**Титульний блок**

* **Дипломна робота**: “What to Watch?”
* **Автор**: SHAPOVAL Ruslan
* **Керівник**: [Прізвище Ім’я По батькові]
* **Заклад**: AFPA, програма DWWM
* **Місце і дата**: [Місто], 2025

**Анотація**

Мета роботи — розробити веб-застосунок “What to Watch?” для каталогізації фільмів із можливістю перегляду, фільтрації, додавання/редагування записів та коментування. Система побудована за клієнт-серверною архітектурою: фронтенд — Angular, бекенд — Node.js + Express, база даних — MySQL, інфраструктура — Docker. Реалізовано REST API, демонстраційну авторизацію та CRUD-операції. Проєкт демонструє компетенції з проєктування інтерфейсу, роботи з реляційною БД, створення API, організації інфраструктури та базових аспектів безпеки у навчальному контексті DWWM.

* **Ключові слова**: Angular, Node.js, Express, REST API, MySQL, Docker, авторизація, коментарі, фільми, JWT (план).

**Зміст**

* Вступ
  1. Опис проєкту й контекст
  2. Аналіз предметної області та вимог
  3. Огляд технологій і вибір стеку
  4. Проєктування системи
  5. Реалізація Front-end
  6. Реалізація Back-end
  7. Інфраструктура та розгортання
  8. Безпека
  9. Тестування і результати
  10. Оглядова аналітика (Veille)
* Висновки
* Список джерел
* Додатки

**Вступ**

Мета — створення простого та зручного каталогу фільмів для особистого використання або невеликих кіноклубів. Користувачі можуть переглядати фільми, фільтрувати їх за жанром і роком, переглядати деталі та залишати коментарі. Авторизовані користувачі можуть додавати та редагувати фільми. Мотивація — опанувати повний цикл веб-розробки від бази даних до інтерфейсу користувача, зрозуміти роботу REST API, організацію сервера та контейнеризацію (Docker).

* **Цілі**:
  + **Навчальні**: практика Angular, Express, MySQL та Docker; розуміння CRUD, форм, роутингу, взаємодії з API, базової безпеки.
  + **Технічні**: реалізувати робочий прототип зі спрощеною авторизацією, фільтрами, коментарями та документацією.
* **Результат**: повністю працюючий CRUD-застосунок із демонстраційною авторизацією та коментарями, інфраструктурою для локального запуску, SQL-скриптами та прикладами запитів.

**1. Опис проєкту й контекст**

**1.1. Короткий опис проєкту**

Проєкт “What to Watch?” — веб-додаток для управління каталогом фільмів із такими можливостями:

* Перегляд списку фільмів.
* Фільтрація за жанром та роком випуску.
* Перегляд детальної інформації (опис, рік, жанр, постер).
* Додавання коментарів до фільмів.
* Реєстрація та вхід у систему (спрощена авторизація).
* Додавання та редагування фільмів (для авторизованих користувачів).

Технічна основа:

* Інфраструктура: Docker (контейнери mysql:8.0 та phpmyadmin/phpmyadmin).
* База даних: MySQL (таблиці users, films, comments).
* Бекенд: Node.js + Express + mysql2, REST API.
* Клієнт: мінімальний інтерфейс для демонстрації роботи API; акцент у цій роботі — на бекенді та БД.
* Тестові дані: один користувач і два фільми для швидкої перевірки прототипу.

**1.2. Призначення та цінність**

* Для кіноманів і кіноклубів: швидкий спосіб вести базу улюблених фільмів, залишати коментарі й обмінюватися враженнями.
* Навчальна цінність: демонстрація повного циклу веб-розробки — від інфраструктури та проєктування БД до реалізації REST API.
* Для захисту DWWM (CP2–CP7): покриття ключових компетенцій на реальному робочому прототипі.
* Додаткова перевага: швидке відтворення середовища для журі завдяки Docker (одна команда).

**1.3. Цільова аудиторія**

* Кіномани, які хочуть структурувати переглянуті та заплановані до перегляду фільми.
* Невеликі кіноклуби для внутрішнього обміну думками.
* Студенти/молодші розробники, що вивчають стек Node.js + MySQL.

**1.4. Межі проєкту (scope)**

* Немає рекомендаційних алгоритмів або персоналізованих добірок.
* Ролі користувачів не деталізовані (адмін/модератор не реалізовано).
* Авторизація базова: без JWT, без хешування паролів, без refresh-токенів.
* Жанри — статичний список, що повертається бекендом (GET /api/genres).
* Фронтенд може бути мінімальним або в статусі інтеграції (фокус — інфраструктура, БД, API).

**1.5. Контекст створення (навчальний/професійний)**

Контекст: навчальний проєкт для підготовки до захисту професійного титулу DWWM (Франція). Мета — продемонструвати компетенції CP2–CP7 на завершеному прототипі:

* CP2–CP4: макетування інтерфейсу, верстка, базова клієнтська логіка (скріншоти/макети і мінімальний клієнт).
* CP5: проєктування реляційної БД (ER-схема, SQL-скрипти, базові зв’язки).
* CP6: доступ до даних через mysql2, параметризовані запити, CRUD.
* CP7: серверна логіка в Express: маршрути films, auth, обробка запитів, валідація на рівні контролера.

**1.6. Короткий огляд архітектури**

* infra/docker-compose.yml: піднімає mysql:8.0 і phpmyadmin/phpmyadmin.
* infra/init.sql: створює таблиці users, films, comments та вставляє тестові дані.
* backend/.env: конфігурація підключення до БД та порт сервера.
* backend/config/db.js: пул з’єднань MySQL + smoke-тест з’єднання.
* backend/routes/films.js: REST-ендпойнти для фільмів і коментарів, фільтрація за жанром/роком.
* backend/routes/auth.js: спрощена реєстрація/логін і middleware isAuthenticated.
* backend/server.js: ініціалізація Express, CORS/JSON, маршрути, GET /api/genres, GET /api/health.

**1.7. Ключові сценарії використання**

* Користувач відкриває каталог фільмів та фільтрує за жанром genre або роком release\_year.
* Переглядає деталі конкретного фільму.
* Додає коментар до фільму (у демо-версії — як авторизований користувач із фіксованим ідентифікатором).
* Реєструється і входить у систему для доступу до створення/редагування фільмів.
* Адмініструє БД через phpMyAdmin: перегляд таблиць, імпорт/експорт даних, перевірка структур.

**1.8. Обмеження та припущення**

* Спрощена безпека: паролі зберігаються у відкритому вигляді; токени — демонстраційні. Ризики усвідомлені; покращення передбачені у дорожній карті.
* Відсутність масштабування: немає шардінгу БД, кешування, балансування навантаження.
* Немає продакшн-обв’язки: централізованого логування, моніторингу, CI/CD.
* Сумісність середовища: розраховано на локальний запуск через Docker і Node.js LTS.

**1.9. Критерії успіху (для навчального контексту)**

* Середовище піднімається однією командою docker-compose up -d.
* База даних створюється та заповнюється з init.sql; таблиці доступні з phpMyAdmin.
* Бекенд стартує (npm run dev), відповідає на GET /api/health, повертає жанри GET /api/genres.
* CRUD-операції з фільмами працюють; коментарі додаються та віддаються по фільму.
* Авторизація в демо-режимі функціонує для реєстрації/логіну.

**1.10. Заплановані покращення (roadmap безпеки й якості)**

* Безпека: хешування паролів (bcrypt), перехід на JWT (access/refresh), політика CORS з переліком дозволених походжень, базова CSP.
* Валідація і помилки: zod/joi для перевірки вхідних даних, уніфікований обробник помилок.
* Права доступу: ролі та ACL, аудит дій.
* Архітектура коду: рефакторинг у шар сервісів, модульні тести ключових маршрутів.
* Документація: OpenAPI/Swagger для API, ER-діаграма, діаграма послідовності запитів.

**2. Аналіз предметної області та вимог**

**2.1. Бізнес-вимоги**

* **Каталог фільмів зі швидким переглядом**: користувач бачить перелік фільмів у зручному картковому або табличному форматі (постер, назва, рік, жанр) з можливістю переходу до деталей.
* **Фільтрація за жанром та роком випуску**: відбір списку за параметрами genre, release\_year без повного перезавантаження сторінки.
* **Детальна сторінка фільму**: повний опис, рік, жанр, постер, агреговані коментарі користувачів.
* **Коментарі до фільмів**: перегляд існуючих коментарів і можливість додавання нових.
* **Реєстрація та вхід користувача**: створення облікового запису та базова авторизація для персоналізації функцій.
* **Додавання та редагування фільмів (для авторизованих користувачів)**: створення нових записів, редагування атрибутів фільмів.

**2.2. Нефункціональні вимоги**

* **Простий запуск інфраструктури (Docker)**: підняття контейнерів MySQL та phpMyAdmin однією командою; бекенд запускається локально.
* **Прийнятна швидкодія для невеликої кількості записів**: коректний відгук інтерфейсу та API при десятках—сотнях записів завдяки індексам та легкому стеку.
* **Базова безпека й підготовка до розширення**: мінімальна авторизація у демо; дорожня карта — bcrypt для паролів, JWT, посилення CORS/CSP.
* **Читабельна структура коду**: поділ на модулі routes, config, controllers; параметризовані SQL-запити; лаконічні контролери.
* **Локалізація інтерфейсу**: підтримка англійської/української з можливістю розширення.

**2.3. Приклади user stories**

* Як гість, я хочу бачити список фільмів і фільтрувати, щоб швидко знайти, що подивитися.
* Як користувач, я хочу бачити деталі фільму, щоб дізнатися опис, рік, жанр.
* Як авторизований користувач, я хочу додавати нові фільми.
* Як авторизований користувач, я хочу залишати коментарі.

**3. Огляд технологій і вибір стеку**

**3.1. Обрані технології**

* **Angular** — фронтенд-фреймворк для SPA з підтримкою роутингу, форм, сервісів, DI та HttpClient для роботи з API.
* **Node.js + Express** — легкий у налаштуванні серверний стек для створення REST API з мінімалістичним ядром і великою екосистемою пакунків.
* **MySQL** — реляційна СУБД для структурованих даних та зв’язків (таблиці users, films, comments).
* **Docker + phpMyAdmin** — контейнеризоване розгортання БД і зручна веб-консоль для адміністрування.

**3.2. Обґрунтування вибору**

* **Angular**:
  + Чітка архітектура (модулі, компоненти, сервіси), двостороння прив’язка даних, вбудовані форми (Reactive/Template-driven).
  + Зріла система роутингу, guard-и для захисту маршрутів, HttpClient з інтерсепторами.
  + Підходить для багатосторінкових інтерфейсів, де важливі формальні патерни та масштабованість.
* **Express (на Node.js)**:
  + Швидке оголошення маршрутів і middleware, гнучкість у побудові контролерів і сервісного шару.
  + Просте підключення до MySQL через mysql2, підтримка параметризованих запитів.
  + Розвинена екосистема: валідація (joi/zod), безпека (helmet, cors), логування (morgan).
* **MySQL**:
  + Звична SQL-мова запитів, індекси, зовнішні ключі, стабільність.
  + Підходить для каталогу фільмів і коментарів із чіткими зв’язками 1:N.
  + Добра інтеграція з інструментами (phpMyAdmin, клієнти, драйвери).
* **Docker + phpMyAdmin**:
  + Відтворюване середовище: docker-compose up -d для підняття БД і консолі.
  + Зручне керування БД (перегляд таблиць, експорт/імпорт, SQL-консоль) без локальних інсталяцій.

**3.3. Як стек відповідає вимогам проєкту**

* **Бізнес-вимоги**: фільтрація, деталі, коментарі, авторизація — реалізуються через Angular-компоненти та REST-ендпойнти Express із доступом до MySQL.
* **Нефункціональні вимоги**:
  + Простий запуск — завдяки Docker.
  + Прийнятна швидкодія на малих обсягах — завдяки легкому Express та індексам у MySQL.
  + Читабельна структура — модульність Angular та розділення routes/config/controllers у бекенді.
  + Локалізація — Angular i18n або ngx-translate.
  + Базова безпека — cors, санітизація даних; підготовка до bcrypt/JWT.

**3.4. Архітектурна ув’язка компонентів**

* Клієнт (Angular) викликає REST API через HttpClient →
* Сервер (Express) приймає HTTP-запити, виконує валідацію та бізнес-логіку →
* Доступ до даних через драйвер mysql2 з параметризованими запитами →
* Зберігання даних у MySQL (таблиці users, films, comments) →
* Адміністрування та інспекція даних через phpMyAdmin у Docker-контейнері.

**3.5. Розгляд альтернатив**

* **Фронтенд**:
  + React або Vue — гнучкі рішення з меншою “рамкою”, але потребують додаткових рішень для роутингу/стану.
  + Angular обрано для швидкого старту в повноцінній SPA-архітектурі та суворої структури проєкту.
* **Бекенд**:
  + NestJS (над Express) — дає модульність, DI та декоратори; підвищує формалізацію, але додає криву навчання.
  + Express обрано за простоту і швидкість навчання/реалізації у навчальному контексті.
* **База даних**:
  + PostgreSQL — сильна у складних запитах і типах даних.
  + Для цього проєкту MySQL достатня й добре знайома, що скорочує час налаштування.

**3.6. Обмеження обраного стеку й план еволюції**

* **Безпека**: базова реалізація; план — bcrypt для паролів, JWT (access/refresh), посилення CORS/CSP.
* **Тестування**: спочатку мануальні перевірки; план — модульні тести (Jest/Supertest) для бекенду.
* **Документація API**: план — специфікація OpenAPI (Swagger UI) для прозорої інтеграції.
* **Організація коду**: поступовий рефакторинг у шар сервісів і репозиторіїв.

# 4. Проєктування системи (чистова версія, готово до вставки)

Нижче — відредагований і узгоджений розділ 4 у форматі Markdown. Можна вставляти безпосередньо у Word.

## 4.1. Архітектура

Архітектура — клієнт–сервер із чітким поділом на шари:

* **Клієнт (Angular)**: відображає інтерфейс, керує навігацією, надсилає запити до REST API через HttpClient, обробляє стани завантаження/помилок.
* **Сервер (Express)**: приймає HTTP-запити, виконує валідацію та бізнес-логіку, взаємодіє з БД через драйвер mysql2, формує узгоджені JSON-відповіді.
* **База даних (MySQL)**: зберігає дані про користувачів, фільми та коментарі, забезпечує зв’язки 1:N та базову цілісність.

Основні принципи:

* **Розділення відповідальностей**: UI ↔ API ↔ Data.
* **HTTP-методи для CRUD**: GET, POST, PUT (і опційно DELETE).
* **Простота і прозорість**: передбачувані маршрути, послідовні статус-коди, уніфікований формат помилок.
* **Модульність**: поділ бекенду на routes/controllers/config.
* **Безпека на рівні MVP**: базова авторизація, CORS, санітизація; план еволюції — bcrypt, JWT, посилення політик.

## 4.2. ER-діаграма БД

Сутності (згідно з фактичним init.sql):

* users(id INT PK AI, username VARCHAR(100) UNIQUE NOT NULL, email VARCHAR(100) UNIQUE NOT NULL, password VARCHAR(255) NOT NULL, created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP)
* films(id INT PK AI, title VARCHAR(255) NOT NULL, release\_year INT, genre VARCHAR(100), description TEXT, poster\_url TEXT, added\_by INT FK → users.id, created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP)
* comments(id INT PK AI, film\_id INT FK → films.id NOT NULL, user\_id INT FK → users.id NOT NULL, content TEXT NOT NULL, created\_at DATETIME DEFAULT CURRENT\_TIMESTAMP)

Зв’язки:

* films.added\_by → users.id (фільм доданий певним користувачем)
* comments.film\_id → films.id (коментарі належать фільму)
* comments.user\_id → users.id (коментар залишено користувачем)

Примітки щодо моделювання:

* **Жанри як текст**: films.genre зберігає значення як VARCHAR(100) — просто й достатньо для MVP.
* **Дати**: використовується DATETIME DEFAULT CURRENT\_TIMESTAMP (без автооновлення полем TIMESTAMP).
* **Зовнішні ключі**: визначені без політик ON DELETE/UPDATE — поведінка за замовчуванням RESTRICT. Це означає, що:
  + видалення користувача з прив’язаними фільмами/коментарями буде заборонено (або потрібно спочатку видалити залежні записи),
  + видалення фільму заблокується, якщо є коментарі.

Рекомендовані індекси (для продуктивності):

* INDEX idx\_films\_year (release\_year)
* INDEX idx\_films\_genre (genre)
* INDEX idx\_comments\_film (film\_id), INDEX idx\_comments\_user (user\_id)
* INDEX idx\_films\_added\_by (added\_by)

[Місце для ER-діаграми — у Додатках]

sql

-- Рекомендовані індекси (не змінюють схему, але пришвидшують запити)

CREATE INDEX idx\_films\_year ON films (release\_year);

CREATE INDEX idx\_films\_genre ON films (genre);

CREATE INDEX idx\_films\_added\_by ON films (added\_by);

CREATE INDEX idx\_comments\_film ON comments (film\_id);

CREATE INDEX idx\_comments\_user ON comments (user\_id);

## 4.3. Модель навігації (Front-end)

Маршрути:

* / — список фільмів із фільтрами
* /film/:id — деталі фільму + коментарі
* /add — додавання фільму
* /edit/:id — редагування фільму
* /login — вхід
* /register — реєстрація

Компоненти:

* Navbar, FilmList, FilmDetail, FilmForm, Login, Register

Приклад оголошення маршрутів (Angular):

ts

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

import { FilmListComponent } from './features/films/film-list.component';

import { FilmDetailComponent } from './features/films/film-detail.component';

import { FilmFormComponent } from './features/films/film-form.component';

import { LoginComponent } from './features/auth/login.component';

import { RegisterComponent } from './features/auth/register.component';

export const routes: Routes = [

{ path: '', component: FilmListComponent },

{ path: 'film/:id', component: FilmDetailComponent },

{ path: 'add', component: FilmFormComponent },

{ path: 'edit/:id', component: FilmFormComponent },

{ path: 'login', component: LoginComponent },

{ path: 'register', component: RegisterComponent },

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

];

## 4.4. Діаграма послідовності (приклади сценаріїв)

### Логін

1. Користувач заповнює форму входу в Angular.
2. POST /api/auth/login з email/password.
3. Сервер (Express):
   * перевіряє користувача та пароль у таблиці users (у MVP — базова перевірка у відкритому вигляді),
   * повертає простий токен/ідентифікатор сесії.
4. Фронтенд зберігає токен у пам’яті застосунку (або sessionStorage для демо).
5. Захищені дії (додавання/редагування) доступні умовно при наявності токена.

Приклад відповіді:

json

{ "user": { "id": 1, "username": "demo" }, "token": "demo-token" }

### Додавання фільму

1. Авторизований користувач відкриває /add та заповнює форму.
2. POST /api/films з тілом: title, release\_year, genre, description, poster\_url.
3. Сервер:
   * валідує поля,
   * записує в films з added\_by = user.id,
   * повертає 201 Created із створеним ресурсом.
4. Фронтенд переходить на /film/:id.

Приклад відповіді:

json

{

"id": 2,

"title": "Inception",

"release\_year": 2010,

"genre": "Science-fiction",

"description": "Un voleur s'introduit dans les rêves...",

"poster\_url": "https://via.placeholder.com/300x450?text=Inception",

"added\_by": 1,

"created\_at": "2025-01-01 12:00:00"

}

## 4.5. Карта REST API (огляд)

* GET /api/films?genre=&year= — перелік з фільтрами
* GET /api/films/:id — деталі фільму (за потреби окремо GET /api/films/:id/comments)
* POST /api/films — створення фільму (авторизовано)
* PUT /api/films/:id — оновлення фільму (авторизовано)
* POST /api/comments або POST /api/films/:id/comments — додання коментаря (авторизовано)
* GET /api/genres — статичний перелік жанрів
* POST /api/auth/register — реєстрація
* POST /api/auth/login — вхід

Статуси: 200, 201, 400, 401, 404, 500.

## 4.6. Формати даних і валідація (узгоджені з БД)

* title — обов’язково, 1–255 символів.
* release\_year — число в межах 1888;поточнийрік1888;поточнийрік.
* genre — до 100 символів; значення з GET /api/genres.
* description — текст довільної довжини (поле TEXT).
* poster\_url — рядок (тип TEXT, зберігає повні URL).
* username — унікальне, до 100 символів; email — унікальне, до 100 символів.

Уніфікований формат помилок:

json

{

"error": {

"code": "VALIDATION\_ERROR",

"message": "Title is required",

"details": { "title": "required" }

}

}

## 4.7. Нефункціональні аспекти реалізації

* **Продуктивність**: додаткові індекси (див. вище) для швидкої фільтрації та JOIN-ів.
* **Локалізація**: Angular i18n/ngx-translate для EN/UA.
* **Безпека**:
  + Наявний стан: паролі зберігаються як plain text (MVP).
  + План: bcrypt для паролів; JWT (access/refresh); обмеження CORS; helmet.
* **Експлуатація**: запуск MySQL/phpMyAdmin через Docker; .env для конфігурації; GET /api/health для перевірки стану.

## 4.8. Узгодження з фактичними seed-даними

Тестові дані з init.sql:

sql

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);

* Передбачається, що перший користувач має id = 1, тому поле added\_by у стартових фільмах валідне.
* Для локальних демо-запусків це дозволяє одразу протестувати список, деталі, фільтри та додавання коментарів.

# 5. Реалізація Front-end (Standalone Angular) — чистова версія, готово до вставки

Повний вихідний код фронтенду та бекенду буде доступний у репозиторії.  
Посилання на репозиторій (заглушка): <https://example.com/repo-will-be-added>  
У тексті диплома це посилання можна замінити на фінальний публічний URL і (рекомендовано) додати тег або хеш коміту для фіксації версії.

## 5.1. Архітектурний огляд фронтенду

* **Підхід: Standalone Angular**. Відсутній AppModule; застосунок завантажується через bootstrapApplication. Налаштування провайдерів у app.config.ts, маршрути в app.routes.ts, компоненти мають standalone: true.
* **Структура проєкту (скорочено)**:
  + src/app/app.config.ts — конфігурація застосунку: provideRouter(routes), provideHttpClient(...)
  + src/app/app.routes.ts — маршрути з loadComponent (lazy)
  + src/app/app.component.ts — кореневий компонент: Navbar + RouterOutlet
  + src/app/services/ — AuthService, FilmService
  + src/app/components/ — navbar/, film-list/, film-detail/, film-form/, auth/login, auth/register
* **Базові принципи**:
  + **Розділення відповідальностей**: компоненти відповідають за UI та стан відображення; сервіси — за роботу з API.
  + **Прозорість і простота**: передбачувані маршрути, уніфіковані відповіді від API, обробка помилок.
  + **Адаптивний інтерфейс**: сітка та утилітарні класи Bootstrap для responsive-верстки.

## 5.2. Ключові компоненти й сервіси

* **NavbarComponent**: глобальна навігація, показує посилання “Додати” для авторизованих, кнопки входу/виходу.
* **FilmListComponent**: перелік фільмів із фільтрами за жанром/роком. Джерело даних — FilmService.getFilms.
* **FilmDetailComponent**: детальна сторінка фільму; завантаження і додавання коментарів.
* **FilmFormComponent**: створення/редагування фільму; проста валідація; POST/PUT через FilmService.
* **LoginComponent / RegisterComponent**: форми автентифікації; використовують AuthService.
* **AuthService**: вхід/реєстрація, просте зберігання токена у sessionStorage, метод isAuthenticated.
* **FilmService**: інкапсульовані запити до /api/films, /api/genres, /api/films/:id/comments.

Взаємодія: компоненти викликають сервіси → сервіси виконують HTTP-запити → компоненти оновлюють локальні стани loading, error, data та шаблони.

## 5.3. Фрагменти коду (репрезентативні)

### 5.3.1. Оголошення standalone-компонента

ts

// src/app/app.component.ts

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>

`

})

export class AppComponent {}

### 5.3.2. Маршрут з lazy loadComponent

ts

// src/app/app.routes.ts

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

export const routes: Routes = [

{ path: '', loadComponent: () => import('./components/film-list/film-list.component').then(m => m.FilmListComponent) },

{ path: 'film/:id', loadComponent: () => import('./components/film-detail/film-detail.component').then(m => m.FilmDetailComponent) },

{ path: 'add', loadComponent: () => import('./components/film-form/film-form.component').then(m => m.FilmFormComponent) },

{ path: 'edit/:id', loadComponent: () => import('./components/film-form/film-form.component').then(m => m.FilmFormComponent) },

{ path: 'login', loadComponent: () => import('./components/auth/login.component').then(m => m.LoginComponent) },

{ path: 'register', loadComponent: () => import('./components/auth/register.component').then(m => m.RegisterComponent) },

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

];

### 5.3.3. Конфігурація застосунку

ts

// src/app/app.config.ts

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

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

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

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

export const appConfig: ApplicationConfig = {

providers: [

provideRouter(routes),

provideHttpClient(withFetch())

]

};

### 5.3.4. Запит через HttpClient у сервісі

ts

// src/app/services/film.service.ts (уривок)

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

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

import { Observable } from 'rxjs';

const BASE = 'http://localhost:3001/api';

@Injectable({ providedIn: 'root' })

export class FilmService {

constructor(private http: HttpClient) {}

getFilms(params: { genre?: string; year?: string | number }): Observable<any[]> {

let p = new HttpParams();

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

if (params.year) p = p.set('year', String(params.year));

return this.http.get<any[]>(`${BASE}/films`, { params: p });

}

}

### 5.3.5. Форма з двостороннім зв’язком [(ngModel)]

ts

// src/app/components/film-form/film-form.component.ts (уривок)

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

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

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

@Component({

standalone: true,

selector: 'app-film-form',

imports: [CommonModule, FormsModule],

template: `

<form (ngSubmit)="submit()" class="row g-3">

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

<label class="form-label">Назва \*</label>

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

</div>

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

<label class="form-label">Рік</label>

<input type="number" class="form-control" [(ngModel)]="model.release\_year" name="release\_year" />

</div>

<div class="col-12 text-end">

<button class="btn btn-primary" type="submit">Зберегти</button>

</div>

</form>

`

})

export class FilmFormComponent {

model: any = { title: '', release\_year: '' };

submit() {/\* ... \*/}

}

## 5.4. Схеми/діаграми (текстові описи)

* **Модель навігації**:
  + / — список фільмів із фільтрами
  + /film/:id — деталі фільму + коментарі
  + /add — створення нового фільму
  + /edit/:id — редагування існуючого фільму
  + /login, /register — сторінки автентифікації
* **Послідовність “Перегляд деталей фільму”**:
  + Користувач переходить на /film/:id.
  + Компонент викликає GET /api/films/:id для завантаження даних фільму.
  + Далі — GET /api/films/:id/comments для коментарів.
  + UI показує дані; авторизований користувач бачить форму додавання коментаря.
* **Взаємодія з API**:
  + Компонент → Сервіс (HttpClient) → REST API (Express) → MySQL.
  + Стан керується локально в компонентах: loading, error, data.

[Візуальні діаграми навігації та послідовності додаються у Додатках.]

## 5.5. Налаштування: інтеграція Bootstrap та базові конфіги

* **Інтеграція Bootstrap у angular.json**:

json

{

"projects": {

"what-to-watch": {

"architect": {

"build": {

"options": {

"styles": [

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

"src/styles.css"

],

"scripts": [

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

]

}

}

}

}

}

}

* **Встановлення залежності**:

bash

npm install bootstrap

* **Кореневе завантаження застосунку**:

ts

// src/main.ts

import { bootstrapApplication } from '@angular/platform-browser';

import { AppComponent } from './app/app.component';

import { appConfig } from './app/app.config';

bootstrapApplication(AppComponent, appConfig).catch(console.error);

## 5.6. Пояснення рішень, компроміси, альтернативи

* **Standalone Angular**:
  + Плюси: менше “модульного” коду, прозорі залежності, зручний lazy loading через loadComponent.
  + Мінуси: вимагає сучасної версії Angular та сумісних бібліотек.
* **Сервіси для роботи з даними**:
  + Плюси: компоненти залишаються “тонкими” і зосередженими на UI.
  + Компроміс: стани зберігаються локально без глобального стору.
* **Форми на [(ngModel)] (Template-driven)**:
  + Плюси: швидка розробка для MVP.
  + Альтернатива: Reactive Forms для складної валідації, але більше коду.
* **Bootstrap як UI-фреймворк**:
  + Плюси: швидке отримання адаптивного інтерфейсу.
  + Альтернативи: Angular Material для більш “native” Angular-компонентів, або Tailwind для утилітарної стилізації.
* **Авторизація (MVP)**:
  + Поточна проста схема: токен у sessionStorage, умовний доступ до дій.
  + Подальший розвиток: HTTP-інтерсептор для Authorization, authGuard для захисту маршрутів, перехід на JWT.
* **Конфігурація адреси API**:
  + Зараз у прикладах: BASE = 'http://localhost:3001/api'.
  + Рекомендація: винести в environment.ts для різних оточень dev/prod.

## 5.7. Посилання на репозиторій

* Заглушка: <https://example.com/repo-will-be-added>
* Після публікації вкажіть у дипломі реальний URL і, за можливості, тег або хеш коміту, наприклад:  
  “Повний вихідний код доступний у репозиторії: <https://github.com/username/what-to-watch> (коміт 3f2c9e7, від 2025‑05‑12).”

# 6. Реалізація Back-end (огляд і приклади) — чистова версія, готово до вставки

У цьому розділі наведено огляд архітектури бекенду, ключові маршрути, репрезентативні фрагменти коду, схеми взаємодії, валідацію/помилки та налаштування запуску. Повний вихідний код міститься у проєктних файлах та/або Додатках.

## 6.1. Архітектура і структура

* **Стек**: Node.js, Express, MySQL (mysql2/promise), cors, dotenv.
* **Підхід**: мінімалістична Express-архітектура без фреймворків-надбудов.
* **Структура папок (скорочено)**:
  + backend/server.js — ініціалізація Express, middleware, health-check, статичні жанри, підключення роутерів.
  + backend/config/db.js — пул підключень MySQL, testConnection().
  + backend/routes/films.js — ендпойнти для фільмів і коментарів.
  + backend/routes/auth.js — ендпойнти автентифікації та проста перевірка авторизації.

## 6.2. Ключові можливості та маршрути

* **Системні ендпойнти**
  + GET /api/health — перевірка доступності API.
  + GET /api/genres — статичний список жанрів для фронтенд-фільтрації.
* **Фільми**
  + GET /api/films — список фільмів із фільтрами ?genre=&year=.
  + GET /api/films/:id — деталі фільму.
  + POST /api/films — створити фільм (перевірка обов’язкових полів).
  + PUT /api/films/:id — оновити фільм.
  + GET /api/films/:id/comments — отримати коментарі з іменем користувача.
  + POST /api/films/:id/comments — додати коментар (демо: userId = 1).
* **Авторизація**
  + POST /api/auth/login — вхід із перевіркою email/password, повертає простий токен.
  + POST /api/auth/register — реєстрація з перевіркою унікальності email/username.
  + GET /api/auth/check — перевірка наявності заголовка Authorization.

## 6.3. Репрезентативні фрагменти коду

### 6.3.1. Ініціалізація сервера та системні маршрути

js

// backend/server.js (уривок)

const express = require('express');

const cors = require('cors');

require('dotenv').config();

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

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

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

const app = express();

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

app.use(cors());

app.use(express.json());

testConnection();

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

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

});

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

res.json([

'Comédie','Drame','Action','Aventure','Thriller','Horreur','Romance',

'Science-fiction','Fantaisie','Historique','Biographique','Policier',

'Musique','Western','Guerre','Documentaire','Crime','Animation','Familial'

]);

});

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

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

app.listen(PORT, () => console.log(`Сервер запущено на порту ${PORT}`));

### 6.3.2. Пул підключень до БД і тест підключення

js

// backend/config/db.js (уривок)

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

require('dotenv').config();

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

});

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 };

### 6.3.3. Список фільмів із фільтрами (параметризований SQL)

js

// backend/routes/films.js (уривок)

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 {

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

}

});

### 6.3.4. Приклад створення ресурсу з валідацією

js

// backend/routes/films.js (уривок)

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 {

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

}

});

### 6.3.5. Простий логін і перевірка авторизації

js

// backend/routes/auth.js (уривок)

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

if (req.headers.authorization) next();

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

};

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 {

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

}

});

router.get('/check', isAuthenticated, (\_req, res) => res.json({ authenticated: true }));

## 6.4. Схеми/послідовності (текстовий опис)

* **Навігація ендпойнтів**
  + /api/health — стан API
  + /api/genres — жанри
  + /api/films[?genre=&year=], /api/films/:id
  + /api/films/:id/comments (GET/POST)
  + /api/auth/login, /api/auth/register, /api/auth/check
* **Послідовність: перегляд деталей фільму**
  + Клієнт викликає GET /api/films/:id → повертаються атрибути фільму.
  + Клієнт викликає GET /api/films/:id/comments → повертаються коментарі з username.
  + UI показує деталі та список коментарів.
* **Послідовність: авторизація**
  + POST /api/auth/login з email/password.
  + Якщо знайдено користувача — повертається простий токен.
  + Подальші запити можуть містити заголовок Authorization (для демо — перевірка в GET /api/auth/check).

## 6.5. Валідація і коди відповідей

* **Обов’язкові поля**:
  + Фільм: title, release\_year, genre, description.
  + Користувач: username, email, password (на реєстрації).
* **Типові статуси**:
  + 200 OK — успішні читання/оновлення
  + 201 Created — створення (фільм, коментар, користувач)
  + 400 Bad Request — пропущені/некоректні поля
  + 401 Unauthorized — відсутня авторизація
  + 404 Not Found — ресурс не знайдено
  + 500 Internal Server Error — системна помилка
* **Формат повідомлень**:
  + Успіх: повертається створений/оновлений ресурс.
  + Помилка: лаконічні {"message": "..."}; за потреби можна уніфікувати як:

json

{

"error": {

"code": "VALIDATION\_ERROR",

"message": "Title is required",

"details": { "title": "required" }

}

}

## 6.6. Налаштування та запуск (коротко)

* **Змінні середовища (.env)**:
  + PORT=3001
  + DB\_HOST=..., DB\_PORT=3306
  + DB\_USER=..., DB\_PASSWORD=..., DB\_DATABASE=...
* **Скрипти npm (приклад)**:
  + start: запуск серверу (наприклад, node backend/server.js)
  + dev: запуск із nodemon для hot-reload
* **Ініціалізація БД**:
  + Створити схеми/таблиці (за вашим init.sql).
  + Переконатися, що облікові записи користувачів існують для тестового login.

## 6.7. Пояснення рішень і компроміси

* **Простий токен, простий isAuthenticated**: демо-рішення для спрощення логіки.
  + Альтернатива для продакшену: JWT, bcrypt для хешування паролів, middleware для витягання userId, ролі й політики доступу (ACL).
* **Параметризовані запити**: захист від SQL-ін’єкцій без ORM.
  + Альтернатива: ORM/Query builders (Sequelize/Prisma/Knex) для складніших проєктів.
* **Статичні жанри**: менше таблиць і простіший бекенд.
  + Альтернатива: таблиця genres та відношення з films.

Примітка: повні листинги server.js, config/db.js, routes/films.js, routes/auth.js наводяться у Додатках або доступні в репозиторії проєкту.

# 7. Інфраструктура та розгортання — чистова версія, готово до вставки

У цьому розділі описано локальну інфраструктуру застосунку “What to Watch?” на базі Docker, яка забезпечує швидкий запуск усіх компонентів системи та однакові умови роботи на різних машинах.

## 7.1. Компоненти інфраструктури

Інфраструктура включає наступні сервіси, що запускаються через Docker Compose:

* **MySQL 8.0** — база даних (контейнер: wtw\_mysql, порт: 3306)
* **phpMyAdmin** — веб-інтерфейс для керування БД (контейнер: wtw\_phpmyadmin, порт: 8080)
* **Ініціалізація БД** — автоматичне створення таблиць та тестових даних через infra/init.sql
* **Змінні середовища** — конфігураційні параметри у файлі .env

## 7.2. Конфігурація Docker Compose

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

- ./infra/init.sql:/docker-entrypoint-initdb.d/init.sql:ro

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:

* **Пояснення конфігурації**:
  + **Версія Docker Compose**: 3.x
  + **Сервіс db** використовує офіційний образ MySQL 8.0
  + **Автоматичний рестарт** контейнерів при збоях (restart: always)
  + **Збереження даних** у томі db\_data для персистентності
  + **Ініціалізація БД**: монтування ./infra/init.sql у /docker-entrypoint-initdb.d/ виконує скрипт при першому старті
  + **Залежність**: phpMyAdmin залежить від БД (depends\_on)
  + **Доступ**: до БД — порт 3306, до phpMyAdmin — порт 8080

## 7.3. Ініціалізація бази даних

Файл infra/init.sql містить SQL-скрипт для створення таблиць та заповнення тестовими даними:

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);

* **Структура БД**:
  + Таблиця users — користувачі системи
  + Таблиця films — фільми з основними атрибутами
  + Таблиця comments — коментарі до фільмів
  + Зовнішні ключі для підтримки цілісності даних

Примітка: для продуктивності рекомендується додати індекси з розділу 4.2.

## 7.4. Конфігурація змінних середовища

Файл .env містить налаштування для бекенд-сервера:

env

# 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

* **Призначення параметрів**:
  + PORT — порт, на якому працює Express-сервер
  + Параметри DB\_\* — конфігурація підключення до MySQL
  + Значення збігаються з налаштуваннями у docker-compose.yml

Порада: якщо бекенд запускається в контейнері поряд із БД, DB\_HOST має бути db (ім’я сервісу з Compose), а не localhost.

## 7.5. Процес розгортання

1. **Запуск інфраструктури БД**

bash

docker-compose up -d

1. **Запуск бекенд-сервера**

bash

cd backend

npm install

npm run dev

1. **Запуск фронтенд-додатка**

bash

cd frontend

npm install

ng serve

## 7.6. Перевірка працездатності

Після успішного запуску всіх компонентів:

1. **Перевірка API**
   * URL: http://localhost:3001/api/health
   * Очікувана відповідь:

json

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

1. **Доступ до phpMyAdmin**
   * URL: http://localhost:8080
   * Логін: root
   * Пароль: root
   * Переконайтесь, що створені таблиці та тестові дані доступні.
2. **Доступ до фронтенду**
   * URL: http://localhost:4200
   * Головна сторінка Angular-додатка.

## 7.7. Переваги архітектури

* **Швидкий старт**: одна команда запускає інфраструктуру БД
* **Уніфіковане середовище**: однакові умови на різних машинах
* **Простота адміністрування**: веб-інтерфейс для роботи з БД
* **Автоматична ініціалізація**: таблиці та тестові дані створюються автоматично
* **Збереження даних**: том Docker зберігає дані між запусками

## 7.8. Опційно: повна контейнеризація застосунку

Для повної контейнеризації всіх компонентів системи (бекенд + фронтенд + БД) можна розширити конфігурацію:

* **Dockerfile для бекенда** — створення образу Node.js-сервера
* **Dockerfile для фронтенда** — збірка Angular-додатка та сервінг через Nginx
* **Оновлений docker-compose.yml** — включення сервісів backend та frontend

Це дозволить запускати весь стек однією командою docker-compose up -d без необхідності локального встановлення Node.js чи Angular, що особливо зручно для демонстрації та тестування.

* **Переваги повної контейнеризації**:
  + Єдина команда для запуску всієї системи
  + Відсутність залежності від локального середовища
  + Ідентичні умови для розробки, тестування та демонстрації
  + Можливість швидкого розгортання на різних платформах

Детальні конфігурації для повної контейнеризації наведено у Додатках проєкту.

# 8. Безпека — чистова версія, готово до вставки

У цьому розділі зафіксовано поточний стан безпеки демо-реалізації, виявлені ризики, пріоритетні та додаткові заходи покращення, а також міграційний план впровадження.

## 8.1. Поточний стан (демо-версія)

Система реалізована з базовими заходами безпеки, достатніми для навчальної демонстрації:

* **Параметризовані SQL-запити** — захист від SQL-ін’єкцій завдяки плейсхолдерам у mysql2/promise.
* **Базовий CORS** — увімкнено міждоменні запити з контрольованими налаштуваннями.
* **Автентифікація (спрощена)** — проста перевірка користувача й повернення нешифрованого “токена” для умовної авторизації на фронтенді.

Обмеження цього підходу усвідомлені; вони прийнятні для демонстраційного прототипу, але не для продакшену.

## 8.2. Ідентифіковані ризики

1. **Зберігання паролів у відкритому вигляді** — паролі в таблиці users не хешуються.
2. **Відсутність суворої валідації** — обмежена перевірка вхідних даних на бекенді (формат, довжина, типи).
3. **Немає розмежування ролей (RBAC)** — усі користувачі мають однакові права доступу.
4. **Спрощена авторизація** — використовується базовий “токен” без терміну дії, підпису, механізму відкликання.
5. **Обмежені заголовки безпеки** — немає системного додавання безпекових заголовків (CSP, HSTS, X-Content-Type-Options).
6. **Потенційні XSS-ризики** — контент коментарів може відображатися без додаткової санітизації на фронтенді.
7. **Конфігурація середовищ** — значення на кшталт DB\_\* і CORS-домени не розділені на dev/prod; ризик витоків при неправильному деплої.

## 8.3. План покращень безпеки

### 8.3.1. Пріоритетні заходи

1. **Хешування паролів**
   * Використати bcrypt або argon2 для зберігання хешів паролів.
   * Міграція: згенерувати хеші для існуючих облікових записів або примусити ресет паролів.
2. **Валідація вхідних даних**
   * Додати схеми валідації через Joi/Zod на всіх API-ендпойнтах.
   * Норми: довжини полів, типи, whitelist жанрів, рік у діапазоні [1888;поточний рік][1888;поточний рік], URL-перевірка для poster\_url.
3. **Справжня токен-базована автентифікація**
   * Запровадити **JWT**: access (короткоживучий) + refresh (довгоживучий).
   * Зберігання: access в пам’яті застосунку; refresh у HttpOnly cookie з SameSite=strict.
4. **Санкціонований CORS**
   * Обмежити origin до переліку дозволених доменів (env-конфігурація).
   * Увімкнути credentials тільки за потреби.

### 8.3.2. Додаткові заходи

1. **Система ролей (RBAC)**
   * Ролі: user, admin (опційно moderator).
   * Middleware доступу до адміністративних ендпойнтів.
2. **Заголовки безпеки**
   * Підключити helmet з кастомним CSP:
     + default-src 'self'
     + img-src 'self' https: data:
     + script-src 'self' (або з контрольованими CDN із integrity)
     + frame-ancestors 'none'
   * Додатково: X-Content-Type-Options: nosniff, Referrer-Policy: no-referrer, Permissions-Policy.
3. **Захист від XSS**
   * На бекенді: санітизація текстових полів (коментарі).
   * На фронтенді: уникати innerHTML, використовувати Angular binding, додаткові пайпи-санітайзери за потреби.
4. **Журнали безпеки й аудит**
   * Логування входів/виходів, змін ролей, помилок авторизації.
   * Маскування чутливих даних у логах.
5. **Обмеження швидкості та анти-брутфорс**
   * rate-limit для /api/auth/login, капчі при підозрілій активності.
6. **Секрети і конфігурації**
   * Зберігати ключі JWT та паролі до БД поза репозиторієм (секрет-менеджери, .env у CI з шифруванням).
   * Розділити dev/stage/prod конфіги.

## 8.4. Міграційний план

1. **Схема БД**
   * Додати поле ролі:

sql

ALTER TABLE users

ADD COLUMN role ENUM('user','admin') NOT NULL DEFAULT 'user';

* + Додати індекси для аудиту/пошуку (за потреби).

1. **Хешування паролів**
   * Оновити реєстрацію: хешувати через bcrypt (saltRounds 10–12).
   * Оновити логін: порівняння bcrypt.compare.
   * Міграція існуючих: або форсований ресет, або поступове перехешування при першому логіні.
2. **JWT і авторизація**
   * Додати видачу access/refresh токенів, ротацію refresh.
   * Додати authMiddleware для перевірки підпису та витягання userId, role.
   * Оновити фронтенд: інтерсептор Authorization: Bearer <access>.
3. **Валідація даних**
   * Додати схеми Joi/Zod для всіх маршрутів (films, comments, auth).
   * Уніфікувати помилки:

json

{

"error": {

"code": "VALIDATION\_ERROR",

"message": "Invalid input",

"details": { "title": "required" }

}

}

1. **RBAC**
   * Додати middleware requireRole('admin') для адмін-операцій (наприклад, редагування/видалення будь-якого фільму, модерація коментарів).
2. **Безпекові заголовки й CORS**
   * Підключити helmet з CSP, налаштувати cors із whitelist.
   * Перевірити сумісність із фронтендом (шрифти/зображення/CDN).
3. **XSS-санітизація**
   * Додати бібліотеку санітизації на бекенді (наприклад, sanitize-html) для полів коментарів.
   * Перевірити рендеринг на фронтенді (без небезпечного HTML).
4. **Rate limiting**
   * Додати express-rate-limit до /api/auth/login, /api/comments (анти-спам).

## 8.5. Висновки

Поточна реалізація забезпечує базовий рівень безпеки, достатній для навчальної демонстраційної версії. Запропонований план покращень дозволяє без зміни загальної архітектури підвищити рівень захисту до промислових стандартів:

* паролі — лише у вигляді хешів,
* токени — JWT із ротацією й термінами дії,
* дані — валідуються централізовано,
* доступ — регулюється ролями,
* клієнт і сервер — захищені заголовками, CORS і санітизацією.

# 9. Тестування і результати — чистова версія, готово до вставки

У цьому розділі зафіксовано підхід до тестування, логічно структуровані тест-кейси для Postman з явними URL, послідовність прогонів, негативні сценарії та підсумкові результати. Всі приклади наведено для локального середовища: http://localhost:3001/api.

## 9.1. Підхід до тестування

* **Ручне тестування інтерфейсу**
  + Перегляд списку фільмів, фільтрація за жанром та роком, відкриття картки фільму, додавання коментаря.
* **Перевірка API через Postman (без cURL)**
  + Використовується колекція запитів, згрупована за ресурсами: Auth, Films, Comments.
  + У кожному запиті вказано явний URL.
* **Набір випробувань (jeu d’essai)**
  + Позитивні та негативні кейси з очікуваними HTTP-статусами, структурою JSON та побічними ефектами (запис/оновлення в БД).

## 9.2. Підготовка Postman

* **Створіть колекцію**: WhatToWatch API
  + Папки: Auth, Films, Comments.
* **Базовий шлях API**: http://localhost:3001/api.
* **Авторизація (демо-версія)**:
  + Перевірка виконується за наявністю заголовка Authorization (будь-яке значення).
  + У прикладах використано simple-auth-token.

Примітка: у поточній демо-реалізації не потрібний префікс Bearer. Після покращень безпеки (розділ 8) формат зміниться на Authorization: Bearer <JWT>.

## 9.3. Тест-кейси в Postman (із явними URL)

### 1) Auth

#### 1.1. Логін (позитивний)

* **Мета**: отримати демо-токен і перевірити успішний вхід.
* **Запит**: POST http://localhost:3001/api/auth/login
* **Headers**: Content-Type: application/json
* **Body**:

json

{ "email": "demo@example.com", "password": "demo\_pass" }

* **Очікування**:
  + Статус: 200 OK
  + JSON: має поля id, username, email, token
* **Tests (Postman)**:

js

pm.test("Status 200", () => pm.response.to.have.status(200));

const data = pm.response.json();

pm.expect(data).to.have.property("token");

pm.environment.set("accessToken", data.token);

#### 1.2. Логін (негативний: невірні дані)

* **Запит**: POST http://localhost:3001/api/auth/login
* **Body**:

json

{ "email": "demo@example.com", "password": "wrong\_pass" }

* **Очікування**: 401 Unauthorized
* **Tests**:

js

pm.test("Status 401", () => pm.response.to.have.status(401));

#### 1.3. Перевірка авторизації (позитивний)

* **Запит**: GET http://localhost:3001/api/auth/check
* **Headers**: Authorization: {{accessToken}}
* **Очікування**:
  + Статус: 200 OK
  + JSON: { "authenticated": true }
* **Tests**:

js

pm.test("Status 200", () => pm.response.to.have.status(200));

pm.expect(pm.response.json().authenticated).to.be.true;

#### 1.4. Перевірка авторизації (негативний: без токена)

* **Запит**: GET http://localhost:3001/api/auth/check
* **Очікування**: 401 Unauthorized
* **Tests**:

js

pm.test("Status 401", () => pm.response.to.have.status(401));

### 2) Films

#### 2.1. Отримати список фільмів (без фільтра)

* **Мета**: базова перевірка списку.
* **Запит**: GET http://localhost:3001/api/films
* **Очікування**: 200 OK, тіло — масив об’єктів.
* **Tests**:

js

pm.test("Status 200", () => pm.response.to.have.status(200));

pm.expect(Array.isArray(pm.response.json())).to.be.true;

#### 2.2. Фільтрація за жанром

* **Мета**: перевірити роботу фільтра за жанром.
* **Запит**: GET http://localhost:3001/api/films?genre=Comédie
* **Очікування**: 200 OK, кожен елемент має genre = "Comédie".
* **Tests**:

js

pm.test("Status 200", () => pm.response.to.have.status(200));

const data = pm.response.json();

pm.expect(Array.isArray(data)).to.be.true;

data.forEach(f => pm.expect(f.genre).to.eql("Comédie"));

#### 2.3. Деталі фільму (існуючий)

* **Запит**: GET http://localhost:3001/api/films/1
* **Очікування**: 200 OK (або 404 залежно від даних), наявність полів id, title, release\_year, genre, description, poster\_url.
* **Tests**:

js

pm.test("Status 200", () => pm.response.to.have.status(200));

const film = pm.response.json();

["id","title","release\_year","genre","description"].forEach(k => pm.expect(film).to.have.property(k));

#### 2.4. Деталі фільму (негативний: неіснуючий id)

* **Запит**: GET http://localhost:3001/api/films/999999
* **Очікування**: 404 Not Found
* **Tests**:

js

pm.test("Status 404", () => pm.response.to.have.status(404));

#### 2.5. Створення фільму (позитивний)

* **Запит**: POST http://localhost:3001/api/films
* **Headers**: Content-Type: application/json
* **Body**:

json

{

"title": "Interstellar",

"release\_year": 2014,

"genre": "Science-fiction",

"description": "Exploration of space and time.",

"poster\_url": "https://via.placeholder.com/300x450?text=Interstellar"

}

* **Очікування**: 201 Created, об’єкт із новим id.
* **Tests**:

js

pm.test("Status 201", () => pm.response.to.have.status(201));

const film = pm.response.json();

pm.expect(film).to.have.property("id");

pm.environment.set("createdFilmId", film.id);

#### 2.6. Створення фільму (негативний: невалідні дані)

* **Запит**: POST http://localhost:3001/api/films
* **Body**:

json

{}

* **Очікування**: 400 Bad Request
* **Tests**:

js

pm.test("Status 400", () => pm.response.to.have.status(400));

#### 2.7. Оновлення фільму (позитивний)

* **Попередня умова**: наявний createdFilmId з п. 2.5.
* **Запит**: PUT http://localhost:3001/api/films/{{createdFilmId}}
* **Headers**: Content-Type: application/json
* **Body**:

json

{

"title": "Interstellar (Updated)",

"release\_year": 2014,

"genre": "Science-fiction",

"description": "Updated description.",

"poster\_url": ""

}

* **Очікування**: 200 OK, поля оновлено.
* **Tests**:

js

pm.test("Status 200", () => pm.response.to.have.status(200));

const film = pm.response.json();

pm.expect(film.title).to.eql("Interstellar (Updated)");

pm.expect(film.description).to.eql("Updated description.");

### 3) Comments

#### 3.1. Перегляд коментарів до фільму

* **Запит**: GET http://localhost:3001/api/films/{{createdFilmId}}/comments (або 1)
* **Очікування**: 200 OK, масив із полями id, film\_id, user\_id, content, username.
* **Tests**:

js

pm.test("Status 200", () => pm.response.to.have.status(200));

const list = pm.response.json();

pm.expect(Array.isArray(list)).to.be.true;

if (list.length) {

["id","film\_id","user\_id","content","username"].forEach(k => pm.expect(list[0]).to.have.property(k));

}

#### 3.2. Додавання коментаря (позитивний)

* **Запит**: POST http://localhost:3001/api/films/{{createdFilmId}}/comments
* **Headers**:
  + Content-Type: application/json
  + Authorization: simple-auth-token
* **Body**:

json

{ "content": "Great movie!" }

* **Очікування**: 201 Created, об’єкт коментаря з username.
* **Tests**:

js

pm.test("Status 201", () => pm.response.to.have.status(201));

const c = pm.response.json();

["id","film\_id","user\_id","content","username"].forEach(k => pm.expect(c).to.have.property(k));

#### 3.3. Додавання коментаря (негативний: порожній контент)

* **Запит**: POST http://localhost:3001/api/films/{{createdFilmId}}/comments
* **Body**:

json

{ "content": "" }

* **Очікування**: 400 Bad Request
* **Tests**:

js

pm.test("Status 400", () => pm.response.to.have.status(400));

## 9.4. Послідовність прогону в Postman

1. Auth → Логін (позитивний) — збережіть accessToken у Postman Environment.
2. Films → Створення фільму — збережіть createdFilmId.
3. Films → Оновлення фільму — використайте createdFilmId.
4. Films → Список / Фільтрація / Деталі — перевірте відображення та структуру.
5. Comments → Додавання коментаря — використайте createdFilmId.
6. Comments → Перегляд коментарів — переконайтесь, що коментар з’явився.
7. Негативні кейси — пройдіть перевірки з помилковими вхідними даними чи умовами.

## 9.5. Результати

* **Базові сценарії працюють стабільно**:
  + Отримання списку і фільтрація фільмів.
  + Отримання детальної інформації про фільм.
  + Створення та оновлення фільмів із відображенням змін.
  + Додавання коментарів із відображенням username.
  + Авторизація (демо) й перевірка доступу за наявності заголовка Authorization.
* **Обробка помилок коректна**:
  + Повертаються відповідні HTTP-коди (4xx/5xx) з інформативними повідомленнями.
* **Реакція інтерфейсу**:
  + Форми відображають стани успіху/помилки відповідно до відповідей API.

Скріншоти запитів у Postman та приклади JSON-відповідей — у Додатках.

## 9.6. Обмеження демо-версії та нотатки

* **Авторизація спрощена**: мідлвар перевіряє лише наявність заголовка Authorization. Це прийнятно для демонстраційних цілей, але не для продакшну.
* **Тестові дані** ініціалізуються через infra/init.sql (користувач demo@example.com з паролем demo\_pass).
* **Повторні прогони**: для чистих повторних прогонів дозволено перезапускати БД у Docker та повторно застосовувати init.sql.

## 9.7. Рекомендації на майбутнє (щодо тестування)

* **Postman Runner + Newman**: автоматизація прогонів колекції в CI.
* **Автотести API**: Jest + supertest для контролю регресій.
* **E2E-тести UI**: Cypress після стабілізації авторизації.
* **Окреме тестове середовище БД**: із seed-скриптами для відтворюваних даних.

# 10. Оглядова аналітика (Veille) — чистова версія, готово до вставки

У процесі розробки дипломного проєкту “WhatToWatch” було проведено цілеспрямоване вивчення сучасних технологій і найкращих практик фронтенд-/бекенд-розробки, безпеки та інфраструктури. Нижче подано підсумок опанованих матеріалів і того, як вони були застосовані на практиці.

## 10.1. Технологічний стек дослідження

### Frontend технології

* **Angular документація** — глибоке вивчення:
  + Робота з формами: Template-driven і Reactive Forms, валідація, кастомні валідатори.
  + HttpClient: перехоплювачі, обробка помилок, withFetch().
  + Маршрутизація: Standalone-підхід, loadComponent для lazy-loading, guard-и.
  + Управління станом: локальні стани компонентів, сервіси як джерела даних, підхід до state co-location.
* **Сучасні підходи до UI/UX** — принципи інтуїтивного інтерфейсу:
  + Простота навігації та помітні CTA, підказки й порожні стани.
  + Адаптивність через Bootstrap utility-класи, читабельна типографіка.

### Backend технології

* **Express Best Practices** — структура застосунку, middleware pipeline, централізоване логування й обробка помилок, параметризовані SQL-запити, separation of concerns у роутерах.
* **REST API design** — зрозумілі ендпойнти, однозначні статус-коди, стабільні контракти JSON-відповідей, ідіоматичні методи HTTP.

### Бази даних

* **MySQL (глибинне вивчення)** — зовнішні ключі, індекси, оптимальні типи даних (INT, VARCHAR, TEXT), транзакції за потреби.
* **Моделювання реляційних БД** — нормалізація, зв’язки між таблицями users, films, comments.

### Безпека застосунків

* **OWASP Top 10** — детальне вивчення ризиків і контрзаходів:
  + Ін’єкції (SQL) — плейсхолдери, параметризовані запити.
  + Несправна автентифікація/авторизація — сесії, токени, надійні зберігання секретів.
  + XSS — санітизація вмісту, CSP, уникнення небезпечних API рендерингу.
  + CSRF — SameSite cookies, токени, коректний CORS.

### Інфраструктура та DevOps

* **Docker кращі практики** — мінімальні базові образи, кешування шарів, зовнішні томи для БД.
* **Контейнеризація БД** — швидке розгортання MySQL і phpMyAdmin, ініціалізація через скрипти.

## 10.2. Практичне застосування отриманих знань

### Підвищення безпеки

* **Параметризовані SQL-запити** — повне усунення SQL-ін’єкцій за рахунок плейсхолдерів у mysql2/promise.
* **Чіткі ендпойнти та статус-коди** — краще тестування в Postman, зручніша підтримка.

### Архітектурні рішення

* **Структура Express-застосунку** — розділення на routes, middleware, config для прозорості й масштабування.
* **Оптимізація БД** — цілісність через FK, підготовка до індексації полів пошуку/фільтрації (genre, release\_year).

### План майбутнього вдосконалення

* **Перехід на JWT/bcrypt** — дорожня карта: хешування паролів, access/refresh токени, ротація, cookie HttpOnly.
* **RBAC** — розмежування прав для користувачів/адміністраторів, middleware рівня доступу.

## 10.3. Інфраструктурні переваги

* **Стандартизоване Docker-оточення** — швидке розгортання MySQL і phpMyAdmin, із ініціалізацією даних.
* **Уніфікована розробка** — однакові умови для всіх учасників команди.
* **Легка масштабованість** — підготовка до перенесення у хмарні середовища, можливість додати контейнеризацію фронтенда/бекенда.

## 10.4. Висновки та перспективи

Аналітика дозволила створити функціональний демонстраційний проєкт і закласти фундамент для розвитку:

1. **Безпека** — базові контрзаходи вже реалізовано, є чіткий план переходу до промислових практик.
2. **Масштабованість** — архітектура REST і розділення відповідальностей полегшують додавання функціоналу.
3. **Підтримка** — зрозуміла структура коду, уніфіковані контракти API, узгоджені повідомлення про помилки.
4. **Продуктивність** — оптимізовані запити та легка індексація полів фільтрації забезпечують швидкий доступ до даних.

Отримані знання й досвід з сучасних технологій веб-розробки є вагомим внеском у професійне зростання та слугують основою для складніших проєктів.

# Висновки

## Досягнення

Проєкт “WhatToWatch” пройшов повний цикл створення сучасного веб-застосунку:

* **Повний стек**: інфраструктура → база даних → сервер → клієнт.
* **Базовий функціонал**: CRUD для фільмів, система коментарів, базова авторизація.
* **Зручне розгортання**: локальний запуск через Docker-контейнери.

## Складнощі та виклики

* **Узгодження даних** між фронтендом і бекендом — вимагало уніфікації схем і контрактів.
* **Валідація та обробка помилок** — вироблення чітких повідомлень для користувача та для логування.
* **Баланс безпеки та простоти** — вибір компромісів для MVP без надмірного ускладнення.

## Подальші кроки розвитку

1. **Безпека**: JWT-авторизація, bcrypt-хешування, ролі користувачів (RBAC), helmet, CSP.
2. **Робота з файлами**: завантаження/зберігання постерів (локальне сховище або S3).
3. **Тестування**: юніт- і інтеграційні тести, CI/CD пайплайни з Newman/Cypress.
4. **Обробка даних**: міграції БД, seed-скрипти, покриття крайніх випадків.
5. **Інтернаціоналізація**: i18n, покращення доступності (a11y), локалізація UI.

Проєкт демонструє міцну платформу для подальшого розвитку повноцінного кінопорталу та є відправною точкою для розширення компетенцій у веб-інженерії.

## Список використаних джерел

1. **Angular Documentation** — офіційна документація фреймворку Angular  
   <https://angular.io/docs>
2. **Express.js** — документація веб-фреймворку для Node.js  
   <https://expressjs.com/>
3. **MySQL Reference Manual** — офіційна документація СКБД MySQL  
   <https://dev.mysql.com/doc/>
4. **OWASP Top 10** — довідник з найбільш критичних ризиків безпеки веб-додатків  
   <https://owasp.org/www-project-top-ten/>
5. **Docker Documentation** — офіційна документація платформи Docker  
   <https://docs.docker.com/>