Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Програмно-технічна система обробки, зберігання та передачі відеоконтенту (серверна частина)
Назва теми

КвРКІ 200105.20.01.04 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-20-1 _____ О.В. Волошин
Підпис Ініціали, прізвище

Керівник _____ В.М. Грига
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ І.О. Засорнова
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«14» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 10 " 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Волошину Олегу Вікторовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмно-технічна система обробки, зберігання та передачі відеоконтенту (серверна частина)

Керівник проекту (роботи) Грига В.М., д.т.н., професор

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Огляд, аналіз та порівняння серверної частини існуючих сервісів для роботи з відеоконтентом

Архітектура системи та вибір програмних та апаратних засобів розробки

Програмно-технічна реалізація серверної частини системи обробки, зберігання та передачі відеоконтенту

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схема архітектури програмно-технічної системи обробки, зберігання та передачі відеоконтенту

Блок-схеми алгоритмів реєстрації та аутентифікації користувача

Блок-схема роботи алгоритму обробки відео прев'ю, мініатюр та якостей

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І.О., доцент кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – огляд, аналіз та порівняння серверної частини існуючих сервісів для роботи з відеоконтентом	01.03.2024	виконано
4	Робота над розділом 2 –опис архітектури та вибір програмних та апаратних засобів розробки	01.04.2024	виконано
5	Робота над розділом 3 – програмно-технічна реалізація серверної частини системи обробки, зберігання та передачі відеоконтенту	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	30.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ О. В. Волошин
Підпис Ініціали, прізвище

Керівник роботи _____ В. М. Грига
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 200105.20.01.04 ПЗ	Пояснювальна записка	71		
			Графічні матеріали			
2		КвРКІ 200105.20.01.04 Е8	Схема архітектури програмно-технічної системи обробки, зберігання та передачі відеоконтенту	1		
3		КвРКІ 200105.20.01.04 Е8	Блок-схеми алгоритмів реєстрації та аутентифікації користувача	1		
4		КвРКІ 200105.20.01.04 Е8	Блок-схема роботи алгоритму обробки відео прев'ю, мініатюр та якостей	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 200105.20.01.04 ВП		
Розробив		Волошин			Відомість проекту	Літера	Аркуш 1 / Аркушів 1
Перевір.		Грига				У	
Н. контр.		Засорнова				ХНУ, КІ2-20-1	
Затв.		Говорущенко		14.06			

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-технічна система обробки, зберігання та передачі відеоконтенту (серверна частина)».

Автор роботи: Волошин Олег Вікторович.

Керівник роботи: Грига Володимир Михайлович.

Пояснювальна записка: 71 с., 33 рис., 7 табл., 4 дод., 55 джерел.

Графічна частина: 3 креслення.

ПРОГРАМНО-ТЕХНІЧНА СИСТЕМА, ОБРОБКА ВІДЕО, ПЕРЕДАЧА ВІДЕО, АРХІТЕКТУРА, БАЗИ ДАНИХ, БРОКЕРИ ПОВІДОМЛЕНЬ, КОНТЕЙНЕРИЗАЦІЯ.

Метою даної дипломної роботи є розробка серверної частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту та забезпечення достатньої ефективності в її функціонуванні.

Об'єктом дослідження є функціонування серверної частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту.

Предметом дослідження є оцінка режимів застосування цієї системи.

Під час проведення даного дослідження був використаний метод систематичного огляду літератури для вивчення і аналізу предметної області даного дослідження з текстових джерел інформації.

Підпис студента

30.05.2024

Дата

ЗМІСТ

ВСТУП

В сучасному світі, де технологічний прогрес стрімко розвивається, автоматизація та інформаційні технології стають основними компонентами розвитку. Цей процес охоплює різні галузі, включаючи науку, промисловість та освіту. Спостерігається тенденція до використання цифрової обчислювальної техніки, що значно полегшує та прискорює багато процесів.

Зокрема, у сфері обробки відеоконтенту ці технологічні досягнення мають велике значення. Відеоконтент стає все більш популярним і використовується в різних галузях, починаючи від розваг та медіа, закінчуючи бізнес-потребами та навчанням. Тому розуміння та використання передових технологій обробки відео є важливим завданням для фахівців у цій області.

Метою даного дослідження є систематизація знань та практичний досвід у сфері обробки відеоконтенту. Важливими завданнями є вивчення та аналіз сучасних методів обробки відеоданих, вибір оптимальних технологій та інструментів для цього, а також розробка та впровадження ефективних рішень у практичну діяльність.

Дослідження також спрямоване на набуття необхідних навичок у проектуванні цифрових систем обробки відео, що стане важливим додатковим етапом у підвищенні кваліфікації та розвитку професійних навичок учасників проєкту. Такий підхід дозволить забезпечити високу якість обробки відеоконтенту та підвищити ефективність роботи у різних сферах використання відеопродукції.

1 ОГЛЯД, АНАЛІЗ ТА ПОРІВНЯННЯ СЕРВЕРНОЇ ЧАСТИНИ ІСНУЮЧИХ СЕРВІСІВ ДЛЯ РОБОТИ З ВІДЕОКОНТЕНТОМ

1.1 Опис технологій зберігання та обробки відео

1.1.1 Використання хмарних сервісів

Використання хмарних сервісів для зберігання та обробки відео набуває все більшої популярності завдяки зростанню обсягів відеоданих та потреби в їх ефективному управлінні та обробці.

Хмарні сервіси дозволяють зберігати великі обсяги відеофайлів без потреби великих фізичних серверних інфраструктур. Користувачі можуть зберігати свої відеодані у безпечному та надійному середовищі з можливістю доступу з будь-якої точки світу за допомогою мережі Інтернет [1].

Хмарні сервіси можуть легко масштабуватись в залежності від потреб користувача. Вони можуть автоматично розширюватись для забезпечення потрібних ресурсів для зберігання та обробки великих обсягів відеоданих. Відеодані, збережені в хмарі, можуть бути доступні з будь-якого пристрою (комп'ютера, планшета, смартфона) з підключенням до мережі Інтернет, що дозволяє користувачам зручно працювати з відео навіть під час пересування.

Багатокористувацькі можливості хмарних сервісів дозволяють різним користувачам працювати з одними та тими ж відеоданими одночасно, спільно редагуючи їх та спільно працюючи над проектами. Деякі хмарні сервіси надають інструменти для обробки відео безпосередньо в хмарному середовищі. Це може включати в себе можливості редагування, конвертації форматів, створення зразків тощо.

Хмарні сервіси можуть надати можливості автоматизації певних процесів обробки відео, таких як індексація, аналіз вмісту, виявлення об'єктів тощо за допомогою штучного інтелекту та машинного навчання. Використання хмарних сервісів для зберігання та обробки відеоданих дозволяє підвищити рівень безпеки

шляхом використання різноманітних механізмів шифрування, автентифікації та контролю доступу.

1.1.2 Кодеки, формати та транскодинг відео

Кодеки (від "кодування/декодування") та формати відео визначають спосіб стиснення та кодування відеоданих, що дозволяє зберігати та передавати їх у компактній формі з меншим обсягом даних [2].

Найпоширеніші кодеки:

1. H.264 (AVC) – один з найпоширеніших стандартів стиснення відео, який використовується для відеостандартів, таких як Blu-ray, YouTube, історія телебачення HDTV.

2. H.265 (HEVC) – покращена версія H.264, яка забезпечує кращу якість при зниженні обсягу даних. HEVC дозволяє зменшити обсяг файлу приблизно на 50% порівняно з H.264.

3. VP9 – розроблений Google, цей кодек стиснення відео використовується у веб-стрімінгу на платформі YouTube та інших веб-сайтах.

4. AV1 – відкритий стандарт стиснення відео, розроблений Альянсом Open Media. AV1 забезпечує високу якість відео при мінімальному обсязі даних.

5. MPEG-2 – широко використовується для стандартного телебачення, DVD та деяких відеоігор.

Найпоширені формати відео:

1. MP4 (MPEG-4 Part 14) – один з найпопулярніших форматів відео, який підтримує різні кодеки, включаючи H.264 та H.265.

2. MKV (Matroska Multimedia Container) – відкритий формат контейнера, який може містити відео, аудіо, субтитри та інші дані в одному файлі.

3. AVI (Audio Video Interleave) – класичний формат відео для Windows, який підтримує різні кодеки, але має обмежені можливості порівняно з сучасними форматами.

4. MOV – формат QuickTime, розроблений Apple, який підтримує відео, аудіо та інші мультимедійні дані.

5. WebM – відкритий формат, розроблений Google, який використовує контейнер Matroska та кодек VP8 або VP9 для відео.

Транскодинг відео - це процес перетворення відеофайлу з одного формату чи кодеку у інший. Це може бути необхідно з різних причин, таких як забезпечення сумісності з пристроями чи програмним забезпеченням, зменшення розміру файлу, покращення якості або забезпечення оптимального потоку даних для мережевого відтворення.

FFmpeg є одним з найпопулярніших інструментів для транскодування відео та обробки мультимедійних даних. Він є відкритим програмним забезпеченням і має великий набір функцій для обробки аудіо та відео. FFmpeg може бути використаний для зчитування, запису, конвертації, обрізання, зміни розміру, зміни швидкості, додавання ефектів і фільтрів, побудови потоків і багато іншого.

Основні можливості FFmpeg:

1. Конвертація форматів – можна конвертувати відео з одного формату в інший, наприклад, з AVI в MP4 або з MOV в FLV.

2. Зміна кодеків – можна змінювати кодеки відео та аудіо, що дозволяє зменшувати розмір файлів або покращувати якість відео.

3. Обрізання і обробка – можна обрізати відео, видаляти або додавати аудіо-доріжки, застосовувати фільтри, ефекти та багато іншого.

1.2 Трансляція та CDN

Трансляція відео та мережа доставки контенту (Content Delivery Network – CDN) є двома важливими складовими для ефективної та надійної доставки відео в онлайн-середовищі.

Трансляція відео - це процес надання можливості перегляду відеоконтенту в реальному часі через Інтернет. Наприклад, онлайн-траснляції вебінарів, спортивних змагань, концертів, геймингу та тому подібне [3].

Основні складові трансляції відео включають в себе:

1. Кодування та стискання відео – відео потрібно закодувати та стиснути, щоб зменшити обсяг даних та забезпечити плавний стрімінг для глядачів.

2. Сервери для розподілення відео – ці сервери відповідають за надання відеоданих користувачам та їх доставку через Інтернет.

3. Платформи трансляції – це програмне забезпечення або платформи, які дозволяють транслювати відео в реальному часі, надають можливості взаємодії з аудиторією та контролюють доступ до відеоконтенту.

CDN – це розподілені серверні мережі, які призначені для швидкої та ефективної доставки контенту до кінцевих користувачів. CDN має ряд переваг, такі як:

1. Швидка доставка контенту – забезпечує швидку доставку контенту завдяки розміщенню серверів у близькості до кінцевих користувачів. Це зменшує час завантаження сторінок, відео, зображень та інших мультимедійних елементів.

2. Зменшення навантаження на початковий сервер – коли використовується CDN, частина запитів на контент обробляється серверами CDN, що зменшує навантаження на ваш початковий сервер. Це може покращити продуктивність та доступність відповідного веб-сайту або додатку.

3. Зменшення витрат на пропускну здатність – може зменшити витрати на пропускну здатність, оскільки він може кешувати контент і обслуговувати його

безпосередньо з серверів CDN, що дозволяє зменшити обсяг трафіку, який потрібно передавати через основний сервер.

4. Збільшення надійності – CDN може покращити надійність вашого веб-сайту або додатку, оскільки він може автоматично маршрутизувати трафік через доступні сервери в разі виникнення проблем на одному з них.

5. Захист від DDoS-атак – CDN може служити проксі для вашого веб-сайту, що дозволяє фільтрувати або відхиляти трафік, пов'язаний з DDoS-атаками, що допомагає зберегти доступність вашого сайту.

6. Підтримка HTTPS і безпека – більшість CDN підтримують HTTPS, що дозволяє зашифровано передавати контент користувачам. Вони також можуть надавати додаткові заходи безпеки, такі як виявлення загроз та захист від зловмисних атак.

7. Географічна гнучкість – CDN дає можливість керувати тим, як ваш контент кешується та доставляється у різних регіонах, що дозволяє забезпечити оптимальне відтворення контенту в різних частинах світу.

Багато провайдерів трансляцій відео використовують CDN для оптимізації доставки свого відеоконтенту. Це допомагає забезпечити швидку та стабільну трансляцію в реальному часі, навіть при великому обсязі аудиторії. Комбінація трансляції відео та CDN забезпечує ефективну та надійну трансляцію подій у реальному часі через Інтернет.

1.3 Порівняння аналогів за характеристиками, пов'язаними з серверною частиною

1.3.1 Швидкість завантаження, обробки та передачі відео

Будемо порівнювати такі платформи, як: YouTube, Vimeo, Dailymotion і Facebook Watch.

Розпочнемо з YouTube (рисунок 1.1) – це онлайн-відеоплатформа, що дозволяє користувачам переглядати, завантажувати, коментувати та ділитися відео. Заснована в 2005 році трьома колишніми співробітниками PayPal, вона швидко стала однією з найпопулярніших веб-сайтів у світі [4].

Рисунок 1.1 – Логотип відеоплатформи YouTube

YouTube має різноманітність контенту – від відеоблогів та музичних кліпів до навчальних матеріалів та розважального контенту. Платформа також надає можливість монетизації для створювачів відео через програму партнерства YouTube, яка дозволяє заробляти на рекламі та інших джерелах доходу. YouTube є важливим культурним та соціальним явищем, яке впливає на медіа, розваги та освіту у всьому світі.

YouTube має велику швидкість завантаження відео, завдяки розгалуженій мережі серверів по всьому світу, відео оброблюються для різних форматів та якостей та ефективно передає відео, завдяки широкосмуговому з'єднанню та оптимізованим протоколам передачі контенту.

Vimeo (рисунок 1.2) – це популярна відеоплатформа, яка дозволяє користувачам завантажувати, переглядати та ділитися відеоконтентом. Вона була заснована в 2004 році і відрізняється від YouTube своїм фокусом на якості та творчості.

Vimeo відомий своєю спрямованістю на професійний контент, включаючи короткометражні фільми, музичні відео, реклами та інші креативні проекти.

Платформа також пропонує інструменти для створення відео, спільноту творчих людей та можливості співпраці. Відмінність Vimeo від YouTube полягає у його спрямованості на якість та творчість, що робить його популярним серед професіоналів у сфері відеопродукції.

Рисунок 1.2 – Логотип відеоплатформи Vimeo

Vimeo також має хорошу швидкість завантаження, завдяки своїм серверам, але може бути трохи повільнішим у порівнянні з YouTube. Відео також оброблюються відносно швидко, проте на це може впливати менша кількість користувачів. Передача відео також є ефективною, але є менш оптимізованою в порівнянні з YouTube.

Facebook Watch (рисунок 1.3) – це сервіс відеострімінгу, що запускається на платформі Facebook. Він був представлений у 2017 році і став спробою Facebook відзначитися на ринку відеоконтенту та конкурувати з іншими провайдерами стрімінгового відео, такими як YouTube, Netflix та Amazon Prime Video [3].

Facebook Watch пропонує широкий спектр відеоконтенту, включаючи оригінальні серіали, реаліті-шоу, спортивні трансляції, новини та багато іншого. Крім того, користувачі можуть створювати та ділитися власним відеоконтентом через цю платформу.

Facebook Watch інтегрується з соціальними функціями Facebook, такими як коментарі, реакції та спільний перегляд, що дозволяє залучити більше глядачів та взаємодіяти з контентом у реальному часі.

Рисунок 1.3 – Логотип відеоплатформи Facebook Watch

Facebook Watch зазвичай має прийнятну швидкість завантаження, але це напряму залежить від загальної навантаженості платформи. Facebook має велику кількість користувачів, тому обробка може займати більше часу у порівнянні з іншими платформами. Передача відео може бути ефективною, але швидкість може звеншуватись при великій кількості одночасних переглядів.

Dailymotion (рисунок 1.4) – це популярна відеоплатформа, яка надає можливість користувачам переглядати, завантажувати та ділитися відеоконтентом. Заснована у 2005 році у Франції, Dailymotion відома своїм широким асортиментом відео, який включає у себе відомі короткометражки, музичні відеокліпи, новини, спортивні події та інші розважальні відеоматеріали. Платформа пропонує також можливості для монетизації контенту для створювачів відео. Хоча Dailymotion менш популярна, ніж YouTube, вона все ще є важливим гравцем на ринку відеострімінгу та володіє великою аудиторією в багатьох країнах світу.

Dailymotion має хорошу швидкість завантаження, подібну до Vimeo. Обробка відео може бути швидшою або повільнішою в залежності від навантаженості серверів. Dailymotion забезпечує ефективну передачу відео, але швидкість може коливатись в залежності від географічного розташування користувачів.

Рисунок 1.4 – Логотип відеоплатформи Dailymotion

Підведемо підсумки порівняння цих платформ, їх можна подивитись в таблиці 1.1.

Таблиця 1.1 – Підсумок порівняння швидкості роботи платформ

Назва платформи	Швидкість завантаження відео	Швидкість обробки відео	Швидкість передачі відео
YouTube	Висока	Швидка	Ефективна
Vimeo	Середня	Швидка	Ефективна
Facebook Watch	Прийнятна	Залежить від навантаження	Може спадати при великих навантаженнях
Dailymotion	Середня	Коливається	Ефективна

1.3.2 Масштабованість

YouTube має найбільшу масштабованість серед перерахованих платформ. Це обумовлено значною аудиторією, яка складається з мільйонів користувачів, які щодня переглядають, коментують і завантажують відео. Щоб забезпечити безперебійну роботу для такої великої кількості користувачів, YouTube має велику інфраструктуру серверів, яка дозволяє обробляти і передавати великі обсяги відеоданих одночасно [5].

Хоча Vimeo також може масштабуватися, вона має меншу аудиторію порівняно з YouTube. Це означає, що Vimeo може мати меншу потребу в такому рівні масштабування. Проте, вона все ще повинна забезпечувати достатню пропускну здатність і стабільність для своїх користувачів.

Facebook має велику користувацьку базу, тому платформа зазвичай масштабується добре. Однак, у випадку великих навантажень, які можуть виникати при масштабних подіях або вірусних відео, Facebook може зазнавати проблем зі швидкістю завантаження або доступності контенту. Тому Facebook постійно вдосконалює свою інфраструктуру, щоб забезпечити кращу масштабованість.

Dailymotion також може масштабуватися, проте вона має менші ресурси порівняно з YouTube та Facebook. Це може призводити до обмежень у масштабуванні, особливо при великих обсягах відео або при пікових навантаженнях. Тим не менш, Dailymotion продовжує розвивати свою інфраструктуру для поліпшення масштабованості і задоволення потреб своїх користувачів. Підсумок оцінки масштабованості можна побачити в таблиці 1.2.

Таблиця 1.2 – Підсумок порівняння масштабованості платформ

Назва платформи	Масштабованість
YouTube	Висока
Vimeo	Середня
Facebook Watch	Висока
Dailymotion	Середня

1.3.3 Безпека та надійність

YouTube має великі фінансові та технічні ресурси, оскільки вона є частиною компанії Google, одного з найбільших технологічних гігантів у світі. Це дозволяє

їй мати розгалужену інфраструктуру з великою кількістю серверів та мережевих засобів. YouTube встановлює високі стандарти безпеки даних, що включають шифрування, моніторинг захисту від вторгнень та застосування кращих практик управління доступом [6].

Vimeo має менші фінансові ресурси порівняно з YouTube. Це може вплинути на їхню здатність інвестувати у розширення інфраструктури та вдосконалення систем безпеки. Незважаючи на це, Vimeo також прагне дотримуватися високих стандартів безпеки даних, проте їхні можливості можуть бути обмеженими у порівнянні з YouTube.

Facebook має значні фінансові та технічні ресурси, подібно до YouTube, що дозволяє їй використовувати розгалужену інфраструктуру та застосовувати передові технології у сфері кібербезпеки. Хоча Facebook зазвичай має високий рівень безпеки та надійності, вона може стикатися з проблемами через кібератаки або технічні неполадки, але вона активно реагує на такі ситуації та постійно вдосконалює свої заходи безпеки.

Dailymotion має обмежені фінансові та технічні ресурси порівняно з гігантами, такими як YouTube та Facebook. Це може обмежувати їхню здатність інвестувати у розширення мережевої інфраструктури та розробку безпеки. Хоча Dailymotion також прагне забезпечити безпеку та надійність для своїх користувачів, їхні можливості можуть бути обмеженими через обмежені ресурси. Підсумок оцінки безпеки та надійності можна побачити в таблиці 1.3.

Таблиця 1.3 – Підсумок порівняння безпеки та надійності платформ

Назва платформи	Безпека	Надійність
YouTube	Висока	Висока
Vimeo	Середня	Середня
Facebook Watch	Висока	Висока
Dailymotion	Середня	Середня

1.4 Використання методів та інструментів для аналізу серверної частини

1.4.1 Навантажувальні тестування

Навантажувальне тестування – це метод оцінки продуктивності та надійності серверної частини під високим навантаженням. Цей метод допомагає визначити максимальну кількість користувачів, яких може обслуговувати сервер за одиницю часу, час відгуку сервера при різних навантаженнях, слабкі місця в системі [7].

Існує два основних типи навантажувальних тестів:

1. Тести на проникнення – ці тести спрямовані на визначення максимального обсягу ресурсів (наприклад, мережевої пропускної здатності, обсягу процесорного часу, обсягу оперативної пам'яті тощо), який може обробляти сервер або програмне забезпечення перед перевантаженням або відмовою. Такі тести зазвичай характеризуються великими обсягами одночасних запитів або штучного трафіку, який генерується на короткий проміжок часу. Можуть використовуватися різноманітні інструменти для генерування трафіку, такі як Apache Jmeter, Gatling, або власноруч написаний код, який створює запити до сервера з певною інтенсивністю. Основна мета полягає у визначенні точки, коли сервер або програмне забезпечення починає проявляти ознаки перевантаження, такі як збільшення часу відповіді, відмови у виконанні запитів, або зниження продуктивності.

2. Тести на витривалість – ці тести спрямовані на визначення того, наскільки добре сервер або програмне забезпечення може підтримувати стабільну роботу протягом тривалого періоду часу. Такі тести зазвичай характеризуються тривалим генеруванням трафіку або навантаженням на сервер, щоб перевірити, чи може система підтримувати стабільну продуктивність протягом довшого часу. Можуть використовуватися інструменти для автоматизації тестування, що дозволяють

створювати стабільне навантаження на сервер, такі як Apache Bench, Siege або власноруч написаний код. Оцінка полягає у виявленні будь-яких проблем, пов'язаних зі зниженням продуктивності або стійкості сервера під час тривалого тестування.

Інструменти для навантажувального тестування:

1. Apache JMeter – є одним з найпопулярніших та розповсюджених інструментів для навантажувального тестування. Він здатний моделювати різноманітні типи навантажень, відправляти запити до сервера, аналізувати відповіді та оцінювати продуктивність системи. Підтримує різні типи протоколів, такі як HTTP, HTTPS, JDBC, JMS, FTP, тощо. Має можливості графічного інтерфейсу користувача та може бути налаштований для автоматизованого тестування через конфігураційні файли.

2. Apache Bench – є однією з найпоширеніших утиліт для виконання навантажувальних тестів. Це інструмент командного рядка, який постачається разом з веб-сервером Apache HTTP Server. Apache Bench дозволяє створювати паралельні HTTP запити до веб-сервера з вказаною інтенсивністю. Він вимірює час відправки запиту, час отримання відповіді і ряд інших метрик, що дозволяє оцінити продуктивність сервера. Користувач може задати різні параметри, такі як кількість паралельних запитів, тривалість тесту, URL або файл з запитами тощо. Apache Bench легкий у використанні, має простий синтаксис командного рядка і добре підходить для швидких тестів на навантаження.

3. Siege – є ще одним популярним інструментом для проведення навантажувальних тестів. Це також командний рядок, але надає більше розширених можливостей порівняно з Apache Bench. Siege дозволяє створювати більш складні сценарії тестування, включаючи можливість налаштування параметрів запитів, таких як заголовки, методи HTTP, куки тощо. Він також може генерувати звіти про продуктивність та навантаження. Siege використовується за допомогою конфігураційних файлів, де можна визначити різні сценарії тестування

та їх параметри. Крім того, він може працювати в режимі неперервного тестування. Siege дозволяє більш гнучко налаштувати тестові сценарії, що робить його корисним для більш складних випадків тестування.

4. Gatling – є інструментом для навантажувального тестування, який базується на Scala. Він спроектований для використання у складних та великих проектах, де потрібна висока продуктивність та ефективність. Gatling використовує асинхронний, не блокуючий I/O та може обробляти велику кількість одночасних користувачів. Має можливості збирання та аналізу даних продуктивності за допомогою зручного інтерфейсу.

5. LoadRunner – є одним з найстаріших та найбільш відомих інструментів для навантажувального тестування, розробленим компанією Micro Focus. Він дозволяє створювати та виконувати тести з великою кількістю одночасних користувачів, щоб оцінити продуктивність системи. LoadRunner має широкі можливості для моделювання складних навантажень, включаючи віртуальних користувачів, мережеві протоколи, бази даних тощо. Він також має інструменти для аналізу результатів тестування.

6. K6 – є відкритим інструментом для навантажувального тестування, який зосереджується на простоті використання та здатності швидко створювати та виконувати тести. Він також може бути легко інтегрований у процеси CI/CD. K6 використовує сучасні архітектурні рішення, що дозволяють ефективно тестувати високонавантажені системи. Він має зручний інтерфейс командного рядка та можливості автоматизації тестування.

1.4.2 Аналіз логів

Аналіз логів - це метод виявлення проблем та помилок в роботі серверної частини. Логи містять інформацію про всі дії, які відбуваються на сервері, а також про будь-які помилки, які виникли.

Інструменти для аналізу логів:

1. ELK Stack (Elasticsearch, Logstash, Kibana) [8] – це набір вільно розповсюджуваних програмних засобів, які спільно використовуються для збору, обробки, зберігання, візуалізації та аналізу логів та інших структурованих даних.

Elasticsearch - це потужний інструмент для зберігання та пошуку даних у реальному часі. Він базується на Apache Lucene і надає розподілений механізм зберігання даних з підтримкою розподіленого пошуку, агрегації та аналізу. Elasticsearch дозволяє швидко виконувати пошук, агрегацію, фільтрацію та сортування великого обсягу даних.

Logstash - це інструмент для збору, обробки та направлення різноманітних типів даних, включаючи логи, події та інші структуровані дані. Він здатен приймати дані з різних джерел, таких як журнали серверів, бази даних, мережеві пристрої, інші сервіси тощо. Після прийому даних Logstash може їх обробляти (наприклад, розбивати на поля, фільтрувати, перетворювати формат) та направляти до Elasticsearch для подальшого зберігання і аналізу.

Kibana - це веб-інтерфейс, який надає можливості візуалізації та аналізу даних, збережених у Elasticsearch. Він дозволяє створювати різноманітні графіки, діаграми, дашборди та звіти на основі даних, що зберігаються в Elasticsearch. Крім того, Kibana надає користувачам інтуїтивний інтерфейс для виконання розширеного пошуку, фільтрації та аналізу даних.

2. Graylog – є відкритою платформою для збору, індексації та аналізу логів. Вона включає інструменти для збору логів з різних джерел, їхньої обробки та аналізу, а також створення звітів та сповіщень про помилки. Graylog дозволяє швидко збирати великі обсяги логів, використовуючи різноманітні джерела, такі як системні журнали, сервери, мережеві пристрої тощо. Вона також має розширені функції аналізу та візуалізації даних для виявлення та вирішення проблем.

3. Splunk – є комерційною платформою для аналізу даних, включаючи логи, метрики та інші дані. Вона дозволяє збирати, індексувати та аналізувати дані з різних джерел для виявлення проблем та вирішення їх. Splunk має потужний

механізм пошуку та аналізу даних, що дозволяє швидко виявляти проблеми та аналізувати їх причини. Вона також має розширені функції візуалізації та створення звітів для представлення результатів аналізу даних.

1.4.3 Моніторинг продуктивності

Моніторинг продуктивності - це метод збору та аналізу даних про роботу серверної частини, а саме: навантаження на CPU, використання пам'яті, мережевий трафік, час відгуку сервера і т.д.

Моніторинг продуктивності допомагає виявити проблеми до того, як вони вплинуть на користувачів, визначити слабкі місця в системі, оптимізувати серверну частину для кращої продуктивності [9].

Інструменти для моніторингу продуктивності:

1. Prometheus – є відкритим інструментом моніторингу та трендів. Він здатен збирати дані про навантаження на CPU, використання пам'яті, мережевий трафік, час відгуку сервера та інші метрики. Prometheus має гнучку систему збору метрик, що дозволяє встановлювати моніторинг для різних типів сервісів. Він також має вбудовану підтримку для алертів та інтеграцію з іншими інструментами.

2. Grafana – це інструмент для візуалізації та аналізу даних з різних джерел, включаючи метрики, які збираються за допомогою Prometheus, New Relic, Datadog тощо. Grafana надає можливості створення графіків, дашбордів та алертів для моніторингу продуктивності. Вона також має широкий набір плагінів та інтеграцій з різними джерелами даних.

3. New Relic – це комерційний сервіс моніторингу продуктивності, який надає розширені можливості для збору та аналізу метрик. New Relic пропонує інструменти для моніторингу навантаження на CPU, використання пам'яті, мережевого трафіку, часу відгуку сервера та інших метрик. Вона також має аналітичні можливості для виявлення та вирішення проблем продуктивності.

4. Datadog – це інтегрована платформа моніторингу, яка дозволяє збирати та аналізувати дані з різних джерел, включаючи метрики серверної частини, логи, трейси тощо. Datadog пропонує розширені можливості моніторингу та аналізу метрик, включаючи автоматизоване виявлення аномалій, аналітику в реальному часі та інтеграцію з різними іншими інструментами.

1.5 Висновки

У розділі 1 проведено аналіз існуючих платформ для роботи з відеоконтентом, що дозволило виявити кілька ключових висновків. Використання хмарних сервісів забезпечує значну гнучкість і масштабованість, але потребує ретельного вибору відповідних рішень для забезпечення безперебійної роботи та безпеки даних.

Виявлено, що ефективність кодеків та процесу транскодингу відео має критичне значення для якості і швидкості обробки відео. Використання сучасних форматів і оптимізованих алгоритмів транскодингу є необхідним для досягнення високої продуктивності.

Трансляція відео і використання CDN дозволяють значно покращити швидкість доставки контенту користувачам, але потребують інтеграції з надійними провайдерами послуг для забезпечення мінімальних затримок і високої доступності.

Подальші кроки включають вибір оптимальних технологій зберігання та обробки відео, інтеграцію ефективних методів трансляції та CDN, а також забезпечення високого рівня безпеки і надійності системи. Використання методів навантажувального тестування, аналізу логів та моніторингу продуктивності дозволить підтримувати стабільну роботу сервісу та швидко реагувати на можливі проблеми.

2 АРХІТЕКТУРА СИСТЕМИ ТА ВИБІР ПРОГРАМНИХ ТА АПАРАТНИХ ЗАСОБІВ РОЗРОБКИ

2.1 Архітектура системи

2.1.1 Загальні відомості

Архітектура є важливим складовим елементом при розробці будь-якого програмного продукту. Вона визначає загальну структуру, взаємозв’язки та взаємодію всіх компонентів системи з метою досягнення поставлених цілей ефективності, масштабованості, надійності та зручності використання [10].

Основними принципами архітектури є модульність, розширюваність та повторне використання коду. Ключові складові архітектури:

1. Структурні компоненти – включає в себе модулі, підсистеми та інші структурні одиниці, які співпрацюють для досягнення цілей системи.

2. Інтерфейси – визначають спосіб взаємодії між компонентами системи. Це може бути API для зовнішніх взаємодій або інтерфейси між внутрішніми модулями.

3. Логіка взаємодії – визначає правила та протоколи, за якими компоненти спілкуються між собою.

4. Дані та бази даних – описує структуру та організацію даних, а також їх зберігання та обробку.

5. Безпека та доступність – враховує заходи безпеки та механізми забезпечення доступності даних та функцій системи.

6. Масштабованість та продуктивність – планує можливості системи розвиватись та масштабуватись відповідно до зростання навантаження та вимог користувачів.

7. Розгортання та управління конфігурацією – описує процеси встановлення, налаштування та управління програмним забезпеченням.

Створення ефективної архітектури програмно-технічної системи вимагає ретельного аналізу вимог до системи, планування та розробки. Правильно спроектована архітектура дозволяє забезпечити розвиток системи з мінімальними витратами часу та ресурсів, а також забезпечує її стійкість та ефективність протягом усього періоду експлуатації.

Одним із варіантів архітектури для проектованої системи є монолітна (рисунок 2.1). Будувати серверний додаток на цій архітектурі просто, але якщо в додатку буде багато коду, то це призведе до сповільнення обслуговування та компіляції проекту, а також кожне оновлення в коді буде приводити до перезапуску всього додатку, що є недопустимо для системи обробки відео, так як призведе до тимчасових перебоїв для користувачів.

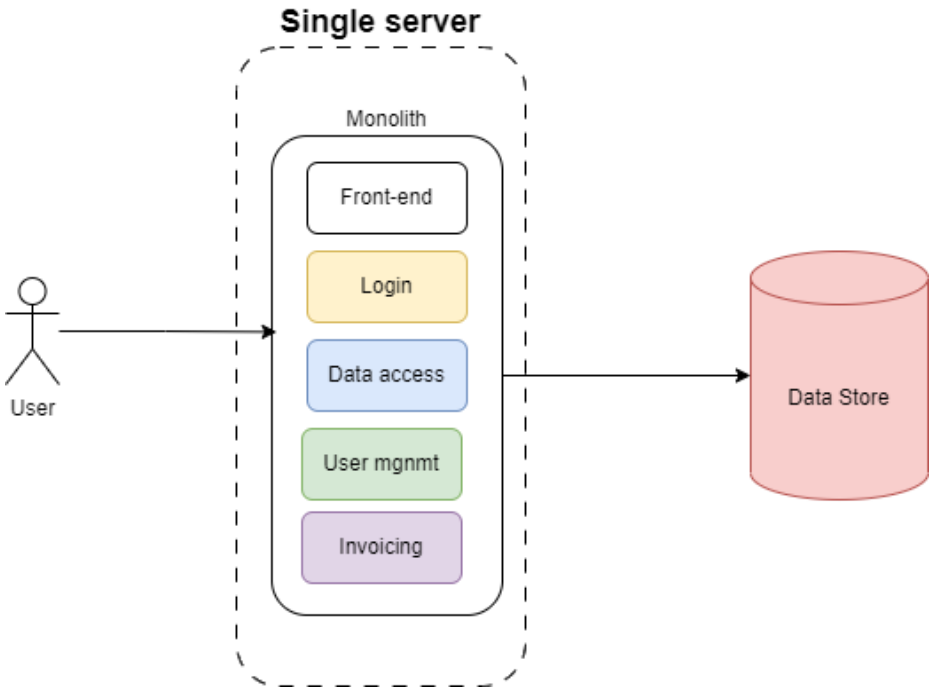


Рисунок 2.1 – Приклад монолітної архітектури

Для системи обробки відео оптимальним вибором буде будувати мікросервісну архітектуру, оскільки монолітна архітектура буде важкою в обслугованні та буде потребувати більшої кількості ресурсів на одному сервері.

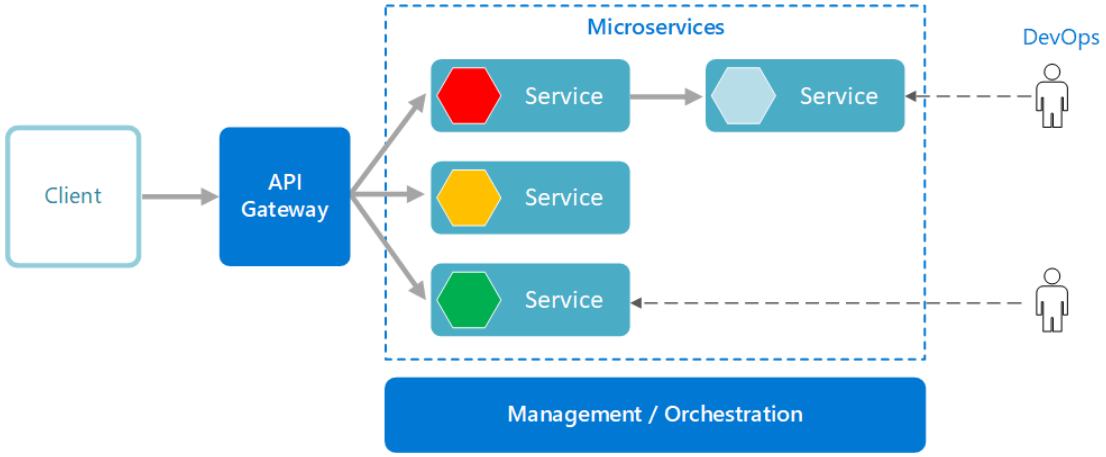


Рисунок 2.2 – Приклад мікросервісної архітектури

Тому під час проектування була обрана саме мікросервісна архітектура, яка складається із багатьох компонентів, а саме із мікросервісів API Gateway, Auth, Community, History, Library, Search, Storage, Subscriptions, Users, Video Manager, Video Processor, Video Store та Web Status. На рисунку 2.3 зображена схема спроектованої архітектури системи.

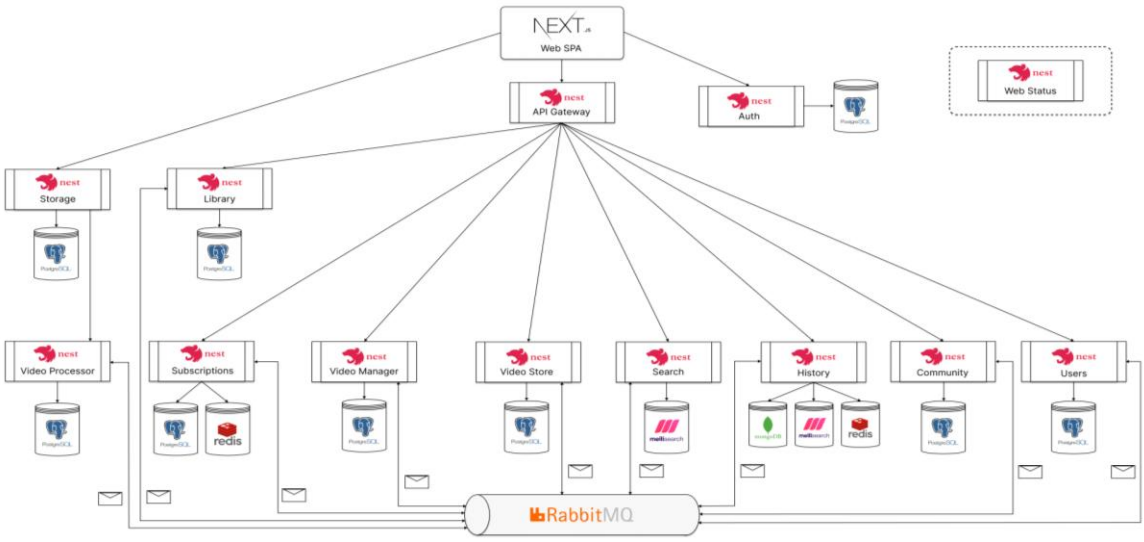


Рисунок 2.3 – Схема спроектованої архітектури системи

2.1.2 Опис елементів архітектури

Мікросервіс Video Manager відіграє ключову роль у системі, керуючи відео та їх метаданими. Вона обробляє запити користувачів на створення нового відео,

генерує токени для завантаження, які дозволяють користувачам завантажувати відеофайли у мікросервіс Storage. Після завантаження відеофайлу мікросервіс реєструє відео в інших мікросервісах, включаючи Video Store Library, Community та History.

Після реєстрації відео у ключових елементах системи, мікросервіс Video Manager публікує інтеграційну подію (через брокер повідомлень RabbitMQ) для Video Processor, щоб ініціювати обробку відео. Після обробки відео він публікує інтеграційні події для сповіщення інших мікросервісів про завершення обробки. Мікросервіс керує та зберігає оригінальні метадані відео, такі як заголовок відео, опис, видимість, теги тощо, і синхронізує оновлені метадані з Video Store для подальшого обслуговування.

Мікросервіс Video Manager також має можливість видаляти відео (в системі для відео буде ставитись статус «Unregistered», для можливості майбутнього відновлення відео) у відповідь на запити користувачів. Також Video Manager відповідальний за надсилання сповіщень користувачам у реальному часі щодо статусу обробки відео.

Мікросервіс Video Processor відповідає за обробку відеофайлів. Цей мікросервіс використовує потужність FFmpeg для обробки відео та генерує різні роздільні здатності та прев'ю завантаженого відео. Для коректної роботи цього мікросервіса FFmpeg повинен бути встановлений заздалегідь, що буде зроблено автоматично за допомогою Dockerfile. Якщо обробка відео буде перервана через тимчасові проблеми, її буде автоматично повторено, але за умови, що спроба не перебільшить ліміт допустимих. Video Processor може автоматично горизонтально масштабуватися за допомогою горизонтального масштабатора подів Kubernetes.

Мікросервіс Video Store відповідає за зберігання та надання метаданих відео користувачам. Цей мікросервіс контролює публікацію відео та забезпечує реплікацію метаданих відео з іншими мікросервісами, такими як Library.

Мікросервіс Storage відповідає за зберігання та обслуговування відео та зображень. Як користувачі, так і інші мікросервіси можуть завантажувати файли

до Storage. Для завантаження файлів користувач повинен надати токен завантаження. Токен завантаження – це JWT (JSON Web Token), згенерований мікросервісом Video Manager, що підтверджує, що користувач має право завантажувати файл. Токен завантаження містить набір ключів (claims), таких як ідентифікатор користувача, ідентифікатор відео, категорія файлу та інші. Після завантаження зображення користувачу повертається інший JWT токен, що містить інформацію про завантажений файл. Цей токен використовується мікросервісом Users, для підтвердження, що зображення використовується, інакше воно буде очищене через певний час.

Після завантаження відео Мікросервіс Storage публікує інтеграційну подію для сповіщення мікросервісу Video Manager про необхідність обробки, не повертаючи токен користувачеві. Різні підходи для зберігання відео та зображень використовуються через те, що відеофайли зазвичай значно більші за розміром і вимагають більше обробки. Існує фоновий сервіс для очищення файлів, які більше не використовуються.

Мікросервіс Auth є ключовим компонентом системи, відповідальним за автентифікацію та авторизацію користувачів. Він забезпечує безпечний доступ до ресурсів та сервісів, використовуючи JWT (JSON Web Token) для керування сесіями користувачів.

Мікросервіс Users відповідає за керування та зберігання профілів користувачів і даних каналів, а саме: створення, зберігання та керування даними профілів, включаючи відображуване ім'я користувача, псевдонім, опис, електронну пошту, прев’ю та банер каналу. Створення профілю користувача вимагає асинхронної розподіленої транзакції для реєстрації профілю користувача в інших мікросервісах, щоб забезпечити коректну роботу всіх мікросервісів

Мікросервіс Subscriptions відповідає за керування підписками і зберігання повідомлень користувачів. Цей мікросервіс дозволяє користувачам підписуватися на бажані канали, а також зберігає повідомлення користувачів. Коли оновлюється

підписаний канал, наприклад, публікується нове відео, цей мікросервіс зберігає відповідні повідомлення в базі даних.

Мікросервіс Library керує плейлистами та надає користувачам доступ до плейлистів і основних метаданих відео, а також забезпечує функціонал голосування за відео. Він керує плейлистами користувачів, такими як плейлист вподобань, плейлист "подивитися пізніше" та індивідуальні плейлисти. Голосування за відео реалізується шляхом додавання відео до плейлисту вподобаних або невподобаних відео, дозволяючи користувачам висловлювати свої вподобання або невподобання.

Мікросервіс Community забезпечує платформу для обговорень користувачів. Він дозволяє користувачам додавати коментарі до відео, відповідати на коментарі інших користувачів, редагувати та видаляти свої коментарі, а також окрім коментування, користувачі можуть голосувати за коментарі, допомагаючи спільноті визначити найкорисніші, найцікавіші або найрозважальніші коментарі.

Мікросервіс History відповідає за керування та зберігання історії переглядів користувачів, а також за підрахунок кількості переглядів кожного відео.

Мікросервіс Search забезпечує функціонал повнотекстового пошуку відео, а також пошуку за тегами, релевантних відео, а також найновіших відео.

API Gateway забезпечує централізовану точку входу для всіх клієнтських запитів і розподіляє їх до відповідних мікросервісів, за винятком мікросервісів Storage (щоб зменшити навантаження на шлюз) та Auth (також зменшення навантаження на шлюз та додавання централізованого сервісу аутентифікації, що в майбутньому дасть можливість авторизовувати інші продукти за допомогою цього ж серверу, як наприклад це робить компанія Google). Додатково шлюз має функціонал тротлингу запитів, щоб зменшити навантаження на систему.

Web Status надає користувацький інтерфейс для перегляду стану системи.

2.2 Вибір програмних засобів

2.2.1 Платформа та цільова операційна система

Всі компоненти системи будуть працювати на платформі Node.js (рисунок 2.4) – це вільне, відкрите середовище для виконання JavaScript-коду поза веб-браузером [11]. В основі Node.js лежить двигун V8, розроблений Google, який використовується також у браузері Chrome [12].

Рисунок 2.4 – Логотип платформи Node.js

Node.js дозволяє JavaScript виконувати серверний код, що відкриває безліч можливостей для розробки веб-додатків, мережевих додатків, API та інших застосунків.

Основні особливості платформи Node.js:

1. Асинхронність – працює за асинхронною моделлю вводу-виводу, що дозволяє обробляти багато запитів одночасно без блокування потоку виконання.

2. Кросплатформенність – може запускатися на багатьох операційних системах, таких як Windows, macOS, Linux і т. д.

3. Пакетний менеджер – ідеально поєднується з npm (Node Package Manager), який є одним з найбільших репозиторіїв пакетів програмного забезпечення. Завдяки npm розробники можуть легко встановлювати, оновлювати та керувати залежностями своїх проектів.

4. Підтримка – Node.js регулярно отримує оновлення, кожен раз покращуючи продуктивність та безпеку платформи.

5. Розширені можливості – за допомогою Node.js можна створювати різноматнітні застосунки, включаючи веб-додатки, веб-сервери, інструменти командного рядка та інше.

6. Спільнота – Node.js має велику та активну спільноту розробників, яка постійно розвивається та надає підтримку новачкам.

Цільовою операційною системою була обрана Ubuntu Server LTS 22.04 (рисунок 2.5) – це самий популярний дистрибитув Linux для серверів. Ця ОС створена на архітектурі Debian і пропонує баланс між стабільністю та актуальними пакетами програмного забезпечення [13].

Рисунок 2.5 – Логотип операційної системи Ubuntu

Ubuntu Server є надійною і масштабованою платформою з акцентом на простоту використання та безпеку. Він пропонує широкий спектр програмних пакетів і має велику спільноту підтримки. Для Ubuntu Server пропонується 10 років підтримки та оновлень для LTS (5 років без платної підписки), стандартний цикл випуску (6 місяців) та сумісність з більшістю програм та програмного забезпечення.

Ubuntu Server має дружній інтерфейс і велику документацію, що робить його простим у налаштуванні та керуванні. Він вигідно відрізняється регулярними оновленнями, довгостроковою підтримкою та великим репозиторієм програмного забезпечення.

Оскільки Ubuntu Server використовує Debian як основу, цей дистрибутив поділяє деякі недоліки свого батьківського. Це робить його залежним від репозиторіїв висхідних потоків для своїх пакетів та деяких критичних оновлень.

Мінімальні вимоги для Ubuntu Server включають 64-бітний процесор, 2 ГБ оперативної пам'яті та 25 ГБ дискового простору.

2.2.2 Протоколи взаємодії

В системі буде використано два протоколи взаємодії:

1. HTTP – це синхронний протокол обміну даних [14], який в розробленій системі використовується для запитів на Auth і Storage мікросервіси, а також на API шлюз з клієнту та сам шлюз робить HTTP запити на решту мікросервісів. Також цей протокол використовується всередині системи для авторизації запитів користувача: коли користувач робить запит, наприклад на отримання свого профілю, він передає токен, який валідується завдяки HTTP запиту на сервіс авторизації, який в свою чергу вже видає базову інформацію користувача, по якій можна отримати потрібні дані. На рисунку 2.6 зображена схема роботи HTTP протоколу.

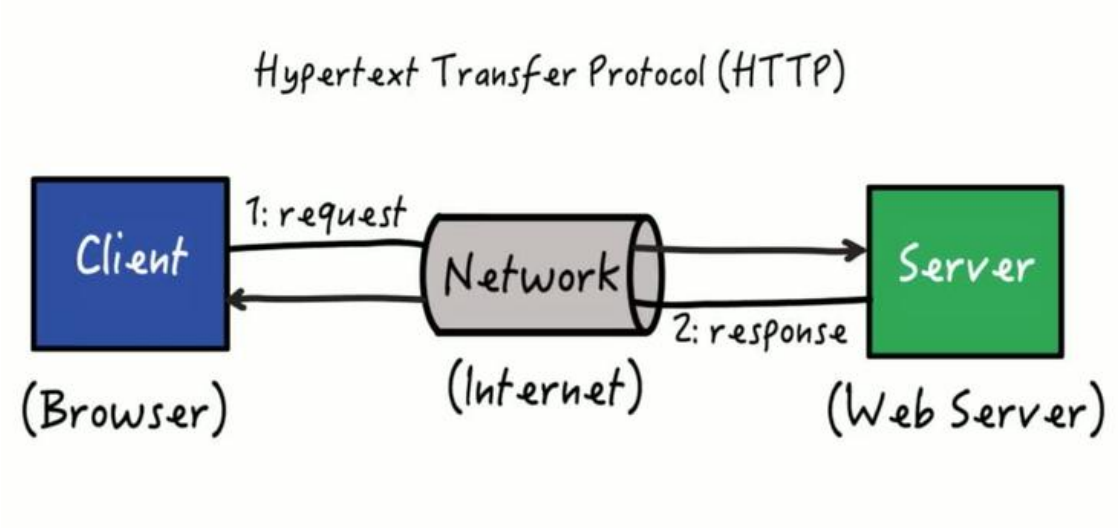


Рисунок 2.6 – Схема роботи HTTP протоколу

2. AMQP – це асинхронний протокол обміну повідомлень, який базується на моделі Publish-Subscribe, тобто якийсь із мікросервісів підписується на отримання повідомлень з якоїсь черги, а інший мікросервіс їх публікує [15]. Цей протокол використаний для публікації внутрішніх подій, наприклад коли користувач реєструється і створює профіль в Users мікросервісі, після його створення система публікує повідомлення в декілька черг брокера RabbitMQ [16], з яких інші мікросевірси їх отримують та опрацюють, коли зможуть. На рисунку 2.7 зображено схему роботи протоколу AMQP.

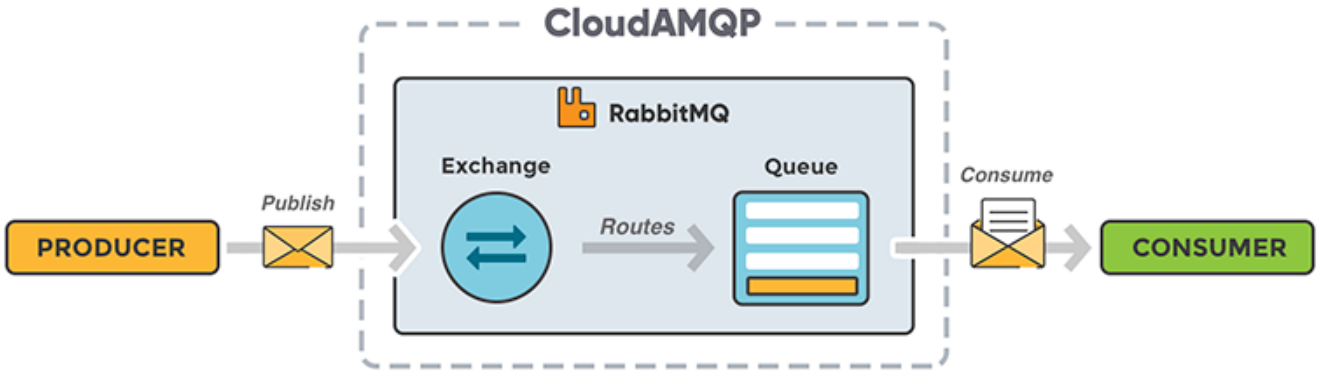


Рисунок 2.7 – Схема роботи протоколу AMQP

2.2.3 Бази даних

В спроектованій системі буде використано декілька баз даних, а саме PostgreSQL, MongoDB, Redis та Meilisearch.

PostgreSQL є однією з найпотужніших та надійних систем управління базами даних у світі. Ця об'єктно-реляційна СУБД відкритого коду надійно дотримується стандартів SQL, що робить її дуже привабливою для розробників, які шукають надійний інструмент для зберігання та керування своїми даними [17]. PostgreSQL підтримує широкий спектр функціональностей, включаючи розширення для роботи з JSON, XML, геоданими та іншими нестандартними типами даних. Він також надає засоби для підтримки високої доступності та резервування, що робить його популярним в середовищах, де вимагається

надійність та стійкість до відмов. Завдяки активній спільноті користувачів та розробників, PostgreSQL постійно розвивається і вдосконалюється, що робить його відмінним вибором для широкого спектру застосувань, від малих веб-додатків до великих корпоративних систем.

В спроектованій системі PostreSQL буде відповідальним за зберігання різних даних в кожному з вказаних мікросервісів:

1. Auth – інформація про дані для входу користувача та токени відновлення.
2. Community – інформація про відео форуми, коментарі та їх метрики.
3. Storage – інформація про завантажені файли людьми та іншими сервісами.
4. Subscription – інформація про підписки користувачів та їх повідомлення.
5. Users – інформація про профілі користувачів.
6. Library – інформація про плейлісти користувачів та метрики лайків/дизлайків відео.
7. Video Manager – інформація про відео, метрики та згенеровані прев'ю.
8. Video Store – інформація про відео та метрики (реплікація з Video Manager).
9. Video Processor – інформація про завантажені відео на обробку та кроки обробки відео.

MongoDB – це документ-орієнтована база даних [18], розроблена 11 лютого 2009 року компанією MongoDB Inc., яка в наші дні є членом альянсу MACH Alliance (в склад якої також входить досить відома американська IT компанія EPAM Systems [19], офіси якої є в Україні). Вона використовує JSON-подібні документи для зберігання даних і відрізняється від традиційних реляційних баз даних, таких як PostgreSQL, оскільки не вимагає схеми заздалегідь визначених даних.

MongoDB дозволяє легко зберігати та опрацьовувати структуровані, неструктуровані та напівструктуровані дані, що робить її популярною для веб-

розробників та додатків з високою масштабованістю. Крім того, MongoDB має вбудовану підтримку геоданих та можливості для розподіленої реплікації та кластеризації, що дозволяє побудувати високодоступні та надійні системи. Завдяки своїй гнучкості та простоті використання, MongoDB часто використовується для розробки сучасних додатків, особливо в середовищах, де змінність даних є нормою.

В спроектованій системі MongoDB буде використовуватись лише одним мікросервісом, а саме History, для зберігання інформації про переглянуті відео користувача та кількості переглядів на кожному відео.

Redis – це динамічна база даних ключ-значення, яка використовується для швидкого зберігання та доступу до даних у вигляді ключів, асоційованих з різними типами значень, такими як рядки, хеші, списки, множини та сортовані множини [19]. Відомий своєю високою продуктивністю та швидкістю, Redis використовується для широкого спектру застосувань, включаючи кешування, сесійне сховище, черги повідомлень та багато іншого. Крім того, Redis підтримує реплікацію даних та федерованість, що дозволяє побудувати розподілені та високодоступні системи. Завдяки своїм потужним можливостям та простоті використання, Redis є популярним рішенням для розробки швидких та масштабованих додатків.

В спроектованій системі Redis буде використовуватись в таких мікросервісах, як: Subscriptions та History. В Subscriptions він буде відповідальний за зберігання повідомлень, а в History для тимчасового зберігання кількості переглядів відео, які система періодично буде отримувати та зберігати в основну базу даних.

Meilisearch – це швидка пошукова система з відкритим вихідним кодом, яка легко налаштовується, розроблена для сучасних веб-додатків [21]. Вона була розроблена у 2018 році Клеманом Думергом і Томасом Парізо, щоб забезпечити ефективну та масштабовану пошукову систему, яку можна легко інтегрувати у веб-додатки [22]. Meilisearch використовує розширені алгоритми для обробки

пошукових запитів і ранжирування результатів пошуку на основі релевантності. Він може обробляти велику кількість даних і трафіку, що робить його придатним як для малих, так і для великих веб-додатків. Пошукова система надає низку розширених функцій пошуку, таких як захист від помилок, фасетний пошук і підтримка синонімів.

В спроектованій системі Meilisearch планується використовуватись в мікросервісах History та Search. В History вона буде використана для реалізації швидкого пошуку історії переглядів користувачів, а в Search – реалізація пошуку відео по тексту, видачі релевантних відео, а також пошуку відео за тегами.

2.2.4 Фреймворки і бібліотеки

Для реалізації усіх програмних елементів системи було обрано фреймворк Nest.js. Це прогресивний фреймворк для створення ефективних, надійних та масштабованих серверних додатків на основі Node.js [23]. Nest.js використовує TypeScript як основну мову програмування та надихається архітектурними принципами Angular, що забезпечує структурованість та зрозумілість коду [24].

Основними перевагами Nest.js є:

1. Модульна архітектура – Nest.js підтримує модульну структуру, що полегшує розподіл функціональності та повторне використання коду.
2. Вбудована підтримка мікросервісів – фреймворк має вбудовані засоби для створення мікросервісів і підтримує різні транспортні протоколи.
3. Декларативний підхід – завдяки використанню декораторів, код стає більш зрозумілим та зручним у підтримці.
4. Висока продуктивність – Nest.js оптимізований для виконання високонавантажених задач, що робить його ідеальним вибором для нашої системи.

Для взаємодії з базами даних СУБД PostgreSQL було обрано бібліотеку Prisma ORM. Це ORM (Object-Relational Mapping) нового покоління з відкритим кодом. Prisma ORM складається з наступних частин:

1. Клієнт Prisma – автоматично створюваний і типізований конструктор запитів для Node.js і TypeScript.
2. Prisma Migrate – система міграції (рисунок 2.8).
3. Prisma Studio – графічний інтерфейс для перегляду та редагування даних у заданій базі даних.

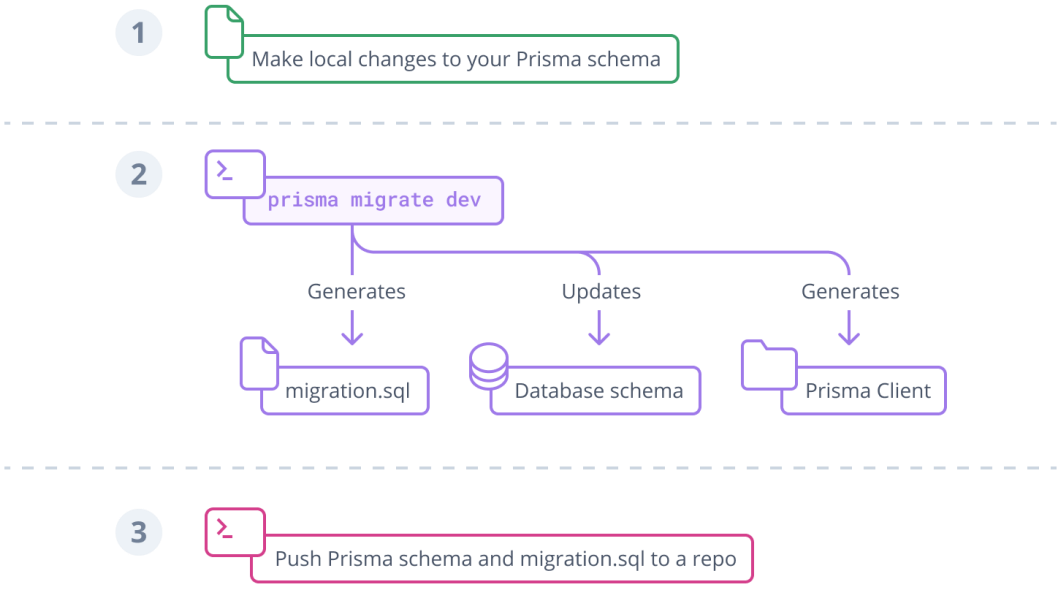


Рисунок 2.8 – Схема робочого процесу міграцій Prisma ORM

Prisma клієнт може бути використаний в любій підтримуваній Node.js версії та серверних додатках, побудованих з допомогою TypeScript, включаючи Serverless додатки та мікросервіси [25]. Це може бути REST API, GraphQL API, gRPC API, або будь що інше, що потребує бази даних. Кожен проект, який використовує Prisma ORM, починається з написання Prisma схеми. Ця схема дозволяє розробникам визначити їхні моделі додатку на інтуітивній мові моделювання даних. Вона також містить рядок підключення до бази даних та визначає генератор типів. Також потрібно зазначити, що коли в проекті

використовується TypeScript, то результати запитів будуть статично типізовані, що допоможе уникнути виклику отримання не існуючих пропсів, в результаті чого буде зменшена кількість багів в системі.

Для взаємодії мікросервісу з базою даних MongoDB буде використано бібліотеку Mongoose. На відміну від Prisma ORM, Mongoose є ODM (Object Data Modeling) бібліотекою, яка керує зв'язками між даними, забезпечує валідацію схеми та використовується для конвертації об'єктів в коді в їх представлення в MongoDB колекціях. На рисунку 2.9 зображена схема відображення об'єктів Node.js і MongoDB за допомогою бібліотеки Mongoose [26].

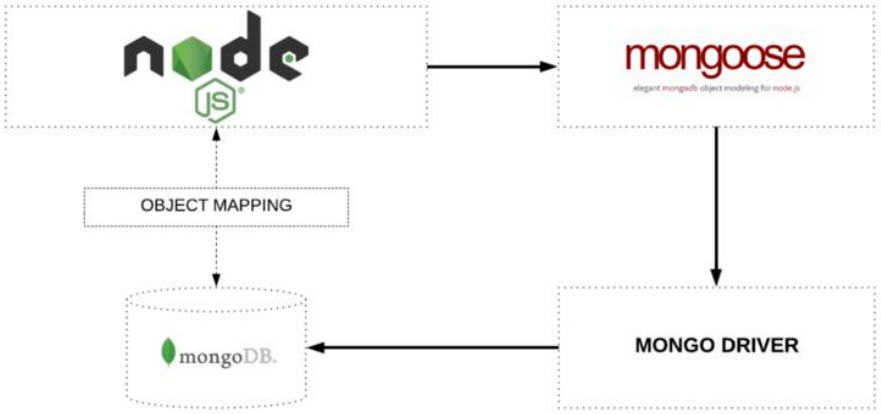


Рисунок 2.9 – Схема відображення об'єктів між Node.js і MongoDB за допомогою бібліотеки Mongoose

За замовчуванням MongoDB має гнучку модель даних. Завдяки цьому бази даних MongoDB дуже легко змінювати та оновлювати в майбутньому. Але багато розробників звикли до жорстких схем. Mongoose забезпечує напівжорстку схему з самого початку. У Mongoose розробники повинні визначити схему та модель. [27] Схема визначає структуру колекції документів, тобто напряму відображає MongoDB колекцію. Модель отримує схему і приймає її до кожного документу в колекції. Моделі відповідальні за CRUD дії для всіх документів.

Для взаємодії із базою даних Redis буде використовуватись бібліотека ioredis. Це надійний, повнофункціональний клієнт Redis, який використовується найбільшою у світі онлайн-компанією Alibaba [28]. Він має такі особливості:

1. Повнофункціональність – підтримує Cluster, Sentinel, Pipelining і, звісно, Lua скрипти та Pub/Sub (з підтримкою бінарних повідомлень).
2. Висока продуктивність.
3. Зручне API – підтримує як Node-style колбеки, так і проміси.
4. Підтримка трансформації аргументів команд і відповідей.
5. Підтримука бінарних даних.
6. Підтримка як TCP/IP, так і UNIX domain сокетів.
7. Підтримка офлайн черги та перевірки готовності.

Для взаємодії з Meilisearch буде використано бібліотеку meilisearch-js. Вона забезпечує зручний і ефективний інтерфейс для взаємодії з цим пошуковим механізмом. Використання цієї бібліотеки дозволяє легко інтегрувати Meilisearch в проект, спрощуючи виконання пошукових запитів, управління індексами та документами [29]. Крім того, бібліотека підтримує асинхронні операції, що підвищує продуктивність додатка. Вона також добре документована і має активну спільноту, що спрощує її використання та розв’язання можливих проблем.

2.3 Вибір апаратних засобів

Розгортання кожного мікросервісу в рамках такої системи має виконуватись на окремому сервері для того, щоб забезпечити кращу продуктивність, надійність та масштабованість системи. Основними причинами кожного мікросервісу на окремому сервері є:

1. Ізоляція збоїв – якщо один мікросервіс виходить з ладу, то це не вплине на працездатність інших мікросервісів, тобто проблема в одному мікросервісі не поширюється на іншу частину системи, забезпечуючи високу надійність всієї системи [30].

2. Масштабованість – різні мікросервіси мають різні вимоги до ресурсів. Можливість масштабування кожного мікросервісу окремо дозволяє ефективніше використовувати ресурси та збільшувати потужність системи відповідно до конкретних потреб [31].

3. Безпека – ізоляція мікросервісів на окремих серверах покращує безпеку системи. У разі компроментації одного із серверів, інші сервіси залишаться захищеними.

4. Управління ресурсами – запуск кожного мікросервісу на окремому сервері дозволяє більш точно контролювати використання ресурсів (центральний процессор, пам’ять, дисковий простір). Це забезпечує оптимальну продуктивність та знижує ризик конфліктів між сервісами.

5. Оновлення та розгортання – можливість оновлювати та розгоратати мікросервіси незалежно один від одного спрощує процес розробки і підтримки. Так можна вносини зміни в один мікросервіс, не порушуючи роботу інших.

6. Спеціалізація апартного забезпечення – для деяких мікросервісів можуть бути потрібні специфічні апаратні ресурси, такі як GPU, для обробки великих обсягів даних. Ізоляція на окремих серверах дозволяє використовувати відповіднре апаратне забезпечення для кожного мікросервісу. Прикладом, в спроектованій системи, є мікросервіс Video Processor, який оброблює відео та зображення використовуючи FFmpeg.

Спочатку необхідно обрати центральний процесор. Для забезпечення високої продуктивності та здатності обробляти велику кількість одночасних запитів було прийнято рішення будувати всі сервери на базі процесорів AMD EPYC 7742 (рисунок 2.10) [32]. Ці процесори мають високу обчислювальну потужність (64 ядра та 128 потоків) та підтримку багатопоточності, що робить їх чудовим вибором для виконання завдань в системі обробки відео.

Рисунок 2.10 – Процесор AMD EPYC 7742

Далі необхідно обрати материнську плату, на якій буде встановлено обраний процесор. Було обрано Supermicro H11DSi-NT (рисунок 2.11) [33], так як ця материнська плата розроблена для підтримки процесорів з великою кількістю ядер, що є необхідно для обробки значних обчислювальних навантажень. Також вона дозволяє додавати кілька високошвідкісних компонентів, таких як GPU (графічний процесор), мережві карти та SSD, а також підтримує великий об’єм ОЗП (оперативної пам’яті), так як має 16 слотів DIMM, що важливо для системи обробки відео. Також ця материнська плата підтримує встановлення одразу двух процесорів, що призведе до високого приросту обчислювальної потужності.

Рисунок 2.11 – Материнська плата Supermicro H11DSi-NT

Кожен сервер буде оснащений 128 ГБ оперативної пам’яті, цього буде достатньо для ефективної обробки великих обсягів даних. А саме було обрано 2 планки Kingston DDR4 64GB 3200 ECC REG RDIMM (KSM32RD4/64HCR) (рисунок 2.12) [34]. Вона має 64 ГБ пам’яті та швидкість 3200 МГц, яка зможе забезпечити швидку передачу даних і покращить загальну продуктивність. Також в майбутньому можливе розширення оперативної пам’яті на ще 14 планок (так як обрана материнська плата підтримує 16 планок ОЗП), тобто на 896 ГБ, тоді в загальному вийде 1 ТБ оперативної пам’яті, що є досить пристойним числом.

Рисунок 2.12 – Оперативна пам’ять Kingston DDR4 64GB 3200 ECC REG RDIMM (KSM32RD4/64HCR)

Також було обрано основний і вторинний накопичувачі. Основне сховище буде використовуватись для операційної системи. NVMe SSD забезпечують високошвидкісний доступ до даних і висока швидкість передачі даних, які є значно швидшими за традиційні SATA SSD. Так як материнська плата підтримує встановлення одразу двух SSD M.2 2280 1TB Samsung (рисунок 2.13) [35], тому буде встановлено одразу два (RAID 1 для резервування). Це дозволить зберегти дані в разі виходу з ладу одної із SSD.

Рисунок 2.13 – Накопичувач SSD M.2 2280 1TB Samsung

Вторинне сховище використовується для зберігання великих обсягів даних. Зберігатись дані будуть на двух SSD 2.5" 8TB Samsung (рисунок 2.14) [36] (RAID 1 для зберігання даних). Хоча SATA SSD і повільніші за NVMe SSD, але вони забезпечують значну місткість зберігання за нижчою вартістю.

Рисунок 2.14 – Накопичувач SSD 2.5" 8TB Samsung

Також необхідно обрати мережеві інтерфейсні карти (NIC), які підключають сервер до мережі, дозволяючи передачу даних між пристроями. Дві 10GbE NIC Intel X550-T2 (рисунок 2.15) [37] забезпечать високошвидкісне мережеве підключення, важличе для задач, що вимагають великих обсягів даних та масштабних передач даних (transfer). Особливостями цих мережевих карт є:

1. Подвійні порти – забезпечують резервування та збільшену пропускну здатність мережі.

2. 10GbE – підтримує високошвидкісні мережеві з'єднання, покращуючи темпи передачі даних і зменшуючи затримки в мережі.

Рисунок 2.15 – Мережева карта Intel X550-T2

Також для серверу, на якому буде запускатись мікросервіс Video Processor потрібно встановити графічний процесор NVIDIA Tesla V100 16GB для PCIe (рисунок 2.16) [38]. Ця відеокарта є еволюційним стрибком у продуктивності та можливості роботи з кількома робочими навантаженнями від центру обробки даних, що поєднує в собі найкращу в своєму класі професійну графіку з потужним обчисленням і прискоренням штучного інтелекту, щоб відповідати сучасним дизайнерським, творчим і науковим викликам. Вона надає професіоналам найсучасніші функції для відтворення з трасуванням променів, моделювання, віртуального виробництва тощо, тому ця відеокарта є ідеальним вибором для мікросервісу, який відповідальний за обробку відео і зображень.

Рисунок 2.16 – Відеокарта NVIDIA Tesla V100 16GB для PCIe

Варто зазначити, що вибір програмного забезпечення також впливає на апаратні вимоги. У спроектованій системі планується використовувати контейнеризацію за допомогою Docker [39] – це інструментарій для управління ізольованими Linux-контейнерами. Він дає змогу ізолювати додатки та їх залежності у контейнерах, забезпечуючи їхню портативність і спрощуючи розгортання.

Оркестрація контейнерів здійснюється за допомогою Kubernetes (K8s) [40] – це система з відкритим вихідним кодом для автоматичного розгортання, масштабування і управління контейнеризованими застосунками Вона підвищує гнучкість та надійність системи, дозволяючи ефективно використовувати апаратні ресурси.

2.4 Висновки

У розділі 2 проаналізовано архітектуру системи, а також здійснено вибір програмних і апаратних засобів для успішної реалізації проекту. Описано основні елементи архітектури системи, що дозволило визначити оптимальну структуру та взаємодію компонентів.

Було обрано платформу та цільову операційну систему, забезпечуючи належну інтеграцію та стабільність роботи. Розглянуто різні протоколи взаємодії, які гарантують ефективну комунікацію між компонентами системи. Обрані бази даних відповідають вимогам проекту щодо зберігання та обробки даних. Зроблено вибір фреймворків і бібліотек, що сприяють швидкій розробці та високій продуктивності системи.

Аналіз апаратних засобів включав вибір серверного обладнання, процесорів, оперативної пам'яті та накопичувачів, орієнтованих на максимальну ефективність і надійність.

3 ПРОГРАМНО-ТЕХНІЧНА РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ СИСТЕМИ ОБРОБКИ, ЗБЕРІГАННЯ ТА ПЕРЕДАЧІ ВІДЕОКОНТЕНТУ

3.1 Реалізація основних модулів програмного забезпечення

Після проектування архітектури і вибору апаратних елементів системи було вирішено одразу почати реалізовувати модулі програмного забезпечення.

Всі модулі були реалізовані на фрейморвці Nest.js, але розглядати ми будемо тільки самі ключові модулі системи, а саме модулі для мікросервісів Auth, Users, Video Manager та Video Processor. Спершу було розроблено модуль аутентифікації і авторизації для мікросервісу Auth. Перелік функцій цього модулю подано в таблиці 3.1.

Таблиця 3.1 – Перелік функцій модулю аутентифікації і авторизації

№ п/п	Назва функції	Опис функції
1	signup	Реєстрація користувача
2	login	Аутентифікація користувача
3	logout	Знищення сесії користувача
4	refresh	Поновлення токену сесії
5	createRecoveryToken	Створення токену відновлення
6	resetPassword	Верифікація вказаного токену відновлення та зміна паролю

Розглянемо тільки основні функції, а саме signup та login більш детально. Функція signup відповідальна за створення аккаунту користувача за допомогою

вказаних полів email та password. Спочатку відбувається пошук кандидата на вказаний email, після чого якщо кандит існує, то викликається виключення з помилкою про вже існуючий email в базі даних, так як email є унікальним на таблицю. Якщо ж кандидата не існує, то за допомогою бібліотеки bcryptjs [41] генерується «сіль» (salt), яка далі використовується для хешування паролю користувача за допомогою функції hash, яка приймає пароль та «сіль» в якості аргументів, з бібліотеки bcryptjs. Далі email та хешований пароль записуються в базу даних та створюється DTO (Data Transfer Object) [42] UserInfoDto з базовою інформацією, яка потрібна для наступого кроку, а саме створення сесії. Для створення сесії в таблицю UserSession записується рядок з парою ключів userAgent (агент користувача передається з браузера автоматично) та userId, а також токен відновлення сесії (refresh token), але спочатку пустий, оскільки його ще не було згенеровано. Після чого генерується пара токенів відновлення та доступу, які в результаті повертаються в кінці функції signup. В додатку А зображено блок-схему алгоритму реєстрації користувача. Фрагмент коду функції signup наведено нижче:

```
async signup({ email, password }: SignupDto, userAgent: string, ip: string) {
  const candidate = await this.prisma.user.findUnique({ where: { email } });
  if (candidate) {
    throw new BadRequestException(`User with provided email already exists`);
  }
  const salt = await genSalt(10);
  const hashPassword = await hash(password, salt);
  const user = await this.prisma.user.create({
    data: { email, password: hashPassword },
  });
  const userInfo = new UserInfoDto(user);
  const session = await this.createSession(userInfo, userAgent, ip);
  return { user: userInfo, ...session };
}
```

Функція login відповідальна за аутентифікацію користувача в системі. Вона приймає такі ж аргументи, як і signup та має майже ідентичний алгоритм дій з лише однією відмінністю, якщо кандита не існує, то викликається виключення з текстом про неправильний логін і пароль, а також замість створення користувача в базі даних, відбувається валідація правильності паролю за допомогою функції

compare з бібліотеки compare, яка приймає в якості аргументів введений пароль та вже існуючий захешований пароль з отриманого кандидата, після чого якщо паролі співпадають, то йдуть ті ж дії, що і в функції signup, а якщо ні – то викликається виключення з помилкою про неправильний логін або пароль. В додатку А зображено блок-схему алгоритму аутентифікації користувача. Фрагмент коду функції login наведено нижче:

```
async login({ email, password }: LoginDto, userAgent: string, ip: string) {
  const candidate = await this.prisma.user.findUnique({ where: { email } });
  if (!candidate) throw new BadRequestException('Invalid email or password');

  const isPasswordValid = await compare(password, candidate.password);
  if (!isPasswordValid)
    throw new BadRequestException('Invalid email or password');

  const userInfo = new UserInfoDto(candidate);
  const session = await this.createSession(userInfo, userAgent, ip);

  return { user: userInfo, ...session };
}
```

В двох із цих функції присутній виклик допоміжної функції createSession. В ній також присутня ще одна допоміжна функція generateTokens, яка генерує два JWT токена і повертає їх. Фрагмент коду функції createSession наведено нижче:

```
private async createSession(userInfo: UserInfoDto, userAgent: string, ip: string){
  const session = await this.prisma.userSession.upsert({
    where: { userAgent_userId:{userAgent, userId: userInfo.id} },
    create: { userId: userInfo.id, refreshToken: '', userAgent,ip},
    update: { ip },
  });
  try {
    const tokens = await this.tokenService.generateTokens(
      { id: userInfo.id, email: userInfo.email },
      session.id,
    );
    await this.prisma.userSession.update({
      where: { id: session.id },
      data: { refreshToken: tokens.refreshToken },
    });
    return { tokens, sessionId: session.id };
  } catch {
    await this.prisma.userSession.deleteMany({ where: { id: session.id } });
    throw new InternalServerErrorException('Unexpected error');
  }
}
```

Наступним було розроблено модуль для менеджементу профілями користувачів для мікросервісу Users. Перелік функцій цього модулю подано в таблиці 3.2.

Таблиця 3.2 – Перелік функцій модулю менеджементу профілями

№ п/п	Назва функції	Опис функції
1	create	Створює профіль користувача
2	update	Оновлює профіль користувача
3	getCreator	Отримати профіль користувача (по ідентифікатору або нікнейму користувача)

Функція create приймає два аргументи, а саме CreateCreatorDto з полями displayName та nickname, і userId. Після чого за допомогою функції create з моделі Creator (Prisma ORM) профіль зберігається в базу даних, після чого надсилається повідомлення в RabbitMQ Pub/Sub (публікація/підписка) [43] чергу для мікросервісів Video Manager, Subscriptions, Library, Community, Video Store, History та Search для створення в них цього користувача, тобто синхронізувати дані між мікросервісами. Нижче наведено фрагмент коду цієї функції:

```
async create(dto: CreateCreatorDto, userId: string) {
  try {
    const creator = await this.prisma.creator.create({
      data: {
        id: userId,
        displayName: dto.displayName,
        nickname: dto.nickname,
      },
    });
    this.logger.log(`Creator created for user (${creator.id})`);
    this.syncCreator(creator);
    return creator;
  } catch (e: any) {
```

```
    this.logger.error(e)
    if (e?.code === 'P2002') {
      throw new ConflictException(e);
    }
    throw new BadRequestException(e);
  }
}
```

Функція update працює по такій же схемі, як і create, тільки для оновлення даних додано нові поля, а саме description, email, thumbnailToken та bannerToken. Поля thumbnailToken та bannerToken містять JWT токени, які в собі містять ключ (claim) url. Тому в началі функції додано фрагмент коду, який відповідає на валідацію вказаних токенів thumbnailToken та bannerToken та декодує ці токени, в результаті чого дістає з них значення ключа url. Всі остальні дії ідентичні до функції create, єдина відмінність, це зміна функції create на update в моделі. Фрагмент коду функції update наведено нижче:

```
async update(dto: UpdateCreatorDto, userId: string) {
  if (Object.values(dto).every(value => isEmpty(value)))
    throw new BadRequestException('Invalid payload');
  const [thumbnailUrl, bannerUrl] = await Promise.all([
    (async () => isNotEmpty(dto?.thumbnailToken)
      ? await this.imageValidatorService.validateImageToken(
        dto.thumbnailToken,userId,USER_UPLOADED_THUMBNAIL)
      : undefined)(),
    (async () => isNotEmpty(dto?.bannerToken)
      ? await this.imageValidatorService.validateImageToken(
        dto.bannerToken,userId,USER_UPLOADED_BANNER)
      : undefined)(),
  ]);
  try {
    const creator = await this.prisma.creator.update({
      where: { id: userId },
      data: {
        displayName: dto?.displayName, nickname: dto?.nickname,
        description: dto?.description, email: dto?.email,
        thumbnailUrl, bannerUrl
      },
    });
    this.logger.log(`Creator updated for user (${creator.id})`);
    this.syncCreator(creator);
    return creator;
  } catch (e: any) {
    if (e?.code === 'P2002') throw new ConflictException(e);
    throw new BadRequestException(e);
  }
}
```

Далі на черзі модуль, який є відповідальним за менеджемент відео для мікросервісу Video Manager. Перелік функцій цього модулю подано в таблиці 3.3.

Таблиця 3.3 – Перелік функцій модулю менеджементу відео

№ п/п	Назва функції	Опис функції
1	createVideo	Створення відео
2	getVideoById	Отримання інформації про відео по його ідентифікатору
3	getVideos	Отримання переліку відео
4	getVideoUploadToken	Генерація токену для доступу до завантаження відео в мікросервіс Storage
5	updateVideo	Оновлення інформації про відео
6	unregisterVideo	Видалення відео

Функція createVideo приймає в якості аргументів ідентифікатор користувача та CreateVideoDto, яке складається із двух полів, а саме title та description (не обов’язкове поле). Функція спочатку перевіряє наявність користувача в базі даних, якщо його не існує, то викликається виключення, якщо ж користувач існує, то тоді створюється відео в базі даних та надсилається повідомлення в RabbitMQ чергу для мікросервісів Video Store, Library, History, Search та Community. В останньому створюється «форум» для відео, в якому користувачі зможуть залишати коментарі та будуть вестись метрики кількості коментарів. Нижче наведено фрагмент коду цієї функції:

```
async createVideo(creatorId: string, dto: CreateVideoDto) {
  const creator = await this.prisma.creator.findUnique({
    where: { id: creatorId },
    select: { id: true }});
  if (!creator) throw new BadRequestException('Creator does not exists');
  const video = await this.prisma.video.create({
    data: {
      creatorId,
      ...dto,
      isPublished: false,
      lengthSeconds: 0,
      processingStatus: 'WaitingForUserUpload',
      visibility: 'Private',
      thumbnailStatus: 'Waiting',
      status: 'Created',
      metrics: { create: {} },
    }});
  this.logger.log(`Video (${video.id}) is created`);
  const pattern = 'create_video';
  this.videoStoreClient.emit(pattern, new VideoStoreCreateVideoEvent(video));
  this.libraryClient.emit(pattern, new LibraryCreateVideoEvent(video));
  this.historyClient.emit(pattern, new HistoryCreateVideoEvent(video));
  this.searchClient.emit(
    pattern, new SearchCreateVideoEvent({
      ...video,
      tags: video?.tags?.split(',') || [],
    }));
  this.communityClient.emit(
    'create_forum', new CreateForumEvent(video.id, creatorId));
  return video;
}
```

Функції getVideoById та getVideo та getVideoUploadToken дуже прості (в останній просто вказується ідентифікатор відео, по якому генерується JWT токен з інформацією про користувача, який створив це відео, категорію відео та його ідентифікатор), а функція updateVideo працює по принципу функцій оновлення вище: оновили інформацію – надіслали повідомлення в RabbitMQ з новою інформацією.

Наступною в черзі на розгляд йде функція unregisterVideo. Вона приймає в якості аргументів ідентифікатори відео та користувача. За допомогою ідентифікатора відео перевіряється його наявність в базі даних, після чого, якщо воно є в базі, перевіряється, чи є в користувача доступ до видалення цього відео, якщо так, тоді в базі даних встановлюється статус Unregistered (відео зникне з переліку, але не видалиться з системи, для можливого його відновлення в майбутньому по запиту користувача) та відправляється повідомлення в RabbitMQ

тим же мікросервісам, що і в функції createVideo для видалення відео. Нижче наведено фрагмент коду цієї функції:

```
async unregisterVideo(videoId: string, creatorId: string) {
  const video = await this.prisma.video.findUnique({
    where: { id: videoId },
    select: { status: true, creatorId: true, processingStatus: true },
  });
  if (!video) throw new BadRequestException('Video not found');
  if (video.creatorId !== creatorId)
    throw new ForbiddenException('Is not your video');
  if (video.status === 'Unregistered') return { status: false };
  try {
    await this.prisma.video.update({
      where: { id: videoId },
      data: { status: 'Unregistered' },
    });
    const pattern = 'unregister_video';
    const data = { videoId };
    if (video.processingStatus === 'VideoBeingProcessed')
      this.videoProcessorClient.emit('process_video_cancel', data);
    this.communityClient.emit('unregister_forum', data);
    this.videoStoreClient.emit(pattern, data);
    this.libraryClient.emit(pattern, data);
    this.historyClient.emit(pattern, data);
    this.searchClient.emit(pattern, data);
    this.logger.log(`Video ${videoId} is unregistered`);
    return { status: true };
  } catch (e) {
    this.logger.error(e);
    return { status: false };
  }
}
```

Наступним буде модуль для обробки відео, які були завантажені в Storage, для мікросервісу Video Processor. Цей модуль є одним із самих ключовх в системі, оскільки він відповідає за обробку відео та зображень. Він є масштабним, тому буде розглянуто лише ключові моменти цього модулю.

Цей модуль одночасно може оброблювати три відео за допомогою бібліотеки bull [44], яка керує чергами завдань, які виконуються на фоні [45]. Коли надходить запит на обробку відео, він записується в базу даних Redis та очікує на старт в черзі, коли в черзі з’явиться місце, обробка відразу розпочнеться. Блок-схему алгоритму обробки відео та зображень можна буде побачити на схемі в додатку В. Перелік функцій цього модулю подано в таблиці 3.4.

Таблиця 3.4 – Таблиця функцій модулю обробки відео

№ п/п	Назва функції	Опис функції
1	start	Розпочинає обробку відео
2	processPreview	Генерує анімоване прев’ю для відео
3	processThumbnail	Генерує трьох мініатюр для відео
4	processVideo	Обробляє відео
5	updateVideoStatus	Оновлює статус відео

Функція start в якості аргумента приймає “payload” з повідомлення черги RabbitMQ з полями videoId, creatorId, originalFileName, videoUrl. Вона завантажує відео по посиланню, яке було вказано в videoUrl з Storage мікровервісу, після цього використовуючи утиліту probe з FFmpeg отримує метадані відеофайлу, зберігає відео в базі даних, надсилає повідомлення в RabbitMQ чергу на Video Manager для зміни статусу відео на VideoBeingProcessed (опрацьовується) та конкуретно (concurrently) [46] визиває три функції, а саме оновлення статусу відео на ProcessingThumbnails (обробка мініатюр), processPreview та processThumbnail. Після обробки мініатюр запускається наступні дві функції, а саме встановлення статусу відео ProcessingVideos (обробка відео) та обробка якостей відео (функція processVideos) теж конкуретно. Варто зазначити, що функція processVideos генерує відео якість одразу розбиту на HLS [47] фрагменти, що дозволить автоматично обирати якість відео в відеоплеєрі на клієнтській частині системи. Після виконання цих функцій відправляється повідомлення в RabbitMQ чергу для Video Manager про завершення обробки відео, в результаті чого відео публікується в системі, та видаляється завантажене відео та всі відео та

зображення, які були оброблені з диску. Фрагмент коду функції start наведено нижче:

```
async start(payload: VideoProcessPayload) {
    const { filePath, outputFolderPath, folderId } =
      await this.networkService.downloadVideo(payload.videoUrl);
    const videoMetadata = await this.ffmpegService.probe(filePath);
    const videoStream = videoMetadata.streams.find((x)=> x.codec_type ==='video');
    const audioStream = videoMetadata.streams.find((x)=> x.codec_type ==='audio');
    const format = videoMetadata.format;
    await this.prisma.video.create({
      data: {
        id: payload.videoId,
        creatorId: payload.creatorId,
        videoFileUrl: payload.videoUrl,
        originalFileName: payload.originalFileName,
        width: videoStream.width,
        height: videoStream.height,
        status: 'Pending',
      },
    });
    await lastValueFrom(
      this.videoManagerClient.send('set_processing_status', {
        videoId: payload.videoId, status: 'VideoBeingProcessed'}));
    try {
      await Promise.all([
        this.updateVideoStatus(payload.videoId, 'ProcessingThumbnails'),
        this.processPreview(payload, filePath, folderId, videoStream, format),
        this.processThumbnail(payload, filePath, folderId, videoStream),
      ]);
      await Promise.all([
        this.updateVideoStatus(payload.videoId, 'ProcessingVideos'),
        this.processVideo(
          payload,
          filePath,
          outputFolderPath,
          videoStream,
          audioStream,
```

```
          format,
        ),
      ]);
      this.videoManagerClient.emit('video_process_finished', {
        videoId: payload.videoId,
      });
      await removeTempFileOrFolder(outputFolderPath);
    } catch (e) {
      this.logger.error(e);
      await lastValueFrom(
        this.videoManagerClient.send('set_processing_status', {
          videoId: payload.videoId,
          status: 'VideoProcessingFailed',
        }),
      );
    }
  }
```

Також оскільки функції processVideos, processThumbnail та processPreview є масштабними, то фрагменти їх коду можна буде побачити в додатку Г.

Фрагмент коду функції updateVideoStatus наведено нижче:

```
private async updateVideoStatus(videoId: string, status: VideoProcessingStatus) {
    await this.prisma.video.update({
      where: { id: videoId },
      data: {
        status,
        processedAt: status === 'Processed' ? new Date() : undefined,
      },
    });
  }
```

3.2 Опис процесу створення баз даних

Для створення баз даних в СУБД PostgreSQL було спроектовано схеми баз даних Prisma ORM та на базі них згенеровані міграції за допомогою утиліти

Prisma Migrate. Кожна міграція автоматично запускається при старті Docker контейнеру.

Для мікросервісу Video Manager було спроектовано схему бази даних (рисунок 3.1), яка складається із моделей Creator, Video, VideoThumbnail, VideoMetrics та VideoPreviewThumbnail, а також із переліків (enums) VideoStatus, VideoVisibility, VideoProcessingStatus та VideoThumbnailStatus.

Відношення між таблицями (SQL relations) [48]:

1. Один до багатьох (One To Many) між таблицями Creator і Video.
2. Один до одного (One To One) між таблицями Video і VideoMetrics.
3. Один до багатьох (One To Many) між таблицями Video і VideoThumbnail.
4. Один до одного (One To One) між таблицями Video і VideoPreviewThumbnail.

Рисунок 3.1 – Візуалізована схема бази даних для мікросервісу Video Manager

Для мікросервісу Video Processor було спроектовано схему бази даних (рисунок 3.2), яка складається із моделей Video та VideoProcessingStep, а також з

переліку (enum) VideoProcessingStatus. Між таблицями Video і VideoProcessingStep відношення один до багатьох (One To Many).

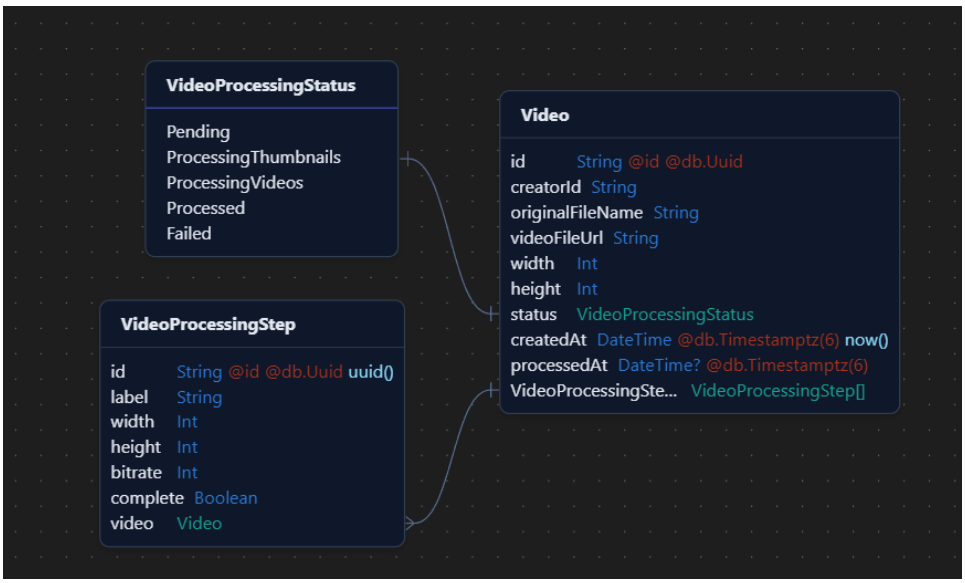


Рисунок 3.2 – Візуалізована схема бази даних для мікросервісу Video Processor

Для мікросервісу Video Store було спроектовано схему бази даних (рисунок 3.3), яка складається із моделей Creator, Video та VideoMetrics, а також із переліків (enums) VideoVisibility та VideoStatus. Тут таблиця Video має відношення один до одного (One To One) з таблицею VideoMetrics, а також відношення багато до одного (Many To One) з таблицею Creator.

Рисунок 3.3 – Візуалізована схема бази даних для мікросервісу Video Store

Для мікросервісу Storage було спроектовано схему бази даних (рисунок 3.4), яка складається із моделей File та FileTracking, а також із переліку (enum) FileStatus. Таблиця File має відношення один до одного (One To One) з таблицею FileTracking.

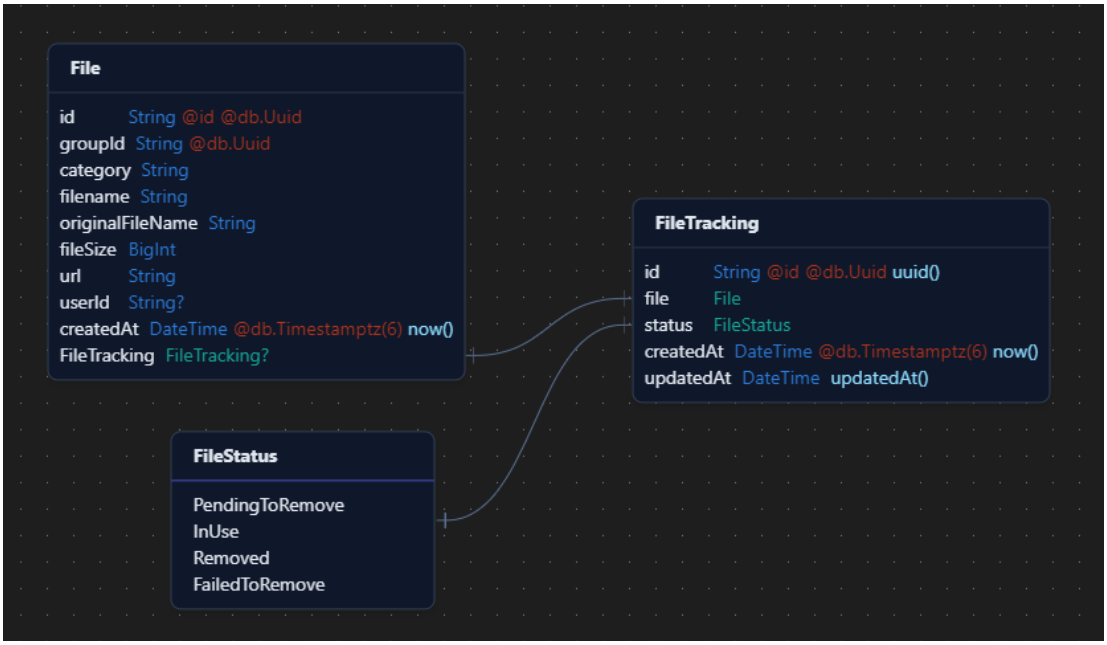


Рисунок 3.4 – Візуалізована схема бази даних для мікросервісу Storage

Для мікросервісу Auth було спроектовано схему бази даних (рисунок 3.5), яка складається із моделей User, UserSession та RecoveryToken. Таблиця User має відношення один до багатьох (One To Many) з таблицями UserSession і RecoveryToken.

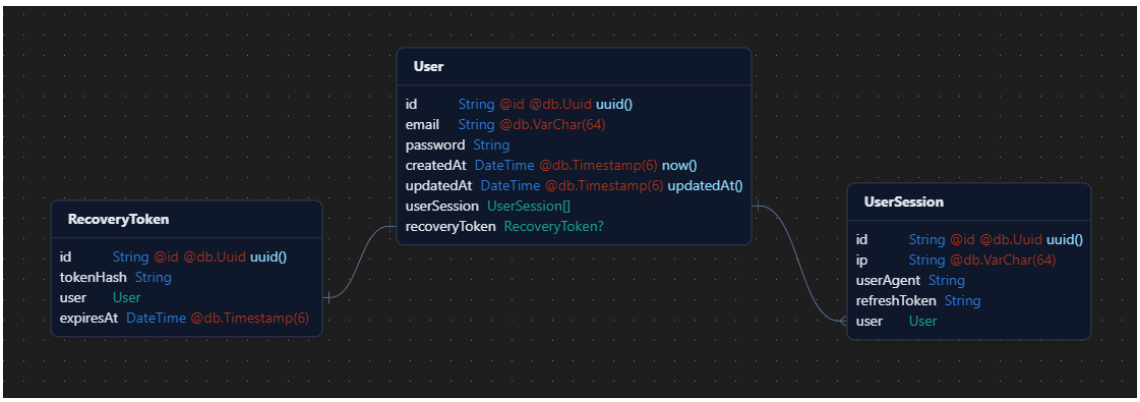


Рисунок 3.5 – Візуалізована схема бази даних для мікросервісу Auth

Для мікросервісу Users було спроектовано схему бази даних (рисунок 3.6), яка складається із моделі Creator та переліку (enum) CreatorStatus.

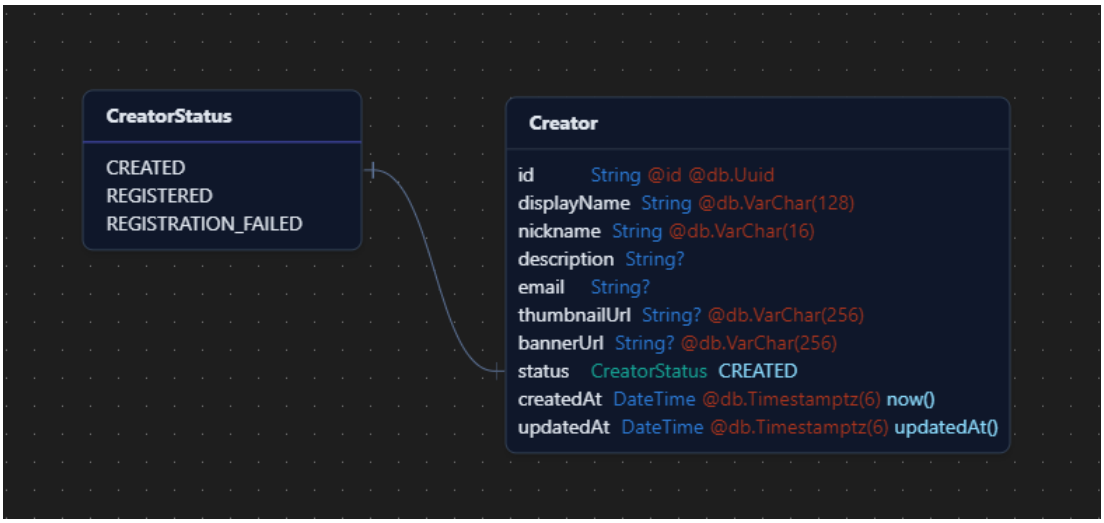


Рисунок 3.6 – Візуалізована схема бази даних мікросервісу Users

Для мікросервісу Subscriptions було спроектовано схему бази даних (рисунок 3.7), яка складається із моделей Creator та Subscription. Між таблицями Creator і Subscription є одночасно два відношення один до багатьох (One To Many) як автор і ціль підписки.

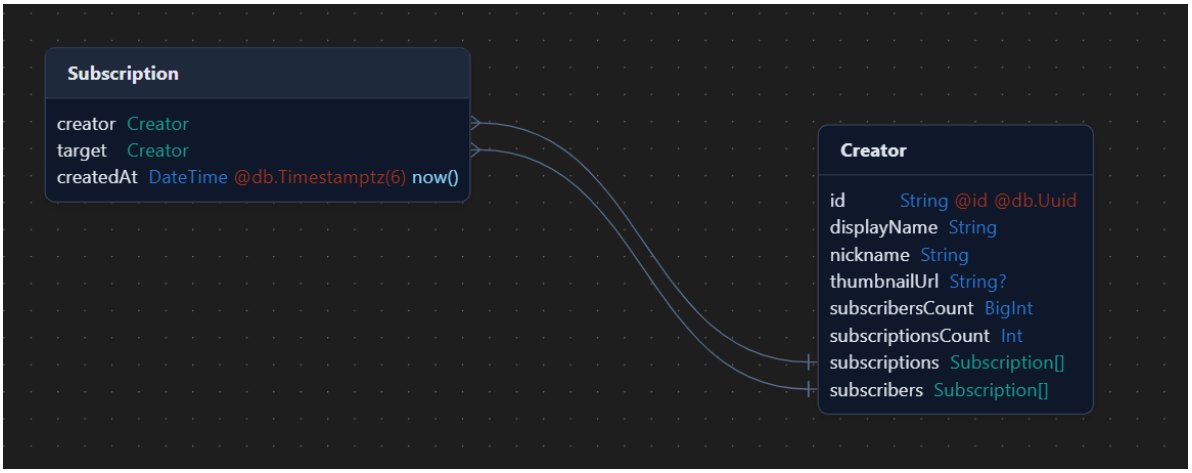


Рисунок 3.7 – Візуалізована схема бази даних для мікросервісу Subscriptions

Для мікросервісу Community було спроектовано схему бази даних (рисунок 3.8), яка складається із моделей Creator, VideoForum, VideoComment, VideoCommentVote, а також із переліків (enums) VideoForumStatus та VideoCommentVoteType.

Відношення між таблицями:

1. Один до багатьох (One To Many) між таблицями Creator і VideoForum.
2. Один до багатьох (One To Many) між таблицями VideoForum і VideoComment.
3. Один до багатьох з посиланням на себе (One To Many, Self-Referencing) між таблицями VideoComment і VideoComment – один “VideoComment” може мати багато “VideoComment” у вигляді відповідей.
4. Один до багатьох (One To Many) між таблицями Creator і VideoComment.
5. Один до багатьох (One To Many) між таблицями VideoComment і VideoCommentVote.
6. Один до багатьох (One To Many) між таблицями Creator і VideoCommentVote.

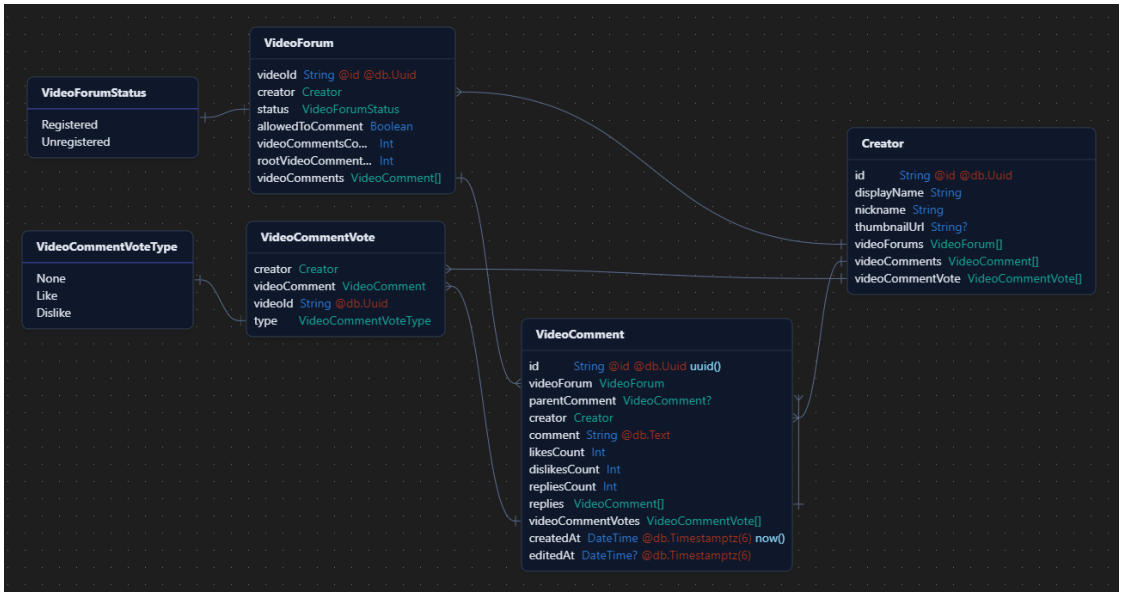


Рисунок 3.8 – Візуалізована схема бази даних мікросервісу Community

Для мікросервісу Library було спроектовано схему бази дани, яка складається із моделей Creator, Video, VideoMetrics, Playlist, PlaylistItem, LikedPlaylist, LikedPlaylistItem, DislikedPlaylist, DislikedPlaylistItem, WatchLaterPlaylist та WatchLaterPlaylistItem, а також із переліків (enums) PlaylistVisibility, VideoVisibility та VideoStatus.

Відношення між таблицями:

1. Один до багатьох (One To Many) між таблицями Creator і Playlist.
2. Один до багатьох (One To Many) між таблицями Creator і Video.
3. Один до одного (One To One) між таблицями Video і VideoMetrics.
4. Один до багатьох (One To Many) між таблицями Playlist і PlaylistItem.
5. Один до багатьох (One To Many) між таблицями Video і PlaylistItem.
6. Один до одного (One To One) між таблицями Creator і LikedPlaylist.
7. Один до багатьох (One To Many) між таблицями LikedPlaylist і LikedPlaylistItem.
8. Один до багатьох (One To Many) між таблицями Video і LikedPlaylistItem.
9. Один до одного (One To One) між таблицями Creator і DislikedPlaylist.
10. Один до багатьох (One To Many) між таблицями Disliked і DislikedPlaylistItem.
11. Один до багатьох (One To Many) між таблицями Video і DislikedPlaylistItem.
12. Один до багатьох (One To Many) між таблицями Creator і WatchLaterPlaylist.
13. Один до багатьох (One To Many) між таблицями WatchLaterPlaylist і WatchLaterPlaylistItem.
14. Один до багатьох (One To Many) між таблицями Video і WatchLaterPlaylistItem.

На рисунку 3.9 зображено спроектовану схему бази даних мікросервісу Library.

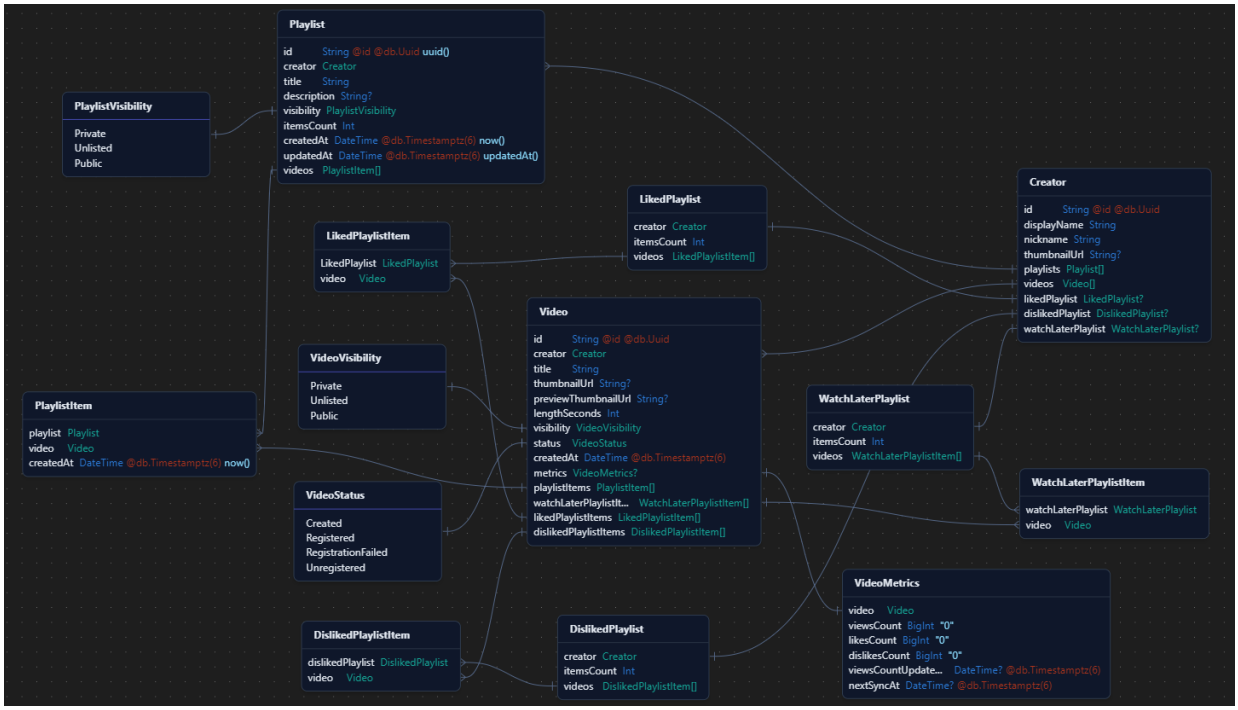


Рисунок 3.9 – Візуалізація схеми бази даних мікросервісу Library

Створення бази даних MongoDB не потребує запуску міграцій, так як вона не має можливості приняти статичну схему колекцій, як це робить будь-яка SQL база для своїх таблиць. Але бібліотека Mongoose дає змогу розробнику програмно визначити схему колекцій, які значно зменшуть ризик похибки при модифікації документів, але це всеріво не дає 100% гарантії, що в документи колекції не буде записано, оновлено або видалено лишніх полів. На рисунку 3.10 зображено програмно визначені поля схем VideoSchema та CreatorSchema.

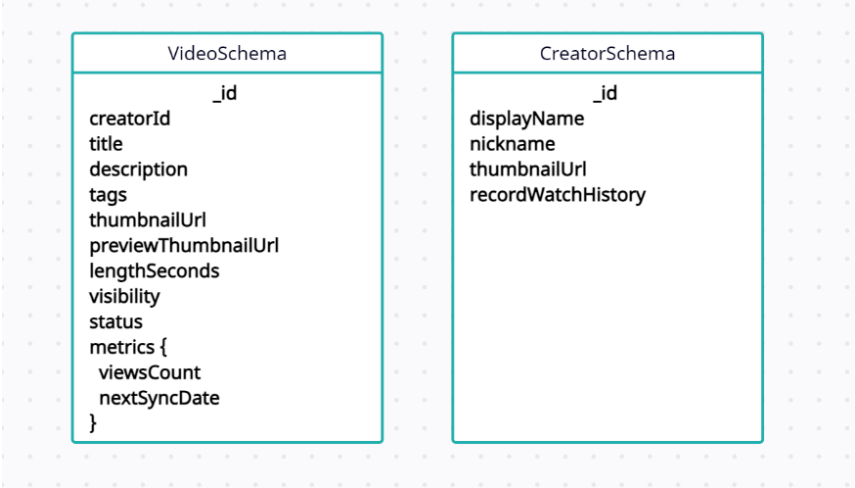


Рисунок 3.10 – Візуалізація схеми колекцій бази даних мікросервісу History

Також було визначено інтерфейси для індексів в пошуковій системі Meilisearch в мікросервісах History та Search. В мікросервісі History визначено інтерфейс UserHistoryIndex, код якого наведено нижче:

```
export interface UserHistoryIndex {
  id: string;
  creatorId: string;
  videoId: string;
  title: string;
  tags: string;
  viewAt: Date;
}
```

В мікросервісі Search визначено інтерфейси VideosIndex та CreatorsIndex, код яких також наведено нижче:

```
export interface VideosIndex {
  id: string;
  title: string;
  description: string;
  tags: string[];
  thumbnailUrl: string;
  previewThumbnailUrl: string;
  lengthSeconds: number;
  creatorId: string;
  metrics: {
    viewsCount: string;
    viewsCountUpdatedAt?: Date;
  };
  status: string;
  visibility: string;
  createdAt: Date;
}

export interface CreatorsIndex {
  id: string;
  displayName: string;
```

```
  nickname: string;
  thumbnailUrl?: string;
}
```

3.4 Пошукові можливості системи

У системі реалізовані потужні пошукові функції, які дозволяють користувачам швидко і ефективно знаходити необхідний відеоконтент. Для забезпечення цих можливостей у серверній частині системи передбачено чотири основних кінцевих точки (endpoints) [49], кожна з яких відповідає за певний тип пошуку. Перед їх оглядом слід зазначити, що любий пошук одразу фільтрує відео за видимістю і статусом, тобто всі відео мають мати статус Registered (зареєстрованого) та Public (публічного). Нижче наведено детальний опис кожної з кінцевих точок:

1. /api/search – ця кінцева точка доступна методом GET [50], та приймає три параметри URL, а саме один обов’язковий q та два необов’язкових page та perPage для посторінкового виводу результатів пошуку. Вона призначена для пошуку відео за ключовими словами або фразами, введеними користувачем. Пошуковий запит передається як параметр у URL [51] в ключ “q”. Система аналізує введений текст і здійснює пошук у метаданих відео, таких як ідентифікатор, назва, опис, теги та ідентифікатор користувача, який створив відео. Кінцевий URL для запиту пошуку по рядку буде виглядати так /api/search?q=music. Нижче наведено фрагмент коду функції для пошуку по «рядку»:

```
async searchByQuery(query: string, pagination: PaginationDto) {
    const videosResponse = await this.videosIndex.search(query, {
      page: pagination?.page ?? 1,
      hitsPerPage: pagination?.perPage ?? 20,
      facets: ['tags'],
      filter: ["status = 'Registered'", "visibility = 'Public'"],
    });
    return this.mapCreatorsToVideosResponse(videosResponse);
```

```
  }
```

Функція mapCreatorsToVideosResponse представляє з себе алгоритм пошуку всіх творців контенту по масиву ідентифікаторів, отриманих з масиву найдених відео, та об’єднання їх з відео. Фрагмент коду цієї функції наведено нижче:

```
async mapCreatorsToVideosResponse(videosResponse: SearchResponse<VideosIndex>) {
  const creatorsResponse = await this.creatorsIndex.getDocuments({
    filter: [
      `id IN [${videosResponse.hits.map((hit) => hit.creatorId).join(',')}]`]});
  if (videosResponse.totalHits > 0) {
    videosResponse.hits = videosResponse.hits.map((vh) => ({
      ...vh, creator:
        creatorsResponse.results.find((ch) => vh.creatorId === ch.id) || {},
    }));
  }
  return videosResponse;
}
```

2. /api/search/latest – ця кінцева точка доступна методом GET, може приймати два параметра URL, а саме page та perPage. Вона використовується для отримання списку найновіших відео, які були завантажені в систему. Вона повертає відео в порядку зворотної хронології, що дозволяє користувачам швидко побачити новий контент. Нижче наведено фрагмент коду функції для пошуку останніх відео:

```
async searchLatest(pagination: PaginationDto) {
  const videoResponse = await this.videosIndex.search(null, {
    sort: ['createdAt:desc'],
    filter: ["status = 'Registered'", "visibility = 'Public'"],
    page: pagination?.page ?? 1,
    hitsPerPage: pagination?.perPage ?? 20,
    facets: ['tags'],
  });
  return this.mapCreatorsToVideosResponse(videoResponse);
```

```
}
```

3. /api/search/by-tags – ця кінцева точка доступа методом POST [52], вона приймає в body поля tags, page та perPage в форматі JSON [53]. Вона дозволяє здійснювати пошук відео за певними тегами. Користувачі можуть вказувати один або кілька тегів, за якими вони хочуть знайти відео. Алгоритм створює запит для індексу відео, де фільтруються відео за зазначеними тегами. Теги формуються у вигляді рядка, де кожен тег обрізається від зайвих пробілів. Пошук здійснюється з обмеженням на кількість результатів на сторінку та з урахуванням поточної сторінки. Отримані результати обробляються, і формується відповідь для користувача. Нижче наведено фрагмент коду функції для пошуку відео за тегами:

```
async searchByTags(tags: string[], pagination: PaginationDto) {
  const videosResource = await this.videosIndex.getDocuments({
    filter: [
      `tags IN [${tags.map((tag) => tag.trim()).join(', ')}]`,
      "status = 'Registered'", "visibility = 'Public'"],
    offset: ((pagination?.page ?? 1) - 1) * (pagination?.perPage ?? 20),
    limit: pagination?.perPage ?? 20,
  });
  return this.mapCreatorsToVideoResource(videosResource);
}
```

4. /api/search/related/:videoId – ця кінцева точка доступна методом GET, може приймати параметри URL page та perPage. Вона призначена для пошуку відео, схожих на конкретне відео, ідентифікатор якого вказується в запиті. Спочатку система намагається знайти відео за заданим ідентифікатором. Якщо відео знайдено, створюється набір фільтрів, щоб знайти інші відео які не мають той самий ідентифікатор. Якщо у відео є теги, додається фільтр, що шукає відео з тими ж тегами або від того ж автора. Якщо тегів немає, використовується фільтр тільки за автором. Потім здійснюється пошук в індексі відео з цими фільтрами, сортуються результати за датою створення, а результати пошуку обробляються для повернення користувачеві. Якщо відео за ідентифікатором не знайдено,

система видає помилку "Video not found". Нижче наведено фрагмент коду функції для пошуку схожих відео:

```
async searchRelated(videoId: string, pagination: PaginationDto) {
    try {
      const video = await this.videosIndex.getDocument(videoId);
      const filter = [
        "status = 'Registered'",
        "visibility = 'Public'",
        `id != ${videoId}`,
      ];
      if (video.tags.length > 0) {
        filter.push(`tags IN [${video.tags.join(',')}] OR creatorId =
${video.creatorId}`);
      } else {
        filter.push(`creatorId = ${video.creatorId}`);
      }
      const videosResponse = await this.videosIndex.search(null, {
        sort: ['createdAt:desc'],
        filter,
        page: pagination?.page ?? 1,
        hitsPerPage: pagination?.perPage ?? 20,
        facets: ['tags'],
      });
      return this.mapCreatorsToVideosResponse(videosResponse);
    } catch {
      throw new BadRequestException('Video not found');
    }
  }
```

3.4 Тестування системи

Для забезпечення високої якості та надійності програмно-технічної системи з обробки, зберігання та передачі відеоконтенту були проведені ретельні тести на основі колекції у Postman [54] (рисунок 3.11). Тестування мало на меті перевірити

правильність роботи всіх функцій системи, а також їх відповідність заявленим вимогам.

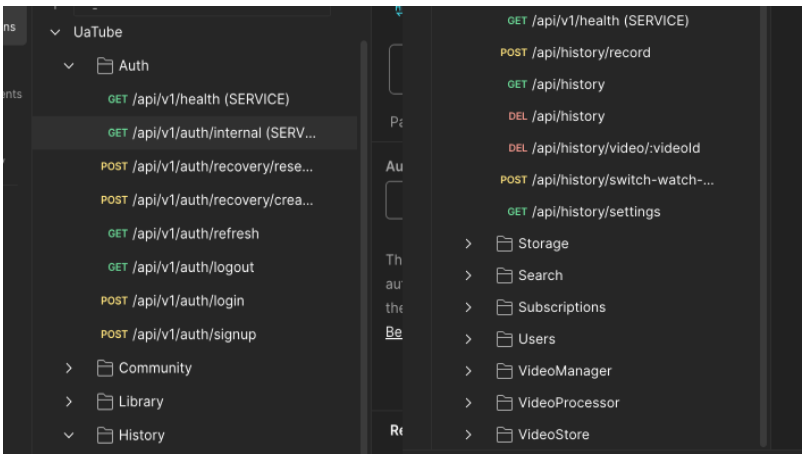


Рисунок 3.11 – Postman колекція

Для приклада використаємо любий елемент з колекції для тестування роботоздатності системи, наприклад /api/v1/auth/signup.

Для тестування необхідно обрати вкладку «Body», далі правіше від неї замість «RAW» обрати «JSON», з’явиться пусте поле, в якому необхідно записати об’єкт з полями email та password. Для поля email необхідно вписати любу валідну електронну адресу, а для password пароль довжиною до 64 символів. Далі необхідно настинути кнопку «Send», та зачекати на результат виклику кінцевої точки. На рисунку 3.12 зображено результат виклику кінцевої точки /api/v1/auth/signup.

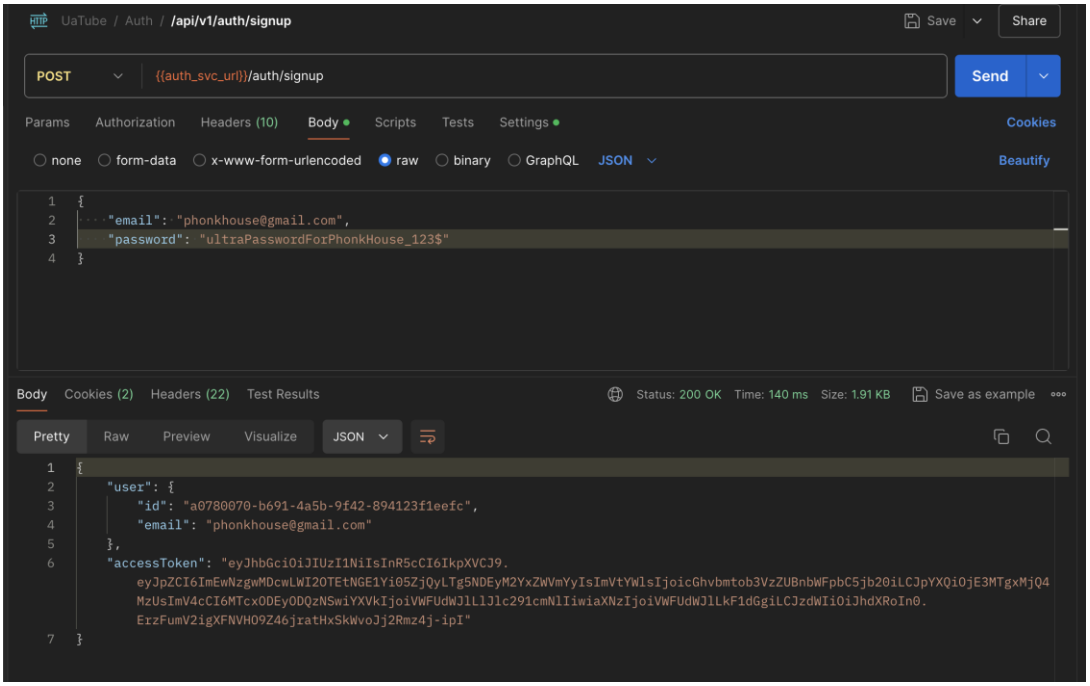


Рисунок 3.12 – Результат виклику кінцевої точки /api/v1/auth/signup

Після першого виклику система відпрацювала правильно, створила нового користувача та видала результат з HTTP кодом 200 (OK) [55] та JSON відповіддю з основною інформацією про створеного користувача та токеном доступу. Тепер необхідно спробувати викликати цю кінцеву точку ще раз, вказавши ті ж дані. Результат виклику зображено на рисунку 3.13.

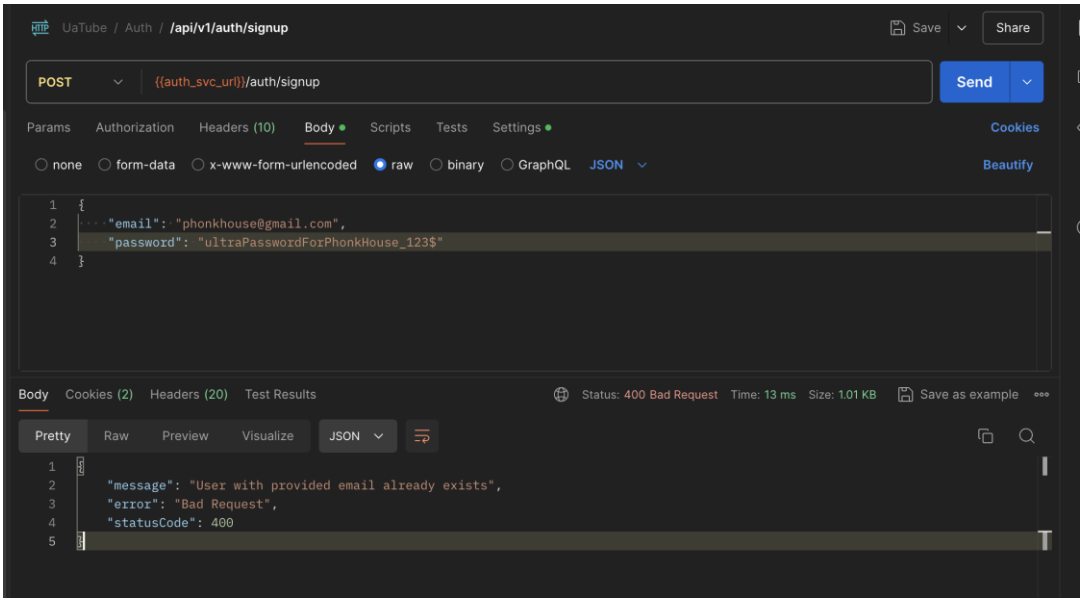


Рисунок 3.13 – Повторний результат виклику кінцевої точки /api/v1/auth/signup з тими ж даними

Як видно, виклик кінцевої точки завершився моментально з HTTP кодом 400 (Bad Request) та помилкою про те, що користувач з вказаним email вже існує. Тому можна зробити висновок, що система працює правильно.

3.5 Висновки

У розділі 3 було описано програмно-технічну реалізацію серверної частини системи обробки, зберігання та передачі відеоконтенту. Було розглянуто реалізацію основних модулів програмного забезпечення, описано процес створення баз даних детально описано, включаючи структуру та організацію даних, що забезпечує ефективне зберігання та швидкий доступ до відеоконтенту. Було розроблено пошукові можливості системи, які дозволяють користувачам швидко знаходити необхідний контент за різними критеріями, що значно покращує зручність використання системи.

Проведено тестування системи, яке включало перевірку функціональності, стабільності та продуктивності всіх модулів. Це забезпечило високу якість та надійність роботи системи в реальних умовах експлуатації.

Завдяки комплексному підходу до розробки, включаючи детальний аналіз і тестування, вдалося створити ефективну та стабільну серверну частину системи обробки, зберігання та передачі відеоконтенту, яка відповідає всім заявленим вимогам та стандартам якості.

ВИСНОВКИ

У кваліфікаційній роботі було розроблено серверну частину системи обробки, зберігання та передачі відеоконтенту.

У розділі 1 було досліджено існуючі рішення для роботи з відеоконтентом. Проаналізовано та визначено переваги та недоліки кожної системи. Визначено основні технології зберігання та обробки відео, включаючи використання хмарних сервісів, кодеків, форматів та транскодингу відео. Проведено огляд процесу трансляції та використання CDN, а також порівняння аналогів за характеристиками, такими як швидкість завантаження, обробки та передачі відео, масштабованість, безпека та надійність. Також проведено огляд методів та інструментів для аналізу серверної частини, зокрема навантажувальні тестування, аналіз логів та моніторингу продуктивності.

У розділі 2 обґрунтовано вибір архітектури системи та програмних і апаратних засобів для розробки. Описано елементи архітектури, що дозволило визначити оптимальну структуру та взаємодію компонентів. Було обрано платформу та цільову операційну систему для належної інтеграції та стабільності роботи. Розглянуто протоколи взаємодії, які гарантують ефективну комунікацію між компонентами системи, та обрані бази даних для зберігання і обробки даних. Також визначено фреймворки і бібліотеки для швидкої розробки та високої продуктивності системи. Проведено аналіз апаратних засобів, включаючи вибір серверного обладнання, процесорів, оперативної пам'яті та накопичувачів.

У розділі 3 було описано програмну реалізацію серверної частини системи. Розглянуто основні модулі програмного забезпечення, що забезпечують функціональність системи та ефективну обробку відеоконтенту. Описано процес створення баз даних, включаючи структуру та організацію даних для ефективного зберігання та швидкого доступу до відеоконтенту. Розроблено пошукові можливості системи, які дозволяють користувачам швидко знаходити необхідний

контент за різними критеріями. Проведено тестування системи, включаючи перевірку функціональності, стабільності та продуктивності всіх модулів, що забезпечило високу якість та надійність роботи системи в реальних умовах експлуатації.

Розроблена серверна частина системи обробки, зберігання та передачі відеоконтенту забезпечить ефективну роботу з відео на різних платформах та пристроях. Надалі можна досліджувати покращення системи, наприклад, розширення функціональності пошукових можливостей, покращення масштабованості та підвищення рівня безпеки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Що таке Інтернет? URL: https://shalabodin.com/shho-take-internet-czikavi-fakty-ta-istoriya-rozvytku-merezhi/ (дата звернення: 12.02.2024)
2. Кодек – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Кодек (дата звернення: 12.02.2024)
3. Livestreaming – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Livestreaming (дата звернення: 12.02.2024)
4. YouTube – Вікіпедія. URL: https://uk.wikipedia.org/wiki/YouTube (дата звернення: 12.02.2024)
5. Інфраструктура як Сервіс (IaaS). URL: https://www.sibis.com.ua/services/cloud-services-for-business/iaas/ (дата звернення: 12.02.2024)
6. Стандарти інформаційної безпеки – огляд. URL: https://www.dqsglobal.com/uk-ua/navchajtesya/blog/standarti-informacijnoyi-bezpeki-oglyad (дата звернення: 13.02.2024)
7. Тестування продуктивності. URL: https://training.qatestlab.com/blog/technical-articles/performance-testing/ (дата звернення: 13.02.2024)
8. Що таке ELK Stack? URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-elk-stack/ (дата звернення: 13.02.2024)
9. Що таке моніторинг сервера? URL: https://www.host-tracker.com/ua/blog/what-is-server-monitoring (дата звернення: 13.02.2024)
10. Системна архітектура – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Системна_архітектура (дата звернення: 28.03.2024)
11. Node.js – Вікіпедія. URL: https://en.wikipedia.org/wiki/Node.js (дата звернення: 28.03.2024)
12. V8 (рушій JavaScript) – Вікіпедія. URL: https://uk.wikipedia.org/wiki/V8_(рушій_JavaScript) (дата звернення: 28.03.2024)

13. Ubuntu. URL: https://uk.wikipedia.org/wiki/Ubuntu (дата звернення: 28.03.2024)

14. HTTP – Вікіпедія. URL: https://uk.wikipedia.org/wiki/HTTP (дата звернення: 28.03.2024)

15. AMQP – Вікіпедія. URL: https://uk.wikipedia.org/wiki/AMQP (дата звернення: 29.03.2024)

16. Rabbitmq. URL: https://www.rabbitmq.com/ (дата звернення: 29.03.2024)

17. PostgreSQL – Wikipedia. URL: https://en.wikipedia.org/wiki/PostgreSQL (дата звернення: 29.03.2024)

18. MongoDB – Wikipedia. URL: https://en.wikipedia.org/wiki/MongoDB (дата звернення: 29.03.2024)

19. MACH Alliance – Wikipedia. URL: https://en.wikipedia.org/wiki/MACH_Alliance (дата звернення: 29.03.2024)

20. Redis – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Redis (дата звернення: 29.03.2024)

21. What is Meilisearch? – Meilisearch documentation. URL: https://www.meilisearch.com/docs/learn/what_is_meilisearch/overview (дата звернення: 29.03.2024)

22. What Is Meilisearch & Is It Right For Your Software Project? URL: https://intuji.com/what-is-meilisearch-is-it-right-for-you/ (дата звернення: 29.03.2024)

23. NestJS – Вікіпедія. URL: https://uk.wikipedia.org/wiki/NestJS (дата звернення: 29.03.2024)

24. Introduction to Nest JS | Refine. URL: https://refine.dev/blog/nest-js/#why-nestjs (дата звернення: 30.03.2024)

25. Prisma Documentation. URL: https://www.prisma.io/docs/orm/overview/introduction/what-is-prisma (дата звернення: 30.03.2024)

26. Introduction to Mongoose for MongoDB. URL: https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/ (дата звернення: 30.03.2024)

27. Getting Started with MongoDB & Mongoose | MongoDB. URL: https://www.mongodb.com/developer/languages/javascript/getting-started-with-mongodb-and-mongoose/ (дата звернення: 30.03.2024)

28. ioredis – npm. URL: https://www.npmjs.com/package/ioredis/v/1.3.4 (дата звернення: 30.03.2024)

29. meilisearch – npm. URL: https://www.npmjs.com/package/meilisearch (дата звернення: 30.03.2024)

30. Що таке мікросервісна архітектура: шлях до гнучкого та масштабованого середовища розробки. URL: https://blog.colobridge.net/uk/2024/01/what-is-microservices-architecture-ua/ (дата звернення 09.04.2024)

31. Microservice architecture style. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices (дата звернення 09.04.2024)

32. AMD EPYC™ 7742 Drivers. URL: https://www.amd.com/en/support/downloads/drivers.html/processors/epyc/epyc-7002-series/amd-epyc-7742.html (дата звернення: 09.04.2024)

33. H11DSi-NT | Motherboards | Super Micro Computer, Inc. URL: https://www.supermicro.com/en/products/motherboard/H11DSi-NT (дата звернення: 09.04.2024)

34. Пам`ять серверна Kingston DDR4 64GB 3200 ECC REG RDIMM (KSM32RD4/64HCR) – MOYO. URL: https://www.moyo.ua/ua/pamyat_servera_kingston_ddr4_64gb_3200_ecc_reg_rdimm/550237.html (дата зверення: 09.04.2024)

35. Накопичувач SSD M.2 2280 1TB Samsung – Brain. URL: https://brain.com.ua/ukr/Nakopichuvach_SSD_M2_2280_1TB_Samsung_MZ-V8P1T0BW-p717864.html (дата звернення: 09.04.2024)

36. Накопичувач SSD 2.5" 8TB Samsung – Brain. URL: https://brain.com.ua/ukr/Nakopichuvach_SSD_25_8TB_Samsung_MZ-77Q8T0BW-p701411.html (дата звернення: 09.04.2024)

37. Intel Ethernet Converged Network Adapter X550T2 Product Specifications. URL: https://ark.intel.com/content/www/us/en/ark/products/88209/intel-ethernet-converged-network-adapter-x550-t2.html (дата звернення: 09.04.2024)

38. NVIDIA Tesla V100 | NVIDIA. URL: https://www.nvidia.com/ru-ru/data-center/tesla-v100/ (дата звернення: 09.04.2024)

39. Docker: Accelerated Container Application Development. URL: https://www.docker.com/ (дата звернення: 09.04.2024)

40. Kubernetes. URL: https://kubernetes.io/uk/ (дата звернення: 09.04.2024)

41. bcryptjs – npm. URL: https://www.npmjs.com/package/bcryptjs (дата звернення: 18.05.2024)

42. DTO – Вікіпедія. URL: https://uk.wikipedia.org/wiki/DTO (дата звернення: 18.05.2024)

43. RabbitMQ tutorial – Publish/Subscribe | RabbitMQ. URL: https://www.rabbitmq.com/tutorials/tutorial-three-javascript (дата звернення: 18.05.2024)

44. OptimalBits/Bull: Premium Queue package for handling distributed jobs and messages. URL: https://github.com/OptimalBits/bull/blob/master/REFERENCE.md (дата звернення: 19.05.2024)

45. Queues | NestJS. URL: https://docs.nestjs.com/techniques/queues (дата звернення 19.05.2024)

46. Concurrency in Node.js. URL: https://betterprogramming.pub/concurrency-in-nodejs-40a0469d819f (дата звереннея: 19.05.2024)

47. HLS Streaming: Definition, Usage, Pros & Cons | Okta. URL: https://www.okta.com/identity-101/hls-streaming/ (дата звернення 19.05.2024)

48. Learn SQL: Types of relations. URL: https://www.sqlshack.com/learn-sql-types-of-relations/ (дата звернення: 19.05.2024)

49. What is Endoint? URL: https://www.paloaltonetworks.com/cyberpedia/what-is-an-endpoint (дата звернення: 19.05.2024)

50. Уніфікований локатор ресурсів – Вікіпедія. URL: https://uk.wikipedia.org/wiki/Уніфікований_локатор_ресурсів (дата звернення: 19.05.2024)

51. GET – HTTP | MDN. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET (дата звернення: 20.05.2024)

52. JSON – Вікіпедія. URL: https://uk.wikipedia.org/wiki/JSON (дата звернення: 20.05.2024)

53. POST – HTTP | MDN. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST (дата зверненя: 20.05.2024)

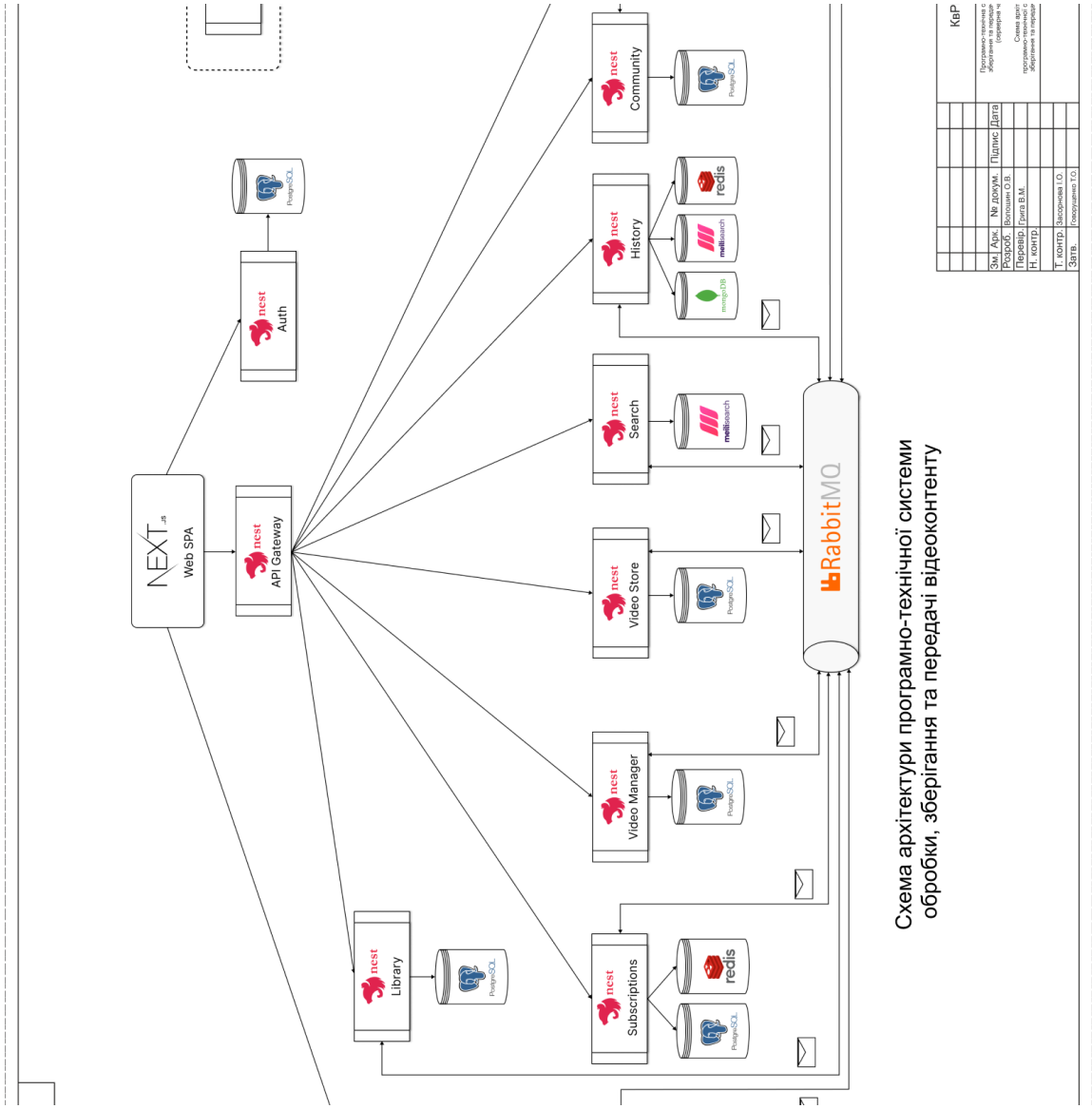
54. Postman API Platform. URL: https://www.postman.com (дата звернення: 20.05.2024)

55. HTTP response status codes – HTTP | MDN. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status (дата звернення: 20.05.2024)

Додаток А

(обов'язковий)

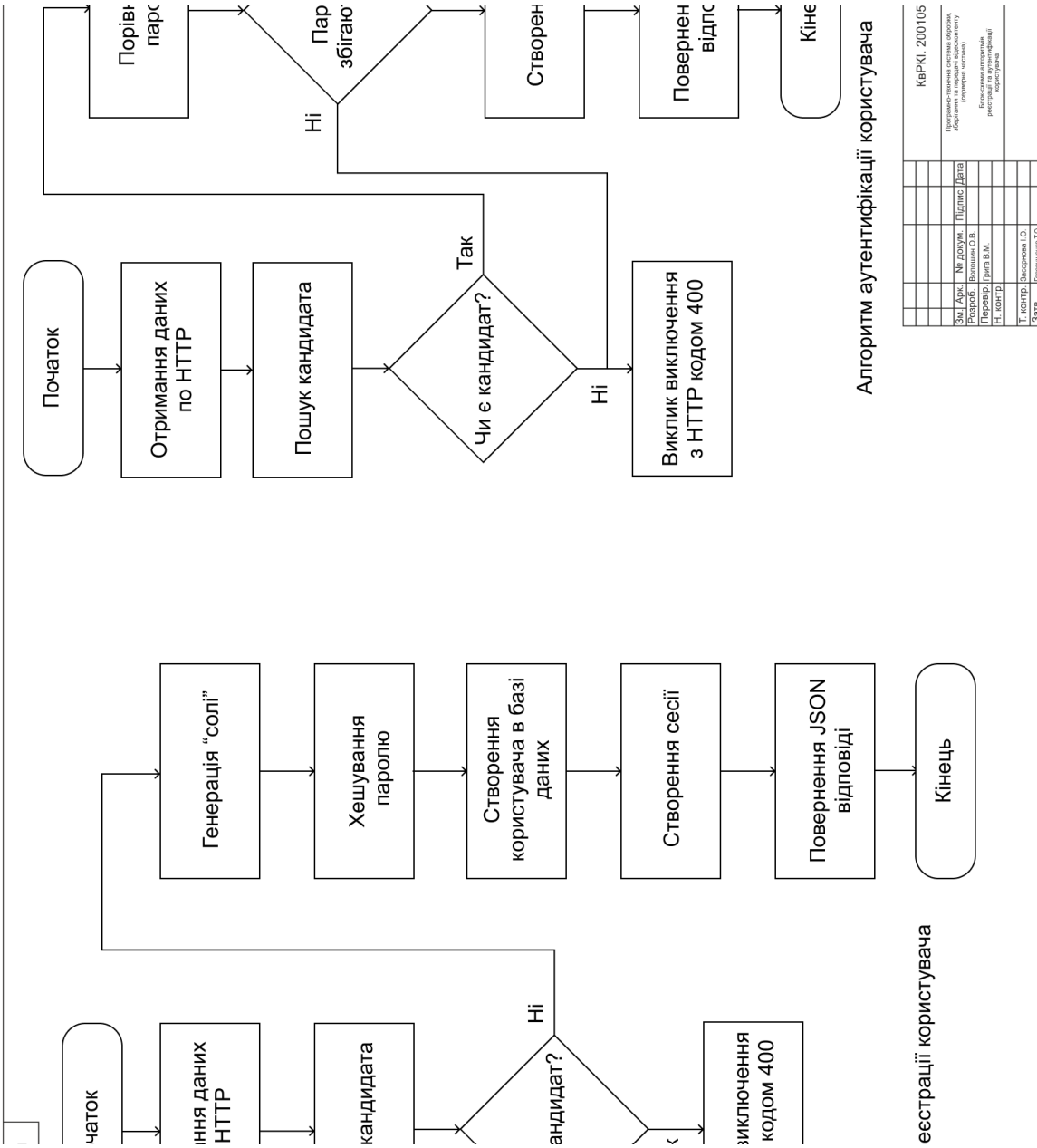
Копія креслення «Схема архітектури програмно-технічної системи обробки, зберігання та передачі відеоконтенту»



Додаток Б

(обов'язковий)

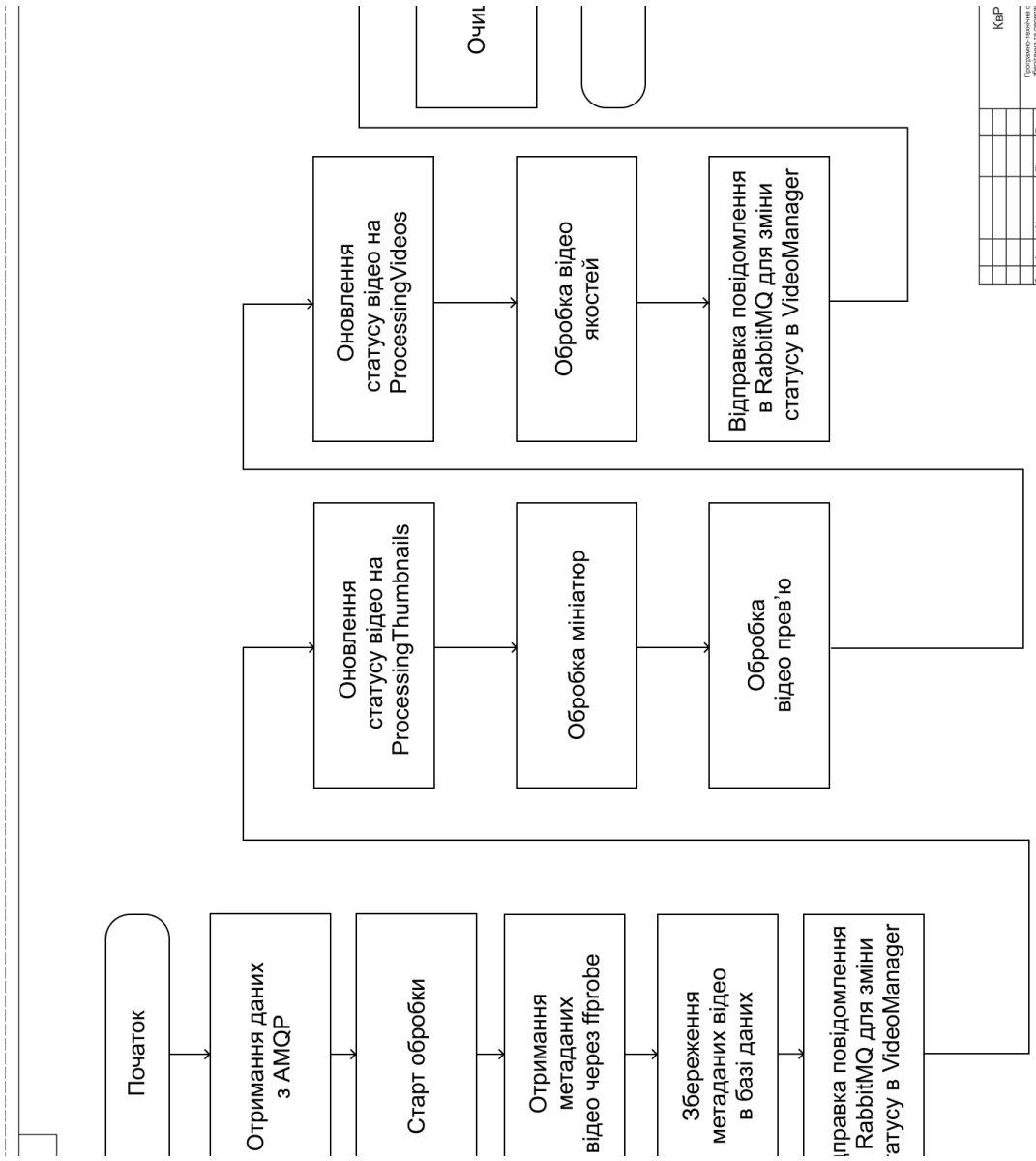
Копія креслення «Блок-схеми алгоритмів реєстрації та аутентифікації користувача»



Додаток В

(обов'язковий)

Копія креслення «Блок-схема роботи алгоритму обробки відео прев'ю, мініатюр та якостей»



Додаток Г

Лістинг коду основних функцій модулю обробки відео

```
private async processVideo(
    payload: VideoProcessPayload,
    filePath: string,
    outputFolderPath: string,
    videoStream: FfprobeStream,
    audioStream: FfprobeStream,
    format: FfprobeFormat,
  ) {
    return new Promise(async (resolve, reject)
=> {
      const steps = RESOLUTIONS.filter((x) =>
x.height <= videoStream.height);
      const processingSteps:
VideoProcessingStep[] = [];
      for (const s of steps) {
        let width = Math.ceil(
          s.height * (videoStream.width /
videoStream.height),
        );
        width = Math.ceil(width / 2.0) * 2.0;

        processingSteps.push(
```

```
        await
this.prisma.videoProcessingStep.create({
          data: {
            videoId: payload.videoId,
            width,
            height: s.height,
            label: s.label,
            bitrate: s.bitrate,
            complete: false,
          },
        }),
      );
    }

    await this.cacheManager.set(`v-
${payload.videoId}-status`, 'work');
    const playlistProcessingSteps:
PlaylistProcessingStep[] = [];

    for (const s of processingSteps) {
      const videoProcessorStatus = await
this.cacheManager.get(
        `v-${payload.videoId}-status`,
      );
      if (videoProcessorStatus === 'canceled') {
        return;
      }

      try {
        await new Promise(async (resolve,
reject) => {
          const hlsFolder =
join(outputFolderPath, s.label);
          await mkdir(hlsFolder, { recursive: true
});

          const cmd =
this.ffmpegService.ffmpeg(filePath);
          cmd
.inputOption(this.ffmpegService.buildInputOpt
ion())
            .outputOption(

this.ffmpegService.buildTranscodeOutputOptio
n(
              s.width,
              s.height,
              s.bitrate,
              !!audioStream,
              hlsFolder,
              s.label,
            ),
            )
            .output(join(hlsFolder,
`${s.label}.m3u8`))
            .on('error', (err: any) => {
              reject(err);
            })
            .on('end', async () => {
              this.logger.log(`HLS for ${s.label}
generated, uploading...`);

              const hlsId = await
this.networkService.uploadHls(
                hlsFolder,
                s.videoId,
              );

              this.logger.log(`HLS ${s.label}
uploaded to storage.`);
              this.logger.log(
                `Emit add_processed_video for
video (${s.videoId})`,
              );

              const lengthSeconds =
                s.label === '144p'
```

```
              ? Math.floor(
                  Number(
                    format?.duration ||
                      videoStream?.duration ||
                      audioStream?.duration,
                  ) || 0,
                )
              : null;

            this.videoManagerClient.emit(
              'add_processed_video',
              new
AddProcessedVideoEvent(s.videoId, s.label,
lengthSeconds),
            );

            playlistProcessingSteps.push({
              ...s,
              hlsId,
            });

            const masterFilePath = await
this.hlsService.writePlaylist(
              outputFolderPath,
              playlistProcessingSteps,
            );
            await
this.networkService.uploadHlsMaster(
              masterFilePath,
              payload.videoId,
            );

            await
this.prisma.videoProcessingStep.update({
              where: { id: s.id },
              data: { complete: true },
            });

            this.logger.log(`Step ${s.label} is
complete`);

            resolve(true);
          })
          .run();

        this.ffmpegService.setCommand(`v-
${s.videoId}-${s.label}`, cmd);
      });
    } catch (e) {
      this.logger.error(e);
      reject(e);
    } finally {
      this.ffmpegService.deleteCommand(`v-
${s.videoId}-${s.label}`);
    }
  }

  resolve(true);
 });
}

private async processPreview(
  payload: VideoProcessPayload,
  filePath: string,
  folderId: string,
  videoStream: FfprobeStream,
  format: FfprobeFormat,
) {
  return new Promise(async (resolve, reject)
=> {
    try {
      const thumbnailId = randomUUID();
      const outputThumbnailPath = join(
        process.cwd(),
        'processor_output',
        folderId,
        thumbnailId + '.webp',
```

```
      );

      const height = Math.min(
        videoStream.height,
PREVIEW_CONFIG.previewThumbnailHeigh
t,
      );
      let width = Math.ceil(
        height * (videoStream.width /
videoStream.height),
      );
      width = Math.ceil((width / 2.0) * 2.0);

      let startPositionSeconds =
        format.duration *
PREVIEW_CONFIG.previewThumbnailStartP
osition;
      const lengthSeconds = Math.min(
        +videoStream.duration,
PREVIEW_CONFIG.previewThumbnailLengt
hSeconds,
      );
      if (
        +videoStream.duration -
startPositionSeconds <
PREVIEW_CONFIG.previewThumbnailLengt
hSeconds
      ) {
        startPositionSeconds = 0.0;
      }

      await new Promise((resolve, reject) => {
        const cmd =
this.ffmpegService.ffmpeg(filePath);
        cmd
.inputOption(this.ffmpegService.buildInputOpt
ion())
          .outputOption(
this.ffmpegService.buildPreviewOutputOption(
            width,
            height,
            startPositionSeconds,
            lengthSeconds,
          ),
        )
          .output(outputThumbnailPath)
          .on('error', (err: any) => {
            reject(err);
          })
          .on('end', async () => {
            this.logger.log(
              `Video preview (${thumbnailId})
processing finished, uploading to storage...`,
            );
            const { id, url } = await
this.networkService.uploadImage(
              outputThumbnailPath,
              payload.videoId,
SERVICE_UPLOADED_THUMBNAIL,
            );
            this.logger.log(
              `Video preview to video
(${payload.videoId}) uploaded to storage.`,
            );
            this.logger.log(`Emit add_preview to
video (${payload.videoId})`);
            this.videoManagerClient.emit(
              'add_preview',
              new AddPreviewEvent(id, url,
payload.videoId),
            );
```

```
          resolve(true);
        })
        .run();

      this.ffmpegService.setCommand(`v-
${payload.videoId}-preview`, cmd);
    });
    resolve(true);
   } catch (e) {
    this.logger.error(e);

this.videoManagerClient.emit('preview_generat
e_failed', payload);
    reject(e);
   } finally {
    this.ffmpegService.deleteCommand(`v-
${payload.videoId}-preview`);
   }
  });
 }

 private async processThumbnail(
  payload: VideoProcessPayload,
  filePath: string,
  folderId: string,
  videoStream: FfprobeStream,
 ) {
  return new Promise(async (resolve, reject)
=> {
   try {
    const outputFilePath = join(
     process.cwd(),
     'processor_output',
     folderId,
    );
    const height =
Math.min(videoStream.height, 360);
    let width = Math.ceil(
      height * (videoStream.width /
videoStream.height),
    );
    width = Math.ceil((width / 2.0) * 2.0);

    await new Promise((resolve, reject) => {
     const cmd =
this.ffmpegService.ffmpeg(filePath);
     cmd
      .on('error', (err: any) => {
       reject(err);
      })
      .on('end', async () => {
       this.logger.log(
        `Thumbnails for video
(${payload.videoId}) generated to folder
(${folderId}), uploading to storage...`,
       );

       const thumbnails: any[] = [];
       for (let i = 1; i <= 3; i++) {
        const filePath = join(outputFilePath,
`tn_${i}.png`);
        const { id, url } = await
this.networkService.uploadImage(
         filePath,
         payload.videoId,

SERVICE_UPLOADED_THUMBNAIL,
        );
        thumbnails.push({ imageFileId: id,
url });
       }

       this.logger.log(`Thumbnails uploaded
to storage.`);
       this.logger.log(
        `Emit add_thumbnails to video
(${payload.videoId})`,
```

```
            );

            this.videoManagerClient.emit(
              'add_thumbnails',
              new
AddThumbnailEvent(payload.videoId,
thumbnails),
            );

            resolve(true);
          })
          .thumbnails({
            count: 3,
            folder: outputFilePath,
            filename: 'tn_%i.png',
            size: `${width}x${height}`,
            timemarks: ['10%', '25%', '50%'],
          });

        this.ffmpegService.setCommand(`v-
${payload.videoId}-thumbnails`, cmd);
      });
    } catch (e) {
      this.logger.error(e);
this.videoManagerClient.emit('thumbnails_gen
erate_failed', payload);
      reject(e);
    } finally {
      this.ffmpegService.deleteCommand(`v-
${payload.videoId}-thumbnails`);
    }
    resolve(true);
  });
 }

 private async updateVideoStatus(
  videoId: string,
  status: VideoProcessingStatus,
 ) {
  await this.prisma.video.update({
    where: { id: videoId },
    data: {
      status,
      processedAt: status === 'Processed' ? new
Date() : undefined,
    },
  });
 }
```

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016353978

Дата перевірки:
13.06.2024 00:14:48 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.06.2024 06:28:38 EEST

ID користувача:
100005591

Назва документа: Волошин_Програмно-технічна система обробки, зберігання та передачі відеоконтенту (сер...

Кількість сторінок: 76 **Кількість слів:** 13264 **Кількість символів:** 105682 **Розмір файлу:** 3.77 MB **ID файлу:** 1016157925

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

4.18%
Схожість

Найбільша схожість: 0.74% з джерелом з Бібліотеки (ID файлу: 1016157929)

3.73% Джерела з Інтернету	354	Сторінка 78
1.88% Джерела з Бібліотеки	148	Сторінка 81

0.2% Цитат

Цитати	1	Сторінка 82

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1
Підозріле форматування	21 сторінка

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 0.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 18%**

ID: 130047 Назва: БКР Програмно-технічна система обробки, зберігання та передачі відеоконтенту (серверна частина) Додано в БД: 2024-06-12 Автора: О.В. Волошин Керівники: В.М. Грига Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	89848	802	786 (1%)	11 (1%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Волошин Олег Вікторович

Тема: Програмно-технічна система обробки, зберігання та передачі відеоконтенту (серверна частина)

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3___ Кількість сторінок записки ___71___

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка та дослідження серверної частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі кваліфікаційної роботи проведено огляд існуючих рішень для роботи з відеоконтентом, що включав визначення переваг та недоліків кожної системи. Було досліджено основні технології зберігання та обробки відео, зокрема використання хмарних сервісів, кодеків, форматів та транскодингу відео. Описано процес трансляції відео та використання CDN, а також проведено порівняння аналогів за характеристиками, такими як швидкість завантаження, обробки та передачі відео, масштабованість, безпека та надійність. У другому розділі кваліфікаційної роботи обґрунтовано вибір архітектури системи та програмних і апаратних засобів для її реалізації. Детально описано елементи архітектури, що дозволило визначити оптимальну структуру та взаємодію компонентів. Було обрано платформу та цільову операційну систему, які забезпечують належну інтеграцію та стабільність роботи. Обрано бази даних для зберігання та обробки даних, а також визначено фреймворки і бібліотеки для швидкої розробки та високої продуктивності системи. Проведено аналіз апаратних засобів, включаючи вибір серверного обладнання, процесорів,

оперативної пам'яті та накопичувачів. У третьому розділі кваліфікаційної роботи описано програмну реалізацію серверної частини системи. Розглянуто основні модулі програмного забезпечення. Описано процес створення баз даних, включаючи структуру та організацію даних для ефективного зберігання та швидкого доступу до відеоконтенту. Розроблено пошукові можливості системи, що дозволяють користувачам швидко знаходити необхідний контент за різними критеріями. Проведено тестування системи, включаючи перевірку функціональності, стабільності та продуктивності всіх модулів, що забезпечило високу якість та надійність роботи системи в реальних умовах експлуатації.

4. Позитивною стороною роботи є висока точність. Система забезпечує високу ефективність обробки, зберігання та передачі відеоконтенту. Система дозволяє швидко та надійно виконувати операції з відео завдяки оптимізованим алгоритмам та використанню сучасних технологій зберігання та передачі даних.

5. Негативні сторони роботи: Складність інтеграції: Незважаючи на високу ефективність системи, її інтеграція в існуючу інфраструктуру може бути складною та вимагати додаткових ресурсів і часу. Це може ускладнити впровадження для організацій з обмеженими технічними можливостями.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно з діючими стандартами оформлення документації. Графічні матеріали чітко ілюструють основні положення роботи, сприяючи кращому розумінню представленої інформації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Федущак М.В. к.т.н., доцент кафедри ПЗЗ

"14" червня 2024 р. _____(підпис)

Завідувачу кафедри КІС
д-р.техн.наук, проф. Говорущенко Т. О.

Волошина Олега Вікторовича
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

11 червня 2024 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Програмно-технічна система обробки, зберігання та передачі відеоконтенту (серверна частина)

Автор: Волошин Олег Вікторович

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Грига Володимир Михайлович, д.т.н, професор.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;
4) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 4.18% і адресується до 502 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В. М. Грига

Гарант ОП _____ С. М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко