МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ІНТЕРАКТИВНА ПЛАТФОРМА ДЛЯ РОЗМІЩЕННЯ ТА ПЕРЕГЛЯДУ ВІДЕОКОНТЕНТУ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____Віктор СЕНІШИН

«___»_____ 2026 р.

Керівник доцент кафедри ІПЗ _____Гліб ГОРБАНЬ

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 20___ р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Сенішина Віктора Вікторовича
(прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Інтерактивна платформа для розміщення та перегляду відеоконтенту

Керівник роботи: Горбань Гліб Валентинович, доцент кафедри ІПЗ

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: масштабована вебплатформа відеохостингу на базі мікросервісної архітектури з підтримкою завантаження та потокового відтворення контенту; асинхронний конвеєр транскодування відеофайлів за допомогою FFmpeg та брокера повідомлень RabbitMQ; набір тестових медіафайлів різних форматів для перевірки системи обробки та відеоплеєра.

4. Перелік питань, що підлягають розробці: аналіз предметної сфери платформ відеохостингу та архітектурних рішень існуючих аналогів; аналіз методів побудови масштабованих мікросервісних архітектур та підходів до асинхронної обробки медіаданих; проєктування та реалізація вебсистеми з незалежними модулями для управління користувачами, каталогізації та потокового відтворення контенту; розробка та інтеграція конвеєра автоматизованого транскодування відеофайлів на базі FFmpeg з використанням брокера повідомлень RabbitMQ; тестування продуктивності системи, оцінювання швидкості обробки відео та перевірка відмовостійкості інфраструктури під навантаженням.

5. Перелік графічних матеріалів: презентація

Керівник роботи _____ Гліб ГОРБАНЬ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Віктор СЕНІШИН
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «21» грудня 2026р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи
Тема: «Інтерактивна платформа для розміщення та перегляду відеоконтенту»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	Виконано
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	Виконано
3	Пошук і опрацювання наукових джерел. Огляд подібних систем для перегляду відеоконтенту	31.01.2026	01.03.2026	Виконано
4	Проєктування архітектури системи на основі мікросервісів та принципів Clean Architecture	02.02.2026	01.04.2026	Виконано
5	Програмна реалізація мікросервісів, розробка конвеєра обробки відео з використанням RabbitMQ та Ffmpeg, Реалізація інтерактивних функцій реального часу через SignalR та розробка фронтенд-частини на Angular 21.	02.04.2026	24.05.2026	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	Виконано
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	

Керівник роботи _____ Гліб ГОРБАНЬ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Віктор СЕНІШИН
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи

студента 402 групи ЧНУ ім. Петра Могили

Сенішин Віктор Вікторович

на тему: «ІНТЕРАКТИВНА ПЛАТФОРМА ДЛЯ РОЗМІЩЕННЯ ТА ПЕРЕГЛЯДУ ВІДЕОКОНТЕНТУ»

Керівник: доцент кафедри ІПЗ Горбань Гліб Валентинович

Об’єкт роботи – процес управління, обробки та потокової доставки відеоконтенту в сучасних вебсистемах.

Предмет роботи – мікросервісна архітектура та технології асинхронного транскодування для створення масштабованої платформи відеохостингу.

Мета – розробка високонавантаженої вебплатформи відеохостингу із забезпеченням надійного завантаження, автоматичної обробки та оптимізованого відтворення медіаконтенту.

У роботі проаналізовано предметну область та існуючі архітектурні рішення, обґрунтовано вибір мікросервісного підходу, спроєктовано архітектуру системи. У першому розділі оцінено аналоги та сформульовано технічні вимоги до системи. Здійснено реалізацію сервісів із конвеєром транскодування на базі FFmpeg та RabbitMQ. Другий розділ присвячено вибору мікросервісної архітектури, методів асинхронної обробки відео та технологій інтерактивності. У третьому розділі описано структуру проєкту, стек та особливості мікросервісів. Четвертий розділ містить опис реалізації модулів, налаштування бази даних та результати функціонального й навантажувального тестування, що підтвердили ефективність та відмовостійкість інфраструктури.

Сторінок – 89, таблиць – 8, рисунків – 16, посилань – 33, додатків – 5.

Ключові слова: *відеохостинг, мікросервісна архітектура, транскодування відео, потокове відтворення, FFmpeg, RabbitMQ, .NET, асинхронна обробка.*

ABSTRACT

to the qualification work by the student of the group 402 of Petro Mohyla Black Sea National University

Viktor Senishyn

«INTERACTIVE PLATFORM FOR HOSTING AND VIEWING VIDEO CONTENT»

Supervisor: Associate Professor of the Software Engineering Department, Hlib Horban.

The object of the study is the process of management, processing, and streaming delivery of video content in modern web systems.

The subject of the study is the use of microservice architecture and asynchronous transcoding technologies to create a scalable video hosting platform

The purpose of the work is the development of a high-load video hosting web platform that ensures reliable uploading, automatic processing, and optimized playback of media content.

In this work, the subject domain and existing architectural solutions are analyzed, the microservice approach is justified, and the system architecture is designed. The first chapter evaluates existing counterparts and formulates technical requirements. The implementation of services includes a transcoding pipeline based on FFmpeg and RabbitMQ. The second chapter focuses on the microservice architecture selection, asynchronous video processing methods, and interactivity technologies. The third chapter details the project structure, stack, and microservice design specifics. The fourth chapter covers module implementation, database configuration, and the results of functional and load testing, which confirm the efficiency and fault tolerance of the infrastructure.

Pages - 89, tables - 8, figures - 16, references - 33, appendices - 5.

Keywords: *video hosting, microservice architecture, video transcoding, streaming, FFmpeg, RabbitMQ, .NET, asynchronous processing.*

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

СУБД – Система управління базами даних

ЕОМ – Електронна обчислювальна машина

ПЗ – Програмне забезпечення

ШІ – Штучний інтелект

ABR – Adaptive Bitrate Streaming

AMQP – Advanced Message Queuing Protocol

API – Application Programming Interface

BOLA – Buffer Occupancy-based Lyapunov Algorithm

CAGR – Compound Annual Growth Rate

CORS – Cross-Origin Resource Sharing

DTO – Data Transfer Object

EDA – Event-Driven Architecture

EF Core – Entity Framework Core

HLS – HTTP Live Streaming

HTTP – Hyper Text Transfer Protocol

SQL – Structured Query Language

JWT – JSON Web Token

CORS – Cross-Origin Resource Sharing

MSA – Microservices Architecture

OVP – Online Video Platforms

QoE – Quality of Experience

QoS – Quality of Service

RPC – Remote Procedure Call

SPA – Single Page Application

SSE – Server-Sent Events

SVOD – Subscription Video on Demand

UGC – User-Generated Content

UI – User Interface

UX – User Experience

YARP – Yet Another Reverse Proxy

VU – Virtual User

ВСТУП

Сьогодні ринок медіаплатформ перенасичений рішеннями, які орієнтовані або на масового споживача з агресивною рекламною моделлю та алгоритмічними рекомендаціями, або на складні корпоративні системи, що мають надмірний функціонал і не передбачають прямої інтерактивної взаємодії. Це створює дефіцит спеціалізованих інструментів для невеликих спільнот, освітніх ініціатив чи приватних проєктів, які потребують гнучкого, кастомізованого майданчика для публікації контенту з безпосереднім зворотним зв'язком. Критичною проблемою існуючих сервісів є закритість архітектури, що обмежує можливості технічної адаптації та позбавляє власників контенту контролю над даними аудиторії.

У межах даної кваліфікаційної роботи розроблено проєкт, що являє собою масштабовану вебплатформу відеохостингу на основі мікросервісної архітектури. Впроваджений підхід дозволяє забезпечити повний цикл життєвого циклу медіаданих: від асинхронного транскодування за допомогою FFmpeg до реалізації інструментів інтерактивної взаємодії користувачів у режимі реального часу.

Об'єктом роботи є архітектурні підходи та технології побудови високонавантажених систем відеохостингу.

Предметом роботи є методи створення мікросервісів для обробки медіаданих, зокрема алгоритми асинхронного транскодування відео та технології надійного двостороннього зв'язку між клієнтом і сервером.

Метою роботи є проєктування та програмна реалізація масштабованої відеоплатформи. Система повинна забезпечувати повний цикл роботи з відеоданими від завантаження до автоматичного транскодування та потокового відтворення. Пріоритетним завданням є створення відмовостійкої інфраструктури, здатної витримувати високі навантаження та горизонтально масштабуватися при різкому збільшенні кількості користувачів або обсягу контенту.

Для досягнення мети необхідно вирішити такі завдання:

– оглянути існуючі стрімінгові платформи і способи взаємодії з ними;

– спроєктувати мікросервісну архітектуру, щоб зробити систему стійкою і з можливістю масштабування;

– створити API Gateway, де за допомогою YARP буде відбуватися маршрутизація запитів та загальна автентифікація;

– зібрати пайплайн для обробки відеофайлів, де за асинхронність відповідатиме RabbitMQ, а транскодування і нарізку картинок робитиме Ffmpeg;

– створити інтерфейс користувача на Angular 21, використовуючи SignalR для миттєвого оновлення даних на клієнті;

– протестувати платформу під навантаженням та перевірити всі її функції.

Сфера застосування результатів:

– спеціалізовані вебпортали: для забезпечення функціоналу відеотрансляції та інтерактивного зворотного зв’язку;

– навчальні ресурси: як майданчик для презентації матеріалів та збору реакцій користувачів;

– персональні проєкти: для автономної публікації медіаконтенту поза межами глобальних соціальних мереж.

Декларація про використання ШІ. Під час підготовки наукової роботи (академічного тексту) було використано інструмент генеративного штучного інтелекту Gemini. Його було застосовано для таких допоміжних завдань: генерування та оптимізація фрагментів програмного коду; розробка шаблонів для візуалізації даних; вичитування, редагування та реформатування чорнових варіантів тексту для адаптації та коригування емоційного тону відповідно до академічного стилю; переклад технічної документації; оцінювання якості матеріалу та отримання рекомендацій щодо покращення структури розділів. Весь основний текст, аналіз архітектурних рішень, інтерпретація результатів моделювання та висновки є результатом власної інтелектуальної праці автора. Було перевірено всю інформацію, отриману від ШІ, на точність та достовірність. Автор несе повну особисту відповідальність за зміст цієї роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

Класичні плеєри вміють тільки лінійно віддавати відеоряд. Для поточних реалій цього функціоналу об’єктивно мало. Їхня внутрішня структура не пристосована під постійні кліки по кадру чи швидку зміну сюжету на льоту. Якщо користувач позбавлений можливості впливати на контент, він швидко втрачає інтерес.

Задачі проєктування стали інакшими. Транслювати медіапотік – це тільки початок. Набагато складніше зробити так, щоб глядач міг у реальному часі взаємодіяти з картинкою. Для цього пишуть окрему клієнтську логіку, яка обробляє запити і миттєво змінює стан плеєра.

Сучасну відеоплатформу не можна розглядати просто як файловий хостинг. Технічно це дуже складна система. Вона складається з багаторівневої програмної логіки. Платформа підтримує постійний двосторонній зв’язок між клієнтом та сервером, а також використовує адаптивні алгоритми для дистрибуції даних [1]. Тут апаратна інфраструктура працює заради однієї мети. Вона перетворює стандартний відеоплеєр на повноцінне середовище для взаємодії з глядачем.

Через ці інновації модель пасивного спостереження повністю відходить у минуле. Глядач отримує частину контролю над відео. Це реалізується через розгалужені сценарії (branching), де користувач самостійно обирає подальший розвиток сюжету. Оповідь перестає бути лінійною. Також інтерактивність забезпечують клікабельні зони всередині кадру (hotspots) або пряма інтеграція електронної комерції (shoppable video) [2]. Для медіаринку це дає два конкретні результати. По-перше, значно зростає показник утримання уваги. По-друге, з’являються нові гнучкі інструменти для монетизації.

1.1 Опис предметної області: визначення, історія та еволюція

У розрізі сучасного стану розвитку інформаційних технологій, інтерактивна відеоплатформа розглядається не просто як сервіс, а як високоефективний

багатокомпонентний програмний комплекс. Його функціональне призначення охоплює повний життєвий цикл медіаданих: від індустріальних процесів завантаження та багаторівневого транскодування до стратегічного зберігання та інтерактивного відтворення контенту. Ключовою відмінністю таких систем є забезпечення безпрецедентного рівня залучення аудиторії через впровадження динамічних елементів інтерфейсу, що перетворюють пасивного глядача на активного учасника цифрової взаємодії [1].

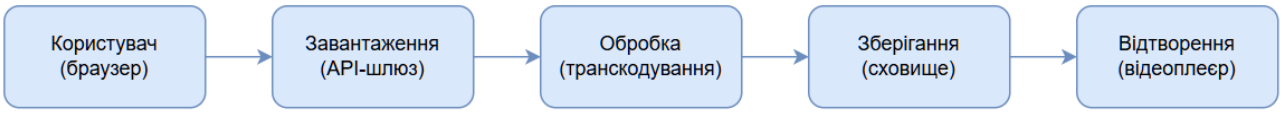

Рисунок 1.1 – Концептуальна схема життєвого циклу медіаданих у системі відеохостингу

Базову концептуальну схему життєвого циклу медіаданих у системі відеохостингу наведено на рис. 1.1.

Історико-технологічний генезис цієї галузі нерозривно пов’язаний із паралельним прогресом у сфері мережевих технологій, теорії стиснення даних та зростанням обчислювальних потужностей кінцевих пристроїв. Перші концептуальні спроби візуалізації та обробки медіа на ЕОМ датуються ще серединою XX століття. Проте, через критично високу вартість апаратного забезпечення та обмеженість інфраструктури передачі даних, реальний технологічний стрибок став можливим лише наприкінці 1980-х років, коли відбулася мініатюризація компонентів та стандартизація перших алгоритмів кодування відео [4].

Еволюційний шлях становлення та розвитку стрімінгових технологій доцільно сегментувати на кілька визначальних етапів, кожен з яких характеризується власним технологічним базисом:

– етап зародження та формування основ (1990–2000). Цей період позначився створенням мережевого фундаменту. Впровадження перших комерційних Ethernet-комутаторів компанією Kalpana у 1990 році заклало

необхідні передумови для функціонування високошвидкісних локальних та глобальних мереж [4]. Знаковою подією 1993 року стала перша жива інтернет-трансляція виступу гурту Severe Tire Damage. Використання експериментальної технології Mbone виявило величезну «голодність» відео до ресурсів: для передачі зображення з вкрай низькою роздільною здатністю (152 × 76 пікселів) було задіяно майже половину всієї пропускної здатності тодішнього сегменту Інтернету [4]. Поява RealPlayer від RealNetworks у 1995 році стала відправною точкою для комерціалізації стрімінгу, надавши користувачам перший доступний інструментарій для відтворення потокового медіа [5];

– етап масового поширення та масштабування (2000–2010). У ці роки логіка роботи з медіа кардинально змінилася. З релізом YouTube у 2005 році запрацювала модель UGC (User-Generated Content), і звичайні користувачі отримали змогу вільно завантажувати власні відео. У 2007 році Netflix відмовився від пересилання фізичних DVD-дисків і перейшов на стрімінг [5]. Технічно такий перехід став можливим лише завдяки масовому прокладанню широкосмугового інтернету. Додатково процеси відтворення уніфікувалися. Завдяки базовому HTTP та стандартизації HTML відео можна було запускати прямо у веббраузері без стороннього ПЗ [4];

– ера високої інтерактивності, мобільності та AI (2010–2026). Сучасний етап визначається синергією мобільних технологій та хмарних обчислень. Стрімка еволюція смартфонів і розгортання мереж 5G зняли обмеження щодо пропускної здатності, зробивши стандартом формати надвисокої чіткості (4K/8K) та складні інтерактивні функції в реальному часі [6]. Спеціалізація ринку призвела до появи платформ на кшталт Twitch, де взаємодія між автором і глядачем стала центральним елементом сервісу [7]. Глобальна пандемія COVID-19 виступила потужним каталізатором цифрової трансформації, змусивши кіностудії та освітні заклади повністю інтегруватися у стрімінгове середовище [8].

Зараз компанії активно інтегрують штучний інтелект. AI все частіше залучають до автоматизації транскодування відео та створення метаданих. Такий підхід знижує витрати платформ і покращує рівень обслуговування [7].

1.2 Статистичний аналіз та тенденції розвитку ринку

Попит на відеоплатформи сьогодні росте дуже швидко. Користувачі хочуть передавати й обробляти контент миттєво, без жодних затримок. Стрімінг вже працює в освіті, бізнесі та розвагах. Тому ринок зараз знаходиться на піку свого розвитку.

Статистика це чітко підтверджує. Очікується, що у 2025 році цей ринок складе 42,36 млрд доларів США. Наступного року сума стрибне до 54,54 млрд доларів. Середньорічний темп зростання (CAGR) тримається на рівні 28,7 [7]. Аудиторія постійно розширюється.

Стрімке зростання ринку зумовлене низкою технологічних факторів, зокрема глобальним розгортанням мереж 5G, підвищенням ефективності алгоритмів стиснення відеоданих та суттєвим збільшенням обчислювальної потужності мобільних пристроїв. За прогнозами експертів, за умови збереження поточної динаміки, до 2030 року обсяг ринку досягне 147,11 млрд доларів США [7].

Саме тому створювати нові платформи для відеоконтенту сьогодні так важливо. Перед розробниками стоять три головні проблеми, які треба вирішувати. Це масштабування архітектури, прискорення обробки відеоданих та гарантування стабільного онлайну для користувачів.

На рис. 1.2 наведено прогнозоване зростання ринку інтерактивного стрімінгу до 2030 року.

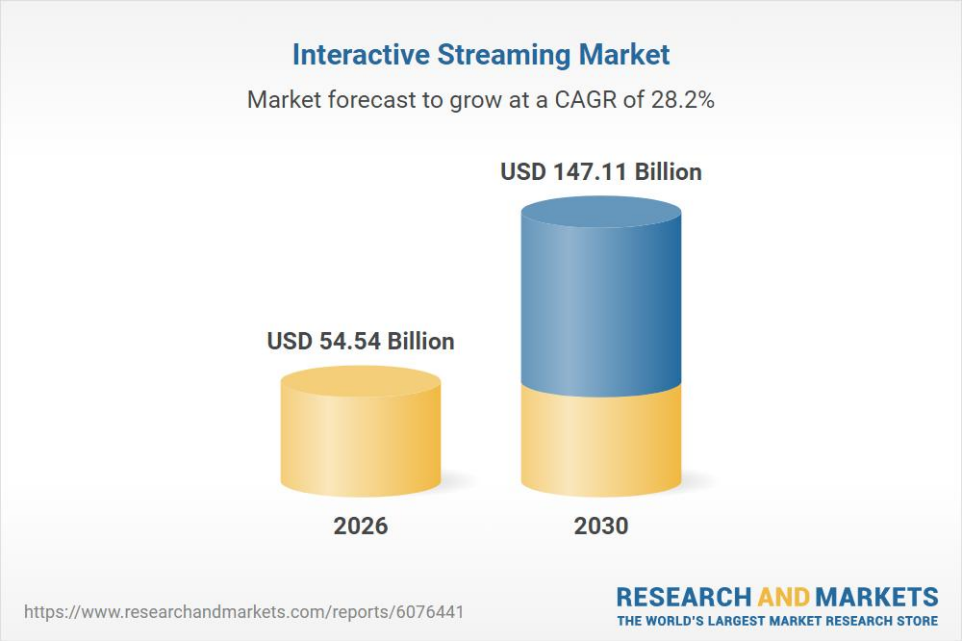

Рисунок 1.2 – Прогнозоване зростання ринку інтерактивного стрімінгу до 2030 року [7]

Глобальні статистичні показники відеоіндустрії на 2026 рік наведено в табл. 1.1.

Таблиця 1.1 – Глобальні статистичні показники відеоіндустрії на 2026 рік

Показник	**Значення**
Частка бізнесів, що використовують відео як інструмент маркетингу	91% [9]
Частка інтернет-трафіку, що припадає на відеоконтент (прогноз Cisco)	82% [9]
Світові витрати на цифрову відеорекламу	$292 млрд [9]
Частка ринку онлайн-відеоплатформ (OVP), що припадає на Live Streaming	54,7% [10]
Кількість користувачів Інтернету у світі (станом на 2023 р.)	5,4 млрд (67% населення) [7]
Середньомісячні витрати фанатів на SVOD сервіси	$71 [8]

Основними драйверами зростання є впровадження 5G, розвиток хмарних технологій та гейміфікація доставки контенту [7]. Зокрема, 47% брендів вже інтегрують інтерактивні елементи, такі як опитування та заклики до дії (CTA), що

підвищує показник завершення перегляду у 3,6 раза порівняно з пасивним відео[11]. Технології штучного інтелекту дозволяють скоротити час редагування відео на 28% та в 4,8 раза збільшити продуктивність продюсерів контенту [9].

1.3 Огляд та критичний аналіз аналогів

Ринок медіахостингів по суті розділений на два табори, які майже не перетинаються. З одного боку масові платформи типу YouTube, заточені під широку аудиторію і рекламну модель. З іншого корпоративні системи на кшталт Kaltura чи Panopto, які коштують десятки тисяч доларів на рік і розраховані суто на великі організації. Щось посередині для середнього бізнесу на ринку знайти важко. Спроби зробити «одне рішення для всіх» зазвичай закінчуються компромісами, які не влаштовують нікого або надто складно для малих команд, або надто обмежено для серйозного навантаження.

Проєктувати систему завжди доводиться з огляду на аудиторію. Мас-маркет вимагає постійного утримання уваги. На рівні інфраструктури це означає налаштування CDN та механізмів попереднього завантаження роликів. Корпоративний сегмент працює за іншою логікою. Там розважальні алгоритми просто зайві. Пріоритетом є захист самого відеопотоку від сторонніх. Інженери реалізують жорстке розмежування прав, шифрування та підключення до Active Directory компанії. Контроль доступу повністю витісняє рекомендаційні моделі [12].

Технічна прірва між продуктами постійно зростає. Публічні сервіси орієнтовані суто на трафік, відео має вантажитися швидко і для всіх. У професійному сегменті правила кардинально інші. Головним стає жорсткий контроль доступу та безпека самого медіапотоку. Відповідно, підхід до проєктування повністю змінюється. Підібрати інструменти на основі особистих симпатій розробника не вийде; фінальне рішення диктують виключно вимоги бізнесу.

На старті проєктування відеоплатформи головне – це зафіксувати цільовий сегмент. Технічні показники не беруться з повітря. Щоб грамотно налаштувати пропускну здатність та забезпечити відмовостійкість, треба чітко розуміти сценарії використання системи. Без цих вхідних даних розробити навіть базовий функціонал об’єктивно не вийде.

Таблиця 1.2 – Порівняльний аналіз провідних відеохостингів та інтерактивних платформ

Назва рішення	**Переваги**	**Недоліки**	**Особливості реалізації**
YouTube	Масштабна аудиторія (2,5 млрд+), безкоштовність, потужна SEO-оптимізація, інтеграція з Google Shopping [13].	Агресивна реклама, складність виходу в топ для нових авторів, сувора цензура [14].	Використання адаптивного стрімінгу DASH, власні кодеки (VP9/AV1), AI-рекомендації [8].
Vimeo	Професійна якість відтворення, відсутність реклами, гнучкі налаштування приватності, інструменти для командної роботи [14].	Висока вартість підписки для бізнесу, обмежене органічне охоплення аудиторії [14].	Орієнтація на 4K/8K контент, підтримка White-label плеєра [3].
Twitch	Найкращі інструменти для Live-взаємодії, розвинена культура стрімінгу, миттєвий зв’язок з аудиторією [15].	Вузька спеціалізація (геймінг/lifestyle), висока конкуренція серед стрімерів [16].	Протокол LL-HLS для мінімізації затримок, система API для сторонніх розширень.
TikTok	Унікальний алгоритм рекомендацій, висока віральність контенту, простота створення мобільного відео [16].	Обмеження по тривалості відео, залежність від мобільних додатків, питання конфіденційності даних [15].	Вертикальний формат, AI-фільтри в реальному часі, інтеграція TikTok Shop [13].
Mindstamp	Глибока інтерактивність: кнопки, запитання, малювання поверх відео, аналітика кожного кліку [3].	Не є платформою для дистрибуції (потребує стороннього хостингу на кшталт Vimeo) [3].	Хмарне накладання інтерактивних шарів, інтеграція з CRM-системами [1].

На ринку відеоплатформ зараз склалася парадоксальна ситуація. Існує дві крайнощі, і обидві мають критичні недоліки.

З одного боку глобальні гіганти (YouTube чи TikTok). Вони дають авторам колосальне охоплення аудиторії. Але за це доводиться платити повною відсутністю кастомізації. Архітектура цих систем залишається закритою, а власник контенту не має контролю над даними своїх глядачів [17]. Якщо корпоративному проєкту потрібен унікальний бренд-досвід та сувора конфіденційність, такі сервіси не підходять.

З іншого боку існують вузькоспеціалізовані інтерактивні системи (Mindstamp, Cinema8). Інструментарій для створення сценаріїв там значно кращий. Проте їхнє впровадження перетворюється на логістичну проблему. Головним їхнім недоліком є необхідність зберігати фізичні відеофайли на сторонніх хмарних сервісах [3]. Через це інфраструктура розпадається на частини. Власник змушений керувати кількома розрізненими системами одночасно. Це ламає наскрізну аналітику та робить сервіс нестабільним.

Така полярність сервісів залишила на ринку порожню нішу. Заповнити її має розробка, де класичний медіахостинг поєднується з можливістю гнучкого підключення інтерактивних модулів під конкретні задачі користувача.

Найкращим фундаментом для розробки такої системи є мікросервісна архітектура. Вона вирішує одразу три ключові проблеми: забезпечує незалежність компонентів, легко масштабується під навантаження та дозволяє адаптувати інтерфейс під замовника. Тільки мікросервісний підхід дозволяє об’єднати процеси зберігання відео та алгоритми інтерактивності в єдину безшовну екосистему.

1.4 Постановка задачі

Аналіз існуючих відеоплатформ виявив такі критичні обмеження, як закритість архітектури, фрагментованість інфраструктури та висока вартість, усунення яких є головним завданням цієї роботи.

Актуальність розробки пояснюється технічними обмеженнями наявних на ринку відеохостингів. Сьогодні компаніям вже недостатньо простого плеєра. Потрібні власні інструменти аналітики, адаптивний стрімінг та вільна інтеграція.

Через те, що комерційні продукти є закритими і не піддаються змінам, створення власної платформи з гнучкою архітектурою повністю вирішує цю проблему.

Метою роботи є проєктування та програмна реалізація масштабованої інтерактивної відеоплатформи VidPortal. Система повинна забезпечувати повний цикл роботи з відеоданими від завантаження до автоматичного транскодування та потокового відтворення. Пріоритетним завданням є створення відмовостійкої інфраструктури, здатної витримувати високі навантаження та горизонтально масштабуватися при різкому збільшенні кількості користувачів або обсягу контенту.

Об’єктом роботи є архітектурні підходи та технології побудови високонавантажених систем відеохостингу.

Предметом роботи є методи створення мікросервісів для обробки медіаданих, зокрема алгоритми асинхронного транскодування відео та технології надійного двостороннього зв’язку між клієнтом і сервером.

Для досягнення мети необхідно вирішити такі завдання:

- оглянути існуючі стрімінгові платформи і способи взаємодії з ними;
- спроєктувати мікросервісну архітектуру. Зробити систему стійкою і з можливістю масштабування;
- створити API Gateway. За допомогою YARP буде відбуватися маршрутизація запитів та загальна автентифікація [18];
- зібрати пайплайн для обробки відеофайлів. За асинхронність відповідатиме RabbitMQ [19], а транскодування і нарізку картинок робитиме Ffmpeg [20];
- створити інтерфейс користувача на Angular 21 [21]. Для миттєвого оновлення даних на клієнті буде використано SignalR;
- протестувати платформу під навантаженням, перевірити всі її функції.

Виконання цих кроків дозволить отримати робочий прототип відеосервісу. Створена архітектура підійде як базова основа для інших ‘вебплатформ, які повинні витримувати великий наплив користувачів без падіння серверів.

Висновки до розділу 1

Аналіз існуючих відеоплатформ виявив такі критичні обмеження, як закритість архітектури, фрагментованість інфраструктури та висока вартість, усунення яких є головним завданням цієї роботи.

Актуальність розробки пояснюється технічними обмеженнями наявних на ринку відеохостингів. Сьогодні компаніям вже недостатньо простого плеєра. Потрібні власні інструменти аналітики, адаптивний стрімінг та вільна інтеграція. Через те, що комерційні продукти є закритими і не піддаються змінам, створення власної платформи з гнучкою архітектурою повністю вирішує цю проблему.

Реалізація цієї платформи дозволить отримати робочий прототип відеосервісу. Створена архітектура підійде як базова основа для інших вебплатформ, які повинні витримувати великий наплив користувачів без падіння серверів.

2 МЕТОДИ ТА ПІДХОДИ ДО РЕАЛІЗАЦІЇ ІНТЕРАКТИВНОЇ ВІДЕОПЛАТФОРМИ

Щоб відеосервіс функціонував успішно, необхідна інтеграція відразу кількох рівнів, а саме надійного бекенду, оптимально налаштованої передачі даних по мережі та швидкого процесингу самих медіафайлів.

Ці компоненти мають працювати таким чином щоб гарантувати якість обслуговування (QoS) та стабільність відеопотоку. Система має продовжувати працювати без збоїв навіть за умов нестабільної мережі або різкого збільшення кількості одночасних з’єднань.

Головне завдання цього розділу – проаналізувати підходи до проєктування застосунку.

Для стабільної роботи платформи має бути обране сучасне технічне програмне забезпечення. У тексті розглядається загальна побудова архітектури. Також досліжуються способи для оптимізації процесів.

2.1 Архітектурна парадигма

Фундаментальним рішенням при проєктуванні платформи VidPortal став вибір мікросервісної архітектури (Microservices Architecture, MSA) як основного паттерну побудови системи. На відміну від традиційних монолітних систем, де програмні компоненти є тісно пов‘язаними, а будь-яка локальна модифікація коду вимагає повного перескладання та повторного розгортання всього додатку, MSA дозволяє здійснити стратегічну декомпозицію системи на сукупність невеликих, автономних та слабко пов’язаних сервісів. Кожен такий сервіс відповідає за конкретну бізнес-функцію та взаємодіє з іншими компонентами через стандартизовані інтерфейси програмування (API) [22].

Такий підхід є критично важливим саме для медіа-платформ, оскільки вони характеризуються нерівномірним розподілом навантаження на різні вузли системи. Наприклад, сервіс перегляду метаданих відео зазвичай опрацьовує велику кількість

легких запитів на читання, тоді як сервіс транскодування відео потребує значних обчислювальних потужностей (CPU/GPU) для тривалих операцій обробки сигналів. Мікросервісна архітектура дозволяє масштабувати ці компоненти незалежно один від одного, забезпечуючи оптимальне використання ресурсів серверної інфраструктури [23].

Внутрішня структура кожного окремого мікросервісу в проєкті реалізована з суворим дотриманням принципів Clean Architecture (відомої також як Onion Architecture або «цибулева архітектура»). Головною перевагою цього підходу є чітка сепарація відповідальності та інверсія залежностей, спрямована всередину до ядра системи. Це гарантує повну ізоляцію ключової бізнес-логіки від впливу зовнішніх факторів, таких як конкретні системи управління базами даних, зовнішні фреймворки або протоколи зв’язку. Логічна структура кожного сервісу в межах VidPortal включає такі рівні:

- Domain: сутності та бізнес-правила (наприклад, стан обробки відео ProcessingStatus);
- Application: юзкейси, інтерфейси сховищ та логіка обробки подій;
- Infrastructure: реалізація доступу до PostgreSQL, клієнти RabbitMQ, інтеграція з файловою системою;
- API: точки входу (Controllers) та налаштування контейнера залежностей.

Візуальну схему розподілу відповідальності та напрямку залежностей між рівнями окремого мікросервісу наведено на рис. 2.1.

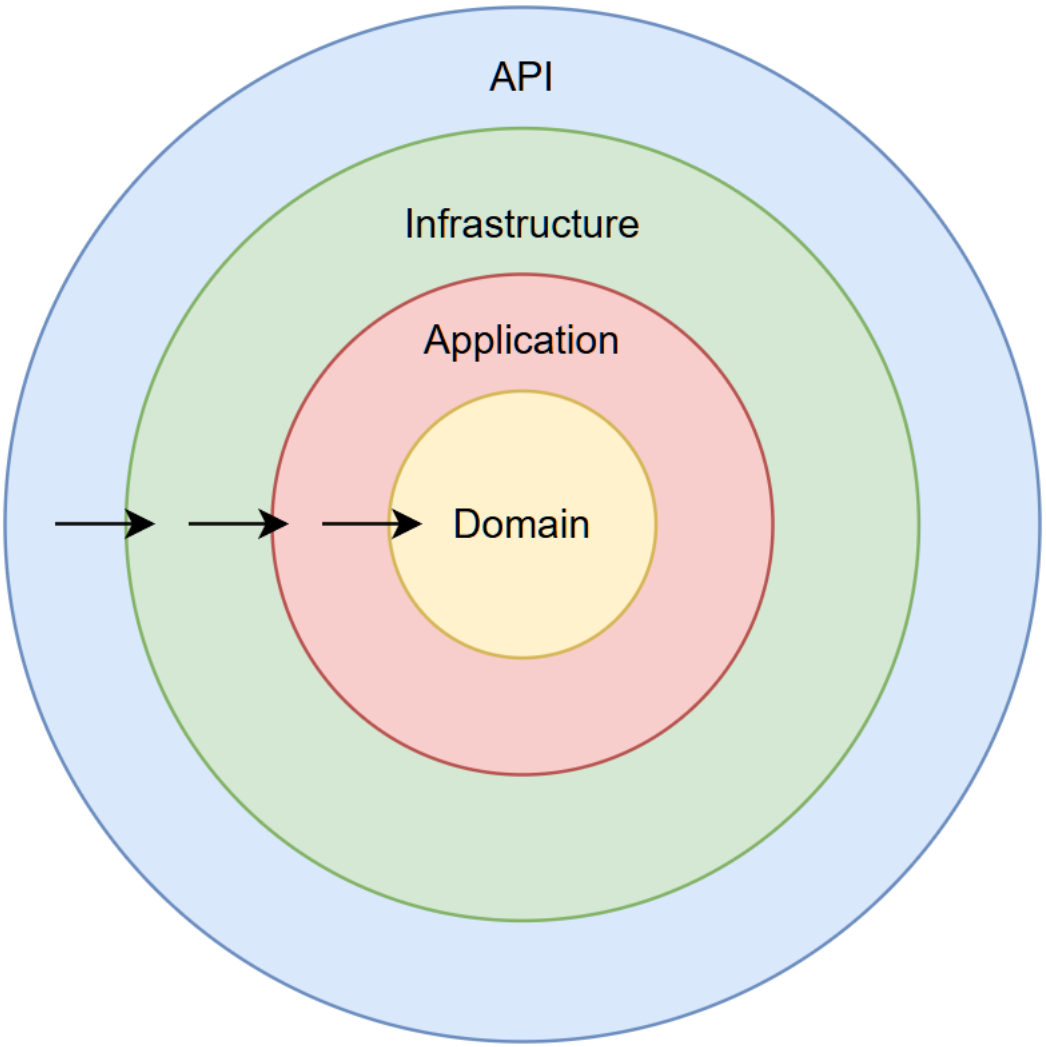

Рисунок 2.1 – Схема рівнів Чистої Архітектури

Для забезпечення ефективного керування вхідними потоками даних та оптимізації взаємодії між клієнтською частиною і серверною інфраструктурою у проєкті VidPortal було впроваджено паттерн API Gateway. Технічна реалізація даного компонента базується на використанні бібліотеки YARP (Yet Another Reverse Proxy) – сучасного, високопродуктивного зворотного проксі-сервера, розробленого компанією Microsoft спеціально для екосистеми .NET. Вибір цього інструменту обумовлений його нативною інтеграцією з платформою, високою гнучкістю конфігурації та здатністю обробляти значні обсяги трафіку з мінімальними затримками.

Впровадження API Gateway на основі YARP дозволяє вирішити декілька фундаментальних архітектурних завдань:

– абстрагування та приховування топології. Шлюз виконує роль єдиного фасаду. Це дає змогу повністю сховати внутрішню архітектуру мікросервісів від зовнішніх користувачів. Наш клієнтський додаток (Angular SPA) робить запити лише до однієї точки входу. Йому не потрібно знати реальні мережеві адреси, порти чи те, скільки саме екземплярів UserService або VideoService зараз працює. Як результат, безпека інфраструктури суттєво зростає, адже всі внутрішні мікросервіси знаходяться в ізольованому контурі, і до них немає прямого доступу з публічної мережі;

– уніфікація точок доступу для різних протоколів. Оскільки платформа передбачає як стандартну REST-взаємодію, так і обмін даними в реальному часі, YARP забезпечує прозору маршрутизацію обох типів трафіку. Він коректно обробляє тривалі HTTP-з’єднання та проксіює WebSockets, що є необхідною умовою для стабільної роботи SignalR-хабів. Таким чином, забезпечується цілісність сесій та мінімізуються складнощі при налаштуванні CORS-політик;

– винесення наскрізної логіки (Cross-cutting concerns). Усі базові завдання системи тепер працюють на рівні шлюзу. До них відносяться валідація JWT, автентифікація, логування та розподіл навантаження. Прикладні сервіси звільнені від цієї роботи. Їхні обчислювальні потужності спрямовані тільки на виконання бізнес-правил та роботу з медіаданими.

Застосування YARP у складі архітектури VidPortal не лише спрощує клієнтську розробку за рахунок наявності єдиного базового URL для всіх запитів, а й створює фундамент для подальшого горизонтального масштабування системи без необхідності внесення змін у код фронтенд-додатка.

2.2 Асинхронна обробка відео та FFmpeg конвеєр

Для розподіленої платформи VidPortal критично важливо правильно налаштувати зв’язок між мікросервісами. Звичайні синхронні HTTP-запити працюють неефективно, оскільки створюють жорстку прив’язку компонентів. Це

класичне «вузьке місце» мікросервісної архітектури якщо один компонент починає гальмувати, відбувається каскадний збій системи.

Транскодування відео високої чіткості створює величезне навантаження на систему. Дешифрування, зміна роздільної здатності та перекодування займають багато часу, тому їх не можна залишати в межах стандартного HTTP-запиту. Якщо виконати ці дії синхронно, сервер швидко вичерпає пул потоків, що неминуче призведе до таймаутів (timeout) та зависання інтерфейсу.

Для уникнення синхронних блокувань архітектуру системи побудовано за подієво-орієнтованою моделлю з використанням брокера RabbitMQ (протокол AMQP). Завдяки такому підходу використовується асинхронний обмін через черги, що гарантує слабку пов’язаність компонентів та ефективну буферизацію трафіку. Як наслідок, під час завантаження медіафайлу основний сервер миттєво повертає клієнту відповідь, а всі ресурсомісткі задачі безпечно делегуються фоновим обробникам.

Механізм роботи конвеєра обробки:

– користувач надсилає файл до VideoService. Файл зберігається, а в базу записуються початкові метадані;

– VideoService публікує подію VideoUploadedEvent у відповідний обмінник (Exchange) RabbitMQ;

– повідомлення потрапляє до черги, звідки його зчитує вільний фоновий сервіс-обробник (Worker), який використовує бібліотеку FFmpeg;

– виконується конвертація у формат H.264 (720p) та генерація мініатюри (Thumbnail). Приклад команди: ffmpeg -i input.mp4 -vf scale=1280:720 -c:v libx264 -crf 23 -c:a aac -b:a 128k output.mp4;

– після успішної обробки воркер публікує подію VideoProcessingCompletedEvent, на яку підписаний VideoService для оновлення статусу відео на «Ready».

RabbitMQ має вбудований механізм підтверджень (message acknowledgments), що гарантує відмовостійкість. Якщо воркер бере завдання, але

раптово зупиняється через збій, брокер не отримує сигнал про успішне виконання (ACK). Відбувається автоматичний NACK, і завдання повертається назад у чергу, де його безпечно підхоплює інший вільний інстанс.

Окрім медіафайлів, такий підхід гарантує цілісність даних і для інших процесів. Наприклад, для статусу верифікації Acknowledge settings у продукті. Щоб змінити цей статус через UI або запустити перевірку, система відправляє відповідну подію в брокер. Навіть якщо база даних тимчасово недоступна, критична зміна не втрачається, а безпечно очікує в черзі.

Ефективне масштабування обчислювальних процесів реалізується через стратегію горизонтального розширення (scaling out) за патерном «Competing Consumers». Усі воркери повністю ізольовані. Існує загальна черга, з якої читають дані одразу кілька інстансів. Завантажені медіафайли динамічно розподіляються між вільними вузлами. Задачі розбираються паралельно (наприклад, перший інстанс перекодовує відео, а другий у цей же час створює для нього прев'ю), що дозволяє максимально ефективно використовувати ресурси серверу.

2.3 Реалізація роботи застосунку в реальному часі

Сприйняття будь-якого застосунку формується його швидкістю. Затримки в інтерфейсі неприпустимі. Якщо для перевірки нових даних доводиться вручну перезавантажувати вкладку, платформа відразу виглядає застарілою. Основний акцент змістився на real-time взаємодію. Метою було побудувати такий канал зв'язку, де фронтенд та бекенд обмінюються даними миттєво і без зайвих HTTP-запитів.

Сьогодні динамічні застосунки не можуть нормально працювати без безперервного з'єднання з клієнтом. Усі дані мають підтягуватися «на льоту». Ніхто вже не змушує користувача тиснути F5, щоб побачити нові коментарі чи зміну лічильника переглядів. Для відеоплатформи це критично важливо. Особливо, коли треба миттєво показати автору, що бекенд нарешті завершив транскодування його файлу. Якщо ж інтерфейс «завмирає» і чекає ручного оновлення, досвід

використання погіршується. У користувачів відразу складається враження, що сервери не витримують навантаження і платформа працює повільно.

Звичайний HTTP-протокол тут не підходить через свою stateless-природу. Щоб отримати свіжі дані, клієнтський додаток змушений регулярно «опитувати» бекенд. Цей механізм (polling) страшенно перевантажує інфраструктуру, особливо якщо на платформі одночасно сидить багато користувачів. Більшість таких запитів є холостими, оскільки дані не змінилися. До того ж, інформація оновлюється не миттєво. Завжди є відчутна затримка між тим, коли подія реально сталася, і коли інтерфейс нарешті її відобразив.

З метою усунення зазначених недоліків у сучасних вебсистемах активно застосовуються технології персистентного з’єднання, які забезпечують постійний двосторонній канал зв’язку між сервером і клієнтом. Використання такого підходу дозволяє реалізувати концепцію Server Push, відповідно до якої сервер самостійно ініціює передачу інформації клієнту в момент виникнення певної події. Це забезпечує мінімальну латентність доставки повідомлень та дозволяє синхронізувати стан інтерфейсу з актуальними даними практично миттєво.

Сучасні відеоплатформи вимагають постійного повнодуплексного каналу. Користувачі повинні взаємодіяти із системою без пауз. Для VidPortal це була базова вимога до архітектури. Завдяки двосторонньому зв’язку користувачі відразу бачать нові перегляди, реакції та статуси обробки відео. Усе оновлюється в момент реальної події. Тому такий підхід – це не просто бонус до основного функціонала. Це основа, яка робить платформу по-справжньому інтерактивною.

У межах реалізації проєкту було обрано технологію SignalR, яка забезпечує високорівневу абстракцію над WebSockets та суттєво спрощує реалізацію функціоналу реального часу у середовищі .NET. На відміну від використання «сирих» WebSockets, SignalR надає готову інфраструктуру для керування підключеннями, автоматичного перепідключення клієнтів та підтримки декількох транспортних механізмів передачі даних.

У проєкті обрано SignalR через його переваги над сирими WebSockets:

– якщо браузер користувача, корпоративний проксі-сервер або мережеве оточення не підтримують WebSockets, SignalR автоматично перемикається на альтернативні механізми обміну даними, такі як Server-Sent Events (SSE) або Long Polling [24]. Це дозволяє забезпечити стабільність функціонування системи навіть у складних або обмежених мережевих умовах;

– SignalR автоматично виконує повторне підключення клієнта у випадку тимчасового розриву мережевого з’єднання, що є критично важливим для підтримки стабільної роботи інтерактивних компонентів. Крім того, бібліотека підтримує механізми групування клієнтів, завдяки чому можна надсилати повідомлення лише певним категоріям користувачів, наприклад глядачам конкретного відео [24];

– SignalR використовує концепцію «хабів», спеціалізованих серверних компонентів, які забезпечують двосторонню взаємодію між клієнтом та сервером. Такий підхід значно спрощує реалізацію RPC-викликів (Remote Procedure Call) та дозволяє викликати методи клієнта без необхідності ручного керування низькорівневими сокетними з’єднаннями [25].

Якщо оцінювати продуктивність мережі, WebSockets має дуже малий overhead (службове навантаження). Щойно завершується початкове встановлення з’єднання (handshake), протокол відкриває прямий повнодуплексний канал. Далі повідомлення ходять майже без зайвих HTTP-заголовків. Саме тому це ідеальний інструмент для обміну даними в реальному часі.

Проте в межах розробки складних розподілених систем, до яких належить VidPortal, використання виключно низькорівневого API WebSockets суттєво ускладнює реалізацію інфраструктурної логіки. Розробнику необхідно самостійно реалізовувати механізми повторного підключення, маршрутизації повідомлень, управління сесіями та підтримки альтернативних транспортів. У такому контексті SignalR виступає як високорівнева абстракція, що дозволяє значно скоротити витрати часу на розробку та зосередитися безпосередньо на реалізації бізнес-логіки застосунку [24].

Найсильніша сторона SignalR - це механізм «Graceful Degradation» (плавного зниження функціональності). Бувають ситуації, коли корпоративний фаєрвол або проксі жорстко блокує пряме WebSocket-з'єднання. Іноді проблема полягає в застарілому браузері. У таких випадках SignalR не обриває роботу, а автоматично перемикається на резервні транспортні протоколи. Ні розробнику, ні користувачеві нічого не треба для цього робити. Платформа просто продовжує стабільно тримати зв'язок за будь-яких налаштувань мережі.

SignalR знімає з розробника багато рутинних завдань. Бібліотека сама стежить за стабільністю каналу. Якщо з'єднання раптово розривається, відбувається автоматичне перепідключення. Управління сесіями та їхніми ідентифікаторами також працює «з коробки». Для відправки повідомлень використовуються хаби (Hubs), які реалізують класичну модель Pub/Sub. Такий підхід позбавив нас від необхідності писати важкий інфраструктурний код як на сервері, так і на клієнті.

Візуальну схему маршрутизації подій від бекенду до підключених клієнтів через SignalR-хаб наведено на рис. 2.2.

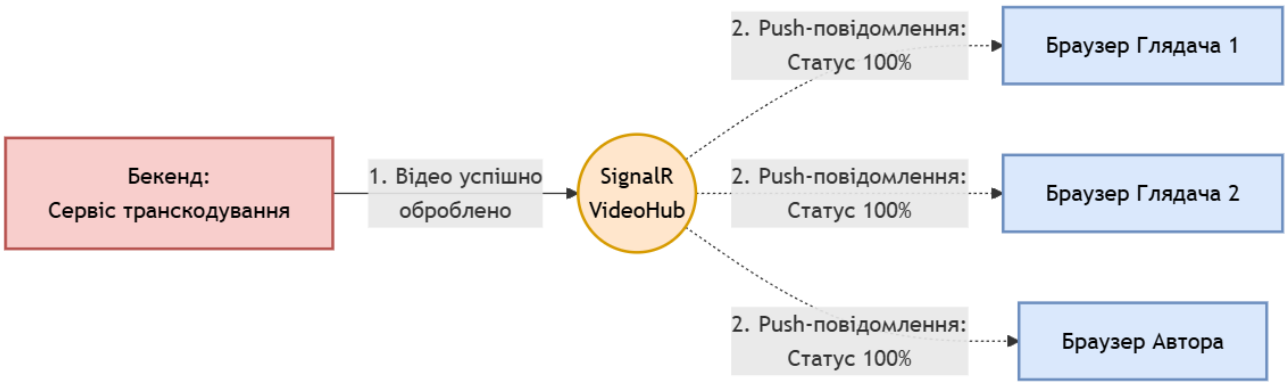

Рисунок 2.2 — Архітектура розсилки повідомлень за моделлю Pub/Sub через SignalR Hub

Таким чином, хоча використання SignalR і супроводжується незначним збільшенням службового мережевого трафіку порівняно з «чистими» WebSockets, зазначений недолік повністю компенсується значним скороченням складності

розробки, підвищенням стабільності роботи системи та забезпеченням надійної доставки повідомлень у різноманітних мережевих середовищах. У результаті застосування SignalR дозволило створити масштабовану, відмовостійку та зручну для підтримки систему взаємодії в режимі реального часу, що повністю відповідає вимогам сучасної мультимедійної вебплатформи.

2.4 Адаптивний стрімінг та досвід користувача (QoE)

Забезпечення безперервного та якісного відтворення мультимедійного контенту в умовах нестабільного мережевого з'єднання є одним із найскладніших викликів при розробці сучасних стрімінгових систем. Для нівелювання негативного впливу коливань пропускної здатності каналу зв'язку в індустрії широко використовується концепція адаптивного бітрейту (Adaptive Bitrate Streaming, ABR). Основна ідея цього підходу полягає у динамічному підлаштуванні якості відеопотоку до поточних технічних можливостей мережі користувача в режимі реального часу, що дозволяє мінімізувати або повністю усунути затримки на буферизацію, які критично знижують лояльність аудиторії.

З технічного погляду, імплементація ABR детермінує необхідність підготовки декількох варіантів одного і того ж відеофайлу з різною роздільною здатністю та рівнем стиснення (бітрейтом). Кожен такий варіант сегментується на невеликі фрагменти тривалістю від 2 до 10 секунд. Під час відтворення клієнтський плеєр постійно аналізує швидкість завантаження попередніх сегментів та стан заповненості буфера, на основі чого приймає інтелектуальне рішення про запит наступного фрагмента у вищій або нижчій якості. Такий механізм гарантує максимальну «плавність» перегляду, навіть при різкому падінні швидкості інтернету система не зупиняє відтворення, а лише тимчасово знижує візуальну чіткість зображення.

Хоча в поточній ітерації платформи VidPortal реалізовано метод прогресивного завантаження (Progressive Download) у форматі MP4, що забезпечує базовий рівень доступності контенту, закладена мікросервісна архітектура

спроєктована з урахуванням майбутнього масштабування та інтеграції просунутих протоколів доставки даних.

Зокрема, системний дизайн підтримує впровадження таких галузевих стандартів, як HLS (HTTP Live Streaming) та MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [26]. Ці протоколи базуються на використанні маніфест-файлів (.m3u8 або .mpd), які виступають дорожньою картою для плеєра, вказуючи посилання на всі доступні сегменти та рівні якості. Перехід до використання цих стандартів у VidPortal дозволить не лише реалізувати повноцінний адаптивний стрімінг, а й забезпечити сумісність із широким спектром пристроїв – від мобільних смартфонів до Smart TV, суттєво підвищуючи загальний показник якості користувацького досвіду (Quality of Experience, QoE).

Технічна реалізація механізму адаптивного стрімінгу базується на використанні спеціалізованого маніфест-файлу (наприклад, специфікації .mpd для стандарту MPEG-DASH або .m3u8 для HLS), який виконує роль інформаційного дескриптора всієї медіасесії. Цей документ містить вичерпний опис доступних профілів якості, включаючи перелік роздільних здатностей, відповідні їм значення бітрейту, параметри кодеків та посилання на окремі сегменти відео. На основі цих даних клієнтський плеєр здійснює інтелектуальний вибір оптимального потоку, що відповідає поточним апаратним та мережевим ресурсам користувача [27].

Для забезпечення максимальної плавності відтворення та мінімізації латентності, сучасні клієнтські плеєри використовують складні математичні алгоритми адаптації, які можна класифікувати за принципом прийняття рішень:

- BOLA (Buffer Occupancy-based Lyapunov Algorithm): даний алгоритм базується на теорії стабільності Ляпунова та орієнтований на управління станом буфера відтворення. Ключовою перевагою BOLA є його здатність приймати рішення щодо вибору бітрейту виключно на основі аналізу поточного рівня заповненості буфера, ігноруючи часто помилкові або нестабільні оцінки миттєвої швидкості мережі. Це дозволяє ефективно уникати критичних ситуацій вичерпання

даних (stalls) та забезпечувати стабільну роботу системи навіть в умовах значних мережевих коливань [27];

– Throughput-based (Метод оцінки пропускної здатності): цей класичний підхід базується на ретроспективному аналізі швидкості завантаження попередніх сегментів відео. Алгоритм безперервно обчислює реальну пропускну здатність каналу зв'язку, використовуючи методи ковзного середнього або інші статистичні фільтри. Якщо розрахована швидкість перевищує поріг наступного рівня якості, плеєр ініціює запит на підвищення бітрейту, намагаючись забезпечити глядачу максимально чітке зображення. Проте даний метод є чутливим до різких короткочасних падінь швидкості, що вимагає впровадження додаткових механізмів згладжування для уникнення частого перемикання планів якості [27].

Дослідження у сфері якості сприйняття відео (Quality of Experience, QoE) показують, що найбільш ефективними є гібридні підходи, які комбінують аналіз пропускної здатності з моніторингом стану буфера. Це дозволяє системі VidPortal не лише адаптуватися до змін у мережі, а й зберігати стійкість відтворення, забезпечуючи безперервний досвід перегляду незалежно від типу клієнтського пристрою та якості з'єднання.

Висновки до розділу 2

У межах другого розділу було здійснено комплексне науково-технічне обґрунтування вибору ключових архітектурних рішень, методів інтелектуальної обробки даних та стратегічних підходів до забезпечення високого рівня інтерактивності платформи VidPortal. На основі проведеного системного аналізу та порівняння сучасних технологічних стеків було сформульовано такі висновки:

– масштабованість архітектури: мікросервіси та Clean Architecture – це найкращий вибір для такого проєкту. Вони дають системі потрібну гнучкість. Бізнес-логіка не залежить від інфраструктури [28]. Через це код набагато легше тестувати та оновлювати. Важкі процеси, як перекодування відео, масштабуються окремо. Весь вхідний трафік проходить через API Gateway на базі YARP [29]. Він

централізує маршрутизацію всіх запитів. Водночас YARP надійно ховає мікросервіси від прямого доступу з інтернету;

– асинхронний конвеєр відео: подієвий підхід чудово підійшов для проєкту. Задачі йдуть через брокер RabbitMQ, а транскодуванням займається FFmpeg у фонових воркерах [30]. Навантаження на інфраструктуру розподіляється рівномірно. Основний плюс – клієнтська частина більше не блокується. Користувач може вільно гортати сторінки, поки система перекодовує файл. Для платформи з великим напливом користувачів це обов'язкова вимога;

– інтерактивність та надійність зв'язку: для функцій реального часу було обрано бібліотеку SignalR. Це найкращий варіант для стабільного повнодуплексного зв'язку. Її головна перевага, вбудовані механізми fallbacks (резервних протоколів). Якщо WebSocket з якихось причин блокується, SignalR автоматично переходить на інші методи передачі. Завдяки цьому платформа гарантовано доставляє системні сповіщення. Лічильники переглядів та реакції користувачів оновлюються миттєво. Зв'язок не обривається навіть при слабкому інтернеті чи обмеженнях старого браузера;

– клієнтський досвід (QoE) та плавне відтворення: сучасна аудиторія не прощає пауз на буферизацію. Тому в проєкті зроблено акцент на технології адаптивного бітрейту (ABR). Якість відео має динамічно змінюватися разом із пропускною здатністю мережі. Було розібрано алгоритми оцінки трафіку (BOLA та Throughput-based). Завдяки цьому вдалося підібрати правильну стратегію доставки медіафайлів. Плеєр автоматично перемикає відеопотоки. Відтворення відбувається максимально плавно, а користувач не дратується через очікування.

Запропонований комплекс методологічних та архітектурних рішень формує цілісну, науково обґрунтовану технологічну базу, яка є необхідним фундаментом для переходу до етапу практичної інженерної розробки та програмної реалізації всіх компонентів екосистеми.

3 СТРУКТУРА СИСТЕМИ ВІДЕОХОСТИНГУ

Програмна частина платформи VidPortal побудована за допомогою розподіленої мікросервісної архітектури з дотриманням сучасних ‘вебстандартів та принципів чистої архітектури. Головними пріоритетами під час розробки були реалізація високої швидкості та доступность всіх елементів застосунку, зручний користувацький досвід. Система здатна легко масштабується і обробляти велику кількість запитів завдяки балансуванню навантаження та розподіленому підходу для зберігання інформації в базах даних. Щоб платформа працювала стабільно навіть під час збою окремих вузлів, компоненти спроєктовано з високим рівнем відмовостійкості.

Клієнтська частина реалізована як SPA. UI зроблено використанням сучасних стилів та бібліотек. Оскільки бекенд розбито на мікросервіси, платформу легко доопрацьовувати і маштабувати без суттєвих змін в коді. Систему просто розширювати за допомогою додавання нових модулів та компонентів, не зачіпаючи ядро системи.

3.1 Структура проєкту та технологічний стек

Процес практичної реалізації інтерактивної відеоплатформи VidPortal базується на використанні синергії сучасних екосистем .NET та Angular, що дозволяє впровадити уніфікований підхід до суворої типізації даних на всіх рівнях архітектури - від серверної логіки до клієнтського інтерфейсу. Вибір даного технологічного стеку обумовлений необхідністю забезпечення екстремальної продуктивності, високої швидкості розробки та здатності системи до горизонтального масштабування в умовах інтенсивного зростання обсягів медіаданих.

У табл. 3.1 наведено детальний перелік базових технологій та інструментальних засобів, які формують технологічний ландшафт платформи VidPortal.

Таблиця 3.1 – Технологічний стек платформи VidPortal

Рівень	Технологія	Роль у системі
Backend	.NET 9 (C#)	Основна платформа для мікросервісів та воркерів.
API Gateway	YARP	Маршрутизація, балансування та безпека.
Бази даних	PostgreSQL	Кожен сервіс має власну БД (Database per Service) [31].
Черги повідомлень	RabbitMQ	Асинхронний зв’язок між сервісами.
Обробка відео	FFmpeg	Транскодування, створення thumbnails.
Real-time	SignalR	Сповіщення, оновлення метрик у реальному часі.
Frontend	Angular 21	Побудова реактивного інтерфейсу користувача.
Стилізація	Tailwind CSS v4	Гнучка та швидка побудова адаптивного дизайну.
Оркестрація	.NET Aspire	Управління життєвим циклом локальних та хмарних ресурсів.
Тестування продуктивності	Grafana k6	Імітація користувацького навантаження, стрес-тестування API та перевірка пропускної здатності системи [32].

Аналіз даних, наведених у табл. 3.1, дозволяє стверджувати, що обраний комплекс технологій охоплює всі критично важливі шари програмної системи. Зокрема, використання останньої версії платформи .NET 9 забезпечує доступ до найсучасніших оптимізацій JIT-компілятора та покращених механізмів роботи з пам’яттю, що є життєво необхідним для сервісів обробки відеопотоків.

Впровадження YARP як інтелектуального шлюзу (API Gateway) дозволяє абстрагувати складну внутрішню топологію мікросервісів від клієнтської частини, забезпечуючи при цьому єдиний стандарт автентифікації та логування запитів. На рівні збереження даних вибір PostgreSQL детермінований надійністю цієї СУБД та її здатністю ефективно працювати зі складними типами даних, що характерно для систем із розгалуженою метаінформацією про контент.

Особливу увагу приділено асинхронній взаємодії, брокер повідомлень RabbitMQ виступає сполучною ланкою, яка дозволяє VideoService не очікувати на завершення тривалих операцій транскодування, передаючи завдання воркерам через черги. Така архітектурна ізоляція у поєднанні з використанням SignalR для зворотного зв’язку дозволяє досягти ефекту «живого» інтерфейсу, де користувач отримує оновлення статусу обробки без необхідності перезавантаження сторінки.

Для реалізації фронтенду використовується Angular 21, цей фреймворк забезпечує модульність та інтерактивність. Фреймворк дозволяє використовувати такі особливості як Standalone Components та Signals, дозволяють оптимізувати change detection та забезпечити плавну роботу інтерфейсу навіть при відображенні великої кількості динамічних елементів. Для роботи зі стилями фреймворк Tailwind CSS v4 доповнює екосистему технологій, надаючи інструменти для швидкої розробки платформи під різні типи пристроїв.

Для управління мікросервісами та їх оркестрації використано .NET Aspire. Це рішення забезпечує зручний service discovery та єдине середовище для моніторингу всіх вузлів. Така інфраструктура дозволяє бекенду витримувати велику кількість одночасних користувацьких запитів і стабільно маршрутизувати завдання на сервіси обробки мультимедіа.

Приклад моніторингу ресурсів та стану сервісів а також їх оркестрації наведенно на рис. 3.1.

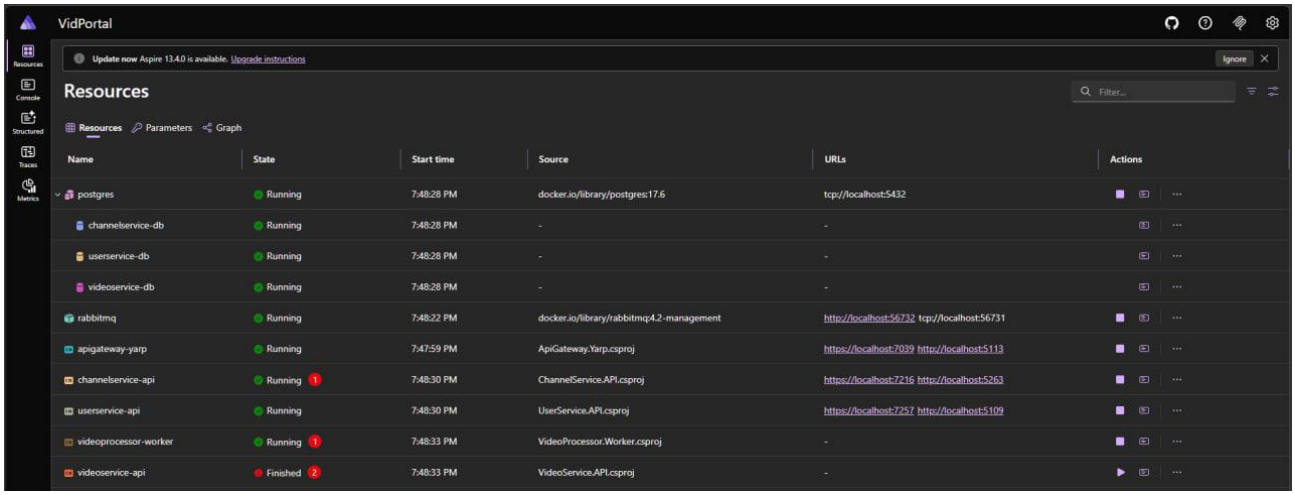

Рисунок 3.1 – Сторінка .Net Aspire для моніторингу сервісів і баз даних

Використання описаного технологічного стеку та засобів оркестрації формує надійний архітектурний фундамент для подальшої програмної реалізації бізнес-логіки системи.

3.2 Мікросервісна архітектура .NET-проєкту

Архітектура серверної частини VidPortal базується на ізольованих мікросервісах, що дає змогу масштабувати та оновлювати конкретні вузли (наприклад, конвеєр транскодування) без зупинки всього застосунку. Внутрішня структура кожного сервісу відповідає вимогам Clean Architecture. Такий поділ кодової бази на чотири стандартизовані рівні гарантує слабку пов’язаність (loose coupling) і зручність тестування, оскільки залежності завжди спрямовані виключно до центру системи:

– ядро системи (Domain): тут лежать лише сутності та основні правила. Ніяких баз даних чи сторонніх фреймворків сюди не підключають, шар є абсолютно ізольованим;

– шар Application: відповідає безпосередньо за логіку роботи сервісу. У ньому знаходяться Use Cases, інтерфейси та DTO. Це потрібно для того, щоб клієнт отримував тільки ті дані, які дійсно запитував;

– інфраструктурний рівень (Infrastructure): тут зібрана вся технічна реалізація: підключення бази через Entity Framework Core, налаштування брокера RabbitMQ та доступ до файлового сховища;

– API (для синхронних запитів) або Worker (для фонових): цей шар просто приймає HTTP-запити чи події з черги, виконує базову перевірку і віддає дані в Application. Жодної бізнес-логіки контролери не містять.

Така уніфікація захищає бізнес-логіку від жорсткої прив’язки до інструментів. Якщо виникне потреба змінити базу даних або провайдера авторизації, достатньо буде переписати лише один інфраструктурний шар відповідного сервісу.

Для забезпечення високої автономності, гнучкого горизонтального масштабування та чіткої декомпозиції бізнес-функцій розроблювану екосистему було розподілено на незалежні мікросервіси та спільні інфраструктурні компоненти:

– мікросервіс UserService: інкапсулює в собі всю логіку безпеки, автентифікації та авторизації платформи. Він слугує базовою точкою входу, де створюються нові профілі, перевіряються паролі та генеруються JWT-токени. На рівні предметної області (Domain) описано тільки дві основні сутності: User і RefreshTokens. Замість реалізації власної криптографії в інфраструктурному шарі застосовано індустріальний інструментарій ASP.NET Core Identity, який надійно хешує паролі та здійснює взаємодію з базою даних. Кінцеві маршрути для логіну, реєстрації чи оновлення сесії користувача винесені у відповідні API-контролери;

– мікросервіс ChannelService: повністю ізолює процеси соціальної взаємодії та організації контенту навколо авторів платформи. Модуль відповідає за створення авторських каналів, роботу з підписками та формування плейлистів. У його предметній області закладено сутності Channel, Playlist, Subscription, а також перелік ролей ChannelRole, необхідний для чіткого розподілу повноважень між власниками, адміністраторами та модераторами спільнот. Щоб гарантувати безпеку, на рівні інфраструктури застосовано policy-based авторизацію, завдяки

чому система автоматично блокує будь-які несанкціоновані спроби модифікації медіаресурсів або опису каналу;

– мікросервіс VideoService: є найбільш високонавантаженим вузлом платформи, який керує життєвим циклом медіаконтенту від моменту приймання оригінальних файлів до аналізу реакцій і дискусій глядачів. Усередині сервісу реалізовані сутності відео та соціальної активності. Для уникнення монолітності логіку в Application-шарі розділено через інтерфейси IVideoService, IDiscoveryService та IEngagementService. Взаємодія з файлами побудована в асинхронному режимі за допомогою брокера повідомлень RabbitMQ (сервіс публікує подію в чергу відразу після завантаження оригіналу), а для клієнтської частини реалізовано SignalR-хаб, який працює паралельно з REST-запитами та забезпечує миттєве оновлення лічильників і статусів обробки на клієнті без перезавантаження сторінки;

– мікросервіс VideoProcessor (воркер): кардинально відрізняється від інших частин системи, оскільки є фоновим обробником (Worker), а не звичайним сервісом із HTTP-контролерами. Його головне призначення ресурсомістке транскодування відео та генерація прев'ю. Цей процес повністю ізольований на рівні інфраструктури, тому важкі обчислювальні виклики FFmpeg жодним чином не гальмують основні API-шлюзи платформи. Модуль реалізовано на базі шаблону BackgroundService у .NET, який безперервно прослуховує чергу в RabbitMQ. Сам пайплайн обробки логічно винесений у шар Application, а взаємодію з файлами та запуск команд утиліти FFmpeg виконує Infrastructure. Після завершення конвертації воркер публікує подію назад у чергу для актуалізації статусу відео. Система є відмовостійкою: у разі виникнення технічного збою повідомлення повертається в чергу для повторної обробки іншим інстансом;

– Загальні компоненти BuildingBlocks: представлені у вигляді спільної крос-сервісної бібліотеки, створеної для усунення дублювання коду та винесення наскрізного функціонала. Вона виступає базовою залежністю для всіх модулів системи та уніфікує їхню структуру на чотирьох рівнях: на рівні Domain бібліотека

містить спільні сутності та універсальну обгортку ApiResponse<T>, що стандартизує структуру JSON-відповідей для фронтенду; на рівні Application & Infrastructure містяться абстракції IMessageBus та інтерфейси файлового сховища, що приховують низькорівневу технічну логіку RabbitMQ і звільняють сервіси від рутинної роботи; на рівні API налаштовано глобальний middleware для обробки винятків та валідації JWT, що дозволило позбутися повторюваних блоків try-catch у контролерах; на рівні Contracts зберігаються записи подій (records), використання яких відправником та отримувачем гарантує коректну десеріалізацію повідомлень у шині даних.

Така централізація архітектурних компонентів та ізоляція сервісів значно спрощує підтримку платформи, усуває каскадні збої та мінімізує технічні розбіжності між доменами при подальшому горизонтальному масштабуванні інфраструктури.

3.3 Проєктування моделей даних та доменів

Для надійної роботи мікросервісів у VidPortal використано патерн Database per Service. Кожен сервіс працює в межах свого обмеженого контексту (Bounded Context) і має власну, ізольовану базу даних.

Це рішення дає кілька переваг:

- автономність: сервіси розвиваються та змінюються незалежно. Відсутня спільна БД, яка часто стає «вузьким місцем» монолітів;
- відмовостійкість: збій в одному сервісі не викликає каскадного падіння всієї платформи, оскільки інші компоненти не залежать від його даних безпосередньо;
- гнучкість ресурсів: підсистеми транскодування, профілів та контенту мають різні вимоги до заліза, тому ізольовані бази дозволяють масштабувати їх окремо.

Такий поділ на доменні області – це базовий спосіб зробити сервіси дійсно незалежними та легкими в управлінні.

3.3.1 Домен управління відеоконтентом (VideoService)

Серцем розроблюваної платформи є VideoService. Це найбільший модуль системи, який повністю бере на себе життєвий цикл відео: від моменту завантаження файлу користувачем до публікації та збору реакцій аудиторії.

Інформаційним та логічним ядром цього сервісу виступає складний агрегат, сутність Video. Вона консолідує текстові метадані (ідентифікатор ID, назву, опис, дату публікації), лічильники популярності, тривалість ролика та ідентифікатор файлу у хмарному сховищі. Без цих атрибутів було б неможливо реалізувати ні пошук, ні алгоритми рекомендацій.

Архітектура передбачає гнучку систему класифікації: відео жорстко прив’язується до певної категорії (сутність Category) та може мати безліч пошукових маркерів завдяки зв’язку багато-до-багатьох із таблицею Tag.

Окремої уваги заслуговує стан обробки контенту ProcessingStatus. Саме він керує життєвим циклом файлу і є ключовим механізмом синхронізації між VideoService та фоновим воркером VideoProcessor:

- Processing: статус після завантаження;
- Ready: статус, коли відео успішно транскодовано і готове до перегляду;
- Failed: статус, що вказує на виникнення помилки під час обробки.

Оскільки платформа застосовує адаптивний стрімінг, сутність VideoSource пов’язує конкретне відео з переліком доступних роздільних здатностей (від 360p до 2160p) та відповідними URL-адресами готових потоків.

Для того, щоб контент був інтерактивним, об’єкт Video накопичує метрики популярності: кількість переглядів, лайків та дизлайків. На основі цих даних формуються тренди платформи. Всі активності глядачів суворо протоколюються: оцінки контенту зберігаються у VideoReaction, історія сесій у VideoView, а скарги на порушення правил акумулюються в VideoReport для подальшого розгляду адміністраторами.

Для організації дискусій створено модель VideoComment, яку прив'язано безпосередньо до відео. Вона підтримує деревоподібну структуру обговорень завдяки самопосилальному необов'язковому полю ParentCommentId, якщо воно заповнене, запис вважається відповіддю на інший коментар. Такий підхід дозволяє тримати всю логіку обговорень у межах єдиного медіаконтексту.

Структуру таблиць для Video Service та зв'язки між ними наведено на рис. 3.2.

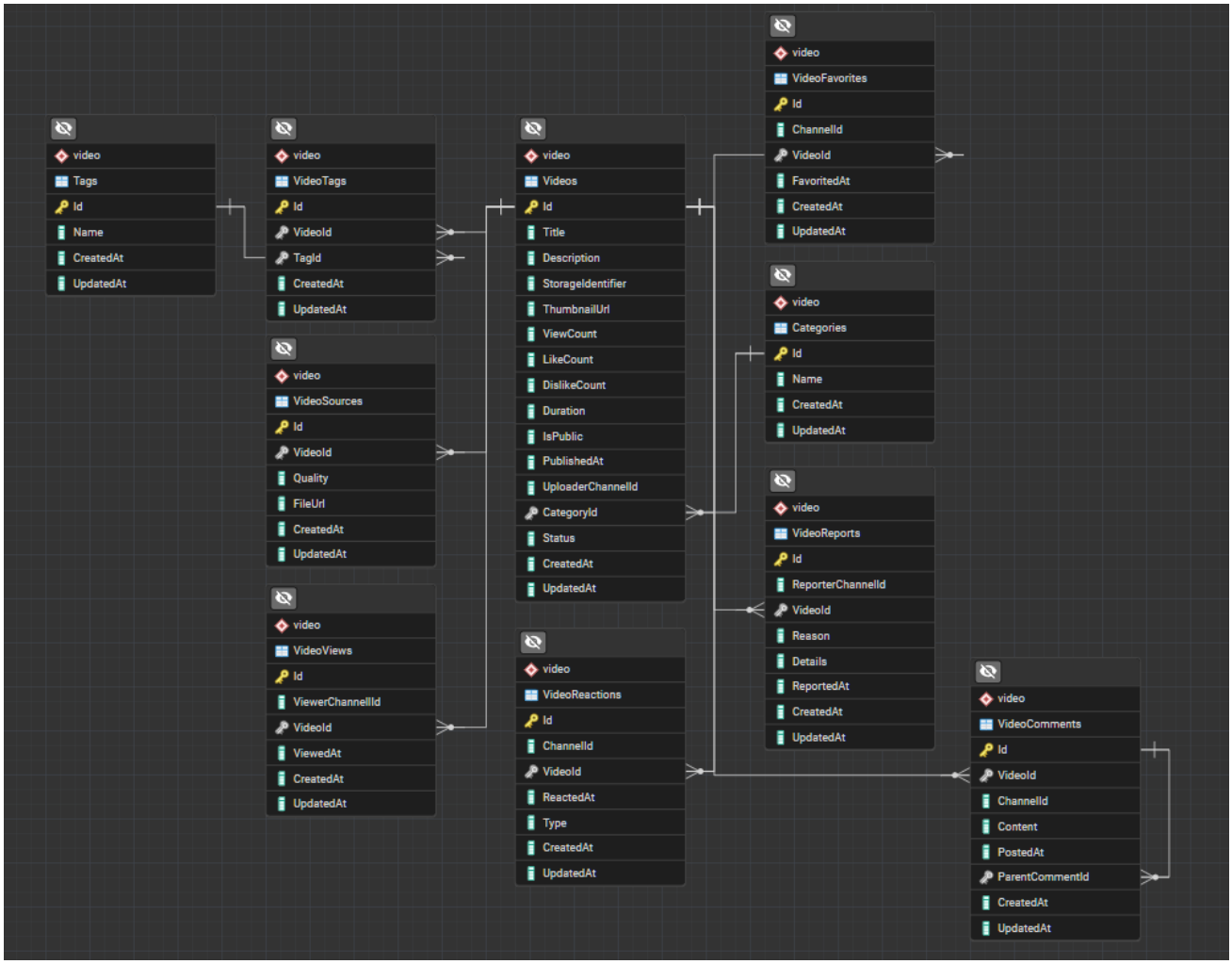

Рисунок 3.2 – ER-діаграма для Video Service

Таким чином, спроєктована структура бази даних дозволяє ефективно керувати життєвим циклом медіаконтенту, забезпечуючи високу швидкість пошуку та надійне збереження всіх інтерактивних взаємодій користувачів.

3.3.2 Домен соціальної взаємодії (ChannelService)

Головна задача ChannelService це організувати контент навколо авторів та налагодити їхній зв’язок з аудиторією.

Центральним елементом цього домену виступає сутність Channel, що представляє публічний профіль автора або цілої спільноти, де концентруються всі підписки та взаємодії. Для спільного управління ресурсом розроблено модель ChannelMember, яка пов’язує користувача з конкретною творчою студією та визначає його роль (від звичайного учасника до власника).

Для перевірки прав доступу під час редагування параметрів каналу використовуються JWT claims. Оскільки потрібна інформація про ролі користувача вже закладена в токені, перевірка доступу відбувається миттєво на рівні сервісу. Це дозволяє суттєво економити ресурси інфраструктури та позбавляє від зайвих синхронних міжсервісних запитів.

Мережа підписок на оновлення авторів реалізована через сутність Subscription. На основі цієї таблиці зв’язків будується персоналізована стрічка рекомендацій та надсилаються сповіщення. Цікавою архітектурною особливістю є те, що об'єктом підписки виступає не сам користувач, а безпосередньо його канал. Для збереження цілісності графа підписок при можливих збоях логіку каскадного видалення тут вимкнено (встановлено поведінку Restrict).

Списки відтворення описуються парою сутностей Playlist та PlaylistItem. Остання містить спеціальне поле для впорядкування відео, а комбінований індекс забезпечує швидке сортування елементів у вебплеєрі.

Структуру таблиць для Channel Service та зв’язки між ними наведено на рис. 3.3.

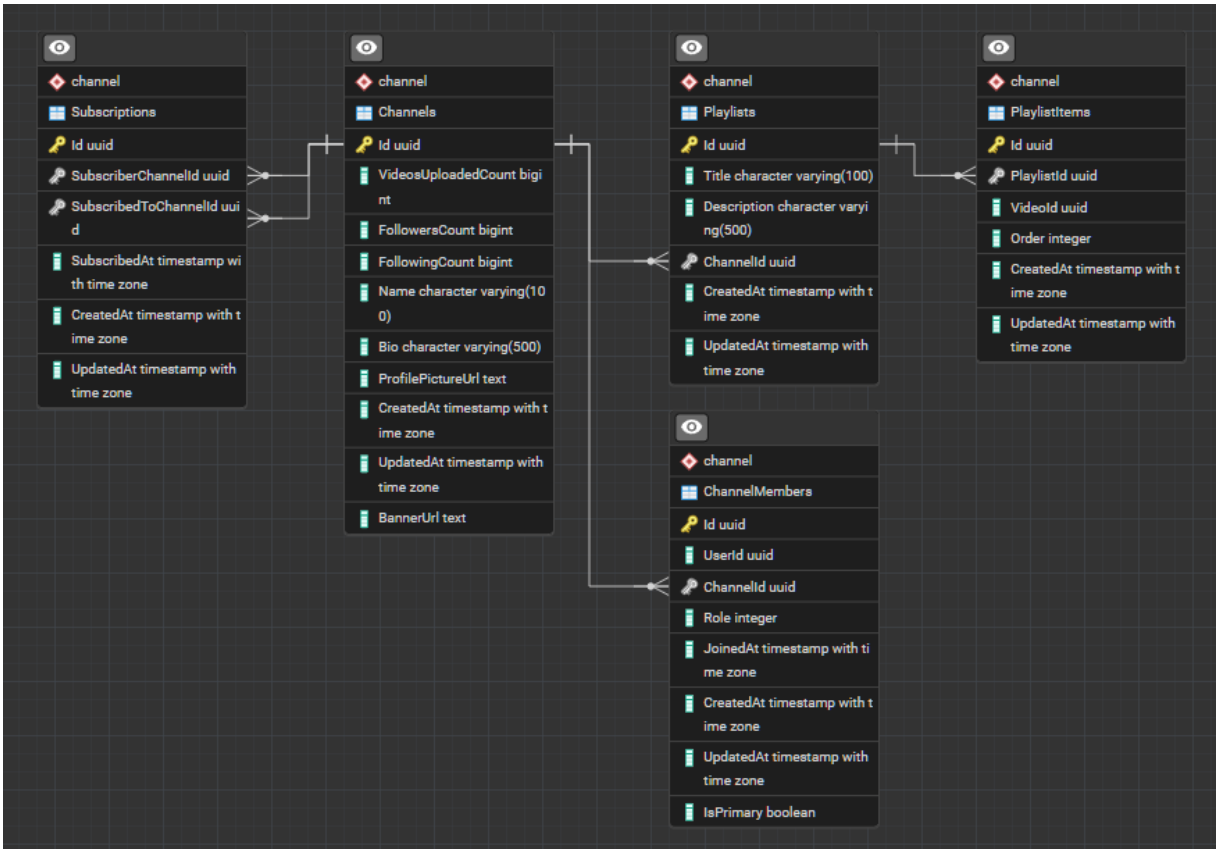

Рисунок 3.3 – ER-діаграма для Channel Service

Розроблена структура таблиць дозволяє ефективно керувати правами доступу всередині каналів та забезпечує цілісність соціальних зв'язків для формування персоналізованого контенту.

3.3.3 Домен управління користувачами (UserService)

Проєктна структура підсистеми профілів розширює базову екосистему ASP.NET Core Identity, що гарантує використання надійних індустріальних алгоритмів автентифікації, рольової авторизації та криптографічного хешування паролів.

Головна сутність домену клас User (який проєктується у таблицю AspNetUsers), що успадковує стандартний функціонал IdentityUser. Цю модель було розширено додатковими атрибутами для аудиту: мітками часу створення (CreatedAt) та модифікації запису (UpdatedAt).

Для забезпечення безпечних довготривалих сесій без необхідності постійного повторного введення облікових даних реалізовано сутність UserRefreshToken (таблиця RefreshTokens). Вона містить безпосереднє значення криптографічного токена, дату закінчення його валідності (ExpiresAt) та маркер дострокового відкликання сесії (IsRevoked).

Архітектура бази даних розроблена таким чином, що один користувач може мати декілька активних сесій одночасно (наприклад, з мобільного пристрою та десктопного веббраузера). Усі допоміжні сутності екосистеми Identity, включаючи призначення ролей (AspNetUserRoles), зовнішні логіни (AspNetUserLogins) та токени (AspNetUserTokens), об'єднані чіткими зв'язками з головною таблицею користувачів. При видаленні облікового запису всі залежні від нього дані автоматично зачищаються завдяки налаштованому механізму каскадного видалення на рівні СУБД, що запобігає появі сирітських записів.

Структуру таблиць для User Service та зв’язки між ними наведено на рис. 3.4.

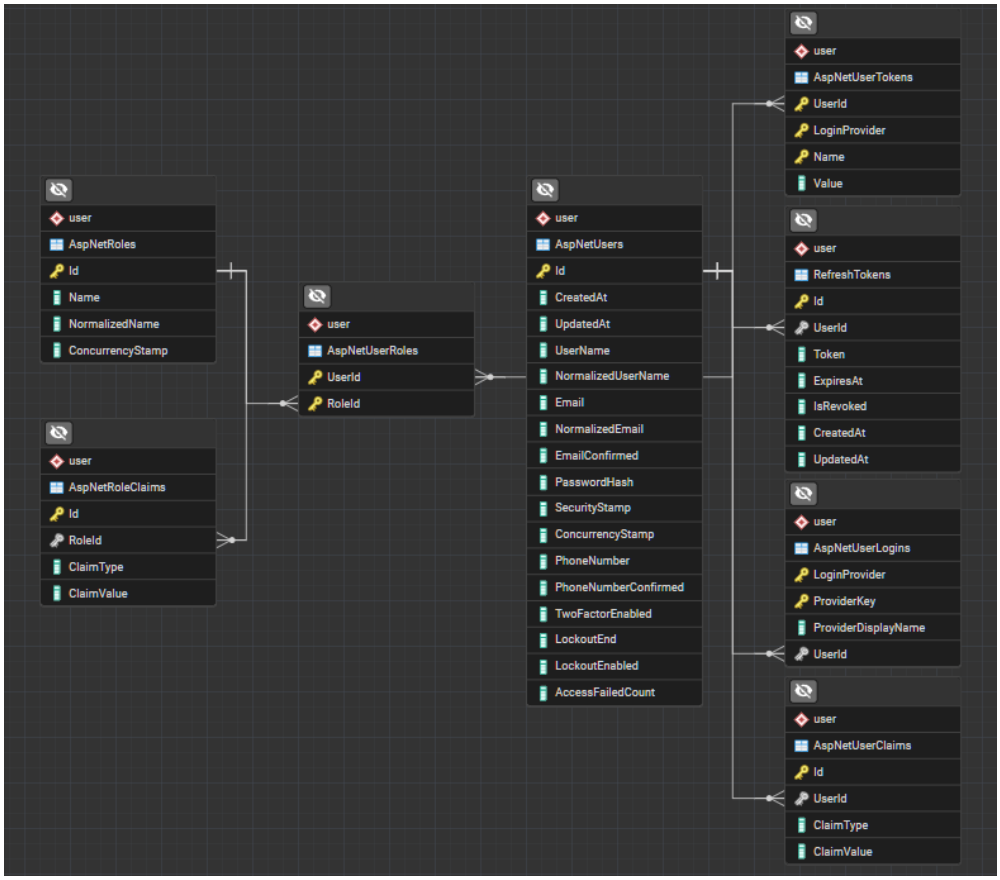

Рисунок 3.4 – ER-діаграма для User Service

Спроєктована схема повністю задовольняє вимоги безпеки щодо захисту персональних даних, автентифікації користувачів та надійного збереження сесійних маркерів в ізольованому контурі мікросервісу.

3.3.4 Переваги обраної архітектури

Досягнення високого рівня автономності сервісів було основним завданням доменної декомпозиції системи. Завдяки тому, що кожен мікросервіс має власну бізнес-логіку, інтерфейси взаємодії (API) та незалежне сховище даних, було мінімалізовано зв’язність між компонентами системи.

Особливістю мікросервісної архітектури є можливість точкового масштабування. VideoService постійно працює з «важкими» завданнями під час обробки файлів, тому потребує більше потужностей. Але ChannelService працює з легкими операціями. Завдяки їх розділенню ресурси серверів можна направляти саме туди, де вони критично необхідні в даний момент.

Використання доменно-орієнтованого підходу в рамках мікросервісної інфраструктури дозволило досягти високої модульності застосунку. Завдяки незалежності сервісів, впровадження нових функціональних можливостей не порушує цілісність інших компонентів системи, що значно спрощує підтримку та забезпечує її стійкість до динамічних змін рівня навантаження.

На рис. 3.5 наведено приклад мікросервісної архітектури платформи для розміщення відеоконтенту.

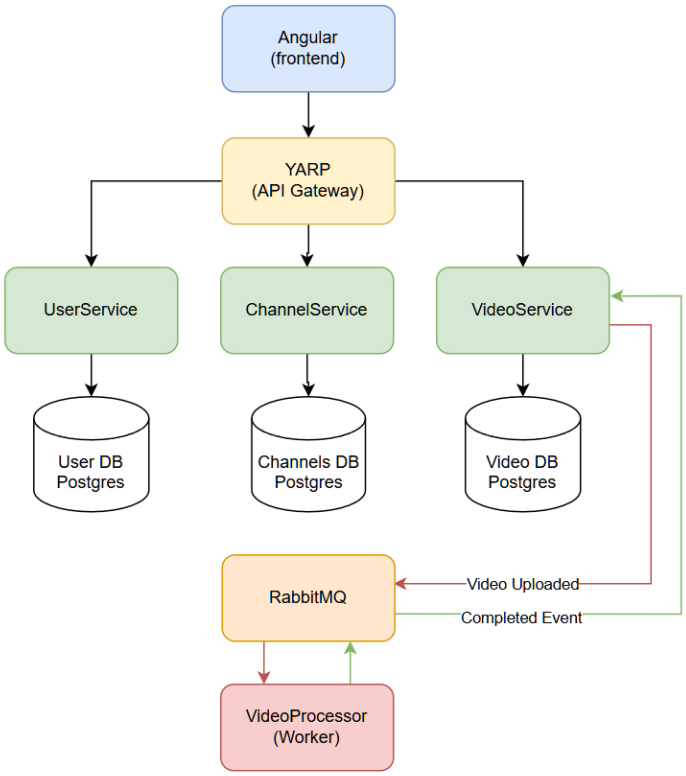

Рисунок 3.5 - Схема взаємодії мікросервісів через API Gateway та RabbitMQ

Архітектура платформи побудована на основі мікросервісного підходу з використанням єдиної точки входу у вигляді API Gateway, реалізованого на базі YARP (Yet Another Reverse Proxy). API Gateway виконує роль центрального маршрутизатора, який приймає HTTP-запити від клієнтського Angular SPA та перенаправляє їх до відповідних сервісів залежно від типу операції.

У системі функціонують три основні незалежні сервіси:

– UserService відповідає за автентифікацію, авторизацію, генерацію JWT-токенів та управління обліковими записами користувачів;

– ChannelService реалізує логіку роботи з каналами, підписками та організацією контенту;

– VideoService забезпечує завантаження, зберігання, обробку відео та підтримку системи коментарів.

Кожен із сервісів використовує власну базу даних PostgreSQL, що забезпечує повну ізоляцію даних та відповідає принципу Database per Service. Такий підхід

підвищує незалежність компонентів та спрощує процес горизонтального масштабування.

Для реалізації асинхронної обробки відеофайлів використовується брокер повідомлень RabbitMQ. Після завантаження файлу VideoService публікує подію VideoUploadedEvent у чергу video-uploads. Далі спеціалізований сервіс VideoProcessor.Worker споживає повідомлення, виконує перекодування відео у формат 720p та генерує thumbnail-зображення за допомогою FFmpeg.

По завершенню транскодування воркер надсилає VideoProcessingCompletedEvent у чергу video-results. Отримавши це повідомлення, VideoService перемикає статус відео на «Ready» і відкриває доступ до контенту для кінцевих користувачів.

Для забезпечення інтерактивної взаємодії та підтримки функціоналу реального часу у системі використовується SignalR. Через дану технологію клієнт отримує оновлення лічильників переглядів, зміни статусу відео та інші події без необхідності перезавантаження сторінки.

Авторизація побудована на JWT-токенах. Оскільки сервіси валідують їх локально, додаткові запити до UserService при кожному зверненні стають непотрібними. Це мінімізує мережеві затримки та знижує навантаження на систему безпеки.

Описана архітектура дозволяє VidPortal працювати з високим навантаженням та масштабувати окремі вузли без зупинки всієї системи. Така модульність спрощує підтримку платформи та дозволяє поступово додавати нові функції, не змінюючи структуру існуючих сервісів.

3.4 Об'єктно-орієнтоване проєктування системи за допомогою UML

Для візуалізації архітектурних рішень та моделювання процесів взаємодії користувачів із платформою VidPortal було використано уніфіковану мову моделювання (UML). Це дозволило стандартизувати опис структури програмного забезпечення на різних рівнях абстракції.

Діаграма прецедентів описує функціональні можливості системи з точки зору кінцевого користувача та взаємодію між зовнішніми акторами і системними процесами. Основні актори системи: неавторизований користувач (Гість), Авторизований користувач (Автор) та Фоновий обробник (Worker).

Актор «Гість» має базовий доступ до пошуку та перегляду відкритого відеоконтенту. Після проходження процедури автентифікації користувач отримує розширені права: можливість завантажувати медіафайли, створювати канали, залишати коментарі та керувати реакціями. Актор «Фоновий обробник» є внутрішнім системним суб'єктом, що відповідає за асинхронне транскодування файлів.

Діаграма варіантів використання системи зображена на рис. 3.6.

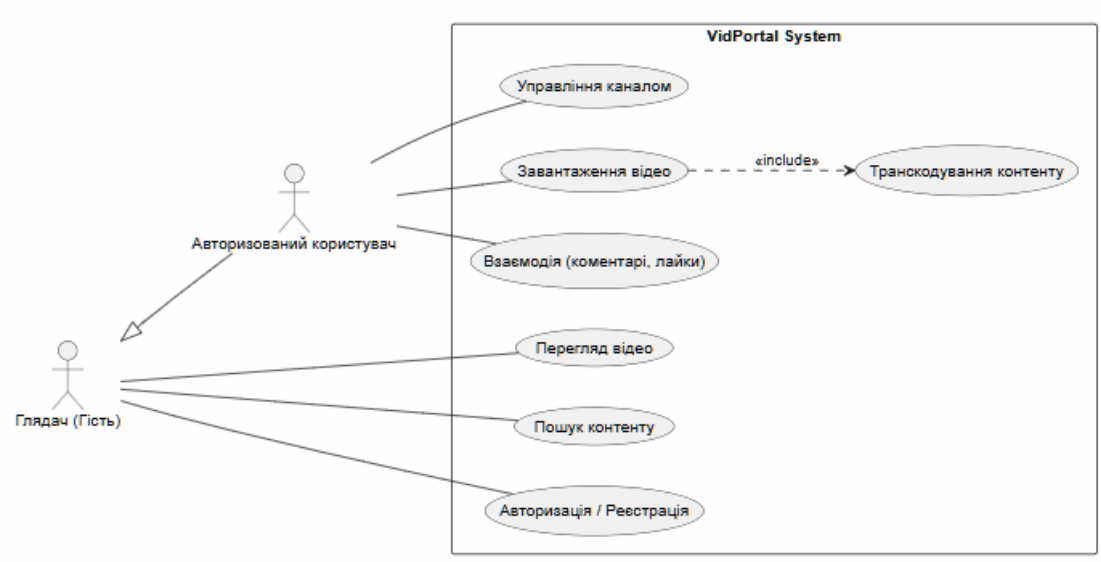

Рисунок 3.6 - Діаграма варіантів використання системи

Для демонстрації об'єктно-орієнтованої структури бази даних та доменних сутностей побудовано діаграму класів на прикладі ключового мікросервісу, VideoService. Цей модуль відповідає за життєвий цикл медіаконтенту.

Центральною сутністю є клас Video, який агрегує текстові метадані, лічильники популярності та ідентифікатор файлу у сховищі. Сутність має зв'язки типу «один-до-багатьох» із класами VideoComment (деревоподібні обговорення),

VideoReaction (оцінки глядачів) та VideoSource (доступні роздільні здатності для адаптивного стрімінгу).

Діаграма класів для мікросервісу VideoService зображена на рис. 3.7.

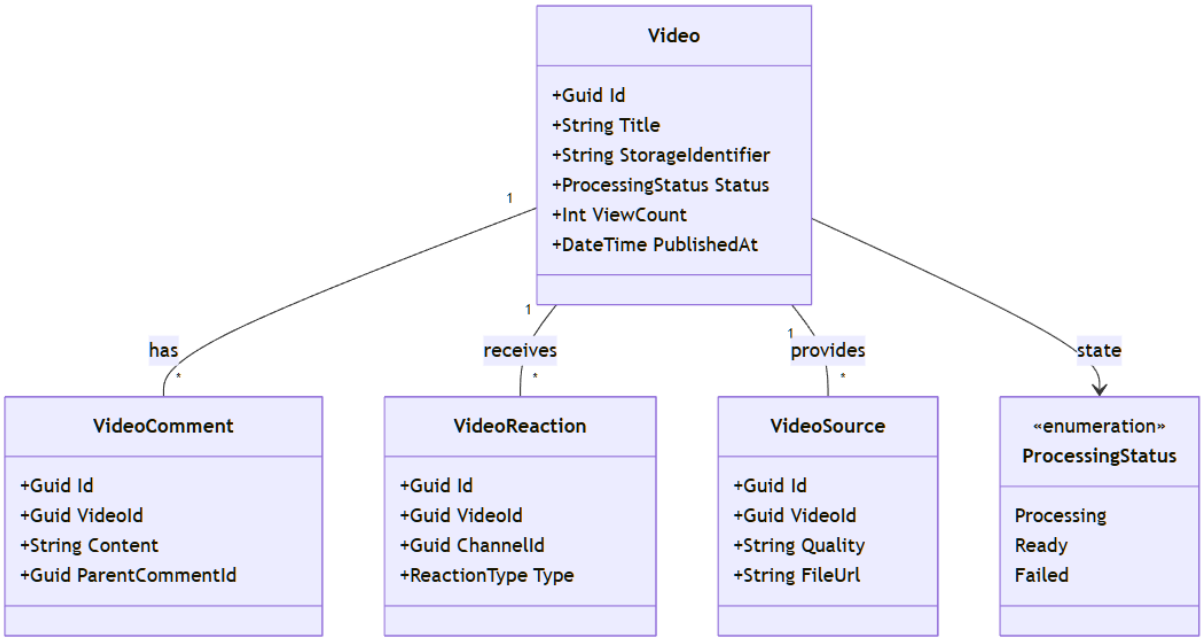

Рисунок 3.7 - Діаграма класів для мікросервісу VideoService

Діаграма компонентів відображає структурну декомпозицію платформи VidPortal на незалежні мікросервіси та логіку їх взаємодії.

Єдиною точкою входу для клієнтського Angular-застосунку є API Gateway на базі YARP, який розподіляє HTTP-трафік та WebSocket-з'єднання (SignalR). Бекенд розділено на три незалежні сервіси (UserService, ChannelService, VideoService), кожен з яких працює з власною базою даних PostgreSQL (патерн Database per Service). Для забезпечення слабкої пов'язаності (loose coupling) при виконанні ресурсомістких завдань VideoService взаємодіє з ізольованим компонентом VideoProcessor виключно через брокер повідомлень RabbitMQ.

Діаграма компонентів платформи зображена на рис. 3.8.

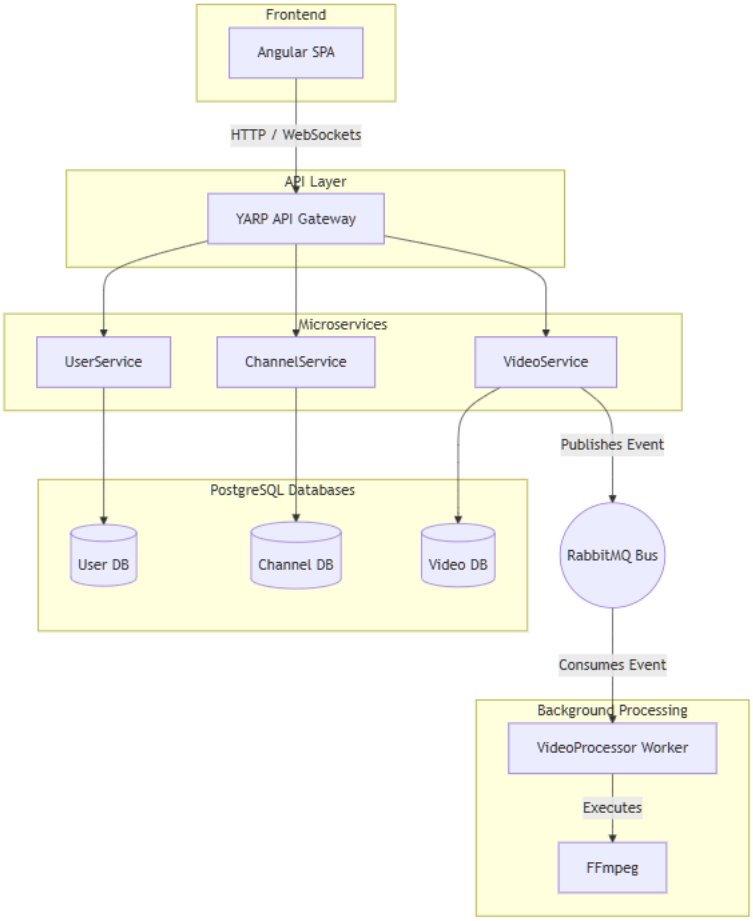

Рисунок 3.8 - Діаграма компонентів платформи

Таким чином, проведення об'єктно-орієнтованого проєктування за допомогою мови UML дозволило сформувати цілісне та несуперечливе уявлення про архітектуру платформи VidPortal, деталізувати логіку взаємодії користувачів із системою та зафіксувати інтерфейси взаємодії між автономними мікросервісами. Створені моделі прецедентів, доменних сутностей VideoService та загальної топології компонентів заклали надійний теоретичний фундамент для подальшої практичної інженерної реалізації, конфігурації баз даних та асинхронних черг повідомлень.

Висновки до розділу 3

У третьому розділі було детально розглянуто процес проєктування та програмної реалізації платформи VidPortal, особливості побудови її архітектури та

механізми взаємодії між компонентами системи. Основну увагу було приділено питанням створення відмовостійкої мікросервісної структури та забезпечення інтерактивної взаємодії в режимі реального часу. На основі виконаної роботи можна зробити такі висновки:

- проєкт використовує .NET 9 та Angular 21. Для зв'язки мікросервісів та моніторингу обрано .NET Aspire. Це значно спростило оркестрацію та налаштування дебагу в реальному часі;
- розроблено комплекс UML-діаграм (прецедентів, класів та компонентів) , що дозволило формалізувати функціональні вимоги до платформи, типізувати доменні моделі мікросервісу VideoService та зафіксувати топологію міжсервісної взаємодії;
- кожен сервіс (VideoService, ChannelService, UserService) має власну базу PostgreSQL. Така ізоляція даних дозволяє масштабувати окремі домени незалежно одне від одного. Маршрутизація запитів робиться через YARP (API Gateway), що забезпечило стабільну роботу вхідного API;
- операції з транскодування через FFmpeg винесені у фоновий воркер VideoProcessor. Для передачі завдань використовується RabbitMQ. Це зняло основне навантаження з API та гарантує стабільність системи навіть при масовій обробці відео;
- інтерфейс реалізовано як SPA на Angular з використанням Signals та Standalone Components. Інтерактивність забезпечує SignalR (оновлення даних без оновлення сторінки) , а верстка Tailwind CSS v4 гарантує адаптивність та швидку підтримку стилів при додаванні нового функціоналу.

Загалом, поєднання аналітичного UML-проєктування та обраної мікросервісної архітектури демонструє високу ефективність для роботи з мультимедійними даними , мінімізує ризики каскадних збоїв і формує надійний, гнучкий фундамент для подальшого масштабування системи за умов зростання обсягів контенту та кількості користувачів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ СИСТЕМИ ВІДЕОХОСТИНГУ

У цьому розділі описано процес програмної реалізації відеохостингу. Розглянуто архітектуру програмного забезпечення, розробленого на платформі .NET, а також структуру та реалізацію бази даних. Крім того, наведено особливості роботи модулів асинхронної обробки відеоконтенту. Також показано, як було реалізовано інтерфейс користувача та організовано взаємодію між основними компонентами системи.

4.1 Програмна реалізація інфраструктури баз даних

У процесі розробки системи відеохостингу VidPortal архітектура збереження даних реалізовувалася з урахуванням принципів розподілених систем. Як основний інструмент об’єктно-реляційного відображення (ORM) використано Entity Framework Core у поєднанні зі СУБД PostgreSQL.

У межах проєкту застосовано підхід Code First: структура бази описується через класи доменних сутностей мовою C#, а зв'язки, індекси та обмеження налаштовуються за допомогою Fluent API у методі OnModelCreating. На основі цих конфігурацій EF Core автоматично формує міграції, які генерують фізичну структуру таблиць.

Для уникнення дублювання коду всі доменні сутності успадковують базовий абстрактний клас BaseEntity, який містить обов’язкові атрибути аудиту: первинний ключ Id типу Guid та поля часових міток CreatedAt і UpdatedAt.

Оскільки серверна частина реалізована як набір незалежних мікросервісів, взаємодія з базою даних логічно та фізично розподілена між трьома окремими класами контексту (DbContext). Кожен із них за допомогою налаштувань EF Core ізольований у власній схемі бази даних:

- UserDbContext базується на стандартному інструментарії ASP.NET Core Identity. Контекст працює зі схемою user, керуючи базовими таблицями

безпеки та кастомною таблицею RefreshTokens. У конфігурації останньої через Fluent API прописано унікальний індекс для токена, щоб унеможливити дублювання сесій;

– ChannelDbContext закріплений за схемою channel. Обслуговує класи створення каналів, налаштування прав доступу для команд авторів, обробку підписок та структурування плейлистів;

– VideoDbContext найбільш навантажений контекст системи, що обслуговує схему video. Він зберігає метадані відеофайлів, інформацію про доступні роздільні здатності та обробляє статистику взаємодії користувачів із контентом (коментарі, реакції).

Для автоматизації обліку часу в системі було перевизначено метод SaveChangesAsync у всіх перелічених контекстах бази даних. Перед відправленням SQL-запиту до PostgreSQL, внутрішній механізм EF Core (Change Tracker) самостійно сканує змінені об’єкти в пам'яті та автоматично встановлює актуальні дати в поля CreatedAt (при додаванні нового запису) або UpdatedAt (при модифікації існуючого). Це знімає рутинне навантаження з бізнес-логіки на рівні сервісів та гарантує єдиний стандарт аудиту записів по всій системі.

4.2 Опис програмної реалізації бекенд-частини

Серверна екосистема платформи відеохостингу VidPortal розроблена на базі кросплатформного фреймворку .NET та мови програмування C#. Замість традиційного монолітного підходу архітектура побудована навколо сервісно-орієнтованої декомпозиції, де кожен мікросервіс ізолює власну бізнес-логіку та шар збереження даних. Таке рішення спрощує кодову базу, усуває жорсткі зв’язки між модулями та закладає надійний фундамент для незалежного горизонтального масштабування окремих компонентів системи під високими навантаженнями.

Єдиною точкою входу для клієнтського Angular-застосунку є API Gateway, реалізований за допомогою YARP Reverse Proxy (VidPortal.ApiGateway). Шлюз повністю приховує внутрішню топологію мережі мікросервісів, розподіляючи

вхідний HTTP-трафік за відповідними внутрішніми адресами (маршрути типу /api/users/* проксуються до сервісу користувачів, а /api/v/* до медіасервісу). Окрім маршрутизації, на рівні шлюзу сконфігуровано глобальну CORS-політику та механізми перевірки безпеки.

Система використовує JWT Bearer-токени. Після перевірки автентичності на API-шлюзі ідентифікатор користувача передається у внутрішні сервіси через HTTP-заголовок X-User-Id. Це спрощує обробку запитів на рівні доменів.

Контроль прав доступу базується на Claims-Based авторизації та доменних політиках. Такий механізм обмежує доступ до адміністративних інструментів та налаштувань студій авторів.

Комунікація між ізольованими компонентами системи розділена на два типи:

- синхронний підхід використовується якщо сервіс потребує негайної відповіді від іншого сервісу. Для цього використовується HttpClient. Наприклад, VideoService запитує в ChannelService права на публікацію контенту, очікуючи на результат у реальному часі;
- асинхронний підхід реалізований через брокер повідомлень RabbitMQ. Застосовується для завдань які не потребують негайної відповіді. Також це дозволяє надійно передавати дані між сервісами.

Прикладом асинхронної взаємодії є обробка відео. Коли файл завантажується на сервер, система створює подію для RabbitMQ. Її підхоплює фоновий воркер, який за допомогою FFmpeg виконує транскодування, перетворює відео у потрібні формати та роздільні здатності.

Оскільки ці складні завдання обробляються у черзі, API не блокується. Це дозволяє інтерфейсу залишатися швидким для користувача, а системі стабільною.

4.2.1 Функціональна реалізація ключових доменних мікросервісів

Програмна логіка платформи розподілена між трьома основними сервісами, кожен з яких використовує власні асинхронні контексти EF Core для роботи зі своїми схемами у PostgreSQL.

Мікросервіс UserService забезпечує повний життєвий цикл облікових записів. Він інтегрований із модулем ASP.NET Core Identity, що гарантує надійне збереження криптографічних хешів паролів та безпечне керування ролями (Admin, User). Окрім стандартних операцій реєстрації та входу, сервіс реалізує механізм ротації сесійних маркерів через пару токенів: короткоживучий access-токен та довготривалий refresh-токен, унікальність якого контролюється окремим індексом на рівні бази даних.

Домен соціальної взаємодії зосереджений у мікросервісі ChannelService. Він оперує логікою створення авторських просторів, налаштуванням їхніх банерів та аватарів, а також обробкою підписок та користувацьких плейлистів. Важливою частиною сервісу є модель розподілу прав усередині каналу (Owner, Manager, Moderator, Member). Всі перевірки прав доступу виконуються через спеціалізовані політики авторизації (наприклад, CanManageMembers або CanUpdateChannel), що дозволяє гнучко керувати великими командними медіаресурсами.

VideoService – основний сервіс для роботи з контентом. Він відповідає за завантаження відео (multipart-запити), зберігає метадані та формує видачу для головної сторінки, трендів і пошуку. Оновлення даних у реальному часі (наприклад, лічильники) працює через SignalR-хаб VideoHub, тому сторінка не потребує перезавантаження.

Оскільки транскодування відео та генерація прев’ю занадто навантажують сервер, VideoService не виконує ці операції самостійно. Він лише приймає оригінальний файл і передає завдання на обробку в ізольовану підсистему воркерів через RabbitMQ. Такий поділ дозволяє уникнути блокування основного API та забезпечити стабільну роботу системи під навантаженням.

4.3 Реалізація модуля обробки відео та черг повідомлень

У системі відеохостингу VidPortal процес кодування та обробки медіаконтенту виділено в ізольований асинхронний модуль, що функціонує незалежно від основного API. Таке архітектурне рішення зумовлене високою

ресурсомісткістю операцій транскодування відеофайлів та генерації графічних мініатюр. Виконання цих процесів у синхронному режимі призвело б до блокування HTTP-запитів клієнтів та вичерпання пулу потоків вебсервера.

Для забезпечення надійної комунікації між сервісами в умовах розподіленого середовища інтегровано брокер повідомлень RabbitMQ. Алгоритм роботи побудовано за принципом відкладеного виконання. Після завантаження файлу медіасервіс фіксує метадані в базі, зберігає оригінал у файловому сховищі та делегує подальшу обробку, публікуючи відповідну подію в чергу. Автономний фоновий процес (воркер) перехоплює це завдання, виконує ресурсомісткі перетворення за допомогою утиліти FFmpeg і повертає результати через зворотну чергу.

Модуль обробки відео консолідує кілька взаємопов’язаних інфраструктурних та програмних компонентів:

- VideoService.API транспортний шар, що приймає multipart-запити на завантаження контенту;
- VideoService.Infrastructure шар бізнес-логіки, який керує збереженням оригіналів та ініціює події транскодування;
- VideoProcessor.Worker виділений фоновий сервіс, що інкапсулює логіку взаємодії з FFmpeg;
- BuildingBlocks.Infrastructure ядро міжсервісної комунікації, що містить універсальну імплементацію шини повідомлень;
- EventBus.Messages бібліотека спільних контрактів обміну даними (Event Contracts);
- RabbitMQ брокер, що гарантує доставку повідомлень та балансування навантаження.

Точкою входу процесу є контролер UploadController. Запит формату multipart/form-data включає фізичний бінарний файл та структурований JSON об’єкт із метаданими (назва, опис, теги, ідентифікатор каналу автора). Перед початком запису на диск система звертається до сервісу каналів через інтерфейс

IChannelVideoManagementAccessVerifier для підтвердження прав користувача на публікацію.

Після успішної валідації система створює сутність Video і зберігає оригінальний файл у локальному сховищі (wwwroot/uploads/videos/{videoId}/original.{extension}). Шлях до файлу фіксується у полі StorageIdentifier бази даних.

Для чіткого відстеження життєвого циклу контенту застосовується статусна модель (перелік VideoStatus):

- Processing: файл успішно прийнято, триває фонове перекодування (стартовий стан);
- Ready: оптимізацію завершено, ролик доступний для трансляції;
- Failed: виникла критична помилка під час роботи FFmpeg;
- Removed: контент вилучено з публічного доступу.

Взаємодія базується на двох ключових подіях. Подія VideoUploadedEvent відправляється у чергу video-uploads та містить навігаційні дані: ідентифікатор відео, відносний шлях до оригіналу та технічні вимоги до кодування. Після завершення операції генерується подія VideoProcessingCompletedEvent для черги video-results, яка містить оновлену тривалість, шлях до згенерованої мініатюри та масив об’єктів EncodedVideoFile із даними про нові відеопотоки.

Шар абстракції IMessageBus інкапсулює роботу з пакетом RabbitMQ.Client. Для забезпечення відмовостійкості черги декларуються з параметром durable: true (збереження на диску). Обробка повідомлень виконується у режимі ручного підтвердження (Manual Acknowledgment). За умови успішного виконання надсилається сигнал BasicAckAsync, а в разі технічного збою - BasicNackAsync(requeue: true), що повертає завдання в чергу для повторної спроби.

Проєкт VideoProcessor.Worker реалізовано на базі шаблону BackgroundService. Оскільки у середовищі Docker використовується спільний

логічний диск (shared volume vidportal_uploads), воркер має прямий доступ до файлів, завантажених медіасервісом, за відносними шляхами.

Безпосередня трансформація делегована сервісу VideoEncoder, що інтегрує бібліотеку FFMpegCore. Алгоритм виконує два паралельні завдання:

- вилучення кадру на першій секунді для створення мініатюри (thumbnail.jpg) з масштабуванням до 1280x720 пікселів;
- перекодування відеоряду в контейнер MP4. Для забезпечення максимальної сумісності з вебплеєрами застосовується відеокодек H.264 та аудіокодек AAC. Використання прапорця faststart оптимізує структуру файлу для миттєвого відтворення відео у браузері ще до повного завантаження (псевдострімінг).

Після публікації результату воркером, фоновий процес медіасервісу (VideoServiceMessagingHostedService) перехоплює подію. Механізм VideoProcessingCompletionHandler аналізує статус виконання, у разі помилки відео переходить у стан Failed, що дозволяє інтерфейсу повідомити автора про збій. У разі успіху відео маркується як Ready, база даних оновлюється інформацією про реальну тривалість, а в таблицю VideoSources додаються записи з прямими URL-адресами до згенерованих файлів якості 720p.

Архітектура підсистеми обробки відео повністю відповідає сучасним стандартам побудови високонавантажених платформ. Розв’язка процесів через RabbitMQ гарантує швидкий відгук клієнтського API, захищає сервери від перевантаження ресурсомісткими задачами FFmpeg та створює надійну основу для подальшого масштабування конвеєра (наприклад, додавання алгоритмів генерації HLS-потоків чи додаткових профілів роздільної здатності).

4.4 Реалізація інтерфейсу користувача

Фронтенд VidPortal – це SPA-застосунок на Angular 21. Така архітектура дозволяє оновлювати контент та навігацію без перезавантаження сторінки, що робить інтерфейс швидким і плавним.

Для розробки обрано:

- TypeScript: для типізації коду;
- RxJS: для роботи з асинхронними потоками (HTTP, WebSocket);
- Angular Signals: для реактивного керування станом компонентів;
- Tailwind CSS v4: для адаптивної верстки та стилізації.

Цей стек забезпечує швидку роботу інтерфейсу та дозволяє легко додавати нові модулі в майбутньому.

4.4.1 Модульна організація та маршрутизація

Проєкт структуровано за принципом функціонального поділу (Feature-Based Structure), що спрощує навігацію кодовою базою та її подальше масштабування:

- Core: містить базову інфраструктуру (HTTP-сервіси взаємодії з бекендом, моделі DTO, механізми перехоплення запитів (interceptors) та керування токенами JWT);
- Features: ізольовані функціональні модулі сторінок (авторизація, перегляд відео, пошук, завантаження контенту, керування творчою студією);
- Shared: набір перевикористовуваних UI-компонентів (картки відео, кнопки, форми, діалогові вікна), які гарантують єдиний візуальний стиль системи.

Система навігації побудована на базі Angular Router із застосуванням технології лінивого завантаження (Lazy Loading). Для захисту приватних розділів застосовуються Route Guards, які перевіряють наявність access-токена.

Уся взаємодія клієнтської частини з мікросервісами інкапсульована у спеціалізовані класи (VideoApiService, ChannelApiService тощо). Для забезпечення безперебійної авторизації налаштовано HTTP Interceptors, які автоматично додають токени до заголовків та обробляють сценарії оновлення сесії (refresh token). Централізована обробка помилок дозволяє перехоплювати збої та виводити їх користувачу через глобальну систему сповіщень. Для оптимізації рендерингу використовується ChangeDetection Strategy.OnPush та Angular Signals.

4.4.2 Головна сторінка та взаємодія з контентом

Інтерфейс платформи огорнутий у глобальний макет (Shell), що містить фіксовану верхню панель навігації, адаптивне бокове меню та основний контейнер для рендерингу маршрутів.

Головна сторінка (HomePage) відповідає за відображення персоналізованої стрічки рекомендацій та фільтрацію відео за системними категоріями. Контент виводиться у вигляді адаптивної сітки, яка підлаштовується під роздільну здатність екрана користувача. Базовим візуальним елементом тут виступає VideoCardComponent – універсальний компонент, що інкапсулює логіку відображення мініатюри, тривалості, назви, каналу автора та дати публікації. Цей самий компонент перевикористовується на сторінках пошуку, трендів та всередині профілів каналів.

На рис. 4.1 наведено інтерфейс головної сторінки сайту. З елементами навігації по сайту та взаємодії з контентом.

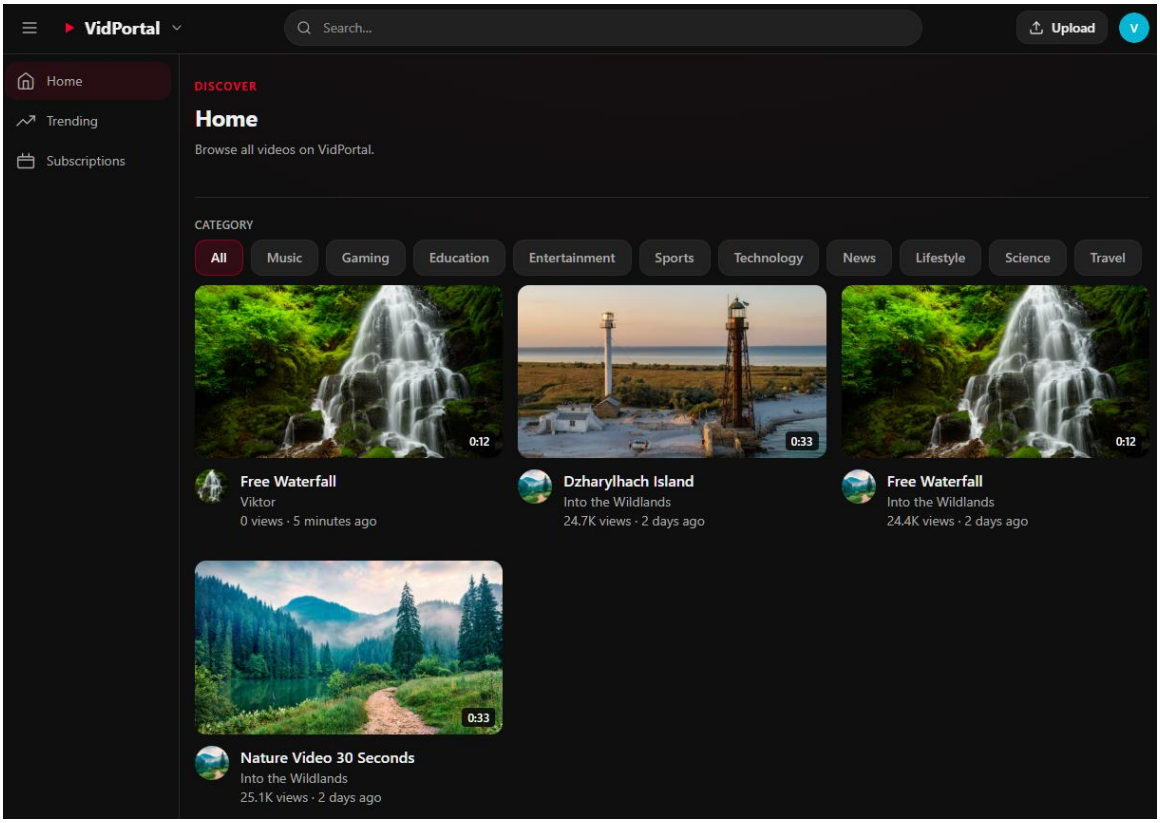

Рисунок 4.1 – Інтерфейс головної сторінки

Сторінка перегляду (WatchPage) є найскладнішим інтерактивним вузлом клієнтської частини. Для відтворення медіапотоків інтегровано популярну бібліотеку Video.js, яка обгорнута у власний Angular-компонент для забезпечення коректного життєвого циклу (ініціалізація джерел та знищення плеєра під час зміни маршруту для уникнення витоків пам’яті).

Відмінною рисою сторінки перегляду є інтеграція з SignalR. За допомогою WebSocket-підключення (VideoHubService) клієнт у реальному часі отримує оновлення лічильників переглядів, що дозволяє відображати динаміку популярності ролика без необхідності ручного оновлення сторінки браузера. Також на цій сторінці реалізовано модулі взаємодії: блок реакцій (лайки/дизлайки) та деревоподібну секцію коментарів.

На рис. 4.2 наведено інтерфейс сторінки для перегляду відео. Також на цій сторінці можна прочитати опис відео, додати коментар та лишити реакцію на відео.

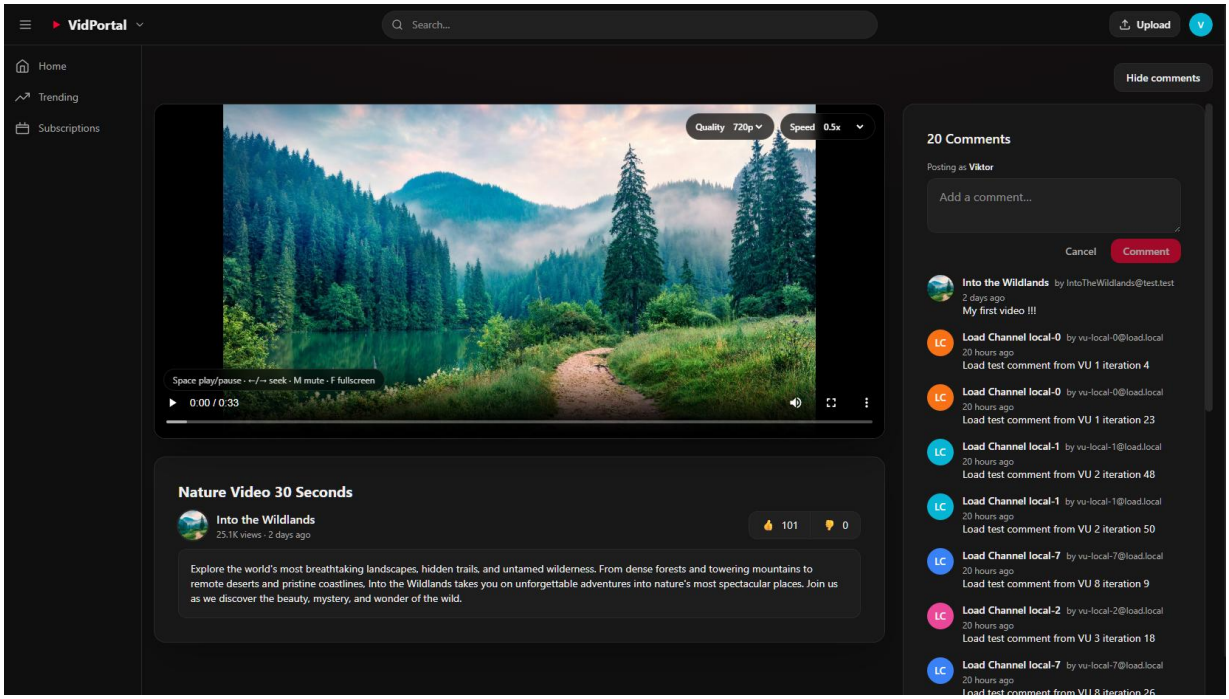

Рисунок 4.2 – Інтерфейс сторінки перегляду відеоконтенту

В результаті розробка головної сторінки та сторінки перегляду забезпечила зручний та адаптивний інтерфейс для роботи з контентом. Завдяки загальному макету (Shell) та єдиному компоненту картки відео сайт виглядає цілісно і швидко

підлаштовується під будь-які екрани. Плеєр Video.js працює стабільно і не перевантажує систему, а технологія SignalR дозволяє користувачам бачити зміну переглядів чи реакцій миттєво в реальному часі, без оновлення сторінки.

4.4.3 Керування профілем та публікація контенту

Реєстрація та вхід у систему супроводжуються строгою клієнтською валідацією за допомогою Angular Reactive Forms. Це дозволяє миттєво інформувати користувача про невідповідність паролів або неправильний формат електронної пошти ще до відправлення запиту на бекенд.

Оскільки архітектура платформи вимагає наявності каналу для будь-якої публічної активності, після першої авторизації реалізовано процес онбордингу (OnboardingPage). Це покроковий майстер, де користувач задає назву творчої студії та завантажує візуальне оформлення. Для роботи з графікою створено компонент ImageUploadZoneComponent, що підтримує технологію Drag-and-Drop та дозволяє обрізати зображення (аватар або банер) безпосередньо у браузері відповідно до заданих пропорцій (наприклад, 16:9 для банера).

Процес додавання відеоконтенту на сторінці UploadPage реалізовано у вигляді багатокрокового майстра за допомогою StepWizardComponent. Такий підхід дозволяє розділити складний процес публікації на логічні та зрозумілі етапи, покращуючи користувацький досвід.

На першому етапі користувач взаємодіє з інтерфейсом для вибору медіафайлу з локальної файлової системи. Передбачено валідацію формату та розміру файлу на стороні клієнта перед початком передачі даних.

На рис. 4.3 наведено інтерфейс для завантаження відео.

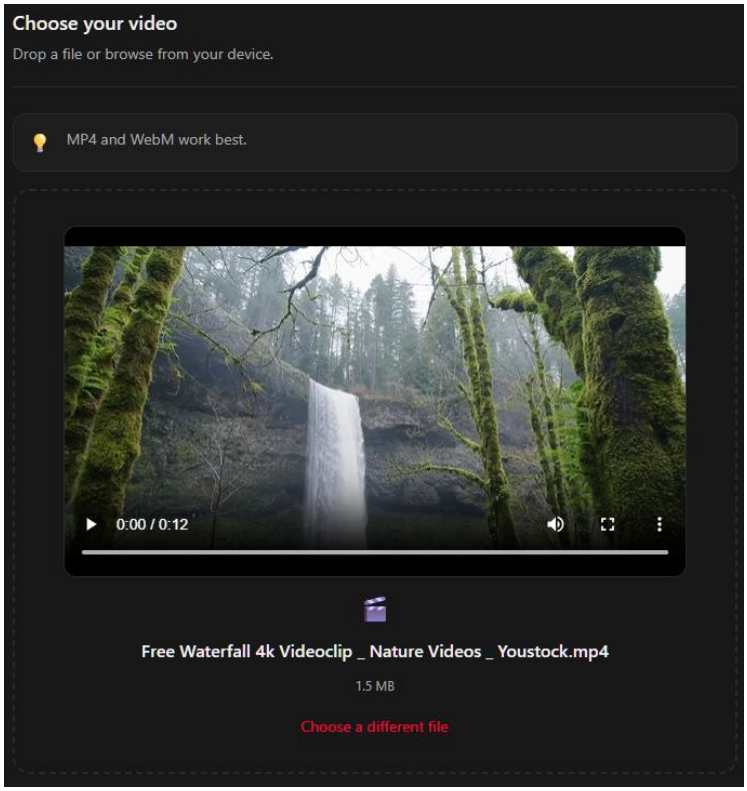

Рисунок 4.3 – Крок вибору вихідного відеофайлу

Після успішного вибору файлу користувач переходить до вводу ключової інформації про відео. Інтерфейс містить поля для введення назви, розширеного опису, а також селектори для вибору системної категорії та додавання релевантних тегів для подальшої індексації та пошуку.

На рис. 4.4 наведено інтерфейс для заповнення метаданих відео.

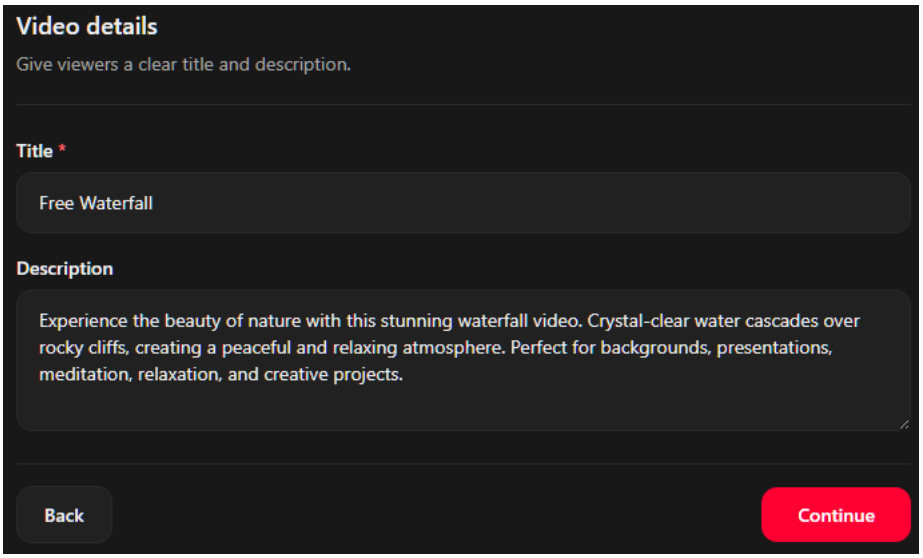

Рисунок 4.4 – Інтерфейс заповнення метаданих відео

Далі налаштовуються параметри доступу. Користувач обирає канал для публікації, налаштовує категорію відео, додає теги та встановлює рівень видимості.

На рис. 4.5 наведено інтерфейс для налаштування публікації.

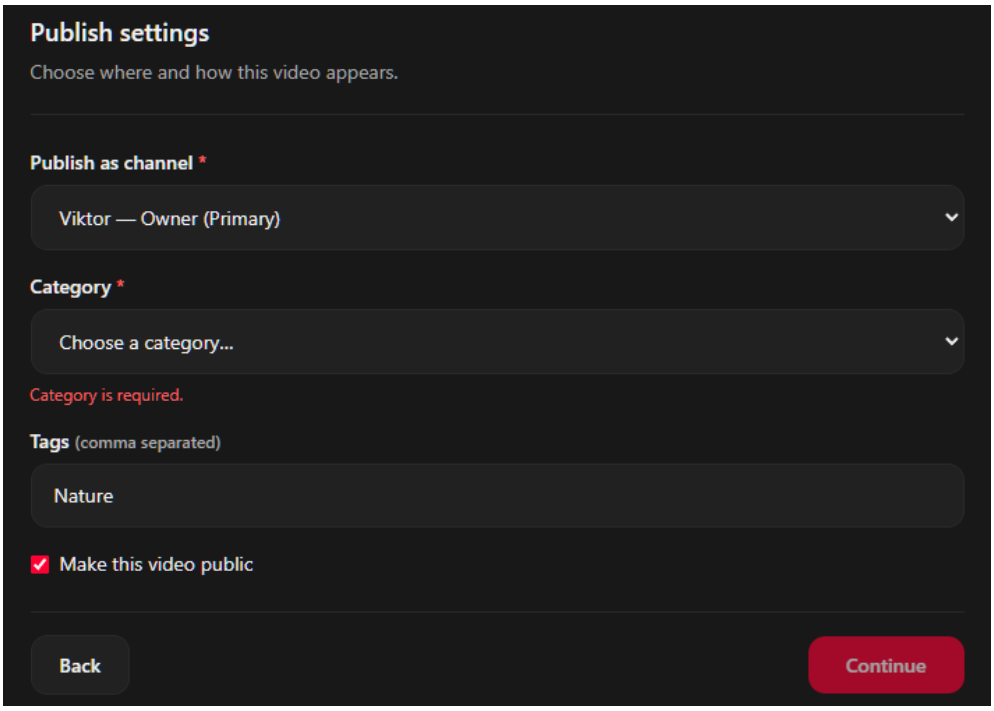

Рисунок 4.5 – Налаштування публікації та видимості

Після налаштування параметрів публікації, користувач може обрати якість в якій буде доступне відео, налаштувати пресет кодування і увімкнути функцію для нормалізації аудіо доріжки за допомогою FFmpeg утиліти.

На рис. 4.6 наведено інтерфейс для налаштування кодування відео.

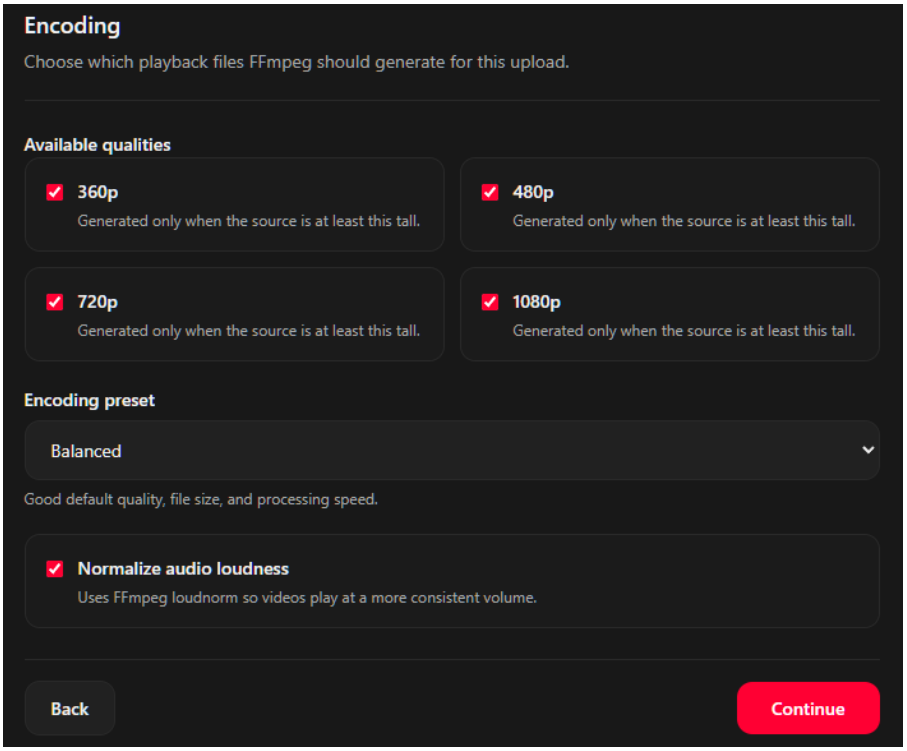

Рисунок 4.6 – Налаштування кодування відео

На наступному кроці користувач може завантажити фото для відео. Цей крок є опціональним і його можна пропустити.

На рис. 4.7 наведено інтерфейс для завантаження фото для відео.

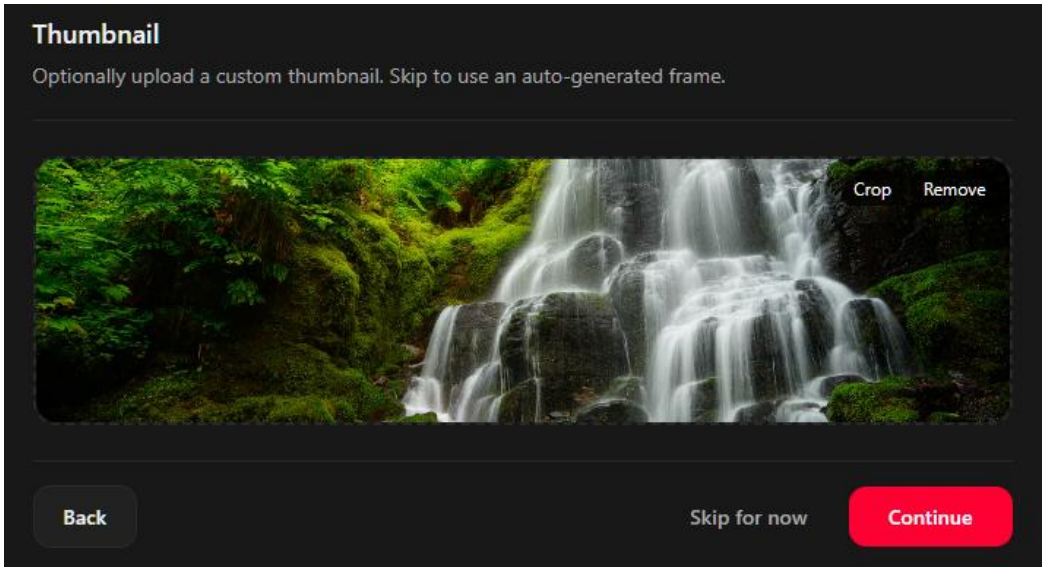

Рисунок 4.7 – Завантаження фото для відео

Після підтвердження всіх даних ініціюється процес передачі файлу. Оскільки відеофайли зазвичай мають великий об’єм, для їх передачі використовуються

Multipart-запити до серверного API. Для візуалізації цього процесу використовується ProgressBarComponent, який у реальному часі відображає відсоток завантажених даних, забезпечуючи зворотний зв'язок для користувача.

На рис. 4.8 наведено інтерфейс для перевірки даних відео.

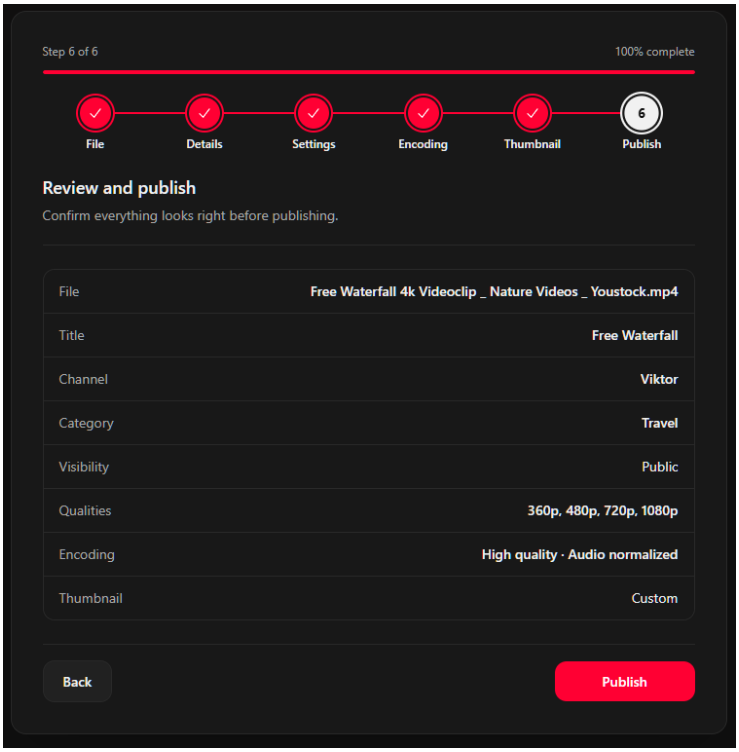

Рисунок 4.8 – Сторінка для перевірки даних відео

Таким чином, розроблений підрозділ керування профілем та публікації контенту демонструє комплексний підхід до побудови інтуїтивно зрозумілого та функціонального інтерфейсу користувача. Завдяки впровадженню строгої клієнтської валідації за допомогою Angular Reactive Forms та покрокових майстрів (зокрема StepWizardComponent), вдалося мінімізувати кількість помилок під час введення даних та розділити складний процес публікації відео на логічні етапи. Інтеграція спеціалізованих інструментів, таких як компоненти для обрізання зображень, динамічні параметри налаштування кодування відео та Multipart-завантаження з реальним відображенням прогресу, дозволила створити гнучку й інтерактивну систему, яка забезпечує високу швидкість обробки контенту та якісний користувацький досвід (UX).

4.5 Тестування та аналіз результатів

Для тестування VidPortal було проведено комплексну перевірку всіх компонентів системи. Протестовано бізнес-логіку, взаємодію між сервісами, роботу RabbitMQ та стійкість системи під навантаженням.

Перевірці підлягали всі вузли платформи:

– фронтенд: Angular застосунок;

– бекенд: YARP Gateway, мікросервіси (User, Channel, Video) та фоновий воркер;

– інфраструктура: бази даних PostgreSQL та брокер повідомлень RabbitMQ.

Цей комплексний підхід до тестування дозволив підтвердити стабільність системи та її готовність до реальних експлуатаційних навантажень.

4.5.1 Наскрізне функціональне тестування бізнес-логіки

На поточному етапі життєвого циклу проєкту основним методом валідації функціональних вимог було обрано наскрізне (end-to-end) мануальне функціональне тестування в локальному ізольованому середовищі Docker. Такий підхід дозволив детально оцінити реальну взаємодію між мікросервісами, чергами повідомлень та розподіленим файловим сховищем.

Подальшим способом покраще системи може бути впровадження unit тестів для фронтенд та бекенд частини. Фреймворки Angular і ASP .Net надають зручний інструментарії для впровадження тестів.

Процес функціонального тестування було розділено на три ключові етапи, що охоплюють основні користувацькі сценарії:

– підсистема керування доступом та каналами. Протестовано реєстрацію, авторизацію та рольову модель доступу. Також перевірено життєвий цикл каналів (створення, налаштування, управління командою) та безпеку ротації JWT-токенів;

– конвеєр завантаження та обробки відео. Протестовано приймання multipart-запитів, валідацію метаданих та збереження файлів. Підтверджено роботу асинхронної черги RabbitMQ: успішну передачу події VideoUploadedEvent, транскодування через FFmpeg та оновлення статусів у БД. Також перевірено обробку помилок: система коректно відхиляє некоректні формати або запити без прав доступу;

– клієнтський інтерфейс та соціальна взаємодія. Виконано тестування Angular-застосунку, зокрема коректності рендерингу відеострічки, роботи алгоритмів пошуку, фільтрації контенту та адаптивності інтегрованого плеєра Video.js. Окремо верифіковано стабільність WebSocket-з'єднання на базі SignalR, яке забезпечує миттєве оновлення лічильників переглядів без перезавантаження сторінки. Перевірку також пройшли модулі соціальної активності користувачів: система вкладених коментарів, механізм реакцій та функціонал підписок на канали.

Результати мануального тестування довели успішне виконання базових функціональних вимог системи та надійність обраних архітектурних рішень у реальних сценаріях використання.

4.5.2 Навантажувальне тестування та оцінка масштабованості

Для перевірки стійкості архітектури за умов інтенсивної експлуатації було проведено навантажувальні тести. Це дозволило виявити максимальні можливості поточної конфігурації та встановити межі деградації часу відгуку системи.

Як інструмент генерації навантаження було використано фреймворк Grafana k6, що дозволяє гнучко емулювати поведінку віртуальних користувачів (VU) та відстежувати часові метрики латентності на рівнях перцентилів p90, p95 та p99. Тестування проводилося через REST API на шлюзі YARP API Gateway.

У табл. 4.1 наведено перелік мікросервісних компонентів, що піддавалися автоматизованому навантаженню.

Таблиця 4.1. Об’єкти навантажувального тестування

Компонент	Функціональне призначення в архітектурі
API Gateway (YARP)	Єдина захищена точка входу, маршрутизація запитів від Angular-клієнта
User Service	Управління профілями, реєстрація, автентифікація користувачів
Video Service	Генерація стрічки, пошук, перегляд медіаконтенту, коментарі, реакції
Channel Service	Адміністрування каналів, облік підписок та рівнів авторських прав
Video Processor Worker	Асинхронна фонова обробка подій транскодування медіа
PostgreSQL 16 / RabbitMQ	Інфраструктурний шар зберігання даних та розподіленого брокеру повідомлень

Модель трафіку формувалася як змішана (mixed) з урахуванням поведінкових чинників: 75% запитів генерували анонімні користувачі (читання стрічки, пошук, перегляд), а 25% авторизовані користувачі (робота з профілем, додавання коментарів, проставлення реакцій). Між ітераціями закладалася випадкова пауза тривалістю 1–3 секунди (think time).

Перед початком основних тестів було сформовано порогові критерії прийнятності (Thresholds), які визначають успішність проходження випробувань (табл. 4.2).

Таблиця 4.2. Встановлені критерії прийнятності (Thresholds)

Контрольована метрика	Гранично допустиме значення
Рівень успішності функціональних перевірок (checks)	> 95%
Рівень помилок протоколу HTTP	< 1%
Загальний час відгуку на рівні перцентиля p95	< 800 ms
Загальний час відгуку на рівні перцентиля p99	< 1500 ms
Рівень помилок внутрішньої бізнес-логіки	< 2%

Візуалізація виконання встановлених порогових критеріїв прийнятності (Thresholds), отриманих за допомогою інструменту Grafana k6, представлена на рис. 4.9.

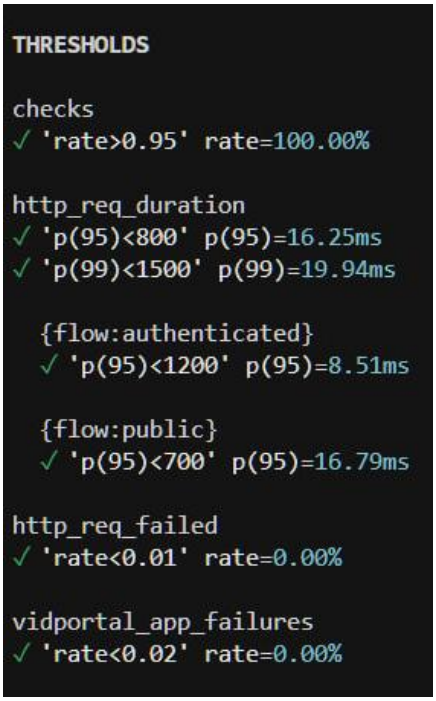

Рисунок 4.9 - Результати перевірки критеріїв прийнятності (Thresholds) під час навантажувального тестування

Розроблена методологія навантажувального тестування на базі Grafana k6, із чітко визначеними моделями трафіку та критеріями прийнятності, забезпечує об’єктивну базу для оцінки продуктивності та масштабованості системи.

4.5.3 Результати симуляції навантаження за профілями

У ході роботи було розгорнуто три послідовні профілі навантаження, які моделювали різні ступені активності користувачів:

– Smoke-тест (миттєве навантаження, 2 VU) призначений для верифікації працездатності розгорнутого API. Результати продемонстрували 100% успішність проходження функціональних чеків (313 з 313) за повної відсутності помилок HTTP. Час відгуку p95 склав 54,5 ms, а для публічного потоку всього 7,5 ms;

– Load-тест (стабільне використання, 25 VU) симулював безперервну тривалу роботу системи протягом 12 хвилин, що відображає стандартні умови експлуатації для середньої аудиторії платформи. Система продемонструвала абсолютну стабільність: було успішно оброблено 42 692 HTTP-запити із середньою

пропускною здатністю ~59 запитів/с. Зафіксовано 0% відмов. Час відгуку p95 утримався на позначці 16,25 ms, що свідчить про ідеальну збалансованість мікросервісів у штатному режимі;

– Peak/Spike-тест (пікове навантаження, до 200 VU) орієнтований на штучне створення стресових умов для визначення меж міцності поточного інфраструктурного вузла. Протягом 4,5 хвилин система досягла пропускної здатності у ~260 запитів/с, обробивши сумарно 71 567 запитів. Загальний показник успішності склав 98,96%, проте рівень HTTP-помилок зріс до 1,04%, що незначно перевищило цільовий поріг (1%). Латентність p95 збільшилася до 581 ms, що залишається у межах комфортної взаємодії (до 1 секунди).

Для детального аналізу поведінки мікросервісів під час стресового Peak-тесту було сформовано матрицю розподілу відмов за типами бізнес-операцій (табл. 4.3).

Таблиця 4.3. Розподіл помилок за типами операцій під час пікового навантаження (200 VU)

Операція / Ендпоінт	Успішні запити	Невдалі запити	Коефіцієнт успіху (%)
Домашня стрічка (GET /api/v/watch/feed)	9 323	46	99,5%
Пошук та фільтрація відео (GET /api/v/watch/search)	9 284	85	99,1%
Читання коментарів (GET .../comments)	9 222	147	98,4%
Реєстрація перегляду (POST .../views)	9 177	192	98,0%
Профіль користувача (GET /api/users/me)	3 010	0	100,0%
Додавання нового коментаря (POST .../comments)	2 979	31	99,0%
Реакція на відео (POST .../reactions)	2 928	82	97,3%

Аналіз метрик свідчить, що операції отримання даних (GET) зберігають високу стабільність. Найбільш вразливими до деградації виявилися транзакційні операції запису (реєстрація переглядів та проставлення лайків), що зумовлено підвищеною конкуренцією за ресурси на рівні реляційної бази даних PostgreSQL

при блокуванні рядків. Натомість сервіси авторизації (User Service) та каналів (Channel Service) відпрацювали із 100% надійністю.

4.5.4 Зведені результати та оцінка пропускної здатності

Для наочного зіставлення поведінки VidPortal при різних рівнях інтенсивності користувачів сформовано порівняльну табл. 4.4.

Таблиця 4.4. Порівняльна матриця результатів навантаження

Метрика продуктивності	Smoke (2 VU)	Load (25 VU)	Peak (200 VU)
Пропускна здатність (запитів/с)	~4	~59	~260
Латентність відгуку p95 (ms)	55 ms	16 ms	581 ms
Латентність відгуку p99 (ms)	112 ms	20 ms	957 ms
Частка HTTP-помилок (%)	0%	0%	1,04%
Загальна оцінка стабільності	Відмінна	Відмінна	Задовільна

На основі емпіричних даних розраховано класифікацію експлуатаційних рівнів системи, яка дозволяє встановити оптимальні та критичні межі робочого навантаження для поточної архітектурної конфігурації (один екземпляр кожного мікросервісу в Docker) (табл. 4.5).

Таблиця 4.5. Рекомендовані межі експлуатаційного навантаження VidPortal

Рівень навантаження	Кількість VU	Показник RPS	Експлуатаційна характеристика
Комфортний	до 25	~60	0% помилок, наднизька латентність p95 < 20 ms. Режим максимальної ефективності.
Робочий	25 – 100	60 – 150	Стабільне функціонування з лінійним зростанням часу відгуку без критичних збоїв в роботі застосунку.
Граничний	100 – 200	150 – 260	Частка помилок у межах 1-2%, p95 наближається до 600 ms. Поява конкуренції за ресурси бази даних.
Критичний	> 200	> 260	Потребує негайного автоматичного або ручного горизонтального масштабування.

За результатами тестування визначено, що базова конфігурація платформи гарантує високу стабільність роботи при навантаженні до 100 одночасних користувачів, тоді як перевищення критичного порогу у 200 користувачів (понад 260 запитів на секунду) вимагає обов'язкового горизонтального масштабування мікросервісів.

4.5.5 Обмеження навантажувального тестування та рекомендації щодо масштабування

При інтерпретації отриманих метрик необхідно враховувати низку конструктивних обмежень поточного процесу тестування:

- тести проводилися на локальних потужностях розробника, без урахування реальних мережевих затримок (network jitter) провайдерів інтернету;
- кожен мікросервіс функціонував в єдиному екземплярі без налаштованих інструментів оркестрації реплік (на кшталт Kubernetes);
- поза межами поточної симуляції залишилися безпосередні процеси бінарного завантаження (Upload) великих відеофайлів та їх потокова дистрибуція (HLS/DASH Streaming), які створюють специфічне навантаження на дискову підсистему I/O.

Для забезпечення безперебійної роботи застосунку та обслуговуванні аудиторії більше 200 VU одночасно рекомендується:

- горизонтальне масштабування сервісів, розгортання додаткових реплік для найбільш завантажених сервісів;
- інтеграція розподіленої пам'яті Redis для збереження статичних категорій, глобальних стрічок рекомендацій та трендів, що дозволить знизити кількість GET-запитів до PostgreSQL на 60–70%;
- налаштування Connection Pooling та асинхронного пакетного оновлення лічильників переглядів через черги повідомлень, щоб уникнути прямих блокувань таблиць при POST-запитах;

– винесення статичних медіаресурсів (прев'ю, аватари, оптимізовані відеопотоки) на спеціалізовані CDN-платформи.

Проведена комплексна програма верифікації, що поєднала наскрізне мануальне функціональне тестування та автоматизовані стрес-випробування за допомогою Grafana k6, підтвердила високу надійність, працездатність та життєздатність розробленої архітектури відеохостингу VidPortal.

Обрана мікросервісна модель у зв'язці з YARP API Gateway повністю виправдала себе, забезпечивши чітку ізоляцію доменів та стабільну обробку до ~260 запитів/с із загальним рівнем успішності 98,96% навіть під піковими стрес-навантаженнями у 200 VU. Асинхронна поднійно-орієнтована модель взаємодії мікросервісів через брокер RabbitMQ успішно вирішила проблему блокування HTTP-запитів, дозволяючи відокремити важкі фонові процеси транскодування відео (FFmpeg) від клієнтського потоку.

Результати випробувань доводять, що створена система відповідає всім задекларованим технічним вимогам, демонструє відмінні показники швидкодії (латентність p95 < 20 ms при нормальному навантаженні) та має надійний, гнучкий технологічний фундамент для подальшого масштабування, розширення функціоналу та інтеграції в сучасні CI/CD конвеєри автоматизації.

Висновки до розділу 4

У ході виконання роботи реалізовано відеохостинг VidPortal. Проєкт побудовано на мікросервісній архітектурі, що дозволяє незалежно масштабувати сервіси користувачів, каналів та відеоконтенту. Кожен сервіс працює з власною схемою PostgreSQL через Entity Framework Core, що забезпечує ізоляцію даних.

Важливою частиною системи є асинхронний конвеєр обробки відео. Складна з точки зору навантаження на систему логіка винесена у окремий фоновий воркер. Комунікація з ним відбувається за допомогою черг повідомлень в RabbitMQ. Цей підхід дозволяє розгорнути воркер на окремому середовищі, це дозволить

використати максимум ресурсів і не впливати на роботоспроможність інших сервісів.

Фронтенд-частина – це SPA застосунок розроблений за допомогою Angular. Для реактивності використано Signals та RxJS, а оновлення даних у реальному часі (лічильники, активність) працює через SignalR. Дизайн адаптовано під мобільні пристрої за допомогою Tailwind CSS.

Стійкість та швидкість роботи системи перевірили через навантажувальні тести в Grafana k6. Це дозволило знайти межі потужності платформи та вузькі місця в обробці запитів. Результати підтвердили, що мікросервісна архітектура добре масштабується, тому додавати нові функції і розширювати існуючий функціонал буде просто.

ВИСНОВКИ

У межах кваліфікаційної роботи реалізовано відеохостинг VidPortal з підтримкою транскодування відео. Мікросервісна архітектура дозволяє масштабувати окремі вузли системи незалежно один від одного. Використання черг завдань для обробки медіа дає змогу ефективно розподіляти ресурси сервера, що зберігає стабільну роботу платформи навіть при високому навантаженні.

Системна архітектура серверної частини побудована за допомогою платформи .NET, що забезпечує швидку та безпечну роботу. Для асинхронної комунікації між мікросервісами використовуєтится RabbitMQ. Процес обробки відео виконує FFmpeg. Це дозволяє автоматично адаптувати контент для будь-яких пристроїв.

У межах реалізації розроблено:

- сервісну інфраструктуру: розгорнуто систему мікросервісів, що забезпечують незалежне масштабування компонентів платформи;
- pipeline транскодування: налаштовано автоматизований процес конвертації відео, що мінімізує час очікування користувача після завантаження файлу;
- механізм надійності: інтегровано систему журналювання та обробки помилок, що гарантує цілісність медіафайлів при високих навантаженнях.

Тестування продуктивності продемонструвало стабільність архітектури при паралельній обробці декількох відеопотоків, а використання обраного технологічного стеку забезпечило відповідність сучасним вимогам до швидкості відгуку та безпеки даних.

Наступні етапи розвитку проєкту спрямовані на покращення продуктивності. Для цього планується впровадити систему кешування. Також передбачена розробка нового мікросервісу, який розширить аналітичні можливості платформи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Interactive Video Platform - D-ID. URL: https://www.d-id.com/resources/glossary/interactive-video-platform/ (дата звернення: 21.04.2026).

2. What is Interactive Video? Master the Art of Creating Them – Firework. URL: https://firework.com/blog/what-is-interactive-video (дата звернення: 23.04.2026).

3. Interactive Video Platforms And Tools – Storista. URL: https://storista.io/blog/interactive-video-platforms (дата звернення: 22.04.2026).

4. A Survey on Video Big Data Analytics: Architecture, Technologies, and Open Research Challenges. URL: https://www.mdpi.com/2076-3417/15/14/8089 (дата звернення: 25.04.2026).

5. A Detailed Look at the History of Streaming Services – Shentel. URL: https://www.shentel.com/en/news/2025/january/history-of-streaming-services (дата звернення: 26.04.2026).

6. Interactive Streaming Market Report 2026 - Research and Markets. URL: https://www.researchandmarkets.com/reports/6076441/interactive-streaming-market-report (дата звернення: 27.04.2026).

7. The history and evolution of video streaming - Everest Cast. URL: https://everestcast.com/blogs/the-history-and-evolution-of-video-streaming (дата звернення: 01.05.2026).

8. The Evolution and Impact of Streaming Services: Changing the Media Landscape. URL: https://www.globalmediajournal.com/open-access/the-evolution-and-impact-of-streaming-services-changing-the-media-landscape.pdf (дата звернення: 05.05.2026).

9. Video Marketing Statistics 2026: 160+ Essential Data - Digital Applied. URL: https://www.digitalapplied.com/blog/video-marketing-statistics-2026-data-points (дата звернення: 06.05.2026).

10. Online Video Platforms Market Size and YoY Growth Rate, 2033.

URL: https://www.coherentmarketinsights.com/industry-reports/online-video-platforms-market (дата звернення: 08.05.2026).

11. 2026 Digital Media Trends: Capturing always-on fandom between releases and seasons – Deloitte. URL: https://www.deloitte.com/us/en/insights/industry/technology/digital-media-trends-consumption-habits-survey.html (дата звернення: 09.05.2026).

12. Enterprise Video Platform vs Online Video Platform: Which is Right for Your Business? URL: https://www.gumlet.com/learn/enterprise-video-platform-vs-online-video-platform/ (дата звернення: 10.05.2026).

13. The Rise of Interactive and Shoppable YouTube Videos: Merging Commerce with Content in 2026 - Marketing Agent Blog. URL: https://marketingagent.blog/2026/02/22/the-rise-of-interactive-and-shoppable-youtube-videos-merging-commerce-with-content-in-2026/
(дата звернення: 12.05.2026).

14. 12 Best YouTube Alternatives in 2026: Explore the Top Video Platforms - Appy Pie Automate. URL: https://www.appypieautomate.ai/blog/alternatives/youtube-alternatives (дата звернення: 14.05.2026).

15. Pros and Cons of the Live Streaming Video Platforms - Show.gg. URL: https://show.gg/pros-and-cons-of-the-live-streaming-video-platforms/
(дата звернення: 15.05.2026).

16. Top 15 YouTube Alternatives – Features, Pros & Cons. URL: https://nextbigtechnology.com/top-15-youtube-alternatives-features-pros-cons/
(дата звернення: 16.05.2026).

17. Top 10 Live Streaming Platforms (+ How to Choose the Right One) – Riverside. URL: https://riverside.com/blog/streaming-platforms
(дата звернення: 17.05.2026).

18. Reverse Proxy with YARP in .NET. URL https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/yarp (дата звернення: 18.05.2026).

19. RabbitMQ Documentation: Consumer Acknowledgements and Publisher

Confirms. URL https://www.rabbitmq.com/docs/confirms/ (дата звернення: 20.05.2026).

20. FFmpeg at Meta: Media Processing at Scale. URL https://engineering.fb.com/2026/03/02/video-engineering/ffmpeg-at-meta-media-processing-at-scale/ (дата звернення: 21.05.2026).

21. Angular Official Documentation. URL https://angular.dev/overview (дата звернення: 22.05.2026).

22. Exploring the Potential of Microservices in Internet of Things: A Systematic Review of Security and Prospects – MDPI. URL: https://www.mdpi.com/1424-8220/24/20/6771 (дата звернення: 25.05.2026).

23. From Microservice to Monolith: A Multivocal Literature Review – MDPI. URL: https://www.mdpi.com/2079-9292/13/8/1452 (дата звернення: 26.05.2026).

24. SignalR vs WebSocket: Real-Time Communication for Developers (2025). URL: https://www.videosdk.live/developer-hub/websocket/signalr-vs-websocket (дата звернення: 27.05.2026).

25. SignalR vs. WebSocket: Choosing the Right Real-Time Communication Technology.
URL: https://www.sparkleweb.in/blog/signalr_vs._websocket:_choosing_the_right_real-time_communication_technology (дата звернення: 28.05.2026).

26. Adaptive Bitrate Streaming with HLS and DASH - Norsk Video. URL: https://norsk.video/adaptive-bitrate-streaming-with-hls-and-dash/ (дата звернення: 01.06.2026).

27. Crowd-Sourced Subjective Assessment of Adaptive Bitrate Algorithms in Low-Latence MPREG-DASH Streaming – MDPI. URL: https://www.mdpi.com/2076-3417/15/24/13092 (дата звернення: 03.06.2026).

28. Clean Architecture in .NET Core. URL: https://medium.com/@jenilsojitra/clean-architecture-in-net-core-e18b4ad229c8 (дата звернення: 07.06.2026).

29. API Gateway in .Net Microservice Architecture.

URL: https://dotnetfullstackdev.medium.com/api-gateway-in-net-microservice-architecture-411cdf52c22d (дата звернення: 09.06.2026).

30. Part 2: RabbitMQ Best Practices - For High Availability and Reliability. URL: https://www.cloudamqp.com/blog/part2-rabbitmq-best-practices-for-high-availability.html (дата звернення: 10.06.2026).

31. The Database-per-Service Pattern. URL: https://medium.com/design-microservices-architecture-with-patterns/the-database-per-service-pattern-9d511b882425 (дата звернення: 11.06.2026).

32. Performance testing with Grafana Cloud k6. URL: https://grafana.com/docs/grafana-cloud/testing/k6/ (дата звернення: 13.06.2026).

ДОДАТОК А

Лістинг коду бізнес-логіки автентифікації (Identity + JWT + refresh)

А.1 Логіка реєстрації користувача

```
public async Task<ApiResponse<AuthResponseDto>> RegisterAsync(RegisterUserDto dto,
CancellationToken cancellationToken = default)
{
    var isExistingUser = await userManager.FindByEmailAsync(dto.Email);
    if (isExistingUser != null)
    {
        logger.LogWarning("Attempt to register with existing email: {Email}", dto.Email);
        return ApiResponse<AuthResponseDto>.Failure("User with this email already
exists.");
    }

    var user = mapper.Map<User>(dto);

    user.CreatedAt = DateTime.UtcNow;
    user.UpdatedAt = DateTime.UtcNow;

    var result = await userManager.CreateAsync(user, dto.Password);

    if (!result.Succeeded)
    {
        var errors = result.Errors.Select(e => e.Description).ToList();
        logger.LogWarning("User registration failed for email: {Email}. Errors: {Errors}",
dto.Email, string.Join(", ", errors));
        return ApiResponse<AuthResponseDto>.Failure(errors);
    }

    try
    {
        await userManager.AddToRoleAsync(user, "User");
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "Failed to add role ‘User’ to user {UserId}", user.Id);
    }

    var roles = new List<string> { "User" };

    var accessToken = tokenProvider.CreateAccessToken(user, roles);
    var refreshToken = tokenProvider.CreateRefreshToken();

    var userRefreshToken = new UserRefreshToken
    {
        UserId = user.Id,
        Token = refreshToken,
        ExpiresAt = DateTime.UtcNow.AddDays(7)
```

```
    };

    await dbContext.RefreshTokens.AddAsync(userRefreshToken);
    await dbContext.SaveChangesAsync(cancellationToken);

    var authResponse = new AuthResponseDto(accessToken, refreshToken);
    logger.LogInformation("User registered successfully with email: {Email}", dto.Email);

    return ApiResponse<AuthResponseDto>.Success(authResponse);
}
```

А.2 Логіка входу в обліковий запис

```
public async Task<ApiResponse<AuthResponseDto>> LoginAsync(LoginUserDto dto,
CancellationToken cancellationToken = default)
{
    var user = await userManager.FindByEmailAsync(dto.Email);
    if (user == null)
    {
        logger.LogWarning("Login attempt with non-existing email: {Email}", dto.Email);
        return ApiResponse<AuthResponseDto>.Failure("Invalid email or password.");
    }

    var isPasswordValid = await userManager.CheckPasswordAsync(user, dto.Password);
    if (!isPasswordValid)
    {
        logger.LogWarning("Invalid password attempt for email: {Email}", dto.Email);
        return ApiResponse<AuthResponseDto>.Failure("Invalid email or password.");
    }

    var roles = await userManager.GetRolesAsync(user);

    var accessToken = tokenProvider.CreateAccessToken(user, roles);
    var refreshToken = tokenProvider.CreateRefreshToken();

    var userRefreshToken = new UserRefreshToken
    {
        UserId = user.Id,
        Token = refreshToken,
        ExpiresAt = DateTime.UtcNow.AddDays(7)
    };

    await dbContext.RefreshTokens.AddAsync(userRefreshToken);
    await dbContext.SaveChangesAsync(cancellationToken);

    var authResponse = new AuthResponseDto(accessToken, refreshToken);
    logger.LogInformation("User logged in successfully with email: {Email}", dto.Email);

    return ApiResponse<AuthResponseDto>.Success(authResponse);
}
```

ДОДАТОК Б

Лістинг коду бізнес-логіки роботи з каналами

Б.1 Логіка створення каналу

```
public async Task<ApiResponse<CreateChannelResultDto>> CreateChannelAsync(Guid userId,
CreateChannelDto dto)
{
    var hasOwnedChannel = await dbContext.ChannelMembers
        .AnyAsync(m => m.UserId == userId && m.Role == ChannelRole.Owner);

    var channel = mapper.Map<Channel>(dto);
    var channelMember = new ChannelMember()
    {
        UserId = userId,
        Channel = channel,
        Role = ChannelRole.Owner,
        IsPrimary = !hasOwnedChannel,
        JoinedAt = DateTime.UtcNow,
    };

    await dbContext.AddAsync(channel);
    await dbContext.AddAsync(channelMember);
    await dbContext.SaveChangesAsync();

    var channelResultDto = mapper.Map<CreateChannelResultDto>(channel);
    return ApiResponse<CreateChannelResultDto>.Success(channelResultDto);
}
```

Б.2 Логіка додавання учасника каналу

```
public async Task<ApiResponse<bool>> AddChannelMemberAsync(Guid channelId, Guid userId,
ChannelRole role)
{
    var channel = await dbContext.FindAsync<Channel>(channelId);
    if (channel == null)
    {
        return ApiResponse<bool>.Failure("Channel not found");
    }

    if (role == ChannelRole.Owner)
    {
        var ownerExists = await dbContext.ChannelMembers
            .AnyAsync(m => m.ChannelId == channelId && m.Role == ChannelRole.Owner);

        if (ownerExists)
        {
            logger.LogWarning("Failed to add member: Channel {ChannelId} already has an
owner.", channelId);
```

```
            return ApiResponse<bool>.Failure("A channel can only have one owner.");
        }
    }

    var isExists = await dbContext.ChannelMembers.AnyAsync(x => x.ChannelId == channelId
&& x.UserId == userId);
    if (isExists)
    {
        return ApiResponse<bool>.Failure("User is already a member");
    }

    var newChannelMember = new ChannelMember()
    {
        ChannelId = channelId,
        UserId = userId,
        Role = role,
        JoinedAt = DateTime.UtcNow,
    };
    await dbContext.ChannelMembers.AddAsync(newChannelMember);
    await dbContext.SaveChangesAsync();

    return ApiResponse<bool>.Success(true);
}
```

ДОДАТОК В

Лістинг коду бізнес-логіки роботи з відео

В.1 Логіка завантаження відео

```
public async Task<ApiResponse<Guid>> UploadVideoAsync(
    Guid uploaderChannelId,
    Guid uploadingUserId,
    CreateVideoDto createVideoDto,
    string originalFileName,
    Stream fileStream,
    CancellationToken cancellationToken)
{
    var isCategoryExists = await dbContext.Categories
        .AnyAsync(c => c.Id == createVideoDto.CategoryId, cancellationToken);

    if (!isCategoryExists)
    {
        return ApiResponse<Guid>.Failure("Category does not exist.");
    }

    var tags = await GetVideoTags(createVideoDto.Tags, cancellationToken);

    var video = mapper.Map<Video>(createVideoDto);
    video.Id = Guid.NewGuid();
    video.UploaderChannelId = uploaderChannelId;
    video.Status = VideoStatus.Processing;
    video.Tags = tags;
    video.PublishedAt = DateTime.UtcNow;

    try
    {
        var extension = Path.GetExtension(originalFileName);
        var fileName = $"original{(string.IsNullOrWhiteSpace(extension) ? ".mp4" :
extension)}";
        var container = $"videos/{video.Id}";

        var fileUrl = await fileStorageService.UploadFileAsync(fileStream, fileName,
container, cancellationToken);
        video.StorageIdentifier = fileUrl;

        dbContext.Videos.Add(video);
        await dbContext.SaveChangesAsync(cancellationToken);

        var relativePath = fileUrl.TrimStart('/');
        var publishResult = await messageBus.PublishAsync(
            new VideoUploadedEvent(
                video.Id,
                relativePath,
                createVideoDto.Title,
```

```
                uploadingUserId,
                DateTime.UtcNow,
                BuildProcessingOptions(createVideoDto.ProcessingOptions)),
            "video-uploads");

        if (!publishResult.IsSuccess)
        {
            logger.LogError(
                "Video {VideoId} saved but VideoUploadedEvent was not published:
{Errors}",
                video.Id,
                publishResult.Errors is null ? "(unknown)" : string.Join("; ",
publishResult.Errors));
        }

        return ApiResponse<Guid>.Success(video.Id);
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "Error uploading video for channel {ChannelId}",
uploaderChannelId);
        return ApiResponse<Guid>.Failure("File upload failed.");
    }
}
```

В.2 Логіка отримання відео за ідентифікатором

```
public async Task<ApiResponse<VideoDetailsDto>> GetVideoByIdAsync(
    Guid videoId,
    CancellationToken cancellationToken)
{
    var video = await dbContext.Videos
        .Include(v => v.Tags)
        .ThenInclude(vt => vt.Tag)
        .Include(v => v.VideoSources)
        .Include(v => v.Comments)
        .AsSplitQuery()
        .AsNoTracking()
        .FirstOrDefaultAsync(v => v.Id == videoId);

    if (video is null)
    {
        return ApiResponse<VideoDetailsDto>.Failure("Video not found.");
    }

    var videoDto = mapper.Map<Video, VideoDetailsDto>(video);
    videoDto = await HydrateVideoDetailsAsync(videoDto, cancellationToken);
    return ApiResponse<VideoDetailsDto>.Success(videoDto);
}
```

ДОДАТОК Г

Лістинг коду бізнес-логіки обробки відео

Г.1 Логіка для обробки відео

```
private async Task ProcessVideoAsync(VideoUploadedEvent videoUploadedEvent,
CancellationToken ct)
{
    logger.LogInformation($"Processing video {videoUploadedEvent.VideoId}...");
    string inputPath = Path.Combine(
        "/storage",
        videoUploadedEvent.RelativePath);

    try
    {
        var options = videoUploadedEvent.Options ?? new VideoProcessingOptions();
        var encodingRequest = new VideoEncodingRequest(
            TargetQualities: ResolveTargetQualities(options),
            EncodingPreset: options.EncodingPreset,
            NormalizeAudio: options.NormalizeAudio);

        ApiResponse<string>? thumbResult = null;
        if (options.ExtractThumbnails)
        {
            thumbResult = await videoEncoder.ExtractThumbnailAsync(
                inputPath,
                ct);
        }

        var encodeResult = await videoEncoder.EncodeAsync(
            inputPath,
            encodingRequest,
            ct);

        if (!encodeResult.IsSuccess || encodeResult.Data is null ||
!encodeResult.Data.IsSuccess)
        {
            var msg = encodeResult.Data?.ErrorMessage
                ?? encodeResult.Errors?.FirstOrDefault()
                ?? "Video encoding failed.";
            await PublishFailureAsync(videoUploadedEvent.VideoId, msg);
            logger.LogWarning("Encoding failed for video {VideoId}: {Message}",
videoUploadedEvent.VideoId, msg);
            return;
        }

        var data = encodeResult.Data;
        string? thumbName = thumbResult?.IsSuccess == true ? thumbResult.Data : null;
        if (string.IsNullOrWhiteSpace(thumbName))
        {
```

```
            logger.LogWarning(
                "Thumbnail extraction failed for video {VideoId}: {Message}",
                videoUploadedEvent.VideoId,
                thumbResult?.Errors?.FirstOrDefault() ?? "Thumbnail extraction was
skipped.");
        }

        var eventFiles = data.Files.Select(f => new EncodedVideoFile(
            f.Quality,
            f.FileUrl,
            f.FileSize,
            f.Width,
            f.Height,
            f.VideoCodec,
            f.AudioCodec,
            f.VideoBitrateKbps,
            f.AudioBitrateKbps,
            f.EncodingPreset,
            f.ConstantRateFactor,
            f.AudioNormalized,
            f.ProcessingTimeMs,
            f.CompressionRatio
        )).ToList();

        var resultEvent = new VideoProcessingCompletedEvent(
            videoUploadedEvent.VideoId,
            true,
            data.Duration,
            thumbName,
            eventFiles
        );

        await bus.PublishAsync(
            resultEvent,
            "video-results");
        logger.LogInformation($"Successfully processed video
{videoUploadedEvent.VideoId}");
    }
    catch (Exception ex)
    {
        logger.LogError(ex, $"Error during FFmpeg processing for
{videoUploadedEvent.VideoId}");
        await PublishFailureAsync(videoUploadedEvent.VideoId, ex.Message);
    }
}
```

ДОДАТОК Д

Лістинг коду роботи з шиною повідомлень

Д.1 Публікації повідомлень у шину повідомлень

```
public async Task<ApiResponse<bool>> PublishAsync<T>(
    T message,
    string queueName)
{
    if (_channel is null)
    {
        return ApiResponse<bool>.Failure("Channel is not initialized");
    }

    await _channel.QueueDeclareAsync(
        queue: queueName,
        durable: true,
        exclusive: false,
        autoDelete: false,
        arguments: null);

    var body = Encoding.UTF8.GetBytes(JsonSerializer.Serialize(message));

    await _channel.BasicPublishAsync(
        exchange: string.Empty,
        routingKey: queueName,
        body: body);

    return ApiResponse<bool>.Success(true);
}
```

Д.2 Підписка на повідомлення з черги

```
public async Task<ApiResponse<bool>> SubscribeAsync<T>(
    string queueName,
    Func<T, Task> handler)
{
    if (_channel is null)
    {
        return ApiResponse<bool>.Failure("Channel is not initialized");
    }

    await _channel.QueueDeclareAsync(
        queueName,
        durable: true,
        exclusive: false,
        autoDelete: false);

    var consumer = new AsyncEventingBasicConsumer(_channel);
```

```
    consumer.ReceivedAsync += async (model, ea) =>
    {
        try
        {
            var body = ea.Body.ToArray();
            var message = JsonSerializer.Deserialize<T>(
                Encoding.UTF8.GetString(body));

            if (message != null)
            {
                await handler(message);
                await _channel.BasicAckAsync(
                    ea.DeliveryTag,
                    false);
            }
        }
        catch (Exception ex)
        {
            _logger.LogError(
                ex,
                "Error processing video message");

            await _channel.BasicNackAsync(
                ea.DeliveryTag,
                false,
                requeue: true);
        }
    };

    await _channel.BasicConsumeAsync(
        queueName,
        autoAck: false,
        consumer: consumer);

    return ApiResponse<bool>.Success(true);
}
```