МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Львівський національний університет імені Івана Франка
Факультет електроніки та комп'ютерних технологій
Кафедра системного проектування

Допустити до захисту

Завідувач кафедри
_____проф. Шувар Р. Я.
«___» _____2023 р.

Кваліфікаційна робота

Бакалавр
(освітній ступінь)

Розробка платформи для хостингу та перегляду відеоматеріалів з можливістю передачі медіаконтенту у режимі реального часу з використанням стеку технологій та засобів Node.js, Nest.js, Amazon Web Services, AWS Cognito, AWS S3, AWS lambda, AWS IVS

Виконав:
Студент IV курсу групи ФЕП – 41
спеціальності 121 – Інженерія
програмного забезпечення

_____ Берніш М.І.

Науковий керівник:
_____ доц. Шувар Р.Я.
«____» _____2023 р.

Рецензент:
_____ доц. Бойко І.М.

Львів 2023

АНОТАЦІЯ

Метою дипломної роботи є розробка платформи для хостингу та перегляду відеоматеріалів з використанням стеку технологій та засобів Node.js, Nest.js, Amazon Web Services (AWS), AWS S3, AWS IVS. Досліджуються технології та інструменти, необхідні для реалізації такої платформи, зокрема Node.js та Nest.js для серверної розробки, а також Amazon Web Services для надання необхідних інфраструктурних можливостей. AWS S3 використовується для зберігання відеофайлів, а AWS IVS забезпечує потокове відтворення відеоконтенту. Результатом дипломної роботи буде розроблена функціональна платформа, що дозволить користувачам зручно хостити, завантажувати та переглядати відеоматеріали.

ABSTRACT

The objective of this diploma project is to develop a platform for hosting and streaming video content using the stack of technologies and tools such as Node.js, Nest.js, Amazon Web Services (AWS), AWS S3, and AWS IVS. The research focuses on exploring the necessary technologies and tools required for implementing such a platform, including Node.js and Nest.js for server-side development, as well as Amazon Web Services for providing the necessary infrastructure capabilities. AWS S3 is utilized for storing video files, and AWS IVS enables video content streaming. The outcome of this project will be a functional platform that allows users to easily host, upload, and view video content.

Зміст

АНОТАЦІЯ 2

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ 5

ВСТУП 6

РОЗДІЛ 1 АНАЛІЗ СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ 8

1.1 Зростання популярності відеоконтенту 8

1.2 Недоліки існуючих платформ для хостингу та перегляду відео 8

1.3 Обмежена масштабованість і надійність платформ 9

1.4 Потреба у передачі медіаконтенту у режимі реального часу 9

1.5 Використання стеку технологій та засобів Node.js, Nest.js, Amazon Web Services 9

1.6 Потреба в ефективному управлінні великим обсягом відеоконтенту 10

1.7 Важливість безпеки та конфіденційності відеоданих 11

1.8 Розподілені системи та вимоги до масштабованості 12

1.9 Аналіз сучасних відеохостингів і їхніх обмежень 12

1.10 Передові можливості стрімінгу відео 13

1.11 Роль соціальних медіа у стрімінгу відео 13

1.12 Вплив мобільних технологій на стрімінг відео 14

1.13 Вплив стрімінгу відео на культуру та суспільство 14

1.14 Виклики забезпечення безпеки та конфіденційності відеоконтенту 15

1.15 Повна локалізація на українську мову 16

РОЗДІЛ 2 ВИКОРИСТАНІ ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ 18

2.1 Вступ до платформ для хостингу та перегляду відеоматеріалів 18

2.1.1 Огляд сучасного відеохостингу та його значення 18

2.1.2 Переваги та виклики стрімінгового відео 18

2.2 Огляд використовуваних технологій та засобів 19

2.2.1 Node.js 19

2.2.2 Nest.js 19

2.2.3 Amazon Web Services (AWS) 20

2.2.4 AWS S3 20

2.2.5 AWS IVS 20

2.2.6 Docker 21

2.2.8 React.js 22
2.2.9 PostgreSQL 23
2.2.10 Git 24
2.2.11 WebStorm 25
2.2.12 TypeORM 26
2.2.13 AWS Cognito 26
2.2.14 Serverless Framework 27
2.3 Архітектура платформи для хостингу та перегляду відеоматеріалів 27
2.3.1 Клієнт-серверна архітектура та її компоненти 27
2.3.2 Розподілена система для масштабованого стрімінгу відео 28
2.3.3 Модель безпеки та управління доступом до контенту 29
РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА 30
3.1 Розробка архітектури платформи для хостингу та перегляду відеоматеріалів 30
3.1.1 Вибір інструментів та технологій для розробки платформи 30
3.1.2 Проектування структури бази даних для зберігання відеофайлів та метаданих 30
3.1.3 Визначення компонентів та модулів платформи для забезпечення функціональності 35
3.2 Реалізація серверної частини платформи 37
3.2.1 Розробка серверних API для управління відеоконтентом, користувачами та іншими ресурсами 37
3.2.2 Інтеграція з Amazon Web Services для забезпечення збереження та керування відеофайлами та аутентифікації. 51
3.2.3 Розгортання інфраструктурі AWS з використанням Serverless Framework 52
3.3 Реалізація клієнтської частини платформи 52
3.3.1 Створення інтерфейсу користувача з використанням HTML, CSS та React.js: 52
3.3.2 Інтеграція з AWS IVS для стрімінгу відео в режимі реального часу 66
ВИСНОВКИ 69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 71
ДОДАТОК А 73

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

CDN - Content Delivery Network (мережа доставки контенту) - розподілена мережа серверів, яка забезпечує швидку доставку контенту до користувачів у різних регіонах світу.

API - Application Programming Interface (інтерфейс програмування додатків) - набір правил та протоколів, що дозволяють різним програмним додаткам взаємодіяти між собою.

HLS - HTTP Live Streaming (протокол HTTP для стрімінгу) - протокол, який дозволяє стрімінгове відтворення відео та аудіо контенту через мережу Інтернет.

UI/UX дизайн - процес проектування та створення інтерфейсу та враження користувача з продукту з урахуванням його функціональності, естетики та легкості використання.

AWS - Amazon Web Services (Амазон Веб-сервіси) - це хмарна платформа, яка надає широкий спектр послуг, таких як обчислення, зберігання даних, бази даних, мережеві послуги та багато іншого, що дозволяє розробникам будувати й розгортати додатки на основі хмарних обчислень.

HTML - HyperText Markup Language (мова розмітки гіпертексту) - стандартна мова розмітки для створення та представлення веб-сторінок та веб-документів.

CSS - Cascading Style Sheets (каскадні таблиці стилів) - мова стилізації, яка використовується для оформлення веб-сторінок та керування їх зовнішнім виглядом.

ВСТУП

Все більше і більше людей стають споживачами відеоконтенту в онлайн-середовищі. Відеоматеріали, такі як вебінари, стріми, відеоуроки та інші формати, стають популярними засобами комунікації, навчання і розваг для мільйонів користувачів по всьому світу. Зростання популярності відеоматеріалів вимагає надійних та ефективних платформ для їх хостингу, зберігання та перегляду.

Цей проект дипломної роботи присвячений розробці такої платформи, яка відповідатиме потребам сучасного відео-контенту. Метою даної роботи є розробка повнофункціональної платформи для хостингу та перегляду відеоматеріалів з використанням сучасних технологій та засобів.

Для реалізації цієї платформи будуть використані потужні інструменти розробки, такі як Node.js та Nest.js, які дозволять побудувати ефективну та масштабовану серверну частину додатку. Крім того, використання Amazon Web Services (AWS) надасть необхідні можливості для забезпечення інфраструктури платформи, включаючи зберігання відеофайлів з використанням AWS S3 та потокове відтворення відеоконтенту за допомогою AWS IVS.

Платформа буде орієнтована на забезпечення зручного та швидкого хостингу відеоматеріалів для користувачів. Користувачі матимуть можливість легко завантажувати свої відеофайли та відтворювати відео на платформі. Особливою увагою буде приділено передачі відеоконтенту у режимі реального часу, щоб користувачі могли проводити стріми, вебінари та інші заходи, які потребують негайної передачі відео.

Ця робота відповідає актуальним вимогам сучасного ринку відео-контенту, який швидко зростає і постійно змінюється. Однак, існуючі платформи для хостингу та перегляду відео не завжди забезпечують оптимальні швидкість, якість та масштабованість передачі відео. Тому розробка нової платформи, яка використовує передові технології та засоби, є важливим завданням.

Проект використовує Node.js та Nest.js, які вже визнані потужними інструментами для розробки веб-додатків. Вони дозволяють ефективно обробляти запити, керувати базами даних та реалізовувати розширену функціональність. Використання Amazon Web Services (AWS) дозволяє використовувати сучасні хмарні рішення, які забезпечують масштабованість, надійність та безпеку платформи.

Завдяки AWS S3, користувачі зможуть зберігати свої відеофайли в безпечному хмарному середовищі. Це забезпечить надійність зберігання та швидкий доступ до відеоконтенту. AWS IVS дозволить забезпечити потокове відтворення відео в режимі реального часу, що стане основою для проведення вебінарів, онлайн-лекцій та інших подій, які потребують негайної передачі відео.

Одним з головних аспектів розробки платформи є забезпечення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів. Користувачі повинні мати можливість легко завантажувати відео, керувати налаштуваннями приватності та ефективно переглядати вміст. Застосування передових технологій та оптимізація взаємодії з платформою допоможуть досягти цих цілей

Цей проект вирішує важливу проблему, пов'язану зі зростанням популярності відеоконтенту та потребою в потужних платформах для його хостингу та перегляду. Його результати можуть бути використані як основа для подальших розробок та вдосконалення відео-платформи. Дана дипломна робота є важливим внеском у сферу веб-розробки та відео-контенту, а також має потенціал для практичного застосування в різних сферах, включаючи освіту, розваги та бізнес.

РОЗДІЛ 1 АНАЛІЗ СТАНУ ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Зростання популярності відеоконтенту

В останні роки спостерігається значний ріст популярності відеоконтенту в онлайн-середовищі. Відео стає одним з найбільш ефективних та привабливих засобів комунікації, який дозволяє передати інформацію з високою емоційною насиченістю. Соціальні медіа, стрімінгові платформи та інші сервіси перегляду відео залучають мільйони користувачів щоденно.

Зростання популярності відеоконтенту вимагає розробки потужних та надійних платформ для хостингу та перегляду відеоматеріалів. Користувачі хочуть мати можливість швидко та зручно споживати відео контент, а компанії, що створюють власний відеоконтент, потребують ефективних інструментів для розміщення та поширення свого вмісту.

1.2 Недоліки існуючих платформ для хостингу та перегляду відео

Багато з наявних платформ для хостингу та перегляду відео мають свої недоліки, які впливають на якість та зручність використання. Одним із найбільш поширених недоліків є недостатня швидкість завантаження та відтворення відео. Користувачі часто зіткнуться з довгими часами очікування, буферизацією та іншими проблемами, які впливають на їх досвід перегляду.

Крім того, деякі платформи не забезпечують достатньої якості відео, особливо при високому обсязі даних або під час одночасного перегляду багатьох користувачів. Це може призводити до розмиття зображення, втрати деталей та зниження загального враження від контенту.

1.3 Обмежена масштабованість і надійність платформ

Існуючі платформи для хостингу відео часто мають обмежену масштабованість, що стає проблемою при зростанні кількості користувачів та обсягу відеоконтенту. Недостатня масштабованість може призводити до перебоїв у роботі, збоїв та низької якості обслуговування. Крім того, деякі платформи не можуть ефективно масштабуватись при збільшенні навантаження, що призводить до затримок та погіршення якості обслуговування.

1.4 Потреба у передачі медіаконтенту у режимі реального часу

З розвитком технологій та зростанням інтерактивності відеоконтенту, стає все більш важливим мати можливість передавати медіаконтент у режимі реального часу. Це особливо актуально для стрімінгових платформ, вебінарів, онлайн-трансляцій та інших подібних заходів, де важлива миттєва передача інформації та взаємодія з глядачами.

Наразі існують певні виклики та обмеження щодо передачі медіаконтенту у режимі реального часу. Затримки при передачі, низька якість зображення та аудіо, а також складності зі синхронізацією та масштабованістю є поширеними проблемами. Ці недоліки впливають на користувачів та їх сприйняття контенту, а також на якість і результативність проведених заходів.

1.5 Використання стеку технологій та засобів Node.js, Nest.js, Amazon Web Services

Для розробки потужної та ефективної платформи для хостингу та перегляду відеоматеріалів з можливістю передачі медіаконтенту у режимі реального часу, використання сучасних технологій та засобів є ключовим.

Node.js є однією з таких технологій, яка забезпечує швидке та масштабоване виконання коду на стороні сервера. Використання Node.js дозволить забезпечити високу продуктивність та ефективну обробку запитів під час хостингу та передачі відеоматеріалів.

Nest.js, як фреймворк для розробки веб-додатків на Node.js, надасть гнучкість, структурованість та розширюваність для платформи. Використання Nest.js сприятиме розробці модульної та масштабованої архітектури, що дозволить легко розширювати та вдосконалювати функціональність платформи.

Amazon Web Services (AWS) є провідним хмарним сервісом, який надає широкий спектр інфраструктурних рішень та послуг для розробки та розгортання програмних додатків. Використання AWS дозволить забезпечити високу доступність, масштабованість та безпеку платформи, а також використовувати спеціалізовані сервіси, такі як AWS S3 та AWS IVS, для зберігання відеоконтенту та передачі його у режимі реального часу відповідно.

Аналіз стану проблемної області підкреслює необхідність розробки нової платформи для хостингу та перегляду відеоматеріалів з можливістю передачі медіаконтенту у режимі реального часу. Дана дипломна робота спрямована на розв'язання проблем, пов'язаних зі швидкістю, якістю, масштабованістю та безпекою передачі відео. Розроблена платформа буде базуватись на стеку технологій та засобів Node.js, Nest.js та Amazon Web Services, зокрема AWS S3 та AWS IVS, що забезпечить високу продуктивність, ефективність та надійність системи.

1.6 Потреба в ефективному управлінні великим обсягом відеоконтенту

Зі зростанням популярності відеоконтенту, особливо на платформах стрімінгу та спільного перегляду відео, виникає потреба в ефективному управлінні великим обсягом відеоматеріалів. Керування великими масивами відеоконтенту, його категоризація, організація та пошук є важливими аспектами для забезпечення зручного та приємного досвіду користувачів.

Існуючі платформи для хостингу та перегляду відео не завжди надають достатньо розширені можливості управління вмістом. Відсутність потужних інструментів для категоризації, маркування та організації відеоконтенту може ускладнювати пошук та доступ до конкретних матеріалів. Додатково, відсутність функцій рекомендацій та персоналізації може обмежувати можливості користувачів відкривати нові контентні пропозиції та збільшувати залученість до платформи.

1.7 Важливість безпеки та конфіденційності відеоданих

У сучасному цифровому світі, де великий обсяг відеоданих обмінюється та зберігається онлайн, забезпечення безпеки та конфіденційності стає надзвичайно важливим завданням. Користувачі платформ для хостингу та перегляду відео очікують, що їх особиста інформація, включаючи дані про перегляд, буде захищена від несанкціонованого доступу та зловживань.

Існуючі платформи не завжди забезпечують належний рівень безпеки та конфіденційності відеоданих[1]. Вразливості, такі як недостатньо захищені механізми аутентифікації та авторизації, можуть ставити під загрозу приватність користувачів. З метою забезпечення надійності та захищеності відеоданих, важливо використовувати сучасні технології шифрування, аутентифікації та контролю доступу[2].

Аналіз стану проблемної області підкреслює потребу у розробці платформи, яка забезпечує ефективне управління великим обсягом відеоконтенту, а також забезпечує високий рівень безпеки та конфіденційності відеоданих. Розроблена платформа, базована на стеку технологій Node.js, Nest.js та Amazon Web Services, зосередиться на розширених можливостях управління вмістом, включаючи категоризацію, організацію та пошук відеоконтенту, а також надання рекомендацій та персоналізованого досвіду користувачам. Крім того, будуть використані заходи забезпечення безпеки та конфіденційності,

включаючи сучасні механізми шифрування, аутентифікації та контролю доступу.

1.8 Розподілені системи та вимоги до масштабованості

У сучасному світі швидкого розвитку технологій та зростання користувацької бази, розподілені системи стають необхідним елементом успішної платформи для хостингу та перегляду відеоконтенту. Розподілені системи дозволяють ефективно розподіляти навантаження та забезпечувати високу доступність та швидкодію платформи.

Існуючі платформи можуть стикатися з обмеженнями в масштабованості, особливо під великим навантаженням або при зростанні користувацької бази. Відсутність гнучкості та масштабованості може призводити до зниження продуктивності, недоступності платформи та негативного впливу на користувацький досвід.

1.9 Аналіз сучасних відеохостингів і їхніх обмежень

Сучасні відеохостинги є популярними інструментами для завантаження, зберігання та перегляду відеоматеріалів. Вони надають зручність і доступність для користувачів, проте, виникають ряд проблем та обмежень, які впливають на їхню функціональність та користувацький досвід.

Однією з найбільших проблем є обмежена масштабованість. При великій кількості користувачів або великому обсязі завантажених відео, відеохостинги можуть стикатися зі зниженням швидкодії та перебоїв у роботі. Це може призводити до негативного впливу на якість перегляду відеоматеріалів та задоволення користувачів.

Крім того, обмежені можливості управління вмістом є іншою проблемою, з якою зіштовхуються відеохостинги. Недостатня підтримка для редагування, сортування та категоризації відео може ускладнити навігацію та пошук контенту

для користувачів. Відсутність інтеграції з іншими сервісами та платформами також обмежує можливості взаємодії та розширення функціоналу.

1.10 Передові можливості стрімінгу відео

Розвиток технологій стрімінгу відео відкриває нові непередбачувані можливості для користувачів та бізнесу. Завдяки вдосконаленню мережевих технологій та швидкості Інтернет-з'єднання, користувачі можуть насолоджуватися високоякісним відеоконтентом у режимі реального часу без необхідності завантажувати його перед переглядом. Це дозволяє їм швидко знаходити та споживати відеоматеріали зі своїх улюблених жанрів, медіа-каналів та контент-провайдерів.

Для бізнесу стрімінг відео відкриває нові можливості просування та монетизації контенту. Компанії можуть створювати власні канали та платформи для стрімінгу відео, привертаючи аудиторію та розміщуючи рекламу чи заряджаючи плату за доступ до ексклюзивного контенту. Такі моделі бізнесу дозволяють підприємствам залучати нові аудиторії, розширювати свою репутацію та отримувати прибуток від свого відео-контенту.

1.11 Роль соціальних медіа у стрімінгу відео

Соціальні медіа відіграють важливу роль у популяризації та поширенні відеоконтенту. Платформи соціальних медіа, такі як YouTube, Instagram і TikTok, стають не лише місцем для споживання відео, але й майданчиком для його створення та поширення. Користувачі можуть легко створювати свої власні відеоматеріали та ділитися ними зі своєю аудиторією через соціальні мережі. Більшість платформ також надають можливість коментувати, вподобати та ділитися відео з іншими користувачами, стимулюючи взаємодію та вірусне поширення контенту.

1.12 Вплив мобільних технологій на стрімінг відео

Зростання популярності мобільних технологій значно змінило спосіб споживання відеоконтенту. Смартфони та планшетні пристрої стали основними засобами доступу до відеоматеріалів для багатьох користувачів. Мобільні технології дозволяють людям переглядати відео на ходу, незалежно від місця перебування, забезпечуючи гнучкість та зручність. Розвиток високошвидкісних мобільних мереж 4G та 5G додатково сприяє зручності стрімінгу відео, забезпечуючи стабільну передачу високоякісного контенту навіть у руху.

Звертаючи увагу на останні тренди та досягнення в області стрімінгу відео, можна зробити висновок, що ця сфера продовжує активно розвиватися, надаючи нові можливості для користувачів та бізнесу. Перехід до реального часу та зручне споживання відеоконтенту на різних пристроях робить стрімінг відео потужним інструментом в сучасному цифровому світі[3].

1.13 Вплив стрімінгу відео на культуру та суспільство

Стрімінговий контент має значний вплив на культуру та суспільство. Він створює нові можливості для спілкування, розваг та освіти. Онлайн-відео стає не тільки засобом розваги, але й платформою для обміну ідеями, культурного обміну та формування глобальних спільнот[4].

Стрімінгові платформи дозволяють широкому колу людей створювати та ділитися своїм власним відеоконтентом. Це сприяє розкриттю талантів, популяризації різноманітних культурних виразів та незалежному творчому вираженню. Відео може впливати на свідомість глядачів, викликати емоції, створювати соціальні дискусії та спонукати до дії.

Окрім того, стрімінговий відеоконтент стає інструментом навчання та освіти. Онлайн-уроки, тренінги та документальні фільми доступні у будь-який час і з будь-якого місця, що дозволяє здобувати знання та розвиватися швидше та ефективніше.

Проте, вплив стрімінгу відео також має свої виклики. Залежність від постійного доступу до відеоконтенту може призвести до втрати продуктивності, відсутності особистого спілкування та негативного впливу на фізичне та психічне здоров'я.

Дослідження впливу стрімінгу відео на культуру та суспільство є актуальним і допомагає краще розуміти цю динамічну область. Дослідники вивчають вплив відеоконтенту на цінності, світогляд, сприйняття та поведінку глядачів. Це дозволяє розробляти стратегії контенту та політики, які відповідають потребам та вимогам сучасного суспільства.

Отже, вивчення впливу стрімінгу відео на культуру та суспільство важливо для розуміння та адаптації до змін, які вносить ця сучасна форма медіа.

1.14 Виклики забезпечення безпеки та конфіденційності відеоконтенту

Забезпечення безпеки та конфіденційності відеоконтенту є надзвичайно важливим завданням для платформ хостингу та перегляду відеоматеріалів. З ростом популярності стрімінгового відео і збільшенням обсягу персональних та конфіденційних даних, які обробляються, виникають нові виклики та загрози, які потребують уваги та заходів безпеки.

Одним з викликів є захист особистих даних користувачів. Платформи повинні розробляти та впроваджувати механізми, які забезпечують безпеку зберігання та передачі особистих даних, включаючи інформацію про облікові записи, платежі та персональні відомості. Важливо дотримуватися найвищих стандартів шифрування та захисту даних, а також забезпечити належний контроль доступу до них.

Несанкціонований доступ до відеоконтенту із злочинними метою також становить загрозу. Платформи повинні розробляти механізми автентифікації та авторизації, які гарантують, що тільки правомірні користувачі мають доступ до контенту. Додатково, моніторинг та виявлення потенційно шкідливого вмісту допомагає запобігти поширенню небажаного або образливого матеріалу.

Зокрема, виникає потреба в розробці механізмів фільтрації контенту, які допомагатимуть ідентифікувати та блокувати матеріал, що порушує авторські права, містить неприпустимий вміст або спричиняє негативні наслідки для користувачів. Це може включати використання алгоритмів машинного навчання та штучного інтелекту для автоматичного виявлення порушень.

Загальна безпека та довіреність платформи також є важливими аспектами. Платформи повинні мати механізми моніторингу, виявлення та реагування на порушення правил використання, які допомагають підтримувати безпеку та чесність серед користувачів. Розробка системи зворотного зв'язку, де користувачі можуть повідомляти про проблеми та отримувати підтримку, також сприяє встановленню довіри та покращенню якості платформи.

Забезпечення безпеки та конфіденційності відеоконтенту є складною задачею, яка вимагає поєднання технічних, організаційних та юридичних заходів. Виклики, пов'язані з цим, постійно зростають, оскільки розвиток технологій і зміна поведінки користувачів призводять до нових загроз і вразливостей. Тому важливо продовжувати дослідження та розробляти інноваційні рішення, які забезпечують високий рівень безпеки та конфіденційності на платформах для хостингу та перегляду відеоматеріалів.

1.15 Повна локалізація на українську мову

Однією з найважливіших особливостей платформи для хостингу та перегляду відеоматеріалів є повна локалізація на українську мову. Це означає, що кожен елемент інтерфейсу, кожне повідомлення, кожен контент на платформі будуть доступні користувачам українською мовою. Повна локалізація на українську мову забезпечує зручність, зрозумілість та доступність платформи для україномовних користувачів.

Окрім локалізації інтерфейсу, важливо також забезпечити наявність українськомовного контенту на платформі. Це можуть бути відео, фільми, серіали, аудіозаписи та інші матеріали, що створені українськими авторами або

присвячені українській культурі, мові, історії тощо. Розширення асортименту україномовного контенту на платформі допоможе задовольнити потреби українських глядачів і сприятиме популяризації та підтримці української культурної сцени.

Наприклад, платформа може співпрацювати з українськими кінокомпаніями, музичними лейблами, телевізійними каналами та іншими виробниками контенту, щоб включити їхні матеріали на платформі. Також можна сприяти розвитку та підтримці українських творчих спільнот, давати їм можливість публікувати свої відео, музику та інший контент на платформі.

Повна локалізація на українську мову та розширення україномовного контенту на платформі мають велике значення для зміцнення національної ідентичності, культурного спадку та мови. Вони сприяють поширенню української мови як мови комунікації та вираження творчого потенціалу. Також вони стимулюють розвиток української медіаіндустрії та креативних галузей, сприяючи залученню нових талантів та формуванню культурного екосистеми в Україні[5].

Українська мова є невід'ємною частиною національної ідентичності та культурного багатства України. Тому розробка платформи для хостингу та перегляду відеоматеріалів з повною локалізацією на українську мову є важливим кроком у збереженні та популяризації української мови, культури та ідентичності.

РОЗДІЛ 2 ВИКОРИСТАНІ ІНСТРУМЕНТИ ТА ТЕХНОЛОГІЇ

2.1 Вступ до платформ для хостингу та перегляду відеоматеріалів

2.1.1 Огляд сучасного відеохостингу та його значення

В сучасному цифровому світі, відеоконтент є одним з найпопулярніших та широко використовуваних форматів змісту. Відео є потужним засобом комунікації, який може передати інформацію, викликати емоції та надихнути глядачів. Інтернет та постійний доступ до високошвидкісного Інтернету сприяють популярності відеоконтенту, а також змінюють спосіб, яким ми споживаємо цей контент.

Відеохостинги стали невід'ємною частиною цифрового вмісту, надаючи платформу для зберігання, керування та поширення відеоматеріалів. Вони дозволяють користувачам завантажувати, зберігати та ділитися своїми відео, а також дозволяють переглядати відеоконтент, створений іншими користувачами. Це дозволяє людям з усього світу споживати відеоматеріали за зручним для них часом та місцем, а також сприяє розвитку культури та спільноти відеостворювачів.

2.1.2 Переваги та виклики стрімінгового відео

Одним із ключових аспектів відеохостингу є можливість стрімінгу відеоконтенту. Стрімінгове відео дозволяє користувачам переглядати відеоматеріали без необхідності завантажувати їх повністю на свої пристрої. Це забезпечує швидкий та зручний доступ до відеоконтенту, що є особливо важливим в епоху мобільних пристроїв та підвищеної мобільності користувачів.

Стрімінгове відео також дозволяє передавати відеоконтент у режимі реального часу. Це особливо важливо для подій у прямому ефірі, вебінарів, спортивних змагань, концертів та інших подій, які потребують миттєвої передачі відеоконтенту глядачам у різних частинах світу. Стрімінговий відеоконтент відкриває безліч можливостей для інтерактивності та взаємодії глядачів з відео, таких як коментування в режимі реального часу, чати та спільне переглядання.

Однак, разом з перевагами стрімінгового відео, існують і виклики, пов'язані з якістю зображення, пропускною здатністю мережі, затримками та іншими технічними аспектами. Ефективне управління цими викликами є ключовим елементом розробки платформи для хостингу та перегляду відеоматеріалів.

2.2 Огляд використовуваних технологій та засобів

2.2.1 Node.js

Node.js є відкритою платформою для розробки серверних додатків з використанням JavaScript. Вона базується на двигуні V8, який є основою для веб-браузерів Google Chrome. Node.js дозволяє виконувати JavaScript на сервері, що робить його ідеальним вибором для розробки веб-додатків з великим обсягом обробки даних та взаємодії з базами даних[6].

Node.js відомий своєю ефективністю та масштабованістю. Він забезпечує асинхронну та подієву модель програмування, що дозволяє обробляти багато одночасних з'єднань без блокування потоку виконання. Це робить Node.js популярним вибором для побудови високонавантажених додатків.

2.2.2 Nest.js

Nest.js є прогресивним фреймворком для розробки серверних додатків на базі Node.js. Він використовує TypeScript для створення масштабованих та

модульних додатків. Nest.js надає зручний механізм роутингу, впроваджує принципи SOLID та сприяє швидкій розробці додатків[7].

Nest.js забезпечує структурований підхід до розробки, де функціональність поділяється на модулі та компоненти[8]. Це сприяє кращій організації коду, його повторному використанню та тестуванню. Nest.js також підтримує використання декораторів, що дозволяє зробити код більш зрозумілим та читабельним.

2.2.3 Amazon Web Services (AWS)

Amazon Web Services (AWS) є провідним хмарним провайдером, який надає широкий спектр послуг та інфраструктури для розробки, розгортання та керування додатками в хмарному середовищі. AWS забезпечує високу доступність, масштабованість та безпеку, що робить його популярним вибором для розробників[9].

2.2.4 AWS S3

AWS S3 (Amazon Simple Storage Service) є сервісом зберігання об'єктів, який надає безпечне та масштабоване сховище для зберігання різних типів даних, включаючи медіаконтент. S3 забезпечує високу доступність та надійність даних, автоматичне резервне копіювання та можливості керування доступом[10].

AWS S3 є ідеальним вибором для зберігання відеоданих, оскільки він дозволяє зберігати великі обсяги даних та забезпечує швидкий доступ до них. За допомогою AWS SDK можна легко взаємодіяти з S3, завантажувати, зчитувати та видаляти файли, а також налаштовувати правила доступу до об'єктів[11].

2.2.5 AWS IVS

AWS IVS (Amazon Interactive Video Service) є сервісом для стрімінгу відео в режимі реального часу. Він надає можливість створювати, кодувати та

відтворювати стріми відео на основі протоколу HTTP. IVS забезпечує низьку затримку та високу якість відео, що робить його ідеальним для розробки платформи з онлайн-відеоконтентом[12].

AWS IVS надає розширені можливості для контролю та монетизації відео, такі як планування трансляцій, захист від несанкціонованого доступу та можливість відтворення реклами. Використання IVS дозволяє створити потужну та гнучку платформу для перегляду відеоматеріалів в режимі реального часу[13].

2.2.6 Docker

Docker є відкритим програмним забезпеченням, що дозволяє упаковувати та розгортати програмні додатки в контейнерах. Він надає зручний та ефективний спосіб ізоляції додатків та їх залежностей, що спрощує процес розробки, розгортання та управління додатками.

Використання Docker у розробці платформи для хостингу та перегляду відеоматеріалів має численні переваги. Завдяки контейнеризації, можна створювати окремі контейнери для різних компонентів платформи, таких як веб-сервер, база даних та інші сервіси. Це спрощує розгортання, масштабування та управління додатком, оскільки кожен контейнер може бути незалежним і має своє середовище[14].

Крім того, Docker дозволяє створювати імейджі, які містять всі необхідні залежності та конфігурацію для запуску додатку. Це дозволяє легко розповсюджувати та відтворювати платформу на різних середовищах, забезпечуючи консистентність та незалежність від конкретної інфраструктури.

Використання Docker у поєднанні з іншими технологіями, такими як Node.js, Nest.js та AWS, дозволить створити потужну та гнучку платформу для хостингу та перегляду відеоматеріалів. Docker допомагає знизити ризик конфліктів та забезпечує швидке розгортання та масштабування додатку, що робить його цінним інструментом у розробці веб-платформи.

2.2.8 React.js

React.js є однією з найпопулярніших JavaScript бібліотек для розробки інтерфейсів користувача. Вона дозволяє розробникам створювати ефективні та інтерактивні компоненти для веб-додатків. React.js базується на концепції компонентного підходу, що дозволяє розбити інтерфейс на невеликі незалежні компоненти, які можуть бути повторно використані та легко підтримувані.

Одним з ключових принципів React.js є віртуальний DOM (Document Object Model). Він забезпечує швидку та ефективну оновлення стану компонентів, що відображаються на сторінці. React.js автоматично визначає тільки необхідні зміни в DOM та виконує їх, що призводить до зменшення навантаження на браузер та покращує продуктивність веб-додатків.

Одним з найсильніших аспектів React.js є його розширюваність та екосистема. За допомогою різних додаткових пакетів та бібліотек, таких як React Router для навігації, Redux для управління станом, або Material-UI для готових компонентів, розробники можуть швидко розширювати функціональність своїх додатків та спрощувати розробку.

React.js також надає можливість використовувати JSX (JavaScript XML), який є розширенням JavaScript, що дозволяє описувати компоненти за допомогою синтаксису, схожого на HTML. Це дозволяє розробникам легко відтворювати структуру інтерфейсу та комбінувати JavaScript логіку з розміткою.

Завдяки своїй популярності та широкому спільноті розробників, React.js має велику кількість документації, уроків та прикладів, що спрощує процес навчання та розробки. Крім того, React Native, що базується на React.js, дозволяє розробляти мобільні додатки для платформ iOS та Android, використовуючи знайомий синтаксис та підхід.

Усе це робить React.js потужним інструментом для розробки веб-додатків з багатим інтерфейсом та високою продуктивністю. Використання React.js у контексті нашої платформи для хостингу та перегляду відеоматеріалів дозволяє

створювати зручні та ефективні інтерфейси для користувачів та забезпечує швидку відповідь на їх дії.

2.2.9 PostgreSQL

PostgreSQL є потужною реляційною базою даних, яка здатна задовольнити потреби розробки платформи для хостингу та перегляду відеоматеріалів. Він пропонує багато функцій та можливостей, які сприяють ефективному зберіганню, керуванню та обробці даних.

Переваги використання PostgreSQL включають:

1. Надійність та стабільність: PostgreSQL відомий своєю стабільністю та надійністю. Він має механізми відновлення після відмови, що забезпечують неперервну роботу системи навіть у разі аварійних ситуацій.
2. Розширені можливості: PostgreSQL підтримує широкий спектр розширень та розширюваних функцій. Він має вбудовану підтримку географічних даних, текстового пошуку, JSON-документів, роботи з графами та багато іншого.
3. Висока продуктивність: PostgreSQL володіє оптимізованим двигуном запитів, який забезпечує швидку та ефективну обробку даних. Він також підтримує паралельну обробку запитів, що дозволяє розподілити завантаження на багатоядерні системи.
4. Безпека даних: PostgreSQL надає різні механізми безпеки для захисту даних. Він підтримує рівень автентифікації, ролей та дозволів, що дозволяє точно керувати доступом до бази даних.
5. Сумісність та підтримка стандартів: PostgreSQL відповідає стандартам SQL та має високий рівень сумісності з іншими реляційними базами даних. Він також має активне співтовариство розробників, яке забезпечує підтримку та постійну розробку.

Завдяки своїм можливостям, PostgreSQL є привабливим вибором для розробки платформи для хостингу та перегляду відеоматеріалів, де зберігання та управління даними є важливою складовою частиною системи.

2.2.10 Git

Git є однією з найпопулярніших систем керування версіями і є незамінним інструментом для розробки платформи для хостингу та перегляду відеоматеріалів. Він дозволяє ефективно керувати репозиторіями коду, відстежувати зміни, співпрацювати з іншими розробниками та здійснювати розвиток програмного продукту в різних гілках.

Основні переваги використання Git включають:

Розподілена система керування версіями: Git працює за розподіленою моделлю, що означає, що кожен розробник має повну копію репозиторію. Це дозволяє незалежно працювати над різними функціями та злити зміни в один центральний репозиторій.

1. Історія змін та версіонування: Git зберігає повну історію змін файлів, що дозволяє відстежувати та повертатися до попередніх версій коду. Це надає зручність та безпеку при розробці, виправленні помилок та відновленні робочого стану.
2. Галуження та злиття: Git дозволяє створювати гілки, що дозволяє розробникам паралельно працювати над різними функціями або виправленнями. Після завершення роботи над гілкою її можна злити з основною гілкою, щоб включити зміни.
3. Співпраця та командна робота: Git надає зручність у співпраці та командній роботі, дозволяючи розробникам обмінюватися змінами, коментувати код та використовувати інструменти для перегляду змін.
4. Git є потужним інструментом, який сприяє ефективному розвитку програмного продукту та спільній роботі над ним. Його використання в розробці платформи для хостингу та перегляду відеоматеріалів

допомагає забезпечити контроль над версіями коду та спрощує співпрацю між розробниками.

2.2.11 WebStorm

WebStorm є інтегрованою середовищем розробки (IDE), спеціально створеною для розробки веб-додатків. Воно забезпечує широкий набір функцій та інструментів, які допомагають розробникам писати, відлагоджувати та управляти кодом більш ефективно.

Основні можливості WebStorm включають:

1. Підтримка мов програмування: WebStorm підтримує широкий спектр мов програмування, включаючи JavaScript, HTML, CSS, TypeScript та інші. Це дозволяє розробникам працювати з будь-якими технологіями та рамками, використовуваними в платформі для хостингу та перегляду відеоматеріалів.
2. Розширена автодоповнення та перевірка коду: WebStorm надає потужні інструменти для автоматичного завершення коду, що прискорює процес розробки. Воно також перевіряє синтаксис, виявляє помилки та надає рекомендації щодо покращення коду.
3. Інтеграція з системами керування версіями: WebStorm добре інтегрується з розповсюдженими системами керування версіями, такими як Git. Воно надає зручний інтерфейс для роботи з репозиторіями, можливість комітити, злити та керувати версіями коду.
4. Відлагодження коду: WebStorm має вбудований інструмент для відлагодження коду, що допомагає виявляти та виправляти помилки. Воно надає можливість крок за кроком виконувати код, перевіряти значення змінних та аналізувати стек викликів.
5. Інструменти для рефакторингу коду: WebStorm надає набір інструментів для полегшення рефакторингу коду. Це дозволяє

розробникам ефективно переписувати, переносити та оптимізувати код без ризику порушення його функціональності.

WebStorm є потужним інструментом для розробки веб-додатків, який допомагає забезпечити швидкий та ефективний процес розробки. Його функціональність і можливості дозволяють розробникам працювати зручно та продуктивно над платформою для хостингу та перегляду відеоматеріалів.

2.2.12 TypeORM

TypeORM є потужним ORM-фреймворком для TypeScript та JavaScript, який дозволяє зручно працювати з реляційними базами даних. Його основна роль полягає в забезпеченні швидкого та ефективного доступу до даних, а також в управлінні схемою бази даних. TypeORM пропонує багатий набір функціональних можливостей, таких як створення та зміна таблиць, визначення взаємозв'язків між об'єктами, виконання складних запитів та міграція бази даних. Він підтримує різні типи даних, включаючи числа, рядки, дати, булеві значення та інші. Благодаря своїм властивостям та зручному синтаксису TypeORM значно спрощує розробку бази даних та дозволяє розробникам зосередитись на бізнес-логіці своїх додатків.

2.2.13 AWS Cognito

AWS Cognito є повнофункціональним сервісом керування ідентифікацією та доступом користувачів, що надається Amazon Web Services. Він дозволяє розробникам легко і безпечно управляти користувачами, аутентифікацією та авторизацією в їх додатках. AWS Cognito забезпечує різні механізми аутентифікації, включаючи вхід за допомогою соціальних мереж, однофакторну та багатофакторну аутентифікацію, інтеграцію з OpenID Connect та SAML. Він

також надає можливість керування ролями та дозволами користувачів, забезпечує безпеку доступу до ресурсів і захищає конфіденційність даних. AWS Cognito інтегрується з іншими сервісами AWS, такими як Amazon S3 і Amazon API Gateway, що дозволяє створювати безпечні та масштабовані додатки з використанням хмарних ресурсів.

2.2.14 Serverless Framework

Serverless Framework є потужним інструментом для розробки та розгортання серверних додатків без необхідності керування інфраструктурою. Він дозволяє розробникам писати функції додатків, які виконуються в хмарному середовищі, і автоматично керує процесом розгортання та масштабування цих функцій. Serverless Framework підтримує різні хмарні платформи, включаючи AWS, Google Cloud Functions та Microsoft Azure Functions. Він надає розробникам зручні інструменти для опису конфігурації додатків, включаючи визначення функцій, налаштування тригерів та обробку подій. Serverless Framework також дозволяє використовувати сторонні бібліотеки та сервіси, що спрощує розробку складних додатків та підвищує продуктивність розробників. Використання Serverless Framework дозволяє розробникам сконцентруватися на бізнес-логіці своїх додатків, не турбуючись про інфраструктуру, і забезпечує гнучкість та ефективність розгортання додатків в хмарному середовищі.

2.3 Архітектура платформи для хостингу та перегляду відеоматеріалів

2.3.1 Клієнт-серверна архітектура та її компоненти

У платформі для хостингу та перегляду відеоматеріалів використовується клієнт-серверна архітектура, яка розділяє функціональність на дві основні частини: клієнтську та серверну. Клієнтська частина відповідає за відображення

інтерфейсу користувача та взаємодію з ним, включаючи перегляд відеоматеріалів. Серверна частина відповідає за обробку запитів користувачів, зберігання та керування медіаконтентом.

У складі клієнт-серверної архітектури платформи можуть бути наступні компоненти:

Клієнтський додаток: Розроблений за допомогою JavaScript-фреймворку, такого як React.js, цей додаток забезпечує інтерактивний інтерфейс користувача для перегляду та управління відеоматеріалами.

Веб-сервер: Використовується для обробки запитів від клієнтів та відправки їм необхідної інформації. Веб-сервер може бути реалізований з використанням Node.js та Nest.js.

Система зберігання: Забезпечує зберігання медіаконтенту, такого як відеофайли, зображення та інші ресурси. У даній платформі можна використовувати сервіс зберігання об'єктів Amazon Web Services S3 (Simple Storage Service).

База даних: Використовується для зберігання даних про користувачів, відеофайли, налаштування та інші важливі дані. Можна використовувати реляційну базу даних, таку як PostgreSQL.

2.3.2 Розподілена система для масштабованого стрімінгу відео

Одним з ключових аспектів платформи для хостингу та перегляду відеоматеріалів є здатність до масштабованості і стрімінгу відео в режимі реального часу. Для досягнення цієї мети можна використовувати розподілену систему.

Розподілена система передбачає розміщення серверів у різних географічних областях для забезпечення швидкості та доступності стрімінгу відео для користувачів з різних регіонів. Також, розподілена система може використовувати кешування відеоданих на ближніх до користувача серверах, що сприяє зменшенню затримок та покращенню швидкості завантаження.

При розробці розподіленої системи можна використовувати різні технології та підходи, такі як використання CDN (Content Delivery Network) для глобального розподілу контенту, використання мікросервісної архітектури для гнучкості та масштабованості системи, а також використання потокових технологій, наприклад, WebRTC, для передачі відеоданих у режимі реального часу.

2.3.3 Модель безпеки та управління доступом до контенту

Забезпечення безпеки та управління доступом до контенту є важливим аспектом платформи для хостингу та перегляду відеоматеріалів. Доступ до відеоконтенту може бути обмежений залежно від рівня авторизації користувача, прав доступу та налаштувань приватності.

Для реалізації моделі безпеки та управління доступом можна використовувати різні методи та технології, включаючи:

Автентифікація та авторизація користувачів: Використання системи автентифікації, такої як OAuth або JWT (JSON Web Tokens), для перевірки ідентифікації користувача та надання йому відповідних прав доступу.

Рівні доступу: Встановлення різних рівнів доступу до контенту в залежності від ролі користувача, наприклад, адміністратора, модератора або звичайного користувача.

Для забезпечення безпеки платформи також слід враховувати заходи щодо захисту від зловмисних атак, таких як SQL-ін'єкції, перехоплення сесій, злам паролів тощо. Для цього можна використовувати механізми шифрування, валідації даних, моніторингу системи та інші заходи безпеки.

Таким чином, архітектура платформи для хостингу та перегляду відеоматеріалів включає клієнт-серверну архітектуру, розподілену систему для масштабованого стрімінгу відео та модель безпеки та управління доступом до контенту. Ці компоненти сприяють ефективному функціонуванню платформи та забезпечують зручний та безпечний перегляд відеоматеріалів для користувачів.

РОЗДІЛ 3 ПРАКТИЧНА ЧАСТИНА

3.1 Розробка архітектури платформи для хостингу та перегляду відеоматеріалів

3.1.1 Вибір інструментів та технологій для розробки платформи

Для розробки платформи для хостингу та перегляду відеоматеріалів необхідно ретельно вибрати найбільш підходящі інструменти та технології. Один з основних інструментів - Node.js - дозволяє виконувати JavaScript на стороні сервера та надає зручне середовище для розробки серверних додатків. Для забезпечення ефективності та масштабованості можна використати фреймворк Nest.js, який побудований на базі Node.js і забезпечує структуру та організацію серверного коду.

При розробці платформи для хостингу та перегляду відеоматеріалів також варто врахувати можливості, які надає Amazon Web Services (AWS). AWS пропонує широкий набір хмарних сервісів, що дозволяють розгортати, масштабувати та керувати додатками у хмарі. AWS Cognito - це сервіс для аутентифікації та авторизації користувачів, який можна використати для захисту доступу до платформи. AWS S3 - це об'єктне сховище, яке забезпечує безпечне та масштабоване зберігання відеофайлів. AWS IVS - це сервіс для стрімінгу відео в режимі реального часу, який можна використати для надання можливості перегляду відеоматеріалів у реальному часі на платформі.

3.1.2 Проектування структури бази даних для зберігання відеофайлів та метаданих

Один з ключових аспектів розробки платформи для хостингу та перегляду відеоматеріалів - це проектування структури бази даних для зберігання відеофайлів та метаданих, що пов'язані з ними. База даних повинна бути

спроектована таким чином, щоб забезпечити ефективне зберігання та отримання відеоданих.

Одним із можливих варіантів є використання реляційної бази даних, такої як PostgreSQL, яка дозволяє організувати дані у відношеннях та використовувати SQL для збереження та отримання даних. Саме її ми і будемо використовувати для збереження даних сервісу. Для забезпечення оптимальної продуктивності можна використати ORM (Object-Relational Mapping), наприклад TypeORM, який дозволяє працювати з базою даних за допомогою об'єктно-орієнтованого підходу.

Під час проектування структури бази даних слід визначити таблиці та сутності, які будуть використовуватися для збереження відеофайлів та відповідних метаданих, таких як назва, опис, тривалість, категорії тощо. Наприклад, можна створити таблиці для відеофайлів, користувачів, категорій відео тощо. Важливо визначити правильні зв'язки між таблицями, щоб забезпечити цілісність та ефективність бази даних. У дипломній роботі було визначено такі таблиці бази даних: Comment, Subsciption, User, Video, user_liked_videos_video.

Пройдемося по кожній з таблиць та опишемо їх функцію

1. User – зберігає дані користувача

   Опишемо дані які зберігає кожне з полей:

   created_at – дата створення профілю користувача

   updated_at – дата останнього редагування даних

   email – електронна почта користувача

   name – ім’я користувача

   is_verified – статус аккаунта

   subscribers_count – кількість підписників

   description – опис каналу

   avatar_path – посилання на зображення профілю

   streamArn – унікальний ідентифікатор для стрімінгу(IVS)

sub – унікальний ідентифікатор для аутентифікації(Cognito)

id – унікальний ідентифікатор в нашому застосунку

2. Video

   Опишемо дані які зберігає кожне з полей:

   created_at – дата створення відео

   updated_at – дата останнього редагування даних відео

   name – назва відео

   is_public – опція публічності відео

   duration – довжина відео

   comments_count – кількість коментарів

   likes – кількість лайків

   description – опис відео

   views – кількість переглядів

   video_path – посилання на відео

   is_processing – стан обробки відео

   thumbnail_path – посилання на прев'ю

   user_id – зовнішній ключ на користувача

   is_stream – чи є стрімом

   is_active_stream – чи є стрімом який ще йде

   stream_arn – унікальний ключ стріма

   stream_key – ключ стріма необхідний для передачі відео

   stream_url – посилання на стрім

   stream_ingest – сервер стріма необхідний для передачі відео

   id – унікальний ідентифікатор відео

3. Comment

   Опишемо дані які зберігає кожне з полей:

   created_at – дата створення коментаря

   updated_at – дата останнього редагування коментаря

   body – вміст коментаря

user_id – зовнішній ключ на користувача який залишив коментар

video_id – зовнішній ключ на відео на якому залишений коментар

id – унікальний ідентифікатор коментаря

4. Subscriptin

   Опишемо дані які зберігає кожне з полей:

   created_at – дата створення підписки

   updated_at – дата останнього редагування підписки

   from_user – зовнішній ключ на користувача який підписаний

   to_user – зовнішній ключ на користувача на якого підписаний

   id – унікальний ідентифікатор підписки

5. User_liked_videos_video

   Опишемо дані які зберігає кожне з полей:

   userId – зовнішній ключ на користувача який лайкнув відео

   videoId – зовіншній ключ на відео яке лайкнув користувач

Тепер можна детально описати зв’язки між таблицями у даній структурі бази даних

1. User – Video зв’язок один до багатьох реалізується завдяки зовнішньому ключу user_id у таблиці Video
2. Video – Comment – зв’язок один до багатьох реалізується завдяки зовнішньому ключу video_id у таблиці Comment
3. User – Comment – зв’язок один до багатьох реалізується завдяки зовнішньому ключу user_id у таблиці Comment
4. User – Subscription – зв’язок один до багатьох реалізується завдяки зовнішньому ключу from_user у таблиці Subscription
5. User – Subscription – зв’язок багато до одного реалізується завдяки зовнішньому ключу to_user у таблиці Comment
6. User – Video - зв’язок багато до багатьох реалізується завдяки додатковій таблиці user_liked_liked_videos_video

З детальною схемою таблиць у базі даних можна ознайомитись на рис.3.1.2.1

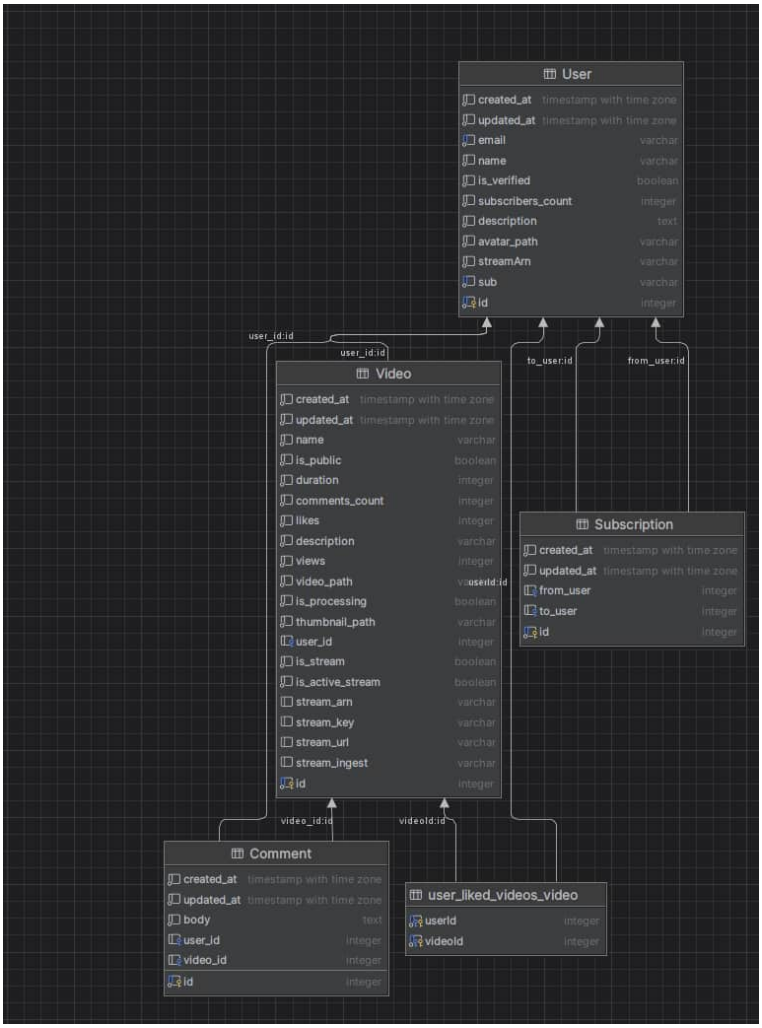


Рисунок 3.1.2.1 – Схема бази даних

Окрім Postgres яка використовується для зберігання метаданих, використовується AWS S3 для зберігання відеоконтенту. AWS S3 серед конкурентів виділяють такі переваги:

1. Масштабованість: AWS S3 дозволяє зберігати практично необмежену кількість даних. Користувачі можуть зберігати великі обсяги відеоконтенту та легко масштабувати ємність зберігання залежно від потреб.
2. Висока доступність: S3 гарантує високу доступність файлів. Він автоматично реплікує дані на різних серверах і дата-центрах, забезпечуючи доступ до файлів навіть у випадку відмови обладнання чи непередбачуваних подій.

3. Безпека: AWS S3 надає різні можливості для забезпечення безпеки даних. Користувачі можуть налаштовувати політику контролю доступу до об'єктів, використовувати шифрування, брандмауери та інші механізми безпеки для захисту відеоконтенту.
4. Простота використання: S3 має простий інтерфейс та надає SDK (набір розробничих інструментів), що дозволяє легко інтегрувати його з різними додатками та системами.
5. Швидкість передачі даних: AWS S3 забезпечує високу швидкість передачі даних, що дозволяє користувачам ефективно завантажувати та отримувати відеофайли.

За допомогою AWS S3, користувачі можуть насолоджуватися масштабованістю, доступністю, безпекою, простотою використання та швидкістю, що надаються цим сервісом для зберігання відеоконтенту.

3.1.3 Визначення компонентів та модулів платформи для забезпечення функціональності

Визначення компонентів та модулів платформи для забезпечення функціональності є ключовим кроком у розробці платформи для хостингу та перегляду відеоматеріалів. Для ефективної роботи та задоволення потреб користувачів, необхідно ретельно розглянути кожен компонент і модуль, які входять до складу системи.

До системи входить 5 компонент, а саме

1. Серверна частина – виконує основні функції управління відеоконтентом та зваємодії з користувачами
2. Клієнтська частина – надає зручний інтерфейс взаємодії з сервісом
3. AWS S3 – відповідає за збереження відеоконтенту
4. AWS IVS – відповідає за надання можливості стрімінгу
5. AWS Cognito – відповідає за аутентифікацію

Одним з найважливіших компонентів є серверна частина, яка виконує основні функції управління відеоконтентом та взаємодії з користувачами. Для досягнення цих цілей було визначено декілька модулів, які забезпечують необхідну функціональність.

Модуль "auth" відповідає за аутентифікацію та авторизацію користувачів на платформі. Цей модуль забезпечує безпеку та конфіденційність, контролюючи доступ до ресурсів та виконуючи перевірку ідентифікаційних даних.

Модуль "comment" відповідає за функціональність коментування відеоматеріалів. Він забезпечує можливість користувачам залишати коментарі, відповідати на них та взаємодіяти з іншими користувачами.

Модуль "media" відповідає за управління відеофайлами та медіаконтентом на платформі. Він забезпечує функціональність завантаження, зберігання та відтворення відеоматеріалів у різних форматах та якості.

Модуль "shared" містить загальні компоненти та функціонал, які можуть бути використані іншими модулями платформи. Це можуть бути загальні функції валідації, кешування даних, обробки помилок тощо.

Модуль "user" відповідає за управління користувачами платформи. Він забезпечує можливість реєстрації, профілювання, керування підписками та налаштуваннями профілю користувача.

Модуль "videos" відповідає за управління відеоконтентом на платформі. Він забезпечує функціональність пошуку, сортування, фільтрації та відображення відеоматеріалів у зручний спосіб для користувачів.

Крім серверної частини, платформа також має клієнтську частину, що надає користувачам зручний інтерфейс для перегляду та взаємодії з відеоконтентом. Клієнтська частина складається з різних компонентів, таких як сторінки channel, home, not-found, profile, register, search-result, studio, subscriptions, trending, video. Кожна з цих сторінок має свою функціональність та взаємодіє з відповідними модулями серверної частини для отримання та відображення необхідної інформації.

Визначені компоненти та модулі платформи співпрацюють між собою, утворюючи єдину систему для забезпечення функціональності хостингу та перегляду відеоматеріалів. Кожен компонент виконує певні завдання, що спрямовані на поліпшення користувацького досвіду та забезпечення безпеки та ефективності роботи платформи.

3.2 Реалізація серверної частини платформи

3.2.1 Розробка серверних API для управління відеоконтентом, користувачами та іншими ресурсами

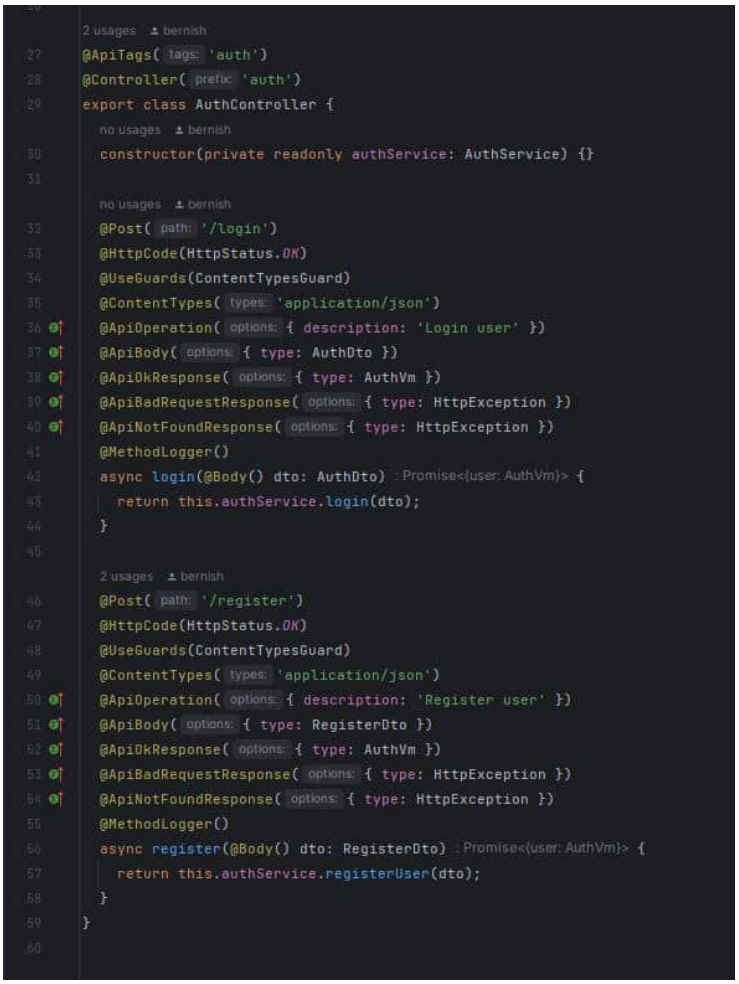
У рамках реалізації серверної частини платформи, розробляються серверні API, які забезпечують взаємодію з клієнтською частиною та управління різними ресурсами. Ці API забезпечують доступ до функцій, пов'язаних з управлінням відеоконтентом, користувачами та іншими складовими системи.

Як було зазначено у Розділі 3.1.3, серверна частина складається з декількох модулів кожен з яких відповідає за свою функцію

Кожен з модулів складається з 2 частин які можна описати MVC паттерном. А саме з Контроллера та Сервісу, який виконує бізнес логіку.

Детально ознайомимся з структурою даних модулів

1. Модуль "auth" відповідає за аутентифікацію та авторизацію користувачів на платформі. Цей модуль забезпечує безпеку та конфіденційність, контролюючи доступ до ресурсів та виконуючи перевірку ідентифікаційних даних.

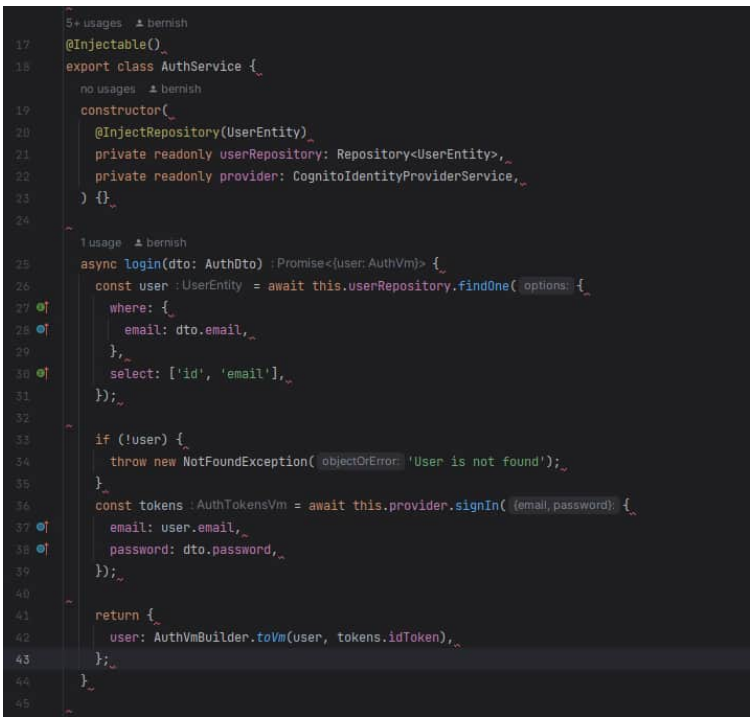


```
2 usages  bernish
@ApiTags( tags: 'auth')
@Controller( prefix: 'auth')
export class AuthController {
  no usages  bernish
  constructor(private readonly authService: AuthService) {}

  no usages  bernish
  @Post( path: '/login')
  @HttpCode(HttpStatus.OK)
  @UseGuards(ContentTypesGuard)
  @ContentTypes( types: 'application/json')
  @ApiOperation( options: { description: 'Login user' })
  @ApiBody( options: { type: AuthDto })
  @ApiOkResponse( options: { type: AuthVm })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @MethodLogger()
  async login(@Body() dto: AuthDto) : Promise<{user: AuthVm}> {
    return this.authService.login(dto);
  }

  2 usages  bernish
  @Post( path: '/register')
  @HttpCode(HttpStatus.OK)
  @UseGuards(ContentTypesGuard)
  @ContentTypes( types: 'application/json')
  @ApiOperation( options: { description: 'Register user' })
  @ApiBody( options: { type: RegisterDto })
  @ApiOkResponse( options: { type: AuthVm })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @MethodLogger()
  async register(@Body() dto: RegisterDto) : Promise<{user: AuthVm}> {
    return this.authService.registerUser(dto);
  }
}
```

Рисунок 3.1.3.1 – код контроллера модуля Auth



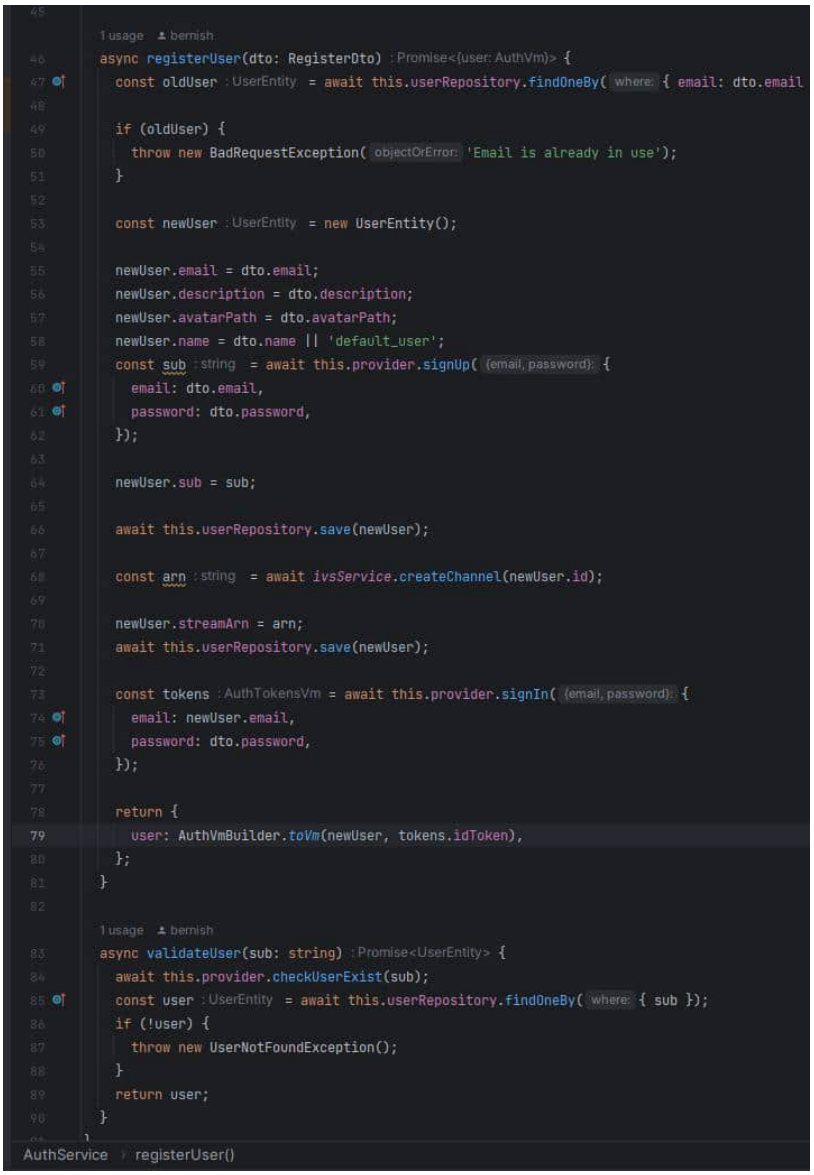
```
5+ usages  bernish
@Injectable()
export class AuthService {
  no usages  bernish
  constructor(
    @InjectRepository(UserEntity)
    private readonly userRepository: Repository<UserEntity>,
    private readonly provider: CognitoIdentityProviderService,
  ) {}

  1 usage  bernish
  async login(dto: AuthDto) : Promise<{user: AuthVm}> {
    const user : UserEntity = await this.userRepository.findOne( options: {
      where: {
        email: dto.email,
      },
      select: ['id', 'email'],
    });

    if (!user) {
      throw new NotFoundException( objectOrError: 'User is not found');
    }
    const tokens : AuthTokensVm = await this.provider.signIn( {email, password}: {
      email: user.email,
      password: dto.password,
    });

    return {
      user: AuthVmBuilder.toVm(user, tokens.idToken),
    };
  }
```

Рисунок 3.1.3.2 – перша частина коду сервісу модуля Auth



```
1 usage  bernish
async registerUser(dto: RegisterDto) : Promise<{user: AuthVm}> {
  const oldUser : UserEntity  = await this.userRepository.findOneBy( where: { email: dto.email

  if (oldUser) {
    throw new BadRequestException( objectOrError: 'Email is already in use');
  }

  const newUser : UserEntity  = new UserEntity();

  newUser.email = dto.email;
  newUser.description = dto.description;
  newUser.avatarPath = dto.avatarPath;
  newUser.name = dto.name || 'default_user';
  const sub : string  = await this.provider.signUp( {email, password}: {
    email: dto.email,
    password: dto.password,
  });

  newUser.sub = sub;

  await this.userRepository.save(newUser);

  const arn : string  = await ivsService.createChannel(newUser.id);

  newUser.streamArn = arn;
  await this.userRepository.save(newUser);

  const tokens : AuthTokensVm = await this.provider.signIn( {email, password}: {
    email: newUser.email,
    password: dto.password,
  });

  return {
    user: AuthVmBuilder.toVm(newUser, tokens.idToken),
  };
}

1 usage  bernish
async validateUser(sub: string) : Promise<UserEntity> {
  await this.provider.checkUserExist(sub);
  const user : UserEntity  = await this.userRepository.findOneBy( where: { sub });
  if (!user) {
    throw new UserNotFoundException();
  }
  return user;
}
}
AuthService  ›  registerUser()
```

Рисунок 3.1.3.3 – друга частина коду сервісу модуля Auth

Він складається з 2 методів в контроллері та 3 методів в сервісі як зображено на рисунках 3.1.3.1, 3.1.3.2, 3.1.3.3

Методи в контроллері виконують функцію валідації та декларації методу і викликають майже одноіменні методи в сервісі.

Login -> Login

Register -> RegisterUser

Методи в сервісі ж виконують такі функції:

Login: аутентифікація і авторизація, повертає JWT токен

RegisterUser: створення аккаунту та авторизація , повертає JWT токен

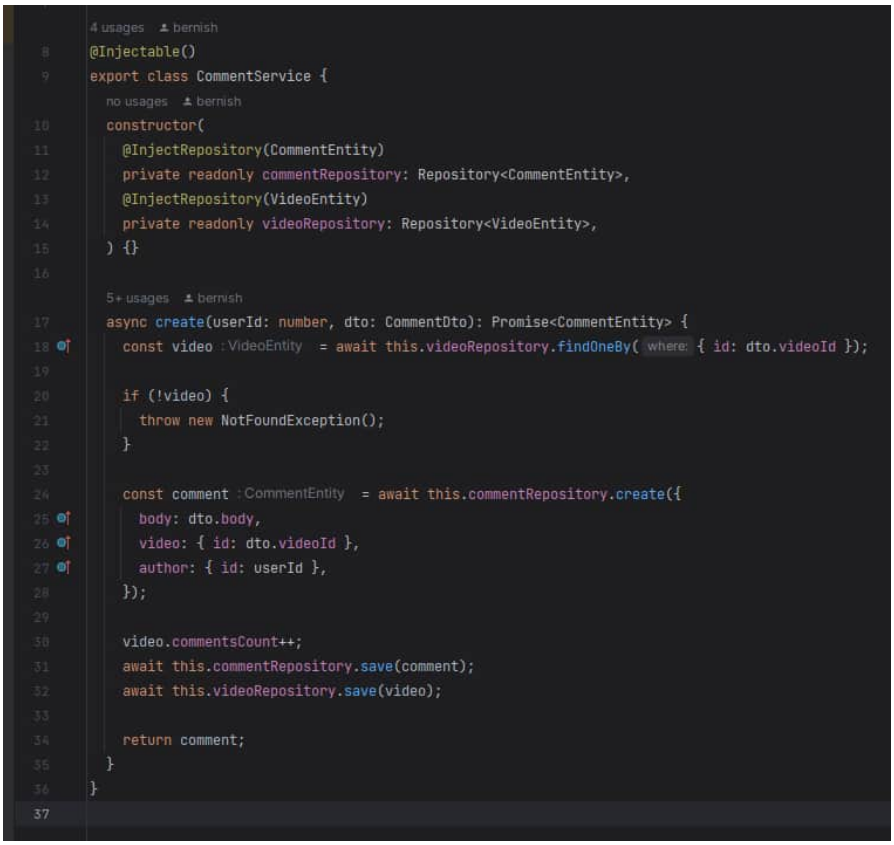
validateUser: слугує для аутентифікації користувача під час активної сессії

2. Модуль "comment" відповідає за функціональність коментування відеоматеріалів. Він забезпечує можливість користувачам залишати коментарі, відповідати на них та взаємодіяти з іншими користувачами.

```
2 usages  bernish
@Controller( prefix: 'comment')
export class CommentController {
  no usages  bernish
  constructor(private readonly commentService: CommentService) {}

  4 usages  bernish
  @Post( path: '/create')
  @UseGuards(JwtUserAuthGuard)
  @ApiBody( options: { type: CommentDto })
  @ApiOkResponse( options: { type: CommentEntity })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @ApiBearerAuth()
  @MethodLogger()
  async create(
    @Req()
    req: IUserRequest,
    @Body() dto: CommentDto,
  ) : Promise<CommentEntity>  {
    return this.commentService.create(+req.user.id, dto);
  }
}
```

Рисунок 3.1.3.4 – код контроллера модуля comment



```
4 usages  bernish
@Injectable()
export class CommentService {
  no usages  bernish
  constructor(
    @InjectRepository(CommentEntity)
    private readonly commentRepository: Repository<CommentEntity>,
    @InjectRepository(VideoEntity)
    private readonly videoRepository: Repository<VideoEntity>,
  ) {}

  5+ usages  bernish
  async create(userId: number, dto: CommentDto): Promise<CommentEntity> {
    const video : VideoEntity  = await this.videoRepository.findOneBy( where: { id: dto.videoId });

    if (!video) {
      throw new NotFoundException();
    }

    const comment : CommentEntity  = await this.commentRepository.create({
      body: dto.body,
      video: { id: dto.videoId },
      author: { id: userId },
    });

    video.commentsCount++;
    await this.commentRepository.save(comment);
    await this.videoRepository.save(video);

    return comment;
  }
}
```

Рисунок 3.1.3.5 – код сервісу модуля comment

Він складається з 1 методів в контроллері та 1 методів в сервісі які зображенні на рисунках 3.1.3.4 та 3.1.3.5

Методи в контроллері виконують функцію валідації та декларації і викликають майже одноіменні методи в сервісі.

create -> create

Методи в сервісі ж виконують такі функції:

create: створює коментар під відео

3. Модуль "media" відповідає за управління відеофайлами та медіаконтентом на платформі. Він забезпечує функціональність завантаження та зберігання відео або зображень.

```
no usages  bernish
@Post( path: '/upload')
@UseGuards(ContentTypesGuard)
@ApiOperation( options: { description: 'Upload file' })
@ContentTypes( types: 'application/json', 'multipart/form-data')
@ApiFile( fileName: 'media')
@ApiConsumes( mimeTypes: 'multipart/form-data')
@ApiBadRequestResponse( options: { type: HttpException })
@UseInterceptors(
  FileFieldsInterceptor(
    uploadFields: [
      {
        name: 'media',
        maxCount: 1,
      },
    ],
    getMulterConfig( config: {
      s3: s3Service.client,
      bucket: configService.uploadFiles.bucket,
      path: configService.uploadFiles.path,
      // fileFilter: filter,
    }),
  ),
)
@MethodLogger()
async upload(@UploadedFiles() files: IUploadFiles) : Promise<{url: string, name: st... {
  if (!files.media?.at(0)) {
    throw new InternalServerErrorException();
  }

  return this.mediaService.saveMedia(files.media[0].key);
}
}
```

Рисунок 3.1.3.6 – код контроллера модуля media

```
4 usages  bernish
@Injectable()
export class MediaService {
  1 usage  bernish
  saveMedia(key: string) : {url: string, name: string}  {
    return {
      url: `${configService.uploadFiles.url}/${key}`,
      name: key,
    };
  }
}
```

Рисунок 3.1.3.7 – код сервісу модуля media

Він складається з 1 методів в контроллері та 1 методів в сервісі зображених на рисунках 3.1.3.6 та 3.1.3.7

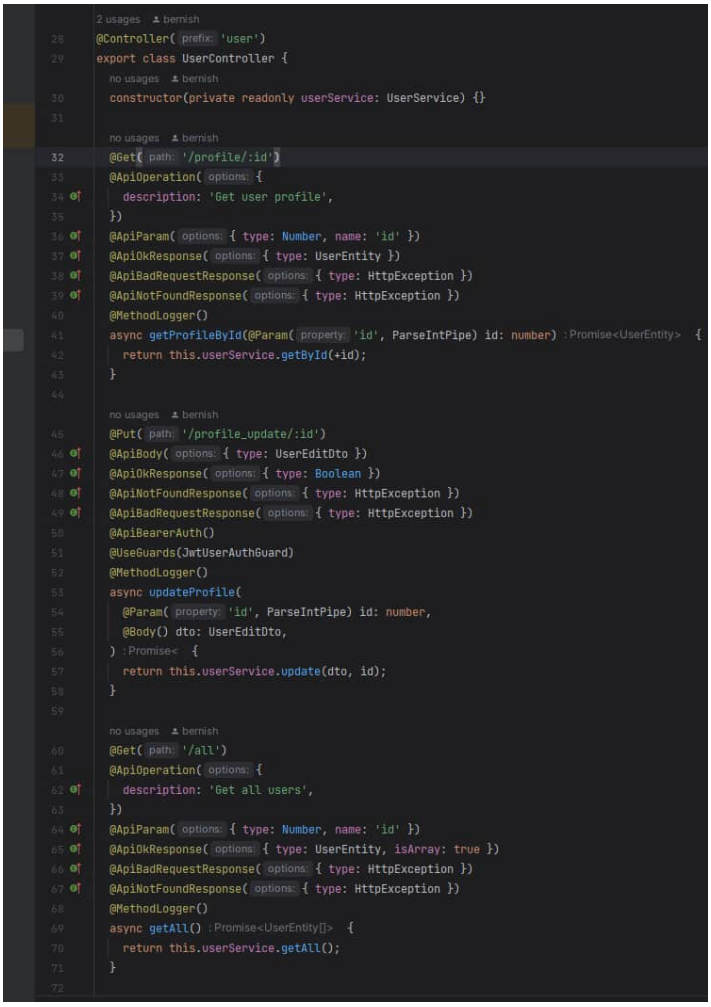
Методи в контроллері виконують функцію валідації та декларації методу і в данному випадку ще завантажують файл в s3 bucket і викликають майже одноіменні методи в сервісі.

Upload -> saveMedia

Методи в сервісі ж виконують такі функції:

saveMedia: Повертає посилання на файл у s3

4. Модуль "shared" містить загальні компоненти та функціонал, які можуть бути використані іншими модулями платформи. Це можуть бути загальні функції валідації, кешування даних, обробки помилок тощо.
5. Модуль "user" відповідає за управління користувачами платформи. Він забезпечує можливість реєстрації, профілювання, керування підписками та налаштуваннями профілю користувача.



```
2 usages  bernish
@Controller( prefix: 'user')
export class UserController {
  no usages  bernish
  constructor(private readonly userService: UserService) {}

  no usages  bernish
  @Get( path: '/profile/:id')
  @ApiOperation( options: {
    description: 'Get user profile',
  })
  @ApiParam( options: { type: Number, name: 'id' })
  @ApiOkResponse( options: { type: UserEntity })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @MethodLogger()
  async getProfileById(@Param( property: 'id', ParseIntPipe) id: number) :Promise<UserEntity>  {
    return this.userService.getById(+id);
  }

  no usages  bernish
  @Put( path: '/profile_update/:id')
  @ApiBody( options: { type: UserEditDto })
  @ApiOkResponse( options: { type: Boolean })
  @ApiNotFoundResponse( options: { type: HttpException })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiBearerAuth()
  @UseGuards(JwtUserAuthGuard)
  @MethodLogger()
  async updateProfile(
    @Param( property: 'id', ParseIntPipe) id: number,
    @Body() dto: UserEditDto,
  ) :Promise<  {
    return this.userService.update(dto, id);
  }

  no usages  bernish
  @Get( path: '/all')
  @ApiOperation( options: {
    description: 'Get all users',
  })
  @ApiParam( options: { type: Number, name: 'id' })
  @ApiOkResponse( options: { type: UserEntity, isArray: true })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @MethodLogger()
  async getAll() :Promise<UserEntity[]>  {
    return this.userService.getAll();
  }
```

Рисунок 3.1.3.8 – перша частина коду контроллера модуля user

```
    no usages  bernish
    @Get( path: '/by_id/:id')
    @ApiOperation( options: {
      description: 'Get user profile',
    })
    @ApiParam( options: { type: Number, name: 'id' })
    @ApiOkResponse( options: { type: UserEntity })
    @ApiBadRequestResponse( options: { type: HttpException })
    @ApiNotFoundResponse( options: { type: HttpException })
    @MethodLogger()
    async getUserById(@Param( property: 'id', ParseIntPipe) id: number) : Promise<UserEntity>  {
      return this.userService.getById(+id);
    }

    no usages  bernish
    @Put( path: '/subscribe')
    @ApiBody( options: { type: SubscribeDto })
    @ApiOkResponse( options: { type: Boolean })
    @ApiNotFoundResponse( options: { type: HttpException })
    @ApiBadRequestResponse( options: { type: HttpException })
    @ApiBearerAuth()
    @UseGuards(JwtUserAuthGuard)
    @MethodLogger()
    async subscribe(@Body() dto: SubscribeDto) : Promise<boolean>  {
      return this.userService.subscribe(dto.userId, dto.channelToSub);
    }
  }
```

Рисунок 3.1.3.9 – друга частина коду контроллера модуля user

```
4 usages  bernish
@Injectable()
export class UserService {
  no usages  bernish
  constructor(
    @InjectRepository(UserEntity)
    private readonly userRepository: Repository<UserEntity>,
    @InjectRepository(SubscriptionEntity)
    private readonly subscriptionRepository: Repository<SubscriptionEntity>,
  ) {}

  1 usage  bernish
  async subscribe(userId: number, userToSubId: number): Promise<boolean> {
    const userToSub : UserEntity  = await this.userRepository.findOneBy( where: { id: userToSubId });
    if (!userToSub) {
      throw new NotFoundException();
    }

    const isSubscribed : SubscriptionEntity  = await this.subscriptionRepository.findOneBy( where: {
      fromUser: { id: userId },
      toUser: { id: userToSubId },
    });

    if (isSubscribed) {
      await this.subscriptionRepository.delete( criteria: {
        id: isSubscribed.id,
      });
      userToSub.subscrubersCount--;
      await this.userRepository.save(userToSub);
      return false;
    }

    const newSub : SubscriptionEntity  = this.subscriptionRepository.create({
      fromUser: { id: userId },
      toUser: { id: userToSubId },
    });
    await this.subscriptionRepository.save(newSub);
    userToSub.subscrubersCount++;
    await this.userRepository.save(userToSub);
    return true;
  }
```

Рисунок 3.1.3.10 – перша частина коду сервісу модуля user

```
2 usages  bernish
async getById(id: number): Promise<UserEntity> {
  const user : UserEntity  = await this.userRepository.findOne( options: {
    where: {
      id: id,
    },
    relations: [
      'videos',
      'subscribers',
      'subscriptions',
      'subscriptions.toUser',
      'likedVideos',
      'videos.user',
    ],
    order: {
      createdAt: 'DESC',
      videos: {
        createdAt: 'DESC',
      },
    },
  });
  if (!user) {
    throw new NotFoundException( objectOrError: 'User not found!');
  }
  return user;
}

1 usage  bernish
async getAll(): Promise<UserEntity[]> {
  return this.userRepository.find();
}

1 usage  bernish
async update(dto: UserEditDto, id: number) : Promise<  {
  const user : UserEntity  = await this.userRepository.findOneBy( where: { id: id });
  if (!user) {
    throw new NotFoundException();
  }
  return await this.userRepository.save( entity: { ...user, ...dto });
}
}
```

Рисунок 3.1.3.11 – друга частина коду сервісу user

Він складається з 5 методів в контроллері та 4 методів в сервісі зображених на рисунках 3.1.3.8, 3.1.3.9 , 3.1.3.10, 3.1.3.11

Методи в контроллері виконують функцію валідації та декларації методу і викликають майже одноіменні методи в сервісі.

getProfileById -> getById

updateProfile -> update

getAll -> getAll

getUserById -> getById

subscribe -> subscribe

Методи в сервісі ж виконують такі функції:

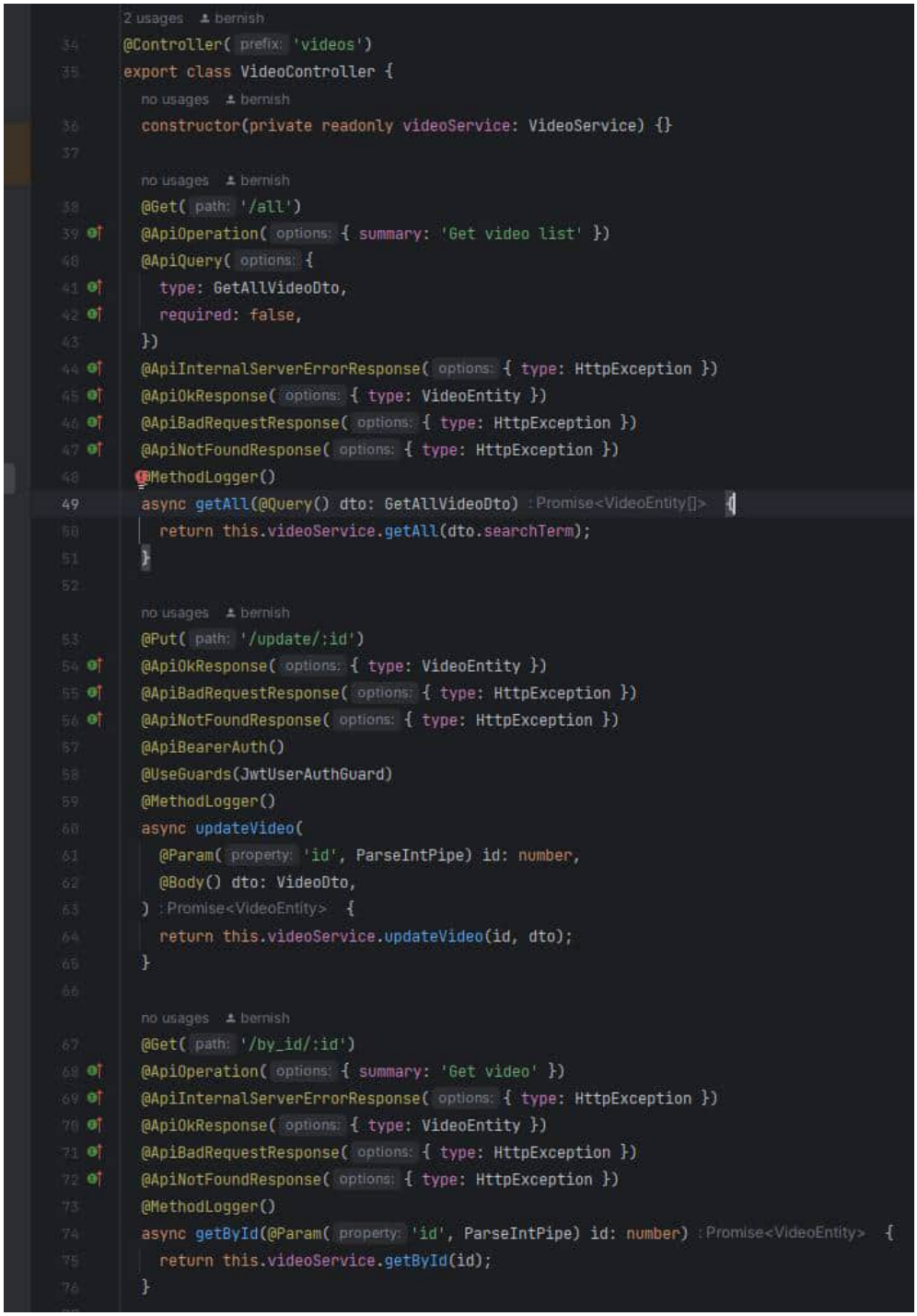
subscribe: Створює підписку на користувача

getById: Повертає дані про користувача по ід

getAll: повертає дані про всіх користувачів

update: змінює дані користувача по ід

6. Модуль "videos" відповідає за управління відеоконтентом на платформі. Він забезпечує функціональність пошуку, сортування, фільтрації та відображення відеоматеріалів у зручний спосіб для користувачів.



```
2 usages  bernish
@Controller( prefix: 'videos')
export class VideoController {
  no usages  bernish
  constructor(private readonly videoService: VideoService) {}

  no usages  bernish
  @Get( path: '/all')
  @ApiOperation( options: { summary: 'Get video list' })
  @ApiQuery( options: {
    type: GetAllVideoDto,
    required: false,
  })
  @ApiInternalServerErrorResponse( options: { type: HttpException })
  @ApiOkResponse( options: { type: VideoEntity })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @MethodLogger()
  async getAll(@Query() dto: GetAllVideoDto) : Promise<VideoEntity[]> {
    return this.videoService.getAll(dto.searchTerm);
  }

  no usages  bernish
  @Put( path: '/update/:id')
  @ApiOkResponse( options: { type: VideoEntity })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @ApiBearerAuth()
  @UseGuards(JwtUserAuthGuard)
  @MethodLogger()
  async updateVideo(
    @Param( property: 'id', ParseIntPipe) id: number,
    @Body() dto: VideoDto,
  ) : Promise<VideoEntity>  {
    return this.videoService.updateVideo(id, dto);
  }

  no usages  bernish
  @Get( path: '/by_id/:id')
  @ApiOperation( options: { summary: 'Get video' })
  @ApiInternalServerErrorResponse( options: { type: HttpException })
  @ApiOkResponse( options: { type: VideoEntity })
  @ApiBadRequestResponse( options: { type: HttpException })
  @ApiNotFoundResponse( options: { type: HttpException })
  @MethodLogger()
  async getById(@Param( property: 'id', ParseIntPipe) id: number) : Promise<VideoEntity>  {
    return this.videoService.getById(id);
  }
```

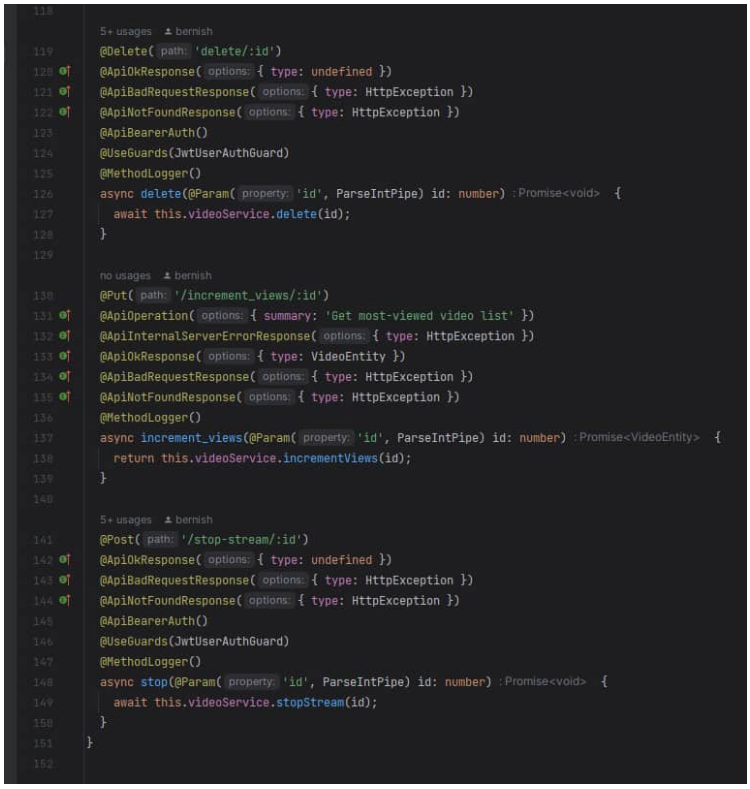
Рисунок 3.1.3.12 – перша частина коду контроллеру модуля videos

```
no usages  bernish
@Get( path: '/most_viewed')
@ApiOperation( options: { summary: 'Get most-viewed video list' })
@ApiInternalServerErrorResponse( options: { type: HttpException })
@ApiOkResponse( options: { type: VideoEntity, isArray: true })
@ApiBadRequestResponse( options: { type: HttpException })
@ApiNotFoundResponse( options: { type: HttpException })
@MethodLogger()
async getMostViewed() :Promise<VideoEntity[]>  {
  return this.videoService.getMostViewed();
}

no usages  bernish
@Put( path: '/update_reaction/:id')
@ApiOkResponse( options: { type: VideoEntity })
@ApiBadRequestResponse( options: { type: HttpException })
@ApiNotFoundResponse( options: { type: HttpException })
@ApiBearerAuth()
@UseGuards(JwtUserAuthGuard)
@MethodLogger()
async updateReactions(
  @Param( property: 'id', ParseIntPipe) id: number,
  @Req()
  req: IUserRequest,
) :Promise<VideoEntity>  {
  return this.videoService.updateReaction(id, +req.user.id);
}

4 usages  bernish
@Post( path: '/create')
@ApiOkResponse( options: { type: VideoEntity })
@ApiBadRequestResponse( options: { type: HttpException })
@ApiNotFoundResponse( options: { type: HttpException })
@ApiBearerAuth()
@UseGuards(JwtUserAuthGuard)
@MethodLogger()
async create(
  @Req()
  req: IUserRequest,
) :Promise<string>  {
  const videoId :number  = await this.videoService.create(+req.user.id);
  return videoId.toString();
}
```

Рисунок 3.1.3.13 – друга частина коду контроллеру модуля videos



```
5+ usages  bernish
@Delete( path: 'delete/:id')
@ApiOkResponse( options: { type: undefined })
@ApiBadRequestResponse( options: { type: HttpException })
@ApiNotFoundResponse( options: { type: HttpException })
@ApiBearerAuth()
@UseGuards(JwtUserAuthGuard)
@MethodLogger()
async delete(@Param( property: 'id', ParseIntPipe) id: number) :Promise<void>  {
  await this.videoService.delete(id);
}

no usages  bernish
@Put( path: '/increment_views/:id')
@ApiOperation( options: { summary: 'Get most-viewed video list' })
@ApiInternalServerErrorResponse( options: { type: HttpException })
@ApiOkResponse( options: { type: VideoEntity })
@ApiBadRequestResponse( options: { type: HttpException })
@ApiNotFoundResponse( options: { type: HttpException })
@MethodLogger()
async increment_views(@Param( property: 'id', ParseIntPipe) id: number) :Promise<VideoEntity>  {
  return this.videoService.incrementViews(id);
}

5+ usages  bernish
@Post( path: '/stop-stream/:id')
@ApiOkResponse( options: { type: undefined })
@ApiBadRequestResponse( options: { type: HttpException })
@ApiNotFoundResponse( options: { type: HttpException })
@ApiBearerAuth()
@UseGuards(JwtUserAuthGuard)
@MethodLogger()
async stop(@Param( property: 'id', ParseIntPipe) id: number) :Promise<void>  {
  await this.videoService.stopStream(id);
}
}
```

Рисунок 3.1.3.14 – третя частина коду контроллеру модуля videos

```
@Injectable()
export class VideoService {
  constructor(
    @InjectRepository(VideoEntity)
    private readonly videoRepository: Repository<VideoEntity>,
    @InjectRepository(UserEntity)
    private readonly userRepository: Repository<UserEntity>,
  ) {}

  async getById(id: number, isPublic = false) {
    const video = await this.videoRepository.findOne({
      where: isPublic
        ? {
            id,
            isPublic: true,
          }
        : { id },
      relations: ['user', 'comments', 'comments.author', 'likedBy'],

      select: {
        user: {
          ...selectUserOptions,
        },
        comments: {
          ...selectCommentOptions,
        },
      },
    });
    if (!video) {
      throw new NotFoundException('Video was not found!');
    }
    return video;
  }
```

Рисунок 3.1.3.15 – перша частина коду сервісу модуля videos

```
async updateVideo(id: number, dto: VideoDto) {
  const video = await this.videoRepository.findOne({
    where: {
      id,
    },
    relations: ['user'],
  });

  if (!video) {
    throw new NotFoundException();
  }

  const user = video.user;

  const isStream = dto.isStream;

  video.name = dto.name;
  video.description = dto.description;
  video.thumbnailPath = dto.thumbnailPath;

  if (isStream) {
    const isActiveStream = await this.videoRepository.findOne({
      where: {
        isStream: true,
        isActiveStream: true,
        user,
      },
    });

    if (isActiveStream) {
      throw new BadRequestException();
    }

    await ivsService.deleteAllStreamKeys(user.streamArn);
    const stream = await ivsService.createStream(user.streamArn);
    const playbackUlr = await ivsService.getPlaybackUrl(user.streamArn);

    video.isActiveStream = true;
    video.isStream = true;
    video.streamKey = stream.value;
    video.streamArn = stream.arn;
    video.isProcessing = false;
    video.isPublic = true;
    video.streamUrl = playbackUlr;
    video.streamIngest = stream.ingest;
  } else {
    video.videoPath = dto.videoPath;
    video.duration = dto.duration;
    video.isProcessing = dto.isProcessing;
    video.isPublic = dto.isPublic;
  }
  return await this.videoRepository.save(video);
}
```

Рисунок 3.1.3.16 – друга частина коду сервісу модуля videos

```
  async getAll(searchTerm?: string) {
    return await this.videoRepository.find({
      where: {
        name: searchTerm ? ILike(`%${searchTerm}%`) : undefined,
        isPublic: true,
        isProcessing: false,
      },
      relations: ['comments', 'user'],
      order: {
        createdAt: 'DESC',
      },
      select: {
        user: {
          ...selectUserOptions,
        },
        comments: {
          ...selectCommentOptions,
        },
      },
    });
  }

  async getMostViewed() {
    const videos = await this.videoRepository.find({
      where: {
        isPublic: true,
        isProcessing: false,
      },
      relations: {
        user: true,
      },
      order: {
        views: 'desc',
      },
      select: {
        user: {
          ...selectUserOptions,
        },
      },
      take: 20,
    });

    return videos;
  }
```

Рисунок 3.1.3.17 – третя частина коду сервісу модуля videos

```
  async create(userId: number): Promise<number> {
    const defaultFields = {
      videoPath: '',
      thumbnailPath: '',
      description: '',
      user: { id: userId },
      name: '',
      isProcessing: true,
    };
    const newVideo = this.videoRepository.create(defaultFields);
    const video = await this.videoRepository.save(newVideo);

    return video.id;
  }

  async delete(id: number) {
    try {
      return await this.videoRepository.delete({ id });
    } catch (error: any) {
      throw new InternalServerErrorException(error);
    }
  }

  async incrementViews(id: number) {
    const video = await this.getById(id);
    video.views++;
    return await this.videoRepository.save(video);
  }

  async updateReaction(id: number, userId: number) {
    const video = await this.getById(id);
    const isLiked = video.likedBy.some((user) => user.id === userId);
    const user = await this.userRepository.findOneBy({ id: userId });

    if (!video) {
      throw new NotFoundException();
    }

    if (!user) {
      throw new NotFoundException();
    }
    if (isLiked) {
      video.likes--;
      video.likedBy = video.likedBy.filter((user) => user.id !== userId);
      return await this.videoRepository.save(video);
    }

    video.likes++;

    video.likedBy.push(user);
    return await this.videoRepository.save(video);
  }
```

Рисунок 3.1.3.18 – четверта частина коду сервісу модуля videos

```
1 usage  ± bernish
async stopStream(videoId: number) :Promise<void>  {
  const activeStream :VideoEntity  = await this.videoRepository.findOne( options: {
    where: {
      isStream: true,
      isActiveStream: true,
      id: videoId,
    },
    relations: ['user'],
  });

  if (!activeStream) {
    throw new BadRequestException();
  }

  activeStream.isActiveStream = false;
  await this.videoRepository.save(activeStream);
  try {
    await ivsService.stopStream(activeStream.user.streamArn);
  } catch (e) {}

  await ivsService.deleteAllStreamKeys(activeStream.user.streamArn);
 }
}
```

Рисунок 3.1.3.17 – п’ята частина коду сервісу модуля videos

Він складається з 5 методів в контроллері та 4 методів в сервісі які зображені на рисунках 3.1.3.12 - 3.1.3.17

Методи в контроллері виконують функцію валідації та декларації методу і викликають майже одноіменні методи в сервісі.

updateVideo -> updateVideo

getById -> getById

getMostViewed -> getMostViewed

updateReactions -> updateReactions

create -> create

delete -> delete

increment_views -> incrementViews

stop -> stop

Методи в сервісі ж виконують такі функції:

getById: Повертає дані про відео по ід

updateVideo: оновлення даних про відео

getAll: Повертає дані про всі відео

getMostViewed: Повертає дані про найпопулярніші відео

create: дозволяє створити відео або стрім
delete: видаляє відео
incrementViews: збільшує кількість переглядів
updateReaction: змінює статус лайку
stopStream: зупиняє пряму трансляцію

Отже серверне API для дозволяють реєструвати нових користувачів, авторизовувати їх, керувати їх профілями, включаючи налаштування, зміну паролю та інші параметри. Вони також надають можливість керування підписками, списками відтворення та іншими функціями, пов'язаними з взаємодією користувачів з платформою. Та управлінням відео контентом.

3.2.2 Інтеграція з Amazon Web Services для забезпечення збереження та керування відеофайлами та аутентифікації.

В рамках реалізації платформи, проводиться інтеграція з Amazon Web Services (AWS), яка спрямована на забезпечення надійного збереження та керування відеофайлами, а також забезпечення безпеки через механізми аутентифікації.

Інтеграція з AWS включає в себе використання різних послуг та сервісів для ефективного збереження відеофайлів. Одним з найважливіших сервісів є Amazon S3 (Simple Storage Service), який надає надійне та масштабоване зберігання об'єктів, включаючи відеофайли. Завдяки інтеграції з Amazon S3, платформа може зберігати великі обсяги відеоданих у безпечному та доступному середовищі.

Окрім збереження та керування відеофайлами, інтеграція з AWS також включає механізми аутентифікації, що забезпечують безпеку та захист платформи. AWS пропонує сервіси, які дозволяють створювати та керувати ідентифікаційними обліковими записами користувачів, такі як Amazon Cognito.

Інтеграція з AWS Cognito дозволяє реалізувати механізми аутентифікації, включаючи можливість створення облікових записів, вхід за допомогою електронної пошти або соціальних мереж, а також керування доступом до ресурсів платформи.

API AWS Cognito використовується у модулі auth для аутентифікації користувача.

API AWS S3 використовується у модулі media для загрузки відео у хмару

3.2.3 Розгортання інфраструктурі AWS з використанням Serverless Framework

Завдяки використанню serverless Framework налаштування сервісів Amazon відбувається завдяки написанню конфігів для нього. З конфігами можна ознайомитись у Додатоку Р-У.

Serverless.yml – головний файл конфігу, в ньому обираються загальні налаштування , підключаються плагіни, імпортуються ресурси.

_resources.yml – слугує для підлючення всіх ресурсів з одного місця

cognito-user-user-pool.yml та cognito-user-user-pool-client.yml – слугують для налаштування cognito

s3-bucket – слугує для налаштування s3

Розгортання всієї необхідної нам інфраструктури у AWS відбувається з виконанням лише одної команди “serverless deploy”

3.3 Реалізація клієнтської частини платформи

3.3.1 Створення інтерфейсу користувача з використанням HTML, CSS та React.js:

Створення інтерфейсу засноване на розробці компонентів, які відповідають різним функціональним частинам платформи. За допомогою HTML структуруються різні елементи, такі як заголовки, кнопки, форми, списки

тощо. CSS використовується для стилізації цих елементів, надаючи їм вигляд і візуальне оформлення.

Одним із ключових аспектів розробки інтерфейсу є використання React.js - популярного JavaScript-фреймворку для побудови користувацьких інтерфейсів. React.js дозволяє створювати компоненти, які забезпечують динамічність і взаємодію з користувачем. За допомогою React.js реалізовуються функціональності, такі як рендерінг списків відеоматеріалів, навігація між сторінками та взаємодія з іншими модулями платформи.

У процесі розробки клієнтської частини було виділено досить багато компонентів, кожен з який описати займе занадто багато часу. Ознайомитись з структурою компонентів можна на Рисунку 3.3.1.1

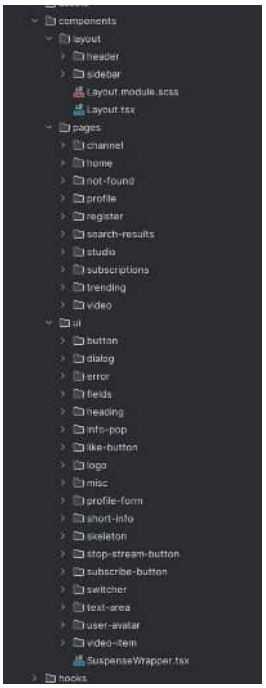


Рисунок 3.3.1.1 – структура компонентів

Серед всіх компонентів можна виділити компоненти які знаходяться у директорії pages – сторінки. Це 10 компонентів які описують головні сторінки клієнтської частини, на них зупинимся детальніше.

1. Компонент channel(Рисунок 3.3.1.2)

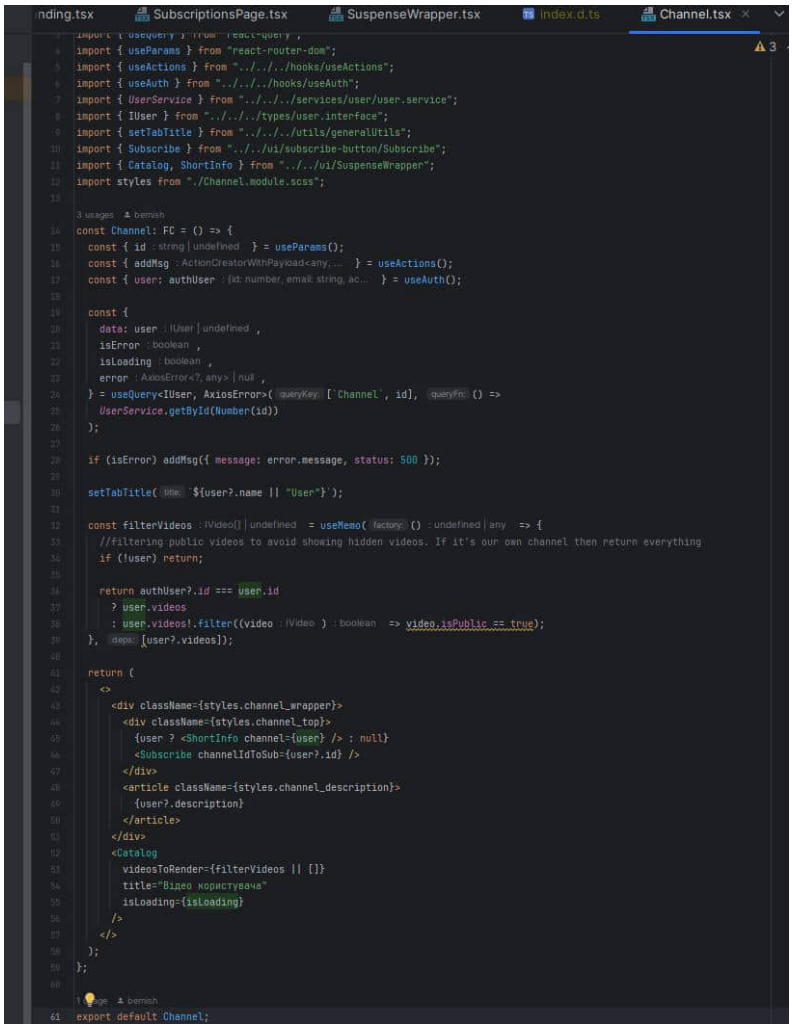


Рисунок 3.3.1.2 – код компоненту channel

Цей компонент відповідає за відображення секції “Мій Канал”

У наведеному у Рисунку 3.3.1.2 коді проходить запит до серверної частини на отримання списку відео користувача, кількості підписників та іншої інформації стосовно каналу

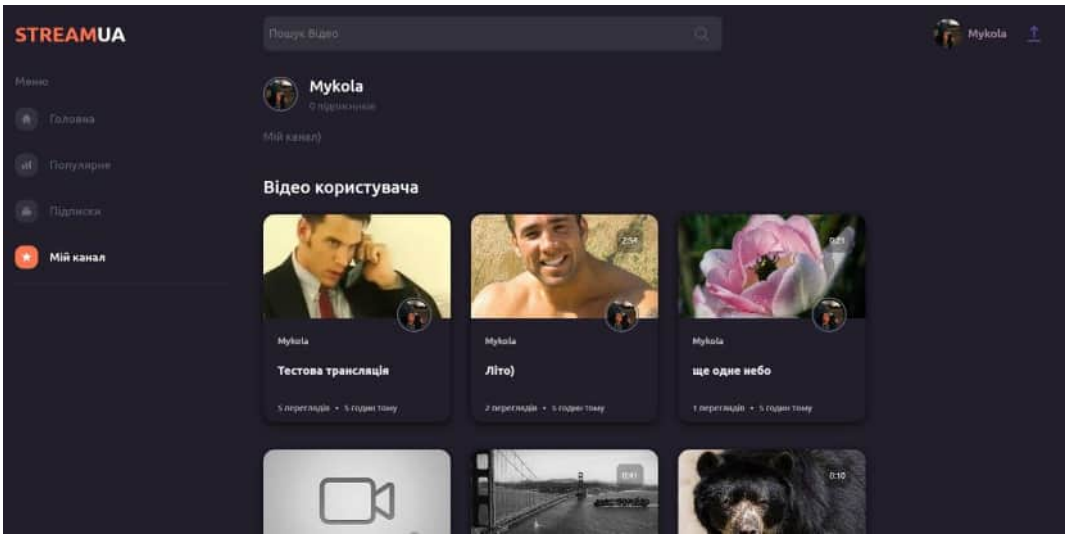


Рисунок 3.3.1.3 – секція “Мій Канал”

На Рисунку 3.3.1.3 видно результат роботи клієнтської частини на секції “Мій Канал”. Видно список відео завантажених цим користувачем, кількість підписників, та інша інформація про канал.

2. Компонент Home(Рисунок 3.3.1.4)

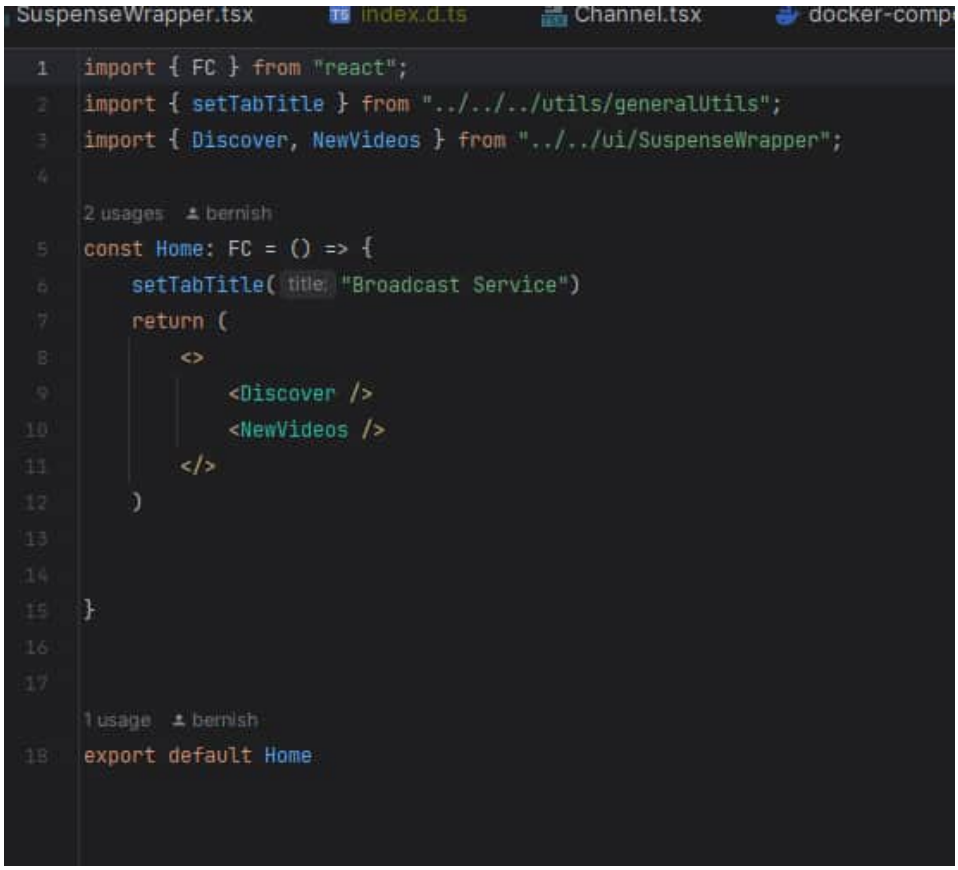


Рисунок 3.3.1.4 – код компоненту home

Цей компонент відповідає за відображення секції “Головна”

У наведеному у Рисунку 3.3.1.4 коді проходить запит до серверної частини на отримання списку найновіших всіх відео.

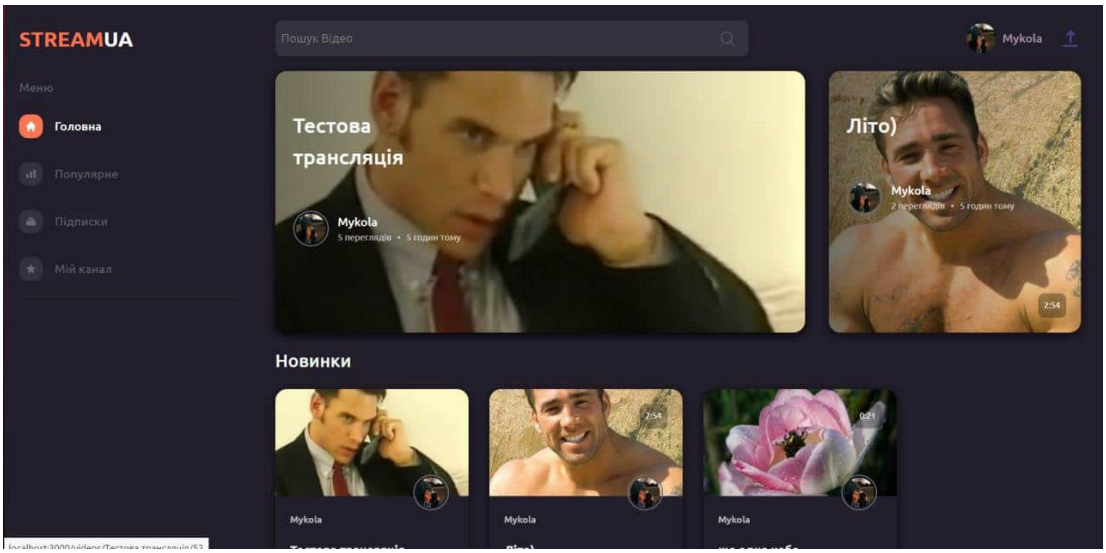


Рисунок 3.3.1.5 – секція “Головна”

На Рисунку 3.3.1.5 видно результат роботи клієнтської частини на секції “Головна”. Видно список найновіших відео та стрімів.

3. Компонент Not-found(Рисунок 3.3.1.6)

```
import { FC } from "react";
import { IoIosLeaf } from 'react-icons/io';
import { setTabTitle } from "../../../utils/generalUtils";
import styles from './NotFound.module.scss';


const NotFoundPage: FC = () => {
    setTabTitle( title: "Not Found")
    return (
        <div className={styles.wrapper}>
            <h2>Sorry. Such page doesn't exist</h2>
            <IoIosLeaf />
        </div>
    )
}


export default NotFoundPage;
```

Рисунок 3.3.1.6 – код компоненту not-found

Цей компонент відповідає за відображення сторінки “Not-Found”

У наведеному у Рисунку 3.3.1.6 коді видно реалізацію при якій коли сторінку не знайдено відображається текст про це(Рисунок 3.3.1.7)

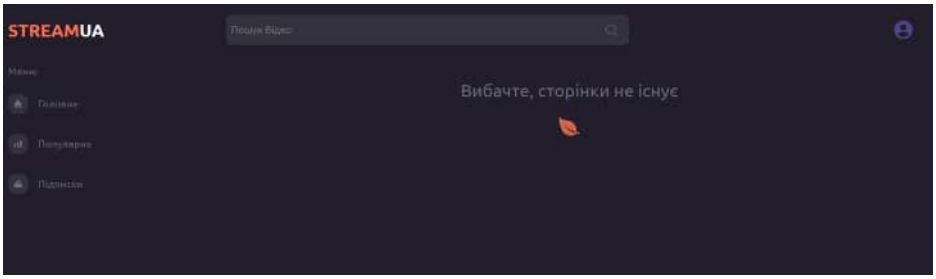


Рисунок 3.3.1.7 – сторінка “Not-Found”

4. Компонент ProfileEdit(Рисунок 3.3.1.8 та Рисунок 3.3.1.9)

```
import { FC, useEffect } from "react";
import { SubmitHandler, useForm } from "react-hook-form";
import { useNavigate, useParams } from "react-router-dom";
import { useActions } from "../../../hooks/useActions";
import { useAuth } from "../../../hooks/useAuth";
import { api } from "../../../store/api/api";
import { setTabTitle } from "../../../utils/generalUtils";
import ProfileEditForm from "../../ui/profile-form/ProfileForm";
import { IProfileEditForm } from "./ProfileEdit.interface";
import styles from "./ProfileEdit.module.scss";

2 usages  bernish
const ProfileEditPage: FC = () => {
  setTabTitle( title: "Edit Profile");
  const { id : string | undefined  } = useParams();
  const { user : {id: number, email: string, ac...  } = useAuth();
  const { data: profile : IUser | undefined  } = api.useGetProfileQuery(user?.id!, {
    skip: !user,
  });
  const navigate : NavigateFunction  = useNavigate();

  const [updateProfile : MutationTrigger<MutationDefini... ] = api.useUpdateProfileMutation();
  const { addMsg : ActionCreatorWithPayload<any, ...  } = useActions();
  const {
    register : UseFormRegister<IProfileEditFo... ,
    formState: { errors : Partial<FieldErrorsImpl<{descr...  },
    handleSubmit : UseFormHandleSubmit<IProfileEd... ,
    control : Control<IProfileEditForm, any> ,
    setValue : UseFormSetValue<IProfileEditFo... ,
  } = useForm<IProfileEditForm>();

  useEffect( effect: () : any  => {
    if (!user) return navigate("/");
    if (user.id !== Number(id)) return navigate("/");
  }, deps: [id]);

  1 usage  bernish
  const onSubmit: SubmitHandler<IProfileEditForm> = (data : IProfileEditForm ) : void  => {
    updateProfile({ data, id: Number(id) })
      .unwrap()
      .then(() =>
        addMsg({ message: "Profile successfully updated", status: 200 })
      );
  };
```

Рисунок 3.3.1.8 – перша частина коду компоненти ProfileEdit

```
};

return (
  <>
    {profile && (
      <section className={styles.container}>
        <ProfileEditForm
          form={{
            register,
            control,
            errors,
            handleSubmit: handleSubmit(onSubmit),
            setValue,
          }}
          title="Редагування профілю"
          buttonTitle="Зберегти"
          fieldsToExclude={{
            email: "email",
            password: "password",
          }}
          defaultValues={{
            name: profile.name,
            description: profile.description,
            avatar: profile.avatarPath,
          }}
        />
      </section>
    )}
  </>
);
};

1 usage  ▲ bernish
export default ProfileEditPage;
```

Рисунок 3.3.1.9 – друга частина коду компоненти ProfileEdit

Цей компонент відповідає за відображення сторінки “Редагування профіль”

У наведеному у Рисунку 3.3.1.8 та Рисунку 3.3.1.9 коді видно реалізацію данної сторінки яка слугує для редагування профілю, і при натисканні кнопочки зберегти – профіль зберігається. Результат роботи компонента(Рисунок 3.3.1.10)

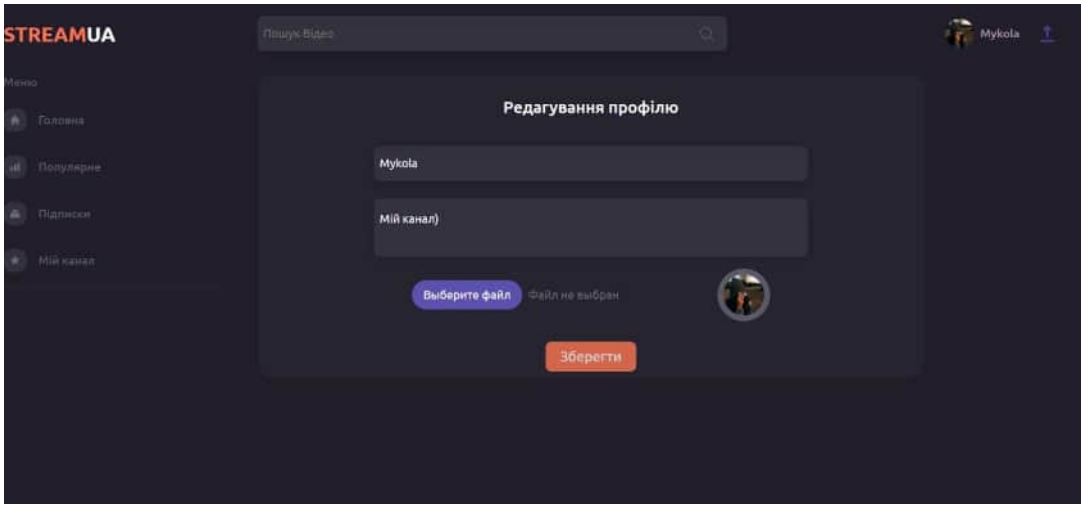


Рисунок 3.3.1.10 – сторінка “Редахування профілю”

5. Компонент Register(Рисунок 3.3.1.11)

```tsx
import { FC, useEffect } from "react";
import { SubmitHandler, useForm } from "react-hook-form";
import { useNavigate } from "react-router-dom";
import { useActions } from "../../../hooks/useActions";
import { useAuth } from "../../../hooks/useAuth";
import { setTabTitle } from "../../../utils/generalUtils";
import RegisterForm from "../../ui/profile-form/ProfileForm";
import { IRegisterForm } from "./Register.interface";
import styles from "./Register.module.scss";

const RegisterPage: FC = () => {
  setTabTitle( title: "Create Account");
  const { user } = useAuth();
  const { register: registerAction } = useActions();
  const navigate = useNavigate();

  useEffect( effect: () => {
    // for redirecting on success (unwrap asyncThunk doesn't work for some reason)
    if (user) navigate("/");
  }, deps: [user]);

  const {
    formState: { errors },
    register,
    control,
    handleSubmit,
    setValue,
  } = useForm<IRegisterForm>( props: {
    mode: "onChange",
  });

  const onSubmit: SubmitHandler<IRegisterForm> = (data) => {
    console.log(data);
    registerAction(data);
  };

  return (
    <section className={styles.container}>
      <RegisterForm
        form={{
          handleSubmit: handleSubmit(onSubmit),
          setValue,
          control,
          register,
          errors,
        }}
        title={"Будь-ласка заповніть всі поля для реєстрації аккаунта"}
        buttonTitle="Створити аккаунт"
      />
    </section>
  );
};

export default RegisterPage;
```

Рисунок 3.3.1.11 –код компоненти Register

Цей компонент відповідає за відображення сторінки “Реєстрації”

У наведеному у Рисунку 3.3.1.11 коді видно реалізацію данної сторінки яка слугує для реєстрації нового користувача. Результат роботи компоненти видно на Рисунку 3.3.1.12

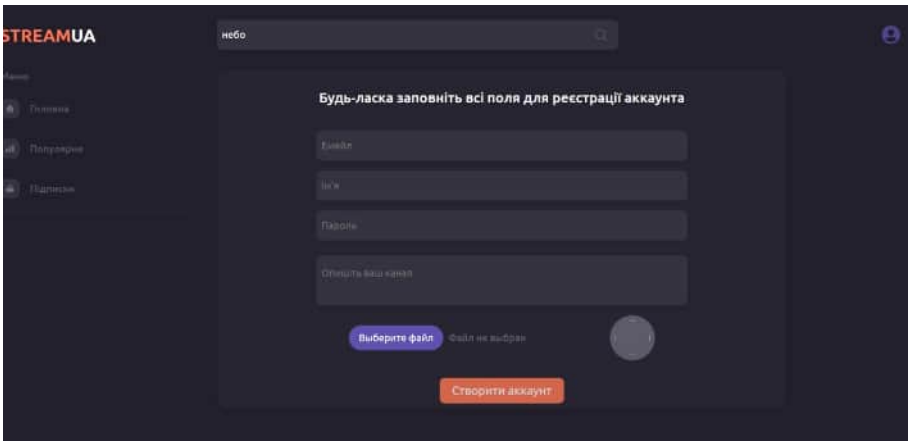


Рисунок 3.3.1.12 – сторінка “Реєстрації”

6. Компонент SearchResults(Рисунок 3.3.1.13)

```tsx
import { FC } from "react";
import { useSearchParams } from "react-router-dom";
import { videoApi } from "../../../store/api/video.api";
import { setTabTitle } from "../../../utils/generalUtils";
import { Catalog } from "../../ui/SuspenseWrapper";

2 usages  bernish
const SearchResult: FC = () => {
  setTabTitle( title: "Search");
  const [searchParams] = useSearchParams();
  const searchTerm : string | null  = searchParams.get("q");

  const { isLoading : boolean , data : IVideo[] | undefined  } = videoApi.useGetBySearchTermQuery(searchTerm!, {
    skip: !searchTerm,
    selectFromResult: ({ data : IVideo[] | undefined , ...rest }) => ({
      data: data?.slice(0, 25),
      ...rest,
    }),
  });

  return (
    <Catalog
      videosToRender={data || []}
      title={`Результати пошуку: ${searchTerm}`}
      isLoading={isLoading}
    />
  );
};

1 usage  bernish
export default SearchResult;
```

Рисунок 3.3.1.13 – код компоненти SearchResults

Цей компонент відповідає за сторінку “Результатів пошуку”

У наведеному у Рисунку 3.3.1.11 коді видно реалізацію данної сторінки яка слугує пошуку та відображення результатів пошуку по відео. Результат роботи компоненти видно на Рисунку 3.3.1.14

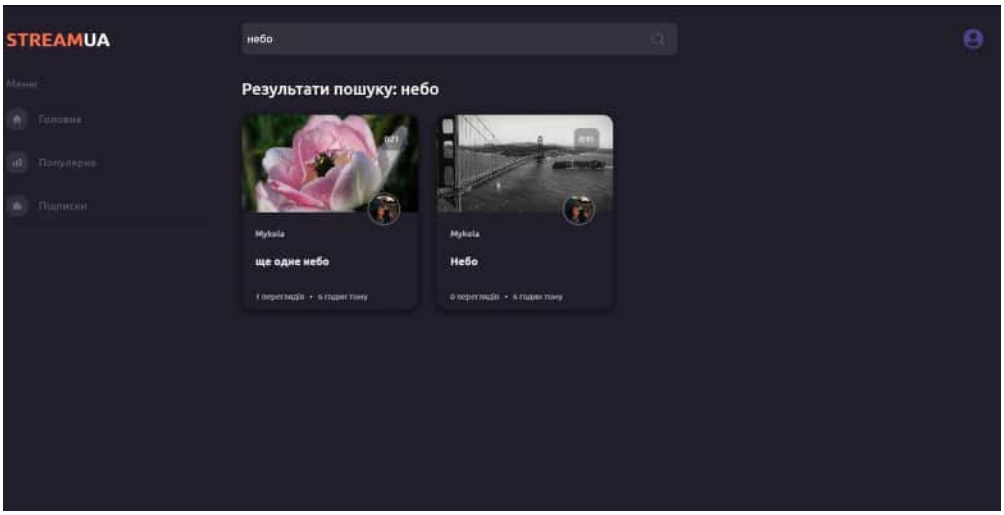


Рисунок 3.3.1.14 – сторінка “Результатів пошуку”

7. Компонент Studio (Рисунок 3.3.1.15)

```tsx
import { FC, useEffect } from "react";
import { useNavigate } from "react-router-dom";
import { useActions } from "../../../hooks/useActions";
import { useAuth } from "../../../hooks/useAuth";
import { api } from "../../../store/api/api";
import { videoApi } from "../../../store/api/video.api";
import { setTabTitle } from "../../../utils/generalUtils";
import { Catalog } from "../../ui/SuspenseWrapper";

const Studio: FC = () => {
  setTabTitle("Studio");
  const { isLoading, user } = useAuth();

  const navigate = useNavigate();
  const { videos } = api.useGetProfileQuery(user?.id!, {
    skip: !user,
    selectFromResult: ({ data }) => ({
      videos: data?.videos,
    }),
  });
  const [deleteVideo] = videoApi.useDeleteMutation();
  const { addMsg } = useActions();

  const handleDelete = (id: number) => {
    deleteVideo(id)
      .unwrap()
      .then(() => addMsg({ message: "Відео було видалено", status: 200 }));
  };

  const handleUpdate = (id: number) => {
    navigate(`/studio/edit/video/${id}`);
  };

  useEffect(() => {
    if (isLoading) return;
    if (!user) {
      navigate("/");
    }
  }, [user, isLoading]);

  return (
    <Catalog
      videosToRender={videos || []}
      title="Мої відео"
      removeHandler={handleDelete}
      updateHandler={handleUpdate}
      isLoading={isLoading}
    />
  );
};

export default Studio;
```

Рисунок 3.3.1.15 – код компоненти Studio

Цей компонент відповідає за сторінку “Студія”

У наведеному у Рисунку 3.3.1.15 коді видно реалізацію данної сторінки яка відображає всі відео які завантажив сам користувач, навіть приватні. Результат роботи компоненти видно на Рисунку 3.3.1.16

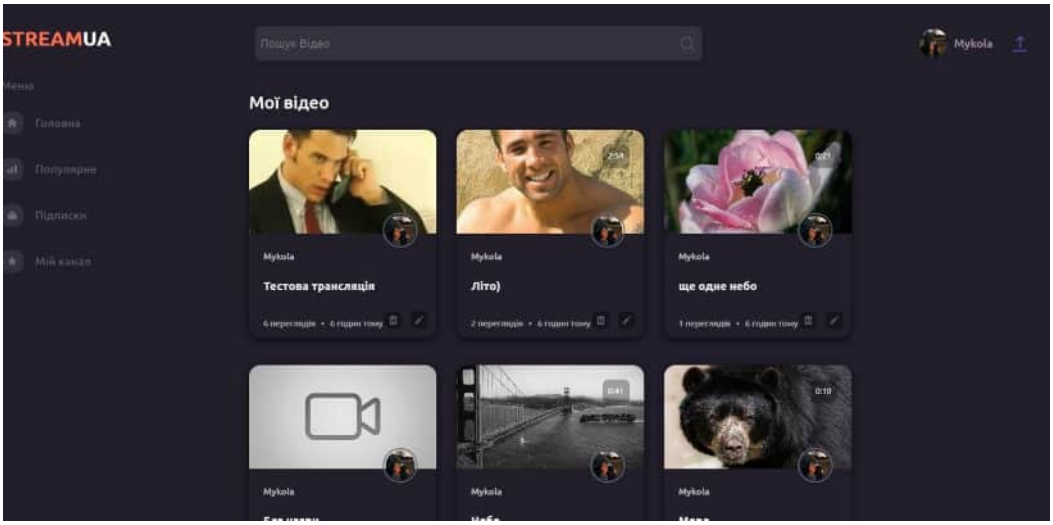
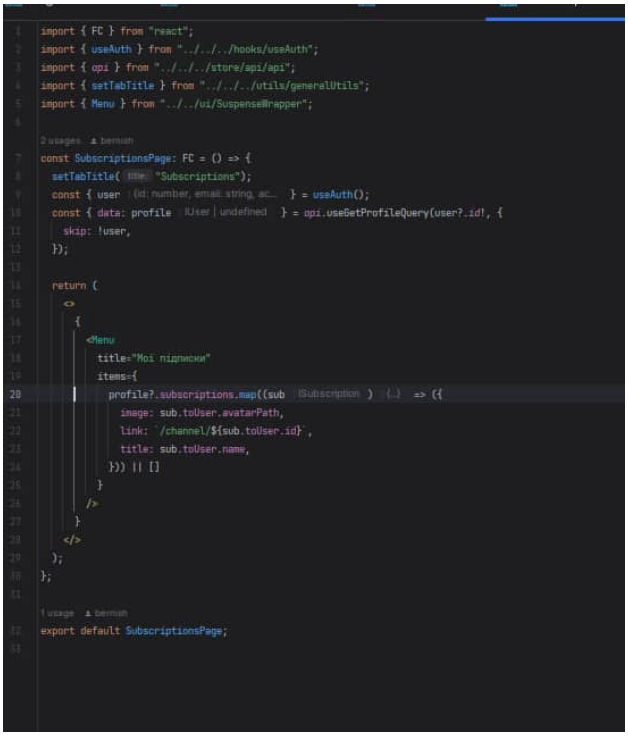


Рисунок 3.3.1.16 – сторінка “Студія”

8. Компонент SubscriptionsPage (Рисунок 3.3.1.17)



```
import { FC } from "react";
import { useAuth } from "../../../hooks/useAuth";
import { api } from "../../../store/api/api";
import { setTabTitle } from "../../../utils/generalUtils";
import { Menu } from "../../ui/SuspenseWrapper";

2 usages  ▲ bernish
const SubscriptionsPage: FC = () => {
  setTabTitle("Subscriptions");
  const { user } = useAuth();
  const { data: profile } = api.useGetProfileQuery(user?.id!, {
    skip: !user,
  });

  return (
    <>
      {
        <Menu
          title="Мої підписки"
          items={
            profile?.subscriptions.map((sub) => ({
              image: sub.toUser.avatarPath,
              link: `/channel/${sub.toUser.id}`,
              title: sub.toUser.name,
            })) || []
          }
        />
      }
    </>
  );
};

1 usage  ▲ bernish
export default SubscriptionsPage;
```

Рисунок 3.3.1.17 – код компоненти SubscriptionsPage

Цей компонент відповідає за сторінку “Мої підписки”

У наведеному у Рисунку 3.3.1.17 коді видно реалізацію данної сторінки яка відображає всі мої підписки по яким можна перейти. Результат роботи компоненти видно на Рисунку 3.3.1.18

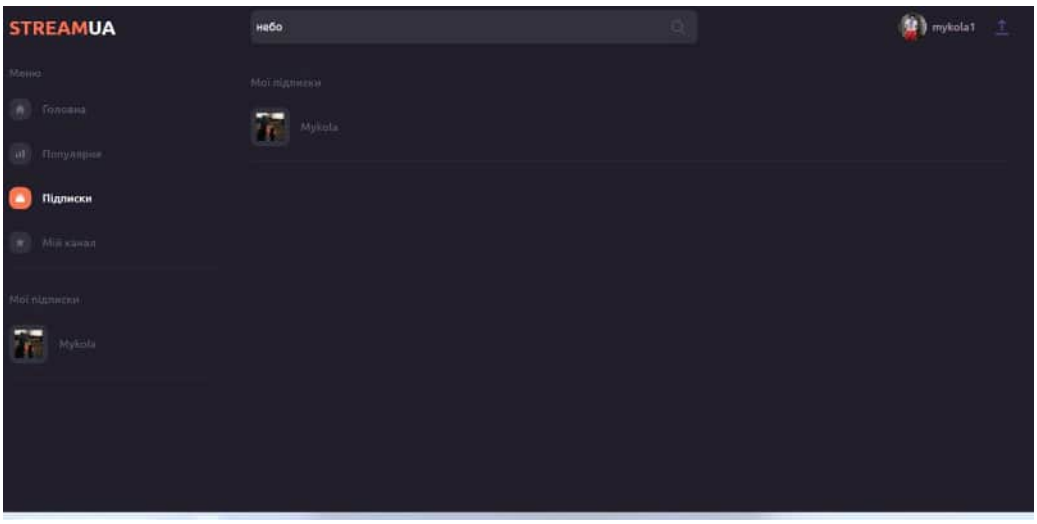


Рисунок 3.3.1.18 – сторінка “ Мої підписки”

9. Компонента Trending (Рисунок 3.3.1.19)

```tsx
import { AxiosError } from "axios";
import { FC } from "react";
import { useQuery } from "react-query";
import { useActions } from "../../../hooks/useActions";
import { VideoService } from "../../../services/video/video.service";
import { IVideo } from "../../../types/video.interface";
import { setTabTitle } from "../../../utils/generalUtils";
import { Catalog } from "../../ui/SuspenseWrapper";


const Trending: FC = () => {
  setTabTitle( title: "Popular");
  const { addMsg } : ActionCreatorWithPayload<any, ...  } = useActions();

  const {
    error : AxiosError<?, any> | null ,
    isLoading : boolean ,
    isError : boolean ,
    data: videos : IVideo[] | undefined ,
  } = useQuery<IVideo[], AxiosError>(
    queryKey: "Videos_query",
    VideoService.getMostViewed
  );

  if (isError) addMsg({ message: error.message, status: 500 });

  return (
    <Catalog
      title="Популярне"
      videosToRender={videos || []}
      isLoading={isLoading}
    />
  );
};


export default Trending;
```

Рисунок 3.3.1.17 – код компоненти Trending

Цей компонент відповідає за сторінку “Популярне”

У наведеному у Рисунку 3.3.1.17 коді видно реалізацію данної сторінки яка відображає найпопулярніші відео. Результат роботи компоненти видно на Рисунку 3.3.1.18

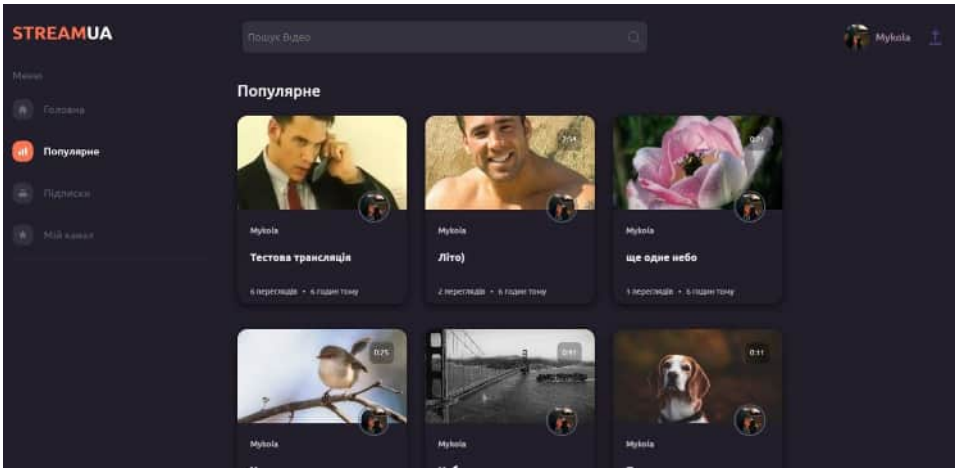


Рисунок 3.3.1.18 – сторінка “ Популярне”

10. Компонент Video (Рисунок 3.3.1.19 та Рисинок 3.3.1.20)

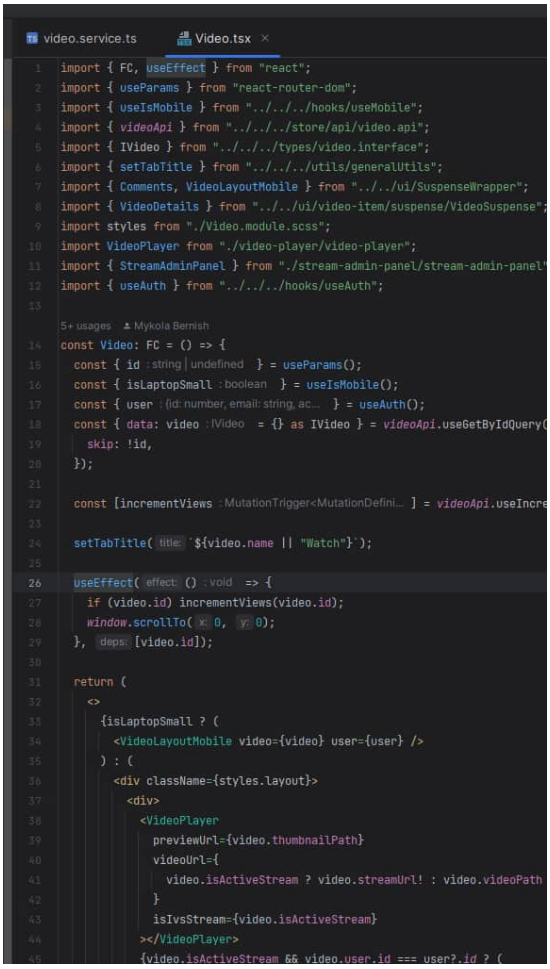


Рисунок 3.3.1.19 – перша частина коду компоненти Video

```
            isIvsStream={video.isActiveStream}
          ></VideoPlayer>
          {video.isActiveStream && video.user.id === user?.id ? (
            <StreamAdminPanel
              streamKey={video.streamKey!}
              streamIngest={video.streamIngest!}
              videoId={video.id}
            ></StreamAdminPanel>
          ) : (
            ""
          )}
          <VideoDetails {...video} />
        </div>

        <Comments videoId={video.id} comments={video.comments || {}} />
      </div>
    )}
  </>
 );
};

1 usage  Mykola Bernish
export default Video;
```

Рисунок 3.3.1.20 – друга частина коду компоненти Video

Цей компонент відповідає за сторінку “Відео”

У наведеному у Рисунку 3.3.1.17 коді видно реалізацію данної сторінки яка сторінку відео де є плеєр для перегляду відео контенту, можливість залишати коментар, ставити вподобайку, підписуватись та інше. Результат роботи компоненти видно на Рисунку 3.3.1.21

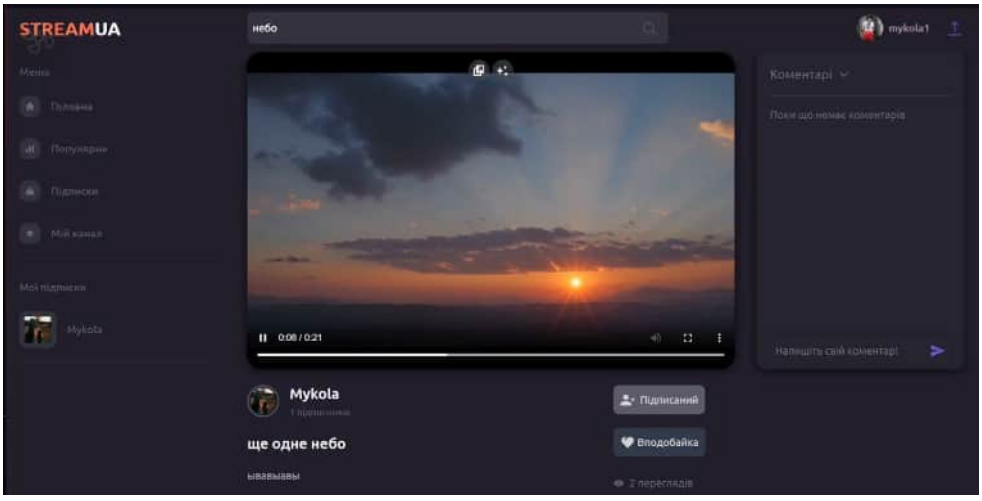


Рисунок 3.3.1.21 – сторінка “ Відео”

Ми розглянули основні компоненти та основний функціонал клієнтської частини.

3.3.2 Інтеграція з AWS IVS для стрімінгу відео в режимі реального часу

Для забезпечення функціональності стрімінгу відео в режимі реального часу можна використати Amazon Interactive Video Service (IVS) - сервіс, який надає можливість використовувати потікове відео для трансляцій, живих подій та інших відеосценаріїв. Інтеграція з AWS IVS дозволить надати користувачам платформи можливість переглядати відеоматеріали в режимі реального часу без необхідності чекати завантаження відеофайлу.

Для початку інтеграції з AWS IVS потрібно створити канал трансляції в AWS IVS. Канал трансляції - це вихідний потік відео, який можна стрімити на платформі. Кожен такий канал привязується до відповідного каналу на нашому сервісі що дозволяє проводити відеотрансляцію.

Після створення каналу трансляції необхідно налаштувати підключення до нього з клієнтської частини платформи. Для цього можна використати AWS SDK або бібліотеки для роботи з AWS IVS, які надаються Amazon. Ці бібліотеки дозволяють отримувати доступ до потоку відео та управляти його відтворенням. Ці налаштування відбуваються “під капотом” на серверній частині під час запуску трансляції.

У клієнтській частині платформи необхідно створити відеоплеєр, який буде відображати стрімоване відео з AWS IVS. Для цього можна використати HTML5-відеоплеєр або спеціалізовані бібліотеки, такі як IVS Player SDK, які надаються Amazon. Ці бібліотеки забезпечують інтеграцію з AWS IVS та дозволяють легко відтворювати стрімоване відео в режимі реального часу у веб-браузері.

```tsx
import ...
1 usage  bernish
interface VideoPlayerProps {
  previewUrl: string;
  videoUrl: string;
  isIvsStream?: boolean;
}

5+ usages  bernish
const VideoPlayer: React.FC<VideoPlayerProps> = ({
  previewUrl : string ,
  videoUrl : string ,
  isIvsStream : boolean  = false,
}) => {
  const videoRef : React.RefObject<HTMLVideoEleme...  = useRef<HTMLVideoElement>( initialValue: null);

  useEffect( effect: () => {
    let player: MediaPlayer;

    2 usages  bernish
    const createAbsolutePath = (assetPath: string) =>
      new URL(assetPath, document.URL).toString();

    if (isIvsStream && videoRef.current && isPlayerSupported) {
      try {
        const worker : string  = createAbsolutePath( assetPath: "/amazon-ivs-wasmworker.min.js");
        const binary : string  = createAbsolutePath( assetPath: "/amazon-ivs-wasmworker.min.wasm");
        player = create({
          wasmWorker: worker,
          wasmBinary: binary,
        });

        player.attachHTMLVideoElement(videoRef.current!);
        attachListeners(player);

        player.setAutoplay(true);
        player.load(videoUrl);
      } catch (error) {
        console.error("IVS Player error:", error);
      }
    }
```

Рисунок 3.3.2.1 – перша частина коду компоненти відеоплеєра і з підтримкою роботи IVS стрімів

```tsx
    }

    return () : void  => {
      if (player) {
        player.delete();
      }
    };
  }, deps: [isIvsStream, videoUrl]);

  1 usage  bernish
  const attachListeners = (player: MediaPlayer) : void  => {
    for (let state of Object.values(PlayerState)) {
      player.addEventListener(state, () : void  => {
        console.log(state);
      });
    }

    player.addEventListener(PlayerEventType.INITIALIZED, () : void  => {
      console.log("INITIALIZED");
    });

    player.addEventListener(PlayerEventType.ERROR, (error: PlayerError) : void  => {
      const statusTooManyRequests : 429  = 429;
      if (
        error.type === ErrorType.NOT_AVAILABLE &&
        error.code === statusTooManyRequests
      ) {
        console.error("Concurrent-viewer limit reached", error);
      } else {
        console.error("ERROR", error);
      }
    });

    player.addEventListener(
      PlayerEventType.QUALITY_CHANGED,
      (quality: Quality) : void  => {
        console.log("QUALITY_CHANGED", quality);
      }
    );

    player.addEventListener(PlayerEventType.TEXT_CUE, (cue: TextCue) : void  => {
      console.log("TEXT_CUE", cue.startTime, cue.text);
    });
```

Рисунок 3.3.2.2 – друга частина коду компоненти відеоплеєра і з підтримкою роботи IVS стрімів

```
    player.addEventListener(PlayerEventType.TEXT_CUE, (cue: TextCue) => {
      console.log("TEXT_CUE", cue.startTime, cue.text);
    });

    player.addEventListener(
      PlayerEventType.TEXT_METADATA_CUE,
      (cue: TextMetadataCue) => {
        console.log("Timed metadata", cue.text);
      }
    );
  };

  return (
    <div className={styles.wrapper}>
      {isIvsStream ? (
        <video
          ref={videoRef}
          poster={previewUrl || mockups.defaultThumbnail}
          controls
          className={styles.player}
        />
      ) : (
        <video
          src={videoUrl}
          poster={previewUrl || mockups.defaultThumbnail}
          controls
          className={styles.player}
        />
      )}
    </div>
  );
};

5+ usages  bernish
export default VideoPlayer;
```

Рисунок 3.3.2.3 – третя частина коду компоненти відеоплеєра і з підтримкою роботи IVS стрімів

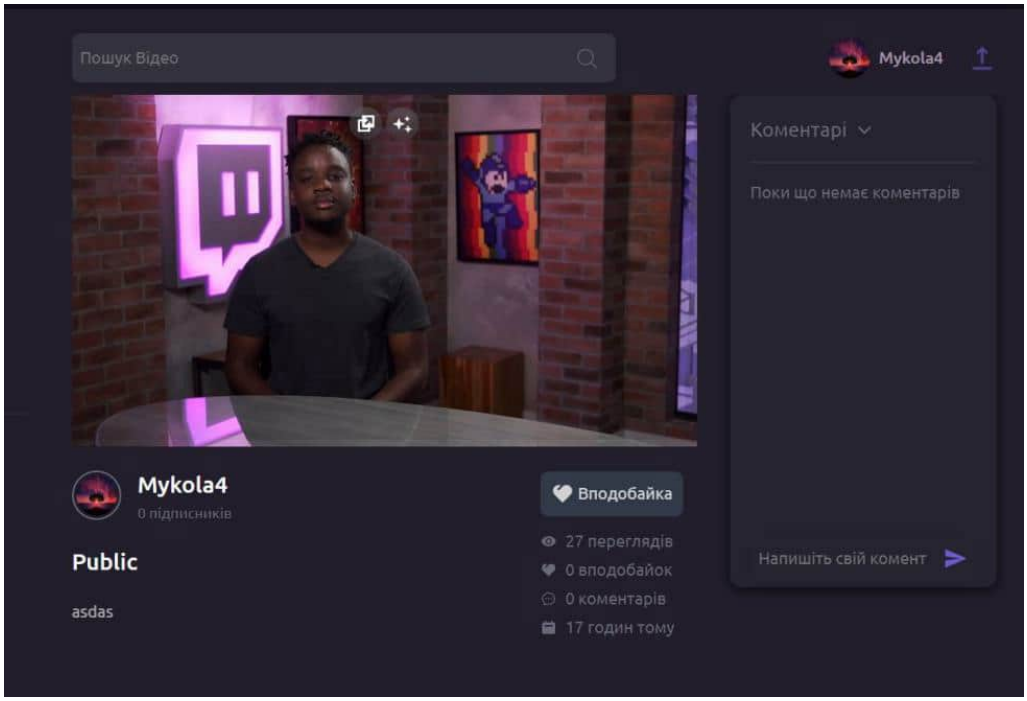


Рисунок 3.3.2.4 – результат відображення прямої трансляції відомого в вузьких кругах стрімера

ВИСНОВКИ

У процесі розробки платформи для хостингу та перегляду відеоматеріалів з використанням технологій Node.js, Nest.js та Amazon Web Services (AWS) було досягнуто наступні результати:

1. Було вибрано відповідні технології, які забезпечують швидку та ефективну розробку платформи з врахуванням потреб користувачів.
2. Розроблено архітектуру, яка дозволяє забезпечити масштабованість та високу доступність платформи для великої кількості користувачів.
3. Застосування хмарних сервісів AWS, таких як S3 та IVS, дозволяє зберігати та стрімити відео в режимі реального часу з високою якістю та швидкістю передачі.

В процесі реалізації платформи було виявлено наступні переваги використовуваних технологій та засобів:

Node.js та Nest.js забезпечують ефективну розробку серверної логіки, а також підтримують асинхронну обробку запитів, що сприяє високій продуктивності системи.

Використання хмарних сервісів Amazon Web Services (AWS) надає гнучку та масштабовану інфраструктуру для розгортання та управління платформою, а також широкий спектр сервісів для зберігання та обробки медіаконтенту.

Використання сервісу AWS S3 дозволяє ефективно зберігати та керувати великим обсягом відеофайлів, а AWS IVS забезпечує надійний та масштабований стрімінг відео в режимі реального часу.

Розроблена архітектура платформи для хостингу та перегляду відеоматеріалів відповідає поставленим вимогам та цілям проекту. Вона забезпечує швидку та зручну навігацію для користувачів, масштабованість системи, безпеку та контроль доступу до контенту.

Розроблена платформа має потенціал для подальшого розширення та вдосконалення. Застосування інструментів та технологій, таких як TypeORM,

AWS Cognito та Serverless Framework, дозволяє легко впроваджувати нові функціональні можливості та вдосконалювати платформу з плинністю.

Загалом, розробка платформи для хостингу та перегляду відеоматеріалів з використанням сучасних технологій та засобів дозволила створити функціональну, ефективну та масштабовану систему, яка задовольняє потреби користувачів у перегляді відео та забезпечує зручну та безпечну навігацію.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Відеохостинг rutube так і не «піднявся» після атаки хакерів 9 травня URL: https://www.ukrinform.ua/rubric-technology/3479538-videohosting-rutube-tak-i-ne-pidnavsa-pisla-ataki-hakeriv-9-travna.html (дата звернення 05.01.2023)
2. Rutube все? Як IT-army of Ukraine веде боротьбу з агресором URL: https://www.imena.ua/blog/rutube-not-working/ (дата звернення 05.01.2023)
3. Як зміниться світ після карантину на думку Viber URL: https://ua.interfax.com.ua/news/blog/661537.html
4. Відеохостинг youtube як засіб інформаційно-культурної соціалізації дітей молодшого шкільного віку. URL:https://www.researchgate.net/publication/349662140_VIDEOHOSTING_YOUTUBE_AK_ZASIB_INFORMACIJNO-KULTURNOI_SOCIALIZACII_DITEJ_MOLODSOGO_SKILNOGO_VIKU (дата звернення: 03.03.2023)
5. Відеохостинги як перспективна платформа для інтеграції українських телеканалів URL: http://www.scientific-notes.com/wp-content/uploads/2020/07/70_7.pdf (дата звернення: 07.03.2023)
6. Introdution to Node Introdution to Node.js . URL: https://nodejs.dev/en/learn/ (дата звернення: 10.03.2023)
7. Documentation | NestJS – A progressive Node.js framework. URL: https://docs.nestjs.com/ (дата звернення 11.03.2023)
8. Кристиан, Хорсдал Мікросервіси на платформі NestJS. К.: Пітер, 2018. - 462 с.
9. What is AWS? URL: https://aws.amazon.com/ru/what-is-aws/ (дата звернення 15.04.2023)
10. Amazon S3 URL: https://aws.amazon.com/ru/s3/ (дата звернення 15.04.2023)

11. Amazon S3 Userguide URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (дата звернення 15.04.2023)
12. Amazon IVS URL: https://aws.amazon.com/ru/ivs/ (дата звернення 20.05.2023)
13. 8 uses cases of Amazon Interactive Video Service URL: https://www.rst.software/blog/8-uses-cases-of-amazon-interactive-video-service-amazon-ivs (дата звернення 01.06.2023)
14. Nigel Poulton Deep Dive Version 4 October 2017 – DockerCon EU edition. - 18 с.