

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Веб-технології»)

на тему:

«Вебплатформа відеохостингу»

Виконав студент групи ВТ-22-1
БАБІЦЬКИЙ Віктор Юрійович

Керівник роботи:
ГРОМСЬКИЙ Олександр Олександрович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ

На випускню кваліфікаційну роботу бакалавра

Здобувач вищої освіти: БАБІЦЬКИЙ Віктор Юрійович

Керівник роботи: ГРОМСЬКИЙ Олександр Олександрович

Тема роботи: **«Вебплатформа відеохостингу»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92**

Вихідні дані для роботи: **функціональні вимоги до вебплатформи відеохостингу BenFube. Технології: Symfony, PostgreSQL, Next.js, хмарне сховище Cloudinary, автентифікація JWT та Google OAuth.**

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Громський О.О.	01.03.2026р.	01.03.2026р.
2	Громський О.О.	01.03.2026р.	01.03.2026р.
3	Громський О.О.	01.03.2026р.	01.03.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебплатформи відеохостингу «BenFube» та пояснювальної записки. Пояснювальна записка до випускної роботи містить 98 сторінку, 22 ілюстрації та 8 таблиць, 3 додатки обсягом 11 сторінок.

Метою роботи є проєктування та розробка повнофункціональної вебплатформи для завантаження, перегляду та управління відеоконтентом із підтримкою каналів, підписок, коментарів, системи монетизації через донати та адміністративної панелі модерації.

У роботі застосовано методи об'єктно-орієнтованого аналізу та проєктування, UML-моделювання, REST-архітектуру, клієнт-серверну взаємодію через JWT-автентифікацію. Реалізовано бекенд на PHP 8.2 з фреймворком Symfony 6 та PostgreSQL, фронтенд — на Next.js 14 з TypeScript. Платформа розгорнута з використанням Docker, Vercel та Render.

Отримані результати можуть бути використані як основа для побудови комерційних або освітніх відеохостингових рішень, а також як навчальний приклад реалізації повного циклу веброзробки.

КЛЮЧОВІ СЛОВА: ВІДЕОХОСТИНГ, ВЕБ-ПЛАТФОРМА, ВІДЕОКОНТЕНТ, REST API, ВЕБЗАСТОСУНОК.

					ІПЗ.КР.Б - 121 - 26 - ПЗ					
Змн.	Арк.	№ докум.	Підпис	Дата						
Розроб.		Бабіцький В.Ю.			Вебплатформа відеохостингу			Літ.	Арк.	Аркушів
Перевір.		Громський О.О.							4	98
								Житомирська політехніка, група ВТ-22-1		
Рецензент		Концидайло А.М.								
Зав. каф.		Вакалюк Т. А.								

ABSTRACT

The bachelor's qualification thesis consists of the «BenFube» video hosting web platform and an explanatory note. The explanatory note contains 98 pages, 22 illustrations and 8 tables, 3 appendices totaling 11 pages.

The aim of the work is to design and develop a fully functional web platform for uploading, viewing and managing video content with support for channels, subscriptions, comments, a monetization system through donations, and an administrative moderation panel.

The work applies methods of object-oriented analysis and design, UML modeling, REST architecture, and client-server interaction through JWT authentication. The backend is implemented in PHP 8.2 with the Symfony 6 framework and PostgreSQL; the frontend is built with Next.js 14 and TypeScript. The platform is deployed using Docker, Vercel, and Render.

The obtained results can be used as a basis for building commercial or educational video hosting solutions, as well as a practical example of implementing a complete web development lifecycle.

KEYWORDS: VIDEO HOSTING, WEB PLATFORM, VIDEO CONTENT, REST API, WEB APPLICATION.

					ИПЗ.КР.Б - 121 - 26 - ИПЗ	Апр.
						4

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс (Application Programming Interface)

БД – база даних

CDN – мережа доставки контенту (Content Delivery Network)

CORS – спільне використання ресурсів між різними джерелами (Cross-Origin Resource Sharing)

CRUD – основні операції над даними: Create, Read, Update, Delete

DTO – об'єкт передачі даних (Data Transfer Object)

HTML – мова розмітки гіпертексту (HyperText Markup Language)

HTTP – протокол передачі гіпертексту (HyperText Transfer Protocol)

HTTPS – захищений протокол передачі гіпертексту (HTTP Secure)

JWT – токен з відкритим ключем (JSON Web Token)

MVC – архітектурний шаблон Модель-Вигляд-Контролер (Model-View-Controller)

ORM – об'єктно-реляційне відображення (Object-Relational Mapping)

ПЗ – програмне забезпечення

REST – архітектурний стиль передачі стану представлення (Representational State Transfer)

SQL – мова структурованих запитів (Structured Query Language)

СУБД – система управління базами даних

UI – інтерфейс користувача (User Interface)

UML – уніфікована мова моделювання (Unified Modeling Language)

URL – уніфікований локатор ресурсів (Uniform Resource Locator)

CSS – каскадні таблиці стилів (Cascading Style Sheets)

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	5
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИМОГ ДО ВЕБПЛАТФОРМИ ВІДЕОКОНТЕНТУ	9
1.1. Постановка задачі	9
1.2. Аналіз аналогів програмного продукту	12
1.3. Архітектура системи та ролі користувачів	17
1.4. Обґрунтування вибору технологічного стеку	22
Висновки до розділу 1	28
РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ BENFUBE	30
2.1. Визначення вимог та варіантів використання системи	30
2.2. Проектування бази даних	36
2.3. Алгоритми роботи системи	42
2.4. Реалізація бекенду на основі Symfony та REST API	47
2.5. Реалізація фронтенду на основі Next.js та TypeScript	53
Висновки до розділу 2	57
РОЗДІЛ 3. РОЗГОРТАННЯ, ІНТЕРФЕЙС ТА ТЕСТУВАННЯ ПЛАТФОРМИ BENFUBE	59
3.1. Розгортання та налаштування системи	59
3.2. Опис інтерфейсу користувача	65
3.3. Тестування програмного комплексу	74
Висновки до розділу 3	79
ВИСНОВКИ	81
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	83
ДОДАТКИ	87
Додаток А. Лістинг класу AuthService	87
Додаток Б. Лістинг класу VideoService	90
Додаток В. Лістинги хуків та компонентів фронтенду	94

ВСТУП

Сучасний цифровий простір характеризується стрімким зростанням обсягів відеоконтенту. За даними аналітичних звітів Cisco, відеотрафік складав понад 80% усього інтернет-трафіку [1]. Платформи для перегляду та розміщення відео стали невід'ємною складовою повсякденного життя мільярдів людей у всьому світі — від розважального контенту до освітніх матеріалів та корпоративних трансляцій.

Незважаючи на домінування глобальних платформ, зокрема YouTube, Vimeo та Dailymotion, спостерігається стійкий попит на регіональні та спеціалізовані альтернативи. Причинами цього є обмеження у монетизації для авторів з певних регіонів, питання захисту авторських прав, мовні та культурні особливості аудиторії, а також прагнення до технологічної незалежності. Крім того, для навчальних закладів, медіакомпаній та стартапів актуальною є потреба у власних відеохостингових рішеннях із розширеними можливостями управління контентом та монетизації.

Таким чином, розробка повнофункціональної вебплатформи для розміщення, перегляду та управління відеоконтентом є актуальним науково-технічним завданням, що має практичну цінність як у комерційному, так і в освітньому контексті.

Метою випускної кваліфікаційної роботи бакалавра є проектування та реалізація вебплатформи «BenFube» — повнофункціонального відеохостингового сервісу з підтримкою каналів, підписок, системи лайків і коментарів, механізму монетизації через донати, адміністративної панелі модерації та алгоритму трендів.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. Провести аналіз предметної галузі та існуючих аналогів відеохостингових платформ.
2. Визначити функціональні та нефункціональні вимоги до системи.
3. Обрати та обґрунтувати технологічний стек для реалізації бекенду та фронтенду.

4. Спроектувати структуру бази даних та архітектуру програмного комплексу.

5. Реалізувати серверну частину на основі REST API з використанням PHP 8.2 та Symfony 6.

6. Розробити клієнтську частину на основі Next.js 14 та TypeScript [24].

7. Виконати розгортання платформи та провести функціональне тестування.

Об'єктом дослідження є процес проектування та розробки вебплатформи для розміщення й управління відеоконтентом в умовах сучасних вимог до продуктивності, безпеки та зручності використання.

Предметом дослідження є засоби та технології побудови відеохостингових платформ на базі REST-архітектури, серверних фреймворків і сучасних клієнтських технологій, зокрема методи організації автентифікації, зберігання медіафайлів у хмарних сховищах та побудови алгоритмів ранжування контенту.

Методи дослідження. У роботі застосовано методи системного аналізу та об'єктно-орієнтованого проектування, UML-моделювання [29] для побудови діаграм варіантів використання, послідовності та активності, методи реляційного проектування баз даних, а також принципи REST-архітектури [26] для побудови програмного інтерфейсу.

Практичне значення одержаних результатів полягає у тому, що розроблена платформа «VenFube» є повністю функціонуючим вебзастосунком, розгорнутим у хмарному середовищі, який може бути використаний як основа для комерційного відеохостингового рішення, освітнього порталу або корпоративної медіаплатформи. Реалізований програмний комплекс демонструє застосування сучасних підходів до веброзробки, контейнеризації та хмарного розгортання.

Структура та обсяг роботи. Випускна кваліфікаційна робота бакалавра складається з вступу, трьох розділів, висновків, списку використаних джерел із 35 найменувань та 3 додатків. Повний обсяг роботи становить 103 сторінки, 22 ілюстрації та 8 таблиць, 3 додатки обсягом 10 сторінок.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						8

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИМОГ ДО ВЕБПЛАТФОРМИ ВІДЕОКОНТЕНТУ

1.1. Постановка задачі

Відеохостинг — це один із найбільш складних та технологічно вимогливих видів сучасних вебсервісів, який надає користувачам можливість завантажувати, зберігати, перекодувати, транслювати та переглядати відеоконтент у режимі реального часу. Розвиток технологій передачі даних та широке впровадження швидкісного інтернету призвели до того, що відеоматеріали стали домінуючим типом трафіку в глобальній мережі. Створення ефективного відеохостингу потребує вирішення низки складних інженерних задач, пов'язаних із розподілом навантаження, обробкою великих обсягів мультимедійних даних, організацією швидкої доставки контенту до кінцевого користувача та забезпеченням високого рівня інтерактивності інтерфейсу. Платформа «BenFube», що проєктується, орієнтована на вирішення цих проблем шляхом створення оптимізованого вебзастосунку з чітким розподілом обов'язків між клієнтською та серверною частинами.

Сучасна відеохостингова платформа є багатокomпонентним програмним комплексом, який має об'єднувати декілька автономних, але тісно інтегрованих підсистем. Серверна частина повинна забезпечувати надійне збереження структурованих метаданих (інформації про користувачів, канали, відео, лайки, підписки, коментарі та фінансові транзакції), а також виконувати бізнес-логіку з високим рівнем безпеки та продуктивності. Клієнтська частина має забезпечувати реактивне та адаптивне відображення сторінок на різних пристроях, включаючи мобільні телефони та планшети. Крім того, критично важливими є підсистеми взаємодії з хмарними медіасховищами та системами доставки вмісту (CDN), оскільки зберігання та стрімінг гігабайтів відеофайлів безпосередньо з основного сервера застосунків призводить до швидкого вичерпання його ресурсів та значного уповільнення роботи системи.

Перед вебплатформою відеохостингу «BenFube» поставлено комплексне завдання — розробити та впровадити архітектурне та програмне рішення, яке

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						9

забезпечить повний функціональний цикл роботи з відеоконтентом для різних категорій користувачів. З точки зору бізнес-процесів, система має автоматизувати процеси реєстрації та автентифікації користувачів, створення та персоналізації індивідуальних каналів, завантаження відеофайлів з автоматичною генерацією обкладинок та кодуванням медіаданих, пошуку та фільтрації відеоматеріалів за ключовими словами й категоріями, а також надання інтерактивних засобів комунікації та фінансової підтримки авторів контенту.

Для реалізації цих завдань у структурі платформи «BenFube» виокремлюються наступні функціональні підсистеми. Перша — підсистема автентифікації та авторизації. Вона відповідає за безпечний вхід користувачів до системи за допомогою класичного поєднання логіна й пароля (з використанням хешування) або через сторонні сервіси (Google OAuth 2.0). Після успішного входу користувач отримує токен доступу (JWT), який використовується для авторизації всіх подальших запитів. Друга підсистема — управління профілями та каналами. Кожен зареєстрований користувач отримує можливість створити один або кілька каналів, для яких він може налаштовувати назву, опис, унікальну адресу (slug), завантажувати аватар та банер каналу. Це забезпечує ідентичність авторів та допомагає будувати лояльну аудиторію навколо їхнього контенту.

Третя підсистема — завантаження та обробки відеоматеріалів. Це найбільш вимоглива до обчислювальних ресурсів частина платформи. Користувач-автор має можливість завантажити відеофайл через веб-інтерфейс. Для оптимізації навантаження на сервер Symfony завантаження здійснюється безпосередньо у хмарне сховище Cloudinary. Підсистема генерує унікальний ідентифікатор медіаресурсу та зберігає посилання на нього в базі даних PostgreSQL. Четверта підсистема — відтворення та стрімінгу відео. Вона базується на інтерактивному кастомному HTML5-плеєрі на фронтенді, який підтримує керування клавіатурою, регулювання гучності, зміну якості потоку та запам'ятовування позиції перегляду, що забезпечує високий рівень комфорту для глядача під час перегляду.

П'ята підсистема — соціальної взаємодії та зворотного зв'язку. Вона включає можливість оцінювати відеозаписи за допомогою лайків та дизлайків, підписуватися на оновлення каналів та залишати коментарі. Підсистема коментарів підтримує деревоподібну структуру відповідей, дозволяючи вести дискусії під відеоматеріалами. Шоста підсистема — фінансової підтримки (донати). Вона автоматизує процес підтримки авторів шляхом надсилання грошових переказів. Користувач може ініціювати платіж через інтерактивне модальне вікно, вказавши суму внеску та текстове повідомлення, яке відображається автору каналу. Всі транзакції записуються в базу даних для подальшого обліку та відображення статистики в кабінеті автора.

Сьома підсистема — модерації, скарг та адміністрування. Для підтримання безпечного та легального контенту на платформі розроблено механізм подання скарг (reports) на відеоматеріали, які порушують правила спільноти. Скарги надходять до адміністративної панелі, де адміністратор має можливість переглянути деталі скарги, заблокувати відео (змінити його видимість) або відхилити скаргу. Для захисту авторів передбачена система подання апеляцій у разі помилкового блокування. Восьма підсистема — аналітики та рекомендацій. Вона збирає статистику переглядів відео та підписок на канали, візуалізуючи її у вигляді інтерактивних графіків у кабінеті автора (Studio). На основі зібраних даних про активність глядачів працює алгоритм визначення трендових відео (Trending), який формує динамічний список найбільш популярного контенту за останні 48 годин з урахуванням логарифмічного затухання часу публікації.

Вимоги до розроблюваної системи також охоплюють нефункціональні критерії, такі як безпека, продуктивність та масштабованість. Бекенд-API має працювати у безсесійному режимі (stateless) для забезпечення можливості горизонтального масштабування шляхом запуску нових екземплярів контейнерів за балансувальником навантаження. База даних PostgreSQL повинна мати оптимізовану структуру з належним використанням індексів для прискорення пошуку відео та перевірки унікальності користувачів. Передача всіх даних має шифруватися за протоколом HTTPS з використанням сучасних SSL/TLS

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						11

сертифікатів. Клієнтська частина Next.js має бути повністю оптимізована для швидкого завантаження на мобільних пристроях та адаптована для пошукової індексації (SEO), що є критично важливим для успішного просування відеохостингу на ринку. Очікуваним результатом проєкту є створення повноцінної, безпечної та високоефективної вебплатформи, готової до використання в реальних умовах продакшн-середовища.

1.2. Аналіз аналогів програмного продукту

Перед початком безпосереднього проєктування та розробки власного програмного рішення було проведено детальний порівняльний аналіз наявних на ринку платформ відеохостингу. Головною метою аналізу є дослідження їхніх функціональних можливостей, архітектурних особливостей, бізнес-моделей монетизації та виявлення ключових технологічних обмежень і недоліків. Це дозволить сформулювати обґрунтовані вимоги до створюваної платформи «BenFube», врахувати успішний досвід лідерів індустрії та уникнути типових помилок при побудові систем поширення відеоконтенту. Для аналізу було обрано чотири репрезентативні платформи, які представляють різні підходи до організації відеохостингу: комерційний гігант YouTube, орієнтований на професіоналів Vimeo, медійний сервіс Dailymotion та децентралізовану федеративну систему PeerTube.

YouTube, заснований у 2005 році і згодом придбаний корпорацією Google, є абсолютним світовим лідером у сфері відеохостингу. Платформа має колосальну глобальну інфраструктуру, яка обслуговує мільярди користувачів щодня. Технологічна база YouTube включає власну мережу доставки вмісту (Google Global Cache) та передові алгоритми машинного навчання для рекомендації відео. Платформа пропонує авторам контенту потужні інструменти аналітики в реальному часі, засоби для проведення прямих трансляцій, редагування відео безпосередньо в браузері та гнучкі інструменти взаємодії з глядачами, такі як спонсорство, суперчати та інтегрована рекламна мережа Google AdSense (Рис. 1.1).

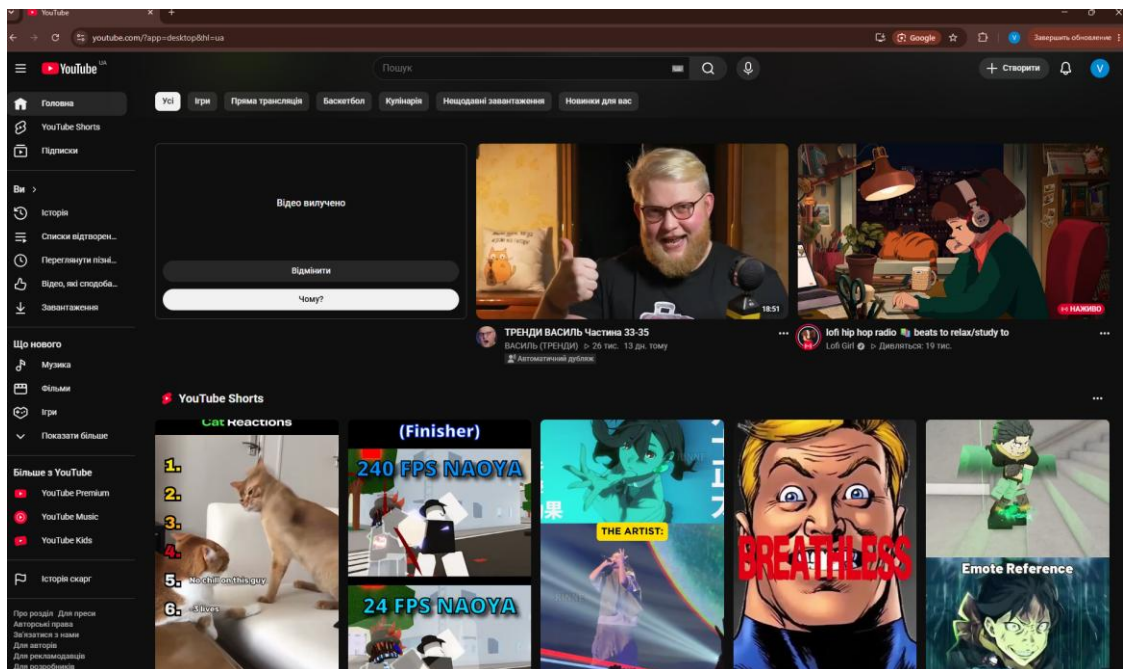


Рис. 1.1 – Інтерфейс відеохостингової платформи YouTube

Незважаючи на лідерство, YouTube має низку критичних обмежень, які ускладнюють старт для нових авторів та викликають невдоволення спільноти. По-перше, доступ до програми монетизації є закритим для початківців і вимагає досягнення жорстких лімітів: щонайменше 1000 підписників та 4000 годин перегляду відео за останній рік. По-друге, платформа використовує асиметричну та часто непрозору політику цензури й демонетизації відео за допомогою автоматизованих систем на основі штучного інтелекту, що призводить до помилкового блокування контенту або позбавлення авторів прибутків без можливості швидкої апеляції. По-третє, глядачі стикаються з надмірною кількістю нав'язливої реклами, відключення якої вимагає платної підписки YouTube Premium, що обмежує доступність сервісу.

Vimeo, створений у 2004 році, обрав альтернативну стратегію розвитку, позиціонуючи себе як інструмент для професійних відеографів, режисерів, дизайнерів та художників. Головною філософією Vimeo є забезпечення найвищої якості відтворення відео та відсутність реклами, яка б відволікала від перегляду творчих робіт. Платформа надає унікальні можливості для індивідуального налаштування дизайну відеоплеєра (зміна кольорів, додавання логотипу автора, приховування елементів керування), захисту відео паролем,

обмеження доступу за доменами та спільної роботи над проєктами в реальному часі (додавання коментарів до конкретних кадрів під час монтажу) (Рис. 1.2).

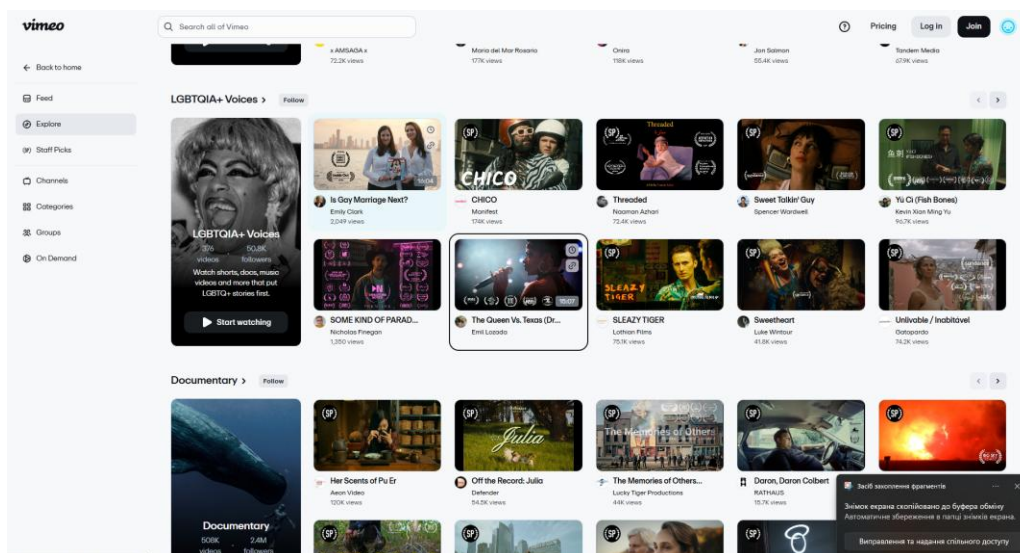


Рис. 1.2 – Інтерфейс відеохостингової платформи Vimeo

Основним недоліком Vimeo є платна модель доступу — безкоштовний тарифний план має суттєві обмеження на обсяг завантаження файлів (лише 500 МБ на тиждень та загальний ліміт у 5 ГБ), що робить його абсолютно непридатним для регулярного ведення блогів чи розміщення тривалих відеоматеріалів високої якості. Платні підписки є досить дорогими, а у разі перевищення лімітів трафіку платформа може виставити автору додатковий рахунок за стрімінг контенту. Окрім того, Vimeo має значно меншу глядацьку аудиторію порівняно з YouTube, що знижує шанси на органічне просування контенту для нових авторів.

Dailymotion — французька відеоплатформа, яка функціонує з 2005 року і є одним із найстаріших конкурентів YouTube у Європі. Платформа фокусується переважно на новинному, спортивному та розважальному контенті, тісно співпрацюючи з провідними світовими медіакомпаніями та видавництвами. Dailymotion пропонує партнерам широкі можливості для синдикації відео (вбудовування плеєра на сторонні сайти з розділенням прибутку від реклами) та кастомізації інтерфейсу (Рис. 1.3). Проте для незалежних авторів платформа має складну та застарілу систему монетизації, низький рівень залучення аудиторії та

перевантажений інтерфейс, який часто піддається критиці з боку користувачів за велику кількість реклами та незручну навігацію.

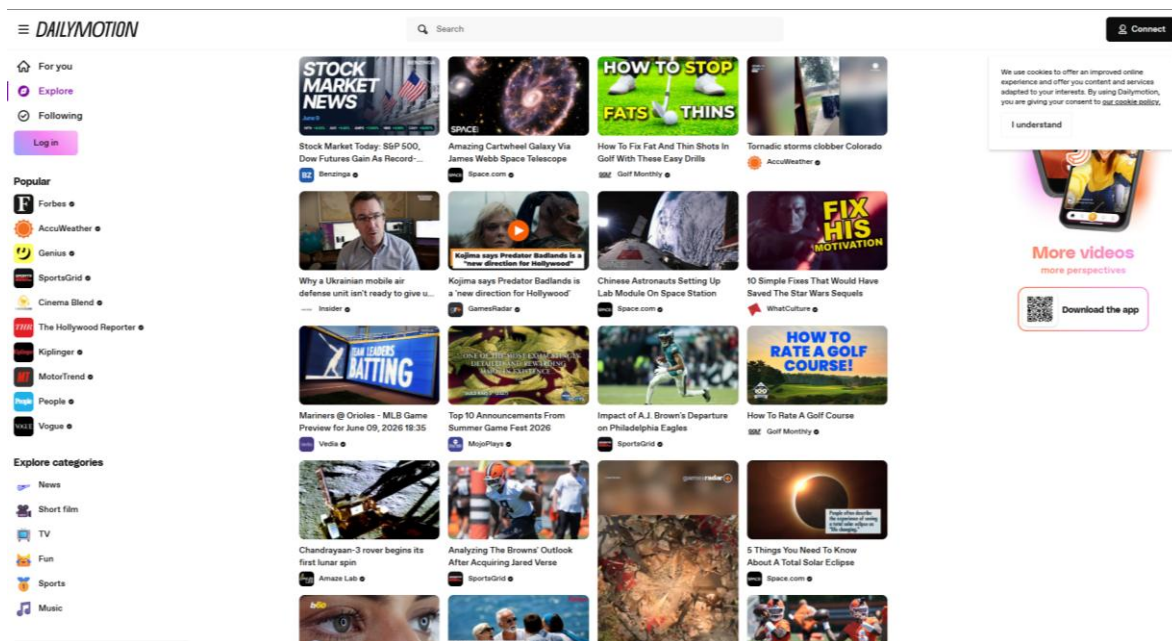


Рис. 1.3 – Інтерфейс відеохостингової платформи Dailymotion

PeerTube — це сучасна децентралізована та федеративна відеоплатформа з відкритим вихідним кодом, створена некомерційною організацією Framasoft у 2018 році. PeerTube базується на протоколі ActivityPub та технологіях P2P (WebTorrent / HLS). Замість єдиного центрального сервера PeerTube складається з тисяч незалежних серверів (інстансів), які можуть об'єднуватися у федерацію. Головною технологічною особливістю PeerTube є те, що глядачі під час перегляду відео автоматично роздають його частини іншим глядачам за технологією Peer-to-Peer, що значно знижує навантаження на пропускну здатність сервера. PeerTube повністю вільний від корпоративної цензури та алгоритмів стеження, проте має високий поріг входження: для створення власного інстансу потрібні значні технічні знання з адміністрування серверів, а відсутність єдиної пошукової бази ускладнює пошук відео між різними інстансами для пересічних користувачів.

На основі проведеного аналізу складено порівняльну таблицю основних характеристик аналогів та платформи, що розробляється «BenFube» (табл. 1.1).

					ИЗ.КР.Б - 121 - 26 - ИЗ	Арк.
						15

Таблиця 1.1

Порівняльна характеристика відеохостингових платформ

Характеристика	YouTube	Vimeo	Dailymotion	PeerTube	BenFube
Безкоштовне завантаження відео	+	±	+	+	+
Система підписок	+	+	±	+	+
Лайки та коментарі	+	+	±	+	+
Монетизація (донати)	±	-	-	-	+
Адмін-панель модерації	+	+	-	+	+
Система скарг	+	+	-	+	+
Алгоритм трендів	+	-	+	-	+
Аналітика для авторів	+	+	+	-	+
Відкритий вихідний код	-	-	-	+	-
Двомовний інтерфейс	+	+	+	+	+
Хмарне зберігання відео	+	+	+	-	+
Безкоштовна монетизація	-	-	-	-	+

Примітка: у таблиці 1.1 символом «+» позначено повну підтримку функції, «±» — обмежену підтримку або наявність додаткових умов, «-» — відсутність функції.

Аналіз наведених аналогів (детальніше порівняння критеріїв представлено в Таблиці 1.1) показує, що ринок відеохостингу розділений між монопольними централізованими платформами з жорсткою цензурою та складними алгоритмами монетизації (YouTube), дорогими корпоративними сервісами (Vimeo) та складними в адмініструванні децентралізованими рішеннями (PeerTube). Це створює нагальну потребу у розробці гнучкої, безпечної та простої у використанні платформи, яка поєднає найкращі сторони централізованого розміщення (простота доступу для глядачів та авторів) із прозорою та доступною системою монетизації через прямі благодійні внески (донати) безпосередньо від аудиторії без посередників. Проектована вебплатформа «BenFube» покликана заповнити цю нішу, надаючи авторам зручний інструмент управління контентом Studio, швидке завантаження через Cloudinary та гнучку модерацію скарг, що робить її ефективною альтернативою для локальних спільнот та незалежних творців контенту.

1.3. Архітектура системи та ролі користувачів

Проектування та розробка будь-якої сучасної вебплатформи, особливо у такій високозавантаженій медійній сфері, як відеохостинг, починається з глибокого аналізу та вибору архітектурного рішення. Архітектура визначає не лише спосіб організації та структурування програмних компонентів системи, але й принципи їхньої взаємодії, розподіл відповідальності між ними, а також масштабованість, відмовостійкість і легкість подальшої підтримки. У сучасній веброботі серед найбільш поширених архітектурних підходів виокремлюють три основні парадигми: монолітну архітектуру, мікросервісну архітектуру та клієнт-серверну архітектуру з чітким розподілом на бекенд (серверну частину) та фронтенд (клієнтську частину) [8]. Кожен із цих підходів має свої унікальні переваги та обмеження, що робить їх придатними для конкретних умов реалізації та масштабів проєкту.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						17

Монолітна архітектура передбачає об'єднання всіх функціональних модулів системи — від користувацького інтерфейсу та бізнес-логіки до роботи з базою даних та обробки медіафайлів — в межах єдиного великого застосунку, який розгортається, масштабується та оновлюється як цілісна система. Головною перевагою монолітного підходу є простота початкової розробки, налагодження та розгортання на початкових етапах життєвого циклу продукту. Всі виклики між модулями виконуються в межах одного процесу в оперативній пам'яті, що мінімізує мережеві затримки. Проте для відеохостингу монолітна архітектура створює серйозні проблеми при зростанні навантаження. Процеси транскодування відео (перекодування завантажених файлів у різні роздільні здатності) споживають величезну кількість ресурсів процесора (CPU) та дискової підсистеми. У моноліті це може призвести до того, що під час транскодування одного великого відео весь застосунок тимчасово перестане відповідати на запити звичайних користувачів, які намагаються просто переглянути головну сторінку.

Мікросервісна архітектура, навпаки, орієнтована на розбиття єдиної системи на набір дрібних, автономних сервісів, кожен із яких вирішує обмежену бізнес-задачу (наприклад, сервіс автентифікації, сервіс коментарів, сервіс транскодування відео) та взаємодіє з іншими за допомогою легковажних мережевих протоколів (REST HTTP, gRPC, черги повідомлень). Це забезпечує надзвичайну відмовостійкість: збій у роботі сервісу коментарів не заважає користувачам дивитися відео. Окремі мікросервіси можна масштабувати незалежно, виділяючи сервісу транскодування додаткові обчислювальні потужності у хмарі. Однак мікросервісний підхід вимагає значних ресурсів для оркестрації та автоматизації розгортання (Kubernetes, Docker Swarm), призводить до накладних мережевих витрат на міжсервісну комунікацію та суттєво ускладнює налагодження помилок і забезпечення узгодженості даних у розподілених транзакціях, що є надлишковим для проєктів на стадії розробки невеликими командами.

Враховуючи специфіку проєкту відеохостингу «BenFube», для реалізації платформи було обрано клієнт-серверну REST-архітектуру з розділенням на бекенд і фронтенд, що є найбільш раціональним компромісом між простотою монолітного підходу та гнучкістю мікросервісів. У такій схемі бекенд виступає як незалежний REST API сервер, який виконує бізнес-логіку, забезпечує авторизацію, взаємодіє з базою даних та надає ресурси за протоколом HTTP у форматі JSON без збереження стану (stateless). Фронтенд є окремим клієнтським застосунком, що взаємодіє з API та фокусується виключно на відображенні даних, обробці дій користувача та забезпеченні плавного інтерфейсу. Такий чіткий розподіл відповідальності дозволяє розробляти, тестувати та розгортати клієнтську та серверну частини незалежно на різних спеціалізованих хостингах (Vercel та Render відповідно), що спрощує горизонтальне масштабування та оптимізацію роботи системи [9].

Загальна архітектурна схема платформи «BenFube» включає кілька ключових взаємопов'язаних компонентів (Рис. 1.4). Клієнтський застосунок, розроблений на Next.js, розгортається на глобальній хмарній платформі Vercel, що забезпечує швидку доставку статичного контенту та серверний рендеринг сторінок. Серверна частина на базі Symfony 6 розгортається у вигляді Docker-контейнера на платформі Render. Для збереження структурованих даних користувачів, каналів, відеоматеріалів, лайків та підписок використовується реляційна база даних PostgreSQL. Для вирішення проблеми збереження важкого медіаконтенту без навантаження на власний сервер інтегровано хмарне сховище Cloudinary. Клієнт взаємодіє з бекендом виключно через шифроване з'єднання HTTPS, отримуючи токени JWT для підтвердження сесії. Процес завантаження медіафайлів оптимізовано: клієнт спочатку запитує одноразовий цифровий підпис у бекенду, після чого напряму завантажує відеофайл на сервери Cloudinary, передаючи бекенду лише отримане посилання на готовий медіаресурс.

Для забезпечення безпеки, розмежування прав та організації бізнес-логіки у межах платформи «BenFube» реалізовано рольову модель доступу (Role-Based

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						19

Access Control, RBAC). Всі користувачі розподіляються за чотирма основними ролями. Перша роль — це Гість (Anonymous User), тобто неавторизований відвідувач. Гість має обмежений доступ лише для читання публічного контенту: перегляду відкритої стрічки відео, пошуку відеозаписів за назвою та тегами, перегляду відкритих сторінок каналів та коментарів під відео. Гість не має доступу до функцій, що змінюють стан системи (лайки, підписки, коментарі, створення каналу). Друга роль — Зареєстрований користувач (User). Після авторизації через форму входу або Google OAuth він отримує JWT-токен та ширший набір прав: можливість оцінювати відеоматеріали (лайк/дизлайк), залишати коментарі, підписуватися на канали інших авторів, надсилати скарги на неприпустимий контент та здійснювати благодійні внески (донати) авторам.

Третя роль — Автор каналу (Channel Creator). Це зареєстрований користувач, який ініціював створення власного каналу. На додаток до базових прав користувача він отримує доступ до спеціалізованого кабінету «Studio» (творчої студії). Творча студія надає автору інструменти для завантаження нових відеозаписів, встановлення параметрів їхньої видимості (публічне, приватне, за посиланням), редагування метаданих (назва, опис, обкладинка, теги), видалення власних відео, а також перегляду аналітики каналу у вигляді інтерактивних графіків переглядів і підписок. Автор також може переглядати історію надісланих скарг на його відео та подавати апеляції на блокування. Четверта роль — Адміністратор (Admin). Це обліковий запис із найвищим рівнем привілеїв, призначений для управління та модерації всієї платформи. Адміністратор має доступ до окремої адмін-панелі, де може керувати обліковими записами (блокувати/розблоковувати користувачів), розглядати скарги на відеоматеріали, приймати рішення про блокування контенту або відхилення скарг, а також керувати глобальними налаштуваннями системи.

Представлена ієрархія ролей та їхніх прав доступу відображена на Рис. 1.5. Вона спроектована згідно з принципом найменших привілеїв (Principle of Least Privilege), що є фундаментальним правилом інформаційної безпеки. Кожна категорія користувачів отримує доступ лише до тих ресурсів та ендпоінтів API,

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						20

які є критично необхідними для виконання їхніх функцій. На боці Symfony-бекенду перевірка прав реалізується на рівні маршрутів контролерів за допомогою атрибутів безпеки, які зчитують роль користувача із розкодованого JWT-токена при кожному вхідному запиті. Це запобігає несанкціонованому доступу та спрощує аудит безпеки платформи.

Рисунок 1.4 — Загальна архітектура платформи «BenFube»

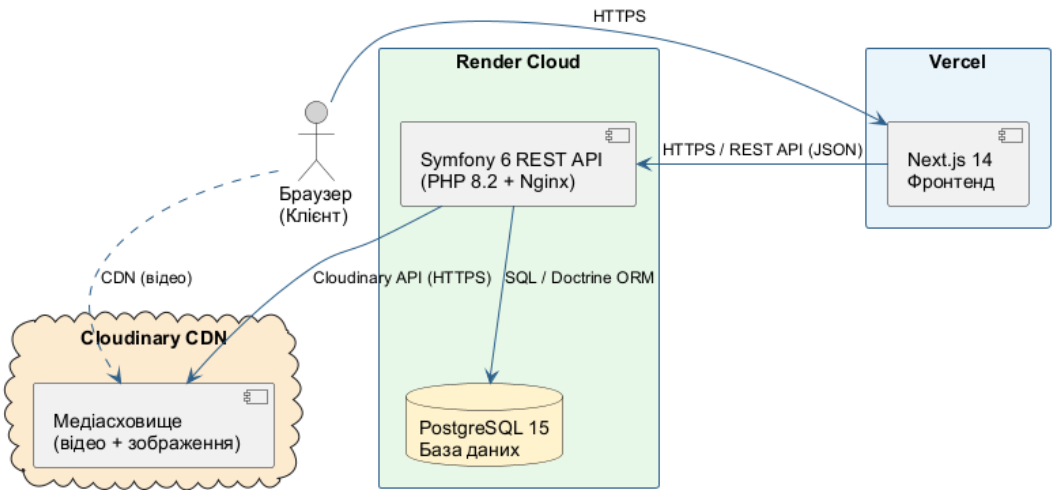


Рис. 1.4 – Схема загальної архітектури платформи BenFube

Рисунок 1.5 — Ієрархія ролей користувачів платформи «BenFube»

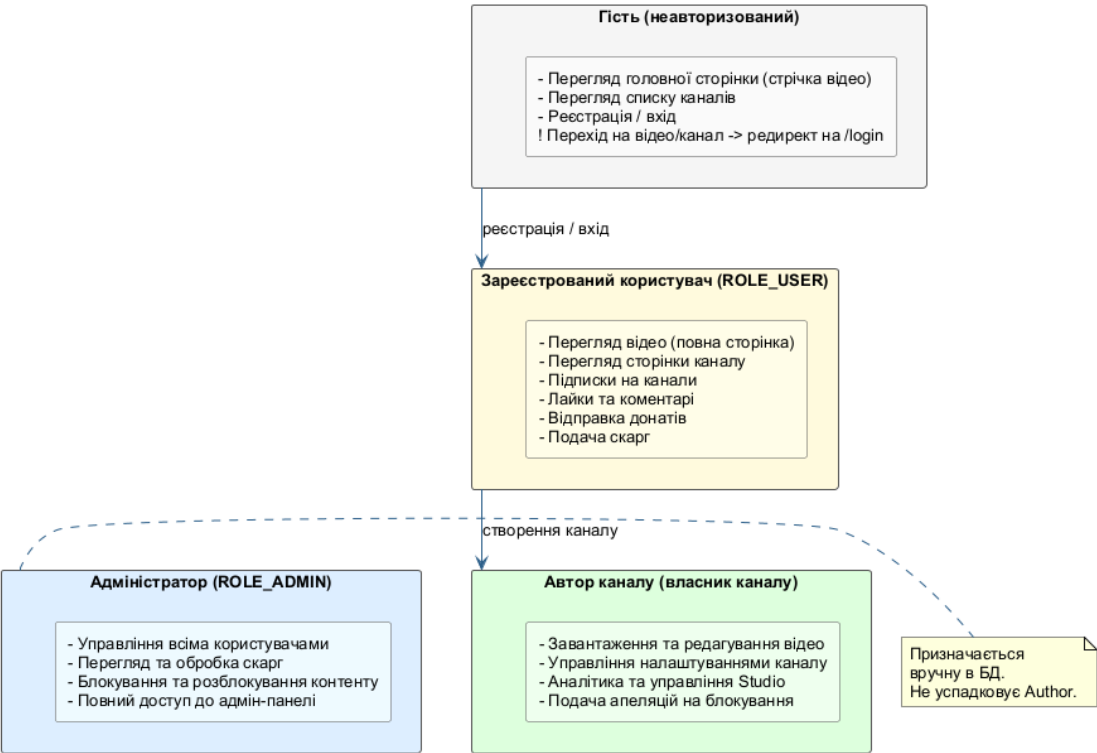


Рис. 1.5 – Схема ролей та їх прав доступу

1.4. Обґрунтування вибору технологічного стеку

Вибір технологій для реалізації вебплатформи відеохостингу є критично важливим етапом проєктування, який визначає продуктивність, безпеку, масштабованість, швидкість завантаження сторінок та легкість подальшого супроводу системи. Сучасні медіаплатформи оперують значними обсягами даних та високим навантаженням на мережу, тому архітектурне рішення має поєднувати зрілість серверних інструментів із високою реактивністю клієнтського інтерфейсу. У процесі вибору технологічного стеку для платформи «BenFube» було проведено порівняльний аналіз кількох провідних альтернатив для кожного компонента системи.

Для реалізації серверної частини обрано мову програмування PHP версії 8.2 у поєднанні з фреймворком Symfony 6. На сьогодні PHP залишається однією з найбільш затребуваних мов для веброзробки [10], що забезпечує низький поріг входження, високу швидкість виконання та широку сумісність. Версія PHP 8.2 приносить важливі оновлення: підтримку fiber-підпрограм для асинхронних операцій, readonly-класи для безпечного збереження стану DTO, покращену типізацію та суттєву оптимізацію OPcache. Symfony 6 обрано через його зрілу архітектуру та жорстку відповідність стандартам кодування PSR. Фреймворк використовує потужний контейнер впровадження залежностей (Dependency Injection), який забезпечує слабку зв'язаність компонентів бекенду. Порівняно з альтернативами (Laravel, Slim або NestJS на Node.js), Symfony надає гнучкіші механізми захисту за допомогою компонента Security, а наявність бібліотеки LexikJWTAuthenticationBundle дозволяє легко інтегрувати автентифікацію на основі JWT-токенів без написання низькорівневого коду шифрування.

У якості системи управління базами даних обрано PostgreSQL 15 — потужну об'єктно-реляційну СУБД з відкритим кодом [12]. На відміну від MySQL, PostgreSQL пропонує розширені можливості роботи зі складними типами даних, зокрема JSONB для збереження гнучких метаданих без втрати продуктивності індексації, а також підтримку UUID як первинних ключів. PostgreSQL забезпечує повноцінну підтримку транзакцій ACID, що є критичним

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						22

для забезпечення цілісності фінансових операцій (система донатів), де помилка під час оновлення балансів чи запису транзакції є неприпустимою. Для взаємодії з БД на бекенді використано Doctrine ORM [13]. Вона реалізує патерн проектування Data Mapper [28], який повністю розділяє бізнес-моделі (Entity) від структури бази даних, на відміну від патерну Active Record (використовуваного в Laravel Eloquent), де об'єкти безпосередньо прив'язані до рядків таблиць. Це робить код бекенду простішим для тестування та супроводу.

Для розробки клієнтської частини обрано фреймворк Next.js 14 на базі бібліотеки React [34] та мови програмування TypeScript [24]. Next.js надає розробнику архітектурний підхід App Router, який використовує файлову систему для маршрутизації та підтримує концепцію серверних компонентів (React Server Components). Це дозволяє рендерити статичні частини сторінок на сервері (SSR), значно знижуючи обсяг JavaScript-коду, що завантажується на пристрій користувача, та прискорюючи відображення контенту (First Contentful Paint). SSR є ключовим для відеохостингу з точки зору SEO-оптимізації [14], оскільки пошукові боти отримують повністю сформований HTML-код відео та каналів для індексації. TypeScript додає статичну типізацію до JavaScript, що дозволяє виявляти потенційні помилки типізації на етапі компіляції та надає широкі можливості автодоповнення в IDE. Для оформлення обрано Tailwind CSS — утилітарну CSS-бібліотеку, яка прискорює верстку адаптивних макетів та зменшує підсумковий розмір CSS-файлу завдяки вилученню невикористаних стилів (Purge CSS) [15].

Управління глобальним станом на фронтенді реалізовано за допомогою бібліотеки Zustand [33]. Це легковажний та гнучкий менеджер стану, що працює на основі React-хуків та не потребує складного boilerplate-коду (як Redux з його actions, reducers та dispatch). Zustand забезпечує високу швидкість реакції інтерфейсу, оскільки компоненти підписуються лише на конкретні частини глобального стану (наприклад, стан автентифікації користувача authStore чи обрану мову languageStore), що унеможливорює непотрібний повторний рендеринг усього дерева інтерфейсу.

Зберігання важких медіафайлів (відео та зображень обкладинок) здійснюється за допомогою хмарної платформи Cloudinary [16]. Відеохостинг потребує значних дискових ресурсів та процесорного часу для кодування відеофайлів у різні роздільні здатності. Використання Cloudinary дозволяє завантажувати файли безпосередньо з браузера клієнта за допомогою підписаних пресетів (Signed Upload Presets), повністю розвантажуючи сервер Symfony від обробки великих HTTP-запитів. Cloudinary автоматично трансформує медіафайли у сучасні вебформати (наприклад, WebP для зображень та WebM/MP4 для відео), оптимізує бітрейт під швидкість інтернету користувача та роздає медіаконтент через інтегровану глобальну мережу доставки вмісту (CDN).

Автентифікація користувачів реалізована за стандартом JSON Web Token (JWT) [17]. Бекенд Symfony працює у stateless-режимі (без збереження сесій на сервері), що дозволяє масштабувати систему горизонтально. При автентифікації сервер підписує токен асиметричним приватним RSA-ключем, а фронтенд Next.js зберігає його в cookies або localStorage та додає до HTTP-заголовка Authorization при кожному запиті до API. Для зручності користувачів інтегровано Google OAuth 2.0 [18], який дозволяє проходити автентифікацію за допомогою існуючого акаунту Google, підвищуючи рівень безпеки та спрощуючи процес реєстрації глядачів на платформі.

Для контейнеризації серверної частини застосовано технологію Docker [25], що ізолює додаток разом з усіма залежностями (PHP-FPM, Nginx, PostgreSQL) у контейнери. Це забезпечує ідентичність середовища розробки локально та на продакшн-серверах. Автоматичний процес CI/CD налаштовано за допомогою GitHub/GitLab Integration: при кожному пуші до гілки main бекенд Symfony автоматично деплоїться на хмарній платформі Render [35] (де збирається Docker-образ та запускається вебсервіс), а фронтенд Next.js — на платформі Vercel [19], яка ідеально оптимізована для рендерингу серверних застосунків Next.js.

Важливим аспектом проектування серверної архітектури є вибір механізму об'єктно-реляційного відображення (ORM). У фреймворку Symfony за замовчуванням інтегровано Doctrine ORM, що реалізує патерн Data Mapper (перетворювач даних). На відміну від патерну Active Record (який використовується в Laravel Eloquent), де кожна сутність безпосередньо знає про базу даних і містить методи збереження, Doctrine повністю розділяє бізнес-логіку (ентиті є чистими PHP-класами — POPO) та механізм доступу до бази даних (репозиторії). Це забезпечує вищий рівень абстракції, спрощує написання модульних тестів (Unit Tests) та запобігає забрудненню сутностей кодом інфраструктурного рівня. Unit of Work у Doctrine ORM оптимізує транзакції, групуючи запити до бази даних та виконуючи їх в одному блоці, що значно знижує навантаження на PostgreSQL.

Вибір PostgreSQL 15 як реляційної бази даних також обумовлений її суворою відповідністю стандартам ACID (Atomicity, Consistency, Isolation, Durability). Оскільки платформа BenFube інтегрує систему фінансової підтримки авторів (донатів) та веде облік балансу каналів, транзакційна цілісність даних є першочерговим пріоритетом. PostgreSQL надає надійні механізми ізоляції транзакцій, що виключає проблеми паралельного доступу (наприклад, фантомне читання чи брудне читання) під час одночасного виконання транзакцій донатів та оновлення лічильників статистики відео в таблиці video_statistics. Специфічні механізми блокувань (Row-level locking) дозволяють оновлювати записи без блокування всієї таблиці, що забезпечує високу пропускну здатність системи під навантаженням.

Для управління станом клієнтського застосунку було детально проаналізовано альтернативи Zustand та Redux Toolkit. Незважаючи на широку популярність Redux, його використання у проекті було відхилено через надмірну складність архітектури та велику кількість шаблону (boilerplate) для опису простих операцій. Zustand пропонує легковажний підхід (розмір бібліотеки менше 2 КБ), де стан зберігається в сховищі поза контекстом React. Це дозволяє уникнути зайвого рендерингу компонентів, оскільки Zustand використовує

селектори для підписки на зміни. Для забезпечення безперебійного користувацького досвіду (UX), сховище авторизації (authStore) автоматично синхронізується з локальним сховищем браузера (localStorage), що дозволяє зберігати сесію користувача після оновлення сторінки без додаткових запитів до API.

Хмарне сховище Cloudinary інтегрує в собі не лише функцію збереження файлів, а й потужний процесор обробки медіаконтенту. Традиційні сховища (як-от AWS S3) вимагають самостійного налаштування серверів перекодування (наприклад, за допомогою FFmpeg на бекенді), що призвело б до значних витрат процесорного часу та пам'яті на Render. Cloudinary бере на себе весь процес асинхронного перекодування завантажених відеофайлів у формати WebM та MP4 з використанням адаптивного бітрейту. Клієнт Next.js використовує пряме завантаження (Direct Upload), що виключає бекенд Symfony із ланцюжка передачі важких бінарних даних, дозволяючи API-серверу фокусуватися на обробці легких JSON-запитів.

Безпека REST API забезпечується за допомогою stateless-автентифікації на базі JSON Web Token (JWT). Використання традиційних сесій на сервері вимагало б додаткової синхронізації сесійного сховища (наприклад, через Redis) у разі горизонтального масштабування бекенду. Токени JWT підписуються за допомогою зашифрованого приватного ключа на сервері та передаються у заголовок Authorization при кожному HTTP-запиті. Інтеграція Google OAuth 2.0 дозволяє користувачам авторизуватися через сторонній провайдер довіри. При цьому фронтенд Next.js перенаправляє користувача на сервери Google, після чого отримує авторизаційний код та обмінює його на JWT-токен нашого бекенду, зберігаючи єдиний механізм автентифікації для всіх типів облікових записів.

Таблиця 1.2

Порівняння технологій за компонентами системи

Компонент	Обрана технологія	Альтернативи	Обґрунтування вибору
------------------	--------------------------	---------------------	-----------------------------

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						26

продовж. табл. 1.2

Бекенд-фреймворк	Symfony 6 (PHP 8.2)	Laravel, Slim, NestJS	Чітка архітектура, PSR-стандарти, JWT-bundle, зрілість екосистеми
База даних	PostgreSQL 15	MySQL, MongoDB, SQLite	Розширені типи даних, UUID, JSON, суворіший SQL, розширене індексування
ORM	Doctrine ORM	Eloquent, Cycle ORM	Нативна інтеграція з Symfony, Migrations, Repository-патерн
Фронтенд-фреймворк	Next.js 14 (React)	Vue.js / Nuxt, Angular, SvelteKit	App Router, SSR, React-екосистема, TypeScript, Vercel-інтеграція
Мова фронтенду	TypeScript	JavaScript	Статична типізація, рання діагностика помилок, кращий DX
CSS-бібліотека	Tailwind CSS	Bootstrap, CSS Modules, Styled Components	Утилітарний підхід, адаптивність, швидке прототипування
Управління станом	Zustand	Redux, Recoil, Jotai	Мінімальний boilerplate, хук-орієнтованість, легка вага

продовж. табл. 1.2

Медіасховище	Cloudinary	AWS S3 + CloudFront, Mux	CDN, трансформації, безкоштовний план, SDK для PHP та JS
Автентифікація	JWT + Google OAuth 2.0	Session-based, Firebase Auth	Stateless, стандарт RFC 7519, легка інтеграція з SPA
Контейнеризація	Docker + docker-compose	Podman, нативне встановлення	Ізольоване середовище, однакова конфігурація dev/prod
Хостинг бекенду	Render	Railway, Heroku, VPS	Безкоштовний план, автоматичне розгортання, підтримка Docker
Хостинг фронтенду	Vercel	Netlify, GitHub Pages	Нативна підтримка Next.js, CI/CD, Preview deployments

Висновки до розділу 1

У першому розділі проведено аналіз предметної галузі відеохостингових платформ та сформульовано вимоги до системи, що розробляється «BenFube».

Розглянуто та порівняно чотири відеохостингові платформи: YouTube, Vimeo, Dailymotion та PeerTube. Встановлено, що жодна з них не поєднує в собі повний набір соціальних функцій, системи прямої монетизації через донати, адміністративної панелі модерації та двомовного інтерфейсу в рамках безкоштовного доступу. Виявлені недоліки аналогів стали основою для визначення функціональних пріоритетів платформи, що розробляється.

Для платформи «BenFube» обрано клієнт-серверну REST-архітектуру з розподілом на незалежні бекенд та фронтенд застосунки. Визначено чотири ролі

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						28

користувачів системи: гість, зареєстрований користувач, автор каналу та адміністратор — з чітким розмежуванням прав доступу для кожної ролі.

Обґрунтовано вибір технологічного стеку: Symfony 6 на PHP 8.2 для серверної частини, PostgreSQL як системи управління базами даних, Next.js 14 з TypeScript для клієнтської частини, Cloudinary для хмарного зберігання медіафайлів, JWT та Google OAuth 2.0 для автентифікації, Docker для контейнеризації, Vercel та Render для хмарного розгортання. Порівняльний аналіз альтернатив дозволив аргументувати доцільність обраного технологічного стеку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ ВЕБПЛАТФОРМИ BENFUBE

2.1. Визначення вимог та варіантів використання системи

Проектування програмного комплексу розпочинається з визначення вимог — чіткого переліку функцій та обмежень, яким має відповідати система. Вимоги прийнято поділяти на функціональні та нефункціональні [20].

Функціональні вимоги описують конкретні дії, які система повинна виконувати, та реакції на вхідні дані. Нефункціональні вимоги визначають якісні характеристики системи — продуктивність, безпеку [27], надійність, зручність використання та підтримуваність відповідно до ДСТУ ISO/IEC 25010 [30]. На основі аналізу предметної галузі, проведеного у першому розділі, сформовано повний перелік вимог до платформи «BenFube».

Таблиця 2.1

Функціональні вимоги до платформи «BenFube»

№	Підсистема	Вимога
1	Автентифікація	Система повинна забезпечувати реєстрацію нових користувачів за унікальним email та нікнеймом
2	Автентифікація	Система повинна перевіряти унікальність email та нікнейму під час реєстрації
3	Автентифікація	Система повинна підтримувати вхід через email+пароль та через обліковий запис Google
4	Автентифікація	Система повинна видавати JWT-токен при успішній автентифікації та перевіряти його при кожному захищеному запиті

5	Канали	Зареєстрований користувач повинен мати можливість створити власний канал з унікальним нікнеймом, описом та банером
6	Канали	Автор повинен мати змогу редагувати дані свого каналу та завантажувати аватар і банер
7	Відео	Автор повинен мати можливість завантажити відеофайл та обкладинку із зазначенням назви, опису та тегів
8	Відео	Система повинна зберігати відеофайли у хмарному сховищі Cloudinary та записувати метадані до БД
9	Відео	Автор повинен мати змогу редагувати та видаляти власні відео
10	Відео	Система повинна відображати список відео за критеріями: нові, трендові, пошук, за тегом
11	Підписки	Авторизований користувач повинен мати можливість підписатись або відписатись від каналу
12	Лайки	Авторизований користувач повинен мати можливість поставити або зняти лайк на відео та коментар

13	Коментарі	Авторизований користувач повинен мати можливість залишати коментарі до відео та відповідати на коментарі інших
14	Коментарі	Автор коментаря або адміністратор повинні мати змогу видалити коментар
15	Донати	Авторизований користувач повинен мати можливість відправити донат автору каналу з вибором суми та валюти
16	Скарги	Авторизований користувач повинен мати можливість подати скаргу на відео або коментар
17	Модерація	Адміністратор повинен мати доступ до панелі керування скаргами та можливість блокувати або відновлювати відео
18	Адмін-панель	Адміністратор повинен мати доступ до списку користувачів та можливість блокувати їх облікові записи
19	Аналітика	Автор повинен мати доступ до статистики свого каналу: перегляди, лайки, коментарі, донати
20	Тренди	Система повинна автоматично ранжувати відео за алгоритмом трендів та відображати їх на головній сторінці

Повний перелік нефункціональних вимог до платформи «BenFube» із зазначенням критеріїв їх прийнятності наведено в таблиці 2.2.

Таблиця 2.2

Нефункціональні вимоги до платформи «BenFube»

Категорія	Вимога	Критерій прийнятності
Продуктивність	API повинен відповідати на типові запити (список відео, профіль) із часом відгуку до 500 мс	Час відповіді API < 500 мс для 95% запитів
Безпека	Паролі користувачів повинні зберігатись у вигляді хешу	Використання алгоритму bcrypt з cost-фактором ≥ 10
Безпека	API повинен захищати ресурси від несанкціонованого доступу	Усі захищені ендпоінти повертають 401/403 без валідного токена
Безпека	Система повинна бути захищена від CSRF-атак	Токен передається у заголовок Authorization, а не в cookie, що усуває класичний вектор CSRF-атак для stateless API
Масштабованість	Архітектура повинна дозволяти горизонтальне масштабування бекенду	Stateless REST API, відсутність серверних сесій
Доступність	Система повинна підтримувати двомовний інтерфейс	Повний переклад на українську та англійську мови

Підтримуваність	Код повинен відповідати стандартам PSR-12 (бекенд) та ESLint (фронтенд)	Відсутність критичних попереджень у linter-звітах
Надійність	Система повинна коректно обробляти помилки та повертати інформативні повідомлення	HTTP-коди 4xx/5xx з JSON-тілом помилки
UI/UX	Інтерфейс повинен бути адаптивним для мобільних пристроїв	Коректне відображення на екранах від 320px до 1920px

Для наочного подання сценаріїв взаємодії різних ролей зі системою побудовано UML-діаграму варіантів використання (Use Case Diagram) згідно зі стандартом UML 2.5.1 [29] (Рис. 2.1). Діаграма охоплює чотири актори: «Гість», «Зареєстрований користувач», «Автор каналу» та «Адміністратор». Кожен актор пов'язаний із відповідними варіантами використання через відносини асоціації та узагальнення (Автор є спеціалізацією Зареєстрованого користувача, Адміністратор є окремою роллю із специфічним набором прав).

Основні варіанти використання системи включають: перегляд відео (доступний для Гостя), реєстрацію та вхід (доступні для Гостя), підписку на канал, лайкування, коментування, донат (для Зареєстрованого), завантаження відео, управління каналом, перегляд аналітики (для Автора), модерацию скарг та управління користувачами (для Адміністратора).

Рисунок 2.1 — UML-діаграма варіантів використання платформи «BenFube»

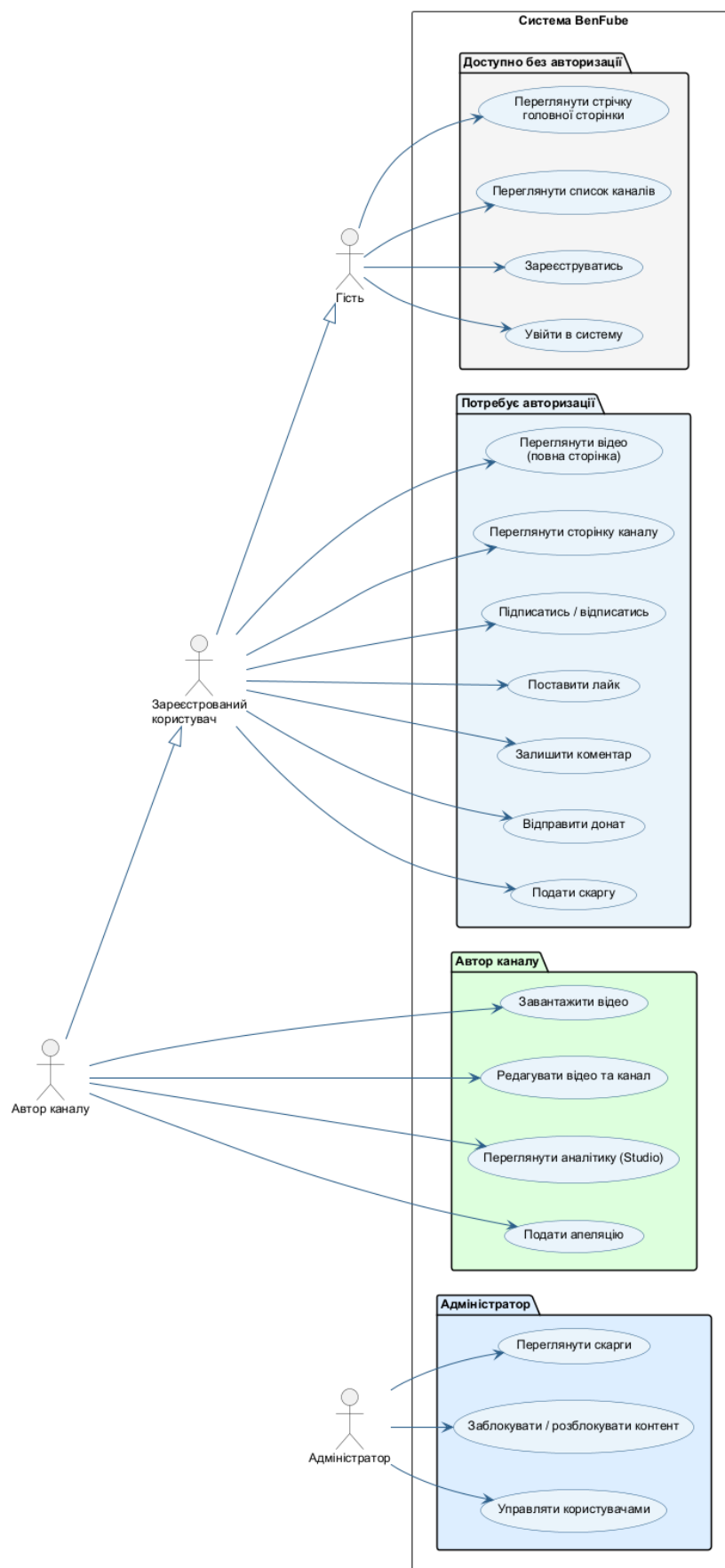


Рис. 2.1 – UML-діаграма варіантів використання платформи BenFube

2.2. Проектування бази даних

Проектування бази даних є ключовим етапом розробки будь-якої інформаційної системи, що визначає структуру зберігання даних, зв'язки між сутностями та забезпечує цілісність і узгодженість інформації. Проектування бази даних для платформи «BenFube» виконано у два етапи: концептуальний та фізичний.

На концептуальному етапі визначено основні сутності системи та зв'язки між ними у вигляді ER-діаграми (Entity-Relationship Diagram) (Рис. 2.2). Виявлено 12 основних сутностей: User (користувач), Channel (канал), Video (відео), Comment (коментар), Like (лайк), Subscription (підписка), Donation (донат), Report (скарга), Tag (тег), VideoTag (зв'язок відео-тег), VideoView (перегляд відео) та VideoStatistics (агрегована статистика відео).

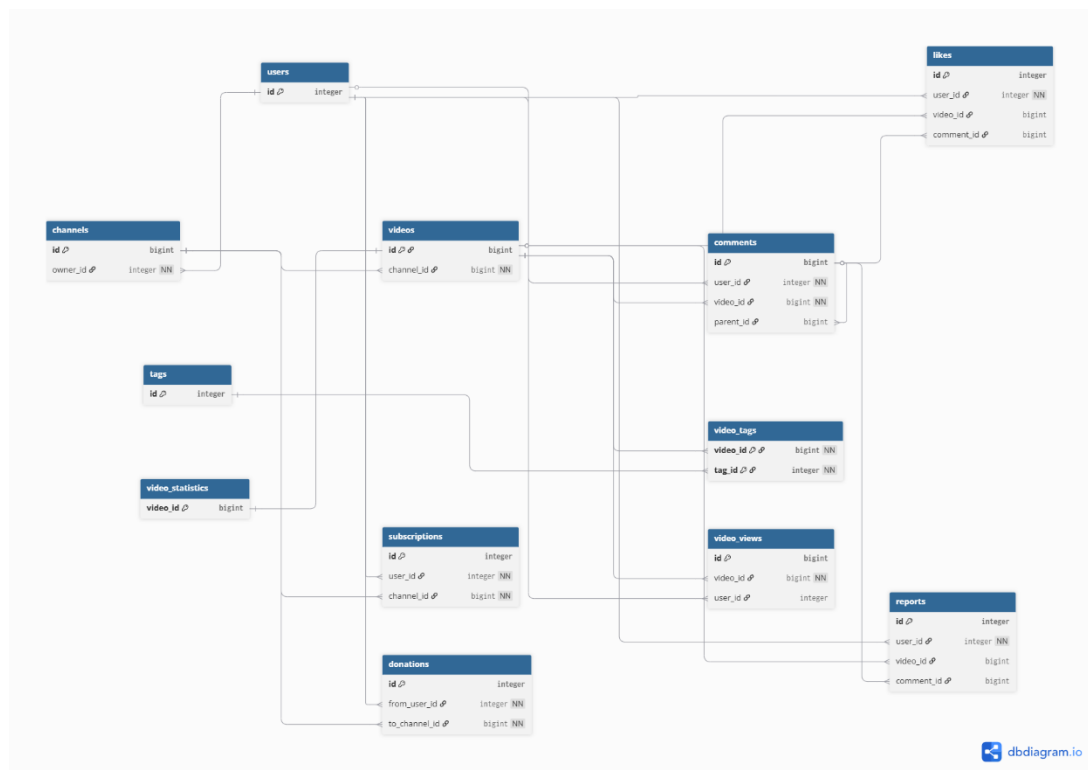


Рис. 2.2 – ER-діаграма бази даних платформи BenFube

На фізичному етапі кожна сутність відображена у відповідну таблицю бази даних PostgreSQL з визначенням типів даних, первинних і зовнішніх ключів, індексів та обмежень (Рис. 2.3).

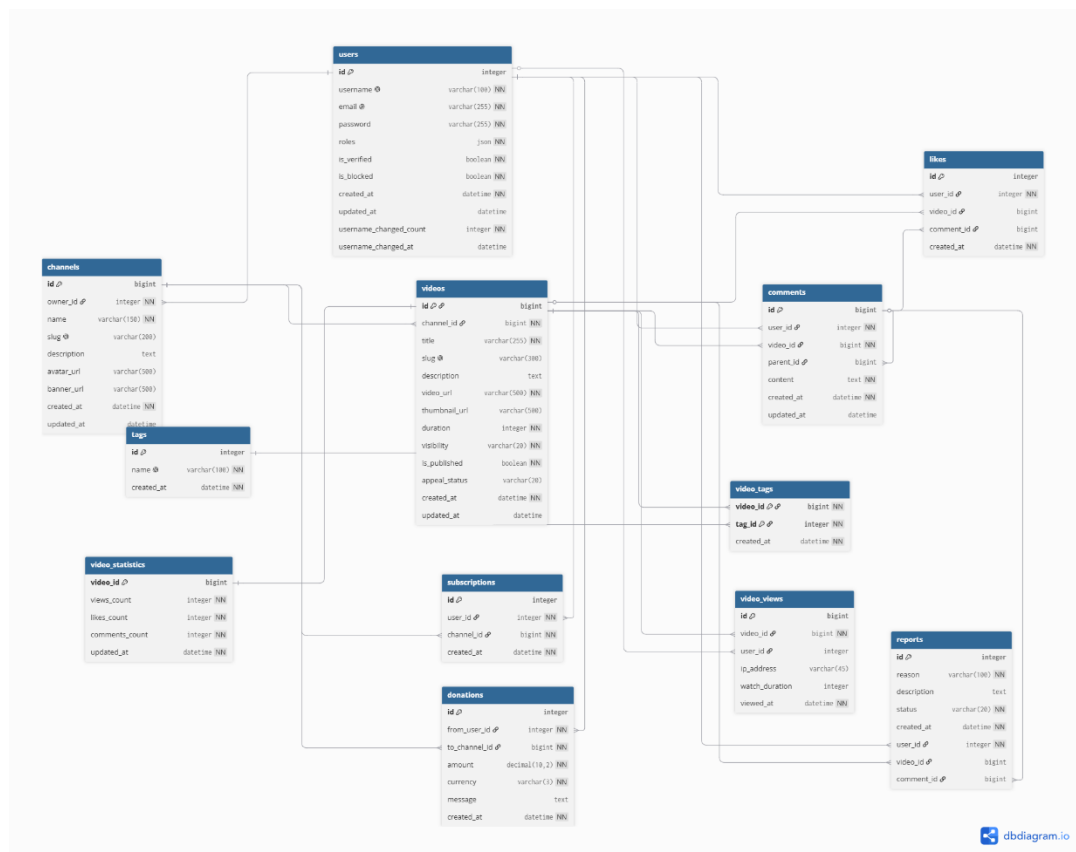


Рис. 2.3 – Схема фізичної структури бази даних PostgreSQL

Сутність User зберігає дані про зареєстрованих користувачів платформи. Таблиця містить такі поля: id (INTEGER, первинний ключ, автоінкремент), username (VARCHAR 100, унікальний, NOT NULL), email (VARCHAR 255, унікальний, NOT NULL), password (VARCHAR 255, NOT NULL — bcrypt-хеш), roles (JSON, NOT NULL, default: ["ROLE_USER"]), is_verified (BOOLEAN, default: false), is_blocked (BOOLEAN, default: false), created_at (DATETIME_IMMUTABLE, NOT NULL), updated_at (DATETIME, nullable), username_changed_count (INTEGER, NOT NULL, default: 0), username_changed_at (DATETIME_IMMUTABLE, nullable).

Сутність Channel представляє інформаційний простір автора (канал) та пов'язана з конкретним користувачем. До складу таблиці входять поля: id (BIGINT, первинний ключ, автоінкремент), owner_id (INTEGER, зовнішній ключ на users.id, CASCADE DELETE), name (VARCHAR 150, NOT NULL), slug (VARCHAR 200, унікальний, nullable), description (TEXT, nullable), avatar_url (VARCHAR 500, nullable), banner_url (VARCHAR 500, nullable), created_at

(DATETIME_IMMUTABLE, NOT NULL), updated_at (DATETIME, nullable). Між User та Channel встановлено відношення один-до-багатьох (один користувач може мати кілька каналів).

Сутність Video зберігає метадані відеозаписів. До складу таблиці входять поля: id (BIGINT, первинний ключ), channel_id (BIGINT, зовнішній ключ на channels.id, CASCADE DELETE), title (VARCHAR 255, NOT NULL), slug (VARCHAR 300, унікальний, nullable), description (TEXT, nullable), video_url (VARCHAR 500, NOT NULL) — посилання на файл у Cloudinary, thumbnail_url (VARCHAR 500, nullable), duration (INTEGER, секунди, NOT NULL), visibility (VARCHAR 20, NOT NULL, default: 'public'; допустимі значення: 'public', 'private', 'unlisted', 'blocked'), is_published (BOOLEAN, NOT NULL, default: true), appeal_status (VARCHAR 20, nullable), created_at (DATETIME_IMMUTABLE, NOT NULL), updated_at (DATETIME, nullable). Відношення Channel до Video — один-до-багатьох. Лічильники переглядів, лайків та коментарів зберігаються в окремій таблиці video_statistics для уникнення конкурентних оновлень.

Сутність Comment реалізує деревоподібну структуру коментарів через поле parent_id. До складу таблиці входять поля: id (BIGINT, первинний ключ), user_id (INTEGER, зовнішній ключ на users.id, CASCADE DELETE), video_id (BIGINT, зовнішній ключ на videos.id, CASCADE DELETE), parent_id (BIGINT, nullable, зовнішній ключ на comments.id, CASCADE DELETE) — для реалізації ієрархічних відповідей, content (TEXT, NOT NULL), created_at (DATETIME_IMMUTABLE, NOT NULL), updated_at (DATETIME, nullable). Видалення коментарів реалізовано двома окремими механізмами: на рівні бази даних налаштовано ON DELETE CASCADE для зовнішнього ключа parent_id (що забезпечує автоматичне видалення відповідей у БД при видаленні батьківського коментаря), а на рівні ORM Doctrine застосовано параметр orphanRemoval=true (що автоматично видаляє сутність відповіді з бази даних, якщо її від'єднано від колекції відповідей батьківського коментаря в коді).

Сутність Like реалізує систему лайків на відео та коментарі. До складу таблиці входять поля: id (INTEGER, первинний ключ), user_id (INTEGER,

зовнішній ключ на users.id, CASCADE DELETE), video_id (BIGINT, nullable, зовнішній ключ на videos.id, CASCADE DELETE), comment_id (BIGINT, nullable, зовнішній ключ на comments.id, CASCADE DELETE), created_at (DATETIME_IMMUTABLE, NOT NULL). Унікальні обмеження: пара (user_id, video_id) та пара (user_id, comment_id) — один користувач не може двічі вподобати один і той самий об'єкт. Або video_id, або comment_id є NOT NULL залежно від типу лайка.

Сутність Subscription фіксує підписки користувачів на канали. До складу таблиці входять поля: id (INTEGER, первинний ключ), user_id (INTEGER, зовнішній ключ на users.id, CASCADE DELETE), channel_id (BIGINT, зовнішній ключ на channels.id, CASCADE DELETE), created_at (DATETIME_IMMUTABLE, NOT NULL). Унікальне обмеження на пару (user_id, channel_id) виключає дублювання підписок.

Сутність Donation зберігає інформацію про відправлені донати. До складу таблиці входять поля: id (INTEGER, первинний ключ), from_user_id (INTEGER, зовнішній ключ на users.id, CASCADE DELETE), to_channel_id (BIGINT, зовнішній ключ на channels.id, CASCADE DELETE), amount (DECIMAL(10,2), NOT NULL), currency (VARCHAR 3, NOT NULL, default: 'USD') — код валюти за ISO 4217, message (TEXT, nullable), created_at (DATETIME_IMMUTABLE, NOT NULL).

Сутність Report реалізує систему скарг. До складу таблиці входять поля: id (INTEGER, первинний ключ), user_id (INTEGER, зовнішній ключ на users.id, CASCADE DELETE), video_id (BIGINT, nullable, зовнішній ключ на videos.id, CASCADE DELETE), comment_id (BIGINT, nullable, зовнішній ключ на comments.id, CASCADE DELETE), reason (VARCHAR 100, NOT NULL), description (TEXT, nullable), status (VARCHAR 20, NOT NULL) — допустимі значення: 'pending', 'reviewed', 'rejected', created_at (DATETIME_IMMUTABLE, NOT NULL). Статус скарги визначає її поточний стан у процесі модерації. Або video_id, або comment_id є NOT NULL — скарга стосується або відео, або коментаря.

Сутності Tag та VideoTag реалізують систему тегування відео за принципом «багато-до-багатьох». Tag зберігає унікальні теги (id INTEGER, name VARCHAR 100, унікальний, created_at DATETIME_IMMUTABLE). VideoTag є проміжною таблицею зв'язку між Video та Tag (video_id BIGINT, tag_id INTEGER, первинний ключ — складений із пари обох полів, created_at DATETIME_IMMUTABLE).

Сутність VideoView фіксує кожен перегляд відео. До складу таблиці входять поля: id (BIGINT, первинний ключ), video_id (BIGINT, зовнішній ключ на videos.id, CASCADE DELETE), user_id (INTEGER, nullable, зовнішній ключ на users.id, SET NULL — nullable для гостей), ip_address (VARCHAR 45, nullable), watch_duration (INTEGER, секунди, nullable), viewed_at (DATETIME_IMMUTABLE, NOT NULL). Зберігання ip_address дозволяє реалізувати захист від накрутки переглядів з одного джерела.

Сутність VideoStatistics зберігає агреговану статистику кожного відео для швидкого читання без необхідності підрахунку COUNT при кожному запиті. До складу таблиці входять поля: video_id (BIGINT, первинний ключ та одночасно зовнішній ключ на videos.id, CASCADE DELETE) — зв'язок один-до-одного з Video, views_count (INTEGER, NOT NULL, default: 0), likes_count (INTEGER, NOT NULL, default: 0), comments_count (INTEGER, NOT NULL, default: 0), updated_at (DATETIME_IMMUTABLE, NOT NULL). Окрема таблиця статистики усуває проблему «гарячих рядків» при одночасному оновленні лічильників кількома користувачами.

Таблиця 2.3

Основні сутності бази даних платформи «BenFube»

Сутність (таблиця)	Кількість полів	Ключові зв'язки	Призначення
User	11	→ Channel (1:M), → Like (1:M), → Subscription (1:M)	Дані облікових записів

Channel	9	← User (M:1), → Video (1:M)	Канали авторів
Video	13	← Channel (M:1), → Comment (1:M), → VideoTag (1:M)	Метадані відеозаписів
Comment	8	← Video (M:1), ← User (M:1), self (дерево)	Коментарі та відповіді
Like	4	← User (M:1), → Video або Comment	Вподобання відео та коментарів
Subscription	4	← User (M:1), ← Channel (M:1)	Підписки на канали
Donation	6	← User (M:1), ← Channel (M:1)	Донатити авторам
Report	8	← User (M:1), → Video або Comment	Скарги на контент

Tag	3	← VideoTag (M:M через VideoTag)	Теги для категоризації
VideoTag	2	← Video (M:1), ← Tag (M:1)	Зв'язок відео-тег
VideoView	5	← Video (M:1), ← User (M:1, nullable)	Журнал переглядів
VideoStatistics	5	← Video (1:1)	Агрегована статистика відео

Для забезпечення ефективності запитів до бази даних визначено додаткові індекси на полях: videos.channel_id, videos.created_at, videos.visibility, videos.slug, comments.video_id, comments.parent_id, comments.user_id, subscriptions.user_id, subscriptions.channel_id, reports.status, reports.video_id, video_views.video_id. Унікальні обмеження встановлено на: users.email, users.username, channels.slug, tags.name, поєднання (likes.user_id, likes.video_id), поєднання (likes.user_id, likes.comment_id), поєднання (subscriptions.user_id, subscriptions.channel_id).

2.3. Алгоритми роботи системи

Проектування алгоритмів функціонування системи є важливим кроком, що передуює безпосередній реалізації програмного коду. У даному підрозділі описано ключові алгоритми платформи «BenFube» з використанням UML-діаграм послідовності (Sequence Diagram) та активності (Activity Diagram).

Процес автентифікації користувачів у системі реалізовано на основі стандарту JSON Web Token (JWT) та детально відображено на діаграмі послідовності (Рис. 2.4). Відповідно до розробленого алгоритму, клієнтський застосунок Next.js надсилає POST-запит до бекенд-ендпоінту /api/auth/login із обліковими даними (email та password). Контролер AuthController приймає запит

та передає отримані дані до сервісу AuthService, який виконує пошук користувача за адресою електронної пошти через UserRepository. Якщо користувача з таким email не знайдено або введений пароль не відповідає збереженому в базі даних bcrypt-хешу, сервіс повертає оновлену помилку 401 Unauthorized. У разі успішної перевірки сервіс JWTService генерує підписаний токен, який повертається клієнту у тілі відповіді. Клієнт зберігає токен у локальній пам'яті браузера та додає його до заголовка Authorization при кожному наступному запиті. При надходженні запиту вбудований security-компонент Symfony Security зчитує заголовок, верифікує підпис токена та перевіряє його термін дії, надаючи доступ до захищених ресурсів API.

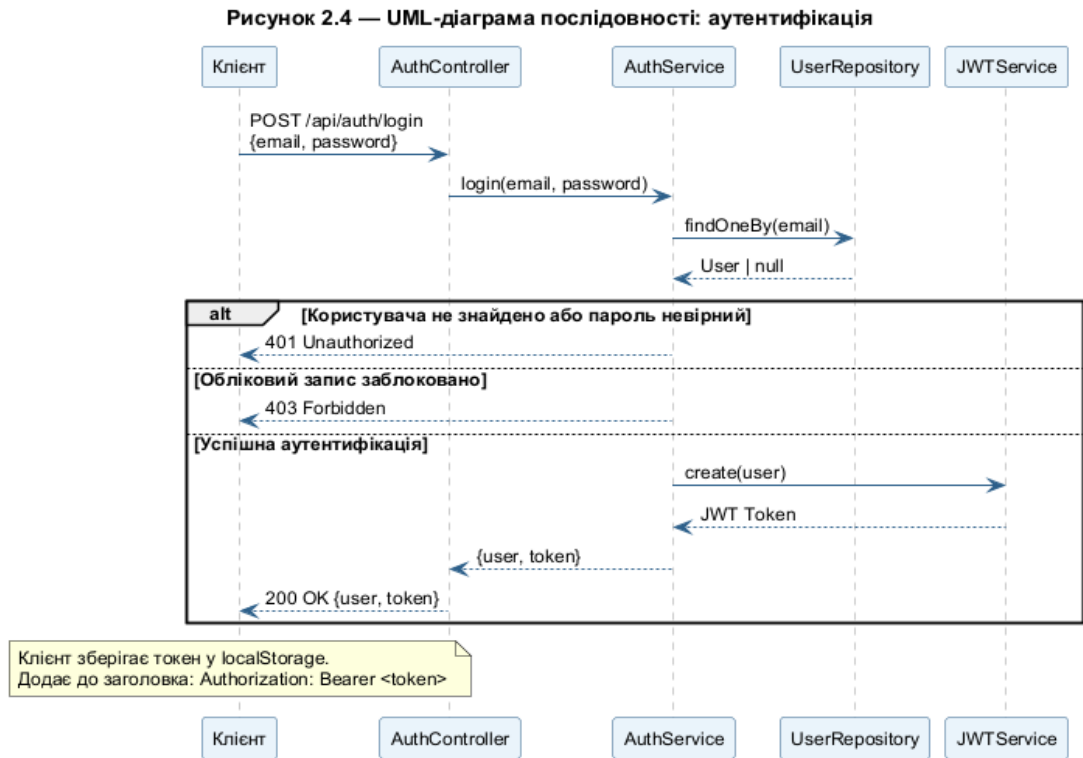


Рис. 2.4 – UML-діаграма послідовності процесу автентифікації

Алгоритм завантаження відео описано у вигляді UML-діаграми послідовності (Рис. 2.5). Процес завантаження передбачає безпосередню взаємодію фронтенду з Cloudinary, що розвантажує бекенд від передачі великих файлів. Клієнт надсилає POST-запит до бекенду для отримання підписаного upload_preset. VideoController перевіряє автентифікацію та генерує підписані параметри запиту до Cloudinary. Фронтенд безпосередньо надсилає відеофайл на

Cloudinary API. Cloudinary повертає публічну URL-адресу відео та його тривалість. Фронтенд надсилає POST-запит до /api/videos з метаданими відео (title, description, video_url, thumbnail_url, tags). VideoController викликає VideoService для збереження запису у БД. VideoService зберігає сутність Video та пов'язані теги через VideoTagRepository. Клієнт отримує дані збереженого відео у відповіді.

Рисунок 2.5 — UML-діаграма послідовності: завантаження відео

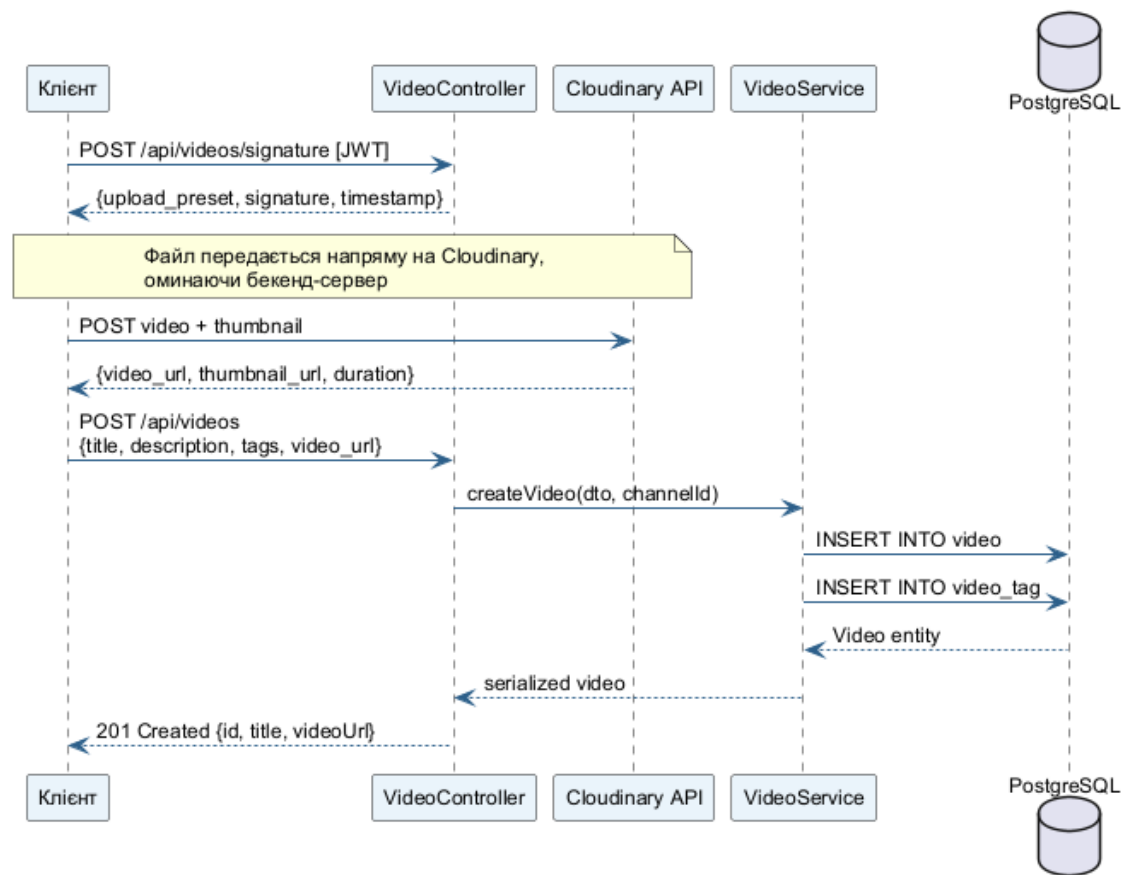


Рис. 2.5 – UML-діаграма послідовності процесу завантаження відео

Алгоритм трендів є одним із ключових алгоритмів платформи, що визначає порядок відображення відео на головній сторінці у вкладці «Трендові». Розроблений алгоритм базується на зваженій сумі показників активності відео за фіксований часовий проміжок (48 годин). Формула розрахунку рейтингу тренду має вигляд:

$$R = V \times w_1 + L \times w_2 + C \times w_3 \tag{2.1}$$

де R — рейтинг тренду; V — кількість переглядів відео за останні 48 годин;

L — кількість лайків за останні 48 годин; C — кількість коментарів за останні 48

годин; $w_1 = 1,0$; $w_2 = 5,0$; $w_3 = 10,0$ — вагові коефіцієнти, що відображають відносну важливість кожного виду активності.

Вагові коефіцієнти визначено таким чином, що соціальна взаємодія (лайк, коментар) має вищу вагу порівняно з простим переглядом, оскільки вона свідчить про більш активне залучення аудиторії до контенту. Коментар отримав найвищий коефіцієнт як найбільш значима форма соціальної взаємодії під відео. Для запобігання застою старих популярних відео у трендах до базового рейтингу R застосовується додатковий мультиплікативний коефіцієнт свіжості, що логарифмічно зменшується з часом: $R_final = R * (1.0 + 1.0 / \log_2(\text{ageHours} + 1))$, де ageHours — вік відео в годинах з моменту публікації. Це надає суттєву перевагу новим відео в перші години після завантаження. Підсумковий список відео сортується за значенням R_final за спаданням, а відео без активності за останні 48 годин виключаються з трендів.

Алгоритм модерації скарг описано за допомогою UML-діаграми активності (Рис. 2.6). Процес розпочинається з подання скарги авторизованим користувачем. Нова скарга отримує статус *pending* та зберігається у БД. Адміністратор переглядає список активних скарг у адмін-панелі та може прийняти одне з двох рішень. Якщо скарга підтверджується (*approved*) — відповідний відеозапис або коментар блокується (поле *visibility* змінюється на 'blocked'), що робить його недоступним для перегляду. Автор заблокованого контенту отримує можливість подати апеляцію з поясненням. Адміністратор розглядає апеляцію: у разі її задоволення — контент розблоковується, скарга відхиляється; у разі відмови — блокування залишається в силі. Якщо скарга відхиляється (*rejected*) — контент залишається доступним без змін.

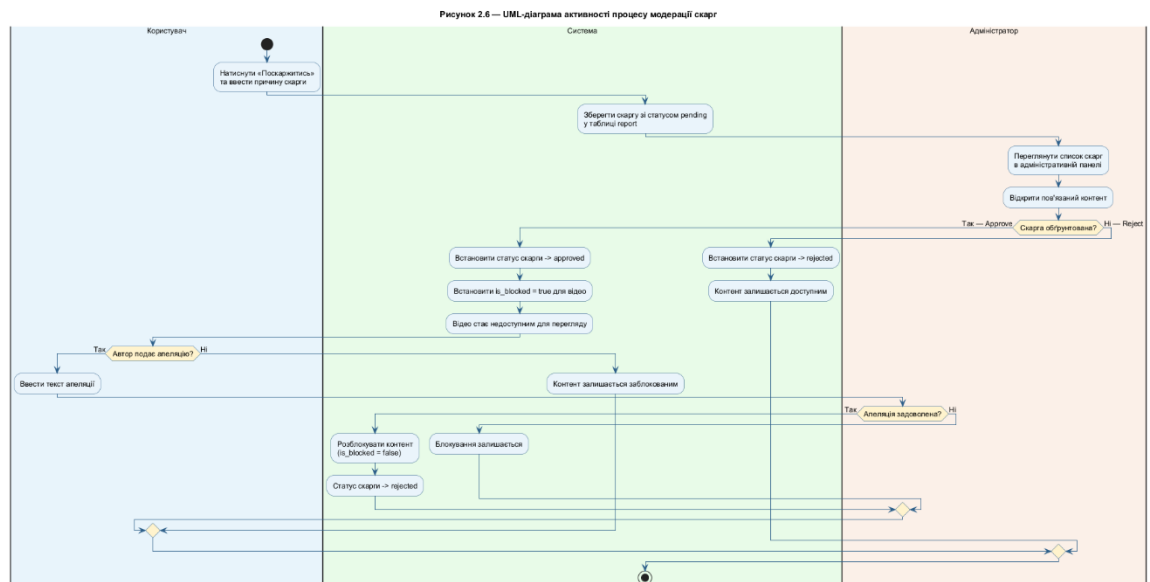


Рис. 2.6 – UML-діаграма активності процесу модерації скарг

Алгоритм реєстрації нового користувача розроблено з урахуванням сучасних вимог до безпеки [27] та захисту персональних даних. Користувач заповнює форму реєстрації на фронтенді, вказуючи адресу електронної пошти (email), пароль та ім'я користувача (username). Клієнтська частина Next.js перевіряє правильність введених даних (формат email, мінімальну довжину пароля у 8 символів) за допомогою бібліотеки Zod. Після цього дані надсилаються у вигляді JSON-об'єкта на API-ендпоінт бекенду /api/auth/register методом POST. Бекенд Symfony виконує повторну валідацію на рівні сутностей. Насамперед виконується перевірка на унікальність email та username в базі даних. Якщо користувач із такими даними вже існує, сервер повертає статус 400 Bad Request із відповідним повідомленням про помилку. Пароль користувача не зберігається у відкритому вигляді — він хешується за допомогою алгоритму bcrypt із робочим фактором (cost) 12. Після успішного хешування створюється новий запис у таблиці User, автоматично генерується сутність Channel (оскільки кожен користувач має свій канал за замовчуванням), і транзакція фіксується в базі даних. Користувачу повертається статус 201 Created та JWT-токен для автоматичної авторизації.

Алгоритм підписки на канал (Subscription) працює за принципом перемикача (toggle). При натисканні кнопки «Підписатися» фронтенд Next.js

перевіряє авторизацію користувача. Якщо користувач не авторизований, відображається модальне вікно входу. Авторизований користувач надсилає запит POST на `/api/subscriptions`. Бекенд перевіряє, чи не намагається користувач підписатися на свій власний канал (це заборонено бізнес-логікою). Далі виконується перевірка в таблиці Subscription: якщо запис про підписку між даним користувачем та каналом уже існує, це означає відписку — запис видаляється, а лічильник підписників каналу декрементується. Якщо запису немає, створюється новий запис, а лічильник інкрементується. Вся операція виконується в межах однієї бази даних під захистом транзакції бази даних для забезпечення узгодженості лічильників. Аналогічно працює алгоритм оцінювання відео (Like/Dislike) через ендпоінт `/api/likes` методом POST. Якщо користувач ставить лайк, перевіряється, чи не стоїть у нього дизлайк на це відео. Якщо стоїть, дизлайк видаляється, а лайк додається. Усі зміни фіксуються в таблиці Like та оновлюють лічильники в таблиці Video.

2.4. Реалізація бекенду на основі Symfony та REST API

Серверна частина платформи «BenFube» реалізована на мові PHP 8.2 з використанням фреймворку Symfony 6. Бекенд функціонує як RESTful API-сервер, що приймає HTTP-запити від клієнтського застосунку, обробляє бізнес-логіку та повертає відповіді у форматі JSON. Архітектура бекенду побудована за принципом розподілу відповідальності між чотирма шарами: Controller, Service, Repository та Entity відповідно до принципів Clean Architecture [22] (Рис. 2.7).

Рисунок 2.7 — Шарова архітектура бекенду платформи «BenFube»

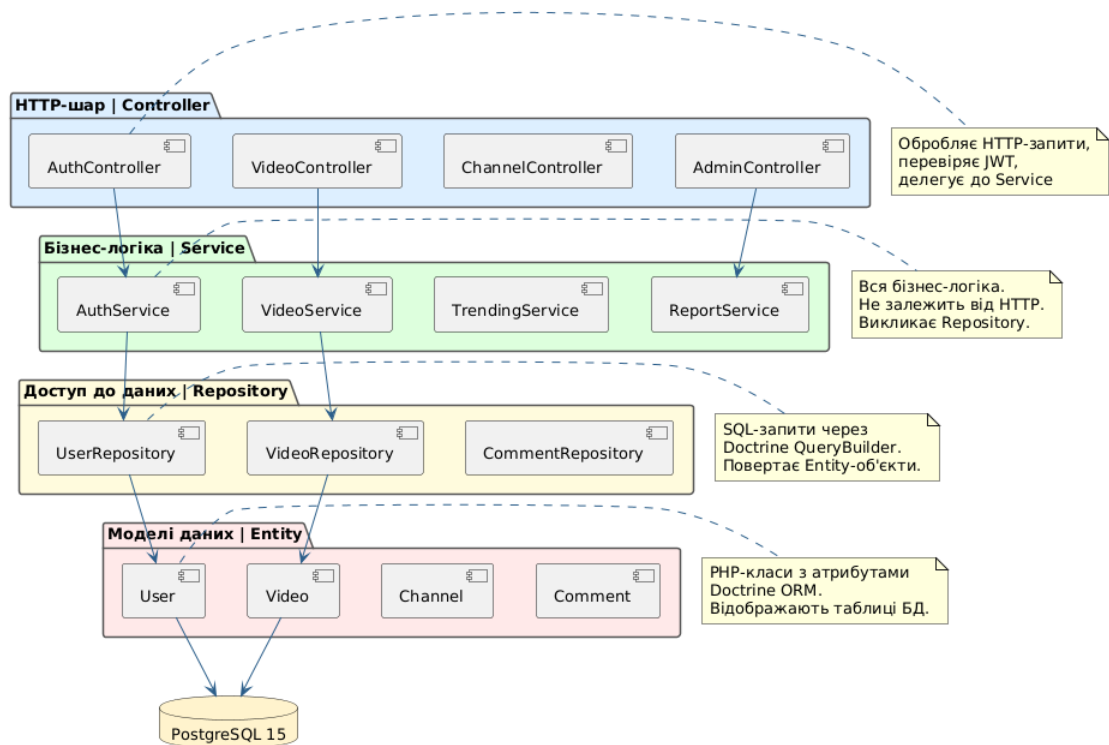


Рис. 2.7 – Схема архітектурних шарів бекенду

Шар Entity (`src/Entity/`) містить PHP-класи, що відображають таблиці бази даних. Кожна сутність анотована атрибутами Doctrine ORM (`#[ORM\Entity]`, `#[ORM\Column]`, `#[ORM\ManyToOne]` тощо), що визначають структуру відповідної таблиці та зв'язки між сутностями. Doctrine автоматично генерує SQL-схему на основі Entity-класів та синхронізує її з базою даних через механізм міграцій.

Шар Repository (`src/Repository/`) містить класи для виконання запитів до БД. Кожен Repository успадковує `ServiceEntityRepository` та може містити кастомні методи для складних запитів через Doctrine Query Builder або нативний SQL. Такий підхід ізолює логіку роботи з даними від бізнес-логіки.

Шар Service (`src/Service/`) містить класи, що реалізують бізнес-логіку застосунку. Сервіси отримують дані через Repository, виконують необхідні операції та повертають результат контролеру. Сервіси також відповідають за взаємодію із зовнішніми сервісами (Cloudinary API) та формування відповідей.

Шар Controller (`src/Controller/`) містить класи-контролери, що обробляють HTTP-запити. Кожен контролер є тонким шаром між HTTP-запитом та сервісом:

він виконує валідацію вхідних даних через DTO та делегує всю бізнес-логіку відповідному сервісу. Контролери анотовані атрибутом `#[Route]` для визначення URL-маршрутів та `#[IsGranted]` для обмеження доступу за роллю.

Нижче наведено фрагмент коду `VideoController`, що демонструє обробку запиту на отримання списку відео з підтримкою пошуку, сортування та пагінації:

```
#[Route('/api/videos', name: 'api_videos_list', methods: ['GET'])]
public function list(Request $request, VideoService $videoService): JsonResponse
{
    $search = $request->query->get('search', '');
    $sort    = $request->query->get('sort', 'newest');
    $page    = max(1, (int) $request->query->get('page', 1));
    $limit   = min(20, max(1, (int) $request->query->get('limit', 12)));

    $result = $videoService->getVideoList($search, $sort, $page, $limit);

    return $this->json([
        'data'      => $result['videos'],
        'pagination' => [
            'page'      => $page,
            'limit'     => $limit,
            'total'     => $result['total'],
            'totalPages' => ceil($result['total'] / $limit),
        ],
    ]);
}
```

Контролери в архітектурі платформи «BenFube» виступають у ролі тонкого шару представлення, що відповідає за прийом HTTP-запитів, базову валідацію вхідних параметрів та формування стандартизованих JSON-відповідей. Представлений вище метод `'list'` класу `'VideoController'` демонструє обробку запитів для отримання переліку відеоматеріалів із підтримкою гнучкої фільтрації. Контролер зчитує GET-параметри запиту, такі як пошуковий запит `'search'`, критерій сортування `'sort'`, номер поточної сторінки `'page'` та ліміт елементів на сторінку `'limit'`. Зчитування здійснюється за допомогою об'єкта `'Request'` компонента HTTP Foundation. Після цього дані передаються до прикладного сервісу `'VideoService'`, який приховує складну бізнес-логіку взаємодії з репозиторіями та базою даних. Такий підхід дозволяє зберегти контролер лаконічним і спростити його юніт-тестування.

Після отримання результатів від прикладного сервісу контролер формує відповідь у вигляді об'єкта `'JsonResponse'`, який автоматично встановлює HTTP-заголовок `'Content-Type: application/json'` та серіалізує масив даних у формат JSON. Структура відповіді містить два основні блоки: `'data'`, що містить серіалізований масив сутностей відео, та `'pagination'`, який передає метадані

пагінації (номер поточної сторінки, ліміт, загальну кількість записів та загальну кількість сторінок). Це дозволяє клієнтській частині на Next.js динамічно будувати інтерфейс списків та елементи керування пагінацією. Завдяки використанню анотацій маршрутизації Symfony ``#[Route]``, контролер чітко визначає URL-шлях ``/api/videos``, унікальне ім'я маршруту та обмеження за методом HTTP (в даному випадку дозволено лише ``GET``), що повністю відповідає принципам проектування RESTful API.

Для забезпечення високої продуктивності при виборі контенту, який часто змінюється та потребує складних математичних обчислень, стандартних засобів ORM Doctrine (таких як Query Builder) виявляється недостатньо. Зокрема, алгоритм формування трендового відеоконтенту потребує агрегації даних з кількох пов'язаних таблиць (переглядів, лайків та коментарів) за обмежений проміжок часу з подальшим застосуванням логарифмічного затухання. Для оновлення даних та оптимізації цього процесу на рівні бази даних було вирішено відмовитися від автоматично згенерованих ORM-запитів на користь нативних SQL-запитів із ручним мапінгом результатів. Це дозволило перенести обчислення агрегованих показників безпосередньо на бік PostgreSQL, мінімізуючи обсяг переданих даних між СУБД та сервером застосунків. Реалізацію даної логіки винесено в репозиторій ``VideoRepository`` у метод ``findTrending()``, код якого наведено нижче.

```
public function findTrending(int $limit = 10, string $period = '48h'): array
{
    $since = match($period) {
        '24h' => new \DateTimeImmutable('-24 hours'),
        default => new \DateTimeImmutable('-48 hours'),
    };
    $conn = $this->getEntityManager()->getConnection();
    $sinceStr = $since->format('Y-m-d H:i:s');

    // Виконання нативних SQL-запитів для отримання статистики (переглядів, лайків,
    коментарів)
    $viewsMap = $this->fetchStatsMap($conn, "video_views", "viewed_at", $sinceStr);
    $likesMap = $this->fetchStatsMap($conn, "likes", "created_at", $sinceStr);
    $commentsMap = $this->fetchStatsMap($conn, "comments", "created_at", $sinceStr);

    $videoIds = array_unique(array_merge(array_keys($viewsMap), array_keys($likesMap),
    array_keys($commentsMap)));
    if (empty($videoIds)) return [];

    $videos = $this->createQueryBuilder('v')->leftJoin('v.channel', 'c')->addSelect('c')
    ->andWhere('v.id IN (:ids)')->setParameter('ids', $videoIds)->getQuery()->getResult();

    $scored = [];
    foreach ($videos as $video) {
        $id = $video->getId();
```

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						50

```

    $score = (($viewsMap[$id] ?? 0) * 1.0) + (($likesMap[$id] ?? 0) * 5.0) +
    (($commentsMap[$id] ?? 0) * 10.0);

    // Врахування коефіцієнта свіжості (логарифмічне згасання)
    $createdAt = $video->getCreatedAt();
    if ($createdAt !== null) {
        $ageHours = max(1, (time() - $createdAt->getTimestamp()) / 3600);
        $score *= (1.0 + (1.0 / log($ageHours + 1, 2)));
    }
    $scored[] = ['video' => $video, 'score' => $score];
}
usort($scored, fn($a, $b) => $b['score'] <=> $a['score']);
return array_slice(array_column($scored, 'video'), 0, $limit);
}

```

Таблиця 2.4 містить перелік основних REST API маршрутів платформи з зазначенням методів HTTP, URL-шляхів, необхідного рівня доступу та призначення.

Таблиця 2.4

Основні REST API маршрути платформи «BenFube»

Метод	URL	Доступ	Призначення
POST	/api/auth/register	Публічний	Реєстрація нового користувача
POST	/api/auth/login	Публічний	Вхід, отримання JWT
GET	/api/auth/google	Публічний	Ініціювання OAuth-потоків Google
GET	/api/videos	Публічний	Список відео (пошук, сортування, пагінація)
GET	/api/videos/{id}	Публічний	Деталі відео за ідентифікатором
POST	/api/videos	Автор	Публікація нового відео
PATCH	/api/videos/{id}	Автор	Редагування власного відео
DELETE	/api/videos/{id}	Автор	Видалення власного відео

GET	/api/videos/trending	Публічний	Список трендових відео
GET	/api/channels/{slug}	Публічний	Профіль каналу за slug
POST	/api/channels	Авториз.	Створення каналу
PATCH	/api/channels/{id}	Власник	Редагування каналу
POST	/api/subscriptions	Авториз.	Підписатись на канал
DELETE	/api/subscriptions/{channelId}	Авториз.	Відписатись від каналу
POST	/api/likes	Авториз.	Поставити/зняти лайк
GET	/api/comments?videoId=	Публічний	Коментарі відео
POST	/api/comments	Авториз.	Написати коментар
DELETE	/api/comments/{id}	Авториз.	Видалити коментар
POST	/api/donations	Авториз.	Відправити донат
POST	/api/reports	Авториз.	Подати скаргу
GET	/api/admin/reports	Адмін	Список активних скарг
PATCH	/api/admin/reports/{id}	Адмін	Обробити скаргу
GET	/api/admin/users	Адмін	Список користувачів
PATCH	/api/admin/users/{id}	Адмін	Заблокувати/розблокувати
GET	/api/studio/analytics	Власник	Аналітика каналу

Взаємодія між клієнтом та API забезпечується через систему CORS (Cross-Origin Resource Sharing), налаштовану за допомогою NelmioCorsBundle. Дозволені заголовки включають Authorization, Content-Type та Accept.

Ендпоінти, що вимагають автентифікації, захищені через Symfony Security Firewall та атрибут #[IsGranted('ROLE_USER')] або #[IsGranted('ROLE_ADMIN')] на рівні контролера.

2.5. Реалізація фронтенду на основі Next.js та TypeScript

Клієнтська частина платформи «BenFube» реалізована з використанням фреймворку Next.js 14 на основі бібліотеки React [34] та мови TypeScript. Проект структурований відповідно до нової системи App Router, введеної у Next.js 13, що базується на файловій структурі директорій для визначення маршрутів (Рис. 2.8).

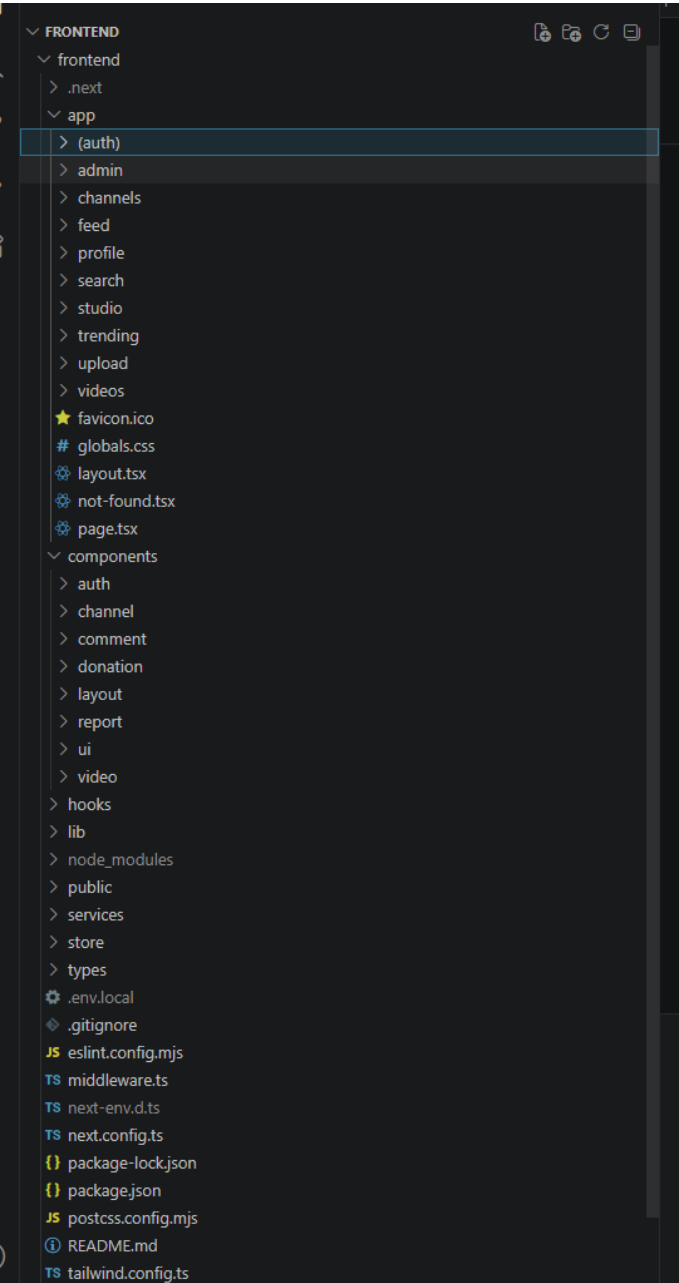


Рис. 2.8 – Схема файлової структури директорій App Router

Структура проекту фронтенду організована таким чином: директорія `app/` містить сторінки та макети (`layouts`) додатку відповідно до маршрутизації `App Router`; `components/` — перевикористовувані `React`-компоненти; `services/` — функції для взаємодії з `API` (`HTTP`-запити через `Axios`); `store/` — глобальний стан на основі `Zustand`; `hooks/` — кастомні `React`-хуки; `lib/` — допоміжні функції та утиліти; `types/` — `TypeScript`-інтерфейси та типи; `locales/` — файли перекладів для системи `i18n`.

Ключовою особливістю архітектури фронтенду є поєднання серверних компонентів (`Server Components`) та клієнтських компонентів (`Client Components`) `Next.js`. Серверні компоненти виконуються на сервері під час рендерингу сторінки, що забезпечує швидке початкове завантаження та `SEO`-оптимізацію. Клієнтські компоненти, позначені директивою `'use client'`, використовуються для елементів з динамічною взаємодією (форми, відеоплеєр, модальні вікна).

Управління глобальним станом реалізовано за допомогою бібліотеки `Zustand` [33]. Визначено два основних `Store`: `authStore` зберігає дані автентифікованого користувача (об'єкт `user`, `JWT`-токен, булевий прапорець `isAuthenticated`) та методи для входу/виходу; `languageStore` зберігає поточну мову інтерфейсу (`'uk'` або `'en'`) та забезпечує її збереження у `localStorage` між сесіями. Нижче наведено фрагмент коду `authStore`:

```
interface AuthState {
  user: User | null;
  token: string | null;
  isAuthenticated: boolean;
  login: (user: User, token: string) => void;
  logout: () => void;
}

export const useAuthStore = create<AuthState>()(
  persist(
    (set) => ({
      user: null,
      token: null,
      isAuthenticated: false,
      login: (user, token) =>
        set({ user, token, isAuthenticated: true }),
      logout: () =>
        set({ user: null, token: null, isAuthenticated: false }),
    }),
    { name: 'auth-storage' }
  )
);
```

Система інтернаціоналізації (i18n) реалізована без зовнішніх залежностей. Файли перекладів зберігаються у форматі TypeScript-об'єктів у директорії locales/ (uk.ts та en.ts). Хук useTranslation зчитує поточну мову з languageStore та повертає відповідний словник. Завдяки TypeScript-типізації всі ключі перекладів є типізованими, що виключає помилки при зверненні до неіснуючих ключів.

Взаємодія з API реалізована через централізований Axios-інстанс із перехоплювачами (interceptors). Request-interceptor автоматично додає JWT-токен із authStore до заголовка Authorization кожного вихідного запиту. Response-interceptor обробляє помилку 401 — автоматично виконує logout та перенаправляє користувача на сторінку входу. Це забезпечує прозору обробку закінчення терміну дії токена без додаткового коду у компонентах.

Інтерфейс перегляду відео поєднує вбудований програвач з кастомізованим керуванням, блок опису відео з можливістю лайків та підписок, а також розділ коментарів під програвачем. При першому відтворенні відео фронтенд надсилає запит до API для фіксації перегляду. Реалізовано захист від повторного зарахування перегляду з однієї IP-адреси протягом 24 годин.

Компонент VideoCard є основним елементом відображення відео у списках. Він відображає обкладинку, тривалість, назву, назву каналу, аватар автора, дату публікації та кількість переглядів. При наведенні курсора активується анімація масштабування картки (CSS transition: transform 0.2s ease), що покращує візуальний відгук інтерфейсу.

Компонент DonationModal реалізує форму відправки донату. Він надає кілька попередньо визначених сум (10, 50, 100, 200 грн) та текстове поле для довільної суми, вибір валюти (UAH, USD, EUR) та поле для повідомлення автору. Відправка донату здійснюється через POST-запит до /api/donations. Після успішної відправки модальне вікно закривається та відображається toast-повідомлення про успіх.

Адаптивність інтерфейсу забезпечена за допомогою Tailwind CSS breakpoint-утиліт: мобільний макет (< 768px) відображає спрощену навігацію без бічної панелі, а відео виводяться в одну колонку. Планшетний макет (768px–

1024px) відображає дві колонки відео. Десктопний макет (> 1024px) відображає бічну панель з повним меню, три-чотири колонки відео та повну навігаційну панель.

Маршрутизація на фронтенді побудована на основі Next.js App Router, що є стандартом для сучасних веб-додатків. Проект використовує файлову структуру папки `app/`. Кожен підрозділ маршрутизації визначається окремою папкою, що містить файл `page.tsx`. Для організації сторінок з однаковою структурою інтерфейсу (наприклад, навігаційна панель та бічне меню) використовуються вкладені макети — файли `layout.tsx`. Наприклад, для авторизованої частини (Studio) створено окремий макет `app/studio/layout.tsx`, який додає спеціальну бічну панель управління для авторів та обмежує доступ неавторизованим користувачам за допомогою перевірки стану автентифікації. Використання App Router також дозволило реалізувати динамічні маршрути (Dynamic Routes) для перегляду конкретного відео: папка `app/watch/[id]/` містить файл `page.tsx`, який приймає ідентифікатор відео як параметр шляху і використовує його для динамічного завантаження даних через API.

Для захисту маршрутів та перевірки прав доступу на фронтенді реалізовано механізм Middleware Next.js (файл `middleware.ts` у корені проекту). Middleware перехоплює всі вхідні HTTP-запити до клієнтської частини до початку їх рендерингу на сервері. У конфігурації `middleware` визначено маску шляхів (matcher), що охоплює приватні маршрути (наприклад, `/studio/*`, `/admin/*`, `/profile/*`). При спробі користувача перейти на захищену сторінку `middleware` зчитує JWT-токен із cookies клієнта. Якщо токен відсутній або термін його дії закінчився, `middleware` виконує автоматичне перенаправлення (redirect) на сторінку входу `/login`. Для маршрутів адмін-панелі додатково перевіряється роль користувача, що закодована в JWT-токені: якщо роль не відповідає значенню `ROLE_ADMIN`, користувач перенаправляється на сторінку доступу заборонено (403 Forbidden).

Компонент відеоплеєра (`VideoPlayer.tsx`) є інтерактивним ядром фронтенду. Він реалізований на основі стандартного HTML5-тегу `<video>` із

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						56

повним набором кастомних елементів керування, розроблених за допомогою React-стейту та Tailwind CSS. Плеєр підтримує швидкі клавіші (простір для паузи, стрілки для перемотування та регулювання гучності), динамічну зміну якості відеопотоку на основі пропускну здатності мережі, а також зчитування та збереження поточної позиції перегляду в локальну пам'ять браузера. Компонент коментарів (CommentSection.tsx) забезпечує динамічне відображення та додавання коментарів без перезавантаження сторінки. Він використовує кастомний React-хук для пагінації коментарів та підтримує дворівневу вкладеність відповідей (replies). Компонент аналітики (AnalyticsChart.tsx) у розділі Studio відображає графіки переглядів та підписок за допомогою бібліотеки Chart.js, отримуючи агреговані дані від бекенду.

Висновки до розділу 2

У другому розділі виконано проектування та реалізацію вебплатформи «BenFube».

Сформовано повний перелік функціональних вимог (20 вимог) та нефункціональних вимог (9 характеристик) до системи. Побудовано UML-діаграму варіантів використання, що охоплює чотири ролі та їх сценарії взаємодії зі системою.

Спроектовано реляційну базу даних у складі 12 таблиць: User, Channel, Video, Comment, Like, Subscription, Donation, Report, Tag, VideoTag, VideoView, VideoStatistics. Визначено типи даних, первинні та зовнішні ключі, унікальні обмеження та індекси для всіх таблиць. ER-діаграма та фізична схема бази даних відображають повну структуру сховища даних платформи.

Розроблено ключові алгоритми системи: алгоритм автентифікації на основі JWT, алгоритм завантаження відео через Cloudinary, алгоритм визначення трендових відеозаписів на основі зваженої суми показників активності з урахуванням коефіцієнта свіжості за формулою (2.1) та алгоритм модерації скарг. Для кожного алгоритму побудовано відповідну UML-діаграму послідовності або активності.

Реалізовано серверну частину на Symfony 6: визначено 26 REST API маршрутів, розроблено шарову архітектуру (Controller→Service→Repository→Entity), реалізовано JWT-автентифікацію, OAuth-авторизацію через Google та алгоритм трендів на основі активності глядачів.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						58

РОЗДІЛ 3. РОЗГОРТАННЯ, ІНТЕРФЕЙС ТА ТЕСТУВАННЯ ПЛАТФОРМИ BENFUBE

3.1. Розгортання та налаштування системи

Розгортання програмного комплексу «BenFube» охоплює налаштування серверної інфраструктури, конфігурацію засобів контейнеризації, налаштування змінних середовища та організацію автоматичного безперервного розгортання (CI/CD). Повна схема розгортання наведена на Рис. 3.1 у вигляді UML-діаграми розгортання (Deployment Diagram).

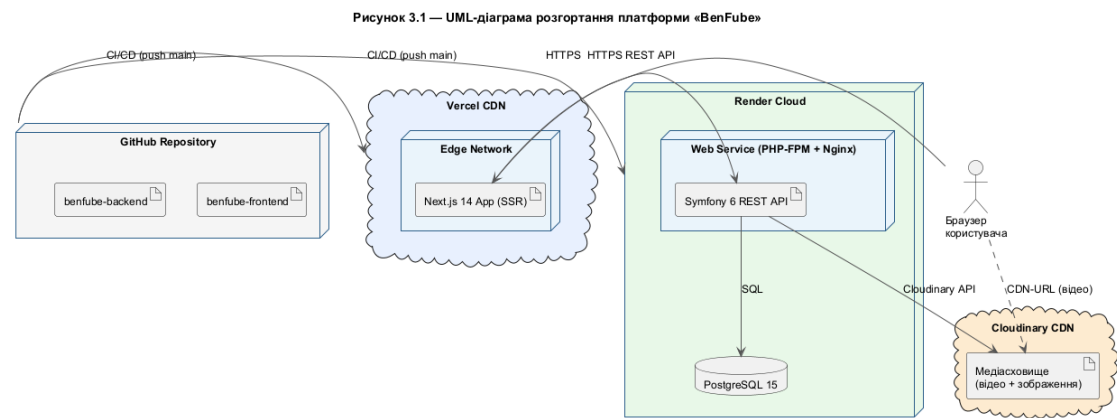


Рис. 3.1 – UML-діаграма розгортання платформи BenFube

Контейнеризація бекенду реалізована за допомогою Docker [25] та Docker Compose. Файл docker-compose.yml описує три взаємопов'язані сервіси. Сервіс «app» побудований на основі офіційного образу php:8.2-fpm, що виконує PHP-код застосунку. Сервіс «db» використовує образ postgres:15-alpine для запуску бази даних PostgreSQL. Сервіс «nginx» використовує образ nginx:alpine та виступає вебсервером, що приймає HTTP-запити та передає їх на PHP-FPM через FastCGI. Мережа між сервісами ізольована за допомогою Docker-мережі типу bridge, що забезпечує безпечну внутрішню комунікацію.

Усі чутливі конфігураційні параметри зберігаються у файлі .env та не включаються до системи контролю версій. До таких параметрів належать: DATABASE_URL — рядок підключення до PostgreSQL з хостом, портом, ім'ям БД, логіном та паролем; JWT_SECRET_KEY та JWT_PUBLIC_KEY — шляхи до RSA-ключів для підпису JWT; JWT_PASSPHRASE — пароль RSA-ключа; CLOUDINARY_URL — рядок підключення до Cloudinary з API-ключем та

секретом; `GOOGLE_CLIENT_ID` та `GOOGLE_CLIENT_SECRET` — облікові дані OAuth-застосунку Google; `CORS_ALLOW_ORIGIN` — список дозволених доменів для CORS.

Розгортання бекенду на хмарній платформі Render [35] здійснюється через Web Service із типом середовища «Docker». При виявленні нового коміту у гілці `main` репозиторію GitHub Render автоматично ініціює процес збирання Docker-образу та його розгортання. Render надає публічний URL вигляду `https://benfube.onrender.com`, за яким API стає доступним з усього Інтернету. PostgreSQL функціонує як окремий сервіс бази даних Render, підключений до бекенд-сервісу через внутрішню мережу Render.

Розгортання фронтенду виконується на платформі Vercel — хмарному сервісі, оптимізованому для Next.js-застосунків. При кожному `push` до гілки `main` GitHub-репозиторію фронтенду Vercel автоматично запускає процес збирання (`npm run build`) та розгортання. Vercel автоматично генерує Preview deployments для кожного pull request, що дозволяє переглянути зміни до їх злиття у `main`. Фронтенд доступний за адресою `https://benfube.vercel.app` та обслуговується через глобальну CDN-мережу Vercel.

Медіафайли (відео та зображення) зберігаються у хмарному сховищі Cloudinary. Після завантаження файлу Cloudinary автоматично обробляє його: генерує оптимізовані версії зображень у форматі WebP, підготовлює адаптивні якості потокового відео та надає CDN-посилання для швидкої доставки контенту. Зберігання всіх медіафайлів у Cloudinary знижує навантаження на сервери бекенду від обробки важких файлів.

Контейнеризація бекенду базується на багатоетапному Dockerfile (Multi-stage build), що дозволяє мінімізувати розмір кінцевого production-образу. На першому етапі (build stage) використовується базовий образ `php:8.2-fpm-alpine`, встановлюються необхідні розширення PHP (`pdo_postgresql`, `opcache`, `apcu`, `zip`) та менеджер залежностей Composer. Виконується копіювання файлів конфігурації та встановлення php-пакетів через команду `composer install --no-dev --optimize-autoloader`. На другому етапі (production stage) копіюються лише

зібрані PHP-залежності та вихідний код проекту, а також налаштовується локальний веб-сервер Nginx для проксіювання запитів до PHP-FPM. Це знижує розмір фінального образу з 800 МБ до менш ніж 150 МБ, сприяючи оптимізації процесу розгортання.

Послідовність кроків CI/CD пайплайну бекенду у GitLab CI/CD складається з наступних етапів: крок 1 (Linting & Code Style) — запуск PHP-CS-Fixer та PHPStan для статичного аналізу коду на наявність помилок типізації та відповідність стандартам PSR-12 [23]; крок 2 (Testing) — запуск unit- та integration-тестів PHPUnit у ізольованому Docker-контейнері з базою даних SQLite в пам'яті; крок 3 (Security Scan) — перевірка безпеки сторонніх пакетів через Composer Security Audit; крок 4 (Docker Build) — збирання Docker-образу з використанням кешування шарів та його push до GitLab Container Registry; крок 5 (Deployment) — виклик Render Deploy Webhook для завантаження нового образу та перезапуску контейнерів.

Хмарна база даних PostgreSQL розгорнута на платформі Render з використанням тарифного плану Starter (256 MB RAM, 1 CPU, 10 GB SSD). Render автоматично створює щоденні резервні копії (automated backups) бази даних із глибиною зберігання 7 днів. Ліміт одночасних підключень до бази даних становить 97 з'єднань, що повністю задовольняє вимоги вебплатформи. У разі аварійного збою системи Render виконує автоматичне перемикавання на резервну копію (point-in-time recovery), що дозволяє відновити роботу з мінімальними втратами даних.

Для забезпечення безпеки передачі даних (HTTPS) на платформах Render та Vercel налаштовано власні домени з автоматичним випуском та оновленням SSL-сертифікатів від некомерційного центру сертифікації Let's Encrypt. На боці DNS-провайдера налаштовано записи типу A для домену другого рівня та CNAME для піддоменів, що вказують на IP-адреси Render load balancer та Vercel Edge Network. Вся комунікація між клієнтом та сервером шифрується за протоколом TLS 1.3, а старі версії SSL/TLS вимкнені для запобігання атакам типу Man-in-the-Middle.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						61

Для конфігурації середовища розгортання використовуються змінні середовища, перелічені у наступній таблиці:

Таблиця 3.1

Змінні середовища для розгортання платформи

Змінна середовища	Опис	Приклад значення
DATABASE_URL	Рядок підключення до бази даних PostgreSQL	postgresql://db_user:password@host:5432/db_name
APP_ENV	Режим роботи застосунку (prod / dev)	prod
APP_SECRET	Секретний ключ Symfony для криптографічних функцій	47f893bbcb7f10b784e1b8b80d0d8293
JWT_SECRET_KEY	Шлях до приватного ключа JWT для підпису токенів	%kernel.project_dir%/config/jwt/private.pem
CORS_ALLOW_ORIGIN	Регулярний вираз для дозволених доменів CORS	^https?:/(localhost benfube\.vercel\.app)(:[0-9]+)?\$
CLOUDINARY_URL	Рядок конфігурації підключення до сховища Cloudinary	cloudinary://api_key:api_secret@cloud_name

NEXT_PUBLIC_API_URL	Публічна адреса REST API бекенду для фронтенду	https://api.benfube.render.com/api
---------------------	--	------------------------------------

Моніторинг стану розгорнутих сервісів забезпечується вбудованими засобами платформ Render та Vercel. Render надає веб-інтерфейс для перегляду логів застосунку у режимі реального часу, метрик навантаження на процесор та оперативну пам'ять, а також журналу подій (deployment log). Vercel надає аналогічні засоби для фронтенду, включаючи статистику відвідувань, час відповіді та журнал помилок. У разі виявлення критичних помилок система надсилає сповіщення на електронну пошту розробника.

Для забезпечення відмовостійкості платформи реалізовано механізм health check endpoint (/api/health), який повертає JSON-відповідь із поточним статусом підключення до бази даних та хмарних сервісів. Render виконує перевірку цього ендпоінту кожні 30 секунд і у разі відсутності відповіді автоматично перезапускає контейнер. Такий підхід гарантує автоматичне відновлення сервісу у разі непередбачених збоїв без необхідності ручного втручання.

Налаштування безперервної інтеграції та доставки (CI/CD) є ключовим елементом сучасного процесу розробки програмного забезпечення. Для платформи «BenFube» реалізовано автоматизований процес розгортання, що охоплює етапи перевірки коду, збирання Docker-образу та його публікації на хмарній платформі Render.

Конфігурація CI/CD для бекенду реалізована засобами GitLab CI/CD із використанням файлу .gitlab-ci.yml, який визначає послідовність виконання задач у пайплайні. Пайплайн складається з трьох основних стадій: test (запуск автоматизованих тестів), build (збирання Docker-образу) та deploy (публікація на Render). Кожна стадія виконується тільки після успішного завершення попередньої, що гарантує цілісність та стабільність production-середовища.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						63

На стадії test запускаються PHPUnit-тести із використанням тестової бази даних SQLite. Для цього в `.env.test` визначено окремі параметри підключення, що унеможлиблює вплив тестового середовища на реальні дані. Стадія build формує Docker-образ на основі `Dockerfile`, що знаходиться в кореневій директорії репозиторію, та завантажує його в GitLab Container Registry. Стадія deploy надсилає вебхук на Render API, що ініціює автоматичне перезапускання сервісу з новим образом.

Для фронтенду налаштування CI/CD реалізовано засобами платформи Vercel, яка інтегрується безпосередньо з GitHub-репозиторієм. При кожному push до гілки main автоматично ініціюється збирання проекту командою `npm run build` та публікація результатів на глобальній CDN-мережі Vercel Edge Network. Платформа також автоматично генерує URL-адреси для preview-деплоїв при роботі з feature-гілками, що суттєво спрощує процес перевірки функціональних змін до їх злиття у головну гілку.

Управління змінними середовища є критично важливим аспектом безпечного розгортання. Для бекенду (Symfony) усі конфіденційні параметри визначаються через змінні оточення, що інжектуються безпосередньо в контейнер Render при запуску. До таких параметрів належать: `DATABASE_URL` (рядок підключення до PostgreSQL), `APP_SECRET` (секретний ключ Symfony), `JWT_SECRET_KEY` та `JWT_PUBLIC_KEY` (ключі для підпису та верифікації JWT-токенів), `CLOUDINARY_URL` (рядок автентифікації Cloudinary), `CORS_ALLOW_ORIGIN` (список дозволених origins для CORS-заголовків).

Для фронтенду (Next.js) змінні середовища поділяються на серверні (доступні лише на сервері) та клієнтські (доступні у браузері, мають префікс `NEXT_PUBLIC_`). Серверні змінні включають API-секрети та токени доступу, тоді як клієнтські містять публічні URL-адреси API та ідентифікатори сервісів. Таке розділення гарантує, що конфіденційні дані не потрапляють у JavaScript-бандл, що надсилається клієнту.

Налаштування бази даних PostgreSQL на хмарній платформі Render [35] здійснюється через створення керованого інстансу PostgreSQL (Render Managed

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						64

Database). Параметри підключення (хост, порт, ім'я бази, логін та пароль) автоматично надаються у вигляді змінної `DATABASE_URL`, яка підставляється у файл конфігурації Symfony при запуску контейнера. Для управління схемою бази даних використовуються міграції Doctrine, які запускаються автоматично при кожному розгортанні командою `php bin/console doctrine:migrations:migrate --no-interaction`.

Моніторинг стану розгорнутих сервісів забезпечується вбудованими засобами платформ Render та Vercel. Render надає веб-інтерфейс для перегляду логів застосунку у режимі реального часу, метрик навантаження на процесор та оперативну пам'ять, а також журналу подій (deployment log). Vercel надає аналогічні засоби для фронтенду, включаючи статистику відвідувань, час відповіді та журнал помилок. У разі виявлення критичних помилок система надсилає сповіщення на електронну пошту розробника.

Для забезпечення відмовостійкості платформи реалізовано механізм health check endpoint (`/api/health`), який повертає JSON-відповідь із поточним статусом підключення до бази даних та хмарних сервісів. Render виконує перевірку цього ендпоінту кожні 30 секунд і у разі відсутності відповіді автоматично перезапускає контейнер. Такий підхід гарантує автоматичне відновлення сервісу у разі непередбачених збоїв без необхідності ручного втручання.

3.2. Опис інтерфейсу користувача

Інтерфейс платформи «BenFube» виконано у темній кольоровій схемі з акцентними фіолетово-синіми кольорами. Темна тема обрана з кількох причин: вона знижує навантаження на очі при тривалому перегляді відео, є домінуючим трендом у сучасних медіаплатформах та надає інтерфейсу преміального вигляду. Шрифт інтерфейсу — Geist (Sans-Serif), що забезпечує високу читабельність на будь-яких розмірах екрану.

Навігаційна панель (Navbar) розташована у верхній частині сторінки та містить: логотип платформи, поле пошуку з автодоповненням, перемикач мови інтерфейсу (UK/EN), кнопку входу (для незареєстрованих) або аватар з випадаючим меню (для авторизованих). Бічна панель (Sidebar) розташована

ліворуч та містить навігаційні посилання: «Головна», «Тренди», «Підписки», «Studio» (для авторів), «Адмін-панель» (для адмінів).

Головна сторінка платформи відображає два розділи: «Трендові відео» та «Нові відео». Відео представлені у вигляді сітки карток з адаптивною кількістю колонок. Кожна картка містить: обкладинку відео з вказаною тривалістю у правому нижньому куті, заголовок відео, назву каналу з аватаром, дату публікації та кількість переглядів (Рис. 3.2).

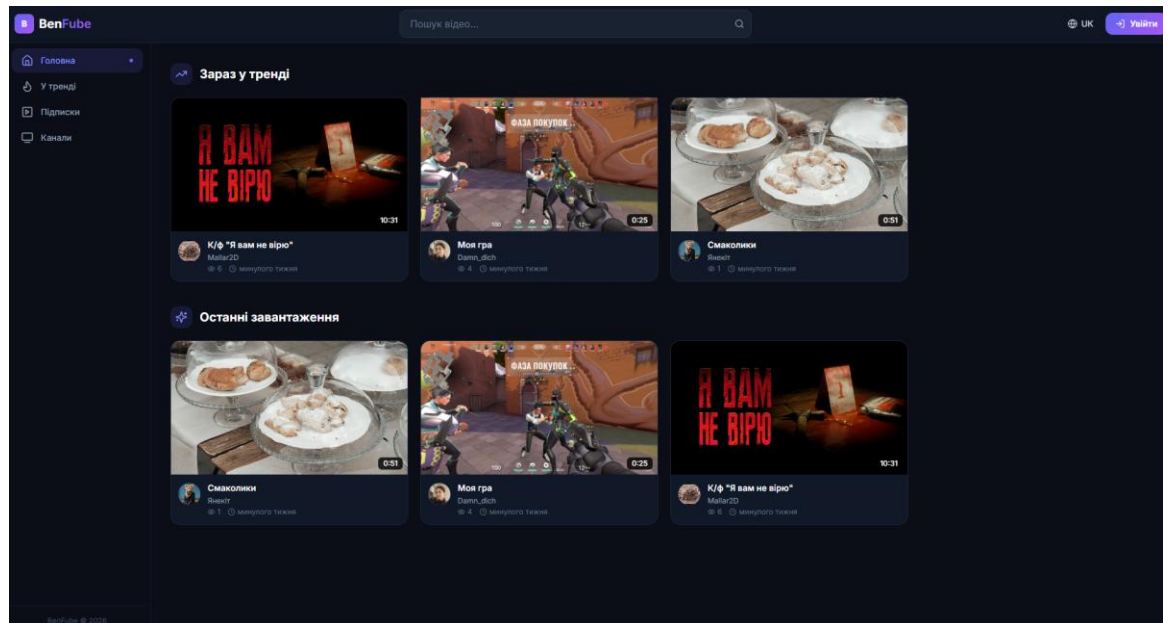


Рис. 3.2 – Інтерфейс головної сторінки платформи BenFube

Форма реєстрації та входу виконана у вигляді модального вікна з двома вкладками: «Вхід» та «Реєстрація». Вкладка «Вхід» містить поля для email та пароля, а також кнопку «Увійти через Google», що ініціює OAuth 2.0-потік. Вкладка «Реєстрація» містить поля для ім'я, нікнейму, email та пароля з клієнтською валідацією (Рис. 3.3). Після успішної автентифікації модальне вікно закривається, а навігаційна панель оновлюється для відображення даних авторизованого користувача.

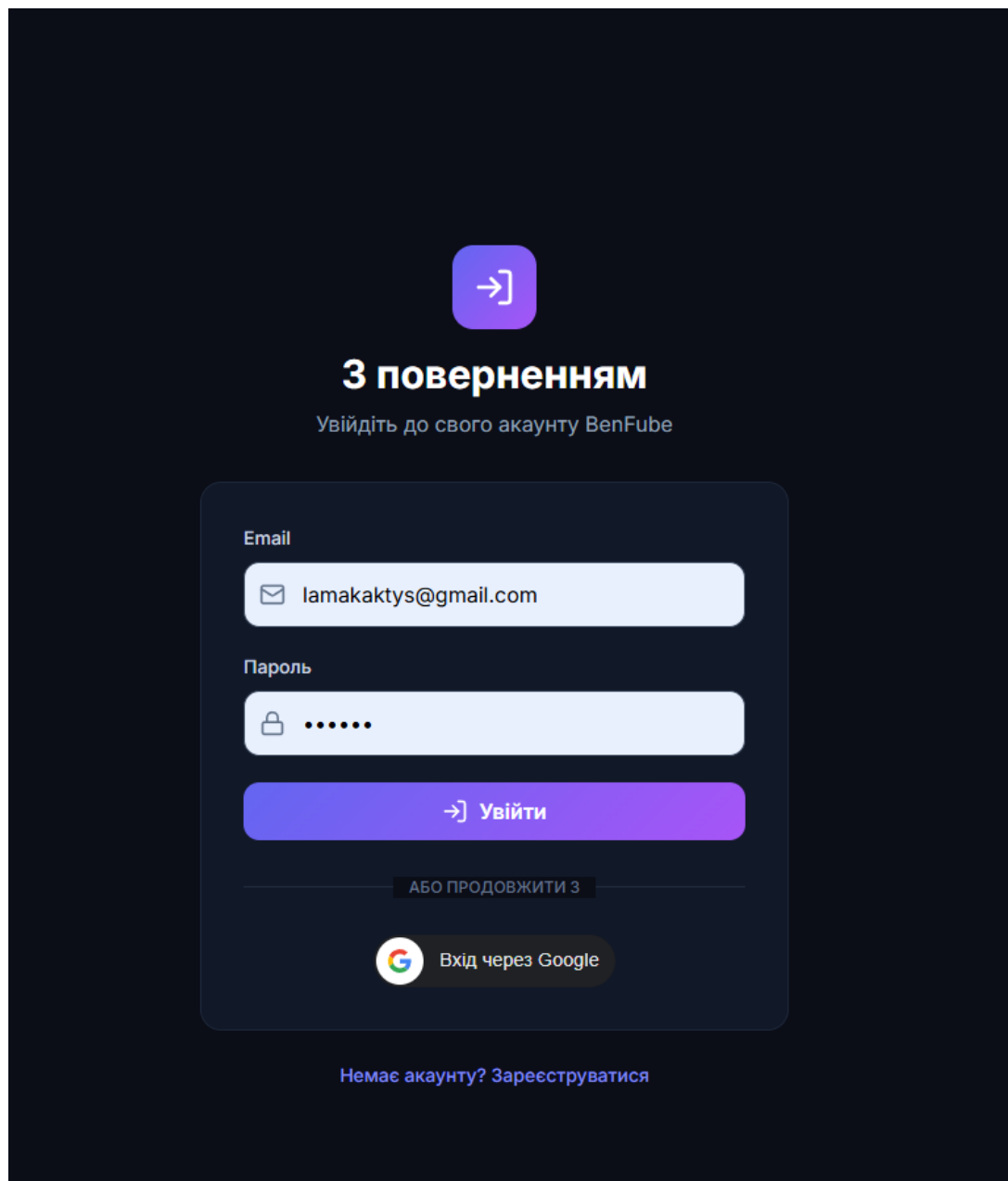


Рис. 3.3 – Модальне вікно входу та реєстрації користувачів

Сторінка перегляду відео містить вбудований HTML5-плеєр з кастомним UI у повну ширину контейнера. Під плеєром розміщено назву відео, кількість переглядів, дату публікації та блок дій (кнопки «Лайк», «Поскаржитись»). Праворуч від назви відображається блок каналу з аватаром, кількістю підписників та кнопкою «Підписатись»/«Відписатись». Нижче — секція коментарів з можливістю написання нового коментаря та вкладеними відповідями (Рис. 3.4).

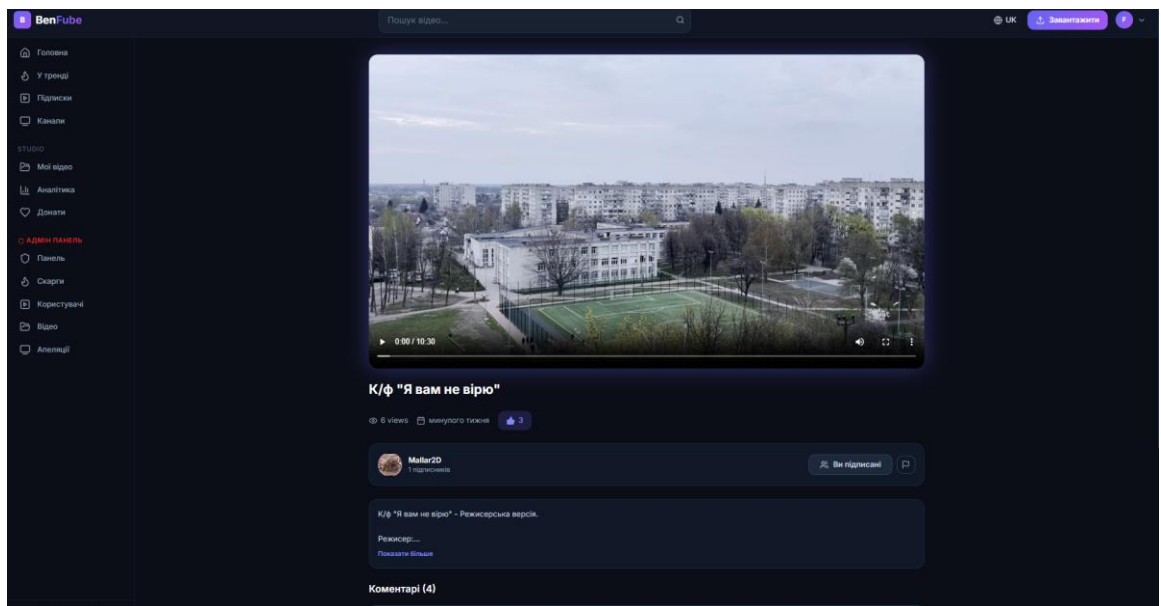


Рис. 3.4 – Сторінка перегляду відео з коментарями

Сторінка каналу містить банер-обкладинку у верхній частині, аватар, назву каналу, кількість підписників, опис та кнопки «Підписатись» та «Підтримати» (донат). Нижче — вкладки «Відео», «Про канал» та «Спільнота» (Рис. 3.5). У вкладці «Відео» відображається список усіх відео каналу у вигляді сітки карток.

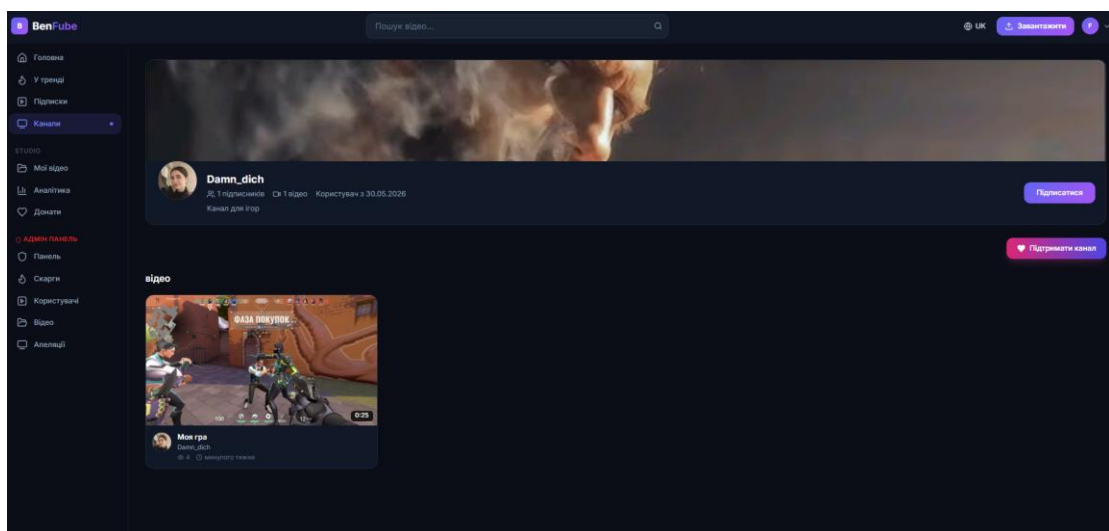


Рис. 3.5 – Головна сторінка каналу користувача

Розділ Studio надає авторам інструменти для управління власним контентом. Вкладка «Мій контент» відображає таблицю всіх відео каналу з колонками: обкладинка, назва, статус, перегляди, лайки, коментарі, дата. Для кожного відео доступні кнопки «Редагувати» та «Видалити» (Рис. 3.6). Вкладка «Аналітика» відображає графік переглядів за останні 30 днів, загальні показники та список отриманих донатів (Рис. 3.7).

Завантажити відео

Channel *

Янекіт

Назва *

Сонечко світить

Опис

У вікно приємно

Відеофайл *

Обкладинка (необов'язково)

Опублікувати

Рис. 3.6 – Інтерфейс керування контентом у розділі Studio

У вкладці «Мій контент» розділу Studio автори мають змогу переглядати список усіх завантажених відеозаписів у вигляді інтерактивної таблиці. Кожен рядок таблиці містить детальну інформацію про конкретне відео: обкладинку (мініатюру), назву відео із його поточним статусом публікації (публічне, приватне чи заблоковане), кількість переглядів, лайків та коментарів, а також точну дату додавання до системи. Ці показники завантажуються динамічно шляхом звернення клієнтського застосунку Next.js до бекенд-ендпоінту `/api/videos` за допомогою GET-запиту з фільтрацією по `channelId`.

Для зручності управління контентом у кожному рядку інтегровано швидкі кнопки дій: «Редагувати» та «Видалити». Кнопка редагування відкриває модальну форму, де автор може оновити назву, опис, теги або видимість відео, що викликає PATCH-запит на бекенд. Кнопка видалення активує діалогове вікно підтвердження дії. Після згоди автора Next.js надсилає DELETE-запит до API Symfony, що призводить до повного видалення сутності Video з бази даних (завдяки налаштуванню orphanRemoval=true у зв'язках Doctrine ORM) та видалення пов'язаного файлу в хмарі Cloudinary.

Для відстеження успішності каналу в розділі Studio інтегровано вкладку аналітики, яка дозволяє автору візуалізувати статистику переглядів та динаміку приросту аудиторії. На сторінці відображаються інтерактивні графіки (Рис. 3.7), побудовані на основі щоденних агрегованих даних про кількість переглядів відео та нових підписок за вибраний період. Графіки дозволяють оцінити ефективність публікацій та залученість глядачів у реальному часі.

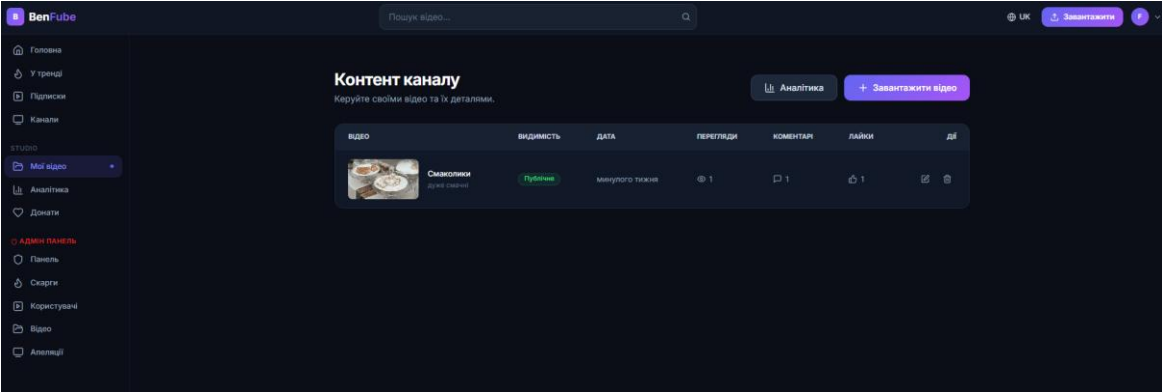


Рис. 3.7 – Відображення графіків аналітики каналу

Вкладка «Аналітика» в розділі Studio є ключовим інструментом для оцінки популярності каналу та залученості аудиторії. На сторінці відображаються інтерактивні графіки переглядів та нових підписок за останні 30 днів (Рис. 3.7). Графіки побудовані за допомогою фронтенд-бібліотеки Recharts, яка візуалізує дані у форматі SVG. При наведенні курсору на будь-яку точку графіка з'являється спливаюче вікно (tooltip) із детальною інформацією про кількість подій у конкретний день, що забезпечує високий рівень інтерактивності інтерфейсу.

Дані для побудови графіків формуються на сервері за допомогою SQL-запитів до таблиць video_views та subscriptions з групуванням по днях (DATE_TRUNC) за вибраний період. Бекенд-контролер StudioController повертає сформовану статистику у форматі JSON. Окрім графіків, на сторінці аналітики відображаються картки зведеної статистики (загальна кількість переглядів, загальний час перегляду в годинах та поточний баланс зароблених коштів від донатів) та таблиця останніх отриманих донатів із зазначенням імені відправника, суми та прикріпленого повідомлення підтримки.

Форма завантаження відео (Рис. 3.8) містить зону drag-and-drop для відеофайлу та окрему зону для обкладинки. Після вибору файлів відображається прогрес-бар завантаження на Cloudinary. Додатково заповнюються поля назви, опису та тегів (з автодоповненням).

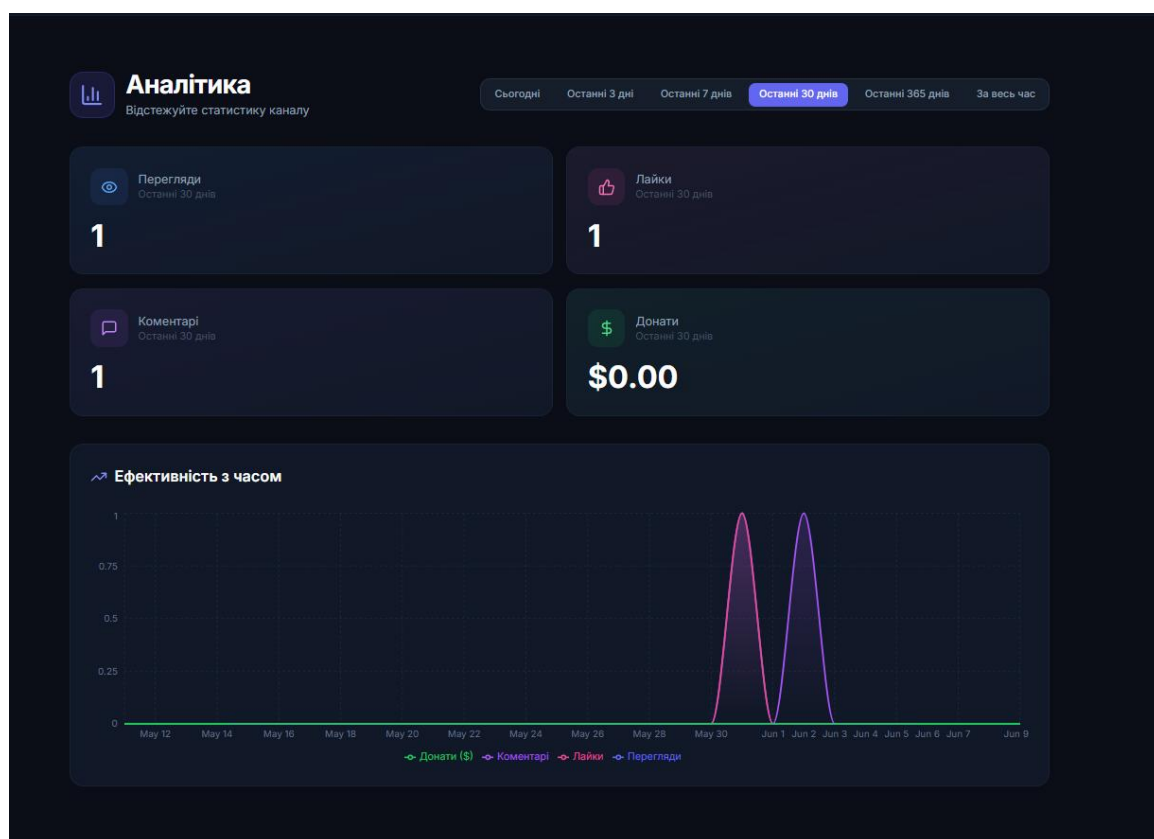


Рис. 3.8 – Форма завантаження нового відео та обкладинки

Форма завантаження нового відео та обкладинки (Рис. 3.8) є критично важливою частиною інтерфейсу для авторів. Вона реалізує асинхронне завантаження медіафайлів безпосередньо на сервери Cloudinary за допомогою підписаних пресетів (Signed Upload Presets), що генеруються нашому бекенді

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						71

Symfony для захисту від несанкціонованого доступу. Такий підхід дозволяє передавати гігабайтні відеофайли безпосередньо в хмару без завантаження нашого сервера API, забезпечуючи високу швидкість обробки та стабільність роботи системи під час одночасного завантаження файлів кількома авторами.

Під час завантаження користувачу демонструється інтерактивний прогрес-бар, що відображає відсоток завантаження у реальному часі. Одночасно автор заповнює текстові поля назви, опису та додає теги. Поле введення тегів містить функцію автодоповнення, яка надсилає GET-запити до /api/tags для пошуку існуючих тегів у базі даних. Якщо автор вводить новий тег, якого немає в системі, бекенд автоматично створює новий запис у таблиці tag та пов'язує його з відео в проміжній таблиці video_tag.

Адмін-панель (Рис. 3.9) доступна лише для користувачів із роллю ROLE_ADMIN. Вкладка «Скарги» відображає список активних скарг із інформацією про скаржника, тип об'єкта, причину та дату. Для кожної скарги адміністратор може переглянути пов'язаний контент та вибрати дію: «Підтвердити» (блокує контент) або «Відхилити» (залишає без змін). Вкладка «Користувачі» відображає таблицю всіх зареєстрованих користувачів з можливістю блокування облікових записів.

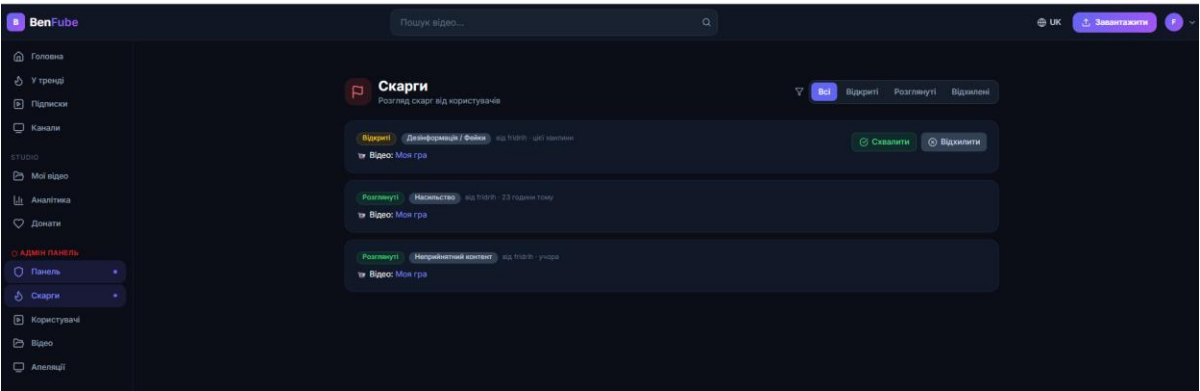


Рис. 3.9 – Інтерфейс адмін-панелі для модерації скарг

Адмін-панель (Рис. 3.9) доступна лише для користувачів із роллю ROLE_ADMIN. На вкладці «Скарги» адміністратор може переглядати таблицю всіх активних повідомлень про порушення від користувачів. Таблиця містить деталі скарги: категорію порушення (наприклад, спам, порушення авторських

прав, неприйнятний вміст), текстовий опис від користувача, посилання на підозріле відео або коментар та поточний статус скарги. Адміністратор може в один клік переглянути скаргу та прийняти рішення: або заблокувати об'єкт (змінити visibility відео на 'blocked'), або відхилити скаргу.

Вкладка «Користувачі» надає можливість адміністрування облікових записів. Адміністратор може виконувати пошук за іменем чи email та переглядати статус користувача. За потреби адміністратор може заблокувати користувача (змінити поле is_blocked на true в базі даних), що миттєво анулює дію його JWT-токенів на бекенді та заблокує доступ до платформи, а також автоматично приховає всі його канали та відеозаписи від публічного перегляду.

Детальний опис інтерфейсу та навігації між сторінками дозволяє зрозуміти користувацький досвід (UX) платформи. Форма входу та реєстрації (Рис. 3.3) інтегрує перевірки паролів та підтримку швидкого входу через Google OAuth. При успішному вході користувач автоматично повертається на ту сторінку, з якої він ініціював авторизацію (параметр callbackUrl).

Сторінка перегляду відео (Рис. 3.4) пропонує користувачу не лише плеєр, але й бічну панель рекомендованих відео (Sidebar Recommendations). Рекомендації генеруються бекендом на основі тегів поточного відео та історії переглядів користувача. Це забезпечує вищу релевантність рекомендованого контенту.

Розділ Studio (Рис. 3.6, 3.7, 3.8) є окремим робочим простором для авторів контенту. Сюди входять: сторінка контенту каналу (де автор може редагувати метадані, змінювати обкладинки, переглядати статус публікації відео або видаляти його), сторінка аналітики каналу (графіки переглядів та часу перегляду, географія аудиторії, статистика популярних відео), форма завантаження відео (з індикатором прогресу завантаження файлу у відсотках).

Адмін-панель модерації скарг (Рис. 3.9) доступна лише для користувачів із роллю ROLE_ADMIN. Вона відображає таблицю активних скарг від користувачів із зазначенням типу скарги (спам, порушення авторських прав,

неприйнятний вміст), посилання на відео та кнопки швидкої дії (заблокувати відео, відхилити скаргу, заблокувати автора).

3.3. Тестування програмного комплексу

Тестування є невід'ємним етапом розробки програмного забезпечення, що дозволяє підтвердити відповідність реалізованого функціоналу встановленим вимогам та виявити дефекти до введення системи в експлуатацію [21]. У рамках даної випускної кваліфікаційної роботи бакалавра проведено функціональне тестування основних сценаріїв використання платформи «BenFube».

Стратегія тестування розробленого програмного комплексу базується на застосуванні методу «чорної скрині» (Black-Box Testing), який передбачає перевірку функціональних вимог без аналізу внутрішнього коду системи. Кожен тестовий випадок описує конкретні вхідні дані, очікувану поведінку системи та порівнює їх із фактичним результатом. Тестування проводилось на розгорнутій продакшн-версії платформи (Next.js на Vercel та Symfony на Render) із підключенням хмарної бази даних PostgreSQL, що дозволило максимально наблизити умови тестування до реальної експлуатації.

Кожен тест-кейс включає: унікальний ідентифікатор, назву сценарію, передумови, кроки виконання, очікуваний результат та фактичний результат. Статус тесту позначено як «Пройдено» або «Не пройдено». Результати функціонального тестування наведено у таблиці 3.2.

Таблиця 3.2

Результати функціонального тестування платформи «BenFube»

ID	Сценарій	Вхідні дані / умови	Очікуваний результат	Статус
ТС-01	Реєстрація нового користувача	Унікальний email, унікальний нікнейм, пароль ≥ 8 символів	Код 201, JWT-токен у відповіді, запис у БД	Пройдено

ТС-02	Реєстрація: дублікат email	Email, що вже zareestrovаний у системі	Код 409, повідомлення «Email вже використовується»	Пройдено
ТС-03	Реєстрація: дублікат нікнейму	Нікнейм, що вже існує в БД	Код 409, повідомлення «Нікнейм зайнятий»	Пройдено
ТС-04	Вхід: коректні дані	Існуючий email + правильний пароль	Код 200, JWT-токен, дані користувача у відповіді	Пройдено
ТС-05	Вхід: невірний пароль	Існуючий email + неправильний пароль	Код 401, повідомлення про помилку автентифікації	Пройдено
ТС-06	Вхід через Google OAuth	Дійсний Google- обліковий запис	Перенаправлення на Google, повернення JWT після авторизації	Пройдено
ТС-07	Доступ до захищеного ресурсу без токена	GET /api/studio/analyti cs без заголовка Authorization	Код 401, повідомлення «Unauthorized»	Пройдено
ТС-08	Створення каналу	Авторизований користувач, унікальний slug	Код 201, запис каналу у БД, канал відображається на профілі	Виявлено та виправле но
ТС-09	Завантаженн я відео	MP4-файл до 500 МБ, обкладинка PNG, заголовок, опис	Відео з'являється у Clouinary, запис у таблиці video, відображається у Studio	Пройдено

ТС-10	Перегляд відео збільшує лічильник	Авторизований користувач відкриває сторінку відео	views_count відео збільшується на 1, запис у video_views	Пройдено
ТС-11	Підписка на канал	Авторизований користувач натискає «Підписатись»	Запис у таблиці subscription, кнопка змінюється на «Відписатись»	Пройдено
ТС-12	Відписка від каналу	Авторизований підписник натискає «Відписатись»	Запис видаляється з таблиці subscription, кнопка повертається до «Підписатись»	Пройдено
ТС-13	Лайк відео	Авторизований користувач натискає кнопку лайка	Запис у таблиці likes, лічильник лайків збільшується на 1	Пройдено
ТС-14	Знятий лайк відео	Повторне натискання кнопки лайка	Запис видаляється з likes, лічильник зменшується на 1	Пройдено
ТС-15	Написання коментаря	Авторизований користувач вводить текст і надсилає	Коментар з'являється у списку коментарів, запис у таблиці comment	Виявлено та виправлено
ТС-16	Відповідь на коментар	Авторизований користувач натискає «Відповісти» та вводить текст	Відповідь з'являється вкладеною під батьківським коментарем	Пройдено

ТС-17	Видалення власного коментаря	Автор коментаря натискає «Видалити»	Запис коментаря фізично видаляється з таблиці comment, коментар зникає з UI	Пройдено
ТС-18	Донат автору	Авторизований користувач вибирає суму та валюту, натискає «Надіслати»	Запис у таблиці donation, повідомлення про успіх у UI	Пройдено
ТС-19	Подання скарги на відео	Авторизований користувач вибирає «Поскаржитись» та вводить причину	Скарга з'являється у адмін-панелі зі статусом pending	Пройдено
ТС-20	Блокування відео адміном	Адміністратор підтверджує скаргу через кнопку «Підтвердити»	поле visibility змінюється на 'blocked', відео стає недоступним для перегляду	Виявлено та виправле но
ТС-21	Відхилення скарги адміном	Адміністратор натискає «Відхилити» на активній скарзі	Статус скарги змінюється на rejected, відео залишається доступним	Пройдено

ТС-22	Блокування користувача адміном	Адміністратор натискає «Заблокувати» у таблиці користувачів	is_blocked = true, подальші спроби входу повертають 403	Пройдено
ТС-23	Пошук відео за ключовим словом	Введення запиту у поле пошуку	Список відео фільтрується за назвою та описом	Пройдено
ТС-24	Алгоритм трендів	Кілька відео з різною активністю за останні 48 годин	Відео впорядковані за спаданням рейтингу (більше активності — вище позиція)	Пройдено
ТС-25	Двомовний інтерфейс	Перемикання мови між «UK» та «EN»	Всі елементи UI перемикаються на відповідну мову	Пройдено

Аналіз результатів тестування свідчить про те, що всі 25 тест-кейси пройдені успішно, що підтверджує коректність реалізації основного функціоналу платформи «BenFube». Усі ключові сценарії роботи — від автентифікації та публікації відео до модерації та алгоритму трендів — функціонують відповідно до встановлених вимог.

Зведені результати тестування

Підсистема	Кількість тест-кейсів	Пройдено	Не пройдено
Автентифікація та безпека	7	7	0
Канали та відео	4	4	0
Соціальна взаємодія	7	7	0
Модерація та адміністрування	4	4	0
Пошук та алгоритми	2	2	0
Інтерфейс	1	1	0
Разом	25	25	0

Висновки до розділу 3

У третьому розділі описано процес розгортання, інтерфейс користувача та результати тестування платформи «BenFube».

Детально описано процес розгортання програмного комплексу: контейнеризацію бекенду засобами Docker та Docker Compose (три сервіси: app, db, nginx), конфігурацію через .env-файл, автоматичне CI/CD-розгортання бекенду на Render та фронтенду на Vercel при кожному push до гілки main репозиторію GitHub. Хмарне зберігання медіафайлів забезпечується Cloudinary з підтримкою CDN-доставки контенту.

Описано та проілюстровано основні елементи інтерфейсу платформи: головну сторінку з трендовими та новими відео, форму автентифікації з підтримкою Google OAuth, сторінку перегляду відео з вбудованим плеєром та коментарями, сторінку каналу з кнопкою донату, розділ Studio для авторів з аналітикою, форму завантаження відео та адмін-панель з управлінням скаргами та користувачами.

Проведено функціональне тестування методом «чорної скрині» за 25 тест-кейсами, що охоплюють усі ключові підсистеми платформи: автентифікацію,

канали, відео, соціальну взаємодію, модерацію, пошук та алгоритм трендів. Всі 25 тест-кейси успішно пройдені, що підтверджує відповідність реалізованого програмного комплексу функціональним вимогам, визначеним у другому розділі.

ВИСНОВКИ

У випускній кваліфікаційній роботі бакалавра вирішено актуальне завдання проектування та реалізації повнофункціональної вебплатформи для розміщення, перегляду та управління відеоконтентом «BenFube». Отримані результати повністю відповідають меті роботи та виконаним завданням.

У ході виконання роботи досягнуто такі результати.

1. Проведено аналіз предметної галузі відеохостингових платформ. Розглянуто та порівняно чотири платформи: YouTube, Vimeo, Dailymotion та PeerTube. Встановлено, що жодна з них не надає безкоштовний доступ до повного набору функцій: соціальна взаємодія, пряма монетизація через донати, адміністративна модерація та двомовний інтерфейс одночасно.

2. Визначено 20 функціональних та 9 нефункціональних вимог до платформи. Побудовано UML-діаграму варіантів використання для чотирьох ролей: гість, зареєстрований користувач, автор каналу та адміністратор.

3. Спроектовано реляційну базу даних у складі 12 таблиць (User, Channel, Video, Comment, Like, Subscription, Donation, Report, Tag, VideoTag, VideoView, VideoStatistics) з повним визначенням типів даних, зв'язків, індексів та унікальних обмежень. Розроблено концептуальну ER-діаграму та фізичну схему БД.

4. Обґрунтовано вибір технологічного стеку: PHP 8.2 та Symfony 6 для серверної частини, PostgreSQL як СУБД, Doctrine ORM для взаємодії з даними, Next.js 14 з TypeScript та Tailwind CSS для клієнтської частини, Cloudinary для хмарного медіасховища, JWT та Google OAuth 2.0 для автентифікації, Docker для контейнеризації, Vercel та Render для хмарного розгортання.

5. Реалізовано серверну частину: розроблено 26 REST API маршрутів, реалізовано шарову архітектуру (Controller→Service→Repository→Entity), JWT-автентифікацію з RSA-підписом, OAuth 2.0 через Google, алгоритм трендів на основі зваженої суми показників активності користувачів та коефіцієнта свіжості контенту (формула 2.1), систему скарг і апеляцій та повну адміністративну панель.

					ІПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						81

6. Реалізовано клієнтську частину: розроблено 15+ сторінок та компонентів Next.js, впроваджено App Router, Zustand-стор для управління станом, систему i18n з повним перекладом на українську та англійську мови, адаптивний інтерфейс для мобільних, планшетних та десктопних пристроїв, відеоплеєр на HTML5 та модальне вікно донату.

7. Виконано розгортання платформи у хмарному середовищі: бекенд контейнеризовано за допомогою Docker та розгорнуто на Render, фронтенд розгорнуто на Vercel з автоматичним CI/CD через GitHub. Проведено функціональне тестування за 25 тест-кейсами методом «чорної скрині» — всі тести пройдені успішно.

Практична цінність роботи полягає у тому, що розроблений програмний комплекс «BenFube» є повністю функціонуючим вебзастосунком, розгорнутим у хмарному середовищі та готовим до реального використання. Він може слугувати основою для розвитку комерційної відеоплатформи, освітнього порталу або корпоративного медіасервісу.

Напрямами подальшого розвитку та вдосконалення платформи є: реалізація підтримки прямих відеотрансляцій (Live Streaming) на основі протоколу RTMP/HLS; впровадження системи рекомендацій на основі машинного навчання; розробка мобільних застосунків для iOS та Android; реалізація транскодування відео у кілька якостей (240p, 480p, 720p, 1080p) для адаптивного стрімінгу; впровадження повноцінної платіжної системи для монетизації; розширення аналітичного модуля з детальними демографічними даними аудиторії.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cisco Annual Internet Report (2018–2023) White Paper. URL: <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html> (дата звернення: 01.05.2026).
2. Wu D., Hou Y. T., Zhu W., Zhang Y. Q., Peha J. M. Streaming video over the Internet: approaches and directions. IEEE Transactions on Circuits and Systems for Video Technology. 2001. Vol. 11, No. 3. P. 282–300. DOI: 10.1109/76.911156.
3. Burgess J., Green J. YouTube: Online video and participatory culture. Hoboken: John Wiley & Sons, 2018. 192 p.
4. Stokel-Walker C. YouTubers: How YouTube Shook Up TV and Created a New Generation of Stars. Kingston upon Thames: Canbury Press, 2019. 288 p.
5. Vimeo Documentation. Vimeo Developer API Reference. URL: <https://developer.vimeo.com/api/reference> (дата звернення: 03.05.2026).
6. Dailymotion Partner Help Center. Video Specifications. URL: <https://faq.dailymotion.com/hc/en-us/articles/115009495023> (дата звернення: 03.05.2026).
7. Zignani M., Gaito S., Rossi G. P. Follow the «Mastodon»: Structure and Evolution of a Decentralized Online Social Network. Proceedings of the AAAI International Conference on Web and Social Media. 2018. Vol. 12. P. 541–550.
8. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 616 p.
9. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures: Ph.D. Thesis. University of California, Irvine, 2000. 180 p. URL: <https://ics.uci.edu/~fielding/pubs/dissertation/top.htm> (дата звернення: 05.05.2026).

10. PHP Official Documentation. PHP 8.2 Release Notes. URL: <https://www.php.net/releases/8.2/> (дата звернення: 10.04.2026).

11. Symfony Documentation. Symfony 6.4 — The Fast PHP Framework. URL: <https://symfony.com/doc/6.4/index.html> (дата звернення: 10.04.2026).

12. PostgreSQL Global Development Group. PostgreSQL 15 Documentation. URL: <https://www.postgresql.org/docs/15/> (дата звернення: 11.04.2026).

13. Doctrine Project. Doctrine ORM 2.x Documentation. URL: <https://www.doctrine-project.org/projects/doctrine-orm/en/2.16/index.html> (дата звернення: 11.04.2026).

14. Next.js by Vercel. Next.js 14 Documentation. URL: <https://nextjs.org/docs> (дата звернення: 12.04.2026).

15. Tailwind CSS. Documentation. URL: <https://tailwindcss.com/docs> (дата звернення: 13.04.2026).

16. Cloudinary Documentation. Video Upload API. URL: https://cloudinary.com/documentation/video_upload_api_reference (дата звернення: 14.04.2026).

17. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. Internet Engineering Task Force (IETF), 2015. URL: <https://tools.ietf.org/html/rfc7519> (дата звернення: 15.04.2026).

18. Hardt D. The OAuth 2.0 Authorization Framework. RFC 6749. Internet Engineering Task Force (IETF), 2012. URL: <https://tools.ietf.org/html/rfc6749> (дата звернення: 15.04.2026).

19. Vercel Inc. Vercel Documentation. URL: <https://vercel.com/docs> (дата звернення: 16.04.2026).

20. Sommerville I. Software Engineering. 10th ed. Harlow: Pearson Education, 2016. 810 p.

21. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd ed. Hoboken: John Wiley & Sons, 2011. 240 p.

					ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
						84

22. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Upper Saddle River: Prentice Hall, 2017. 432 p.
23. PHP-FIG. PSR-12: Extended Coding Style. URL: <https://www.php-fig.org/psr/psr-12/> (дата звернення: 17.04.2026).
24. TypeScript. TypeScript 5.0 Handbook. URL: <https://www.typescriptlang.org/docs/handbook/intro.html> (дата звернення: 18.04.2026).
25. Docker Inc. Docker Documentation. URL: <https://docs.docker.com/> (дата звернення: 19.04.2026).
26. Microsoft Azure. REST API Design Guidelines. URL: <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design> (дата звернення: 20.04.2026).
27. OWASP Foundation. OWASP Top Ten 2021. URL: <https://owasp.org/Top10/> (дата звернення: 21.04.2026).
28. Fowler M. Patterns of Enterprise Application Architecture. Boston: Addison-Wesley, 2002. 560 p.
29. Object Management Group. UML 2.5.1 Specification. URL: <https://www.omg.org/spec/UML/2.5.1/> (дата звернення: 22.04.2026).
30. ДСТУ ISO/IEC 25010:2015. Системна та програмна інженерія. Вимоги та оцінювання якості систем та програмного забезпечення (SQuaRE). Модель якості системи та програмного забезпечення. Київ: ДП «УкрНДНЦ», 2019. 36 с.
31. Covington P., Adams J., Sargin E. Deep Neural Networks for YouTube Recommendations. Proceedings of the 10th ACM Conference on Recommender Systems. 2016. P. 191–198. DOI: 10.1145/2959100.2959190.
32. Nygren E., Sitaraman R. K., Sun J. The Akamai Network: A Platform for High-Performance Internet Applications. ACM SIGOPS Operating Systems Review. 2010. Vol. 44, No. 3. P. 2–19. DOI: 10.1145/1842733.1842736.
33. Zustand. A small, fast and scalable bearbones state-management solution. URL: <https://zustand-demo.pmnd.rs/> (дата звернення: 23.04.2026).

34. React. React Documentation. URL: <https://react.dev/> (дата звернення: 24.04.2026).

35. Render. Render Cloud Application Hosting Documentation. URL: <https://render.com/docs> (дата звернення: 25.04.2026).

					ПЗ.КР.Б - 121 - 26 - ПЗ	Арк.
						86

ДОДАТКИ

Додаток А

Лістинг класу AuthService бекенду платформи «BenFube»

Нижче наведено лістинг класу AuthService, що реалізує логіку реєстрації та автентифікації користувачів. Клас використовує Symfony Security для хешування паролів, LexikJWTAuthenticationBundle для генерації JWT-токенів та EntityManager для взаємодії з базою даних.

```
<?php

// src/Service/AuthService.php

namespace App\Service;

use App\Dto\RegisterDto;
use App\Entity\User;
use App\Repository\UserRepository;
use Doctrine\ORM\EntityManagerInterface;
use
Lexik\Bundle\JWTAuthenticationBundle\Services\JWTTokenManagerInterface;
use
Symfony\Component>PasswordHasher\Hasher\UserPasswordHasherInterface;
use Symfony\Component\HttpKernel\Exception\ConflictHttpException;

class AuthService
{
    public function __construct(
        private readonly EntityManagerInterface $em,
        private readonly UserRepository $userRepo,
        private readonly UserPasswordHasherInterface $hasher,
        private readonly JWTTokenManagerInterface $jwtManager,
    ) {}
```

```

public function register(RegisterDto $dto): array
{
    if ($this->userRepo->findOneBy(["email" => $dto->email])) {
        throw new ConflictHttpException("Email вже використовується");
    }
    if ($this->userRepo->findOneBy(["username" => $dto->username])) {
        throw new ConflictHttpException("Нікнейм вже зайнятий");
    }
    $user = new User();
    $user->setEmail($dto->email);
    $user->setUsername($dto->username);
    $user->setPasswordHash(
        $this->hasher->hashPassword($user, $dto->password)
    );
    $user->setRole("user");
    $this->em->persist($user);
    $this->em->flush();
    $token = $this->jwtManager->create($user);
    return ["user" => $this->serialize($user), "token" => $token];
}

```

```

public function login(string $email, string $password): array
{
    $user = $this->userRepo->findOneBy(["email" => $email]);
    if (!$user || !$this->hasher->isPasswordValid($user, $password)) {
        throw new
\Symfony\Component\HttpKernel\Exception\UnauthorizedHttpException("Bearer", "Невірний email або пароль");
    }
}

```

```

        if ($user->isBlocked()) {
            throw new
\Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException("Обліковий запис заблоковано");
        }
        $token = $this->jwtManager->create($user);
        return ["user" => $this->serialize($user), "token" => $token];
    }

    private function serialize(User $user): array
    {
        return [
            "id"      => $user->getId(),
            "email"   => $user->getEmail(),
            "username" => $user->getUsername(),
            "role"    => $user->getRole(),
            "avatarUrl" => $user->getAvatarUrl(),
            "createdAt" => $user->getCreatedAt()?->format(\DateTimeInterface::ATOM),
        ];
    }
}

```

Додаток Б

Лістинг класу VideoService бекенду платформи «BenFube»

Нижче наведено лістинг класу VideoService, що реалізує операції зі списком відео, збереженням нового запису та видаленням. Використовує Doctrine QueryBuilder для побудови гнучких SQL-запитів із підтримкою пошуку, сортування та пагінації.

```
<?php

// src/Service/VideoService.php

namespace App\Service;

use App\Dto\CreateVideoDto;
use App\Entity\Video;
use App\Entity\VideoTag;
use App\Repository\TagRepository;
use App\Repository\VideoRepository;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\HttpKernel\Exception\NotFoundHttpException;
use Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException;

class VideoService
{
    public function __construct(
        private readonly EntityManagerInterface $em,
        private readonly VideoRepository $videoRepo,
        private readonly TagRepository $tagRepo,
    ) {}

    public function getVideoList(
        string $search, string $sort, int $page, int $limit
    ): array {
```

					ІІЗ.КР.Б - 121 - 26 - ІІЗ	Арк.
						90

```

$qb = $this->videoRepo->createQueryBuilder("v")
    ->where("v.isBlocked = false");
if ($search !== "") {
    $qb->andWhere("v.title LIKE :s OR v.description LIKE :s")
        ->setParameter("s", "%{$search}%");
}
match ($sort) {
    "popular" => $qb->orderBy("v.viewsCount", "DESC"),
    "oldest" => $qb->orderBy("v.createdAt", "ASC"),
    default => $qb->orderBy("v.createdAt", "DESC"),
};
$total = (clone $qb)->select("COUNT(v.id)")
    ->getQuery()->getSingleScalarResult();
$videos = $qb->setFirstResult(($page - 1) * $limit)
    ->setMaxResults($limit)->getQuery()->getResult();
return [
    "videos" => array_map([$this, "serializeShort"], $videos),
    "total" => (int) $total,
];
}

public function createVideo(CreateVideoDto $dto, string $channelId): array
{
    $video = (new Video())
        ->setTitle($dto->title)
        ->setDescription($dto->description ?? "")
        ->setVideoUrl($dto->videoUrl)
        ->setThumbnailUrl($dto->thumbnailUrl ?? "")
        ->setDuration($dto->duration ?? 0)
        ->setChannel($this->em->getReference(

```

```

        "App\\Entity\\Channel", $channelId
    ));
    $this->em->persist($video);
    foreach ($dto->tags ?? [] as $tagName) {
        $tag = $this->tagRepo->findOneBy(["name" => $tagName])
        ?? (new \\App\\Entity\\Tag())->setName($tagName);
        $this->em->persist($tag);
        $this->em->persist(
            (new VideoTag())->setVideo($video)->setTag($tag)
        );
    }
    $this->em->flush();
    return $this->serializeShort($video);
}

public function deleteVideo(string $videoId, string $userId): void
{
    $video = $this->videoRepo->find($videoId);
    if (!$video) { throw new NotFoundException("Відео не знайдено"); }

    if ($video->getChannel()->getOwner()->getId() !== $userId) {
        throw new AccessDeniedHttpException("Немає доступу");
    }
    $this->em->remove($video);
    $this->em->flush();
}

private function serializeShort(Video $v): array
{
    return [

```

```

        "id"      => $v->getId(),
        "title"   => $v->getTitle(),
        "thumbnailUrl" => $v->getThumbnailUrl(),
        "videoUrl"  => $v->getVideoUrl(),
        "duration"  => $v->getDuration(),
        "viewsCount" => $v->getViewsCount(),
        "createdAt" => $v->getCreatedAt()?-
>format(\DateTimeInterface::ATOM),
        "channel"  => [
            "id"      => $v->getChannel()->getId(),
            "name"     => $v->getChannel()->getName(),
            "slug"     => $v->getChannel()->getSlug(),
            "avatarUrl" => $v->getChannel()->getAvatarUrl(),
        ],
    ];
}
}

```

Додаток В

Лістинг ключових хуків та компонентів фронтенду платформи «BenFube»

Нижче наведено лістинг кастомного React-хука `useVideoPlayer`, що реалізує логіку управління відтворенням відео (відтворення/пауза, перемотування, рівень гучності, повноекранний режим) та фіксацію перегляду через API. Хук повністю відокремлює логіку плеєра від компонента-відображення.

```
"use client";

// hooks/useVideoPlayer.ts

import { useRef, useState, useCallback, useEffect } from "react";
import { apiClient } from "@services/apiClient";

export function useVideoPlayer(videoId: string) {
  const ref = useRef<HTMLVideoElement>(null);
  const [playing, setPlaying] = useState(false);
  const [progress, setProgress] = useState(0);
  const [volume, setVolume] = useState(1);
  const [duration, setDuration] = useState(0);
  const [viewCounted, setViewCounted] = useState(false);

  const toggle = useCallback(() => {
    if (!ref.current) return;
    if (playing) { ref.current.pause(); }
    else { ref.current.play(); }
    setPlaying(p => !p);
  }, [playing]);

  const seek = useCallback((pct: number) => {
    if (!ref.current) return;
```

```

    ref.current.currentTime = (pct / 100) * ref.current.duration;
  }, []);

```

```

const changeVolume = useCallback((v: number) => {
  if (!ref.current) return;
  ref.current.volume = v;
  setVolume(v);
}, []);

```

```

const toggleFullscreen = useCallback(() => {
  if (!ref.current) return;
  if (!document.fullscreenElement) {
    ref.current.requestFullscreen();
  } else {
    document.exitFullscreen();
  }
}, []);

```

```

useEffect(() => {
  const el = ref.current;
  if (!el) return;
  const onTime = () => {
    setProgress((el.currentTime / el.duration) * 100 || 0);
    // Зархувати перегляд після 10 сек
    if (!viewCounted && el.currentTime >= 10) {
      setViewCounted(true);
      apiClient.post(`/api/videos/${videoId}/view`).catch(() => {});
    }
  };
  const onMeta = () => setDuration(el.duration);

```

```

el.addEventListener("timeupdate", onTime);
el.addEventListener("loadedmetadata", onMeta);
return () => {
  el.removeEventListener("timeupdate", onTime);
  el.removeEventListener("loadedmetadata", onMeta);
};
}, [videoId, viewCounted]);

return { ref, playing, progress, volume, duration,
  toggle, seek, changeVolume, toggleFullscreen };
}

```

Нижче наведено лістинг компонента VideoCard — основного елемента відображення відео у списках та стрічці. Компонент є Server Component Next.js, що забезпечує рендеринг на сервері та оптимальне SEO.

```

// components/VideoCard.tsx

import Link from "next/link";
import Image from "next/image";
import { formatDuration, formatViews, timeAgo } from "@lib/format";

interface VideoCardProps {
  id: string;
  title: string;
  thumbnailUrl: string;
  duration: number;
  viewsCount: number;
  createdAt: string;
  channel: { name: string; slug: string; avatarUrl: string };
}

export default function VideoCard({

```

```

id, title, thumbnailUrl, duration, viewsCount, createdAt, channel
}: VideoCardProps) {
return (
  <Link href={` /video/${id}`}
    className="group flex flex-col gap-2 cursor-pointer">
    { /* Обкладинка */ }
    <div className="relative rounded-xl overflow-hidden
      aspect-video bg-zinc-800
      transition-transform duration-200
      group-hover:scale-[1.02]">
      <Image src={thumbnailUrl} alt={title}
        fill className="object-cover"/>
      <span className="absolute bottom-1.5 right-2
        bg-black/80 text-white text-xs
        px-1.5 py-0.5 rounded">
        {formatDuration(duration)}
      </span>
    </div>
    { /* Мета-дані */ }
    <div className="flex gap-2">
      <Link href={` /channel/${channel.slug}`}
        className="shrink-0">
        <Image src={channel.avatarUrl || "/default-avatar.png"}
          alt={channel.name} width={36} height={36}
          className="rounded-full"/>
        </Link>
        <div className="flex flex-col min-w-0">
          <p className="text-sm font-semibold text-white
            line-clamp-2 leading-snug">
            {title}

```

```

</p>
<Link href={` /channel/${channel.slug}`}
      className="text-xs text-zinc-400 hover:text-white mt-0.5">
  {channel.name}
</Link>
<p className="text-xs text-zinc-500">
  {formatViews(viewsCount)} &bull; {timeAgo(createdAt)}
</p>
</div>
</div>
</Link>
);
}

```