# Дипломна робота: What to Watch?

* Автор: SHAPOVAL Ruslan
* Керівник: [ПІБ керівника]
* Заклад: AFPA
* Дата: 2025

## Анотація

Мета роботи — розробити веб-застосунок “What to Watch?” для каталогізації фільмів з можливістю перегляду, фільтрації, додавання/редагування записів та коментування. Застосунок побудовано за клієнт-серверною архітектурою: фронтенд на Angular, бекенд на Node.js + Express, база даних MySQL, інфраструктура з Docker. Реалізовано REST API, базову авторизацію та CRUD-операції. Проєкт демонструє компетенції з проєктування інтерфейсу, роботи з базою даних, створення API, організації інфраструктури і базових аспектів безпеки.

Ключові слова: Angular, Node.js, Express, REST API, MySQL, Docker, аутентифікація, коментарі, фільми.

## 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

Тема — створення простого та зручного каталогу фільмів для особистого використання або невеликих кіноклубів. Користувачі можуть переглядати фільми, фільтрувати їх за жанром і роком, бачити деталі та коментувати. Авторизовані користувачі можуть додавати та редагувати фільми.

Мотивація: хочу навчитися будувати повний веб-застосунок від бази даних до інтерфейсу користувача, зрозуміти, як працює REST API, як організувати сервер і як упакувати все в просте розгортання через Docker.

Цілі:

* Навчальні: практика Angular, Express, MySQL та Docker; розуміння CRUD, форм, роутингу, взаємодії з API, базової безпеки.
* Технічні: реалізувати робочий прототип з авторизацією, фільтрами, коментарями, і документацією.

Результат: повністю працюючий CRUD-застосунок із базовою авторизацією та коментарями, інфраструктурою для локального запуску, скриптами БД і прикладами запитів.

## 1. Опис проєкту й контекст

Проєкт “What to Watch?” — це веб-додаток, що дозволяє:

* Переглядати список фільмів
* Фільтрувати за жанром та роком
* Переглядати деталі фільму (опис, рік, жанр, постер)
* Додавати коментарі до фільмів
* Реєструватися та входити (спрощено)
* Додавати/редагувати фільми (для авторизованих)

Цільова аудиторія: кіномани, невеликі клуби, навчальні проєкти.

Межі проєкту:

* Без реалізації рекомендаційних алгоритмів та складних ролі користувачів.
* Авторизація — базова (демо) без JWT та хешування паролів (заплановано до покращення).
* Жанри — статичний список ендпойнту.

Контекст навчання: проєкт створено для демонстрації компетенцій DWWM (CP2–CP7) — з акцентом на простоту, ясність і робочий прототип.

## 2. Аналіз предметної області та вимог

### 2.1. Бізнес-вимоги

* Каталог фільмів зі швидким переглядом.
* Фільтрація за жанром та роком випуску.
* Детальна сторінка фільму.
* Коментарі до фільмів.
* Реєстрація та вхід користувача.
* Додавання та редагування фільмів авторизованими користувачами.

### 2.2. Нефункціональні вимоги

* Простий запуск інфраструктури (Docker).
* Прийнятна швидкодія для невеликої кількості записів.
* Базова безпека та підготовка до її розширення.
* Читабельна структура коду.
* Локалізація інтерфейсу: англійська/українська (можна розширити).

### 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

Обрані технології:

* Angular — потужний фреймворк для SPA з роутингом, формами, сервісами та HttpClient.
* Node.js + Express — легкий у налаштуванні сервер для REST API.
* MySQL — реляційна база даних для структурованих таблиць та зв’язків.
* Docker + phpMyAdmin — швидкий запуск БД та зручне керування.

Обґрунтування:

* Angular добре підходить для багатосторінкових інтерфейсів і має багато готових рішень.
* Express дозволяє швидко оголосити маршрути та логіку, а екосистема велика.
* MySQL — знайома реляційна БД, зручна через SQL та інтеграцію з інструментами.
* Docker спрощує локальний запуск і однакове середовище для всіх.

Альтернативи (коротко):

* React або Vue замість Angular; NestJS замість Express; PostgreSQL замість MySQL. Обрано поточний стек через простоту та швидкість навчання.

## 4. Проєктування системи

### 4.1. Архітектура

Архітектура клієнт-сервер:

* Клієнт (Angular) відображає інтерфейс, надсилає запити до REST API.
* Сервер (Express) обробляє запити, виконує логіку, звертається до БД.
* База даних (MySQL) зберігає користувачів, фільми, коментарі.

Основні принципи:

* Розділення відповідальностей (UI ↔ API ↔ Data).
* Використання HTTP-методів для CRUD: GET, POST, PUT.
* Простота й прозорість структур і відповідей.

### 4.2. ER-діаграма БД

Сутності:

* users(id, username, email, password, created\_at)
* films(id, title, release\_year, genre, description, poster\_url, added\_by, created\_at)
* comments(id, film\_id, user\_id, content, created\_at)

Зв’язки:

* films.added\_by → users.id
* comments.film\_id → films.id
* comments.user\_id → users.id

Примітки:

* Мінімальна нормалізація: окремі таблиці для фільмів, користувачів, коментарів.
* Жанри зберігаються як текст у films.genre (просто для MVP).

[Місце для ER-діаграми у Додатках]

### 4.3. Модель навігації (Front-end)

Маршрути:

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

Компоненти:

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

### 4.4. Діаграма послідовності (приклади)

Логін:

* Користувач заповнює форму → POST /api/auth/login → сервер перевіряє → повертає простий токен → фронт зберігає токен (в пам’яті) → доступ до захищених дій (умовно).

Додавання фільму:

* Користувач заповнює форму → POST /api/films → перевірки на сервері → запис у БД → повернення створеного запису → фронт переходить на деталі.

## 5. Реалізація Front-end

### 5.1. Стек і структура

Стек:

* Angular, TypeScript, HTML, CSS (Bootstrap класами).

Структура:

* Сервіси: FilmService, AuthService — інкапсулюють виклики API.
* Компоненти: списки, деталі, форми, сторінки автентифікації.
* Маршрутизація: оголошення шляхів, зчитування параметрів через ActivatedRoute.

### 5.2. Компоненти

* FilmListComponent: відображає картки фільмів, фільтр за жанром/роком, звертається до GET /api/films?genre=&year=.
* FilmDetailComponent: показує дані фільму, підвантажує коментарі GET /api/films/:id/comments.
* FilmFormComponent: форма для створення/редагування; валідує обов’язкові поля; надсилає POST/PUT запити.
* LoginComponent, RegisterComponent: форми для входу і реєстрації; використовують AuthService.
* NavbarComponent: посилання на основні сторінки, кнопки входу/виходу.

### 5.3. Сервіси

* FilmService:
  + getFilms({genre, year})
  + getFilm(id)
  + createFilm(data)
  + updateFilm(id, data)
  + getComments(filmId)
  + addComment(filmId, content)
* AuthService:
  + login(email, password)
  + register(username, email, password)
  + check() — перевірка токена (умовно, через заголовок)
  + збереження простого токена в пам’яті

### 5.4. Фронтенд: деталі реалізації

* Двосторонній зв’язок [(ngModel)] для форм.
* HttpClient для викликів API, підписка на Observable.
* Стани: завантаження, успіх, помилка (просте відображення повідомлень).
* Bootstrap-сітка для responsive.

[Місце для скріншотів інтерфейсу: список, деталі, форма, логін/реєстрація — у Додатках]

## 6. Реалізація Back-end

### 6.1. Сервер і налаштування

* backend/server.js: ініціалізує Express, підключає cors і express.json.
* Ендпойнти для жанрів і health-check (/api/genres, /api/health).
* Підключення роутерів для фільмів і авторизації.

### 6.2. Підключення до БД

* backend/config/db.js: пул з’єднань через mysql2/promise.
* testConnection() перевіряє підключення при старті.
* Використання параметризованих запитів ? для безпеки.

### 6.3. Маршрути (огляд)

Фільми (backend/routes/films.js):

* GET /api/films — список, підтримка ?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 приєднаними іменами користувачів.
* POST /api/films/:id/comments — додати коментар (у демо прив’язано до userId=1 або поточного).

Авторизація (backend/routes/auth.js):

* POST /api/auth/login — перевіряє пару email/password у БД, повертає простий токен.
* POST /api/auth/register — створює користувача, перевіряє унікальність email/username.
* GET /api/auth/check — перевіряє наявність заголовка Authorization.

### 6.4. Валідація та помилки

* На вході перевіряються обов’язкові поля (наприклад, title, release\_year, genre, description).
* Повертаються відповідні коди статусу: 200/201 при успіху, 400 для поганого запиту, 401 для неавторизованого, 404 якщо даних немає, 500 при помилці сервера.
* Повідомлення — прості та зрозумілі.

## 7. Інфраструктура та розгортання

Інфраструктура локального запуску через Docker:

* docker-compose.yml піднімає:
  + MySQL 8 (порт 3306)
  + phpMyAdmin (порт 8080) для керування БД
* Ініціалізація БД:
  + infra/init.sql створює таблиці users, films, comments, заповнює тестові дані.
* Змінні середовища — у .env для бекенда (порт, креденшіали до БД).
* Запуск:
  + БД: docker compose up -d
  + Бекенд: npm run dev у папці backend/
  + Фронтенд: ng serve у папці frontend/
* Перевірка:
  + http://localhost:3001/api/health
  + http://localhost:8080 (phpMyAdmin)

Переваги:

* Швидкий підйом оточення.
* Однакові умови запуску на різних машинах.

## 8. Безпека

Поточний стан (демо):

* Параметризовані SQL-запити.
* Базовий CORS.
* Спрощена авторизація без хешування паролів і без JWT.

Ризики:

* Зберігання паролів у відкритому вигляді.
* Відсутність суворої валідації на бекенді.
* Немає розмежування ролей (адмін/користувач).

План покращень:

* Використати bcrypt для хешування паролів.
* Додати JWT з терміном дії й refresh-токенами, httpOnly cookies.
* Валідувати вхідні дані (схеми валідації, наприклад Joi).
* Обмежити CORS, додати заголовки безпеки (CSP).
* Додати ролі (RBAC) для операцій додавання/редагування/видалення.
* Анти-XSS і CSRF-заходи.

## 9. Тестування і результати

Підхід:

* Ручне тестування через інтерфейс.
* Перевірка API через cURL/Postman.
* Набір випробувань (jeu d’essai) для ключових сценаріїв.

Приклади тестів:

* Фільтрація: GET /api/films?genre=Comédie → очікуємо лише комедії.
* Деталі фільму: GET /api/films/1 → перевіряємо поля.
* Створення фільму: POST /api/films з валідними даними → 201 і запис у БД.
* Оновлення фільму: PUT /api/films/:id → 200 і оновлені поля.
* Коментар: POST /api/films/:id/comments → 201 і коментар з username.
* Авторизація: POST /api/auth/login → отримання токена; GET /api/auth/check з заголовком Authorization.

Результати:

* Базові сценарії працюють стабільно.
* Помилки обробляються з поверненням коду і повідомлення.
* Інтерфейс реагує на успіх/помилки форм.

[Скріншоти і JSON-відповіді — у Додатках]

## 10. Оглядова аналітика (Veille)

Що вивчалося:

* Документація Angular (форми, HttpClient, маршрути).
* Express Best Practices (структура застосунку, обробка помилок).
* MySQL (зовнішні ключі, індекси, типи даних).
* OWASP Top 10 (ін’єкції, автентифікація, XSS, CSRF).
* Docker (кращі практики для локальної розробки).

Як допомогло:

* Параметризовані запити підвищили безпеку.
* Чіткі ендпойнти і статус-коди — краща підтримка і тестування.
* План переходу на JWT/bcrypt — дорожня карта до продакшну.
* Стандарти Docker дозволили швидко розгорнути БД і інструменти.

## Висновки

Досягнення:

* Реалізовано повний цикл: інфраструктура → база даних → сервер → інтерфейс.
* Працює CRUD для фільмів, коментарі, базова авторизація.
* Зручний локальний запуск через Docker.

Складнощі:

* Узгодження формату даних між фронтом і беком.
* Організація валідації і повідомлень про помилки.
* Планування безпеки без ускладнення MVP.

Подальші кроки:

* Повна авторизація з JWT, хешування паролів, ролі.
* Завантаження постерів (збереження файлів або S3).
* Юніт- та інтеграційні тести, CI/CD.
* Міграції БД, seed-скрипти, покриття edge-cases.
* I18n (міжнародна локалізація) та покращення доступності (a11y).

## Список джерел

1. Angular Documentation: <https://angular.io/docs>
2. Express.js: <https://expressjs.com/>
3. MySQL Reference Manual: <https://dev.mysql.com/doc/>
4. OWASP Top 10: <https://owasp.org/www-project-top-ten/>
5. Docker Documentation: <https://docs.docker.com/>

## Додатки

Нижче — матеріали, які можна одразу вставити у документ або роздрукувати окремо.

### Додаток A. ER-діаграма (текстовий опис)

* Таблиця users:
  + id INT PK AUTO\_INCREMENT
  + username VARCHAR(100) UNIQUE NOT NULL
  + email VARCHAR(255) UNIQUE NOT NULL
  + password VARCHAR(255) NOT NULL
  + created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP
* Таблиця films:
  + id INT PK AUTO\_INCREMENT
  + title VARCHAR(255) NOT NULL
  + release\_year INT NOT NULL
  + genre VARCHAR(100) NOT NULL
  + description TEXT NOT NULL
  + poster\_url VARCHAR(500) NULL
  + added\_by INT NOT NULL REFERENCES users(id)
  + created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP
* Таблиця comments:
  + id INT PK AUTO\_INCREMENT
  + film\_id INT NOT NULL REFERENCES films(id)
  + user\_id INT NOT NULL REFERENCES users(id)
  + content TEXT NOT NULL
  + created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

Зв’язки:  
users (1) → (N) films через films.added\_by  
users (1) → (N) comments через comments.user\_id  
films (1) → (N) comments через comments.film\_id

[Можна додати графічну ER-схему як зображення]

### Додаток B. Приклади HTTP-запитів (cURL) та очікувані відповіді

* Перевірка API:

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

Очікувано:

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

* Список фільмів:

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

* Фільтрація:

curl "http://localhost:3001/api/films?genre=Comédie&year=2020"

* Деталі фільму:

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

* Створення фільму:

curl -X POST "http://localhost:3001/api/films" \

-H "Content-Type: application/json" \

-d '{"title":"Inception","release\_year":2010,"genre":"Science-fiction","description":"A dream heist.","poster\_url":"https://..."}'

* Оновлення фільму:

curl -X PUT "http://localhost:3001/api/films/1" \

-H "Content-Type: application/json" \

-d '{"title":"Inception (Updated)","release\_year":2010,"genre":"Science-fiction","description":"Updated desc."}'

* Коментарі:

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

* Додати коментар:

curl -X POST "http://localhost:3001/api/films/1/comments" \

-H "Content-Type: application/json" \

-d '{"content":"Чудовий фільм!"}'

* Авторизація (login):

curl -X POST "http://localhost:3001/api/auth/login" \

-H "Content-Type: application/json" \

-d '{"email":"user@example.com","password":"password"}'

* Реєстрація:

curl -X POST "http://localhost:3001/api/auth/register" \

-H "Content-Type: application/json" \

-d '{"username":"newuser","email":"n@example.com","password":"secret"}'

### Додаток C. Витяги коду бекенду (ключові фрагменти)

* backend/server.js (скорочено):

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) => {

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);

});

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

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

app.listen(PORT, () => {

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

});

* backend/routes/auth.js (скорочено):

const express = require('express');

const router = express.Router();

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

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 (e) {

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

}

});

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 [existing] = await pool.query('SELECT \* FROM users WHERE email = ? OR username = ?', [email, username]);

if (existing.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 (e) {

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

}

});

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

module.exports = { router, isAuthenticated };

### Додаток D. SQL-скрипти (приклад ініціалізації)

CREATE TABLE users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(100) NOT NULL UNIQUE,

email VARCHAR(255) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE films (

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255) NOT NULL,

release\_year INT NOT NULL,

genre VARCHAR(100) NOT NULL,

description TEXT NOT NULL,

poster\_url VARCHAR(500),

added\_by INT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT fk\_films\_user FOREIGN KEY (added\_by) REFERENCES users(id)

);

CREATE TABLE comments (

id INT AUTO\_INCREMENT PRIMARY KEY,

film\_id INT NOT NULL,

user\_id INT NOT NULL,

content TEXT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

CONSTRAINT fk\_comments\_film FOREIGN KEY (film\_id) REFERENCES films(id),

CONSTRAINT fk\_comments\_user FOREIGN KEY (user\_id) REFERENCES users(id)

);

### Додаток E. Приклади фронтенд-коду (фрагменти)

* Сервіс фільмів (приклад):

@Injectable({ providedIn: 'root' })

export class FilmService {

private base = '/api/films';

constructor(private http: HttpClient) {}

getFilms(params?: { genre?: string; year?: number }) {

let p = new HttpParams();

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

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

return this.http.get<Film[]>(this.base, { params: p });

}

getFilm(id: number) { return this.http.get<Film>(`${this.base}/${id}`); }

createFilm(data: Partial<Film>) { return this.http.post<Film>(this.base, data); }

updateFilm(id: number, data: Partial<Film>) { return this.http.put<Film>(`${this.base}/${id}`, data); }

getComments(id: number) { return this.http.get<Comment[]>(`${this.base}/${id}/comments`); }

addComment(id: number, content: string) { return this.http.post(`${this.base}/${id}/comments`, { content }); }

}

### Додаток F. Скріншоти інтерфейсу

* Список фільмів — [вставити зображення]
* Деталі фільму — [вставити зображення]
* Форма фільму — [вставити зображення]
* Логін/реєстрація — [вставити зображення]

### Додаток G. Таблиця “Компетенції → Докази”

| **Компетенція** | **Доказ/місце у проєкті** |
| --- | --- |
| CP2 (макети/архітектура UI) | Маршрути, компоненти, скріншоти UI |
| CP3 (HTML/CSS) | Шаблони Angular, Bootstrap-верстка |
| CP4 (динаміка/JS) | Форми, HttpClient, обробка станів |
| CP5 (БД) | ER-схема, init.sql |
| CP6 (доступ до даних) | Пул mysql2/promise, параметризовані запити |
| CP7 (бек-логіка) | Роутери films, auth, коди статусів |
| CP8 (інфра/деплой — опц.) | Docker compose, phpMyAdmin |

Кінець документа