the component calls the `deleteMovie()` method in the service to remove it from the list.
    - The list is refreshed to show the updated data.
4. **Two-Way Binding with `[(ngModel)]`:**
    - This allows us to bind the input field to a variable in the component (`newMovieName`).
    - Whatever the user types in the input is automatically stored in the variable.

Here’s an example from the component:

```
addMovie() {
  const newMovie = {
```

```
    id: Date.now(), // Unique ID for the movie
    name: this.newMovieName, // Movie name from the input field
  };
  this.movieService.addMovie(newMovie); // Add the movie using the
service
  this.newMovieName = ''; // Clear the input field
  this.movies = this.movieService.getMovies(); // Refresh the movie
list
}
```

---

3. The Template (HTML)

The HTML template is what the user sees and interacts with. It contains:

1. **A List of Movies:**
    - The `<ul>` element lists all the movies using `*ngFor`, which loops through the `movies` array and displays each movie.

Example:

```
<ul>
  <li *ngFor="let movie of movies">
    <span>{{ movie.name }}</span>
    <button (click)="deleteMovie(movie.id)">Delete</button>
  </li>
</ul>
```

2.
3. **A Form to Add Movies:**
    - The form contains an input field for typing the movie name and a button to submit the form.
    - The `[(ngModel)]` binds the input field to the `newMovieName` variable in the component.

Example:

```
<form (ngSubmit)="addMovie()">
  <input type="text" placeholder="Movie name"
[(ngModel)]="newMovieName" name="movieName" required />
  <button type="submit">Add</button>
```

```
</form>
```

4.
5. **A Delete Button:**
    - Each movie has a "Delete" button that calls the `deleteMovie()` method when clicked.

---

4. Putting It All Together

1. **Service:** Handles all the data and saves it to local storage.
2. **Component:** Displays the list, gets data from the service, and allows the user to add or delete movies.
3. **Template:** Provides the structure and interaction for the user interface.