# Створення проекту "Що мені подивитись?" з використанням Angular 19

Давайте поетапно створимо проект з двомовними коментарями в коді та англійським інтерфейсом.

## Етап 1: Налаштування проекту Angular 19

### Крок 1: Встановлення Angular CLI та створення проекту

bash

npm install -g @angular/cli@19

ng new frontend --standalone

При створенні проекту виберіть:

* "Yes" для Angular routing
* "CSS" для стилів

### Крок 2: Перейдіть у директорію проекту

bash

cd frontend

### Крок 3: Встановлення Bootstrap

bash

npm install bootstrap@5.3.2 @popperjs/core

### Крок 4: Налаштування Bootstrap в проекті

Відкрийте файл angular.json і додайте Bootstrap у секцію "architect > build > options":

json

"styles": [

"node\_modules/bootstrap/dist/css/bootstrap.min.css",

"src/styles.css"

],

"scripts": [

"node\_modules/@popperjs/core/dist/umd/popper.min.js",

"node\_modules/bootstrap/dist/js/bootstrap.min.js"

]

### Крок 5: Створення базової структури проекту

bash

mkdir -p src/app/components/navbar

mkdir -p src/app/components/film-list

mkdir -p src/app/components/film-detail

mkdir -p src/app/components/film-form

mkdir -p src/app/components/login

mkdir -p src/app/components/register

mkdir -p src/app/models

mkdir -p src/app/services

### Створення компонентів-заглушок

Виконайте такі команди в терміналі для створення компонентів-заглушок:

bash

ng generate component components/navbar --standalone

ng generate component components/film-list --standalone

ng generate component components/film-detail --standalone

ng generate component components/film-form --standalone

ng generate component components/login --standalone

ng generate component components/register --standalone

## Етап 2: Створення моделей даних

### Крок 1: Створення моделі для фільму

Файл: src/app/models/film.model.ts

typescript

/\*\*

\* Film interface defines the structure of film objects in the application

\*

\* Інтерфейс Film визначає структуру об'єктів фільмів у додатку

\*/

export interface Film {

id: number; // Unique film identifier / Унікальний ідентифікатор фільму

title: string; // Film title / Назва фільму

release\_year: number; // Year when film was released / Рік випуску фільму

genre: string; // Film genre / Жанр фільму

description: string; // Film description / Опис фільму

poster\_url: string; // URL to the film poster image / URL-адреса постера фільму

added\_by: number; // ID of the user who added this film / ID користувача, який додав фільм

created\_at: string; // Date and time when film was added / Дата і час додавання фільму

}

### Крок 2: Створення моделі для користувача

Файл: src/app/models/user.model.ts

typescript

/\*\*

\* User interface defines the structure of user objects

\*

\* Інтерфейс User визначає структуру об'єктів користувачів

\*/

export interface User {

id: number; // User ID / ID користувача

username: string; // Username / Ім'я користувача

email: string; // Email address / Електронна адреса

}

/\*\*

\* AuthResponse interface defines the structure of authentication response from the server

\*

\* Інтерфейс AuthResponse визначає структуру відповіді автентифікації від сервера

\*/

export interface AuthResponse {

id: number; // User ID / ID користувача

username: string; // Username / Ім'я користувача

email: string; // Email address / Електронна адреса

token: string; // Authentication token / Токен автентифікації

}

### Крок 3: Створення моделі для коментаря

Файл: src/app/models/comment.model.ts

typescript

/\*\*

\* Comment interface defines the structure of comments for films

\*

\* Інтерфейс Comment визначає структуру коментарів до фільмів

\*/

export interface Comment {

id: number; // Comment ID / ID коментаря

film\_id: number; // ID of the film this comment belongs to / ID фільму, до якого відноситься коментар

user\_id: number; // ID of the user who wrote this comment / ID користувача, який написав коментар

content: string; // Comment content / Зміст коментаря

created\_at: string; // Date and time when comment was created / Дата і час створення коментаря

username?: string; // Username (optional, may be fetched separately) / Ім'я користувача (необов'язкове)

}

## Етап 3: Налаштування маршрутизації

Файл: src/app/app.routes.ts

typescript

/\*\*

\* Application routes configuration

\*

\* Конфігурація маршрутів додатку

\*/

import { Routes } from '@angular/router';

export const routes: Routes = [

{

path: '',

loadComponent: () => import('./components/film-list/film-list.component').then(c => c.FilmListComponent)

},

{

path: 'film/:id',

loadComponent: () => import('./components/film-detail/film-detail.component').then(c => c.FilmDetailComponent)

},

{

path: 'add-film',

loadComponent: () => import('./components/film-form/film-form.component').then(c => c.FilmFormComponent)

},

{

path: 'edit-film/:id',

loadComponent: () => import('./components/film-form/film-form.component').then(c => c.FilmFormComponent)

},

{

path: 'login',

loadComponent: () => import('./components/login/login.component').then(c => c.LoginComponent)

},

{

path: 'register',

loadComponent: () => import('./components/register/register.component').then(c => c.RegisterComponent)

},

{ path: '\*\*', redirectTo: '' }

];

## Етап 4: Створення сервісів

### Крок 1: Створення сервісу для авторизації

Спочатку створіть сервіс за допомогою Angular CLI:

bash

ng generate service services/auth

Потім відредагуйте файл src/app/services/auth.service.ts:

typescript

/\*\*

\* Authentication service for user management

\*

\* Сервіс автентифікації для управління користувачами

\*/

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

import { HttpClient } from '@angular/common/http';

import { BehaviorSubject, Observable, tap } from 'rxjs';

import { User, AuthResponse } from '../models/user.model';

@Injectable({

providedIn: 'root'

})

export class AuthService {

// API URL for authentication endpoints

// URL API для ендпоінтів автентифікації

private apiUrl = 'http://localhost:3001/api';

// BehaviorSubject to track the current user state

// BehaviorSubject для відстеження стану поточного користувача

private currentUserSubject = new BehaviorSubject<User | null>(null);

// Observable to subscribe to user changes

// Observable для підписки на зміни користувача

public currentUser$ = this.currentUserSubject.asObservable();

constructor(private http: HttpClient) { }

/\*\*

\* Log in a user with email and password

\*

\* Вхід користувача за допомогою електронної пошти та пароля

\*/

login(email: string, password: string): Observable<AuthResponse> {

return this.http.post<AuthResponse>(`${this.apiUrl}/auth/login`, { email, password })

.pipe(tap(response => {

const user: User = {

id: response.id,

username: response.username,

email: response.email

};

// Update the current user state

// Оновлення стану поточного користувача

this.currentUserSubject.next(user);

}));

}

/\*\*

\* Register a new user

\*

\* Реєстрація нового користувача

\*/

register(username: string, email: string, password: string): Observable<any> {

return this.http.post(`${this.apiUrl}/auth/register`, { username, email, password });

}

/\*\*

\* Log out the current user

\*

\* Вихід поточного користувача

\*/

logout(): void {

this.currentUserSubject.next(null);

}

/\*\*

\* Check if a user is currently authenticated

\*

\* Перевірка, чи користувач автентифікований

\*/

isAuthenticated(): boolean {

return this.currentUserSubject.value !== null;

}

/\*\*

\* Get the current user information

\*

\* Отримання інформації про поточного користувача

\*/

getCurrentUser(): User | null {

return this.currentUserSubject.value;

}

}

### Крок 2: Створення сервісу для фільмів

Спочатку створіть сервіс за допомогою Angular CLI:

bash

ng generate service services/film

Потім відредагуйте файл src/app/services/film.service.ts:

typescript

/\*\*

\* Film service for managing films and comments

\*

\* Сервіс фільмів для управління фільмами та коментарями

\*/

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

import { HttpClient, HttpParams } from '@angular/common/http';

import { Observable } from 'rxjs';

import { Film } from '../models/film.model';

import { Comment } from '../models/comment.model';

@Injectable({

providedIn: 'root'

})

export class FilmService {

// Base API URL

// Базовий URL API

private apiUrl = 'http://localhost:3001/api';

constructor(private http: HttpClient) { }

/\*\*

\* Get a list of films with optional filtering by genre and year

\*

\* Отримання списку фільмів з необов'язковою фільтрацією за жанром та роком

\*/

getFilms(genre?: string, year?: number): Observable<Film[]> {

let params = new HttpParams();

if (genre) params = params.set('genre', genre);

if (year) params = params.set('year', year.toString());

return this.http.get<Film[]>(`${this.apiUrl}/films`, { params });

}

/\*\*

\* Get a single film by its ID

\*

\* Отримання одного фільму за ID

\*/

getFilmById(id: number): Observable<Film> {

return this.http.get<Film>(`${this.apiUrl}/films/${id}`);

}

/\*\*

\* Get comments for a specific film

\*

\* Отримання коментарів для конкретного фільму

\*/

getComments(filmId: number): Observable<Comment[]> {

return this.http.get<Comment[]>(`${this.apiUrl}/films/${filmId}/comments`);

}

/\*\*

\* Add a new comment to a film

\*

\* Додавання нового коментаря до фільму

\*/

addComment(filmId: number, content: string): Observable<Comment> {

return this.http.post<Comment>(`${this.apiUrl}/films/${filmId}/comments`, {

content

});

}

/\*\*

\* Create a new film

\*

\* Створення нового фільму

\*/

createFilm(filmData: Partial<Film>): Observable<Film> {

return this.http.post<Film>(`${this.apiUrl}/films`, filmData);

}

/\*\*

\* Update an existing film

\*

\* Оновлення існуючого фільму

\*/

updateFilm(id: number, filmData: Partial<Film>): Observable<Film> {

return this.http.put<Film>(`${this.apiUrl}/films/${id}`, filmData);

}

}

## Етап 5: Оновлення головного компонента та налаштування HTTP

### Крок 1: Оновлення файлу src/app/app.config.ts

typescript

/\*\*

\* Application configuration

\*

\* Конфігурація додатку

\*/

import { ApplicationConfig } from '@angular/core';

import { provideRouter } from '@angular/router';

import { provideHttpClient, withFetch } from '@angular/common/http';

import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {

providers: [

provideRouter(routes),

provideHttpClient(withFetch())

]

};

### Крок 2: Оновлення головного компоненту src/app/app.component.ts

typescript

/\*\*

\* Main application component

\*

\* Головний компонент додатку

\*/

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

import { RouterOutlet } from '@angular/router';

import { NavbarComponent } from './components/navbar/navbar.component';

@Component({

selector: 'app-root',

standalone: true,

imports: [RouterOutlet, NavbarComponent],

template: `

<app-navbar></app-navbar>

<router-outlet></router-outlet>

`,

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'film-forum';

}

## Перевірка структури проекту

Після виконання цих кроків у вас повинна бути така структура проекту:

arduino

frontend/

├── src/

│ ├── app/

│ │ ├── components/

│ │ │ ├── navbar/

│ │ │ ├── film-list/

│ │ │ ├── film-detail/

│ │ │ ├── film-form/

│ │ │ ├── login/

│ │ │ └── register/

│ │ ├── models/

│ │ │ ├── film.model.ts

│ │ │ ├── user.model.ts

│ │ │ └── comment.model.ts

│ │ ├── services/

│ │ │ ├── auth.service.ts

│ │ │ └── film.service.ts

│ │ ├── app.component.ts

│ │ ├── app.config.ts

│ │ └── app.routes.ts

│ ├── assets/

│ └── index.html

├── angular.json

└── package.json

На цьому етапі ви успішно налаштували базову структуру проекту з моделями даних та сервісами. Для перевірки, чи все працює правильно, запустіть команду:

bash

ng serve

Після перевірки структури проекту ми можемо перейти до наступного етапу - створення компонентів.

# Створення компонентів для проекту "Що мені подивитись?"

Для нашого проекту нам потрібно створити наступні компоненти:

1. **Navbar Component** (navbar)
   * Навігаційна панель для всього додатку
   * Відображення стану авторизації
   * Меню навігації
2. **Film List Component** (film-list)
   * Головна сторінка додатку
   * Список фільмів з карточками
   * Фільтрація за жанром та роком
3. **Film Detail Component** (film-detail)
   * Детальна інформація про фільм
   * Коментарі до фільму
   * Можливість додавання нових коментарів
4. **Film Form Component** (film-form)
   * Форма для додавання нового фільму
   * Форма для редагування існуючого фільму
   * Валідація форми
5. **Login Component** (login)
   * Форма для входу користувача
   * Валідація полів вводу
6. **Register Component** (register)
   * Форма для реєстрації нового користувача
   * Валідація полів вводу

Давайте почнемо зі створення цих компонентів по одному. Ви можете надати мені готовий варіант компонента, який ви хочете реалізувати першим, і ми його адаптуємо з двомовними коментарями та англійським інтерфейсом.

З якого компонента ви хотіли б почати?

# Створення Navbar Component з англійським інтерфейсом

Давайте адаптуємо ваш код з двомовними коментарями та перекладемо інтерфейс на англійську мову.

## Крок 1: Оновлення navbar.component.ts

typescript

/\*\*

\* Navbar component for application navigation

\*

\* Компонент навігаційної панелі для переміщення додатком

\*/

import { Component, OnInit } from '@angular/core';

import { Router, RouterLink } from '@angular/router';

import { CommonModule } from '@angular/common';

import { AuthService } from '../../services/auth.service';

import { User } from '../../models/user.model';

@Component({

selector: 'app-navbar',

standalone: true,

imports: [CommonModule, RouterLink],

templateUrl: './navbar.component.html',

styleUrls: ['./navbar.component.css']

})

export class NavbarComponent implements OnInit {

// Current authenticated user (null if not logged in)

// Поточний автентифікований користувач (null якщо не увійшов)

currentUser: User | null = null;

constructor(private authService: AuthService, private router: Router) { }

/\*\*

\* Initialize component and subscribe to user authentication changes

\*

\* Ініціалізація компонента та підписка на зміни автентифікації користувача

\*/

ngOnInit(): void {

this.authService.currentUser$.subscribe(user => {

this.currentUser = user;

});

}

/\*\*

\* Handle user logout and redirect to home page

\*

\* Обробка виходу користувача та перенаправлення на головну сторінку

\*/

logout(): void {

this.authService.logout();

this.router.navigate(['/']);

}

}

## Крок 2: Оновлення navbar.component.html з англійським інтерфейсом

html

<nav class="navbar navbar-expand-lg navbar-dark bg-dark">

<div class="container">

<!-- App logo/name with link to home page -->

<!-- Логотип/назва додатку з посиланням на головну сторінку -->

<a class="navbar-brand" routerLink="/">🎬 What to Watch?</a>

<!-- Main navigation links -->

<!-- Основні навігаційні посилання -->

<div class="navbar-nav me-auto">

<a class="nav-link" routerLink="/">Home</a>

<!-- Only show "Add Film" link if user is logged in -->

<!-- Показувати посилання "Додати фільм" тільки якщо користувач увійшов -->

<a class="nav-link" routerLink="/add-film" \*ngIf="currentUser">Add Film</a>

</div>

<!-- Authentication related links -->

<!-- Посилання, пов'язані з автентифікацією -->

<div class="navbar-nav">

<!-- Show login/register links when user is not logged in -->

<!-- Показати посилання для входу/реєстрації, коли користувач не увійшов -->

<ng-container \*ngIf="!currentUser">

<a class="nav-link" routerLink="/login">Login</a>

<a class="nav-link" routerLink="/register">Register</a>

</ng-container>

<!-- Show username and logout button when user is logged in -->

<!-- Показати ім'я користувача та кнопку виходу, коли користувач увійшов -->

<ng-container \*ngIf="currentUser">

<span class="navbar-text me-3">Hello, {{ currentUser.username }}</span>

<button class="btn btn-outline-light btn-sm" (click)="logout()">Logout</button>

</ng-container>

</div>

</div>

</nav>

## Крок 3: Тестування компонента

Щоб перевірити, чи правильно працює навігаційна панель, запустіть додаток:

bash

ng serve

Відкрийте браузер за адресою http://localhost:4200/ і перевірте, що:

1. Навігаційна панель відображається коректно
2. Посилання "Home" працює і перенаправляє на головну сторінку
3. Посилання "Login" і "Register" відображаються (оскільки користувач ще не ввійшов)
4. Посилання "Add Film" не повинно відображатися

На цьому етапі ви побачите помилки, пов'язані з відсутністю інших компонентів, але навігаційна панель повинна відображатися правильно.

# Створення Film List Component з англійським інтерфейсом

Адаптуємо Film List Component з двомовними коментарями та англійським інтерфейсом.

## Крок 1: Оновлення film-list.component.ts

typescript

/\*\*

\* Film List Component - Displays the list of films with filtering options

\*

\* Компонент списку фільмів - Відображає список фільмів з можливостями фільтрації

\*/

import { Component, OnInit } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { RouterLink } from '@angular/router';

import { HttpClient } from '@angular/common/http';

import { FilmService } from '../../services/film.service';

import { Film } from '../../models/film.model';

@Component({

selector: 'app-film-list',

standalone: true,

imports: [CommonModule, FormsModule, RouterLink],

templateUrl: './film-list.component.html',

styleUrls: ['./film-list.component.css']

})

export class FilmListComponent implements OnInit {

// List of films to display

// Список фільмів для відображення

films: Film[] = [];

// Available genres for filtering

// Доступні жанри для фільтрації

genres: string[] = [];

// Available years for filtering

// Доступні роки для фільтрації

years: number[] = [];

// Selected filter values

// Вибрані значення фільтрів

selectedGenre = '';

selectedYear = '';

// State flags for UI

// Прапори стану для інтерфейсу

loading = true;

error = '';

constructor(

private filmService: FilmService,

private http: HttpClient

) { }

/\*\*

\* Initialize component and load necessary data

\*

\* Ініціалізує компонент і завантажує необхідні дані

\*/

ngOnInit(): void {

this.loadGenres();

this.loadYears();

this.loadFilms();

}

/\*\*

\* Load available genres from the API

\*

\* Завантажує доступні жанри з API

\*/

loadGenres(): void {

this.http.get<string[]>('http://localhost:3001/api/genres').subscribe({

next: (genres) => this.genres = genres,

error: (err) => console.error('Error loading genres:', err)

});

}

/\*\*

\* Generate list of years for the filter (from current year back to 1950)

\*

\* Генерує список років для фільтра (від поточного року до 1950)

\*/

loadYears(): void {

const currentYear = new Date().getFullYear();

for (let year = currentYear; year >= 1950; year--) {

this.years.push(year);

}

}

/\*\*

\* Load films based on selected filters

\*

\* Завантажує фільми на основі вибраних фільтрів

\*/

loadFilms(): void {

this.loading = true;

const genre = this.selectedGenre || undefined;

const year = this.selectedYear ? parseInt(this.selectedYear) : undefined;

this.filmService.getFilms(genre, year).subscribe({

next: (films) => {

this.films = films;

this.loading = false;

},

error: (err) => {

this.error = 'Error loading films';

this.loading = false;

console.error(err);

}

});

}

/\*\*

\* Apply selected filters and reload films

\*

\* Застосовує вибрані фільтри і перезавантажує фільми

\*/

applyFilters(): void {

this.loadFilms();

}

/\*\*

\* Reset all filters and reload films

\*

\* Скидає всі фільтри і перезавантажує фільми

\*/

resetFilters(): void {

this.selectedGenre = '';

this.selectedYear = '';

this.loadFilms();

}

}

## Крок 2: Оновлення film-list.component.html

html

<div class="container mt-4">

<!-- Filters section -->

<!-- Секція фільтрів -->

<div class="filters-container mb-4">

<h3>Filters</h3>

<div class="row">

<div class="col-md-4">

<label class="form-label">Genre</label>

<select class="form-select" [(ngModel)]="selectedGenre" name="genre">

<option value="">All genres</option>

<option \*ngFor="let genre of genres" [value]="genre">{{ genre }}</option>

</select>

</div>

<div class="col-md-4">

<label class="form-label">Year</label>

<select class="form-select" [(ngModel)]="selectedYear" name="year">

<option value="">All years</option>

<option \*ngFor="let year of years" [value]="year">{{ year }}</option>

</select>

</div>

<div class="col-md-4 d-flex align-items-end gap-2">

<button class="btn btn-primary" (click)="applyFilters()">Apply Filters</button>

<button class="btn btn-secondary" (click)="resetFilters()">Reset</button>

</div>

</div>

</div>

<!-- Loading indicator -->

<!-- Індикатор завантаження -->

<div class="text-center my-5" \*ngIf="loading">

<div class="spinner-border text-primary"></div>

<p>Loading films...</p>

</div>

<!-- Error message display -->

<!-- Відображення повідомлення про помилку -->

<div class="alert alert-danger" \*ngIf="error">{{ error }}</div>

<!-- Films Grid - displays all films in card format -->

<!-- Сітка фільмів - відображає всі фільми у форматі карток -->

<div class="row" \*ngIf="!loading">

<div class="col-md-4 mb-4" \*ngFor="let film of films">

<div class="card film-card h-100" [routerLink]="['/film', film.id]">

<!-- Film poster -->

<!-- Постер фільму -->

<img [src]="film.poster\_url" class="card-img-top film-poster-small" [alt]="film.title">

<div class="card-body">

<!-- Film title -->

<!-- Назва фільму -->

<h5 class="card-title">{{ film.title }}</h5>

<p class="card-text">

<!-- Genre and year badges -->

<!-- Бейджі жанру та року -->

<span class="badge bg-primary me-2">{{ film.genre }}</span>

<span class="badge bg-secondary">{{ film.release\_year }}</span>

</p>

<!-- Truncated description -->

<!-- Скорочений опис -->

<p class="card-text">{{ film.description | slice:0:100 }}...</p>

</div>

</div>

</div>

<!-- Message when no films are found -->

<!-- Повідомлення, коли фільми не знайдено -->

<div class="col-12" \*ngIf="films.length === 0">

<div class="text-center">

<h4>No films found</h4>

<p>Try changing your filters.</p>

</div>

</div>

</div>

</div>

## Крок 3: Створення CSS файлу для стилізації списку фільмів

Створіть файл film-list.component.css:

css

/\*\*

\* Styles for the film list component

\*

\* Стилі для компонента списку фільмів

\*/

/\* Make film cards clickable with hover effect \*/

/\* Робить картки фільмів клікабельними з ефектом при наведенні \*/

.film-card {

cursor: pointer;

transition: transform 0.2s, box-shadow 0.2s;

}

.film-card:hover {

transform: translateY(-5px);

box-shadow: 0 10px 20px rgba(0, 0, 0, 0.2);

}

/\* Style for film posters to maintain consistent size \*/

/\* Стиль для постерів фільмів, щоб підтримувати однаковий розмір \*/

.film-poster-small {

height: 300px;

object-fit: cover;

}

/\* Styling for the filters container \*/

/\* Стилізація для контейнера фільтрів \*/

.filters-container {

background-color: #f8f9fa;

padding: 20px;

border-radius: 8px;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);

}

## Крок 4: Тестування Film List компонента

Для тестування Film List компонента запустіть додаток:

bash

ng serve

Відкрийте браузер за адресою http://localhost:4200/ і перевірте, що:

1. Сторінка з фільмами завантажується
2. Фільтри відображаються коректно
3. Картки фільмів відображаються у сітці
4. При наведенні на картку фільму є ефект підняття
5. При відсутності фільмів відображається відповідне повідомлення

Якщо API бекенду ще не налаштоване, ви побачите помилку завантаження або порожній список.

# Створення Film Detail Component з англійським інтерфейсом

Адаптуємо Film Detail Component з двомовними коментарями та англійським інтерфейсом.

## Крок 1: Оновлення film-detail.component.ts

typescript

/\*\*

\* Film Detail Component - Displays detailed information about a specific film and its comments

\*

\* Компонент деталей фільму - Відображає детальну інформацію про конкретний фільм та його коментарі

\*/

import { Component, OnInit } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { ActivatedRoute, RouterLink } from '@angular/router';

import { FilmService } from '../../services/film.service';

import { AuthService } from '../../services/auth.service';

import { Film } from '../../models/film.model';

import { Comment } from '../../models/comment.model';

import { User } from '../../models/user.model';

@Component({

selector: 'app-film-detail',

standalone: true,

imports: [CommonModule, FormsModule, RouterLink],

templateUrl: './film-detail.component.html',

styleUrls: ['./film-detail.component.css']

})

export class FilmDetailComponent implements OnInit {

// Current film data

// Дані поточного фільму

film: Film | null = null;

// Comments for the current film

// Коментарі до поточного фільму

comments: Comment[] = [];

// Currently authenticated user

// Поточний автентифікований користувач

currentUser: User | null = null;

// New comment being written

// Новий коментар, який пишеться

newComment = '';

// State flags for UI

// Прапори стану для інтерфейсу

loading = true;

error = '';

commentLoading = false;

commentError = '';

constructor(

private route: ActivatedRoute,

private filmService: FilmService,

private authService: AuthService

) { }

/\*\*

\* Initialize component and load necessary data

\*

\* Ініціалізує компонент і завантажує необхідні дані

\*/

ngOnInit(): void {

// Subscribe to user authentication changes

// Підписка на зміни автентифікації користувача

this.authService.currentUser$.subscribe(user => {

this.currentUser = user;

});

// Get film ID from URL and load data

// Отримання ID фільму з URL і завантаження даних

this.route.params.subscribe(params => {

const filmId = parseInt(params['id']);

this.loadFilm(filmId);

this.loadComments(filmId);

});

}

/\*\*

\* Load film details by ID

\*

\* Завантажити деталі фільму за ID

\*/

loadFilm(id: number): void {

this.filmService.getFilmById(id).subscribe({

next: (film) => {

this.film = film;

this.loading = false;

},

error: (err) => {

this.error = 'Film not found';

this.loading = false;

}

});

}

/\*\*

\* Load comments for the current film

\*

\* Завантажити коментарі для поточного фільму

\*/

loadComments(filmId: number): void {

this.filmService.getComments(filmId).subscribe({

next: (comments) => this.comments = comments,

error: (err) => console.error('Error loading comments:', err)

});

}

/\*\*

\* Add a new comment to the current film

\*

\* Додати новий коментар до поточного фільму

\*/

addComment(): void {

if (!this.newComment.trim() || !this.currentUser || !this.film) return;

this.commentLoading = true;

this.commentError = '';

this.filmService.addComment(this.film.id, this.newComment).subscribe({

next: (comment) => {

// Add comment to the list

// Додавання коментаря до списку

this.comments = [comment, ...this.comments];

this.newComment = '';

this.commentLoading = false;

},

error: (err) => {

console.error('Error adding comment:', err);

this.commentError = 'Error adding comment';

this.commentLoading = false;

}

});

}

}

## Крок 2: Оновлення film-detail.component.html

html

<div class="container mt-4">

<!-- Loading indicator -->

<!-- Індикатор завантаження -->

<div class="text-center my-5" \*ngIf="loading">

<div class="spinner-border text-primary"></div>

<p>Loading film...</p>

</div>

<!-- Error message -->

<!-- Повідомлення про помилку -->

<div class="alert alert-danger" \*ngIf="error">{{ error }}</div>

<!-- Film Details Section -->

<!-- Секція деталей фільму -->

<div \*ngIf="film && !loading">

<div class="row">

<!-- Film Poster -->

<!-- Постер фільму -->

<div class="col-md-4">

<img [src]="film.poster\_url" class="img-fluid film-poster" [alt]="film.title">

<!-- Edit button (only for logged-in users) -->

<!-- Кнопка редагування (лише для авторизованих користувачів) -->

<div class="mt-3" \*ngIf="currentUser">

<a [routerLink]="['/edit-film', film.id]" class="btn btn-warning w-100">

Edit this film

</a>

</div>

</div>

<!-- Film Details -->

<!-- Деталі фільму -->

<div class="col-md-8">

<h1>{{ film.title }}</h1>

<p class="lead">

<span class="badge bg-primary me-2">{{ film.genre }}</span>

<span class="badge bg-secondary">{{ film.release\_year }}</span>

</p>

<h4>Description</h4>

<p>{{ film.description }}</p>

<!-- Back to list button -->

<!-- Кнопка повернення до списку -->

<a routerLink="/" class="btn btn-secondary">← Back to list</a>

</div>

</div>

<!-- Comments Section -->

<!-- Секція коментарів -->

<div class="mt-5">

<h3>Comments</h3>

<!-- Add Comment Form (only for authenticated users) -->

<!-- Форма додавання коментаря (лише для автентифікованих користувачів) -->

<div class="card mb-4" \*ngIf="currentUser">

<div class="card-body">

<h5>Add a comment</h5>

<!-- Error message for comments -->

<!-- Повідомлення про помилку для коментарів -->

<div class="alert alert-danger" \*ngIf="commentError">

{{ commentError }}

</div>

<div class="mb-3">

<textarea class="form-control" rows="3" [(ngModel)]="newComment" placeholder="Your comment..."

[disabled]="commentLoading"></textarea>

</div>

<button class="btn btn-primary" (click)="addComment()" [disabled]="!newComment.trim() || commentLoading">

<span \*ngIf="commentLoading" class="spinner-border spinner-border-sm me-1"></span>

Submit

</button>

</div>

</div>

<!-- Login message for non-authenticated users -->

<!-- Повідомлення про вхід для неавтентифікованих користувачів -->

<div class="alert alert-info" \*ngIf="!currentUser">

<a routerLink="/login">Log in</a> to add a comment.

</div>

<!-- Comments List -->

<!-- Список коментарів -->

<div \*ngIf="comments.length === 0" class="text-muted">

No comments for this film.

</div>

<div \*ngFor="let comment of comments" class="card mb-3">

<div class="card-body">

<h6 class="card-title">{{ comment.username || 'User' }}</h6>

<p class="card-text">{{ comment.content }}</p>

<small class="text-muted">{{ comment.created\_at | date:'short' }}</small>

</div>

</div>

</div>

</div>

</div>

## Крок 3: Створення CSS файлу для стилізації деталей фільму

Створіть файл film-detail.component.css:

css

/\*\*

\* Styles for the film detail component

\*

\* Стилі для компонента деталей фільму

\*/

/\* Style for the film poster \*/

/\* Стиль для постера фільму \*/

.film-poster {

width: 100%;

border-radius: 8px;

box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);

}

/\* Style for film title \*/

/\* Стиль для назви фільму \*/

h1 {

margin-bottom: 10px;

}

/\* Style for comment cards \*/

/\* Стиль для карток коментарів \*/

.card {

transition: transform 0.2s;

}

.card:hover {

transform: translateY(-2px);

}

/\* Style for comments section \*/

/\* Стиль для секції коментарів \*/

.comment-section {

margin-top: 40px;

padding-top: 20px;

border-top: 1px solid #dee2e6;

}

## Крок 4: Тестування Film Detail компонента

Для тестування Film Detail компонента:

bash

ng serve

Відкрийте браузер за адресою http://localhost:4200/ і перейдіть до деталей фільму (для цього спочатку потрібно натиснути на карточку фільму у списку фільмів).

Перевірте, що:

1. Детальна інформація про фільм відображається коректно
2. Бейджі для жанру та року випуску відображаються
3. Коментарі відображаються (якщо вони є)
4. Форма додавання коментаря відображається лише для авторизованих користувачів
5. Кнопка "Edit this film" відображається лише для авторизованих користувачів

# Створення Login та Register компонентів з англійським інтерфейсом

Адаптуємо компоненти авторизації з двомовними коментарями та англійським інтерфейсом.

## Крок 1: Створення компонентів

Спочатку створімо компоненти:

bash

ng generate component components/login --standalone

ng generate component components/register --standalone

## Крок 2: Login Component

### Файл: src/app/components/login/login.component.ts

typescript

/\*\*

\* Login Component - Handles user authentication

\*

\* Компонент входу - Обробляє автентифікацію користувача

\*/

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

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { Router, RouterLink } from '@angular/router';

import { AuthService } from '../../services/auth.service';

@Component({

selector: 'app-login',

standalone: true,

imports: [CommonModule, FormsModule, RouterLink],

templateUrl: './login.component.html',

styleUrls: ['./login.component.css']

})

export class LoginComponent {

// Form fields for user credentials

// Поля форми для облікових даних користувача

email = '';

password = '';

// Error message for failed login attempts

// Повідомлення про помилку для невдалих спроб входу

error = '';

constructor(private authService: AuthService, private router: Router) { }

/\*\*

\* Handle form submission for user login

\*

\* Обробка відправки форми для входу користувача

\*/

onSubmit(): void {

this.authService.login(this.email, this.password).subscribe({

next: () => this.router.navigate(['/']),

error: () => this.error = 'Incorrect email or password'

});

}

}

### Файл: src/app/components/login/login.component.html

html

<div class="container mt-5">

<div class="row justify-content-center">

<div class="col-md-6">

<div class="card">

<div class="card-header">

<!-- Login form title -->

<!-- Заголовок форми входу -->

<h3 class="text-center">Login</h3>

</div>

<div class="card-body">

<!-- Error message display -->

<!-- Відображення повідомлення про помилку -->

<div class="alert alert-danger" \*ngIf="error">{{ error }}</div>

<!-- Login form -->

<!-- Форма входу -->

<form (ngSubmit)="onSubmit()">

<div class="mb-3">

<label class="form-label">Email</label>

<input type="email" class="form-control" [(ngModel)]="email" name="email" required>

</div>

<div class="mb-3">

<label class="form-label">Password</label>

<input type="password" class="form-control" [(ngModel)]="password" name="password" required>

</div>

<button type="submit" class="btn btn-primary w-100">Sign In</button>

</form>

<!-- Link to registration page -->

<!-- Посилання на сторінку реєстрації -->

<div class="text-center mt-3">

<a routerLink="/register">Create an account</a>

</div>

</div>

</div>

</div>

</div>

</div>

### Файл: src/app/components/login/login.component.css

css

/\*\*

\* Styles for the login component

\*

\* Стилі для компонента входу

\*/

/\* Style for the login card \*/

/\* Стиль для картки входу \*/

.card {

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

border-radius: 8px;

}

.card-header {

background-color: #007bff;

color: white;

border-radius: 8px 8px 0 0;

}

/\* Style for form inputs \*/

/\* Стиль для полів форми \*/

.form-control:focus {

border-color: #007bff;

box-shadow: 0 0 0 0.2rem rgba(0, 123, 255, 0.25);

}

/\* Style for submit button \*/

/\* Стиль для кнопки відправки \*/

.btn-primary {

background-color: #007bff;

border-color: #007bff;

}

## Крок 3: Register Component

### Файл: src/app/components/register/register.component.ts

typescript

/\*\*

\* Register Component - Handles new user registration

\*

\* Компонент реєстрації - Обробляє реєстрацію нових користувачів

\*/

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

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { Router, RouterLink } from '@angular/router';

import { AuthService } from '../../services/auth.service';

@Component({

selector: 'app-register',

standalone: true,

imports: [CommonModule, FormsModule, RouterLink],

templateUrl: './register.component.html',

styleUrls: ['./register.component.css']

})

export class RegisterComponent {

// Form fields for new user registration

// Поля форми для реєстрації нового користувача

username = '';

email = '';

password = '';

// State flags for UI feedback

// Прапори стану для відгуку інтерфейсу

error = '';

success = false;

constructor(private authService: AuthService, private router: Router) { }

/\*\*

\* Handle form submission for user registration

\*

\* Обробка відправки форми для реєстрації користувача

\*/

onSubmit(): void {

this.authService.register(this.username, this.email, this.password).subscribe({

next: () => {

this.success = true;

// Redirect to login page after 2 seconds

// Перенаправлення на сторінку входу через 2 секунди

setTimeout(() => this.router.navigate(['/login']), 2000);

},

error: (err) => this.error = err.error?.message || 'Registration error'

});

}

}

### Файл: src/app/components/register/register.component.html

html

<div class="container mt-5">

<div class="row justify-content-center">

<div class="col-md-6">

<div class="card">

<div class="card-header">

<!-- Registration form title -->

<!-- Заголовок форми реєстрації -->

<h3 class="text-center">Register</h3>

</div>

<div class="card-body">

<!-- Success message display -->

<!-- Відображення повідомлення про успіх -->

<div class="alert alert-success" \*ngIf="success">

Registration successful! Redirecting...

</div>

<!-- Error message display -->

<!-- Відображення повідомлення про помилку -->

<div class="alert alert-danger" \*ngIf="error">{{ error }}</div>

<!-- Registration form (hidden after successful registration) -->

<!-- Форма реєстрації (приховується після успішної реєстрації) -->

<form (ngSubmit)="onSubmit()" \*ngIf="!success">

<div class="mb-3">

<label class="form-label">Username</label>

<input type="text" class="form-control" [(ngModel)]="username" name="username" required>

</div>

<div class="mb-3">

<label class="form-label">Email</label>

<input type="email" class="form-control" [(ngModel)]="email" name="email" required>

</div>

<div class="mb-3">

<label class="form-label">Password</label>

<input type="password" class="form-control" [(ngModel)]="password" name="password" required>

</div>

<button type="submit" class="btn btn-primary w-100">Sign Up</button>

</form>

<!-- Link to login page (hidden after successful registration) -->

<!-- Посилання на сторінку входу (приховується після успішної реєстрації) -->

<div class="text-center mt-3" \*ngIf="!success">

<a routerLink="/login">Already have an account? Sign in</a>

</div>

</div>

</div>

</div>

</div>

</div>

### Файл: src/app/components/register/register.component.css

css

/\*\*

\* Styles for the register component

\*

\* Стилі для компонента реєстрації

\*/

/\* Style for the registration card \*/

/\* Стиль для картки реєстрації \*/

.card {

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

border-radius: 8px;

}

.card-header {

background-color: #28a745;

color: white;

border-radius: 8px 8px 0 0;

}

/\* Style for form inputs \*/

/\* Стиль для полів форми \*/

.form-control:focus {

border-color: #28a745;

box-shadow: 0 0 0 0.2rem rgba(40, 167, 69, 0.25);

}

/\* Style for submit button \*/

/\* Стиль для кнопки відправки \*/

.btn-primary {

background-color: #28a745;

border-color: #28a745;

}

/\* Style for success message \*/

/\* Стиль для повідомлення про успіх \*/

.alert-success {

border-color: #d4edda;

background-color: #d4edda;

color: #155724;

}

## Крок 4: Тестування компонентів авторизації

Для тестування компонентів Login і Register:

bash

ng serve

Відкрийте браузер за адресою http://localhost:4200/ і перевірте:

### Login Component (/login):

1. Форма входу відображається коректно
2. Поля email і password працюють
3. Посилання на реєстрацію працює
4. При неправильних даних відображається повідомлення про помилку

### Register Component (/register):

1. Форма реєстрації відображається коректно
2. Всі поля (username, email, password) працюють
3. При успішній реєстрації відображається повідомлення про успіх
4. Автоматичне перенаправлення на сторінку входу через 2 секунди
5. Посилання на сторінку входу працює

# Створення Film Form Component з англійським інтерфейсом

Адаптуємо Film Form Component з двомовними коментарями та англійським інтерфейсом для додавання та редагування фільмів.

## Крок 1: Створення компонента

Спочатку створімо компонент:

bash

ng generate component components/film-form --standalone

## Крок 2: Оновлення film-form.component.ts

typescript

/\*\*

\* Film Form Component - Handles adding new films and editing existing films

\*

\* Компонент форми фільму - Обробляє додавання нових фільмів та редагування існуючих фільмів

\*/

import { Component, OnInit } from '@angular/core';

import { CommonModule } from '@angular/common';

import { FormsModule } from '@angular/forms';

import { Router, ActivatedRoute, RouterLink } from '@angular/router';

import { FilmService } from '../../services/film.service';

import { HttpClient } from '@angular/common/http';

@Component({

selector: 'app-film-form',

standalone: true,

imports: [CommonModule, FormsModule, RouterLink],

templateUrl: './film-form.component.html',

styleUrls: ['./film-form.component.css']

})

export class FilmFormComponent implements OnInit {

// Film object to hold form data

// Об'єкт фільму для зберігання даних форми

film = {

id: 0,

title: '',

release\_year: new Date().getFullYear(),

genre: '',

description: '',

poster\_url: ''

};

// Available genres for selection

// Доступні жанри для вибору

genres: string[] = [];

// State flags for component behavior

// Прапори стану для поведінки компонента

isEditing = false;

loading = false;

error = '';

success = '';

constructor(

private filmService: FilmService,

private router: Router,

private route: ActivatedRoute,

private http: HttpClient

) { }

/\*\*

\* Initialize component and determine if editing or creating

\*

\* Ініціалізує компонент і визначає, чи це редагування або створення

\*/

ngOnInit(): void {

// Load available genres for dropdown

// Завантаження доступних жанрів для випадаючого списку

this.loadGenres();

// Check if this is editing mode by checking for film ID in URL

// Перевірка чи це режим редагування через перевірку ID фільму в URL

this.route.params.subscribe(params => {

const filmId = params['id'];

if (filmId) {

this.isEditing = true;

this.loadFilm(parseInt(filmId));

}

});

}

/\*\*

\* Load available genres from the API

\*

\* Завантажує доступні жанри з API

\*/

loadGenres(): void {

this.http.get<string[]>('http://localhost:3001/api/genres').subscribe({

next: (genres) => this.genres = genres,

error: (err) => console.error('Error loading genres:', err)

});

}

/\*\*

\* Load film data for editing

\*

\* Завантажує дані фільму для редагування

\*/

loadFilm(id: number): void {

this.loading = true;

this.filmService.getFilmById(id).subscribe({

next: (film) => {

this.film = film;

this.loading = false;

},

error: (err) => {

this.error = 'Film not found';

this.loading = false;

}

});

}

/\*\*

\* Handle form submission for both create and update operations

\*

\* Обробка відправки форми для операцій створення та оновлення

\*/

onSubmit(): void {

this.loading = true;

this.error = '';

this.success = '';

// Prepare film data for submission

// Підготовка даних фільму для відправки

const filmData = {

title: this.film.title,

release\_year: this.film.release\_year,

genre: this.film.genre,

description: this.film.description,

poster\_url: this.film.poster\_url

};

if (this.isEditing) {

// Update existing film

// Оновлення існуючого фільму

this.filmService.updateFilm(this.film.id, filmData).subscribe({

next: (film) => {

this.success = 'Film updated successfully!';

this.loading = false;

setTimeout(() => this.router.navigate(['/film', film.id]), 1500);

},

error: (err) => {

console.error('Update error:', err);

this.error = 'Error updating film';

this.loading = false;

}

});

} else {

// Create new film

// Створення нового фільму

this.filmService.createFilm(filmData).subscribe({

next: (film) => {

this.success = 'Film added successfully!';

this.loading = false;

setTimeout(() => this.router.navigate(['/film', film.id]), 1500);

},

error: (err) => {

console.error('Creation error:', err);

this.error = 'Error adding film';

this.loading = false;

}

});

}

}

}

## Крок 3: Оновлення film-form.component.html

html

<div class="container mt-4">

<!-- Loading indicator for initial data -->

<!-- Індикатор завантаження для початкових даних -->

<div class="text-center my-5" \*ngIf="loading && !film.title">

<div class="spinner-border text-primary"></div>

<p>Loading film...</p>

</div>

<div class="row justify-content-center" \*ngIf="!loading || film.title">

<div class="col-md-8">

<div class="card">

<div class="card-header">

<!-- Dynamic title based on mode (edit/create) -->

<!-- Динамічний заголовок залежно від режиму (редагування/створення) -->

<h3>{{ isEditing ? 'Edit Film' : 'Add New Film' }}</h3>

</div>

<div class="card-body">

<!-- Success message display -->

<!-- Відображення повідомлення про успіх -->

<div class="alert alert-success" \*ngIf="success">

{{ success }}

</div>

<!-- Error message display -->

<!-- Відображення повідомлення про помилку -->

<div class="alert alert-danger" \*ngIf="error">

{{ error }}

</div>

<!-- Film form -->

<!-- Форма фільму -->

<form (ngSubmit)="onSubmit()">

<div class="mb-3">

<label class="form-label">Film Title</label>

<input type="text" class="form-control" [(ngModel)]="film.title" name="title" required>

</div>

<div class="row">

<div class="col-md-6 mb-3">

<label class="form-label">Release Year</label>

<input type="number" class="form-control" [(ngModel)]="film.release\_year" name="release\_year" min="1900" max="2100" required>

</div>

<div class="col-md-6 mb-3">

<label class="form-label">Genre</label>

<select class="form-select" [(ngModel)]="film.genre" name="genre" required>

<option value="" disabled selected>Select a genre</option>

<option \*ngFor="let genre of genres" [value]="genre">{{ genre }}</option>

</select>

</div>

</div>

<div class="mb-3">

<label class="form-label">Poster URL</label>

<input type="url" class="form-control" [(ngModel)]="film.poster\_url" name="poster\_url" required

placeholder="https://example.com/poster.jpg">

<small class="form-text text-muted">Enter an image URL (300x450px recommended)</small>

</div>

<div class="mb-3">

<label class="form-label">Description</label>

<textarea class="form-control" [(ngModel)]="film.description" name="description" rows="5" required></textarea>

</div>

<!-- Form action buttons -->

<!-- Кнопки дій форми -->

<div class="d-flex gap-2">

<button type="submit" class="btn btn-primary" [disabled]="loading">

<span \*ngIf="loading" class="spinner-border spinner-border-sm me-1"></span>

{{ isEditing ? 'Update' : 'Save' }}

</button>

<a routerLink="/" class="btn btn-secondary">Cancel</a>

</div>

</form>

</div>

</div>

</div>

</div>

</div>

## Крок 4: Створення CSS файлу для стилізації форми

Створіть файл film-form.component.css:

css

/\*\*

\* Styles for the film form component

\*

\* Стилі для компонента форми фільму

\*/

/\* Style for the form card \*/

/\* Стиль для картки форми \*/

.card {

box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);

border-radius: 8px;

}

.card-header {

background-color: #17a2b8;

color: white;

border-radius: 8px 8px 0 0;

}

/\* Style for form inputs \*/

/\* Стиль для полів форми \*/

.form-control, .form-select {

border-radius: 6px;

}

.form-control:focus, .form-select:focus {

border-color: #17a2b8;

box-shadow: 0 0 0 0.2rem rgba(23, 162, 184, 0.25);

}

/\* Style for textarea \*/

/\* Стиль для текстової області \*/

textarea.form-control {

resize: vertical;

min-height: 120px;

}

/\* Style for buttons \*/

/\* Стиль для кнопок \*/

.btn-primary {

background-color: #17a2b8;

border-color: #17a2b8;

}

.btn-primary:hover {

background-color: #138496;

border-color: #117a8b;

}

/\* Style for button container \*/

/\* Стиль для контейнера кнопок \*/

.d-flex.gap-2 {

gap: 0.5rem;

}

/\* Style for form validation \*/

/\* Стиль для валідації форми \*/

.form-control.is-invalid, .form-select.is-invalid {

border-color: #dc3545;

}

/\* Style for helper text \*/

/\* Стиль для допоміжного тексту \*/

.form-text {

font-size: 0.875rem;

}

## Крок 5: Тестування Film Form компонента

Для тестування Film Form компонента:

bash

ng serve

Відкрийте браузер і перевірте:

### Режим додавання нового фільму (/add-film):

1. Форма відображається з заголовком "Add New Film"
2. Всі поля форми працюють коректно
3. Випадаючий список жанрів завантажується з API
4. Кнопка "Save" активна
5. При успішному додаванні відображається повідомлення та перенаправлення

### Режим редагування фільму (/edit-film/:id):

1. Форма відображається з заголовком "Edit Film"
2. Поля форми заповнюються існуючими даними фільму
3. Кнопка показує "Update"
4. При успішному оновленні відображається повідомлення та перенаправлення

### Загальні функції:

1. Валідація полів працює
2. Кнопка "Cancel" перенаправляє на головну сторінку
3. Відображаються повідомлення про помилки
4. Loading індикатори працюють під час операцій

## Підсумок створених компонентів

Тепер у нас є всі необхідні компоненти для проекту "What to Watch?":

1. ✅ **Navbar Component** - Навігаційна панель
2. ✅ **Film List Component** - Список фільмів з фільтрацією
3. ✅ **Film Detail Component** - Деталі фільму та коментарі
4. ✅ **Login Component** - Авторизація користувача
5. ✅ **Register Component** - Реєстрація користувача
6. ✅ **Film Form Component** - Додавання/редагування фільмів

Всі компоненти мають англійський інтерфейс та двомовні коментарі в коді. Наступним кроком буде створення та налаштування бекенду для повної функціональності додатка.

# Структура бекенду проекту "What to Watch?" з повним кодом

Тепер надам весь код точно як в документі для ваших нотаток.

## Етап 1: Підготовка папок проекту

**Пояснення**: Створюємо базову структуру папок для всього проекту.

### Розташування файлів:

scss

what-to-watch/ (кореневу папку створюємо в будь-якому місці)

├── backend/ (папка для серверної частини)

├── frontend/ (папка для клієнтської частини)

└── infra/ (папка для інфраструктури Docker)

### Команди для створення:

bash

mkdir what-to-watch

cd what-to-watch

mkdir backend

mkdir frontend

mkdir infra

cd infra

## Етап 2: Створюємо docker-compose.yml

**Пояснення**: Цей файл налаштовує контейнери для бази даних MySQL та веб-інтерфейсу phpMyAdmin для управління базою даних.

### Розташування: infra/docker-compose.yml

bash

touch docker-compose.yml

yaml

version: '3'

services:

db:

image: mysql:8.0

container\_name: wtw\_mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: root

MYSQL\_DATABASE: wtw\_db

MYSQL\_USER: wtw\_user

MYSQL\_PASSWORD: wtw\_pass

ports:

- "3306:3306"

volumes:

- db\_data:/var/lib/mysql

phpmyadmin:

image: phpmyadmin/phpmyadmin

container\_name: wtw\_phpmyadmin

restart: always

ports:

- "8080:80"

environment:

PMA\_HOST: db

PMA\_USER: root

PMA\_PASSWORD: root

depends\_on:

- db

volumes:

db\_data:

**Файл містить налаштування для:**

* MySQL 8.0 контейнер на порті 3306
* phpMyAdmin контейнер на порті 8080
* Автоматичне створення бази даних wtw\_db
* Користувача wtw\_user з паролем wtw\_pass

## Етап 3: Запуск контейнерів

**Пояснення**: Запускаємо створені контейнери в фоновому режимі.

### Команди з папки infra/:

bash

docker-compose up -d

docker-compose ps

## Етап 4: Перевірка доступу до phpMyAdmin

**Пояснення**: Переконуємося, що веб-інтерфейс для управління базою даних доступний.

* **URL**: [http://localhost:8080](http://localhost:8080/)
* **Логін**: root
* **Пароль**: root

## Етап 5: Створення таблиць через SQL-скрипт

**Пояснення**: Створюємо структуру бази даних з таблицями для користувачів, фільмів та коментарів.

### Розташування: infra/init.sql

bash

touch init.sql

sql

-- Users

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(100) NOT NULL UNIQUE,

email VARCHAR(100) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP

);

-- Films

CREATE TABLE IF NOT EXISTS films (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

release\_year INT,

genre VARCHAR(100),

description TEXT,

poster\_url TEXT,

added\_by INT,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (added\_by) REFERENCES users(id)

);

-- Comments

CREATE TABLE IF NOT EXISTS comments (

id INT AUTO\_INCREMENT PRIMARY KEY,

film\_id INT NOT NULL,

user\_id INT NOT NULL,

content TEXT NOT NULL,

created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (film\_id) REFERENCES films(id),

FOREIGN KEY (user\_id) REFERENCES users(id)

);

-- Тестові дані

INSERT INTO users (username, email, password) VALUES

('demo', 'demo@example.com', 'demo\_pass');

INSERT INTO films (title, release\_year, genre, description, poster\_url, added\_by)

VALUES

('Le Fabuleux Destin d''Amélie Poulain', 2001, 'Comédie', 'Une jeune serveuse décide de changer la vie de ceux qui l''entourent.', 'https://via.placeholder.com/300x450?text=Amelie', 1),

('Inception', 2010, 'Science-fiction', 'Un voleur s''introduit dans les rêves pour voler des secrets.', 'https://via.placeholder.com/300x450?text=Inception', 1);

**Імпорт через phpMyAdmin:**

1. Відкрити [http://localhost:8080](http://localhost:8080/)
2. Вибрати базу wtw\_db
3. Import → Browse → вибрати init.sql → Go

### Тест підключення:

bash

docker exec -it wtw\_mysql mysql -uroot -proot -e "USE wtw\_db; SHOW TABLES;"

## Етап 2: Створення базового бекенду (Node.js + Express)

**Пояснення**: Розробляємо серверну частину на Node.js з Express фреймворком для обробки API запитів.

### Крок 1: Ініціалізація проекту Node.js

**Розташування команд**: з папки backend/

bash

cd ../backend

npm init -y

ls -la

### Крок 2: Встановлення необхідних пакетів

**Пояснення**: Встановлюємо основні залежності для роботи сервера.

bash

npm install express mysql2 cors dotenv

npm install nodemon --save-dev

cat package.json

### Крок 3: Створення структури файлів

**Розташування**: всі файли в папці backend/

bash

touch .env

touch server.js

mkdir routes

touch routes/films.js

touch routes/auth.js

mkdir config

touch config/db.js

### Крок 4: Налаштування змінних середовища

**Розташування**: backend/.env

ini

# Порт сервера

PORT=3001

# Налаштування бази даних

DB\_HOST=localhost

DB\_PORT=3306

DB\_USER=wtw\_user

DB\_PASSWORD=wtw\_pass

DB\_DATABASE=wtw\_db

### Крок 5: Налаштування підключення до бази даних

**Розташування**: backend/config/db.js

javascript

const mysql = require('mysql2/promise');

require('dotenv').config();

// Створення пулу підключень

const pool = mysql.createPool({

host: process.env.DB\_HOST,

port: process.env.DB\_PORT,

user: process.env.DB\_USER,

password: process.env.DB\_PASSWORD,

database: process.env.DB\_DATABASE,

waitForConnections: true,

connectionLimit: 10,

queueLimit: 0

});

// Функція для тестування підключення

async function testConnection() {

try {

const connection = await pool.getConnection();

console.log('БД підключена успішно!');

connection.release();

return true;

} catch (error) {

console.error('Помилка підключення до БД:', error.message);

return false;

}

}

module.exports = {

pool,

testConnection

};

### Крок 6: Створення базових маршрутів для фільмів

**Розташування**: backend/routes/films.js

javascript

const express = require("express");

const router = express.Router();

const { pool } = require("../config/db");

// GET /api/films - отримати всі фільми

router.get("/", async (req, res) => {

try {

const { genre, year } = req.query;

let query = "SELECT \* FROM films WHERE 1=1";

const params = [];

if (genre) {

query += " AND genre = ?";

params.push(genre);

}

if (year) {

query += " AND release\_year = ?";

params.push(parseInt(year));

}

query += " ORDER BY id DESC";

const [rows] = await pool.query(query, params);

res.json(rows);

} catch (error) {

res.status(500).json({ message: "Помилка сервера при отриманні фільмів" });

}

});

// GET /api/films/:id - отримати один фільм за ID

router.get("/:id", async (req, res) => {

try {

const filmId = req.params.id;

const [rows] = await pool.query("SELECT \* FROM films WHERE id = ?", [

filmId,

]);

if (rows.length === 0) {

return res.status(404).json({ message: "Фільм не знайдено" });

}

res.json(rows[0]);

} catch (error) {

res.status(500).json({ message: "Помилка сервера при отриманні фільму" });

}

});

// GET /api/films/:id/comments - отримати коментарі до фільму

router.get("/:id/comments", async (req, res) => {

try {

const filmId = req.params.id;

const [rows] = await pool.query(

`

SELECT c.\*, u.username

FROM comments c

JOIN users u ON c.user\_id = u.id

WHERE c.film\_id = ?

ORDER BY c.created\_at DESC

`,

[filmId]

);

res.json(rows);

} catch (error) {

res.status(500).json({ message: "Помилка сервера при отриманні коментарів" });

}

});

// POST /api/films/:id/comments - додати коментар до фільму

router.post("/:id/comments", async (req, res) => {

try {

const filmId = req.params.id;

const { content } = req.body;

const userId = 1;

const [films] = await pool.query("SELECT \* FROM films WHERE id = ?", [filmId]);

if (films.length === 0) {

return res.status(404).json({ message: "Film non trouvé" });

}

const [result] = await pool.query(

"INSERT INTO comments (film\_id, user\_id, content) VALUES (?, ?, ?)",

[filmId, userId, content]

);

const [comments] = await pool.query(

`

SELECT c.\*, u.username

FROM comments c

JOIN users u ON c.user\_id = u.id

WHERE c.id = ?

`,

[result.insertId]

);

res.status(201).json(comments[0]);

} catch (error) {

res.status(500).json({ message: "Помилка сервера при додаванні коментаря" });

}

});

// POST /api/films - створити новий фільм

router.post('/', async (req, res) => {

try {

const { title, release\_year, genre, description, poster\_url } = req.body;

if (!title || !release\_year || !genre || !description) {

return res.status(400).json({ message: 'Tous les champs obligatoires doivent être remplis' });

}

const userId = 1;

const [result] = await pool.query(

'INSERT INTO films (title, release\_year, genre, description, poster\_url, added\_by) VALUES (?, ?, ?, ?, ?, ?)',

[title, release\_year, genre, description, poster\_url || '', userId]

);

const [films] = await pool.query('SELECT \* FROM films WHERE id = ?', [result.insertId]);

res.status(201).json(films[0]);

} catch (error) {

res.status(500).json({ message: 'Помилка сервера при створенні фільму' });

}

});

// PUT /api/films/:id - оновити фільм

router.put('/:id', async (req, res) => {

try {

const filmId = req.params.id;

const { title, release\_year, genre, description, poster\_url } = req.body;

if (!title || !release\_year || !genre || !description) {

return res.status(400).json({ message: 'Tous les champs obligatoires doivent être remplis' });

}

const [films] = await pool.query('SELECT \* FROM films WHERE id = ?', [filmId]);

if (films.length === 0) {

return res.status(404).json({ message: 'Film non trouvé' });

}

await pool.query(

'UPDATE films SET title = ?, release\_year = ?, genre = ?, description = ?, poster\_url = ? WHERE id = ?',

[title, release\_year, genre, description, poster\_url || '', filmId]

);

const [updatedFilms] = await pool.query('SELECT \* FROM films WHERE id = ?', [filmId]);

res.json(updatedFilms[0]);

} catch (error) {

res.status(500).json({ message: 'Помилка сервера при оновленні фільму' });

}

});

module.exports = router;

### Крок 7: Створення базових маршрутів для авторизації

**Розташування**: backend/routes/auth.js

javascript

const express = require('express');

const router = express.Router();

const { pool } = require('../config/db');

// Функція перевірки авторизації

const isAuthenticated = (req, res, next) => {

if (req.headers.authorization) {

next();

} else {

res.status(401).json({ message: 'Необхідна авторизація' });

}

};

// POST /api/auth/login - вхід користувача

router.post('/login', async (req, res) => {

try {

const { email, password } = req.body;

const [users] = await pool.query(

'SELECT \* FROM users WHERE email = ? AND password = ?',

[email, password]

);

if (users.length === 0) {

return res.status(401).json({ message: 'Невірний email або пароль' });

}

const user = users[0];

delete user.password;

res.json({

id: user.id,

username: user.username,

email: user.email,

token: 'simple-auth-token'

});

} catch (error) {

res.status(500).json({ message: 'Помилка сервера при авторизації' });

}

});

// POST /api/auth/register - реєстрація нового користувача

router.post('/register', async (req, res) => {

try {

const { username, email, password } = req.body;

if (!username || !email || !password) {

return res.status(400).json({ message: 'Усі поля обов\'язкові' });

}

const [existingUsers] = await pool.query(

'SELECT \* FROM users WHERE email = ? OR username = ?',

[email, username]

);

if (existingUsers.length > 0) {

return res.status(400).json({ message: 'Користувач з таким email або ім\'ям вже існує' });

}

const [result] = await pool.query(

'INSERT INTO users (username, email, password) VALUES (?, ?, ?)',

[username, email, password]

);

res.status(201).json({

id: result.insertId,

username,

email,

message: 'Користувач успішно зареєстрований'

});

} catch (error) {

res.status(500).json({ message: 'Помилка сервера при реєстрації' });

}

});

// GET /api/auth/check - перевірка авторизації

router.get('/check', isAuthenticated, (req, res) => {

res.json({ authenticated: true });

});

module.exports = {

router,

isAuthenticated

};

Нижче — оновлений файл з мінімальними змінами для зберігання паролів у хешованому вигляді та перевірки під час логіну. Я чітко позначив, де саме змінив код, коментарями “ЗМІНА: …”.

Перед застосуванням: встанови залежність bcrypt

bash

cd backend

npm i bcrypt

Оновлений файл backend/routes/auth.js

js

const express = require('express');

const router = express.Router();

const { pool } = require('../config/db');

// ЗМІНА: додаємо bcrypt для хешування/перевірки паролів

const bcrypt = require('bcrypt'); // ЗМІНА

// Функція перевірки авторизації

const isAuthenticated = (req, res, next) => {

if (req.headers.authorization) {

next();

} else {

res.status(401).json({ message: 'Необхідна авторизація' });

}

};

// POST /api/auth/login - вхід користувача

router.post('/login', async (req, res) => {

try {

const { email, password } = req.body;

// ЗМІНА: спочатку знаходимо користувача тільки за email (без порівняння пароля в SQL)

const [users] = await pool.query(

'SELECT \* FROM users WHERE email = ?',

[email]

); // ЗМІНА: змінив SQL і параметри

if (users.length === 0) {

return res.status(401).json({ message: 'Невірний email або пароль' });

}

const user = users[0];

// ЗМІНА: порівнюємо переданий пароль із хешем у БД

const isMatch = await bcrypt.compare(password, user.password); // ЗМІНА

if (!isMatch) {

return res.status(401).json({ message: 'Невірний email або пароль' });

}

// Залишаємо як є: не повертаємо пароль

delete user.password;

res.json({

id: user.id,

username: user.username,

email: user.email,

token: 'simple-auth-token'

});

} catch (error) {

res.status(500).json({ message: 'Помилка сервера при авторизації' });

}

});

// POST /api/auth/register - реєстрація нового користувача

router.post('/register', async (req, res) => {

try {

const { username, email, password } = req.body;

if (!username || !email || !password) {

return res.status(400).json({ message: 'Усі поля обов\'язкові' });

}

const [existingUsers] = await pool.query(

'SELECT \* FROM users WHERE email = ? OR username = ?',

[email, username]

);

if (existingUsers.length > 0) {

return res.status(400).json({ message: 'Користувач з таким email або ім\'ям вже існує' });

}

// ЗМІНА: хешуємо пароль перед збереженням

const passwordHash = await bcrypt.hash(password, 10); // ЗМІНА

// ЗМІНА: записуємо в БД хеш (а не відкритий пароль)

const [result] = await pool.query(

'INSERT INTO users (username, email, password) VALUES (?, ?, ?)',

[username, email, passwordHash] // ЗМІНА

);

res.status(201).json({

id: result.insertId,

username,

email,

message: 'Користувач успішно зареєстрований'

});

} catch (error) {

res.status(500).json({ message: 'Помилка сервера при реєстрації' });

}

});

// GET /api/auth/check - перевірка авторизації

router.get('/check', isAuthenticated, (req, res) => {

res.json({ authenticated: true });

});

module.exports = {

router,

isAuthenticated

};

Пояснення змін

* Додавання bcrypt.
* Під час реєстрації: пароль хешується bcrypt.hash(password, 10) і до БД пишеться саме хеш.
* Під час логіну: користувач витягується лише за email, далі bcrypt.compare(plain, hash) перевіряє відповідність.
* SQL-запит логіну змінено: з пошуку по email+password на пошук лише по email.

Цього достатньо, щоб паролі зберігалися у “закритому” вигляді в базі даних. Застосовуй без додаткових опцій.

### Крок 8: Налаштування основного файлу сервера

**Розташування**: backend/server.js

javascript

const express = require('express');

const cors = require('cors');

require('dotenv').config();

// Імпорт модулів

const { testConnection } = require('./config/db');

const filmsRouter = require('./routes/films');

const { router: authRouter } = require('./routes/auth');

// Ініціалізація Express

const app = express();

const PORT = process.env.PORT || 3001;

// Middleware

app.use(cors());

app.use(express.json());

// Тестування підключення до бази даних при старті

testConnection();

// Базовий маршрут для перевірки роботи API

app.get('/api/health', (req, res) => {

res.json({ status: 'OK', message: 'API працює' });

});

// Жанри фільмів (статичні дані для фільтрації)

app.get('/api/genres', (req, res) => {

const genres = [

'Comédie', 'Drame', 'Action', 'Aventure', 'Thriller',

'Horreur', 'Romance', 'Science-fiction', 'Fantaisie', 'Historique',

'Biographique', 'Policier', 'Musique', 'Western', 'Guerre',

'Documentaire', 'Crime', 'Animation', 'Familial'

];

res.json(genres);

});

// Маршрути API

app.use('/api/films', filmsRouter);

app.use('/api/auth', authRouter);

// Запуск сервера

app.listen(PORT, () => {

console.log(`Сервер запущено на порту ${PORT}`);

});

### Крок 9: Налаштування скриптів запуску в package.json

**Розташування**: backend/package.json (замінити секцію "scripts")

json

"scripts": {

"start": "node server.js",

"dev": "nodemon server.js",

"test": "echo \"Error: no test specified\" && exit 1"

}

### Крок 10: Запуск і тестування сервера

**Команди з папки backend/**:

bash

npm run dev

**Очікувані повідомлення:**

yaml

Сервер запущено на порту 3001

БД підключена успішно!

### Крок 11: Тестування API за допомогою браузера або curl

**1. Перевірка здоров'я API:**

bash

curl http://localhost:3001/api/health

**Очікуваний результат:**

json

{"status":"OK","message":"API працює"}

**2. Отримання списку фільмів:**

bash

curl http://localhost:3001/api/films

**3. Отримання списку жанрів:**

bash

curl http://localhost:3001/api/genres

**4. Перевірка фільтрації за жанром:**

bash

curl "http://localhost:3001/api/films?genre=Comédie"

**5. Отримання деталей фільму за ID:**

bash

curl http://localhost:3001/api/films/1

## Загальна архітектура бекенду

bash

backend/

├── config/

│ └── db.js # Підключення до MySQL через пул з'єднань

├── routes/

│ ├── films.js # CRUD операції для фільмів + коментарі

│ └── auth.js # Спрощена авторизація без JWT

├── .env # Змінні середовища (порт, БД)

├── server.js # Express сервер + middleware + маршрути

└── package.json # Залежності та скрипти NPM

**API ендпоінти що створюються:**

* /api/health - статус сервера
* /api/genres - список жанрів (статичний)
* /api/films - CRUD для фільмів
* /api/films/:id/comments - робота з коментарями
* /api/auth/login|register - авторизація

Вся структура створює повноцінний REST API для додатку управління фільмами з базовою авторизацією та можливістю коментування.

# Вітаю! Проект "What to Watch?" повністю працює!

Чудово, що ви успішно запустили весь проект! Тепер у вас є повноцінний веб-додаток для керування фільмами.

## ✅ Що зараз працює:

### 🔧 ****Бекенд (Node.js + Express + MySQL)****

* **Сервер**: [http://localhost:3001](http://localhost:3001/)
* **База даних**: MySQL через Docker
* **API ендпоінти**: всі маршрути для фільмів, коментарів та авторизації
* **phpMyAdmin**: [http://localhost:8080](http://localhost:8080/) для управління БД

### 🌐 ****Фронтенд (Angular)****

* **Додаток**: [http://localhost:4200](http://localhost:4200/)
* **Компоненти**: всі сторінки працюють (список фільмів, деталі, авторизація, форми)
* **Маршрутизація**: перехід між сторінками
* **Інтерфейс**: англійською мовою з Bootstrap стилями

## 🚀 **Основний функціонал:**

1. **Перегляд фільмів** - список з фільтрацією за жанром і роком
2. **Деталі фільму** - повна інформація + коментарі
3. **Авторизація** - вхід і реєстрація користувачів
4. **Додавання фільмів** - форма для нових фільмів
5. **Редагування** - оновлення існуючих фільмів
6. **Коментарі** - додавання коментарів до фільмів

## 📝 **Для ваших нотаток:**

Проект повністю готовий для:

* ✅ Демонстрації роботи
* ✅ Подальшого розвитку
* ✅ Документації в дипломній роботі
* ✅ Презентації функціоналу

Якщо потрібно щось додати або змінити в проекті - просто скажіть! 🎬