Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Проєктування та розробка програмного забезпечення на основі архітектури Next.js та Serverless технологій для веборієнтованої платформи менеджменту та дистрибуції цифрового контенту

Виконав: студент IV курсу, групи СП-43
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Талалай Р.П. (прізвище та ініціали)

Керівник (підпис) Петрик М.Р. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) Палка О.В. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет *комп'ютерно-інформаційних систем і програмної інженерії*
(повна назва факультету)

Кафедра *програмної інженерії*
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *бакалавр*
(назва освітнього ступеня)

за спеціальністю *121 «Інженерія програмного забезпечення»*
(шифр і назва спеціальності)

студенту Талалай Роман Петрович
(прізвище, ім'я, по батькові)

1. Тема роботи Проєктування та розробка програмного забезпечення на основі архітектури Next.js та Serverless технологій для веборієнтованої платформи менеджменту та дистрибуції цифрового контенту

Керівник роботи Петрик Михайло Романович, д.ф-м.н, професор
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *6* » *квітня* 2026 року № 4/9-171

2. Термін подання студентом завершеної роботи *22.06.2026*

3. Вихідні дані до роботи *Технічне завдання*

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1. Аналіз вимог та предметної області
2. Проєктування
3. Конструювання та тестування
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
Слайди презентації, діаграма варіантів використання системи, схема бази даних, схема архітектури застосунку, схема розгортання системи у безсерверному середовищі.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			
Нормоконтроль	Стоянов *Ю.М. к.т.н., доц. каф ПІ.*		

7. Дата видачі завдання *06.04.2026р.*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Розробка технічного завдання*	*6.04 – 12.04*	*Виконано*
2	*Робота над першим розділом «Аналіз вимог до програмної системи»*	*13.04 – 26.04*	*Виконано*
3	*Робота над другим розділом «Проєктування та розробка програмної системи»*	*27.04 – 03.05*	*Виконано*
4	*Робота над третім розділом «Тестування, впровадження та аналіз результатів»*	*04.05 – 17.05*	*Виконано*
5	*Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»*	*18.05 – 24.05*	*Виконано*
6	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	*Виконано*
7	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
8	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
9	*Захист кваліфікаційної роботи бакалавра*		

Студент _____ Талалай Р.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Петрик М.Р.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Проєктування та розробка програмного забезпечення на основі архітектури Next.js та Serverless технологій для веборієнтованої платформи менеджменту та дистрибуції цифрового контенту // Кваліфікаційна робота освітнього рівня «Бакалавр» // Талалай Роман Петрович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-43 //Тернопіль, 2026 // С.62, рис. – 14, табл. – 10, слайдів – , додат. – 3, бібліогр. – 21.

Ключові слова: веборієнтована платформа, цифровий відеоконтент, Next.js, безсерверні технології, серверні дії, запис екрана, автоматична транскрипція, реляційна база даних, чиста архітектура.

Мета роботи — підвищення масштабованості та керованості платформи менеджменту й дистрибуції відеоконтенту шляхом її програмної реалізації на основі Next.js і безсерверних технологій із дотриманням принципів чистої архітектури та SOLID.

Для досягнення мети проаналізовано предметну область і сформовано вимоги, обґрунтовано стек, спроєктовано архітектуру та модель даних, реалізовано ключові модулі й засоби безпеки та виконано тестування.

У першому розділі проаналізовано предметну область і сформовано вимоги; у другому — обґрунтовано стек, спроєктовано архітектуру й модель даних та реалізовано ключові модулі; у третьому — виконано тестування, впровадження та аналіз відповідності вимогам; у четвертому — розглянуто безпеку життєдіяльності та основи охорони праці.

Об'єктом дослідження є розробка платформ дистрибуції цифрового контенту, предметом — методи й засоби її побудови на основі Next.js і безсерверних технологій.

ABSTRACT

Design and development of software based on Next.js architecture and Serverless technologies for a web-oriented platform for management and distribution of digital content // Bachelor's qualification work // Talalay Roman Petrovich // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, group SP-43 // Ternopil, 2026 //
P. 62, fig. – 14, tab. – 10, slides – , app. – 3, ref. – 21.

Keywords: web-oriented platform, digital video content, Next.js, serverless technologies, server actions, screen recording, automatic transcription, relational database, clean architecture.

The aim of the work is to improve the scalability and manageability of a platform for management and distribution of video content through its software implementation based on Next.js and serverless technologies, following the principles of clean architecture and SOLID.

To achieve this aim, the subject area was analyzed and the requirements formulated, the stack justified, the architecture and data model designed, the key modules and security implemented, and the system tested.

The first section analyzes the subject area and formulates the requirements; the second justifies the stack, designs the architecture and data model, and implements the key modules; the third covers testing, deployment, and conformance analysis; the fourth addresses life safety and occupational health and safety.

The object of research is the development of digital content distribution platforms; the subject is the methods and tools for building such a platform on Next.js and serverless technologies.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

Безсерверні обчислення (serverless) – модель виконання, за якої обчислювальні ресурси виділяються провайдером на вимогу, без постійно запущеного сервера застосунку.

Запис екрана – захоплення зображення екрана й звуку засобами браузера для створення відеоматеріалу.

Підписаний URL – тимчасове посилання, що надає обмежене право завантажити файл у сховище без відкритого доступу на запис.

Серверні дії (Server Actions) – функції, що виконуються на сервері й викликаються з клієнтського коду без створення окремого програмного інтерфейсу.

Транскрипція – автоматичне перетворення мовлення у відеозаписі на текст (субтитри).

Чиста архітектура (Clean Architecture) – підхід до організації коду з однонапрямленою залежністю шарів та ізоляцією бізнес-логіки від зовнішніх деталей.

API (Application Programming Interface) – програмний інтерфейс взаємодії між компонентами системи та зовнішніми сервісами.

Arcjet – сервіс захисту вебзастосунку: екранування запитів, виявлення автоматизованих клієнтів і обмеження частоти звернень.

AssemblyAI – зовнішній сервіс автоматичного розпізнавання мовлення, що використовується для формування субтитрів.

Better Auth – бібліотека автентифікації для застосунків на Next.js: вхід за поштою й паролем та через зовнішні облікові записи.

CDN (Content Delivery Network) – мережа доставки вмісту для віддачі статичних ресурсів користувачам.

Drizzle ORM – типобезпечний інструмент об’єктно-реляційного відображення для TypeScript із керуванням міграціями.

ER-діаграма (Entity-Relationship Diagram) – діаграма сутностей і зв’язків для моделювання структури бази даних.

ESLint – статичний аналізатор коду JavaScript і TypeScript, що виявляє помилки та порушення правил оформлення.

HTTPS (HyperText Transfer Protocol Secure) – захищений протокол передавання даних у мережі.

JSON (JavaScript Object Notation) – текстовий формат обміну структурованими даними.

Next.js – фреймворк для вебзастосунків на основі React із підтримкою рендерингу на сервері та серверних дій.

next-intl – бібліотека локалізації застосунків на Next.js.

OAuth – відкритий протокол авторизації, що дає змогу входити через зовнішній обліковий запис (зокрема Google).

ORM (Object-Relational Mapping) – відображення об’єктів програми на структури реляційної бази даних.

React – бібліотека для побудови інтерфейсів користувача з компонентів.

SOLID – сукупність п’яти принципів об’єктно-орієнтованого проєктування, спрямованих на підтримуваність коду.

SSR (Server-Side Rendering) – формування HTML-сторінки на боці сервера перед надсиланням клієнтові.

Supabase – хмарна платформа; у роботі використано її об’єктне сховище для зберігання відео й обкладинок.

Tailwind CSS – утилітарний CSS-фреймворк для оформлення інтерфейсу користувача.

TypeScript – мова програмування зі статичною типізацією, що розширює JavaScript.

UML (Unified Modeling Language) – уніфікована мова моделювання програмних систем.

WebVTT (Web Video Text Tracks) – текстовий формат субтитрів для відео.

UUID (Universally Unique Identifier) – універсальний унікальний ідентифікатор.

ЗМІСТ

ВСТУП

Відео стало однією з домінуючих форм цифрового контенту: на відеотрафік припадає більша частина обсягу інтернет-трафіку, а засоби запису екрана та асинхронного обміну відеоповідомленнями активно доповнюють синхронну комунікацію в дистанційному навчанні, технічній підтримці та розподілених командах розробників. Це формує технічну вимогу до платформ, які одночасно забезпечують завантаження великих файлів, їх надійне зберігання, структуроване керування метаданими та контрольовану дистрибуцію контенту кінцевим користувачам.

Монолітний серверний застосунок, що самостійно приймає, обробляє та віддає медіапотоки, погано масштабується за нерівномірного навантаження й потребує постійного резервування обчислювальних ресурсів. Архітектура, що поєднує рендеринг на боці сервера за моделлю Next.js App Router із винесенням бізнес-логіки у функції безсерверного середовища (serverless), виконує обчислення на вимогу та масштабується автоматично відповідно до фактичного навантаження. Окремою інженерною проблемою є розмежування відповідальності між шарами: концентрація логіки в компонентах інтерфейсу чи глобальних контекстах («God Object») підвищує зв’язність і вартість супроводу, тоді як дотримання принципів чистої архітектури та SOLID локалізує зміни й зберігає тестованість коду.

Метою кваліфікаційної роботи є підвищення масштабованості та керованості веборієнтованої платформи менеджменту й дистрибуції цифрового контенту шляхом проєктування та програмної реалізації застосунку на основі архітектури Next.js і безсерверних технологій із дотриманням принципів чистої архітектури та SOLID.

Для досягнення поставленої мети визначено такі задачі:

1) проаналізувати предметну область менеджменту й дистрибуції відеоконтенту та сформувати вимоги до програмної системи;
2) обґрунтувати технологічний стек і спроєктувати архітектуру з розмежуванням шарів інтерфейсу, бізнес-логіки та доступу до даних;

3) спроєктувати модель даних і схему реляційної бази даних для метаданих контенту та облікових записів;
4) реалізувати ключові модулі: автентифікацію, запис і завантаження контенту через підписані URL-адреси, серверні дії та автоматичну транскрипцію;
5) реалізувати механізми безпеки й контролю доступу до серверного середовища;
6) виконати тестування програмної системи та оцінити її відповідність сформованим вимогам.

Об’єктом дослідження є процес проєктування та розробки веборієнтованих платформ менеджменту й дистрибуції цифрового контенту. Предметом дослідження є методи, архітектурні підходи та програмні засоби побудови такої платформи на основі Next.js і безсерверних технологій.

Новизна одержаних результатів полягає в обґрунтуванні архітектурного підходу, що поєднує серверний рендеринг Next.js, серверні дії (Server Actions) як єдину контрольовану точку доступу до даних і безсерверне зберігання медіафайлів через підписані URL-адреси, з послідовним застосуванням принципів чистої архітектури для ізоляції бізнес-логіки та усунення антипатерну «God Object».

Практичне значення роботи полягає у створенні готового до розгортання програмного продукту для запису, зберігання, керування та дистрибуції відеоконтенту в умовах нерівномірного навантаження без постійно зарезервованої серверної інфраструктури. Запропонована архітектура придатна для подальшого розширення функціональності платформи.

Робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел і додатків. Перший розділ присвячено аналізу предметної області та формуванню вимог, другий — проєктуванню архітектури й моделі даних і реалізації ключових модулів, третій — тестуванню, впровадженню та підтримці, четвертий — безпеці життєдіяльності й основам охорони праці

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Розділ присвячено аналізу вимог до веборієнтованої платформи менеджменту й дистрибуції цифрового відеоконтенту. Метою аналізу є формування цілісного уявлення про предметну область, межі та функції майбутньої системи до початку її проєктування й реалізації. Коректно сформульовані вимоги визначають обсяг робіт, обмеження та критерії якості програмного продукту і знижують вартість подальших змін.

У межах розділу проаналізовано предметну область менеджменту та дистрибуції відеоконтенту, виконано постановку завдання й цілей розробки, визначено акторів і варіанти використання системи та описано ключові з них. Аналіз спирається на фактичні процеси розроблюваного застосунку: запис екрана, завантаження контенту в хмарне сховище, керування метаданими, автоматичну транскрипцію та контрольовану дистрибуцію матеріалів.

Результати розділу є вихідними даними для розділу 2, у якому виконано проєктування архітектури, моделі даних і програмну реалізацію системи.

1.1 Аналіз предметної області

Предметною областю роботи є веборієнтовані платформи менеджменту та дистрибуції цифрового контенту — програмні системи, що забезпечують повний життєвий цикл медіаматеріалу від його створення до доставки кінцевому споживачеві. У сегменті відеоконтенту такі платформи обслуговують асинхронну відеокомунікацію: запис екрана, обмін короткими відеоповідомленнями в дистанційному навчанні, технічній підтримці та розподілених командах розробників. Обсяг відеоданих у глобальному інтернет-трафіку зростає, що підвищує вимоги до пропускної здатності завантаження, обсягів зберігання та швидкості доставки матеріалів [1].

Життєвий цикл одиниці контенту в межах такої платформи охоплює кілька послідовних процесів: створення (запис або імпорт медіафайлу), завантаження у

сховище, каталогізацію з формуванням метаданих (заголовка, опису, тегів, обкладинки), додаткову обробку, зокрема автоматичне формування субтитрів, публікацію із заданим рівнем доступу та власне дистрибуцію — перегляд і поширення матеріалу. Кожен із цих процесів висуває окремі технічні вимоги: завантаження великих файлів потребує надійного каналу й контролю цілісності, зберігання — масштабованого об’єктного сховища, а дистрибуція — розмежування прав доступу між публічними та приватними матеріалами.

Виокремлення ключових сутностей предметної області та їхніх атрибутів є відправним етапом моделювання програмної системи [2]. Центральними сутностями тут є одиниця контенту, її автор і політика доступу. У розроблюваній системі одиниця контенту описується записом із набором атрибутів: унікальний ідентифікатор, заголовок, опис, посилання на відеофайл та обкладинку у сховищі, перелік тегів, текст субтитрів, тривалість, кількість переглядів і часові позначки створення та оновлення. Кожна одиниця контенту належить зареєстрованому користувачеві і має атрибут видимості зі значеннями «публічний» або «приватний», що визначає доступність матеріалу стороннім користувачам.

Менеджмент контенту зводиться до набору операцій над одиницями контенту: створення запису з метаданими, перегляд переліку власних і доступних матеріалів, оновлення рівня видимості, облік переглядів і видалення матеріалу разом із відповідними файлами у сховищі. Окрему групу становлять операції автентифікації та авторизації: змінювати видимість чи видаляти матеріал може лише його власник, а приватні матеріали приховуються від інших користувачів. Дистрибуція реалізується через перегляд опублікованого матеріалу за посиланням і поширення цього посилання, тоді як приватні матеріали залишаються доступними лише автору.

Окремою складовою предметної області є взаємодія користувача з платформою. Ефективність такої взаємодії визначається зрозумілістю інтерфейсу завантаження, перегляду й керування контентом, оскільки надмірна складність екранних форм знижує продуктивність роботи з матеріалами [3]. Звідси випливає вимога щодо простого та послідовного інтерфейсу основних сценаріїв — запису,

завантаження, публікації та перегляду контенту.

Суттєвою для предметної області є додаткова обробка контенту, що підвищує його доступність і придатність до пошуку. Автоматичне формування субтитрів дає змогу сприймати матеріал без звуку, поліпшує доступність для користувачів із вадами слуху та створює текстове подання відео, придатне для повнотекстового пошуку. У розроблюваній системі текст субтитрів зберігається разом із метаданими одиниці контенту й формується асинхронно після завантаження, не блокуючи публікацію матеріалу.

Дистрибуція контенту в межах платформи передбачає не лише його доставку, а й засоби впорядкування та оцінювання матеріалів. Теги забезпечують тематичну класифікацію та пошук, а лічильник переглядів відображає рівень зацікавленості аудиторії. На відміну від систем потокового мовлення в реальному часі, розглянута предметна область охоплює асинхронну роботу з наперед записаним контентом, що зміщує технічні акценти з мінімізації затримки відтворення на надійність завантаження, зберігання та керування доступом.

Проведений аналіз предметної області окреслює склад сутностей, процесів і операцій, які має підтримувати розроблювана платформа, і слугує підставою для огляду наявних рішень та формування вимог до системи, поданих у наступних підрозділах.

1.2 Огляд та аналіз існуючих рішень

Для обґрунтування потреби в розроблюваній платформі та виявлення її ніші виконано огляд наявних рішень, що забезпечують запис, зберігання та дистрибуцію відеоконтенту. Розглянуто чотири поширені продукти — Loom, Vimeo, YouTube і Screencastify, — які охоплюють спектр від асинхронної відеокомунікації до масового відеохостингу. Для кожного визначено ключові можливості та обмеження з погляду цільового сценарію роботи.

Loom — платформа асинхронної відеокомунікації, що належить компанії Atlassian. Вона забезпечує запис екрана й камери одним натисканням через

розширення браузера та настільний застосунок, миттєве створення посилання на запис і вбудовування відео у сторонні сервіси, автоматичне формування субтитрів, аналітику переглядів і захищене поширення з паролем та обмеженням за доменом [4]. Основними обмеженнями є пропрієтарність і робота виключно як хмарного сервісу: безкоштовний тариф обмежує тривалість запису й кількість відео, а розширені можливості доступні лише за передплатою. Користувач не контролює місце зберігання даних і не може розгорнути рішення у власній інфраструктурі.

Vimeo — платформа професійного відеохостингу з розвиненими засобами керування приватністю. Вона підтримує кілька рівнів видимості (публічний, приватний, прихований, за паролем, для організації), обмеження вбудовування за доменом, аналітику переглядів і корпоративну автентифікацію через SSO [5]. Vimeo орієнтована передусім на зберігання та показ готових відео, а не на швидкий запис робочих повідомлень; більшість розширених налаштувань приватності доступні лише на платних тарифах. Як і Loom, це закритий хмарний сервіс без можливості самостійного розгортання.

YouTube — найбільша платформа масового відеохостингу та дистрибуції, що належить компанії Google. Вона надає три рівні видимості (публічний, за посиланням, приватний), автоматичні субтитри, детальну аналітику та монетизацію, а доставка контенту оптимізована під широке охоплення й алгоритмічні рекомендації [6]. Платформа призначена насамперед для публічного поширення, а не для приватного обміну робочим відео: вона фінансується рекламою, а режим доступу за посиланням не гарантує конфіденційності, оскільки посилання може бути вільно поширене. Запис екрана та керування власним сховищем засобами платформи не передбачені.

Screencastify — розширення браузера Chrome для запису екрана, вкладки або камери з редагуванням безпосередньо у браузері, автоматичною транскрипцією на платних тарифах та інтеграцією із сервісами Google [7]. Продукт орієнтований на освітній сегмент і забезпечує швидкий запис та поширення посиланням. Його суттєвими обмеженнями є прив’язка до браузера Chrome, відсутність підтримки інших платформ, обмеження безкоштовного тарифу за кількістю та тривалістю

відео, а також зберігання даних у хмарі постачальника без можливості самостійного розгортання.

Узагальнене порівняння розглянутих рішень із розроблюваною платформою за ключовими критеріями наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння розроблюваної платформи з наявними аналогами

Критерій	**Loom**	**Vimeo**	**YouTube**	**Screencastify**	**Розроблювана система**
Запис екрана у браузері	так	частково	ні	так	так
Автоматичні субтитри	так	так	так	частково	так
Контроль видимості	так	так	так	так	так
Самостійне розгортання й контроль сховища	ні	ні	ні	ні	так
Відкритий код, розширюваність	ні	ні	ні	ні	так
Модель доступу	підписка	підписка	реклама	підписка	власний хостинг

Проведений огляд показує, що наявні рішення є зрілими хмарними сервісами з широкою функціональністю, проте всі вони пропрієтарні, прив’язані до інфраструктури постачальника й обмежують частину можливостей передплатою. Жодне з них не дає змоги самостійно розгорнути платформу та контролювати місце зберігання контенту.

1.3 Постановка завдання та формування вимог

На основі проведеного аналізу предметної області та огляду наявних рішень сформульовано завдання розробки. Необхідно створити веборієнтовану платформу менеджменту й дистрибуції цифрового відеоконтенту, яка відтворює ключові сценарії роботи з відео — запис, завантаження, зберігання, керування метаданими, автоматичну транскрипцію, контроль видимості та поширення — і водночас усуває

основні обмеження аналогів: пропрієтарність, прив’язку до інфраструктури постачальника та обмеження можливостей передплатою. Платформа має бути розгортуваним рішенням на основі Next.js і безсерверних технологій із повним контролем над сховищем контенту.

Сформульовані вимоги поділено на функціональні, що визначають дії системи, та нефункціональні, що визначають її якісні характеристики [8]. Такий поділ відповідає усталеній практиці інженерії вимог і дає змогу окремо перевіряти повноту функціональності та відповідність якісним обмеженням.

1.3.1 Функціональні вимоги

Функціональні вимоги описують дії, які система має виконувати для досягнення поставленої мети. Для розроблюваної платформи визначено такі функціональні вимоги:

1) реєстрація та автентифікація користувача за допомогою електронної пошти й пароля або облікового запису Google; операції з контентом доступні лише автентифікованим користувачам;
2) запис екрана та звуку засобами браузера з подальшим переходом до завантаження записаного матеріалу;
3) видача короткочасних підписаних URL-адрес і пряме завантаження відеофайлу й обкладинки у хмарне сховище без потреби в публічному доступі до запису;
4) збереження метаданих контенту: заголовка, опису, тегів, тривалості та рівня видимості;
5) автоматичне формування субтитрів для завантаженого відео з кешуванням результату та коректним опрацюванням матеріалів без звукової доріжки;
6) перегляд переліку доступних матеріалів із пошуком за назвою та тегами, сортуванням і посторінковим виведенням;

7) перегляд окремого матеріалу з обліком кількості переглядів, причому приватні матеріали доступні лише їх власникові;
8) керування рівнем видимості матеріалу (публічний або приватний) його власником;
9) поширення матеріалу через посилання на сторінку перегляду;
10) видалення матеріалу разом із відповідними об’єктами у сховищі;
11) розмежування прав доступу: змінювати чи видаляти матеріал може лише його власник.

Наведені функціональні вимоги охоплюють повний життєвий цикл одиниці контенту — від запису до дистрибуції — і визначають межі функціональності системи.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи, що не залежать від конкретної функції, але впливають на її надійність, безпеку та зручність використання. Для розроблюваної платформи визначено такі нефункціональні вимоги:

1) масштабованість — виконання серверної логіки у функціях безсерверного середовища на вимогу, без постійного резервування обчислювальних ресурсів;
2) продуктивність завантаження — пряме завантаження файлів у сховище за підписаними URL-адресами без проходження через сервер застосунку, із підтримкою файлів обсягом до 50 МБ;
3) безпека — захист звернень засобами проміжного шару (екранування підозрілих запитів, виявлення автоматизованих клієнтів та обмеження частоти запитів), недоступність службового ключа сховища на боці клієнта, відсутність анонімного запису у сховище, перевірка автентифікації та прав доступу на рівні серверних дій;

4) надійність і цілісність даних — централізована обробка помилок серверних операцій, відкат завантаження з видаленням уже збережених об’єктів у разі збою, каскадне видалення пов’язаних записів;
5) зручність використання — простий і послідовний інтерфейс основних сценаріїв та локалізація інтерфейсу українською й англійською мовами;
6) підтримуваність і розширюваність — ізоляція доступу до даних у виділеному шарі серверних дій, дотримання принципів чистої архітектури та SOLID, уникнення антипатерну надмірно перевантаженого об’єкта;
7) переносність розгортання — конфігурування через змінні середовища та незалежність від конкретного постачальника в межах безсерверного середовища.

Сформульовані функціональні та нефункціональні вимоги є основою для визначення акторів і варіантів використання системи, поданих у підрозділі 1.4.

1.4 Актори та варіанти використання

Для формалізації поведінки системи на початковому етапі проєктування використано уніфіковану мову моделювання UML, зокрема діаграму варіантів використання [2]. Діаграма описує взаємодію зовнішніх дійових осіб (акторів) із системою та її ключові сценарії, що дає змогу узгодити функціональні вимоги з очікуваною поведінкою системи.

У системі виділено двох основних акторів. Зареєстрований користувач (автор) є головним актором: він автентифікується, записує та завантажує контент, керує метаданими й видимістю, переглядає, поширює та видаляє власні матеріали. Гість — неавтентифікований відвідувач, який може лише переглядати публічні матеріали за посиланням. Допоміжними зовнішніми акторами є провайдер автентифікації Google, сервіс транскрипції та об’єктне сховище, які забезпечують відповідні функції системи.

Діаграму варіантів використання системи наведено на рисунку 1.1.

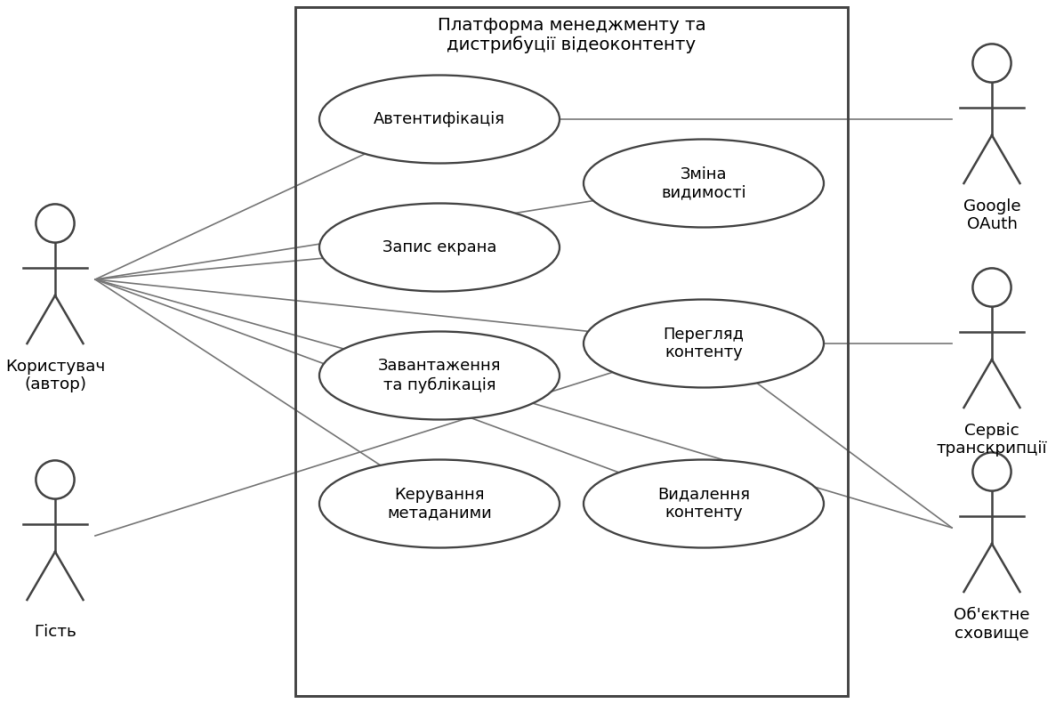

Рисунок 1.1 – Діаграма варіантів використання

До ключових варіантів використання належать автентифікація, запис екрана, завантаження та публікація контенту, керування метаданими, зміна видимості, перегляд контенту та видалення контенту. Більшість сценаріїв доступна лише авторові, тоді як перегляд публічних матеріалів доступний також гостю.

Сценарії пов’язані між собою відношеннями включення та розширення: завантаження контенту передбачає обов’язкову автентифікацію автора й супроводжується формуванням метаданих, а запис екрана виступає окремим джерелом матеріалу, що переходить до того самого сценарію завантаження. Допоміжні актори долучаються опосередковано: провайдер автентифікації Google підтверджує особу користувача під час входу, об’єктне сховище приймає відеофайл та обкладинку за підписаними URL-адресами, а сервіс транскрипції асинхронно формує субтитри після завантаження.

Детальну специфікацію ключового варіанта використання «Завантаження та публікація контенту» наведено в таблиці 1.2.

Таблиця 1.2 – Специфікація варіанта використання «Завантаження та публікація контенту»

Елемент	Опис
Назва	Завантаження та публікація контенту
Актор	Зареєстрований користувач (автор)
Передумови	Користувач автентифікований; підготовлено відеофайл (запис екрана або імпорт) та зображення обкладинки
Основний потік	1) система видає підписані URL-адреси для відео й обкладинки; 2) браузер завантажує файли безпосередньо у сховище; 3) користувач заповнює метадані (заголовок, опис, теги, рівень видимості); 4) система зберігає запис контенту й повертає його ідентифікатор
Альтернативні сценарії	У разі збою завантаження система видаляє вже збережені об’єкти; у разі перевищення дозволеної частоти запитів операція відхиляється
Постумови	Матеріал збережено та доступний відповідно до рівня видимості; для відео зі звуком асинхронно формуються субтитри

Визначені актори та варіанти використання узгоджують функціональні вимоги з очікуваною поведінкою системи й завершують аналіз вимог. У розділі проаналізовано предметну область менеджменту та дистрибуції відеоконтенту, оглянуто наявні рішення та виявлено нішу розроблюваної платформи, сформовано функціональні й нефункціональні вимоги та побудовано модель варіантів використання. Отримані результати є вихідними даними для проєктування архітектури, моделі даних і програмної реалізації системи, поданих у розділі 2.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

Розділ присвячено проєктуванню та програмній реалізації веборієнтованої платформи менеджменту й дистрибуції відеоконтенту відповідно до вимог, сформованих у розділі 1. Проєктні рішення спрямовані на масштабованість, безпеку та підтримуваність системи за умови безсерверного розгортання.

У розділі обґрунтовано вибір технологій та засобів реалізації, описано архітектуру застосунку з поділом на шари, спроєктовано модель даних і схему бази даних, наведено програмну реалізацію ключових модулів і реалізацію інтерфейсу користувача. Проєктування та реалізацію виконано з дотриманням принципів чистої архітектури та SOLID.

Викладення починається з обґрунтування технологічного стеку, що визначає подальші архітектурні та реалізаційні рішення.

2.1 Обґрунтування вибору технологій та засобів реалізації

Вибір технологій підпорядковано сформованим вимогам: безсерверному масштабуванню, безпеці, продуктивному завантаженню медіафайлів, типобезпеці та підтримуваності коду. З огляду на ці критерії для реалізації обрано стек на основі екосистеми Next.js, придатний для розгортання у безсерверному середовищі без постійного резервування обчислювальних ресурсів [9].

Основою системи є фреймворк Next.js 15 з маршрутизатором App Router і компонувальником Turbopack, мова TypeScript та бібліотека React 19. Next.js поєднує рендеринг на боці сервера, серверні компоненти та серверні дії (Server Actions), що дає змогу розміщувати бізнес-логіку поряд з інтерфейсом і виконувати її у функціях безсерверного середовища [9]. TypeScript забезпечує статичну типізацію, яка зменшує кількість помилок на етапі компіляції та підвищує підтримуваність коду.

Для оформлення інтерфейсу використано утилітарний CSS-фреймворк Tailwind CSS, а допоміжні бібліотеки clsx і tailwind-merge спрощують умовне

формування переліку класів без конфліктів стилів.

Автентифікацію реалізовано засобами бібліотеки Better Auth, що підтримує вхід за електронною поштою й паролем та через обліковий запис Google за протоколом OAuth. Вибір зумовлено тісною інтеграцією з Next.js, серверним зберіганням сесій і відсутністю потреби в окремому сервері автентифікації.

Метадані контенту й облікові записи зберігаються в реляційній базі даних PostgreSQL, доступ до якої здійснюється через об’єктно-реляційне відображення Drizzle ORM із драйвером postgres. Drizzle забезпечує типобезпечні запити, узгоджені зі схемою, та керування міграціями засобами drizzle-kit, а PostgreSQL сумісний із безсерверними середовищами завдяки пулу з’єднань.

Відеофайли та зображення обкладинок зберігаються в об’єктному сховищі Supabase Storage. Завантаження виконується за короткочасними підписаними URL-адресами безпосередньо з браузера, що знімає навантаження із сервера застосунку та усуває потребу в публічному доступі на запис до сховища.

Автоматичне формування субтитрів реалізовано через зовнішній сервіс розпізнавання мовлення AssemblyAI, який повертає субтитри у форматі WebVTT; результат кешується в базі даних, щоб уникати повторного оброблення під час кожного перегляду.

Захист застосунку забезпечує сервіс Arcjet, інтегрований у проміжний шар і серверні дії: він екранує підозрілі запити, виявляє автоматизованих клієнтів та обмежує частоту звернень, а на етапі автентифікації додатково перевіряє коректність електронної пошти.

Локалізацію інтерфейсу українською та англійською мовами реалізовано засобами бібліотеки next-intl із вибором мови на рівні застосунку.

Підсумок використаного технологічного стеку та призначення кожного складника наведено в таблиці 2.1.

Таблиця 2.1 – Технологічний стек платформи

Складник	Технологія	Призначення
Фреймворк і мова	Next.js 15, React 19, TypeScript	рендеринг на сервері, серверні дії, статична типізація
Оформлення	Tailwind CSS, clsx, tailwind-merge	стилізація інтерфейсу
Автентифікація	Better Auth	вхід за поштою й паролем та через Google OAuth
База даних	PostgreSQL, Drizzle ORM	зберігання метаданих контенту й облікових записів
Сховище	Supabase Storage	зберігання відео й обкладинок за підписаними URL-адресами
Транскрипція	AssemblyAI	автоматичні субтитри у форматі WebVTT
Безпека	Arcjet	екранування запитів, виявлення ботів, обмеження частоти
Локалізація	next-intl	інтерфейс українською та англійською мовами

Обраний стек узгоджений із вимогами щодо масштабованості, безпеки й підтримуваності та визначає архітектурні рішення, розглянуті в підрозділі 2.2.

2.2 Архітектура застосунку

Архітектуру застосунку побудовано так, щоб розмежувати відповідальність між шарами, забезпечити масштабованість у безсерверному середовищі та спростити супровід. В основу покладено багатошарову організацію з однонапрямленою спрямованістю залежностей: зовнішні шари залежать від внутрішніх, але не навпаки [10].

Систему поділено на чотири основні шари — клієнтський інтерфейс, проміжний шар, шар застосунку із серверними діями та шар доступу до даних, — а також зовнішні сервіси, що надають автентифікацію, транскрипцію та зберігання. Загальну архітектуру наведено на рисунку 2.1.

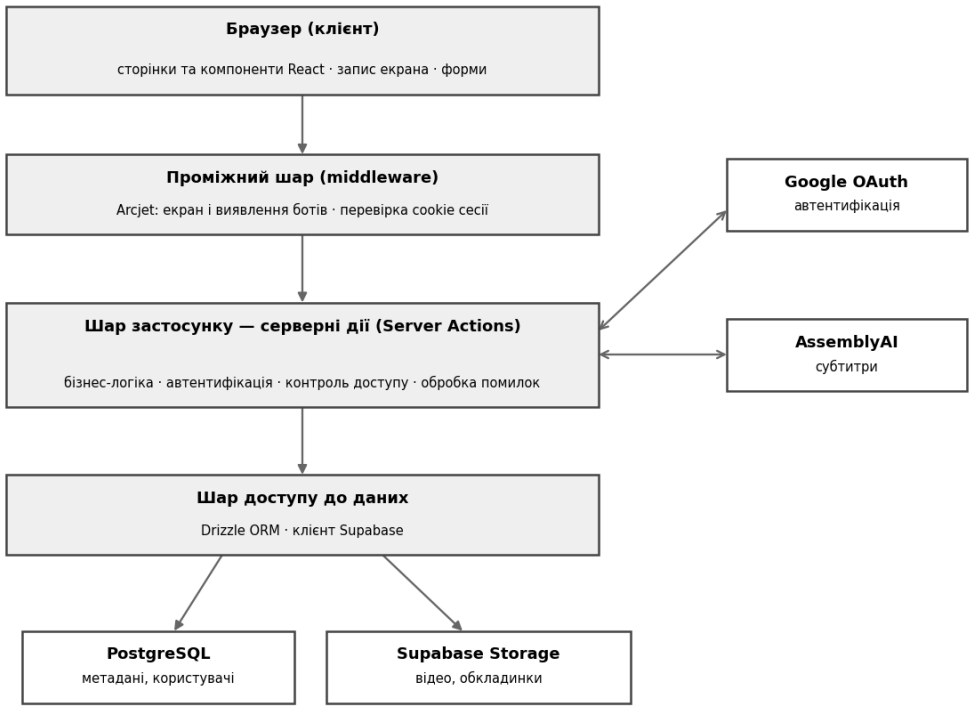

Рисунок 2.1 – Багатошарова архітектура застосунку

Клієнтський шар утворюють сторінки та компоненти React у каталозі app, що відповідають за відображення інтерфейсу, запис екрана й заповнення форм. Проміжний шар (middleware) перехоплює кожен запит, застосовує захист Arcjet (екранування та виявлення ботів) і виконує оптимістичну перевірку cookie сесії в середовищі Edge без звернення до бази даних. Шар застосунку реалізовано серверними діями з директивою «use server», які зосереджують бізнес-логіку, автентифікацію, контроль доступу та обробку помилок. Шар доступу до даних інкапсулює роботу з базою даних через Drizzle ORM і з об’єктним сховищем через клієнт Supabase. Зовнішні сервіси — Google OAuth, AssemblyAI та сховище — взаємодіють із системою через визначені інтерфейси.

Обрана модель Next.js App Router передбачає рендеринг компонентів на боці сервера й виконання серверних дій як функцій безсерверного середовища на вимогу, без постійно запущеного сервера застосунку [9; 11]. Повну перевірку сесії винесено в серверні дії та сторінки, тоді як проміжний шар виконує лише швидку оптимістичну перевірку наявності cookie, придатну для середовища Edge. Такий

поділ зменшує затримку маршрутизації й не покладає на крайовий шар операцій, що потребують доступу до бази даних.

Фізичну структуру проєкту, що відображає поділ на шари, наведено в лістингу 2.1.

Лістинг 2.1 – Структура проєкту

```
.
├── app/              # App Router: маршрути, макети, API
│   ├── (auth)/       # потік входу
│   ├── (root)/       # головна, завантаження, відео, профіль
│   ├── api/auth/     # обробник Better Auth + Arcjet
│   └── layout.tsx    # кореневий макет + провайдер i18n
├── components/       # інтерфейс: плеєр, рекордер, навбар
├── lib/              # серверні дії, автентифікація, утиліти
│   ├── actions/      # доступ до даних ("use server")
│   └── hooks/        # запис екрана, вибір файлу
├── drizzle/          # схема, клієнт БД, міграції
├── i18n/             # конфігурація next-intl
├── messages/         # переклади en.json, uk.json
├── public/           # статичні ресурси
└── middleware.ts     # проміжний шар: Arcjet + cookie
```

Каталог app містить маршрути, макети та обробник автентифікації; каталог components — елементи інтерфейсу; каталог lib — серверні дії (actions), конфігурацію автентифікації, клієнти сховища, хуки та допоміжні функції; каталог drizzle — схему й клієнт бази даних; каталоги i18n та messages — конфігурацію й каталоги перекладів. Файл middleware.ts реалізує проміжний шар. Така організація закріплює межі між шарами на рівні структури каталогів.

Дотримання принципів чистої архітектури та SOLID забезпечено кількома рішеннями. Доступ до даних зосереджено винятково в серверних діях і шарі Drizzle, тож компоненти інтерфейсу не звертаються до бази даних безпосередньо. Кожна серверна дія виконує одну задачу (принцип єдиної відповідальності), а наскрізні аспекти — отримання ідентифікатора сесії, перевірку прав власника, обмеження частоти запитів і обробку помилок — винесено в окремі багаторазові функції (getSessionUserId, assertVideoOwner, validateWithArcjet, withErrorHandling). Це запобігає утворенню антипатерну надмірно перевантаженого об’єкта («God

Object»), коли одна сутність зосереджує невластиву їй логіку.

Централізована обгортка обробки помилок withErrorHandling уніфікує реагування на збої в усіх серверних діях, а пакет server-only унеможливлює випадкове використання серверного коду й службових ключів на боці клієнта. Описана архітектура визначає місце моделі даних, проєктування якої розглянуто в підрозділі 2.3.

Узагальнену архітектурну модель системи з її основними складниками та зв'язками між ними наведено на рисунку 2.2. Клієнтська частина (інтерфейс на React у складі Next.js) взаємодіє із сервером Next.js, де серверні дії виконують бізнес-логіку; усі запити попередньо проходять через проміжний шар, що поєднує захист Arcjet і перевірку автентифікації Better Auth [10].

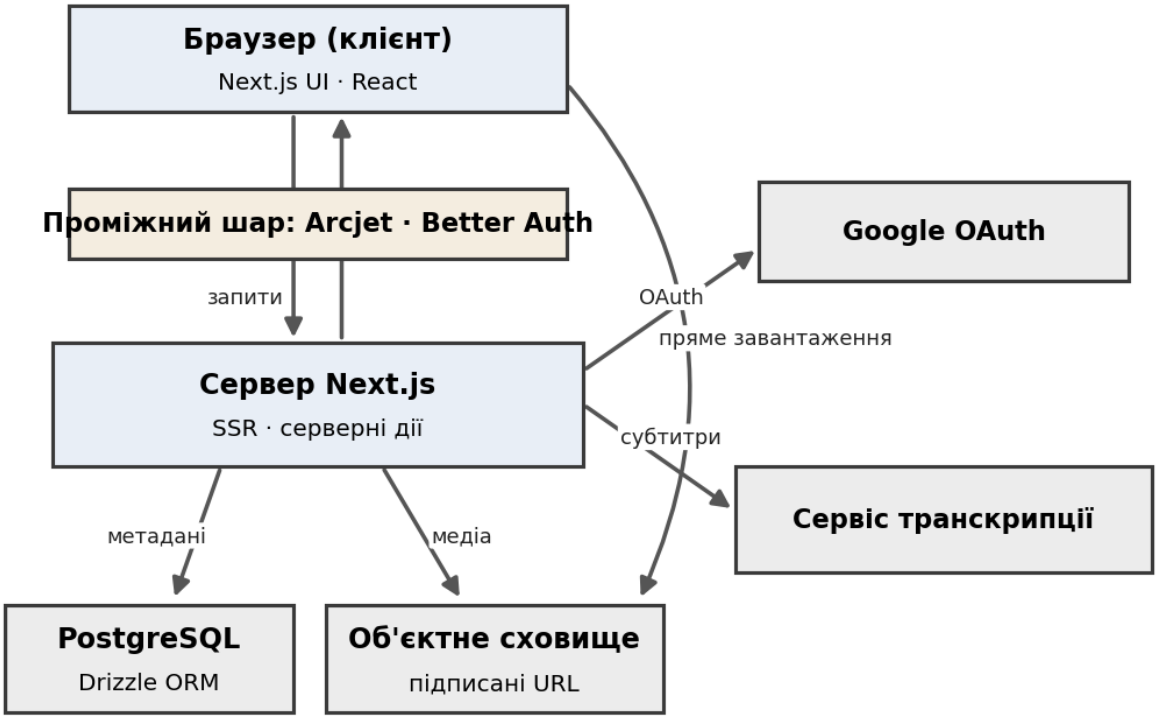

Рисунок 2.2 – Архітектурна модель системи

Метадані контенту й облікові записи зберігаються в реляційній базі даних через ORM, медіафайли — в об'єктному сховищі, доступ до якого надається за підписаними URL-адресами, а субтитри формуються зовнішнім сервісом

транскрипції. Зовнішні сервіси відокремлені від ядра застосунку, що дає змогу масштабувати систему в безсерверному середовищі без постійного резервування ресурсів [11].

Послідовність взаємодії складників під час основного сценарію — від запису екрана до перегляду відео із субтитрами — наведено на рисунку 2.3.

Рисунок 2.3 – Потік даних основного сценарію

Користувач записує екран у браузері, після чого клієнт запитує в серверної дії підписані URL-адреси та завантажує медіафайли безпосередньо в об'єктне сховище; Розмежування завантаження файлів і збереження метаданих знижує навантаження на серверні функції та відповідає принципам безсерверної архітектури [11].

2.3 Проєктування моделі даних і схеми бази даних

Модель даних відображає сутності предметної області та забезпечує цілісне зберігання метаданих контенту й облікових записів. Її спроєктовано за реляційною

моделлю з дотриманням цілісності через первинні та зовнішні ключі [12]. Виокремлені на етапі аналізу сутності — користувач і одиниця контенту — доповнено службовими сутностями, потрібними для автентифікації.

Схема бази даних PostgreSQL складається з п’яти таблиць. Таблиця user зберігає облікові записи користувачів, таблиця videos — метадані відеоматеріалів, а таблиці session, account і verification обслуговують автентифікацію засобами Better Auth (сесії, зовнішні облікові записи та підтвердження). Центральними для предметної області є таблиці user і videos, пов’язані відношенням «один до багатьох»: один користувач може мати багато матеріалів.

Концептуальну схему бази даних із сутностями та зв’язками наведено на рисунку 2.4.

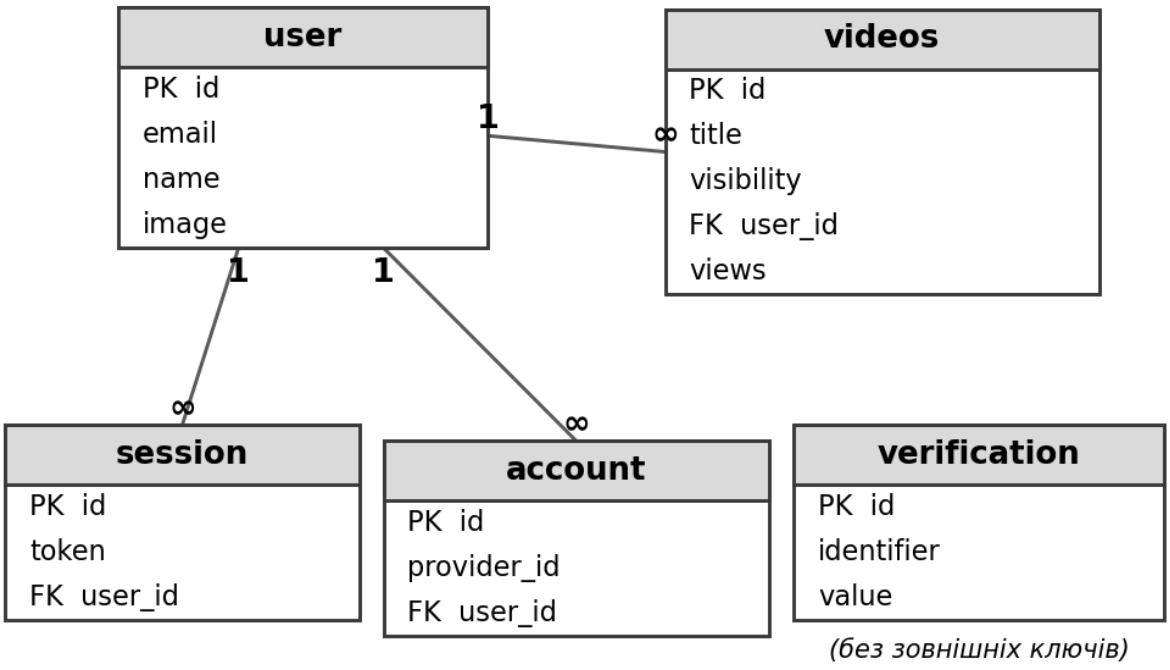

Рисунок 2.4 – Схема бази даних

Основною таблицею предметної області є videos, структуру якої наведено в таблиці 2.2.

Таблиця 2.2 – Структура таблиці videos

Поле	Тип	Призначення
id	uuid	первинний ключ, генерується автоматично
title	text	заголовок матеріалу
description	text	опис матеріалу
video_url	text	посилання на відеофайл у сховищі
video_id	text	публічний ідентифікатор матеріалу
thumbnail_url	text	посилання на обкладинку
visibility	text	рівень видимості: public або private
tags	text[]	перелік тегів
transcript	text	кешований текст субтитрів (може бути відсутнім)
user_id	text	зовнішній ключ на user.id (автор матеріалу)
views	integer	лічильник переглядів
duration	integer	тривалість відео, секунд
created_at	timestamp	час створення запису
updated_at	timestamp	час останнього оновлення

Первинним ключем таблиці videos є ідентифікатор типу UUID, що генерується на боці бази даних і не розкриває порядку створення записів. Поле visibility обмежено значеннями public і private та керує доступом до матеріалу. Перелік тегів зберігається як масив рядків, що спрощує тематичну класифікацію без додаткової таблиці. Поле transcript є необов’язковим і містить кешований текст субтитрів, а лічильник переглядів і часові позначки мають значення за замовчуванням.

Зв’язок між матеріалом та його автором реалізовано зовнішнім ключем user_id, що посилається на user.id з каскадним видаленням: разом з обліковим записом вилучаються й пов’язані з ним матеріали. Аналогічні зовнішні ключі з каскадним видаленням пов’язують із користувачем таблиці session та account. Таке обмеження цілісності запобігає появі записів-сиріт.

Схему описано декларативно засобами Drizzle ORM, а її версії та зміни структури бази даних відстежуються міграціями. Модель даних є підґрунтям для програмної реалізації ключових модулів, розглянутої в підрозділі 2.4.

2.4 Програмна реалізація ключових модулів

У підрозділі описано програмну реалізацію ключових модулів системи: автентифікації та контролю доступу, запису екрана, завантаження контенту, серверних дій доступу до даних і автоматичної транскрипції. Фрагменти реалізації подано мовою TypeScript.

2.4.1 Автентифікація та контроль доступу

Автентифікацію реалізовано засобами Better Auth із підтримкою входу за електронною поштою й паролем та через обліковий запис Google; конфігурацію адаптера бази даних і провайдерів задано централізовано. Сторінку входу наведено на рисунку 2.3.

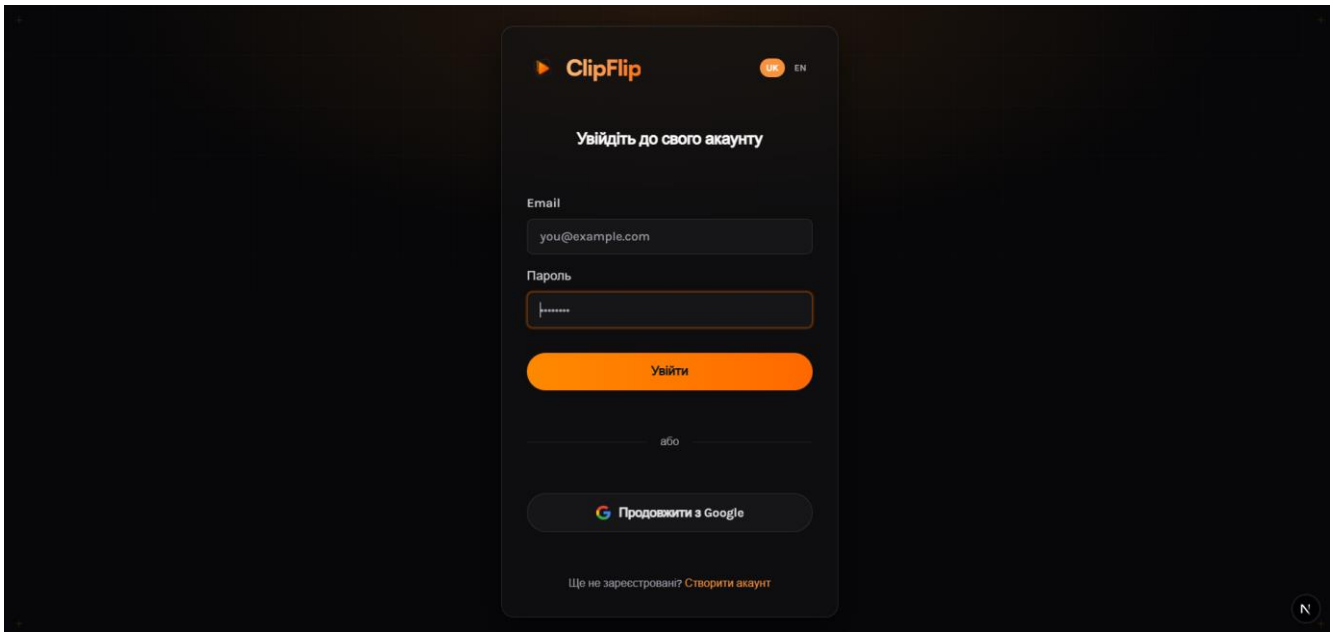

Рисунок 2.3 – Сторінка входу

Контроль доступу реалізовано на рівні серверних дій. Допоміжна функція отримання ідентифікатора сесії перериває виконання для неавтентифікованих користувачів, а перевірка власника не дозволяє змінювати чужі матеріали (лістинг 2.2).

Лістинг 2.2 – Перевірка автентифікації та прав власника

```
const getSessionUserId = async (): Promise<string> => {
  const session = await auth.api.getSession({
    headers: await headers(),
  });
  if (!session) throw new Error("Unauthenticated");
  return session.user.id;
};

const assertVideoOwner = async (videoId: string) => {
  const userId = await getSessionUserId();
  const [video] = await db
    .select()
    .from(videos)
    .where(eq(videos.videoId, videoId));
  if (video.userId !== userId) {
    throw new Error("No permission to modify this video");
  }
  return { video, userId };
};
```

Завдяки винесенню цих перевірок в окремі функції кожна серверна дія застосовує єдину логіку контролю доступу без її дублювання.

2.4.2 Запис екрана у браузері

Запис екрана реалізовано на боці клієнта хуком useScreenRecording на основі браузерних інтерфейсів захоплення екрана та MediaRecorder. В залежності від браузера можливості запиВікно запису наведено на рисунку 2.4.

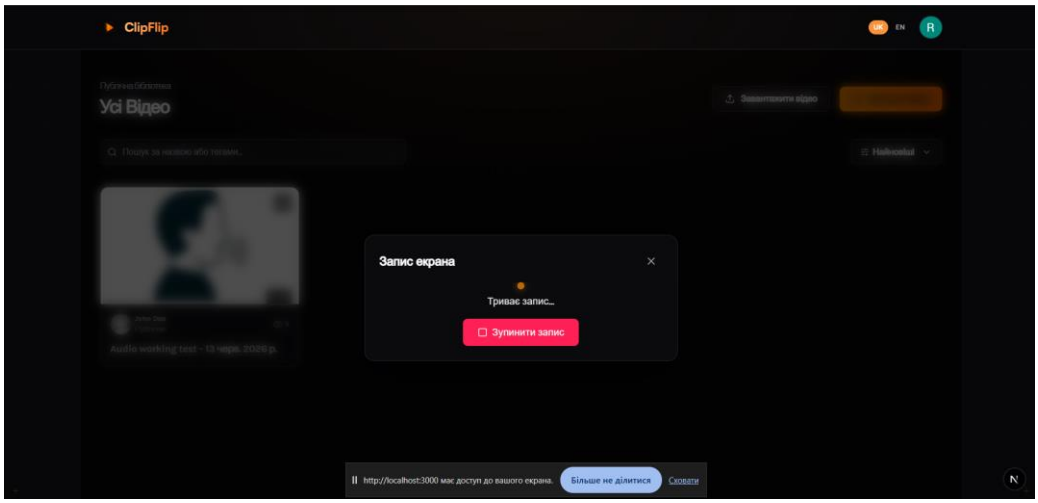

Рисунок 2.4 – Вікно запису екрана

Під час старту захоплюються відеопотік екрана та, за потреби, звук мікрофона; аудіоджерела зводяться в один потік через AudioContext, після чого комбінований потік передається в MediaRecorder (лістинг 2.3).

Лістинг 2.3 – Запуск запису екрана

```
const startRecording = async (withMic = true) => {
  stopRecording();
  const { displayStream, micStream, hasDisplayAudio } =
    await getMediaStreams(withMic);

  const combined = new MediaStream();
  displayStream
    .getVideoTracks()
    .forEach((t) => combined.addTrack(t));

  audioContextRef.current = new AudioContext();
  const dest = createAudioMixer(
    audioContextRef.current,
    displayStream, micStream, hasDisplayAudio,
  );
  dest?.stream
    .getAudioTracks()
    .forEach((t) => combined.addTrack(t));

  mediaRecorderRef.current = setupRecording(combined, {
    onDataAvailable: (e) =>
      e.data.size && chunksRef.current.push(e.data),
    onStop: handleRecordingStop,
  });
  mediaRecorderRef.current.start(1000);
```

Записаний матеріал зберігається у в оперативній пам’яті застосунку та передається на сторінку завантаження.

2.4.3 Завантаження контенту через підписані URL-адреси

Завантаження відео й обкладинки виконується безпосередньо з браузера в об’єктне сховище за підписаними URL-адресами. Форму завантаження наведено на рисунку 2.5.

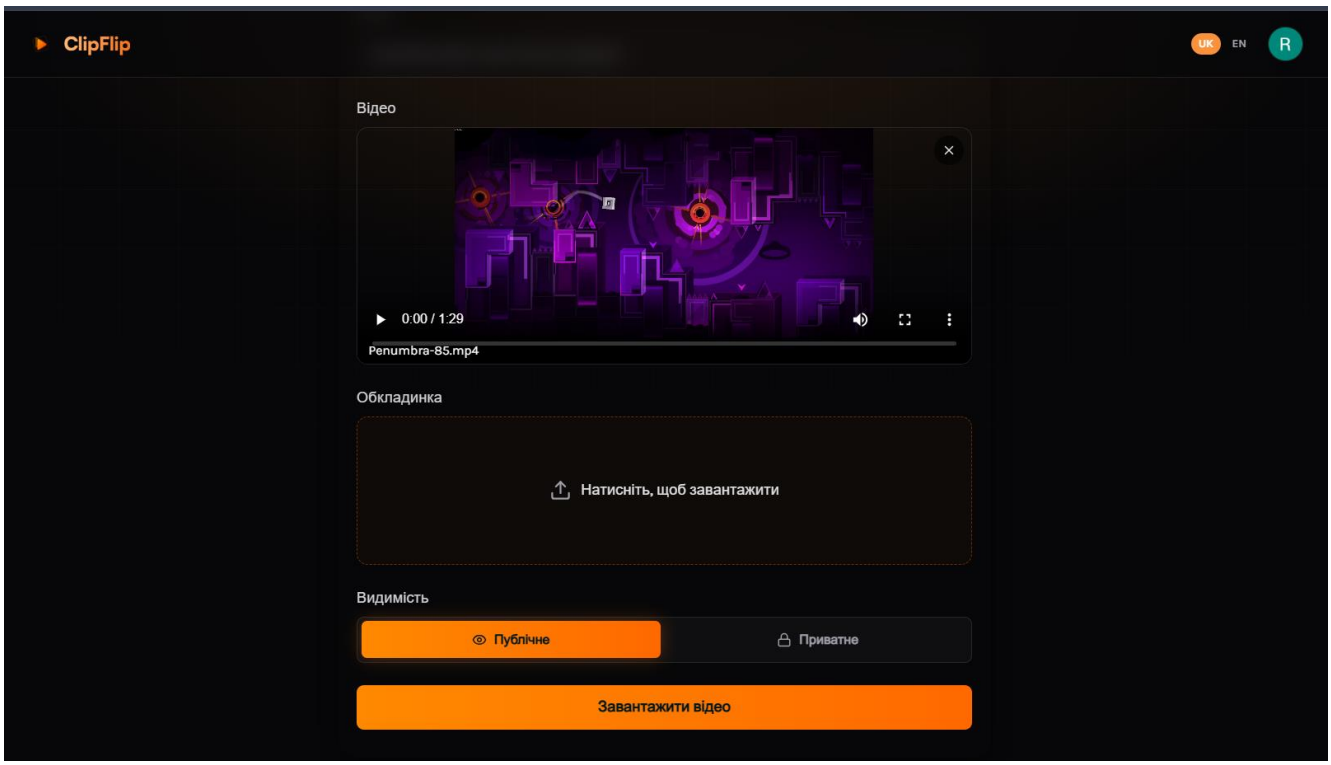

Рисунок 2.5 – Форма завантаження відео

Серверна дія createUploadUrls, доступна лише автентифікованим користувачам, формує короткочасні підписані URL-адреси для відео й обкладинки за допомогою службового клієнта сховища (лістинг 2.4).

Лістинг 2.4 – Формування підписаних URL-адрес

```
export const createUploadUrls = withErrorHandling(
  async (videoFileName, thumbnailFileName) => {
    await getSessionUserId();
    const videoId = crypto.randomUUID();
    const videoPath =
      `${videoId}-${cleanFileName(videoFileName)}`;
    const thumbPath =
      `thumb-${videoId}-${cleanFileName(thumbnailFileName)}`;

    const [videoSigned, thumbSigned] = await Promise.all([
      supabaseAdmin.storage.from(BUCKET)
        .createSignedUploadUrl(videoPath),
      supabaseAdmin.storage.from(BUCKET)
        .createSignedUploadUrl(thumbPath),
    ]);
    if (videoSigned.error || thumbSigned.error) {
      throw new Error("Failed to create upload URL");
    }
    return {
```

```
        videoId,
        video: { path: videoPath,
                 token: videoSigned.data.token },
        thumbnail: { path: thumbPath,
                     token: thumbSigned.data.token },
      };
    }
);
```

Браузер завантажує файли за отриманими адресами, після чого метадані зберігаються окремою серверною дією. У разі збою на будь-якому кроці вже завантажені об’єкти видаляються зі сховища, що запобігає появі неузгоджених даних.

2.4.4 Серверні дії доступу до даних

Доступ до метаданих контенту реалізовано серверними діями, що інкапсулюють запити до бази даних. Для зручності пошуку необхідного матеріалу реалізовано пошук який працює як і за назвою так і за тегами відповідного матеріалу. Крім цього є перелік матеріалів із пошуком, сортуванням і посторінковим виведенням наведено на рисунку 2.6.

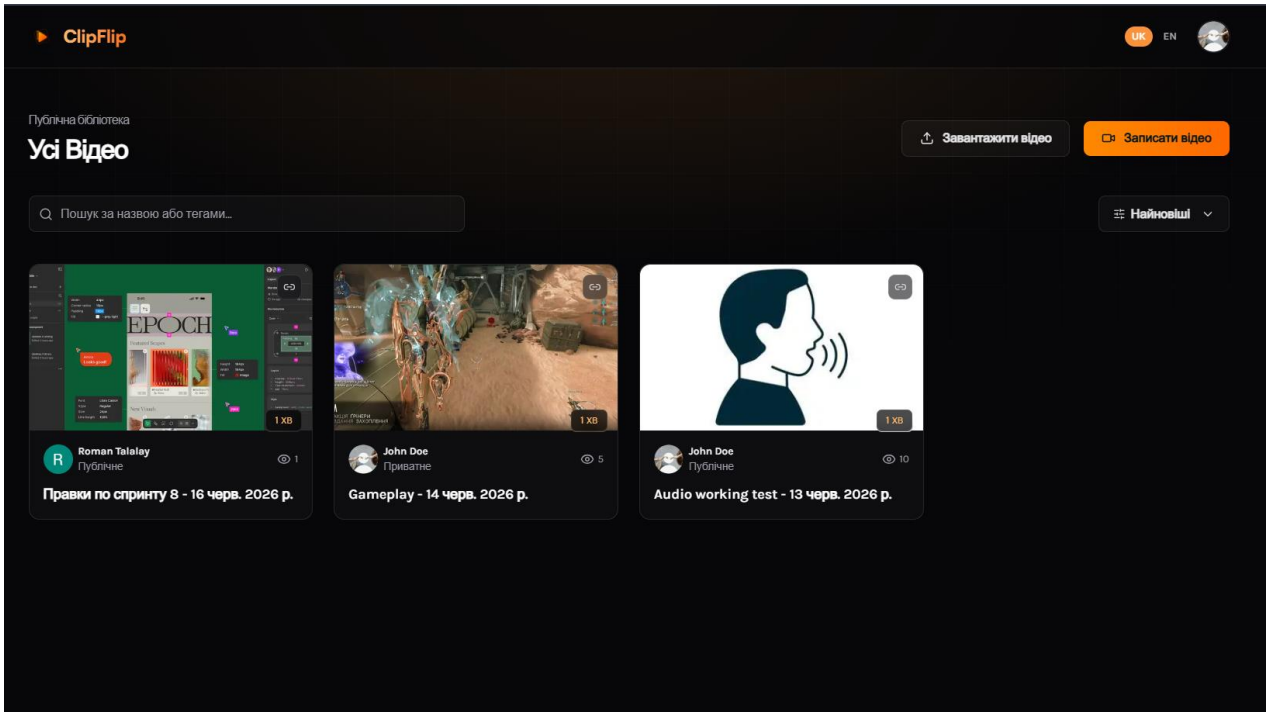

Рисунок 2.6 – Перелік відеоматеріалів

Дія отримання переліку матеріалів формує умову видимості, за якою повертаються лише публічні матеріали та власні матеріали поточного користувача, і застосовує пошук, сортування та межі вибірки (лістинг 2.5).

Лістинг 2.5 – Отримання переліку матеріалів

```
export const getAllVideos = withErrorHandling(
  async (searchQuery = "", sortFilter, page = 1, size = 8) => {
    const session = await auth.api.getSession({
      headers: await headers(),
    });
    const currentUserId = session?.user.id;

    const canSee = or(
      eq(videos.visibility, "public"),
      currentUserId
        ? eq(videos.userId, currentUserId)
        : undefined,
    );
    const where = searchQuery.trim()
      ? and(canSee, matchesSearch(videos, searchQuery))
      : canSee;

    const records = await buildVideoWithUserQuery()
      .where(where)
      .orderBy(sortFilter
        ? getOrderByClause(sortFilter)
        : sql`${videos.createdAt} DESC`)
      .limit(size)
      .offset((page - 1) * size);
    return { videos: records };
  }
);
```

Окремі серверні дії реалізують перегляд одного матеріалу з приховуванням приватних записів від сторонніх користувачів, зміну видимості, облік переглядів і видалення матеріалу разом із файлами у сховищі. Зміна видимості та збереження метаданих додатково обмежені за частотою запитів.

2.4.5 Автоматична транскрипція

Транскрипція формується автоматично зовнішнім сервісом розпізнавання мовлення. Відтворення відео із транскрипцією наведено на рисунку 2.7.

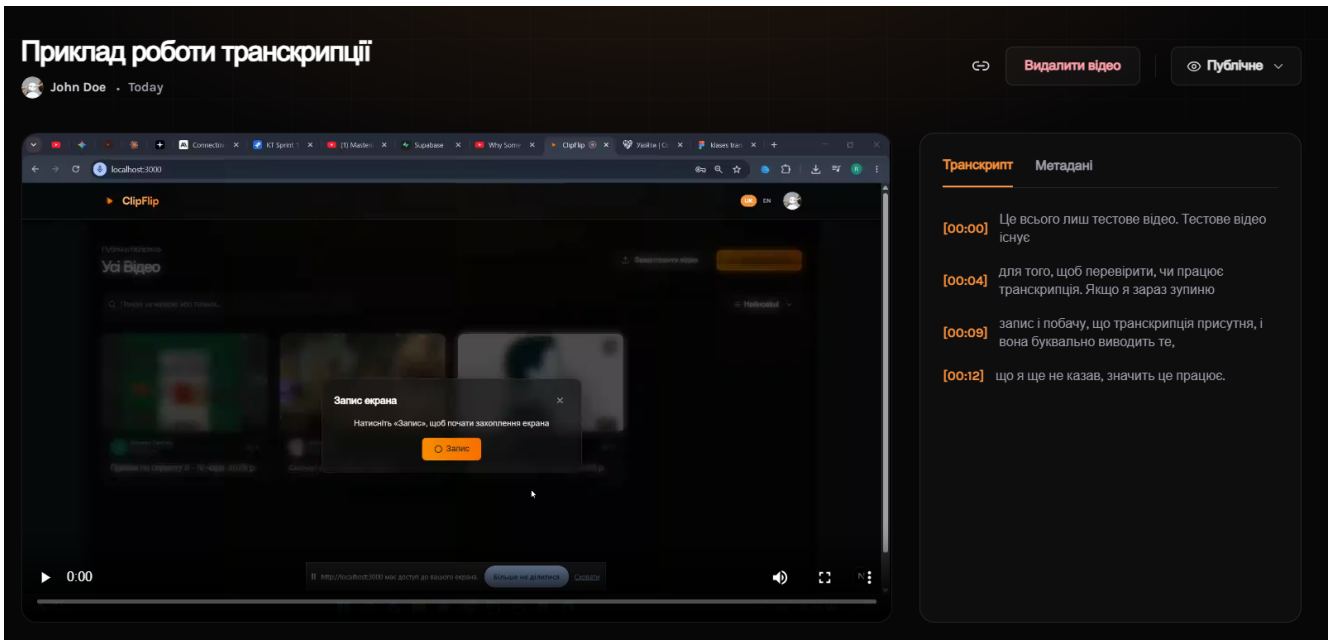

Рисунок 2.7 – Відтворення відео із транскрипцією

Серверна дія getTranscript повертає кешований текст транскрипції, а за його відсутності звертається до сервісу транскрипції та зберігає результат; відсутність звукової доріжки фіксується спеціальним маркером, щоб не повторювати оброблення (лістинг 2.6).

Лістинг 2.6 – Отримання та кешування транскрипції

```
export const getTranscript = withErrorHandling(
  async (videoId: string): Promise<TranscriptResult> => {
    const [video] = await db
      .select({
        transcript: videos.transcript,
        videoUrl: videos.videoUrl,
      })
      .from(videos)
      .where(eq(videos.videoId, videoId));
    if (!video) return { status: "ready", vtt: "" };
    if (video.transcript === NO_AUDIO_MARKER)
      return { status: "no_audio" };
    if (video.transcript)
      return { status: "ready", vtt: video.transcript };

    const vtt = await transcribeToVtt(video.videoUrl);
    await db.update(videos)
      .set({ transcript: vtt })
      .where(eq(videos.videoId, videoId));
    return { status: "ready", vtt };
  }
```

Звернення до сервісу AssemblyAI реалізовано асинхронно: завдання на транскрипцію створюється, його стан опитується періодично, а готові субтитри експортуються у форматі WebVTT. Реалізовані модулі забезпечують виконання функціональних вимог; реалізацію інтерфейсу користувача розглянуто в підрозділі 2.5.

2.5 Реалізація інтерфейсу користувача

Інтерфейс користувача побудовано з багаторазових компонентів React, оформлених засобами Tailwind CSS. Сторінки, що не потребують інтерактивності, рендеряться на боці сервера, а інтерактивні елементи позначено директивою «use client». Текст інтерфейсу винесено в окремі каталоги перекладів і підставляється засобами локалізації, що дає змогу перемикати мову без зміни коду компонентів.

Інтерфейс складається з кількох основних екранів: головної сторінки з переліком матеріалів, сторінок запису та завантаження, сторінки перегляду відео й сторінки профілю. Головна сторінка є серверним компонентом, що отримує матеріали серверною дією та компонує перелік із багаторазових компонентів, відображаючи порожній стан за відсутності матеріалів і посторінкову навігацію (лістинг 2.7).

Лістинг 2.7 – Композиція головної сторінки

```
const page = async ({ searchParams }) => {
  const { query, filter, page } = await searchParams;
  const t = await getTranslations("browse");
  const { videos, pagination } = await getAllVideos(
    query, filter, Number(page) || 1,
  );

  return (
    <main className="wrapper page">
      <SharedHeader title={t("allVideos")} />
      {videos?.length > 0 ? (
        <section className="video-grid">
          {videos.map(({ video, user }) => (
            <VideoCard
              key={video.id}
              id={video.videoId}
```

```
                title={video.title}
                thumbnail={video.thumbnailUrl}
                views={video.views}
                visibility={video.visibility}
              />
            ))}
          </section>
        ) : (
          <EmptyState title={t("emptyVideosTitle")} />
        )}
        {pagination?.totalPages > 1 && (
          <Pagination
            currentPage={pagination.currentPage}
            totalPages={pagination.totalPages}
          />
        )}
      </main>
    );
};
```

Повторно використовувані компоненти інкапсулюють окремі елементи інтерфейсу: компонент поля форми підтримує текстове поле, багаторядкове поле та список вибору; компонент завантаження файлу відображає попередній перегляд відео або зображення; компонент картки відео подає матеріал у переліку, а компонент посторінкової навігації формує елементи переходу між сторінками. Така композиція зменшує дублювання розмітки та узгоджує вигляд однотипних елементів.

Сторінка профілю відображає матеріали окремого користувача та дані його облікового запису. Її наведено на рисунку 2.8.

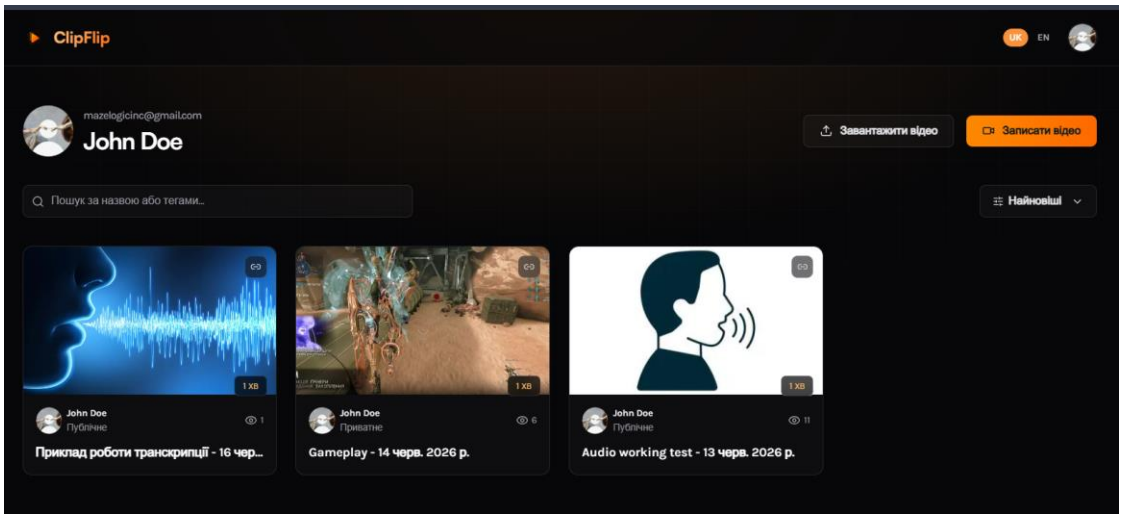

Рисунок 2.8 – Сторінка профілю користувача

Сторінка перегляду відео поєднує програвач, відомості про матеріал і панель транскрипції, сформованої автоматично. Її наведено на рисунку 2.9.

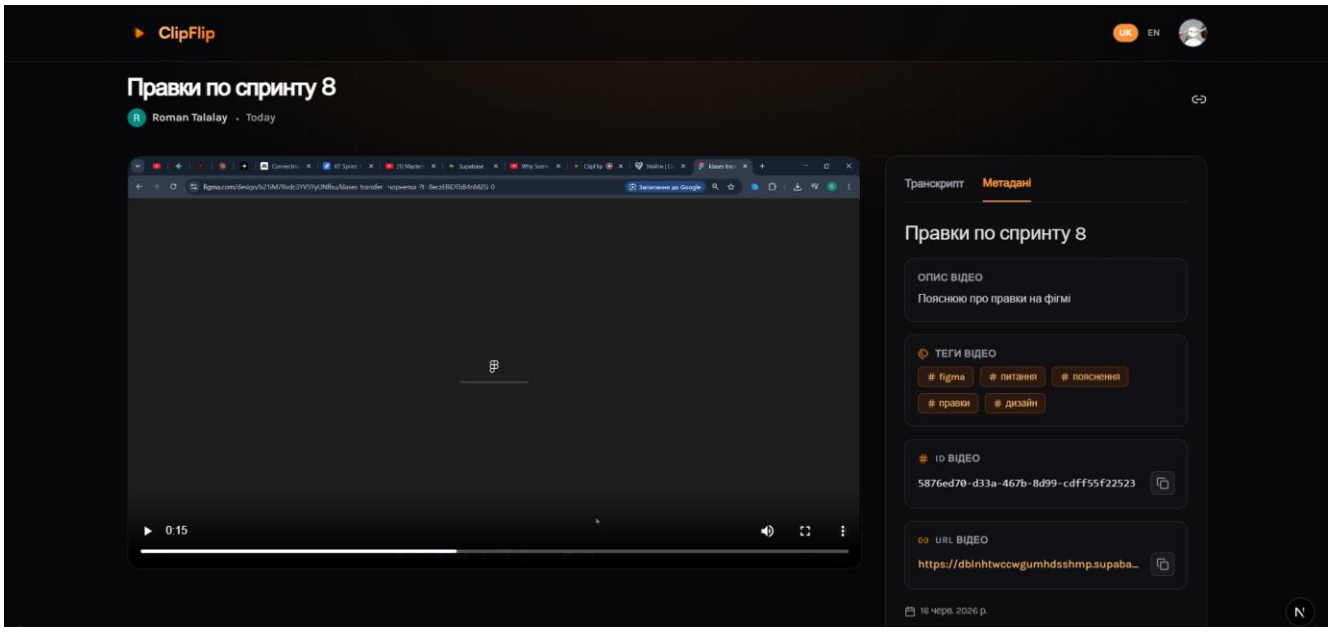

Рисунок 2.9 – Сторінка перегляду відео

Локалізацію інтерфейсу реалізовано бібліотекою next-intl: повідомлення зберігаються в окремих файлах для української та англійської мов, компоненти отримують підписи через відповідні функції перекладу, а перемикач мови змінює поточну мову й оновлює сторінку. Мовою інтерфейсу за замовчуванням є українська.

2.6 Висновки до розділу 2

У розділі обґрунтовано вибір технологічного стеку на основі Next.js і безсерверних технологій, спроєктовано багатошарову архітектуру застосунку з розмежуванням шарів інтерфейсу, серверних дій і доступу до даних та з дотриманням принципів чистої архітектури і SOLID. Спроєктовано модель даних і схему реляційної бази даних, а також виконано програмну реалізацію ключових модулів: автентифікації та контролю доступу, запису екрана, завантаження контенту за підписаними URL-адресами, серверних дій доступу до даних і автоматичної транскрипції, та реалізацію інтерфейсу користувача.

3 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

Розділ присвячено перевірці працездатності розробленої платформи, її впровадженню та оцінюванню відповідності сформованим у розділі 1 вимогам. Послідовно описано підхід до тестування й отримані результати, процес розгортання системи в безсерверному середовищі та аналіз покриття функціональних і нефункціональних вимог реалізацією. Метою цих робіт є підтвердження того, що реалізована система коректно виконує визначені сценарії та придатна до експлуатації.

Тестування спрямоване на підтвердження коректності ключових сценаріїв роботи та раннє виявлення дефектів, впровадження — на відтворюване й безпечне розгортання системи, а аналіз результатів — на оцінювання повноти реалізації вимог і досягнення мети роботи.

Виклад побудовано так: спершу описано тестування системи та його результати, далі — впровадження й розгортання системи, і нарешті — аналіз відповідності реалізації сформованим вимогам з оцінкою досягнення мети роботи.

3.1 Тестування програмної системи

Тестування програмної системи виконано на кількох рівнях: статичного аналізу коду, функціонального тестування ключових сценаріїв і перевірки відповідності вимогам. Поєднання автоматичного статичного аналізу з функціональною перевіркою дає змогу виявляти як технічні дефекти коду, так і відхилення поведінки системи від очікуваної, що узгоджується з усталеними практиками інженерії програмного забезпечення [8].

Зважаючи на архітектуру системи, основну увагу приділено статичному аналізу та функціональному тестуванню на системному рівні. Перевірку відповідності вимогам, що спирається на результати тестування, винесено в окремий підрозділ 3.3.

Рівні тестування та їх послідовність наведено на рисунку 3.1.

Рисунок 3.1 – Рівні тестування програмної системи

Кожен наступний рівень спирається на попередній: статичний аналіз гарантує технічну коректність коду, функціональне тестування підтверджує очікувану поведінку сценаріїв, а перевірка відповідності зіставляє реалізовані можливості з вимогами. Перші два рівні описано в цьому підрозділі.

3.1.1 Статичний аналіз коду

Статичний аналіз здійснюється на етапі збирання проєкту й не потребує запуску застосунку. Система типів мови TypeScript перевіряє узгодженість типів даних між модулями, сигнатури функцій і звернення до полів сутностей, унаслідок чого значну частину потенційних помилок виявляють до виконання коду. Статичний аналізатор ESLint додатково контролює дотримання прийнятих правил оформлення коду та попереджає про конструкції, що часто призводять до дефектів.

Збирання проєкту в робочому режимі поєднує перевірку типів і лінтинг, тому успішне завершення збирання можливе лише за відсутності помилок типізації та

порушень правил. Це робить збирання автоматичним бар’єром якості: код із виявленими статичним аналізом проблемами не може бути розгорнутий.

Окрім раннього виявлення помилок, статична типізація полегшує супровід коду: під час змін компілятор сигналізує про всі місця, яких стосується змінений тип або сигнатура функції, що знижує ризик неузгоджених правок. Це особливо важливо для серверних дій, які працюють зі спільною схемою бази даних.

Типовими проблемами, які виявляє статичний аналіз, є невідповідність типу аргументу очікуваному, звернення до неіснуючого поля сутності, відсутність обробки можливого порожнього значення та порушення домовленостей щодо структури коду. Виявлення таких проблем на етапі збирання, а не під час виконання, зменшує вартість їх усунення.

3.1.2 Функціональне тестування

Функціональне тестування полягало в ручній перевірці ключових сценаріїв використання системи. Для кожного сценарію було визначено початкові умови та очікуваний результат, після чого сценарій відтворювали в розгорнутому застосунку й зіставляли фактичний результат з очікуваним. Перевірялися як основні сценарії — реєстрація, вхід, запис екрана, завантаження, перегляд і керування матеріалами, — так і допоміжні операції.

Функціональне тестування виконували в розгорнутому застосунку із застосуванням сучасних браузерів, оскільки запис екрана спирається на браузерні інтерфейси захоплення зображення та звуку. Перевіряли як сценарії автентифікованого користувача, так і поведінку системи для неавтентифікованого відвідувача.

Для відтворення сценаріїв підготовлено тестові облікові записи та зразкові відеоматеріали з різними рівнями видимості. Сценарії, що залежать від прав доступу, перевіряли як від імені власника матеріалу, так і від імені іншого користувача, щоб переконатися в коректному розмежуванні доступу.

Окрему увагу приділено межовим випадкам, що перевіряють стійкість

системи до некоректних або зловмисних дій: спробам виконати операції з контентом без автентифікації, доступу до приватних матеріалів сторонніми користувачами, перевищенню дозволеної частоти запитів і збоям під час завантаження файлів. Такі випадки підтверджують роботу механізмів контролю доступу, обмеження частоти запитів і відкату незавершених операцій. Перелік тест-кейсів та їх результати наведено в таблиці 3.1.

Таблиця 3.1 – Тест-кейси функціонального тестування

№	Сценарій	Очікуваний результат	Статус
1	Реєстрація за поштою й паролем	створено обліковий запис і виконано вхід	Пройдено
2	Вхід через обліковий запис Google	виконано автентифікацію, створено сесію	Пройдено
3	Дія з контентом без автентифікації	операцію відхилено	Пройдено
4	Запис екрана у браузері	отримано відеозапис зі звуком	Пройдено
5	Завантаження відео й обкладинки	файли збережено, створено матеріал	Пройдено
6	Збій під час завантаження	вже збережені об’єкти видалено	Пройдено
7	Автоматичне формування субтитрів	субтитри сформовано та збережено	Пройдено
8	Відео без звукової доріжки	стан зафіксовано без повторного оброблення	Пройдено
9	Перегляд публічного матеріалу за посиланням	матеріал відтворюється	Пройдено
10	Перегляд приватного матеріалу сторонньою особою	доступ відсутній	Пройдено
11	Зміна видимості власником	рівень видимості оновлено	Пройдено
12	Видалення матеріалу власником	матеріал і файли вилучено	Пройдено
13	Перевищення дозволеної частоти запитів	операцію відхилено	Пройдено

За результатами функціонального тестування всі перевірені сценарії, зокрема межові випадки, відпрацювали відповідно до очікувань. Виявлені під час розробки недоліки було усунуто до остаточної фіксації результатів, а повторна перевірка підтвердила коректність роботи відповідних сценаріїв.

3.2 Впровадження та розгортання системи

Впровадження системи полягає в її розгортанні у безсерверному середовищі, сумісному з Next.js. У такому середовищі застосунок виконується як набір безсерверних функцій разом зі статичними ресурсами та сторінками, що рендеряться на сервері, тому немає потреби в постійно запущеному сервері, а обчислювальні ресурси виділяються на вимогу відповідно до навантаження [9]. Це відповідає сформованій вимозі щодо масштабованості без постійного резервування ресурсів.

3.2.1 Розгортання у безсерверному середовищі

Розгортання організовано як послідовність відтворюваних кроків: встановлення залежностей, збирання застосунку з перевіркою типів і лінтингом, застосування міграцій бази даних і публікація в безсерверному середовищі. Оскільки застосунок не має постійно запущеного сервера, оновлення версії не потребує простою — нові запити обслуговуються новими екземплярами безсерверних функцій, а горизонтальне масштабування забезпечується середовищем виконання автоматично, залежно від кількості одночасних запитів.

Збирання проєкту є частиною конвеєра розгортання й виконується перед кожною публікацією, тому виявлення помилок типізації або порушень правил лінтингу зупиняє розгортання до усунення проблеми. Це забезпечує сталу якість коду, що потрапляє в середовище експлуатації.

Загальну схему розгортання системи з основними потоками взаємодії наведено на рисунку 3.2.

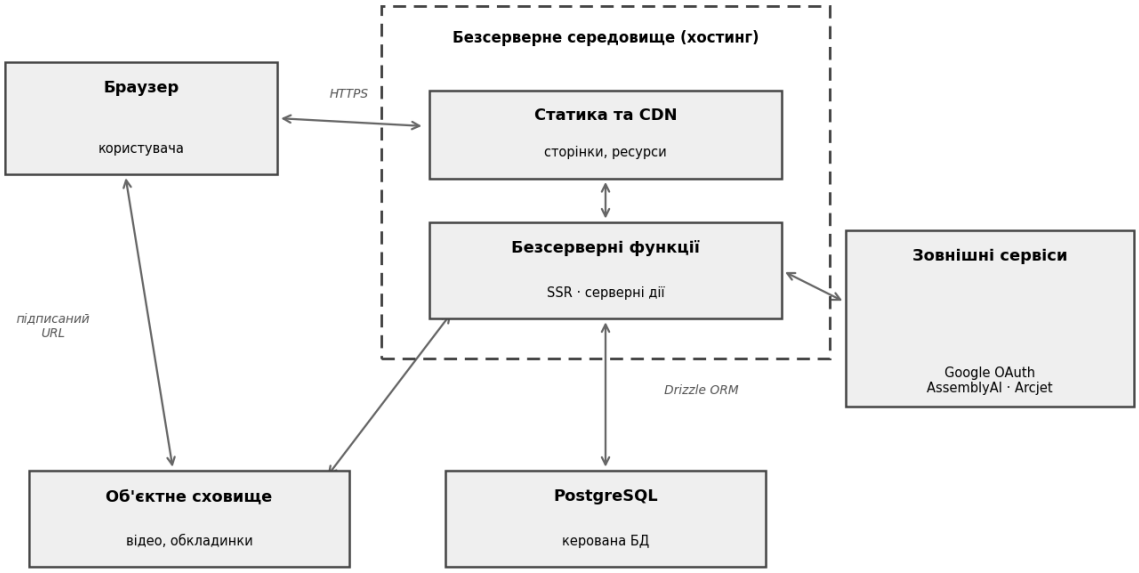

Рисунок 3.2 – Схема розгортання системи

Клієнтський застосунок звертається до хостингу за протоколом HTTPS, отримуючи статичні ресурси та сторінки від мережі доставки вмісту, а динамічні запити — від безсерверних функцій, які виконують рендеринг і серверні дії. Завантаження відеофайлів відбувається безпосередньо з браузера в об’єктне сховище за підписаними URL-адресами, тоді як безсерверні функції звертаються до бази даних через Drizzle ORM, керують об’єктами сховища та взаємодіють із зовнішніми сервісами автентифікації, транскрипції й захисту.

Така топологія розвантажує сервер застосунку: статичні ресурси віддаються з мережі доставки вмісту, а великі файли не проходять через функції. Безсерверні функції відповідають лише за логіку застосунку та доступ до даних.

Маршрути застосунку відображаються на функції безсерверного середовища: статичні сторінки та ресурси віддаються без виклику функцій, а маршрути, що потребують виконання коду на сервері, обслуговуються окремо. Це дає змогу масштабувати навантажені ділянки незалежно та зменшує витрати ресурсів на обслуговування статичного вмісту.

Параметри розгортання відокремлено від коду й винесено у змінні середовища, що дає змогу розгортати систему в різних середовищах без зміни вихідного коду. Перелік основних змінних середовища та їх призначення наведено в таблиці 3.2.

Таблиця 3.2 – Основні змінні середовища

Змінна	Призначення
NEXT_PUBLIC_BASE_URL	базова адреса застосунку
DATABASE_URL	рядок підключення до PostgreSQL
NEXT_PUBLIC_SUPABASE_URL	адреса проєкту Supabase
SUPABASE_SERVICE_ROLE_KEY	службовий ключ сховища (лише на сервері)
BETTER_AUTH_SECRET	секрет для підпису сесій
BETTER_AUTH_URL	базова адреса служби автентифікації
GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET	облікові дані Google OAuth
ARCJET_API_KEY	ключ сервісу захисту Arcjet
ASSEMBLYAI_API_KEY	ключ сервісу транскрипції

Винесення параметрів у змінні середовища дає змогу використовувати різні значення в середовищах розробки та промислової експлуатації без зміни коду, а також відокремити чутливі дані — службовий ключ сховища, секрет автентифікації та ключі зовнішніх сервісів — від вихідного коду. Доступ до таких значень мають лише серверні складники системи.

Процес розгортання охоплює встановлення залежностей, збирання проєкту з перевіркою типів і лінтингом та застосування міграцій бази даних. Структуру бази даних описано декларативно, а її зміни оформлюються міграціями, що генеруються та застосовуються інструментом drizzle-kit. Основні команди процесу розгортання наведено в лістингу 3.1.

Лістинг 3.1 – Основні команди розгортання

```
# встановлення залежностей
npm install
# збирання з перевіркою типів і лінтингом
npm run build
# генерація та застосування міграцій бази даних
npx drizzle-kit generate --name <name>
npx drizzle-kit migrate
```

Об’єктне сховище налаштовано так, що анонімний запис до нього неможливий: завантаження виконуються лише за короткочасними підписаними URL-адресами, а службовий ключ сховища використовується винятково на боці сервера й не передається клієнтові. Завдяки тому, що збирання перевіряє типи та правила лінтингу, у середовище розгортання потрапляє лише код, який пройшов статичний аналіз.

Спостережуваність системи в експлуатації забезпечується централізованою обробкою помилок серверних дій, яка перехоплює та реєструє збої, не перериваючи роботу інтерфейсу некоректним чином. Це спрощує діагностику проблем після розгортання та зменшує ймовірність неконтрольованих відмов.

3.2.2 Супровід системи

Супровід системи спрощується завдяки декларативній схемі бази даних із керованими міграціями, централізованій обробці помилок серверних дій і модульній структурі коду, що локалізує зміни в межах окремих модулів. Завдяки дотриманню принципів чистої архітектури розширення функціональності здебільшого не потребує змін у наявних шарах.

Робочий процес зміни структури бази даних передбачає генерування міграції зі схеми, її перегляд і застосування. Зберігання міграцій разом із кодом дає змогу узгоджувати зміни схеми зі змінами застосунку та відтворювати стан бази даних у новому середовищі.

Безсерверна модель виконання передбачає, що екземпляри функцій створюються за потреби: за відсутності навантаження обчислювальні ресурси не

споживаються, а за його зростання кількість екземплярів збільшується. Це забезпечує економність використання ресурсів, однак потребує врахування затримки першого запуску функції під час проєктування взаємодії.

Відповідальність за збереження даних розподілено між керованою базою даних і об’єктним сховищем, що забезпечують резервне копіювання та реплікацію на рівні відповідних сервісів. Це знижує ризик втрати даних без потреби в окремій інфраструктурі резервування.

Керовані міграції зберігають історію змін структури бази даних, що дає змогу відтворювати схему в будь-якому середовищі та відстежувати її версії. Оновлення залежностей і самого застосунку виконується без зміни даних користувачів.

Описаний процес забезпечує відтворюване розгортання системи в безсерверному середовищі та її подальший супровід без значних витрат на адміністрування інфраструктури.

3.2.3 Контроль версій та зберігання вихідного коду

Для зберігання та впорядкування вихідного коду в проєкті застосовано систему контролю версій Git. Контроль версій є складником управління конфігурацією програмного забезпечення й забезпечує єдину авторитетну копію кодової бази, з якої відтворюється поточний стан проєкту [8]. Локальний репозиторій доповнено віддаленим репозиторієм на платформі GitHub, що виконує роль резервного сховища коду.

Розміщення коду у віддаленому репозиторії захищає проєкт від втрати внаслідок збою локального обладнання та забезпечує доступ до актуальної версії коду з різних робочих місць. Відновлення робочого середовища на іншому комп’ютері виконується повним клонуванням репозиторію, без ручного перенесення файлів.

Склад репозиторію визначає файл .gitignore, який вилучає з-під контролю версій дані, що не належать до вихідного коду. Зокрема, з репозиторію виключено каталог залежностей node_modules і каталог збирання .next, які відтворюються автоматично, а також файл змінних середовища .env зі службовими ключами —

ключем сервісної ролі сховища, ключами зовнішніх сервісів і секретами автентифікації. Це запобігає потраплянню конфіденційних даних у спільне сховище та зменшує розмір репозиторію.

Завдяки фіксації у репозиторії вихідного коду разом із файлом маніфесту залежностей package.json репозиторій є самодостатнім знімком проєкту: після клонування й встановлення залежностей застосунок відтворюється в робочому стані. Такий репозиторій слугує типовою вхідною точкою для розгортання в безсерверному середовищі, де платформа збирає та публікує застосунок безпосередньо з його вмісту.

Отже, контроль версій використано насамперед для надійного зберігання та впорядкування кодової бази, захисту конфіденційних даних і забезпечення відтворюваності проєкту, що створює основу для його подальшого супроводу та розвитку.

3.3 Аналіз результатів та оцінка відповідності вимогам

Оцінювання результатів полягає в зіставленні реалізованих можливостей системи зі сформованими в розділі 1 вимогами. Розглянуто окремо покриття функціональних вимог, що визначають дії системи, та нефункціональних вимог, що визначають її якісні характеристики.

3.3.1 Відповідність функціональних вимог

Усі визначені функціональні вимоги реалізовано відповідними модулями системи. Відповідність кожної функціональної вимоги конкретному засобу реалізації наведено в таблиці 3.3.

Таблиця 3.3 – Відповідність функціональних вимог реалізації

Вимога	Засіб реалізації	Статус
Реєстрація та автентифікація	Better Auth (пошта/пароль, Google)	Виконано
Запис екрана у браузері	хук useScreenRecording	Виконано

Вимога	Засіб реалізації	Статус
Завантаження за підписаними URL	createUploadUrls, Supabase Storage	Виконано
Збереження метаданих контенту	saveVideoDetails, таблиця videos	Виконано
Автоматичні субтитри	getTranscript, AssemblyAI	Виконано
Перегляд переліку з пошуком і сортуванням	getAllVideos	Виконано
Перегляд матеріалу з обліком переглядів	getVideoById, incrementVideoViews	Виконано
Керування видимістю	updateVideoVisibility	Виконано
Поширення посиланням	сторінка перегляду, копіювання посилання	Виконано
Видалення матеріалу з файлами	deleteVideo, removeStorageObjects	Виконано
Розмежування прав доступу	getSessionUserId, assertVideoOwner	Виконано

Реалізована функціональність охоплює повний життєвий цикл відеоматеріалу й відтворює ключові можливості розглянутих у підрозділі 1.2 аналогів, водночас усуваючи їхнє основне обмеження: систему можна самостійно розгорнути й контролювати місце зберігання контенту.

3.3.2 Відповідність нефункціональних вимог

Нефункціональні вимоги також задоволено відповідними проєктними та реалізаційними рішеннями. Відповідність нефункціональних вимог наведено в таблиці 3.4.

Таблиця 3.4 – Відповідність нефункціональних вимог реалізації

Вимога	Реалізація
Масштабованість	безсерверне виконання обчислень на вимогу
Продуктивність завантаження	пряме завантаження у сховище за підписаними URL-адресами
Безпека	Arcjet, контроль доступу, серверне зберігання службового ключа
Надійність і цілісність даних	обробка помилок, відкат завантаження, каскадне видалення
Зручність використання	простий інтерфейс і локалізація (українська та англійська)

Вимога	Реалізація
Підтримуваність і розширюваність	чиста архітектура, SOLID, ізоляція доступу до даних
Переносність розгортання	конфігурування через змінні середовища

Найважливіші нефункціональні характеристики підтверджуються архітектурними рішеннями. Масштабованість досягається безсерверним виконанням, за якого обчислювальні ресурси виділяються пропорційно навантаженню; продуктивність завантаження великих файлів забезпечується їх прямою передачею у сховище, що не навантажує сервер застосунку; безпеку посилено поєднанням захисту звернень, контролю доступу на рівні серверних дій і серверного зберігання службових ключів. Підтримуваність системи зумовлена ізоляцією доступу до даних та дотриманням принципів чистої архітектури і SOLID.

Виконання нефункціональних вимог підтверджується не лише архітектурними рішеннями, а й результатами функціонального тестування: коректність контролю доступу та обмеження частоти запитів перевірено відповідними межовими тест-кейсами, а відкат незавершеного завантаження — сценарієм збою.

Серед обмежень поточної реалізації — застосування переважно ручного функціонального тестування без автоматизованого набору тестів і зберігання медіафайлів у межах одного сховища. Напрямами подальшого вдосконалення є запровадження автоматизованих модульних і наскрізних тестів, розширення аналітики переглядів та підтримка додаткових постачальників автентифікації. Зазначені обмеження не перешкоджають виконанню визначених вимог.

Зіставлення реалізації з вимогами показує, що розроблена система задовольняє всі визначені функціональні та нефункціональні вимоги. Це підтверджує, що мету роботи — підвищення масштабованості та керованості платформи менеджменту й дистрибуції відеоконтенту шляхом її проєктування та реалізації на основі Next.js і безсерверних технологій — досягнуто.

3.4 Висновки до розділу 3

У розділі виконано тестування програмної системи на рівнях статичного аналізу коду та функціональної перевірки ключових сценаріїв, зокрема межових випадків. Описано впровадження системи в безсерверному середовищі з відокремленням параметрів розгортання у змінні середовища, застосуванням міграцій бази даних і безпечним налаштуванням сховища, а також розглянуто питання супроводу системи.

Проаналізовано відповідність реалізації сформованим вимогам: підтверджено покриття всіх функціональних і нефункціональних вимог. Отримані результати свідчать про працездатність системи та досягнення мети роботи. Питання безпеки життєдіяльності й основ охорони праці розглянуто в розділі 4.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

Розробка та експлуатація платформи відбуваються переважно за персональним комп’ютером, що зумовлює тривалу розумову працю в умовах статичного навантаження, зорового та нервово-емоційного напруження. Тому поряд із технічними питаннями важливо враховувати безпеку життєдіяльності розробника та вимоги охорони праці до його робочого місця.

У розділі розглянуто життєдіяльність людини у системі «людина – машина – середовище існування» стосовно діяльності розробника програмного забезпечення, а також гігієнічні вимоги до організації та обладнання робочих місць з відеодисплейними терміналами. Виклад спирається на чинні нормативні документи України з охорони праці.

4.1 Характеристика життєдіяльності людини у системі «людина – машина – середовище існування»

Безпека життєдіяльності розглядає діяльність людини як функціонування складної системи «людина – машина – середовище існування», у якій взаємодіють три компоненти: людина як суб’єкт праці, технічні засоби (машина), якими вона користується, та середовище, у якому ця взаємодія відбувається [17]. Стан і властивості кожного компонента, а також характер їх взаємодії визначають як результативність діяльності, так і рівень її безпеки.

Стосовно розробки програмного забезпечення роль людини виконує розробник, діяльність якого має переважно розумовий характер і пов’язана з опрацюванням інформації, прийняттям рішень та тривалою взаємодією з обчислювальною технікою. Така праця супроводжується статичним навантаженням на опорно-руховий апарат, напруженням зору та нервово-емоційним навантаженням, що визначає підвищені вимоги до решти компонентів системи.

Роль машини в цій системі відіграють персональний комп’ютер, відеодисплейний термінал і програмні засоби, з якими працює розробник. Від характеристик машини — якості зображення на екрані, зручності засобів введення, організації взаємодії з програмним забезпеченням — залежить навантаження на зоровий аналізатор і кисті рук, а отже стомлюваність та ризик порушень здоров’я. Вимоги до екранних пристроїв як складника системи встановлено окремим нормативним документом [14].

Середовищем існування в межах системи є виробниче приміщення з його параметрами: мікрокліматом, освітленням, рівнем шуму та складом повітря. Відхилення цих параметрів від нормованих значень погіршує самопочуття та працездатність людини навіть за справної машини, тому середовище розглядають як рівноправний компонент системи, що впливає на безпеку діяльності.

Працездатність людини-оператора в такій системі не є сталою: упродовж робочого дня вона змінюється, а тривала одноманітна діяльність призводить до втоми та зниження уваги. Основними небезпечними та шкідливими чинниками, що діють на розробника у системі «людина – машина – середовище існування», є:

1) тривале статичне навантаження, зумовлене вимушеною робочою позою;

2) напруження зору під час роботи з екраном;

3) нервово-емоційне навантаження та монотонність праці;

4) несприятливі параметри мікроклімату й освітлення робочої зони;

5) ризик ураження електричним струмом і пожежна небезпека, пов’язані з використанням електрообладнання.

Метою врахування безпеки життєдіяльності є узгодження компонентів системи так, щоб мінімізувати дію шкідливих чинників: машина має відповідати ергономічним вимогам, середовище — нормам виробничої санітарії, а організація праці — раціональному режиму роботи та відпочинку. Конкретні гігієнічні вимоги до робочого місця, що реалізують цей підхід, розглянуто в підрозділі 4.2.

4.2 Гігієнічні вимоги до організації та обладнання робочих місць з ВДТ

Організація робочого місця з відеодисплейним терміналом (ВДТ) має забезпечувати безпечні та комфортні умови праці й відповідати вимогам законодавства про охорону праці [13]. Для діяльності розробника програмного забезпечення, що належить до робіт легкої важкості (категорія Іб) за фізичним навантаженням, визначальними є вимоги до планування робочого місця, мікроклімату, освітлення та електробезпеки.

Вимоги до організації та обладнання робочих місць з екранними пристроями встановлено відповідним нормативним документом [14]. Робоче місце має бути спроєктоване так, щоб працівник мав достатньо простору для зміни робочого положення, а екран, клавіатуру та засоби введення розташовано з урахуванням зручної робочої пози. Екран розміщують на відстані 50–70 см від очей користувача, при цьому верхній край екрана має бути на рівні очей або трохи нижче, що зменшує напруження зору та шиї. Поверхня екрана не повинна давати відблисків, а клавіатуру й маніпулятор розміщують так, щоб забезпечити природне положення кистей рук.

Параметри мікроклімату у приміщеннях з ВДТ повинні відповідати санітарним нормам [15]. Для робіт категорії Іб встановлено оптимальні значення температури, відносної вологості та швидкості руху повітря, що створюють комфортне теплове відчуття й не спричиняють перенапруження систем терморегуляції. Нормовані значення основних параметрів виробничого середовища наведено в таблиці 4.1.

Таблиця 4.1 – Нормовані параметри виробничого середовища для робіт категорії Іб

Параметр	Період року	Нормоване значення
Температура повітря	холодний	21–23 °C
Температура повітря	теплий	22–24 °C
Відносна вологість повітря	—	40–60 %
Швидкість руху повітря	холодний	0,1 м/с

Параметр	Період року	Нормоване значення
Швидкість руху повітря	теплий	0,1–0,2 м/с
Освітленість робочої поверхні	—	300–500 лк

Освітлення робочого місця має забезпечувати достатній рівень освітленості без засліплення та значних перепадів яскравості в полі зору. Природне й штучне освітлення приміщень нормується відповідними будівельними нормами [16]; для робіт з ВДТ рекомендована освітленість робочої поверхні становить 300–500 лк. Робоче місце доцільно розташовувати так, щоб уникати прямого потрапляння сонячного світла на екран і відблисків від світильників.

Для запобігання перевтомі під час тривалої роботи з ВДТ передбачають раціональний режим праці та відпочинку з регламентованими перервами. Перерви дають змогу зменшити статичне й зорове навантаження; під час них доцільно виконувати вправи для очей та м’язів спини й шиї. Тривалі безперервні сесії роботи за комп’ютером розбивають на етапи з короткими перервами.

Робоче місце з ВДТ повинно відповідати вимогам електро- та пожежної безпеки. Електроживлення здійснюють через справні розетки із захисним заземленням; забороняється використання пошкоджених кабелів живлення та подовжувачів. Застосовувана обчислювальна техніка має бути справною та сертифікованою, а приміщення — оснащеним засобами первинного пожежогасіння й мати вільні шляхи евакуації.

Дотримання наведених гігієнічних вимог до організації та обладнання робочого місця з ВДТ забезпечує безпечні й комфортні умови праці розробника та знижує ризик порушень здоров’я, пов’язаних із тривалою роботою за комп’ютером.

4.3 Висновки до розділу 4

У розділі розглянуто життєдіяльність розробника програмного забезпечення у системі «людина – машина – середовище існування», у якій взаємодіють людина-оператор, обчислювальна техніка з програмними засобами та виробниче середовище. Визначено основні небезпечні та шкідливі чинники, що діють на

розробника під час тривалої роботи за комп’ютером, і показано, що безпека діяльності залежить від узгодження всіх компонентів системи.

Сформульовано гігієнічні вимоги до організації та обладнання робочих місць з відеодисплейними терміналами: до планування робочого місця й розташування екрана, параметрів мікроклімату та освітлення, режиму праці й відпочинку, а також електро- та пожежної безпеки. Дотримання цих вимог, установлених чинними нормативними документами, забезпечує безпечні умови праці й збереження здоров’я працівника.

Для робочого місця розробника визначено нормовані значення параметрів виробничого середовища — оптимальні показники температури, відносної вологості та швидкості руху повітря для відповідної категорії робіт, а також рівні освітленості робочої поверхні. Дотримання цих значень у поєднанні з раціональним режимом праці й відпочинку зменшує зорове та статичне м’язове навантаження під час тривалої роботи за відеодисплейним терміналом і знижує ризик передчасної втоми.

Окремо розглянуто вимоги електро- та пожежної безпеки під час експлуатації обчислювальної техніки, дотримання яких запобігає ураженню електричним струмом і виникненню займання. Загалом виконання сформульованих вимог безпеки життєдіяльності та охорони праці зменшує вплив небезпечних і шкідливих чинників, сприяє збереженню здоров’я та працездатності розробника й створює умови для безпечного та продуктивного виконання професійних завдань.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано задачу проєктування та програмної реалізації веборієнтованої платформи менеджменту й дистрибуції цифрового відеоконтенту на основі архітектури Next.js і безсерверних технологій. Поставлену мету — підвищення масштабованості та керованості такої платформи із дотриманням принципів чистої архітектури та SOLID — досягнуто, а всі визначені завдання виконано.

Основні результати роботи такі:

1) проаналізовано предметну область менеджменту й дистрибуції відеоконтенту, оглянуто наявні рішення та сформовано функціональні й нефункціональні вимоги до системи, визначено акторів і варіанти використання;

2) обґрунтовано технологічний стек і спроєктовано багатошарову архітектуру з розмежуванням шарів інтерфейсу, серверних дій та доступу до даних, що дало змогу ізолювати бізнес-логіку й уникнути антипатерну «God Object»;

3) спроєктовано модель даних і схему реляційної бази даних для метаданих контенту та облікових записів користувачів;

4) реалізовано ключові модулі системи: автентифікацію, запис і завантаження контенту через підписані URL-адреси, серверні дії та автоматичну транскрипцію;

5) реалізовано механізми безпеки й контролю доступу до серверного середовища;

6) виконано тестування системи на рівнях статичного аналізу та функціональної перевірки й підтверджено відповідність реалізації всім сформованим функціональним і нефункціональним вимогам.

Новизна роботи полягає в обґрунтуванні архітектурного підходу, що поєднує серверний рендеринг Next.js, серверні дії як єдину контрольовану точку доступу до даних і безсерверне зберігання медіафайлів через підписані URL-адреси з послідовним застосуванням принципів чистої архітектури для ізоляції бізнес-логіки.

Практичне значення роботи полягає у створенні готового до розгортання програмного продукту для запису, зберігання, керування та дистрибуції відеоконтенту в умовах нерівномірного навантаження без постійно зарезервованої серверної інфраструктури.

Достовірність одержаних результатів підтверджено статичним аналізом коду — перевіркою типів і лінтингом у складі збирання — та функціональним тестуванням ключових сценаріїв, які відпрацювали відповідно до очікувань.

Запропонована архітектура придатна для подальшого розширення функціональності: запровадження автоматизованих модульних і наскрізних тестів, розширення аналітики переглядів та підтримки додаткових постачальників автентифікації.

У роботі також розглянуто питання безпеки життєдіяльності та основ охорони праці стосовно робочого місця розробника. Таким чином, у кваліфікаційній роботі досягнуто поставленої мети, виконано всі основні завдання та отримано практично значущі результати, придатні для впровадження й подальшого розвитку платформи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sandvine. The Global Internet Phenomena Report. Sandvine, 2023. URL: https://www.sandvine.com/phenomena (дата звернення: 14.06.2026).

2. Петрик М. Р., Мудрик І. Я., Стоянов Ю. М. Методичні вказівки до лабораторних робіт з дисципліни «Архітектура та проєктування програмного забезпечення» для здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Інженерія програмного забезпечення». Тернопіль : ТНТУ ім. І. Пулюя, 2026. 54 с.

3. Остапчук О., Цуприк Г. Технічні особливості взаємодії між клієнтом та сервером у реальному часі. Інформаційні моделі, системи та технології : матеріали X наук.-техн. конф. ТНТУ ім. І. Пулюя (7–8 грудня 2022 р.). Тернопіль : ТНТУ, 2022. С. 72.

4. Loom. Screen recorder & async video. URL: https://www.loom.com (дата звернення: 14.06.2026).

5. Vimeo. Video Privacy. URL: https://vimeo.com/features/video-privacy (дата звернення: 14.06.2026).

6. YouTube Help. Change video privacy settings. URL: https://support.google.com/youtube/answer/157177 (дата звернення: 14.06.2026).

7. Screencastify. Screen recorder for Google Chrome. URL: https://www.screencastify.com (дата звернення: 14.06.2026).

8. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p.

9. Vercel Inc. Next.js Documentation. URL: https://nextjs.org/docs (дата звернення: 14.06.2026).

10. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O’Reilly Media, 2020. 400 p.

11. Li Z., Guo L., Cheng J., Chen Q., He B., Guo M. The Serverless Computing Survey: A Technical Primer for Design Architecture. ACM Computing Surveys. 2022. Vol. 54, No. 10s. Article 220. 34 p. DOI: 10.1145/3508360.

12. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed.

New York : McGraw-Hill, 2020. 1376 p.

13. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 17.06.2026).

14. Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями : наказ Міністерства соціальної політики України від 14.02.2018 № 207 (НПАОП 0.00-7.15-18). URL: https://zakon.rada.gov.ua/laws/show/z0508-18 (дата звернення: 17.06.2026).

15. Санітарні норми мікроклімату виробничих приміщень : ДСН 3.3.6.042-99. URL: https://zakon.rada.gov.ua/rada/show/va042282-99 (дата звернення: 17.06.2026).

16. Сєріков Я. О., Коженевські Л. Ф., Хворост М. В. Безпека життєдіяльності та охорона праці : підручник : у 2 ч. Ч. 1 : Безпека життєдіяльності. Харків : ХНУМГ ім. О. М. Бекетова ; Краків : ЄАС, 2021. 255 с.

17. Грибан В. Г., Фоменко А. Є., Казначеєв Д. Г. Безпека життєдіяльності та охорона праці : підручник. Дніпро : Дніпроп. держ. ун-т внутр. справ, 2022. 388 с.

18. Martin R. C. Clean Craftsmanship: Disciplines, Standards, and Ethics. Boston : Addison-Wesley Professional, 2021. 416 p.

19. React. Офіційна документація бібліотеки. Meta Open Source. URL: https://react.dev (дата звернення: 18.04.2026).

20. TypeScript Documentation: The TypeScript Handbook. Microsoft. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 18.04.2026).

21. PostgreSQL 16 Documentation. The PostgreSQL Global Development Group, 2024. URL: https://www.postgresql.org/docs (дата звернення: 18.04.2026).

ДОДАТКИ

ДОДАТОК А

Лістинги коду ключових модулів

У додатку наведено лістинги вихідного коду ключових модулів програмної системи: опису структури бази даних, конфігурації автентифікації та захисту, проміжного програмного забезпечення й серверних дій доступу до даних.

Лістинг А.1 – Опис структури бази даних (drizzle/schema.ts)

```
import {
  pgTable,
  text,
  timestamp,
  boolean,
  uuid,
  integer,
} from "drizzle-orm/pg-core";

export const user = pgTable("user", {
  id: text("id").primaryKey(),
  name: text("name").notNull(),
  email: text("email").notNull().unique(),
  emailVerified: boolean("email_verified").notNull(),
  image: text("image"),
  createdAt: timestamp("created_at").notNull(),
  updatedAt: timestamp("updated_at").notNull(),
});

export const session = pgTable("session", {
  id: text("id").primaryKey(),
  expiresAt: timestamp("expires_at").notNull(),
  token: text("token").notNull().unique(),
  createdAt: timestamp("created_at").notNull(),
  updatedAt: timestamp("updated_at").notNull(),
  ipAddress: text("ip_address"),
  userAgent: text("user_agent"),
  userId: text("user_id")
    .notNull()
    .references(() => user.id, { onDelete: "cascade" }),
});

export const account = pgTable("account", {
  id: text("id").primaryKey(),
  accountId: text("account_id").notNull(),
  providerId: text("provider_id").notNull(),
  userId: text("user_id")
    .notNull()
    .references(() => user.id, { onDelete: "cascade" }),
  accessToken: text("access_token"),
  refreshToken: text("refresh_token"),
  idToken: text("id_token"),
  accessTokenExpiresAt: timestamp("access_token_expires_at"),
  refreshTokenExpiresAt: timestamp("refresh_token_expires_at"),
  scope: text("scope"),
  password: text("password"),
```

```
  createdAt: timestamp("created_at").notNull(),
  updatedAt: timestamp("updated_at").notNull(),
});

export const verification = pgTable("verification", {
  id: text("id").primaryKey(),
  identifier: text("identifier").notNull(),
  value: text("value").notNull(),
  expiresAt: timestamp("expires_at").notNull(),
  createdAt: timestamp("created_at"),
  updatedAt: timestamp("updated_at"),
});

export const videos = pgTable("videos", {
  id: uuid("id").primaryKey().defaultRandom().unique(),
  title: text("title").notNull(),
  description: text("description").notNull(),
  videoUrl: text("video_url").notNull(),
  videoId: text("video_id").notNull(),
  thumbnailUrl: text("thumbnail_url").notNull(),
  visibility: text("visibility").$type<"public" | "private">().notNull(),
  tags: text("tags").array().notNull().default([]),
  transcript: text("transcript"),
  userId: text("user_id")
    .notNull()
    .references(() => user.id, { onDelete: "cascade" }),
  views: integer("views").notNull().default(0),
  duration: integer("duration"),
  createdAt: timestamp("created_at").notNull().defaultNow(),
  updatedAt: timestamp("updated_at").notNull().defaultNow(),
});

export const schema = {
  user,
  session,
  account,
  verification,
};
```

Лістинг А.2 – Конфігурація автентифікації (lib/auth.ts)

```
import { db } from "@/drizzle/db";
import { schema } from "@/drizzle/schema";
import { betterAuth } from "better-auth";
import { drizzleAdapter } from "better-auth/adapters/drizzle";

export const auth = betterAuth({
  database: drizzleAdapter(db, {
    provider: "pg",
    schema,
  }),

  emailAndPassword: {
    enabled: true,
  },

  socialProviders: {
    google: {
      clientId: process.env.GOOGLE_CLIENT_ID!,
      clientSecret: process.env.GOOGLE_CLIENT_SECRET!,
```

```
    },
  },

  baseURL: process.env.NEXT_PUBLIC_BASE_URL,
});
```

Лістинг А.3 – Конфігурація захисту Arcjet (lib/arcjet.ts)

```
import arcjet, {
  detectBot,
  fixedWindow,
  shield,
  request,
  validateEmail,
  slidingWindow,
  ArcjetDecision,
  createMiddleware,
} from "@arcjet/next";
import { getEnv } from "./utils";

export {
  detectBot,
  fixedWindow,
  shield,
  request,
  slidingWindow,
  validateEmail,
  createMiddleware,
  ArcjetDecision,
};

const aj = arcjet({
  key: getEnv("ARCJET_API_KEY"),
  rules: [],
});

export default aj;
```

Лістинг А.4 – Проміжне програмне забезпечення (middleware.ts)

```
import { NextRequest, NextResponse } from "next/server";
import { getSessionCookie } from "better-auth/cookies";
import aj, { createMiddleware, detectBot, shield } from "./lib/arcjet";

// Arcjet protection: shield + bot detection, applied to every matched request.
const validate = aj
    .withRule(
        shield({
            mode: "LIVE",
        })
    )
    .withRule(
        detectBot({
            mode: "LIVE",
            allow: ["CATEGORY:SEARCH_ENGINE", "G00G1E_CRAWLER"],
        })
    );
```

```
// Auth gate: redirect to /sign-in when the Better Auth session cookie is absent.
// Optimistic, cookie-only check (no network call) so it runs safely in the Edge
// runtime. Full session validation still happens in server actions/pages via
// auth.api.getSession().
const authGate = async (request: NextRequest) => {
    const sessionCookie = getSessionCookie(request);

    if (!sessionCookie) {
        return NextResponse.redirect(new URL("/sign-in", request.url));
    }

    return NextResponse.next();
};

// Run Arcjet first; if the request is allowed, fall through to the auth gate.
export default createMiddleware(validate, authGate);

export const config = {
    matcher: ["/((?!api|_next/static|_next/image|favicon.ico|sign-in|assets).*)"],
};
```

Лістинг А.5 – Серверні дії доступу до даних (фрагмент модуля lib/actions/video.ts)

```
"use server";

import {db} from "@/drizzle/db";
import {videos, user} from "@/drizzle/schema";
import {headers} from "next/headers";
import {revalidatePath} from "next/cache";
import {and, desc, eq, or, sql} from "drizzle-orm";
import {auth} from "@/lib/auth";
import {getOrderByClause, matchesSearch, withErrorHandling} from "@/lib/utils";
import aj, {fixedWindow, request} from "../arcjet";
import {supabaseAdmin} from "@/lib/supabaseAdmin";
import {transcribeToVtt} from "@/lib/transcription";

const BUCKET = "videos";
const cleanFileName = (name: string) => name.replace(/[^a-zA-Z0-9.]/g, "_");

const validateWithArcjet = async (fingerPrint: string) => {
    const rateLimit = aj.withRule(
        fixedWindow({
            mode: "LIVE",
            window: "1m",
            max: 2,
            characteristics: ["fingerprint"],
        })
    );
    const req = await request();
    const decision = await rateLimit.protect(req, {fingerprint: fingerPrint});
    if (decision.isDenied()) {
        throw new Error("Rate Limit Exceeded");
    }
};

const revalidatePaths = (paths: string[]) => {
```

```
    paths.forEach((path) => revalidatePath(path));
};

const getSessionUserId = async (): Promise<string> => {
    const session = await auth.api.getSession({headers: await headers()});
    if (!session) throw new Error("Unauthenticated");
    return session.user.id;
};

// Loads a video and verifies the current session user owns it.
const assertVideoOwner = async (videoId: string) => {
    const userId = await getSessionUserId();
    const [video] = await db
        .select()
        .from(videos)
        .where(eq(videos.videoId, videoId));

    if (!video) throw new Error("Video not found");
    if (video.userId !== userId) {
        throw new Error("You do not have permission to modify this video");
    }

    return { video, userId };
};

const buildVideoWithUserQuery = () =>
    db
        .select({
            video: videos,
            user: {id: user.id, name: user.name, image: user.image},
        })
        .from(videos)
        .leftJoin(user, eq(videos.userId, user.id));

export const saveVideoDetails = withErrorHandling(
    async (videoDetails: VideoDetails) => {
        const userId = await getSessionUserId();
        await validateWithArcjet(userId);

        const now = new Date();
        await db.insert(videos).values({
            ...videoDetails,
            userId,
            createdAt: now,
            updatedAt: now,
        });

        revalidatePaths(["/"]);
        return {videoId: videoDetails.videoId};
    }
);

// Auth-gated: mints short-lived signed upload URLs so the browser can upload
// directly to Storage without the bucket needing public (anon) write access.
export const createUploadUrls = withErrorHandling(
    async (videoFileName: string, thumbnailFileName: string) => {
        await getSessionUserId();

        const videoId = crypto.randomUUID();
        const videoPath = `${videoId}-${cleanFileName(videoFileName)}`;
        const thumbnailPath = `thumb-${videoId}-
${cleanFileName(thumbnailFileName)}`;
```

```
        const [videoSigned, thumbnailSigned] = await Promise.all([
            supabaseAdmin.storage.from(BUCKET).createSignedUploadUrl(videoPath),
            supabaseAdmin.storage.from(BUCKET).createSignedUploadUrl(thumbnailPath
),
        ]);

        if (videoSigned.error || !videoSigned.data) {
            throw new Error("Failed to create video upload URL");
        }
        if (thumbnailSigned.error || !thumbnailSigned.data) {
            throw new Error("Failed to create thumbnail upload URL");
        }

        return {
            videoId,
            video: { path: videoPath, token: videoSigned.data.token },
            thumbnail: { path: thumbnailPath, token: thumbnailSigned.data.token },
        };
    }
);

// Server-side cleanup of orphaned Storage objects (uses the elevated client).
export const removeStorageObjects = withErrorHandling(async (paths: string[]) => {
    if (paths.length > 0) {
        await supabaseAdmin.storage.from(BUCKET).remove(paths);
    }
    return {};
});

export const updateVideoVisibility = withErrorHandling(
    async (videoId: string, visibility: Visibility) => {
        const { userId } = await assertVideoOwner(videoId);
        await validateWithArcjet(userId);

        await db
            .update(videos)
            .set({ visibility, updatedAt: new Date() })
            .where(eq(videos.videoId, videoId));

        revalidatePaths(["/", `/video/${videoId}`]);
        return {};
    }
);

export const incrementVideoViews = withErrorHandling(
    async (videoId: string) => {
        await db
            .update(videos)
            .set({ views: sql`${videos.views} + 1`, updatedAt: new Date() })
            .where(eq(videos.videoId, videoId));

        revalidatePaths([`/video/${videoId}`]);
        return {};
    }
);
```

ДОДАТОК Б

Схема бази даних мовою SQL

У додатку наведено опис структури бази даних мовою SQL, отриманий генерацією міграцій засобом Drizzle Kit. Наведено оператори створення таблиць і визначення зовнішніх ключів.

Лістинг Б.1 – SQL-опис структури бази даних

```
CREATE TABLE "user" (
    "id" text PRIMARY KEY NOT NULL,
    "name" text NOT NULL,
    "email" text NOT NULL,
    "email_verified" boolean NOT NULL,
    "image" text,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    CONSTRAINT "user_email_unique" UNIQUE("email")
);

CREATE TABLE "session" (
    "id" text PRIMARY KEY NOT NULL,
    "expires_at" timestamp NOT NULL,
    "token" text NOT NULL,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL,
    "ip_address" text,
    "user_agent" text,
    "user_id" text NOT NULL,
    CONSTRAINT "session_token_unique" UNIQUE("token")
);

CREATE TABLE "account" (
    "id" text PRIMARY KEY NOT NULL,
    "account_id" text NOT NULL,
    "provider_id" text NOT NULL,
    "user_id" text NOT NULL,
    "access_token" text,
    "refresh_token" text,
    "id_token" text,
    "access_token_expires_at" timestamp,
    "refresh_token_expires_at" timestamp,
    "scope" text,
    "password" text,
    "created_at" timestamp NOT NULL,
    "updated_at" timestamp NOT NULL
);

CREATE TABLE "verification" (
    "id" text PRIMARY KEY NOT NULL,
    "identifier" text NOT NULL,
    "value" text NOT NULL,
    "expires_at" timestamp NOT NULL,
    "created_at" timestamp,
    "updated_at" timestamp
```

```
);

CREATE TABLE "videos" (
    "id" uuid PRIMARY KEY DEFAULT gen_random_uuid() NOT NULL,
    "title" text NOT NULL,
    "description" text NOT NULL,
    "video_url" text NOT NULL,
    "video_id" text NOT NULL,
    "thumbnail_url" text NOT NULL,
    "visibility" text NOT NULL,
    "tags" text[] DEFAULT '{}' NOT NULL,
    "transcript" text,
    "user_id" text NOT NULL,
    "views" integer DEFAULT 0 NOT NULL,
    "duration" integer,
    "created_at" timestamp DEFAULT now() NOT NULL,
    "updated_at" timestamp DEFAULT now() NOT NULL,
    CONSTRAINT "videos_id_unique" UNIQUE("id")
);

ALTER TABLE "account" ADD CONSTRAINT "account_user_id_user_id_fk"
    FOREIGN KEY ("user_id") REFERENCES "public"."user"("id")
    ON DELETE cascade ON UPDATE no action;

ALTER TABLE "session" ADD CONSTRAINT "session_user_id_user_id_fk"
    FOREIGN KEY ("user_id") REFERENCES "public"."user"("id")
    ON DELETE cascade ON UPDATE no action;

ALTER TABLE "videos" ADD CONSTRAINT "videos_user_id_user_id_fk"
    FOREIGN KEY ("user_id") REFERENCES "public"."user"("id")
    ON DELETE cascade ON UPDATE no action;
```

ДОДАТОК В

Посилання на репозиторій GitHub

https://github.com/Roman1375/Clip-Flip