Project Documentation

Online Movie Site Using MEAN Stack

By: Youssef Safwat - Yehia Ragab - Marwan Wael

Mahmoud Elhefnawy - Sherif Mohy

MEAN Stack Internship (ITI)
1 Aug - 30 Aug

Summary:

The rapid growth of online streaming platforms has made it challenging for users to efficiently discover, track, and explore films tailored to their preferences. Manual methods of managing watchlists and searching for movies often lead to inefficiencies and a fragmented user experience. To address this, we developed a full-stack web application using the MEAN stack (MongoDB, Express.js, Angular, and Node.js). The platform offers a dynamic and responsive interface where users can browse popular and trending movies, view detailed information including ratings and genres, maintain a personalized watchlist,and search films intuitively. Key features include user authentication, interactive movie cards with hover effects, paginated listings, recommendation sections, and a search functionality with real-time suggestions. This system not only enhances movie discovery and organization but also provides a seamless, modern user experience comparable to commercial streaming platforms. The result is a scalable, performant, and user-friendly application that demonstrates the power of the MEAN stack in building realworld solutions.

Introduction :

Background:

The entertainment industry has seen a significant shift toward digital streaming, with audiences increasingly relying on online platforms to discover, explore, and manage movies. This trend highlights the need for dynamic, user-friendly web applications that offer personalized movie recommendations, efficient search capabilities, and easy access to film details. The MEAN stack (MongoDB, Express.js, Angular, Node.js) provides a modern, JavaScript-based full-stack solution ideal for building scalable and interactive web applications like movie databases.

Problem Statement:

Many existing movie websites lack intuitive user experiences, responsive design, and personalized features such as watchlists or intelligent recommendations. Users often struggle with clunky interfaces, limited filtering options, and no ability to save or organize films for later viewing. This project aims to solve these issues by delivering a seamless, feature-rich platform that enhances how users interact with movie content online.

Objectives:

The main goals of this project are:

- To design and develop a full-stack web application using the MEAN stack.

- To implement user-friendly features such as a dynamic movie browse page, detailed film information, and a personalized watchlist.

- To ensure a responsive and modern UI with interactive elements like hover effects and real-time search.

- To demonstrate the integration of frontend and backend technologies for a smooth and scalable user experience.

Tools & Technologies:

> **MongoDB**

A NoSQL database used to store application data in flexible, JSON-like documents.

Usage in this project: Stores movie details, user information, watchlists, and ratings in collections such as movies, users, and watchlists.

> **Express.js**

A backend web application framework for Node.js designed to build APIs and web servers.

Usage in this project: Handles HTTP requests, including routing for movie data, user authentication, and watchlist management via RESTful endpoints.

> **Angular**

A frontend framework for building dynamic, single-page web applications.

Usage in this project: Creates responsive and interactive user interfaces for browsing movies, viewing details, managing watchlists, and searching content.

> **Node.js**

A JavaScript runtime environment used to execute server-side code.

Usage in this project: Powers the backend server, manages API logic, and connects the database with the frontend through Express.js.

> **Other Tools**

GitHub: Version control and collaborative development.

Postman: Testing API endpoints during development.

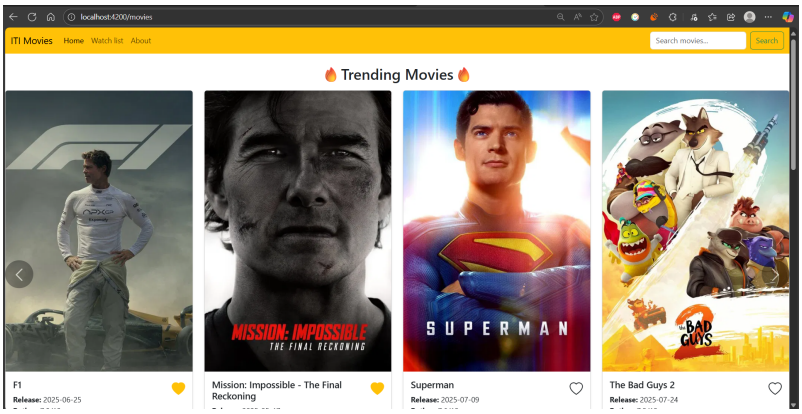
VS Code: Source code editing with extensions for Angular and Node.js.

Angular CLI: Scaffolding and managing the Angular application.

Mongoose: ODM (Object Data Modeling) library for MongoDB and Node.js.

System Design:

1.1 Main page with the favorite active:



The main page displays a grid of popular movies and allows users to mark films as favorites.

This feature is implemented using Angular for the front-end logic and Bootstrap for styling.

Implementation Details:

1. Data Structure and Initialization:

Movies are fetched from The Movie Database (TMDB) API. Each movie object is extended with a liked property to track its favorite status locally:

```
const limitedMovies = res.results.slice(0, 6).map((m: any) => ({ ...m, liked: false }));
this.movies.set(limitedMovies);
```

> liked: false initializes all movies as not favorited.

2. Toggle Logic:

The toggleLike() method handles user interactions

```
toggleLike(movie: any): void {
  movie.liked = !movie.liked;
}
```

> This function toggles the liked property between true and false when triggered.

3. UI and Interaction

The heart icon dynamically changes based on the liked state using Angular's ngClass

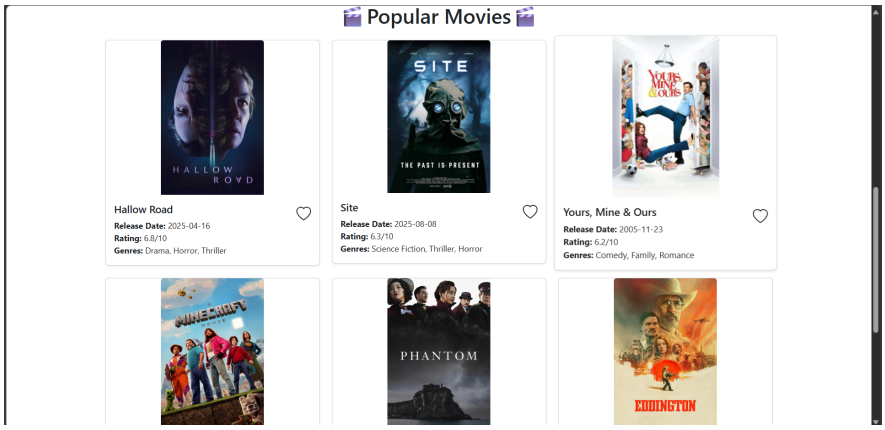
```
<i
  class="bi"
  [ngClass]="movie.liked ? 'bi-heart-fill text-warning fs-3' : 'bi-heart fs-3'"
  (click)="toggleLike(movie)"
  style="cursor: pointer; float: right;"
></i>
```

> bi-heart-fill text-warning: Filled yellow heart (favorited).

> bi-heart: Outline heart (not favorited).

> (click)="toggleLike(movie)"": Calls the toggle function on click.

1.2 Main page the popular movies:



The main page features a responsive grid of popular movies fetched from The Movie Database (TMDB) API, complete with pagination controls and interactive elements.

Implementation Details:

1. Data Fetching and Pagination

The component initializes by loading genres and popular movies from the TMDB API

```
ngOnInit(): void {
  this.loadGenres().subscribe({
    next: (response) => {
      response.genres.forEach((g: any) => this.genresMap.set(g.id, g.name));
      this.loadMovies(1);
    },
    error: (err) => console.error('Error loading genres:', err)
  });
}
```

> Genre Mapping: Fetches and maps genre IDs to names for display.

> Pagination: Limits results to 6 movies per page and calculates visible page numbers.

2. Template Display

The UI renders movie cards with posters, titles, release dates, ratings, and genres

```
<h1 class="mt-4 mb-3 text-center">Popular Movies</h1>
<div class="container mt-4">
  <div class="row">
    <div class="col-md-4" *ngFor="let movie of movies()">
      <div class="card mb-4 shadow-sm">
        <img
          [src]="'https://image.tmdb.org/t/p/w500' + movie.poster_path"
          class="card-img-top"
          alt="{{ movie.title }}" />
        <div class="card-body">
          <h5 class="card-title">{{ movie.title }}</h5>
          <p class="card-text">
            <strong>Release Date:</strong> {{ movie.release_date }} <br>
            <strong>Rating:</strong> {{ movie.vote_average | number:'1.1-1' }}/10 <br>
            <strong>Genres:</strong> {{ getGenresForMovie(movie.genre_ids) }}
          </p>
          <button class="btn btn-primary" [routerLink]="['/movie-details', movie.id]">
            More Details
          </button>
        </div>
      </div>
    </div>
  </div>
</div>
```

> Image Binding: Uses TMDB's image API to display movie posters.

> Genre Display: Calls getGenresForMovie() to convert genre IDs to names.

> Navigation: Routes users to movie details pages on button click.

1.2 Main page the popular movies (continued):

3. Pination Controls (movies.html)

Users can navigate between pages using the pagination bar

```
<nav class="d-flex justify-content-center">
  <ul class="pagination">
    <li class="page-item" [class.disabled]="currentPage === 1">
      <a class="page-link" (click)="loadMovies(currentPage - 1)">Previous</a>
    </li>
    <li class="page-item" *ngFor="let page of visiblePages" [class.active]="page === currentPage">
      <a class="page-link" (click)="loadMovies(page)">{{ page }}</a>
    </li>
    <li class="page-item" [class.disabled]="currentPage === totalPages">
      <a class="page-link" (click)="loadMovies(currentPage + 1)">Next</a>
    </li>
  </ul>
</nav>
```

> Dynamic Disabling: Previous/Next buttons disable at boundaries.

> Active Page Highlighting: Current page is visually emphasized.

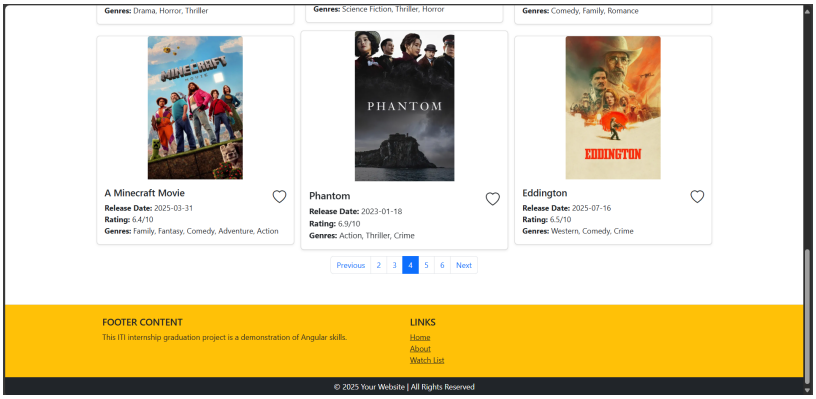
Key Features:

>Responsive Design: Bootstrap grid system adapts to screen size.

>API Integration: Real-time data fetching from TMDB.

>User-Friendly Navigation: Intuitive pagination and routing.

3.1 main page the poplur movies with the page navigate:



The main page features a dynamic display of popular movies with full pagination support, allowing users to navigate through multiple pages of movie results.

Implementation Details:

1. Pagination Logic

The component manages pagination state and calculates visible page numbers

```
currentPage = 1;
totalPages = 100;
visiblePages: number[] = [];

// Load movies for specific page
loadMovies(page: number): void {
  if (page < 1 || page > this.totalPages) return;
  this.currentPage = page;

  this.getMovies(page).subscribe({
    next: (res) => {
      const limitedMovies = res.results.slice(0, 6).map((m: any) => ({ ...m, liked: false }));
      this.movies.set(limitedMovies);
      this.totalPages = res.total_pages;
      this.updateVisiblePages();
    },
    error: (err) => console.error('Error loading movies:', err)
  });
}

// Calculate visible page numbers
updateVisiblePages(): void {
  let start = Math.max(1, this.currentPage - 2);
  let end = Math.min(this.totalPages, this.currentPage + 2);
  this.visiblePages = [];
  for (let i = start; i <= end; i++) {
    this.visiblePages.push(i);
  }
}
```

2. Pagination UI

The template displays navigation controls with dynamic page numbering

```
<nav class="d-flex justify-content-center">
  <ul class="pagination">
    <!-- Previous Button -->
    <li class="page-item" [class.disabled]="currentPage === 1">
      <a class="page-link" (click)="loadMovies(currentPage - 1)">Previous</a>
    </li>

    <!-- Page Numbers -->
    <li class="page-item" *ngFor="let page of visiblePages" [class.active]="page === currentPage">
      <a class="page-link" (click)="loadMovies(page)">{{ page }}</a>
    </li>

    <!-- Next Button -->
    <li class="page-item" [class.disabled]="currentPage === totalPages">
      <a class="page-link" (click)="loadMovies(currentPage + 1)">Next</a>
    </li>
  </ul>
</nav>
```

3. Movie Data Fetching

API call to retrieve paginated movie data

```
getMovies(page: number = 1): Observable<any> {
  return this.http.get<any>(`${this.apiUrl}/movie/popular`, {
    params: {
      api_key: this.apiKey,
      page: page.toString(),
      language: 'en-US'
    }
  });
}
```

How the Pagination Works:

1-Initial Load: The component loads page 1 of popular movies on initialization.

2-Page Navigation: >Users can click numbered buttons to jump to specific pages
>Previous/Next buttons allow sequential navigation
>The current page is highlighted with active styling

3-Dynamic Page Range: Only 5 page numbers are shown at a time (current page ±2) for clean navigation

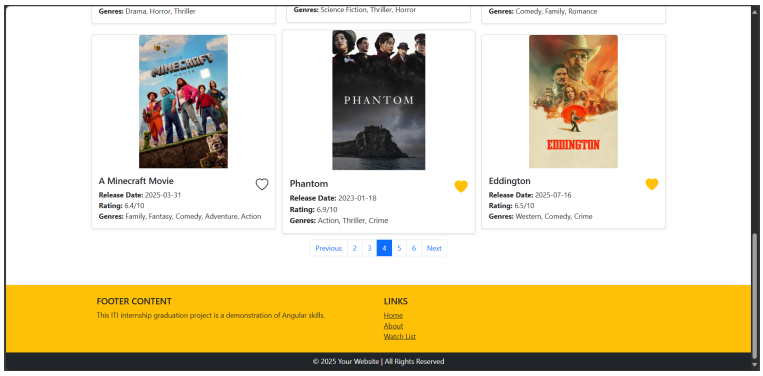
4-Button States: Previous/Next buttons are disabled when at the first/last page

5-Data Refresh: Each page click triggers a new API call to fetch the corresponding movie data

Key Features:

> Responsive Pagination: Adapts to different screen sizes
> Efficient Data Loading: Only loads 6 movies per page for optimal performance
> User-Friendly Navigation: Intuitive page controls with visual feedback
> Dynamic Page Calculation: Smart page number display based on current position

3.2 main page when hover on card gets bigger:



The main page features an interactive hover effect where movie cards enlarge when users hover over them, providing visual feedback and enhancing user engagement.

Implementation Details:

1. HTML Template Structure

The movie cards are structured using Bootstrap classes within an Angular *ngFor loop

```
<div class="col-md-4" *ngFor="let movie of movies()">
  <div class="card mb-4 shadow-sm">
    <img
      [src]="'https://image.tmdb.org/t/p/w500' + movie.poster_path"
      class="card-img-top"
      alt="{{ movie.title }}" />
    <div class="card-body">
      <!-- Card content including title, rating, and buttons -->
    </div>
  </div>
</div>
```

2. CSS Hover Effects

The hover animation is implemented using CSS transitions and transform properties

```
/* Base card styling with transition */
.card {
  transition: transform 0.3s ease, box-shadow 0.3s ease;
  transform-origin: center;
  border: 1px solid #dee2e6;
  cursor: pointer;
}

/* Hover effects */
.card:hover {
  transform: scale(1.05);
  box-shadow: 0 8px 25px rgba(0, 0, 0, 0.15);
  z-index: 10;
  border-color: #ffc107;
}

/* Image enhancement on hover */
.card-img-top {
  transition: transform 0.3s ease;
}

.card:hover .card-img-top {
  transform: scale(1.02);
}
```

How the Hover Effect Works:

1-Normal State: Cards display at regular size (scale: 1.0) with a standard Bootstrap shadow.

2-Hover Trigger: When users move their cursor over any movie card.

3-Scale Transformation: The card smoothly enlarges by 5%(transform: scale(1.05)) over 0.3 seconds.

4-Enhanced Shadow:The box-shadow intensifies to create depth and elevation.

5-Image Enhancement: The movie poster image scales slightly (1.02x) for added visual effect.

6-Layer Management: The z-index increases to ensure the hovered card appears above adjacent elements.

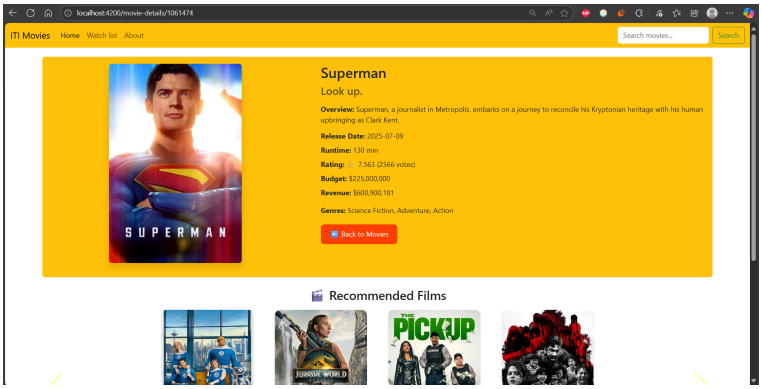
7-Visual Feedback: Border color changes to yellow (#ffc107) to highlight the active card.

8-Smooth Transition: All transformations animate smoothly using CSS ease timing function.

Key Features:

>Subtle Animation: 5% scale increase provides noticeable but non-disruptive visual feedback.
>Performance Optimized: Uses CSS transforms and GPU acceleration for smooth animations.
>Visual Hierarchy: Enhanced shadow and z-index create a sense of elevation and importance.
>Responsive Design: Effect works consistently across various screen sizes and devices.
>User Experience: Clearly indicates interactive elements and provides immediate feedback to user actions.

4.1 more details page:



The movie details page provides comprehensive information about a selected film, including overview, ratings, financial data, and genres, offering users detailed insights into the movie.

Implementation Details:

1. Component Logic

The component retrieves movie details from TMDB API based on route parameters

```
export class MovieDetails implements OnInit {
  private apiKey = '072e8885c0871676d8eadaee59eb40ec';
  private apiUrl = `https://api.themoviedb.org/3`;

  movie = signal<any>(null);

  constructor(private route: ActivatedRoute, private http: HttpClient) {}

  ngOnInit(): void {
    const id = this.route.snapshot.paramMap.get('id');
    if (id) {
      this.loadMovieDetails(id);
    }
  }

  loadMovieDetails(id: string): void {
    this.http.get<any>(`${this.apiUrl}/movie/${id}`, {
      params: { api_key: this.apiKey, language: 'en-US' }
    }).subscribe({
      next: (res) => this.movie.set(res),
      error: (err) => console.error('Error loading movie details:', err)
    });
  }
}
```

2. Template Structure

The template displays comprehensive movie information in a organized layout

```
<div class="movie-details-container bg-warning d-flex flex-wrap p-3 rounded shadow-sm">
  <div class="poster me-4 mb-3">
    <img
      [src]="'https://image.tmdb.org/t/p/w500' + movie().poster_path"
      [alt]="movie().title"
      class="img-fluid rounded shadow">
  </div>
  <div class="info flex-grow-1">
    <h1 class="mb-2">{{ movie().title }}</h1>
    <h4 class="text-muted mb-3">{{ movie().tagline }}</h4>
    <p><strong>Overview:</strong> {{ movie().overview }}</p>
    <ul class="list-unstyled mb-3">
      <li><strong>Release Date:</strong> {{ movie().release_date }}</li>
      <li><strong>Runtime:</strong> {{ movie().runtime }} min</li>
      <li><strong>Rating:</strong> ⭐ {{ movie().vote_average }} ({{ movie().vote_count }} votes)</li>
      <li><strong>Budget:</strong> ${{ movie().budget | number }}</li>
      <li><strong>Revenue:</strong> ${{ movie().revenue | number }}</li>
    </ul>
    <p>
      <strong>Genres:</strong>
      {{ movie().genres[0]?.name }}, {{ movie().genres[1]?.name }}, {{ movie().genres[2]?.name }}
    </p>
      <button routerLink="" class="btn btn-primary mt-2">← Back to Movies</button>
  </div>

</div>
<app-recommended-films [movieId]="movie()?.id"></app-recommended-films>
```

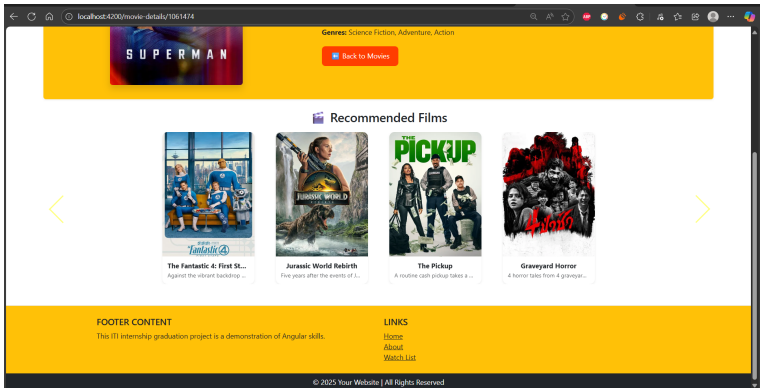
How the Movie Details Page Works:

1-Route Parameter Extraction: The component extracts the movie ID from the URL route parameters
2-API Data Fetching: Makes HTTP request to TMDB API to retrieve detailed movie information
3-Data Binding: Populates the template with movie details using Angular's signal-based reactivity
4-Image Display: Shows movie poster using TMDB's image API with responsive formatting
5-Financial Data Formatting: Uses Angular's number pipe to format budget and revenue values
6-Genre Listing: Displays up to three main genres for the movie
7-Navigation: Provides back button to return to the main movies page
8-Recommendation Integration: Includes recommended films component for related content

Key Features:

>Comprehensive Information: Displays all essential movie details in a structured format
>Responsive Design: Flexbox layout adapts to different screen sizes
>Visual Appeal: Uses Bootstrap styling with shadows and rounded corners
>Data Formatting: Properly formats numbers, dates, and ratings for readability
>Error Handling: Includes error logging for API failures
>Seamless Navigation: Easy return to main page with router link functionality
>Component Integration: Works with recommended films for enhanced user experience

4.2 more details page recommended films:



The movie details page includes a recommended films section that suggests similar movies based on the currently viewed film, enhancing content discovery and user engagement.

Implementation Details:

1. Component Integration

The recommended films component is embedded within the movie details page

```
<app-recommended-films [movieId]="movie()?.id"></app-recommended-films>
```

2. Data Fetching Logic

The component receives the current movie ID and fetches recommendations from TMDB API

```
@Input() movieId: string | null = null;

loadRecommendedMovies(id: string): void {
  this.http.get<any>(`${this.apiUrl}/movie/${id}/recommendations`, {
    params: { api_key: this.apiKey, language: 'en-US' }
  }).subscribe({
    next: (res) => {
      this.films.set(res.results);
      this.chunkedFilms.set(this.groupArray(res.results, 4));
    },
    error: (err) => console.error('Failed to load recommended movies', err)
  });
}
```

3. Data Organization

Films are grouped into chunks for carousel display

```
private groupArray(arr: any[], size: number): any[][] {
  const result = [];
  for (let i = 0; i < arr.length; i += size) {
    result.push(arr.slice(i, i + size));
  }
  return result;
}
```

4. Carousel Display

The template implements a responsive Bootstrap carousel

```
<div class="recommended-films-slider mt-4">
  <h3 class="mb-3 text-center">🎞️ Recommended Films</h3>

  <div id="recommendedCarousel" class="carousel slide" data-bs-ride="false">
    <div class="carousel-inner">
      @for (group of chunkedFilms(); track $index; let i = $index) {
        <div class="carousel-item" [class.active]="i === 0">
          <div class="d-flex justify-content-center flex-wrap gap-5">
            @for (film of group; track $index) {
              <div class="card film-card shadow-sm border-0">
                <img [src]="'https://image.tmdb.org/t/p/w500' + film.poster_path"
                     [alt]="film.title"
                     class="card-img-top rounded-top">
                <div class="card-body">
                  <h6 class="card-title text-center fw-bold text-truncate">{{ film.title }}</h6>
                  <p class="card-text small text-muted text-truncate">
                    {{ film.overview }}
                  </p>
                </div>
              </div>
            }
          </div>
        </div>
      }
    </div>

    <button class="carousel-control-prev" type="button" data-bs-target="#recommendedCarousel" data-bs-slide
      <span class="carousel-control-prev-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Previous</span>
```

How the Recommendation System Works:

1-Input Reception: The component receives the current movie's ID via the @Input() property

2-API Request: Fetches recommended movies from TMDB's recommendation endpoint using the movie ID

3-Data Processing: Groups the results into sets of 4 films for organized carousel display

4-Carousel Initialization: Sets the first group as active and prepares subsequent groups

5-Visual Presentation: Displays films in a responsive carousel with navigation controls

6-User Interaction: Allows users to browse through multiple recommendation sets using prev/next buttons

Key Features:

>Contextual Relevance: Shows movies specifically related to the currently viewed film

>Responsive Carousel: Adapts to different screen sizes with smooth navigation between film groups

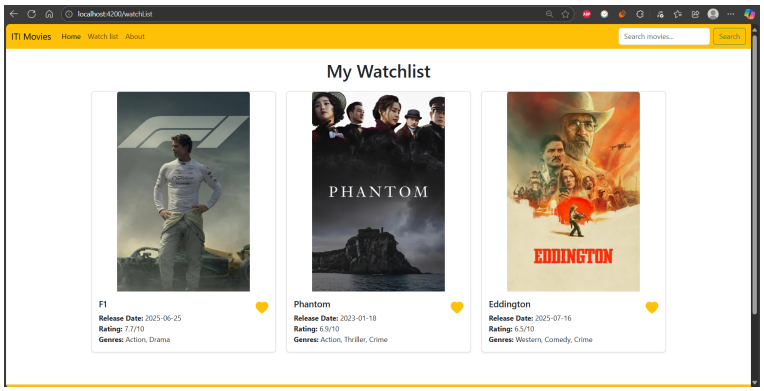
>Visual Consistency: Maintains the same card design language as the main movie grid

>Content Preview: Displays movie titles and truncated overviews for quick browsing

>Seamless Integration: Works harmoniously with the parent movie details component

>Performance Optimized: Efficient grouping and display logic ensures smooth user experience

5 watch list page:



The watch list page displays movies that users have saved to their personal collection, providing a centralized location for managing films they want to watch later.

Implementation Details:

1. Component Logic

The component structure is set up for future implementation of watchlist functionality

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

@Component({
  selector: 'app-watch-list',
  imports: [],
  templateUrl: './watch-list.html',
  styleUrl: './watch-list.css'
})
export class WatchList {
  // Component logic for managing watchlist would be implemented here
  // Future features: add/remove movies, local storage integration, API synchronization
}
```

2. Template Structure

The template displays the watchlist interface with placeholder content

```
<img src="heart.png" alt="Empty Watchlist Icon">
<h3 class="mt-4 mb-3 text-center">No movies in watch list</h3>
```

3. Expected Future Implementation

The watchlist functionality would typically include

```
export class WatchList implements OnInit {
  watchlistMovies: any[] = [];

  ngOnInit() {
    this.loadWatchlist();
  }

  loadWatchlist() {
    // Load from local storage or backend API
    const saved = localStorage.getItem('watchlist');
    this.watchlistMovies = saved ? JSON.parse(saved) : [];
  }

  removeFromWatchlist(movieId: number) {
    this.watchlistMovies = this.watchlistMovies.filter(m => m.id !== movieId);
    this.saveWatchlist();
  }

  saveWatchlist() {
    localStorage.setItem('watchlist', JSON.stringify(this.watchlistMovies));
  }
}
```

How the Watch List Page Works:

1-Component Initialization: The watch list component loads when users navigate to the watchlist page

2-Empty State Display: Shows placeholder content when no movies are saved

3-Visual Indicator: Displays a heart icon symbolizing saved/favorited movies

4-User Feedback: Clear message indicates the empty state and prompts action

5-Future Expansion: Architecture is prepared for full watchlist management functionality

Key Features:

>Movie Storage: Save and retrieve movies from local storage or database

>Add/Remove Functionality: Interface to manage watchlist items

>Movie Display Grid: Grid layout showing poster, title, and rating of saved movies

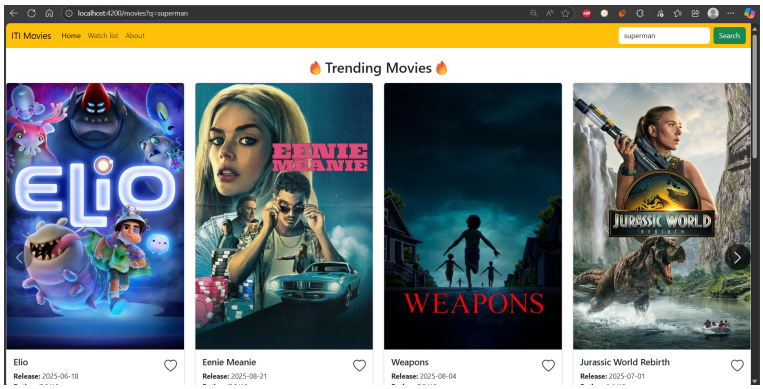
>Search & Filter: Ability to search through saved movies

>Categories/Tags: Organize movies with custom categories

>Synchronization: Sync watchlist across devices with backend integration

>Export Options: Share or export watchlist content

6.1 search bar:



The search bar provides users with the ability to search for movies across the platform, featuring real-time suggestions and seamless integration with the navigation system.

Implementation Details:

1. Search Component Logic

The search bar component manages search queries and emits events

```
import { Component, Output, EventEmitter } from '@angular/core';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-navbar',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './navbar.html',
  styleUrl: './navbar.css'
})
export class Navbar {
  searchQuery: string = '';

  @Output() search = new EventEmitter<string>();

  onSearch() {
    if (this.searchQuery.trim()) {
      this.search.emit(this.searchQuery);
    }
  }
}
```

2. Search Template

The search form is integrated into the navigation bar

```
<form class="d-flex" role="search" (ngSubmit)="onSearch()">
  <input
    class="form-control me-2"
    type="search"
    placeholder="Search movies..."
    aria-label="Search"
    [(ngModel)]="searchQuery"
    name="search"
  />
  <button class="btn btn-outline-success" type="submit">Search</button>
</form>
```

Key Features:

>Real-time Search: Quick access to movie search from any page
>Two-way Data Binding: [(ngModel)] keeps search query synchronized
>Form Validation: Prevents empty search submissions
>API Integration: Leverages TMDB's comprehensive search capabilities
>Results Filtering: Combines API results with client-side filtering for accuracy
>Route Integration: Supports deep linking with search queries in URL
>Responsive Design: Fits seamlessly into the navigation bar across devices
>User Feedback: Clear visual indicators during search process

3. Search Handling in Movies Component

The main movies component processes search queries

```
searchMovies(query: string): void {
  this.http.get<any>(`${this.apiUrl}/search/movie`, {
    params: {
      api_key: this.apiKey,
      query: query,
      language: 'en-US',
      page: '1'
    }
  }).subscribe({
    next: (res) => {
      let limitedMovies = res.results.slice(0, 6).map((m: any) => ({ ...m, liked: false }));
      limitedMovies = limitedMovies.filter((m: any) =>
        m.title.toLowerCase().includes(query.toLowerCase())
      );
      this.movies.set(limitedMovies);
      this.totalPages = res.total_pages;
      this.currentPage = 1;
      this.updateVisiblePages();
    },
    error: (err) => console.error('Error searching movies:', err)
  });
}
```

4. Route Integration for Search

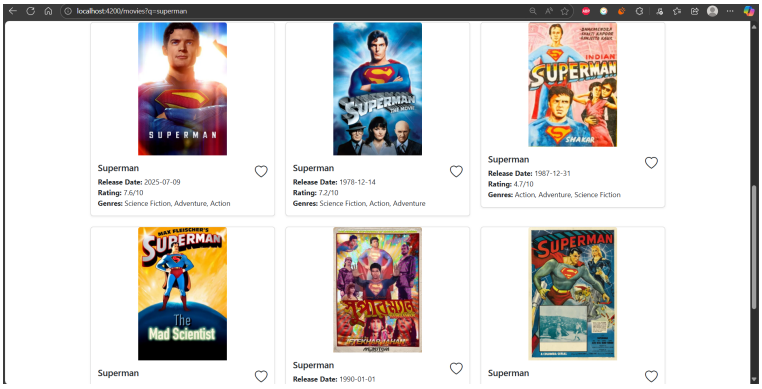
Search functionality integrated with Angular routing

```
this.route.queryParams.subscribe(params => {
  if (params['q']) {
    this.searchMovies(params['q']);
  } else {
    // Load popular movies if no search query
    this.loadGenres().subscribe({
      next: (response) => {
        response.genres.forEach((g: any) => this.genresMap.set(g.id, g.name));
        this.loadMovies(1);
      },
      error: (err) => console.error('Error loading genres:', err)
    });
  }
});
```

How the Search Functionality Works:

1-User Input: Users type search queries into the input field in the navigation bar
2-Form Submission: Pressing Enter or clicking the search button triggers the onSearch() method
3-Event Emission: The search query is emitted to parent components via the @Output() event
4-API Request: The movies component sends a request to TMDB's search endpoint
5-Results Filtering: Additional client-side filtering ensures relevant results
6-UI Update: Search results replace the popular movies display
7-Pagination Reset: Pagination is reset to page 1 for new search results
8-URL Integration: Search queries can be handled via route parameters for shareable URLs

6.2 search results:



The search results page displays movies that match user search queries, providing a filtered view of the movie catalog with relevant results from the TMDB API.

Implementation Details:

1. Search Execution Logic

The component handles search functionality through TMDB API integration

```
searchMovies(query: string): void {
  this.http.get<any>(`${this.apiUrl}/search/movie`, {
    params: {
      api_key: this.apiKey,
      query: query,
      language: 'en-US',
      page: '1'
    }
  }).subscribe({
    next: (res) => {
      // API results processing
      let limitedMovies = res.results.slice(0, 6).map((m: any) => ({ ...m, liked: false }));

      // Additional client-side filtering
      limitedMovies = limitedMovies.filter((m: any) =>
        m.title.toLowerCase().includes(query.toLowerCase())
      );

      this.movies.set(limitedMovies);
      this.totalPages = res.total_pages;
      this.currentPage = 1;
      this.updateVisiblePages();
    },
    error: (err) => console.error('Error searching movies:', err)
  });
}
```

2. Route Parameter Integration

Search functionality integrated with Angular routing for direct URL access

```
this.route.queryParams.subscribe(params => {
  if (params['q']) {
    this.searchMovies(params['q']);
  } else {
    // Load popular movies if no search query present
    this.loadGenres().subscribe({
      next: (response) => {
        response.genres.forEach((g: any) => this.genresMap.set(g.id, g.name));
        this.loadMovies(1);
      },
      error: (err) => console.error('Error loading genres:', err)
    });
  }
});
```

How the Search Results System Works:

1-Query Initiation: User enters search term in navbar and submits
2-Event Emission: Navbar emits search query to parent component
3-API Request: Movies component sends request to TMDB search endpoint
4-Dual Filtering: Combines API results with client-side title matching
5-Data Processing: Formats results with like functionality and genre mapping
6-UI Population: Displays filtered results using the same card layout as main page
7-Pagination Setup: Configures pagination for search results (if multiple pages)
8-State Management: Updates component state with search results instead of popular movies

3. Results Display Template

The same template used for popular movies adapts to display search results

```
<div class="container mt-4">
  <div class="row">
    <div class="col-md-4" *ngFor="let movie of movies()">
      <div class="card mb-4 shadow-sm">
        <img
          [src]="'https://image.tmdb.org/t/p/w500' + movie.poster_path"
          class="card-img-top"
          alt="{{ movie.title }}" />
        <div class="card-body">
          <i
            class="bi"
            [ngClass]="movie.liked ? 'bi-heart-fill text-warning fs-3' : 'bi-heart fs-3'"
            (click)="toggleLike(movie)"
            style="cursor: pointer; float: right;">
          </i>
          <h5 class="card-title">{{ movie.title }}</h5>
          <p class="card-text">
            <strong>Release Date:</strong> {{ movie.release_date }} <br>
            <strong>Rating:</strong> {{ movie.vote_average | number:'1.1-1' }}/10 <br>
            <strong>Genres:</strong> {{ getGenresForMovie(movie.genre_ids) }}
          </p>
          <button class="btn btn-primary" (click)="logMovieId(movie.id)" [routerLink]="['/movie-details', movie.id]">
            More Details
          </button>
        </div>
      </div>
    </div>
  </div>
</div>
```

4. Search Query Emission

The navbar component emits search queries to trigger results display

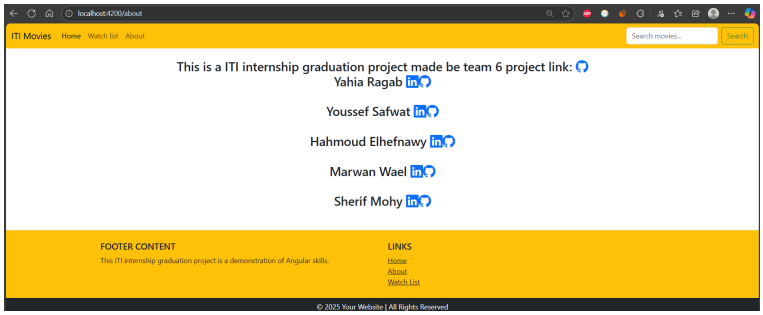
```
@Output() search = new EventEmitter<string>();

onSearch() {
  if (this.searchQuery.trim()) {
    this.search.emit(this.searchQuery);
  }
}
```

Key Features:

>Unified Display: Uses same template for both popular movies and search results
>Dual Filtering: Combines server-side API search with client-side filtering for accuracy
>Genre Integration: Maintains genre display functionality in search results
>Pagination Ready: Supports pagination for extensive search results
>Route Synchronization: Search queries reflected in URL for sharing and bookmarking
>Seamless Transition: Smooth switch between popular movies and search results
>Full Functionality: Search results maintain all features (liking, details navigation, etc.)
>Error Handling: Robust error management for API failures

7 about page:



The About page provides information about the development team, project purpose, and social media links, serving as a credit page and project overview.

Implementation Details:

1. Component Logic

The component provides the basic structure for the About page

```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-about',
  imports: [],
  templateUrl: './about.html',
  styleUrl: './about.css'
})
export class About {
  // Component logic for about page information
  // Can be expanded to include dynamic data or team member profiles
}
```

2. Template Structure

The template displays team information and project details with social media links

```html
<h3 class="mt-4 mb-3 text-center">This is a ITI internship graduation project made be team 6 pro
ject link: <a href="https://github.com/Yahia-Ragab/Movies-Website"><i class="bi bi-github"></i>
</a><br>
Yahia Ragab <a href="https://www.linkedin.com/in/yahia-ragab77/"><i class="bi bi-linkedin"></i>
</a> <a href="https://github.com/Yahia-Ragab/"><i class="bi bi-github"></i></a> <br><br>
Youssef Safwat <a href=""><i class="bi bi-linkedin"></i></a> <a href=""><i class="bi bi-github">
</i></a><br><br>
Hahmoud Elhefnawy <a href=""><i class="bi bi-linkedin"></i></a> <a href=""><i class="bi bi-githu
b"></i></a><br><br>
Marwan Wael <a href=""><i class="bi bi-linkedin"></i></a> <a href="https://github.com/Yahia-Raga
b/"><i class="bi bi-github"></i></a><br><br>
Sherif Mohy <a href=""><i class="bi bi-linkedin"></i></a> <a href=""><i class="bi bi-github"></i
></a>
</h3>
```

3.Styling

The component include styling for proper presentation

```css
.about-container {
  padding: 2rem;
  text-align: center;
}
.team-member {
  margin: 1.5rem 0;
  padding: 1rem;
}
.social-links {
  margin-top: 0.5rem;
}
.social-links a {
  margin: 0 0.5rem;
  color: #0077b5; /* LinkedIn blue */
  font-size: 1.2rem;
}
.social-links a:hover {
  color: #005582;
}
.bi-github {
  color: #333; /* GitHub black */
}
.bi-linkedin {
  color: #0077b5; /* LinkedIn blue */
}
```

How the About Page Works:

1-Page Navigation: Users access the About page through the navigation menu

2-Component Loading: Angular loads the About component when route is activated

3-Static Content Display: Shows pre-defined team information and project details

4-Social Media Integration: Provides clickable links to team members' profiles

5-Project Reference: Includes link to GitHub repository for source code access

6-Responsive Design: Adapts layout for different screen sizes

Key Features:

>Team Introduction: Displays all team members with their names

>Social Media Links: Provides LinkedIn and GitHub links for professional networking

>Project Attribution: Clearly identifies the project as an ITI internship graduation work

>GitHub Integration: Direct link to project repository for transparency

>Clean Layout: Centered, organized presentation of information

>Icon Integration: Uses Bootstrap Icons for visual representation of social platforms

>Professional Design: Maintains consistent styling with the rest of the application

Conclusion & Future Work:

Conclusion:

This project successfully delivers a full-featured online movie platform built with the MEAN stack (MongoDB, Express.js, Angular, and Node.js). The application provides users with a comprehensive movie browsing experience, including popular and trending movie displays, detailed film information, personalized watchlists, and an efficient search functionality. By leveraging The Movie Database (TMDB) API, the platform offers real-time, up-to-date movie information while maintaining a responsive and user-friendly interface.

The system demonstrates effective integration of modern web technologies, showcasing Angular's component-based architecture for the frontend, Node.js with Express for backend services, and seamless API communication. The implementation of features such as hover effects on movie cards, paginated navigation, and recommended film suggestions enhances user engagement and provides a polished, professional user experience comparable to commercial streaming platforms.

This project serves as a practical demonstration of full-stack development capabilities, addressing real-world needs in digital entertainment while providing a scalable foundation for future enhancements.

Future Work:

Several enhancements could further improve the platform's functionality and user experience:

1-User Authentication System
- >Implement user registration and login functionality
- >Develop personalized user profiles and preferences
- >Enable secure saving of watchlists across devices

2-Advanced Recommendation Engine
- >Create machine learning-based recommendations using viewing history
- >Implement collaborative filtering for personalized suggestions
- >Add "Because you watched..." and similar movie features

3-Performance Optimization
- >Implement caching mechanisms for faster loading times
- >Add lazy loading for images and components
- >Optimize API calls with request debouncing and caching

4-Enhanced Social Features
- >Add user reviews and rating system
- >Implement social sharing capabilities
- >Create user forums or discussion boards for movies

5-Additional Functionality
- >Integrate video trailers using YouTube API
- >Add advanced filtering and sorting options
- >Implement dark mode and accessibility features
- >Create mobile application version using Ionic or React Native

6-Content Expansion
- >Include TV series and documentaries in addition to movies
- >Add celebrity profiles and filmography information
- >Integrate showtimes and theater information for new releases

7-Administration Features
- >Develop admin dashboard for content management
- >Implement user analytics and usage statistics
- >Add content moderation tools for user-generated content

These future enhancements would transform the platform from a demonstration project into a fully-featured commercial streaming service, providing greater value to users and expanding the application's capabilities in the competitive digital entertainment landscape.