НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри

_____ _____
(підпис)

“__” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”

на тему: Платформа для відеохостингу ShowMe

Виконав: студент __4__ курсу, групи ІО-03
(шифр групи)

_____ _____ _Бондаренко Роман Ігорович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент, Череватенко О.В.__ _____ _____ __
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент Іваніщев Б. В. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент __ доц. каф. ІСТ, ФІОТ, к.т.н. Коган А. В. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Комп’ютерні системи та мережі”

спеціальності 123 “Комп’ютерна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис)

“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Бондаренка Романа Ігоровича

1. Тема проєкту *Платформа для відеохостингу ShowMe*
керівник проєкту Череватенко Олексій Володимирович, асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 27.05 2024 року № 2112-с
2. Термін здачі студентом закінченого проєкту 14.06.2024 .
3. Вихідні дані до проєкту *технічна документація. теоретичні дані.*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Дослідження наявних аналогів.

Розділ 2. Дослідження та вибір технологій розробки.

Розділ 3. Проектування веб-застосунку.

Розділ 4. Представлення та аналіз веб-застосунку.

5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень).
Схема принципова (архітектурна схема), Схема структурна (схема бази даних), Схема функціональна (діаграма класів).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Іваніщев Б. В.		

7. Дата видачі завдання 1 грудня 2023 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2023-15.12.2023*	
2.	*Вивчення та аналіз завдання*	*15.12.2023-15.03.2024*	
3.	*Розробка архітектури та загальної структури системи*	*15.03.2024-25.03.2024*	
4.	*Розробка структур окремих підсистем*	*25.03.2024-5.04.2024*	
5.	*Програмна реалізація системи*	*5.04.2024-15.04.2024*	
6.	*Оформлення пояснювальної записки*	*15.04.2024-20.05.2024*	
7.	*Захист програмного продукту*	*25.04.2024*	
8.	*Передзахист*	*23.05.2024*	
9.	*Захист*	*17.06.2024*	

Студент _____ Роман БОНДАРЕНКО
(підпис)

Керівник проєкту _____ Олексій ЧЕРЕВАТЕНКО
(підпис)

АННОТАЦІЯ

Дослідження, проведене в рамках бакалаврської роботи, було присвячено створенню нового веб-сервісу під назвою «ShowMe», який є платформою для розміщення відеофайлів. Таким чином, різні сфери, такі як освіта, розваги, інформація, комерція та творчість, можуть використовувати цей сервіс для створення, обміну або взаємодії з відеоконтентом.

Для реалізації ShowMe застосовували сучасні технології, такі як Java, Spring Boot, MySQL, React, Docker, Maven та AWS. Java і Spring Boot забезпечують надійність і швидкість серверного додатку. MySQL використовується для зберігання даних про відео та користувачів. Клієнтський інтерфейс розроблено на React, що полегшує процес розробки. Docker спрощує налаштування та управління середовищем, а Maven автоматизує збірку проєктів. AWS забезпечує масштабованість та високу доступність веб-сервісів.

Ключові слова: відеохостинг, масштабованість, контейнеризація, Java, Spring Boot, AWS.

ANNOTATION

The research conducted for the bachelor's thesis focused on creating a new web service called "ShowMe," a platform for hosting video files. Thus, various fields such as education, entertainment, information, commerce, and creativity can use this service for creating, sharing, or interacting with video content.

To implement ShowMe, modern technologies such as Java, Spring Boot, MySQL, React, Docker, Maven, and AWS were used. Java and Spring Boot ensure the reliability and speed of the server application. MySQL is used for storing data about videos and users. The client interface was developed using React, which simplifies the development process. Docker facilitates environment setup and management, while Maven automates project builds. AWS ensures scalability and high availability of web services.

Keywords: video hosting, scalability, containerization, Java, Spring Boot, AWS.

Справки	Формат	*Занчення*	*Найменування*	Кіл. листів	№ екземпляра	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	**ІАЛЦ.467200.002 ТЗ**	Додаток для	4		
			відеохостигну ShowMe			
			Технічне завдання			
	А4	**ІАЛЦ.467200.003 ПЗ**	Додаток для	69		
			відеохостигну ShowMe			
			Пояснювальна записка			
	А4	**ІАЛЦ.467200.004 Д1**	Додаток для	1		
			відеохостигну ShowMe			
			Структурна схема			
			системи			
	А4	**ІАЛЦ.467200.005 Д2**	Додаток для	1		
			відеохостигну ShowMe			
			Структурна схема			
			Бази даних			
	А4	**ІАЛЦ.467200.006 Д3**	Додаток для	1		
			відеохостигну ShowMe			
			Функціональна схема			
			(діаграма класів)			
	А4	**ІАЛЦ.467200.007 Д4**	Додаток для	39		
			відеохостигну ShowMe			
			Лістинг програмного коду			

					ІАЛЦ.467200.001 ОА			
Зм	***Лист***	***№ докум.***	***Підп***	***Дата***				
Розроб		Бондаренко Р. І.			*Платформа для відеохостингу ShowMe*	Літ.	Аркуш	Аркушів
Перев		Череватенко О. В.					1	1
					Опис альбому	***КПІ ім. І. Сікорського, ФІОТ, Група ІО-03***		

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Платформа для відеохостингу ShowMe*»

Київ – 2024

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ..... 2
2. ПІДСТАВИ ДЛЯ РОЗРОБКИ..... 2
3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ..... 2
4. ДЖЕРЕЛА РОЗРОБКИ..... 3
5. ТЕХНІЧНІ ВИМОГИ..... 3
5.1. Вимоги до розробленого продукту..... 3
5.2. Вимоги до програмного забезпечення.....4
5.3. Вимоги до апаратної частини.....4
6. ЕТАПИ РОЗРОБКИ.....4

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Це технічне завдання передбачає створення інноваційного сервісу, який дозволить користувачам створювати, ділитися та спілкуватися за допомогою відеоконтенту у різних сферах: навчання, розваги, інформація, комерція та творчість. Вона створена для того, щоб забезпечити зручну та ефективну платформу для обміну ідеями, знаннями та враженнями через відео.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр комп'ютерної інженерії», який був затверджений факультетом "Інформатики та обчислювальної техніки" кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3. МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням розробки є надати зручний та інтуїтивно простий інтерфейс для завантаження, перегляду та поширення відео на будь-яку тематику за рахунок реалізації відповідного веб-застосунку для хостингу відео.

4. ДЖЕРЕЛА РОЗРОБКИ

Основою для розробки даного дипломного проекту є безкоштовні публікації та статті в мережі Інтернет, офіційна документація та науково-технічна література, які виступають як джерела інформації та сприяють поглибленню знань у вибраній темі.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Система має відповідати наступним вимогам:

- Зрозумілий та зручний інтерфейс.
- Реєстрація та авторизація користувачів.
- Можливість завантажувати відео на платформу.
- Можливість переглядати відео.
- Пошук, сортування та фільтрація відео.
- Уподобання та коментування відео.
- Підписка на інших користувачів з метою сповіщення про завантаження нових відео.
- Масштабування задля підтримки зростаючої кількості користувачів.
- Захист додатку від різних кібератак.

5.2. Вимоги до програмного забезпечення

- Операційна система Windows, Linux або MacOS
- IDE IntelliJ IDEA Ultimate 2024.1 або вище.

5.3. Вимоги до апаратної частини

- ЦП не нижче за Intel® Core (TM) i3-2100T або AMD Athlon II X3 425.
- ROM не менше ніж 240 ГБ.
- RAM не менше ніж 8 ГБ.

6. ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	15.12.2023
Вивчення та аналіз завдання	15.01.2024
Розробка архітектури та загальної структури системи	14.02.2024
Розробка структур окремих частин системи	01.05.2024
Програмна реалізація системи	10.05.2024
Виправлення помилок	20.05.2024
Оформлення пояснювальної записки	1.06.2024

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Платформа для відеохостингу ShowMe*»

Київ – 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ.....3
ВСТУП.....4
РОЗДІЛ 1. ДОСЛІДЖЕННЯ НАЯВНИХ АНАЛОГІВ.....5
1.1 Актуальність та доцільність проєкту.....5
1.2 Огляд наявних альтернатив.....6
1.2.1 YouTube.....7
1.2.2 Vimeo.....11
1.2.3 Dailymotion.....14
1.3 Side-to-side порівняння.....17
ВИСНОВОК ДО РОЗДІЛУ 1.....19
РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТА ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ.....21
2.1 Архітектура IT систем.....21
2.1.1 Моноліт.....21
2.1.2 Мікросервіси.....23
2.2 Хмарні середовища.....25
2.2.1 AWS.....26
2.2.2 Microsoft Azure.....27
2.3 Вибір технологій розробки.....28
2.3.1 Java.....28
2.3.2 Spring Boot.....30
2.3.3 MySQL.....32
2.3.4 React.....34
2.3.5 Maven.....35
2.3.6 Docker.....36
ВИСНОВОК ДО РОЗДІЛУ 2.....37

РОЗДІЛ 3. ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ 39
3.1 Архітектура застосунку 39
3.1.1 Сервер (бекенд) 39
3.1.2 Клієнт (фронтенд) 48
3.1.3 База даних 51
ВИСНОВОК ДО РОЗДІЛУ 3 53
РОЗДІЛ 4. ПРЕДСТАВЛЕННЯ ТА АНАЛІЗ ВЕБ-ЗАСТОСУНКУ 54
4.1 Демонстрація функціональності 54
4.2 Розвиток проекту 65
ВИСНОВОК ДО РОЗДІЛУ 4 65
ВИСНОВКИ 66
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 68
Додаток 1 70
Додаток 2 72
Додаток 3 74
Додаток 4 76

ПЕРЕЛІК СКОРОЧЕНЬ

HTTP	HyperText Transfer Protocol (Протокол передачі гіпертексту)
API	Application Programming Interface (Інтерфейс програмування додатків)
ORM	Object Relational Mapping (Об'єктно реляційне відображення)
REST	Representational State Transfer (Передача репрезентативного стану)
JWT	JSON Web Token (JSON веб токен)
AWS	Amazon Web Services (Веб сервіси Amazon)
HTML	HyperText Markup Language (Мова гіпертекстової розмітки)
CSS	Cascading Style Sheets (Каскадні таблиці стилів)

ВСТУП

У сучасному світі відеохостинг став одним із найпопулярніших методів обміну інформацією та розваг. Платформи для відеохостингу, такі як YouTube, Vimeo та Dailymotion, налічують мільйони користувачів, які щодня завантажують, переглядають та коментують мільярди годин відеоконтенту.

Стрімке зростання популярності відеоплатформ можна пояснити кількома факторами: доступністю, простотою використання, різноманітністю контенту та інтерактивністю.

Метою даної дипломної роботи є розробка платформи для відеохостингу під назвою "ShowMe". Платформа буде призначена для надання користувачам можливості завантажувати, переглядати та коментувати відеоконтент.

В результаті дослідження буде розроблена платформа для відеохостингу "ShowMe", яка буде відповідати всім вимогам та функціям, визначеним на етапі аналізу. Платформа буде проста у використанні та буде пропонувати користувачам широкий спектр можливостей для завантаження, перегляду та коментування відеоконтенту.

Розроблена платформа для відеохостингу "ShowMe" може бути використана для різних цілей, включаючи навчання, маркетинг, рекламу, навчання співробітників, розваги та соціальні мережі. Платформа може бути корисною для користувачів будь-якого віку та професії.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ НАЯВНИХ АНАЛОГІВ

На даний момент існує безліч платформ для відеохостингу, які пропонують користувачам широкий спектр можливостей для завантаження, перегляду та коментування відеоконтенту. Деякі з найпопулярніших платформ включають YouTube, Vimeo та Dailymotion.

Однак, незважаючи на різноманітність існуючих платформ, багато користувачів стикаються з труднощами при їх використанні. Деякі платформи, такі як YouTube, можуть бути занадто складними та перевантаженими функціями, що ускладнює пошук потрібного відеоконтенту та його перегляд. Інші платформи, такі як Vimeo, можуть бути більш орієнтовані на професійних користувачів і не пропонувати достатньо функцій, які б задовольнили потреби звичайних глядачів.

Метою даного розділу є дослідження існуючих аналогів платформи для відеохостингу "ShowMe" та визначення їхніх сильних та слабких сторін. Це допоможе нам розробити платформу, яка буде відповідати потребам користувачів та пропонувати їм простий та зручний інтерфейс.

1.1 Актуальність та доцільність проєкту

Відеоконтент стає неймовірно популярним для широкого спектру цілей; від надання інформації та освіти до розваги людей або спілкування з ними. Як видно з популярності YouTube, Vimeo та інших сайтів відеохостингу, цю тенденцію підживлює цифрова ера, коли користувачі хочуть виражати себе за допомогою відео та споживати вміст у цьому форматі.

Проєкт "ShowMe" є актуальним з кількох причин:

- Щороку все більше людей покладаються на відеоконтент для навчання, роботи, розваг і спілкування. ShowMe створено для задоволення цього попиту, надаючи платформу, де користувачі можуть легко створювати, зберігати та ділитися відео.
- Поточні послуги відеохостингу мають обмеження, які перешкоджають їх використанню деякими людьми чи організаціями за певних обставин. Однак «ShowMe» можна буде адаптувати для різних сфер, таких як освіта, бізнес, розваги, а також творчість, що розширює його привабливість для багатьох користувачів.
- Важливість відеоконтенту в сучасній комунікації. Відео є дуже потужним засобом комунікації, який дозволяє передати більше інформації за короткий час, викликаючи при цьому вищі емоційні реакції аудиторії. ShowMe допомагає покращити комунікацію в різних секторах, включаючи освіту та маркетинг.

1.2 Огляд наявних альтернатив

На даний момент існує безліч платформ для відеохостингу, які пропонують користувачам широкий спектр можливостей для завантаження, перегляду та коментування відеоконтенту. Ми розглянемо три найпопулярніші.

YouTube: Найвідоміша платформа для відеохостингу з мільярдами активних користувачів щомісяця. Пропонує широкий спектр відеоконтенту на будь-яку тему, від музичних відео та розваг до новин та навчальних матеріалів.

Vimeo: Платформа, орієнтована на професійних користувачів та творців контенту. Пропонує високоякісне відео, інструменти для редагування та аналітики, а також можливість монетизації контенту.

Dailymotion: Міжнародна платформа для відеохостингу з великою кількістю користувачів у Європі. Пропонує широкий спектр відеоконтенту, включаючи телешоу, фільми та музичні відео.

Розглянемо і дослідимо кожен із них. Визначимо їх переваги та недоліки. Проаналізуємо всі сильні сторони щоб застосувати їх у ShowMe.

1.2.1 YouTube

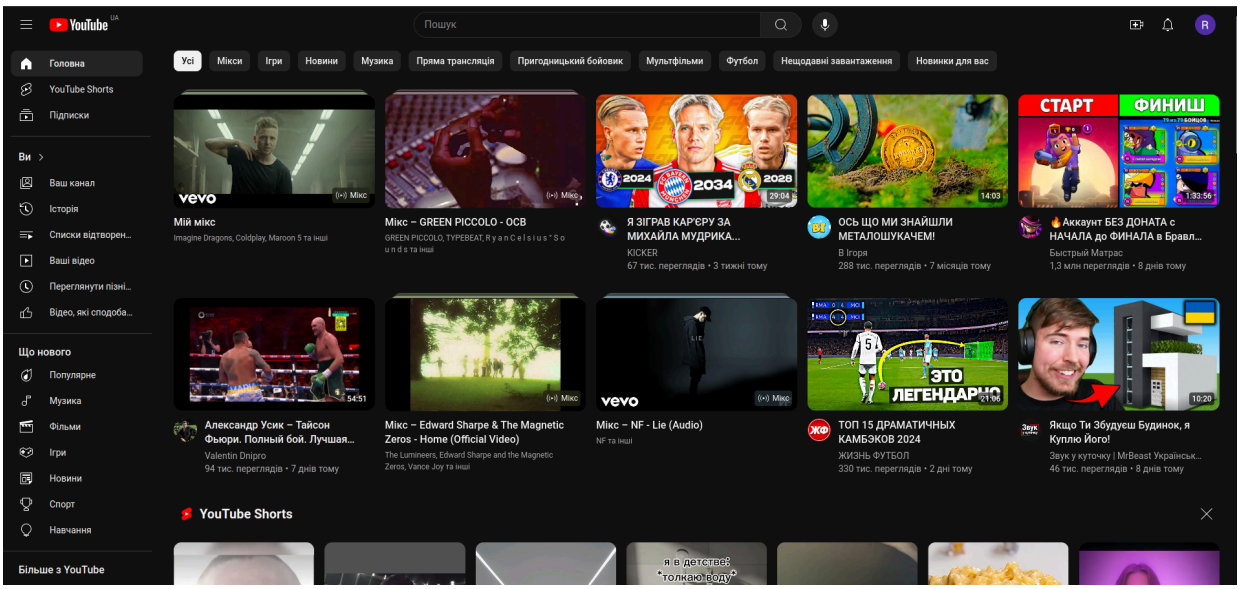


Рис. 1.1 - Домашня сторінка YouTube

YouTube - це безкоштовна платформа для відеохостингу, що належить Google, де користувачі можуть завантажувати, переглядати та коментувати відео. Заснована в 2005 році, YouTube став одним з найпопулярніших веб-сайтів у світі, з мільярдами активних користувачів щомісяця.

YouTube пропонує дуже багато функцій, розглянемо коротко кожну з них:

- Завантаження відео: Користувачі можуть завантажувати відео з будь-якого пристрою, підключеного до Інтернету. Підтримуються різні формати та розширення, включаючи MP4, MOV, FLV, AVI та WMV. Максимальна тривалість відео становить 12 годин, але для верифікованих користувачів доступна опція завантаження необмежених за часом відео.

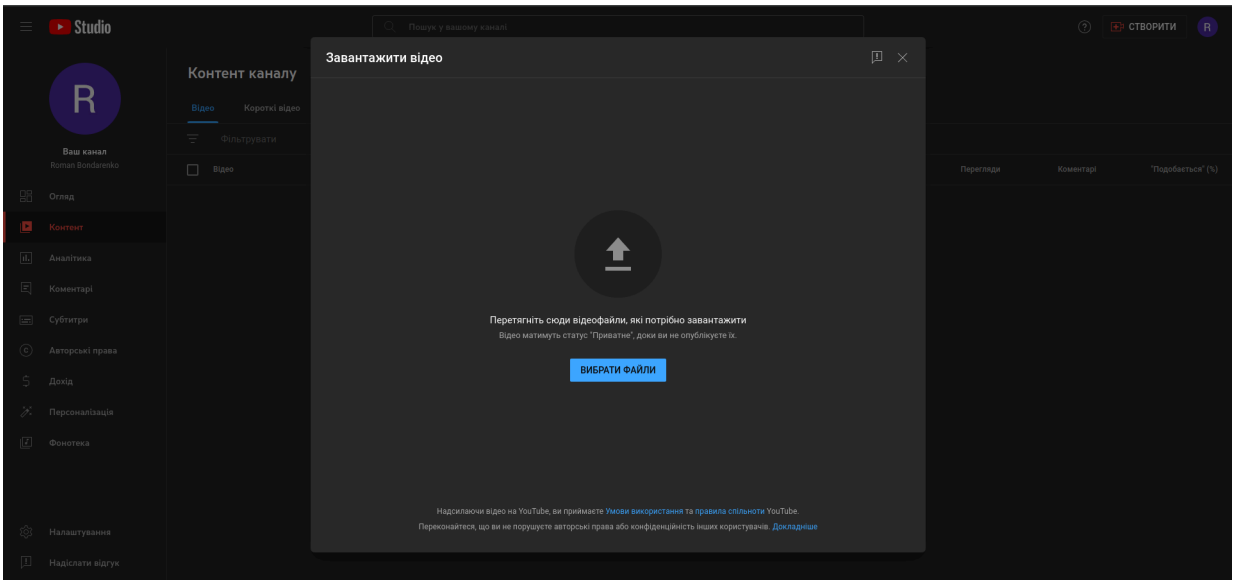


Рис. 1.2 - Вікно завантаження відео на YouTube

- Перегляд відео: Користувачі можуть переглядати мільярди відео, доступних на платформі. Відео можна шукати за ключовими словами, тегами, категоріями, каналами, назвами та іншими параметрами. Доступні різні режими перегляду, включаючи стандартний, HD, 4K та VR.
- Коментування та оцінювання відео: Користувачі можуть залишати коментарі та оцінювати відео за п'ятизірковою шкалою. Це дозволяє їм ділитися своїми враженнями та допомагати іншим користувачам знаходити якісні відео.
- Підписка на канали: Користувачі можуть підписуватися на канали, які їм подобаються, щоб отримувати сповіщення про нові відео, що публікуються. Це дозволяє їм бути в курсі контенту, який їм цікавий.

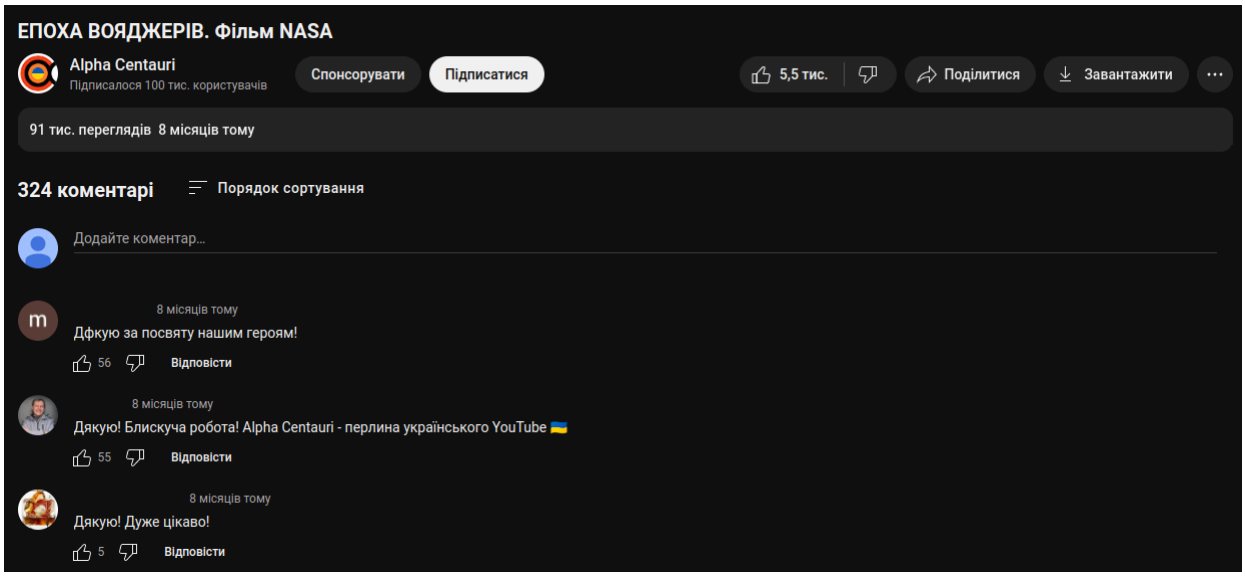


Рис. 1.3 - Секція коментування, оцінювання відео та підписки на канал

- Створення власних каналів: Користувачі можуть створювати власні канали, щоб завантажувати та публікувати свої відео. Це може бути чудовим способом поділитися своїми знаннями, ідеями та творчістю з іншими людьми.
- Прямі трансляції: Користувачі можуть транслювати відео в прямому ефірі з будь-якого місця. Це дозволяє їм спілкуватися з аудиторією в режимі реального часу та ділитися подіями, що відбуваються.
- Монетизація відео: Власники каналів YouTube можуть монетизувати свої відео за допомогою реклами, партнерських програм та інших методів. Це дозволяє їм отримувати дохід від свого контенту.
- Доступність: YouTube доступний через веб-браузер, мобільні додатки та смарт-телевізори.
- Рекомендації: YouTube використовує алгоритми машинного навчання, щоб рекомендувати користувачам відео, які їм можуть бути цікаві. Це ґрунтується на історії переглядів, підписках, лайках, коментарях та інших факторах.

Як ми бачимо, YouTube пропонує величезний список можливостей, який з одного боку, вражає, а з іншого лякає своєю кількістю і може збити з пантелику нових користувачів. Щодня розробники працюють над розширенням можливостей та функцій YouTube.

1.2.2 Vimeo

Vimeo – це платформа для відеохостингу, яка була заснована у 2004 році. Вона призначена для професійних творців відеоконтенту, кінематографістів, музикантів, рекламних агенцій та бізнесів, які потребують високоякісних відео рішень.

Що стосується функціональності, то вона дещо менша ніж у YouTube, але досі надає безкоштовний доступ до завантаження, перегляду, оцінки та коментування відео.

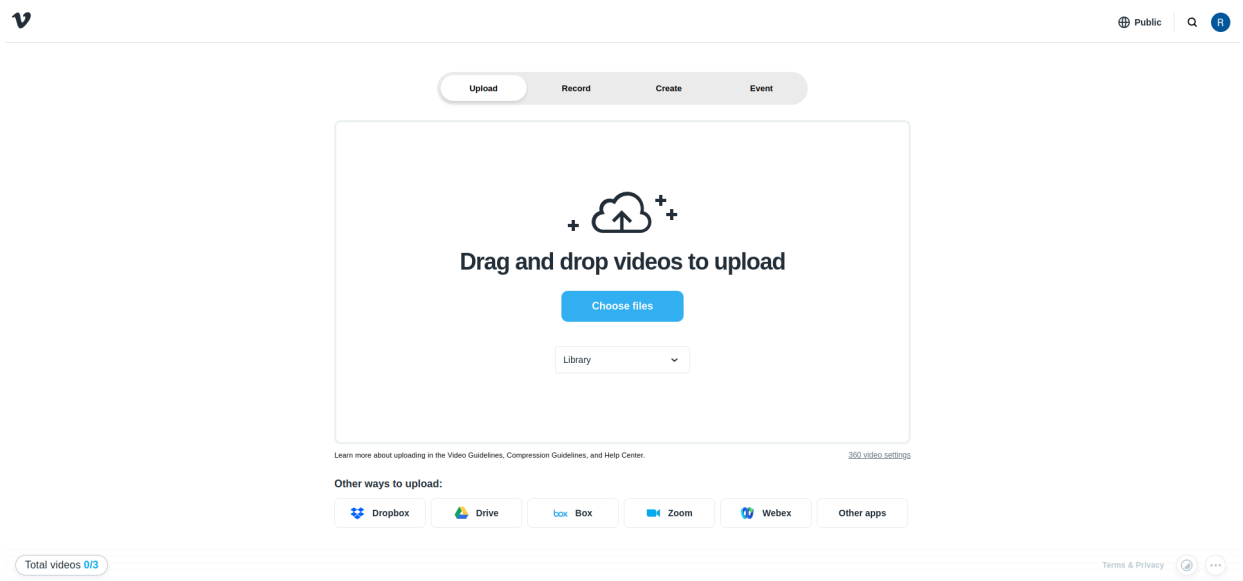


Рис. 1.4 - Вікно завантаження відео на Vimeo

Значною перевагою Vimeo є те, що відео можна завантажити не тільки із комп’ютера, а й з хмарних середовищ зберігання файлів. Також на Vimeo можна безпосередньо створювати відео, чого не дозволяє YouTube.

Недоліком Vimeo у порівнянні з YouTube є не дуже інтуїтивний інтерфейс в плані пошуку та перегляду відео.

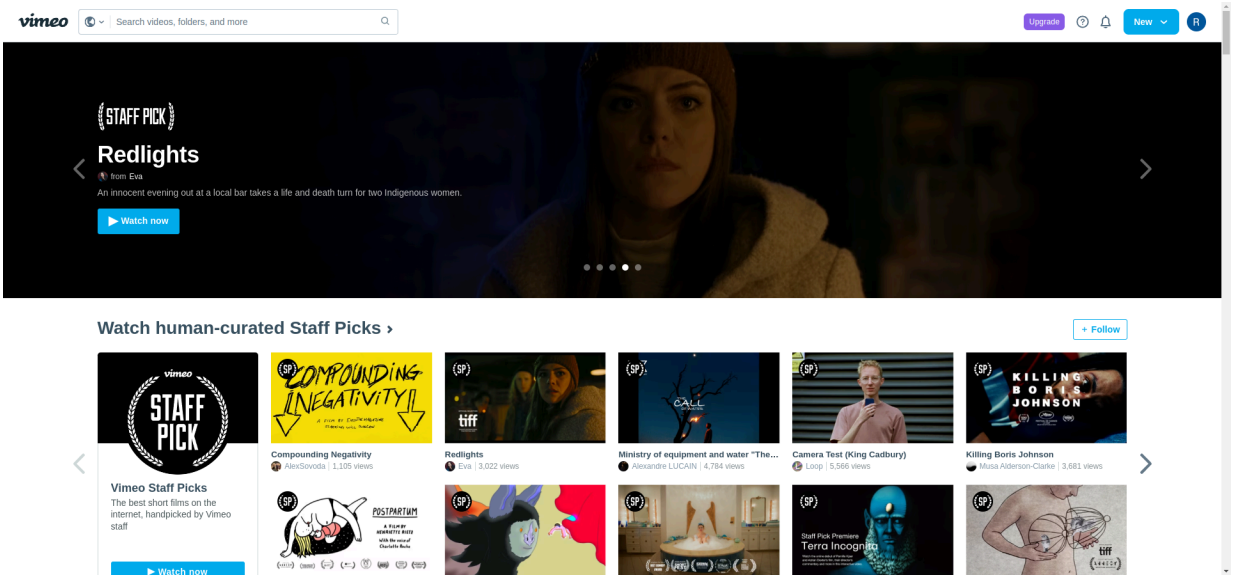


Рис. 1.5 - Сторінка зі списком відео на Vimeo

Тут відео поділені по категоріям, що є плюсом, але на YouTube це реалізовано простіше та інтуїтивно зрозуміліше.

Розглянемо сторінку відтворення відео. Зразу можна відмітити дуже гарну якість самого відео, зручність плеєру, та елементів навігації. Мінусом можна назвати не дуже зручну секцію коментарів, приходиться гортати до самого низу сторінки щоб залишити свій коментар. Візьмемо це до уваги при розробці ShowMe.

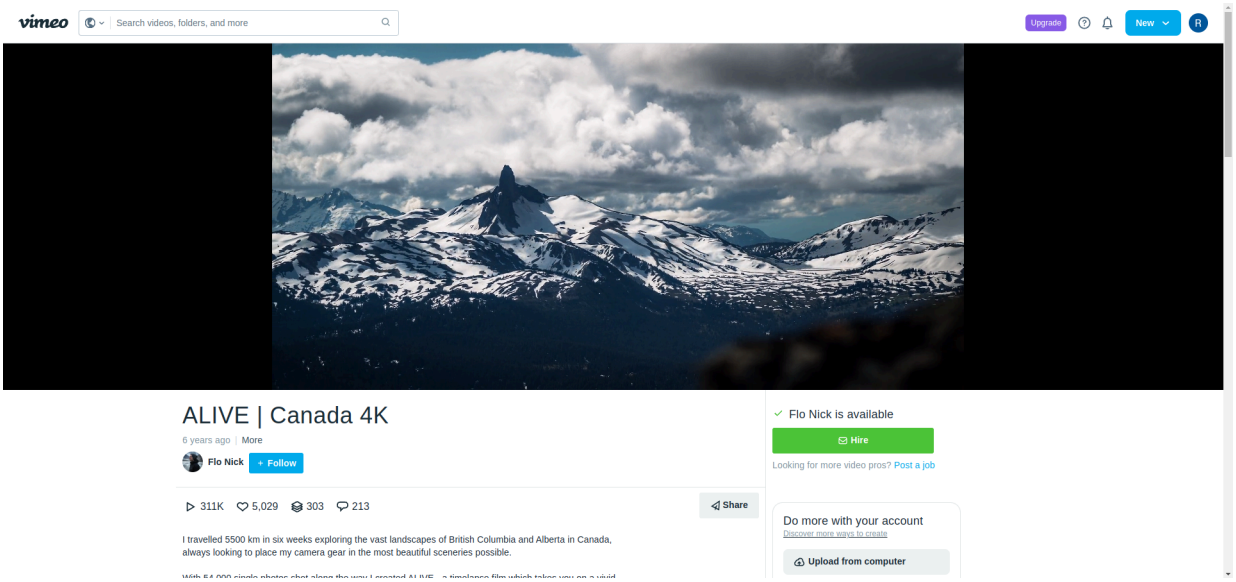


Рис. 1.6 - Сторінка відтворення відео

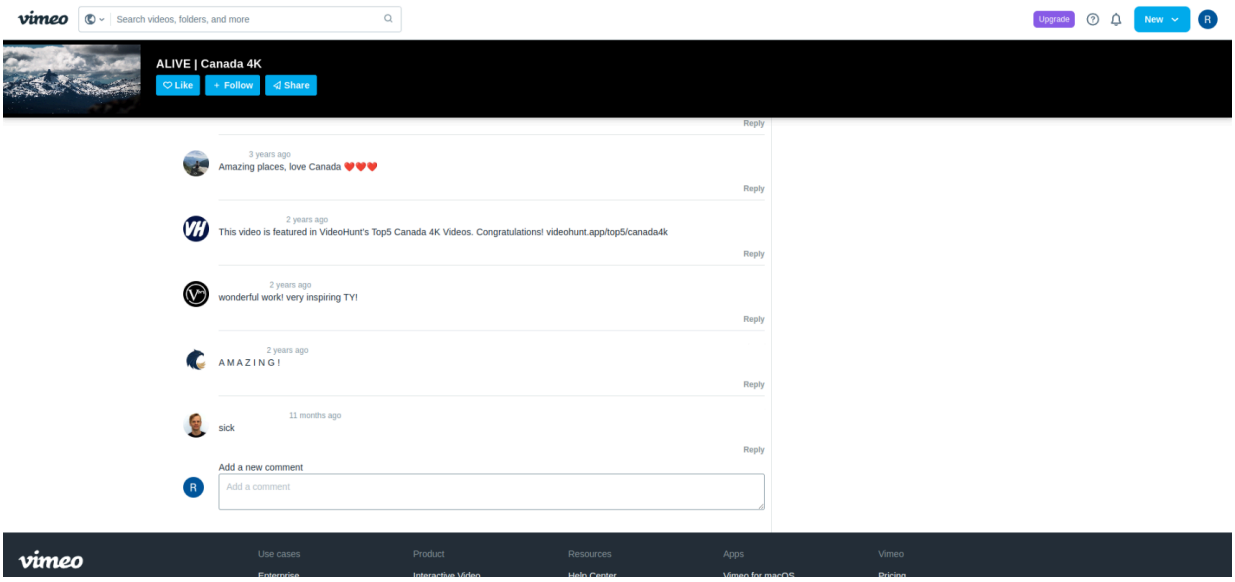


Рис. 1.7 - Секція з коментарями

Підводячи підсумок, Vimeo - це онлайн-платформа для відеохостингу, яка відрізняється від YouTube тим, що фокусується на творчому та професійному відеоконтенті. Вона пропонує користувачам широкий спектр можливостей для завантаження, перегляду та обміну відео, але з більшою увагою до якості та спільноти творців.

Ця платформа має свої плюси і мінуси. Перевагою Vimeo є розширений набір джерел з яких можна завантажувати відео, також платформа дозволяє створювати відео безпосередньо на платформі. Vimeo дозволяє дивитись відео без перерв на рекламу. В свою чергу безкоштовна версія Vimeo має обмежений функціонал. Для розширення потрібно буде заплатити. Головними недоліками Vimeo є не дуже інтуїтивно зрозумілий інтерфейс, не зручне розташування форми для коментування. Також у Vimeo менша аудиторія, а отже й різноманітність відео нижча у порівнянні з YouTube.

1.2.3 Dailymotion

Dailymotion – це платформа для відеохостингу, заснована у 2005 році у Франції. Вона дозволяє користувачам завантажувати, переглядати та ділитися відеоконтентом. Dailymotion є однією з найбільших платформ для відеохостингу в світі, з аудиторією, що охоплює мільйони користувачів.

Платформа надає доступ до відео, популярних у певних регіонах, особливо в Європі. В інших частинах світу вона користується меншою популярністю.

Зразу хочеться відмітити простоту та зручність інтерфейсу, який дозволяє переглядати відео просто на одній сторінці за допомогою скролу. Елементи контролю відео (уподобання, поширення, зберігання) теж знаходяться у зручному місці, що гарно впливає на користувацький досвід.

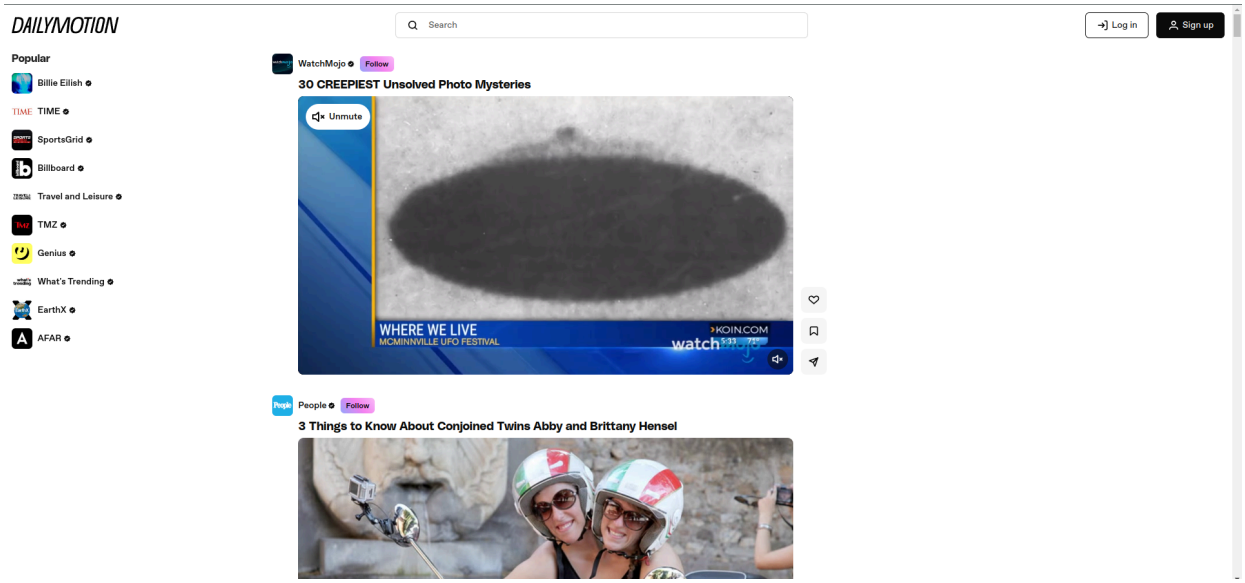


Рис. 1.8 - Домашня сторінка Dailymotion

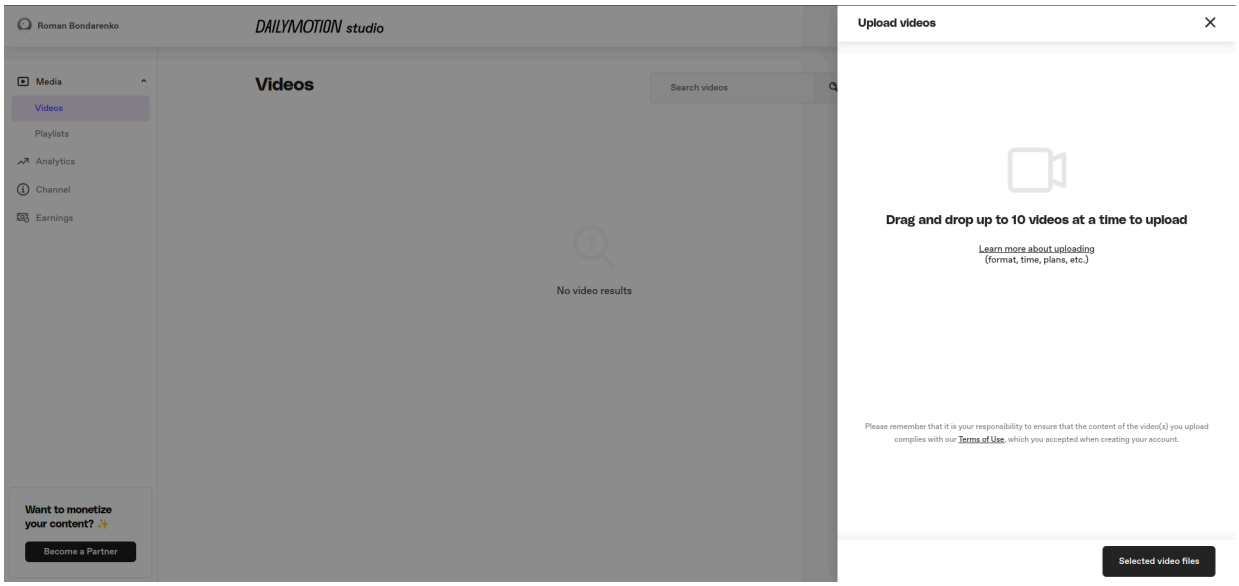


Рис. 1.9 - Сторінка додавання відео

Зразу можна відмітити відсутність варіативності джерел додавання відео, та не дуже зручні елементи навігації.

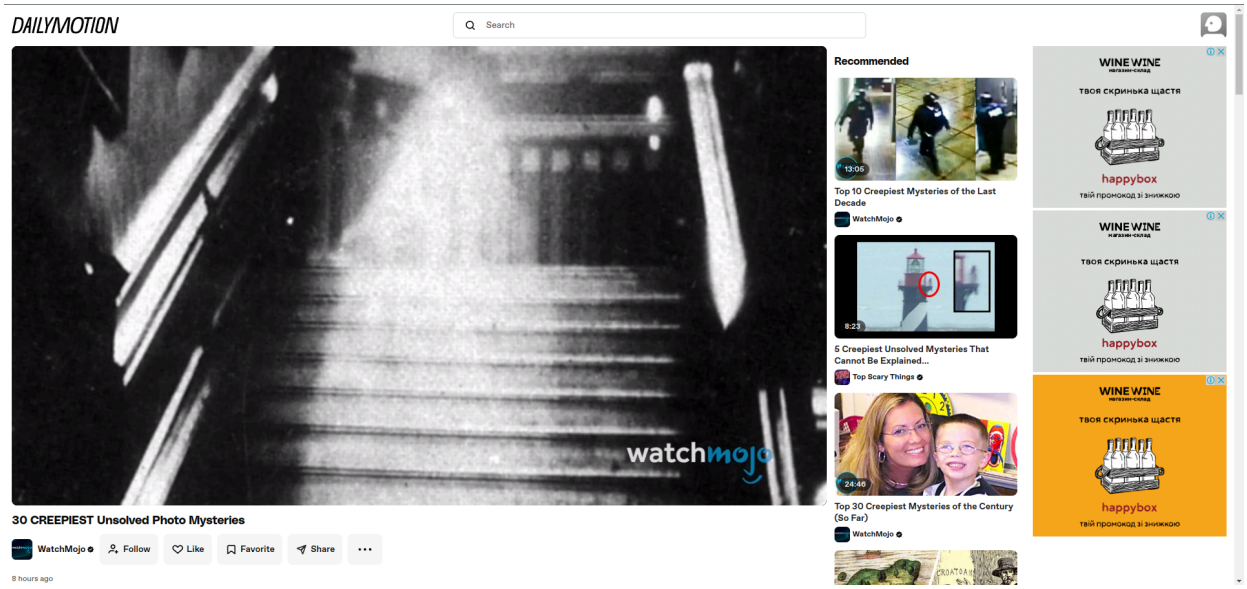


Рис. 1.10 - Сторінка відтворення відео Dailymotion

Із позитивних аспектів можна зазначити зручні елементи взаємодії із відеоконтентом, такі як підписка, уподобання, збереження і поширення. Великим недоліком є неможливість залишати коментарі, та наявність реклами.

Підсумовуючи, можна сказати що Dailymotion це доволі популярна платформа у деяких регіонах, здебільшого у Європі. Відповідно, аудиторія у цієї платформи менша ніж у YouTube, а отже й різноманітність відеоконтенту теж на нижчому рівні. Із переваг можна зазначити зручну домашню сторінку та інтуїтивно зрозумілий плеєр з повним функціоналом.

1.3 Side-to-side порівняння

Дослідивши ці три аналоги платформи для відеохостингу можемо скласти таблицю порівняння і наочно поглянути на спільні і відмінні риси цих платформ.

Таблиця 1.1 - Порівняння доступних аналогів

	YouTube	Vimeo	Dailymotion
Фокус	Загальний	Творчий та професійний контент	Загальний, з акцентом на Європу
Кількість контенту	Найбільша	Менша	Середня
Якість відео	Різна	Висока	Різна
Обмеження безкоштовного тарифу	На обсяг завантажуваного відео	На обсяг завантажуваного відео	На обсяг завантажуваного відео
Реклама	Є на безкоштовних відео	Відсутня на безкоштовних відео	Може бути менше, ніж на YouTube
Спільнота	Велика та різноманітна	Менша, але більш тісна	Активна, але може бути меншою
Монетизація	Є	Деякі платні тарифи	Деякі платні тарифи

Інструменти для редагування	Базові безкоштовні інструменти	Деякі платні тарифи пропонують інструменти	Деякі базові інструменти
Вартість	Безкоштовний з платними тарифами	Безкоштовний з платними тарифами	Безкоштовний з платними тарифами
Переваги	Величезна кількість контенту, безкоштовний, простий у використанні	Висока якість відео, відсутність реклами, орієнтація на спільноту	Широкий спектр відео, локальний контент, менше реклами
Недоліки	Може бути перевантажений контентом, алгоритми ранжирування можуть бути непрозорими	Менша кількість контенту, обмеження безкоштовного тарифу	Менша кількість контенту, менша аудиторія, менш зручний інтерфейс
Підходить для	Користувачів, які шукають різноманітний відеоконтент	Творців, які хочуть завантажувати та ділитися відео високої якості, глядачів, які шукають професійний контент	Користувачів, які шукають різноманітний відеоконтент, особливо в Європі

ВИСНОВОК ДО РОЗДІЛУ 1

В цьому розділі було проведено поглиблене дослідження трьох основних платформ для відеохостингу: YouTube, Vimeo та Dailymotion. Кожна з них володіє унікальними характеристиками, які роблять їх придатними для різних аудиторій та потреб.

YouTube – це гігантський веб-сайт, що лідирує в галузі відеохостингу, пропонуючи безмежний спектр контенту та широку аудиторію. Платформа доступна як безкоштовно, так і за платними тарифами, надаючи широкий спектр функцій для завантаження, перегляду, коментування та монетизації відео. Однак, через свою масштабність, YouTube може бути перевантажений контентом.

Vimeo – це платформа, орієнтована на творчий та професійний контент, надаючи користувачам можливість ділитися відео високої якості без реклами. Вона підтримує атмосферу орієнтованості на спільноту, де творці можуть отримувати конструктивну критику та знаходити потенційних клієнтів. Проте, Vimeo має меншу кількість контенту та обмеження безкоштовного тарифу.

Dailymotion – це платформа, що пропонує широкий спектр відео, з акцентом на локальний контент, особливо в Європі. Вона має менше реклами, ніж YouTube, але також меншу кількість контенту та меншу аудиторію.

Вибір платформи для відеохостингу залежить від ваших конкретних потреб та цілей. Для тих, кому потрібна величезна кількість контенту та широка аудиторія, YouTube – це найкращий вибір. Тим, хто цінує високу якість відео та відсутність реклами, рекомендується Vimeo. Якщо ви шукаєте локальний контент та меншу кількість реклами, Dailymotion може бути цікавою альтернативою.

Також, проаналізувавши переваги і недоліки цих систем, я виділив пункти, які потрібно реалізувати у системі ShowMe щоб вона почала користуватись популярністю.

РОЗДІЛ 2. ДОСЛІДЖЕННЯ ТА ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

2.1 Архітектура IT систем

Архітектура IT-системи визначає структуру та організацію її компонентів, а також способи їх взаємодії. Це основа будь-якої успішної IT-системи, яка забезпечує її масштабованість, надійність, безпеку та простоту обслуговування. Архітектура IT систем може бути реалізована у двох найпопулярніших підходах: моноліт та мікросервіси. Кожен з цих підходів має свої переваги та недоліки, і вибір між ними залежить від вимог та особливостей конкретного проекту. Розглянемо кожен підхід і визначимо який з них буде задовольняти нас більше.

2.1.1 Моноліт

Монолітна архітектура – це підхід до розробки програмного забезпечення, при якому всі компоненти системи об'єднані в один єдиний додаток. У монолітній архітектурі всі його частини, такі як користувацький інтерфейс, бізнес логіка і доступ до даних, працюють в рамках однієї програми.

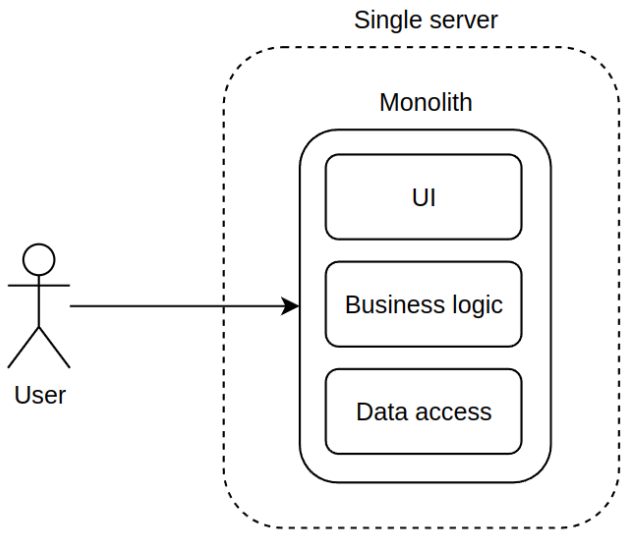


Рис. 2.1 - Структура монолітної архітектури

Як і всі інші підходи до розробки інфраструктури IT систем, моноліт має свої переваги та недоліки. Почнемо з плюсів:

- Простота розробки та тестування: Всі компоненти системи зібрані в одному місці, що спрощує розробку та тестування.
- Єдина кодова база: Наявність однієї кодової бази спрощує управління залежностями та версіями програмного забезпечення.
- Просте розгортання: Монолітні додатки легко розгортаються, оскільки вони представляють собою єдиний блок, який потрібно запустити.
- Надійність: Монолітні програми, як правило, більш стійкі до збоїв, оскільки всі компоненти щільно інтегровані.
- Швидка розробка: Монолітна архітектура може бути швидшою у розробці для простих проектів.
- Менше оверхеду: Менше оверхеду на мережеву взаємодію між компонентами, оскільки всі вони знаходяться в одному процесі.

З недоліків моноліту можна виділити:

- Складність масштабування: Важко масштабувати окремі компоненти. Додаток може масштабуватися лише цілком, що може бути неефективним.
- Складність управління: Зростання розміру коду може призвести до складнощів у його розумінні та підтримці. Впровадження нових функцій або змін може впливати на весь додаток, підвищуючи ризик збоїв.
- Деплоймент: Кожна зміна в коді вимагає перезавантаження всього додатку, що може призвести до простоїв.
- Менша гнучкість у виборі технологій: Всі компоненти системи повинні використовувати одні й ті ж технології, що обмежує можливості використання нових або спеціалізованих рішень.

2.1.2 Мікросервіси

Мікросервісна архітектура – це підхід до розробки програмного забезпечення, при якому система розбивається на набір незалежних сервісів, кожен з яких виконує одну чітко визначену функцію. Кожен сервіс працює автономно і може бути розгорнутий, масштабований та оновлений незалежно від інших. Мікросервіси спілкуються між собою через добре визначені інтерфейси, такі як HTTP API або месенджери.

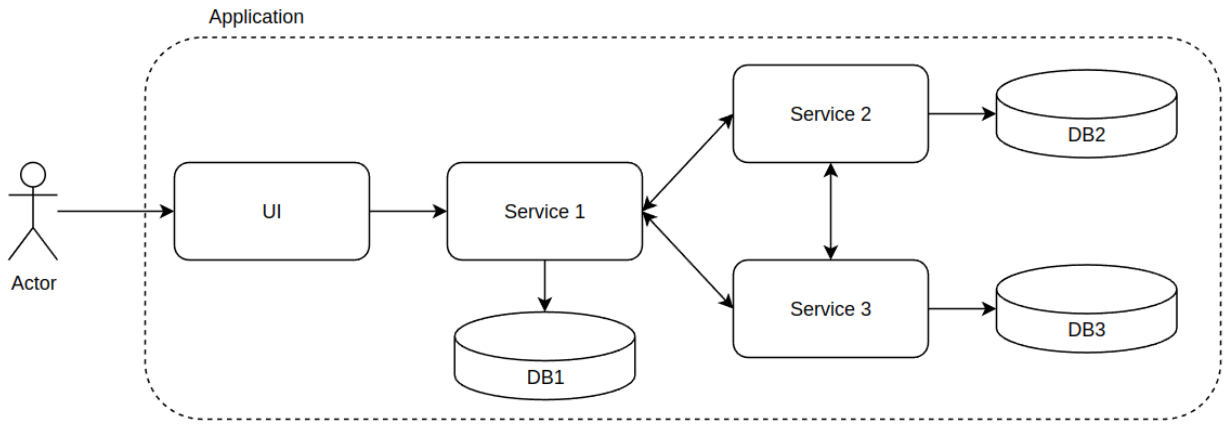


Рис. 2.2 - Структура мікросервісної архітектури

Тепер поглянемо на переваги мікросервісної архітектури над іншими:

- Масштабованість: Кожен сервіс можна масштабувати незалежно, що дозволяє оптимально використовувати ресурси.
- Швидкі оновлення та релізи: Кожен сервіс можна оновлювати та розгортати незалежно, що прискорює процес релізів.
- Гнучкість у виборі технологій: Кожен сервіс може бути реалізований за допомогою найбільш підходящої для нього технології.
- Покращена надійність: Проблеми в одному сервісі не впливають на роботу інших сервісів, що підвищує загальну надійність системи.
- Простота підтримки великих проектів: Легше управляти та розробляти окремі малі сервіси, ніж великий монолітний додаток.
- Модульність: Легше підтримувати та змінювати окремі сервіси, що покращує загальну модульність та гнучкість системи.
- Незалежний деплоймент: Сервіси можуть деплоюватися та оновлюватися незалежно один від одного, що зменшує ризик простоїв.

Тепер щодо недоліків цієї архітектури:

- Складність управління: Підтримка великої кількості мікросервісів потребує складного оркестрування та управління.
- Мережевий оверхед: Взаємодія між сервісами через мережу додає оверхед і може призвести до затримок.
- Тестування: Тестування взаємодії між мікросервісами складніше, ніж тестування монолітного додатку.
- Інфраструктура: Потребує більш складної інфраструктури для управління сервісами, контейнерами та їх оркестрацією (наприклад, Kubernetes).
- Забезпечення цілісності даних: Оскільки дані можуть бути розподілені між різними сервісами, забезпечення їх цілісності та узгодженості може бути викликом.

2.2 Хмарні середовища

Хмарні технології стали невід'ємною частиною сучасного розробки та розгортання програмного забезпечення. Завдяки їм з'являється доступ до інфраструктури, платформ та сервісів, які дозволяють створювати, тестувати, впроваджувати та масштабувати додатки через Інтернет. Це дає компаніям можливість зосередитися на ключових аспектах свого бізнесу.

Одні з найбільш популярних хмарних середовищ це Amazon Web Services від компанії Amazon та Microsoft Azure від Microsoft.

2.2.1 AWS

AWS (Amazon Web Services) – це одна з найбільших і найпопулярніших хмарних платформ, що надає понад 200 повнофункціональних сервісів для обчислень, зберігання даних, баз даних, машинного навчання, аналітики та багато іншого. Багато великих компаній використовують AWS, що забезпечує надійність і перевірені рішення.

AWS це хмарне середовище типу IaaS (Infrastructure as a Service), де віртуальна інфраструктура для обчислення, зберігання даних і мережевих ресурсів, надається компанією через Інтернет.

AWS гарантує:

- Велику кількість сервісів для будь-яких потреб.
- Датацентри, які розташовані по всьому світу.
- Високий рівень надійності та масштабованості.

Найбільш використовувані сервіси на AWS:

- Обчислення: EC2, Lambda, ECS, EKS
- Зберігання: S3, EBS, Glacier
- Бази даних: RDS, DynamoDB, Aurora
- Аналітика: Redshift, Athena, Kinesis
- Машинне навчання: SageMaker, Rekognition
- Інструменти розробки: CodeCommit, CodeBuild, CodeDeploy
- Контейнеризація: ECS, EKS, Fargate

2.2.2 Microsoft Azure

Microsoft Azure – це хмарна платформа типу IaaS від Microsoft, яка є аналогом AWS із власними перевагами і набором сервісів.

Перевагами Azure можна вважати:

- Інтеграцію з Windows Server, Active Directory, Office 365 та іншими продуктами Microsoft, що полегшує перехід до хмари для існуючих користувачів.
- Велику кількість сервісів для різних потреб, від обчислень і зберігання до машинного навчання і аналітики.
- Датацентри Azure, які розташовані по всьому світу, що забезпечує низькі затримки та високу доступність.
- Високий рівень безпеки що відповідає багатьом галузевим стандартам і сертифікаціям.

Найпопулярніші сервіси Azure:

- Обчислення: Virtual Machines, App Services, Azure Kubernetes Service (AKS), Azure Functions.
- Зберігання: Blob Storage, Disk Storage, File Storage, Archive Storage
- Бази даних: Azure SQL Database, Cosmos DB, Azure Database for PostgreSQL, Azure Database for MySQL
- Аналітика: Azure Synapse Analytics, HDInsight, Stream Analytics
- Машинне навчання: Azure Machine Learning, Cognitive Services
- Інструменти розробки: Azure DevOps, Visual Studio та Visual Studio Code

2.3 Вибір технологій розробки

Для реалізації проекту було погоджено використовувати мову програмування Java, фреймворк для бекенд частини Spring Boot, СУБД (систему управління базами даних) MySQL. Для написання фронтенд частини було прийнято використовувати React, який є дуже популярною бібліотекою для написання користувацьких інтерфейсів, написаний на мові JavaScript. Для деплойменту та розгортання проекту на хмарному середовищі будемо використовувати уже згадане середовище AWS (Amazon Web Services). З додаткових інструментів задіємо Maven для збірки проекту та Docker для контейнеризації нашого додатку.

2.3.1 Java

Java – це об’єктно-орієнтована мова програмування, яка була розроблена компанією Sun Microsystems (зараз частина Oracle Corporation) у 1995 році. Вона є однією з найбільш популярних і широко використовуваних мов програмування у світі, особливо у сфері розробки корпоративних додатків, веб-додатків, мобільних додатків і системного програмного забезпечення.

Java характеризується наступними аспектам:

- Об’єктно-орієнтованість
    - Класи та об’єкти: Вся функціональність в Java організована в класи, які є шаблонами для створення об’єктів.
    - Наслідування: Класи можуть успадковувати властивості та методи інших класів.

- Інкапсуляція: Дані класу можуть бути приховані від зовнішнього доступу і доступні лише через методи класу.
- Поліморфізм: Об’єкти можуть приймати різні форми в залежності від контексту.

- Платформна незалежність: Java була створена для того, щоб програми, написані на ній, могли виконуватися на будь-якій платформі без змін в коді. Це досягається завдяки використанню байт-коду, який інтерпретується або компілюється в машинний код на конкретній платформі за допомогою JVM (Java Virtual Machine).
- Безпека: Java надає кілька механізмів для забезпечення безпеки, включаючи контроль доступу, криптографію, перевірку коду під час виконання та ізоляцію програм в окремих середовищах виконання.
- Автоматичне управління пам’яттю: Java має автоматичний збирач сміття (garbage collector), який автоматично звільняє пам’ять, що більше не використовується.
- Багата стандартна бібліотека: Java має велику стандартну бібліотеку, що включає класи для роботи з файлами, мережею, графічним інтерфейсом, колекціями та багато іншого.

2.3.2 Spring Boot

Spring Boot – це фреймворк на основі Spring, який полегшує створення та розгортання додатків Java з мінімальною кількістю налаштувань. Він надає інструменти та функції, які спрощують процес розробки, конфігурації та розгортання Java-додатків.

Spring Boot характеризується наступними аспектами:

- Автоматична конфігурація: Spring Boot використовує автоматичну конфігурацію для налаштування додатків на основі залежностей, що використовуються в проекті. Розробникам не потрібно вручну налаштовувати більшість параметрів.
- Вбудований сервер: Spring Boot включає вбудовані сервери, такі як Tomcat, Jetty або Undertow, що дозволяє запускати додаток як самостійний виконуваний файл без необхідності розгортання на окремому сервері.
- Стартери (Starters): Spring Boot надає набір "стартерів" – попередньо налаштованих залежностей, які дозволяють швидко додати необхідні функції в додаток (наприклад, spring-boot-starter-web для веб-додатків, spring-boot-starter-data-jpa для роботи з базами даних).
- Spring Initializr: Це веб-інструмент для швидкого створення нових Spring Boot проектів. Він генерує базову структуру проекту з необхідними залежностями.

- Легкість розгортання: Завдяки підтримці вбудованих серверів та автоматичної конфігурації, Spring Boot додатки легко розгортати в різних середовищах, включаючи локальні машини, хмарні платформи, контейнери Docker тощо.
- Моніторинг та керування: Spring Boot Actuator надає вбудовані інструменти для моніторингу та управління додатком, включаючи показники продуктивності, інформацію про стан додатка та інші важливі метрики.
- Документація та підтримка: Spring Boot має велику та детальну документацію, а також активну спільноту розробників.

Основними компонентами Spring Boot є:

- Spring Boot Starters
  - spring-boot-starter-web: Включає необхідні залежності для створення веб-додатків з використанням Spring MVC.
  - spring-boot-starter-data-jpa: Надає залежності для роботи з базами даних через JPA (Java Persistence API).
  - spring-boot-starter-security: Включає залежності для додавання функцій безпеки до додатку.
- Spring Boot Auto Configuration
  - Автоматично налаштовує додаток на основі залежностей, що використовуються, зменшуючи кількість конфігураційних файлів, необхідних для запуску додатку.

- Spring Boot Actuator
    - Набір інструментів для моніторингу та управління додатками в реальному часі. Включає кінцеві точки для отримання інформації про стан додатка, показники продуктивності, збір логів тощо.
- Spring Boot DevTools
    - Набір інструментів для полегшення розробки, включаючи автоматичне перезавантаження додатка при зміні коду, доступ до консольних команд тощо.

2.3.3 MySQL

MySQL – це система управління реляційними базами даних (RDBMS), яка є однією з найпопулярніших у світі. Вона розроблена компанією MySQL AB, яка була придбана Sun Microsystems, а потім – Oracle Corporation. MySQL є відкритим програмним забезпеченням і широко використовується для веб-додатків, корпоративних систем та інших проектів.

Основні характеристики MySQL:

- Реляційна модель даних: Дані організовані у вигляді таблиць, які складаються з рядків і стовпців, що дозволяє маніпулювати даними у структурованому вигляді.
- SQL (Structured Query Language): MySQL використовує стандартну мову SQL для запитів до бази даних.
- Широка підтримка платформ: MySQL працює на різних операційних системах, включаючи Windows, Linux, macOS.

- Масштабованість і продуктивність: MySQL забезпечує високу продуктивність і може масштабуватися від невеликих додатків до великих, високо-навантажених систем.
- Надійність і безпека: MySQL надає засоби для забезпечення надійності даних. Серед них транзакції, журналювання та резервне копіювання.
- Реплікація і кластеризація: MySQL підтримує різні види реплікації (master-slave, master-master), що дозволяє створювати резервні копії даних і забезпечувати високу доступність.

MySQL це не просто база даних. До неї входять всілякі інструменти для полегшення роботи з основним компонентом MySQL Server. Це такі інструменти, як:

- MySQL Client: Інтерфейс командного рядка для взаємодії з MySQL Server, який дозволяє виконувати SQL-запити, адмініструвати базу даних та управляти користувачами.
- MySQL Workbench: Графічний інструмент для дизайну, розробки та адміністрування баз даних MySQL. Він забезпечує зручний інтерфейс для створення схем, написання SQL-запитів та управління серверами.

Для того щоб підключитися до бази даних MySQL програмно, тобто через якусь мову програмування, існує MySQL Connector. Це набір драйверів та бібліотек для підключення до MySQL з різних мов програмування, таких як Java, .NET, Python, PHP та інші.

2.3.4 React

React – це популярна бібліотека для створення користувацьких інтерфейсів, розроблена і підтримувана Facebook. Вона використовується для розробки односторінкових додатків (SPA) та дозволяє створювати складні інтерфейси шляхом розбиття їх на дрібні, ізольовані компоненти.

React побудований на основі компонентів, які є незалежними, повторно використовуваними частинами інтерфейсу. Кожен компонент може мати свій власний стан і логіку.

React використовує віртуальний DOM для ефективного оновлення інтерфейсу. Коли змінюється стан додатку, React оновлює віртуальний DOM і порівнює його з попереднім станом, мінімізуючи кількість операцій з реальним DOM, що покращує продуктивність.

JSX – це синтаксичне розширення JavaScript, яке дозволяє описувати структуру інтерфейсу в схожому на HTML форматі. JSX полегшує написання компонентів, оскільки він забезпечує інтуїтивно зрозумілий спосіб визначення структури UI.

Основні складові бібліотеки React:

- Компоненти:
    - Функціональні компоненти: Прості функції, які приймають props і повертають JSX. Вони не мають свого стану і використовуються для відображення статичного контенту.

    - Класові компоненти: Компоненти, що базуються на класах і можуть мати власний стан. Вони використовуються для більш складної логіки і управління станом.
- State і Props:
    - State: Об'єкт, який зберігає локальний стан компонента. Зміни в state викликають повторний рендеринг компонента.
    - Props: Властивості, які передаються від батьківського компонента до дочірнього і використовуються для передачі даних і налаштувань.
- Hooks: React Hooks дозволяють використовувати стан та інші можливості React у функціональних компонентах. Деякі з основних хуків включають useState, useEffect, useContext та інші.
- Контекст: Контекст надає спосіб передавати дані через дерево компонентів без необхідності явно передавати props на кожному рівні. Це зручно для глобальних налаштувань, таких як тема або аутентифікація користувача.

2.3.5 Maven

Apache Maven – це інструмент для управління проєктами та автоматизації збірки, що використовує XML-файл конфігурації (pom.xml) для опису структури проєкту, залежностей та налаштувань. Maven забезпечує стандартизований підхід до управління життєвим циклом програмних проєктів.

Основні характеристики Maven:

- POM (Project Object Model): Конфігураційний файл, який містить інформацію про проєкт, залежності і налаштування збірки.
- Фази життєвого циклу: Стандартні фази: validate, compile, test, package, verify, install, deploy.
- Профілі: Різні конфігурації збірки для різних середовищ (девелопмент, тестування, продакшн).
- Плагіни: Розширюють функціональність Maven, наприклад, для компіляції (maven-compiler-plugin), запуску тестів (maven-surefire-plugin), створення JAR-файлів (maven-jar-plugin).
- Управління залежностями: Автоматичне завантаження та оновлення бібліотек з центрального репозиторію Maven (Maven Central).

2.3.6 Docker

Docker - це платформа для розробки, розгортання та управління контейнеризованими додатками. Docker дозволяє упаковувати програмне забезпечення та всі його залежності в єдину одиницю, що називається контейнером.

Основні характеристики Docker:

- Контейнеризація: Docker використовує технологію контейнеризації для упаковки та ізоляції додатків та їх залежностей.
- Кросплатформенність: Контейнери Docker можуть запускатися на будь-якій платформі, що підтримує Docker Engine, незалежно від операційної системи або хост-системи.

- Легковаговість: Контейнери Docker мають невеликий розмір і швидко запускаються, що сприяє ефективному використанню ресурсів.
- Масштабованість: Docker дозволяє легко масштабувати додатки в залежності від навантаження.
- Інтеграція з іншими інструментами: Docker інтегрується з різними інструментами та технологіями, такими як Kubernetes для оркестрації контейнерів або Jenkins для автоматизації процесу CI/CD.

Основні поняття Docker

- Імейджі (Images): Імейдж - це шаблон для створення контейнера. Він містить у собі всі необхідні файли та конфігурації.
- Контейнери (Containers): Контейнер - це інстанс імейджа, який запущений у середовищі Docker. Він містить у собі виконуваний код та всі необхідні залежності.
- Dockerfile: Dockerfile - це текстовий файл, що містить інструкції для створення імейджа. В ньому описується кроки для побудови імейджа, такі як копіювання файлів, налаштування середовища тощо.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було проведено дослідження та аналіз сучасних технологій розробки, необхідних для створення веб-сервісу "ShowMe" - платформи для хостингу відео. Були розглянуті та порівняні різні аспекти технологій, такі як мови програмування, фреймворки, бази даних, інструменти контейнеризації та хмарні середовища.

Після дослідження було зроблено вибір технологій, які найбільш відповідають потребам та вимогам проєкту "ShowMe". Для реалізації бекенду обрано Java у поєднанні з фреймворком Spring Boot. Для фронтенду обрано React. В якості системи управління базами даних було вирішено використати MySQL. Крім того, для забезпечення контейнеризації додатку та його легкого розгортання вибрано Docker. Щодо хмарних середовищ, для розгортання та керування інфраструктурою було обрано Amazon Web Services (AWS).

РОЗДІЛ 3. ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКУ

У цьому розділі роботи ми розглянемо структуру застосунку із технічної точки зору, дослідимо які архітектурні рішення були застосовані при розробці додатку.

3.1 Архітектура застосунку

Для забезпечення гнучкості, простоти менеджменту та підтримки застосунку, я розбив його на кілька складових: фронтенд частину, бекенд частину, базу даних та сховище статичних файлів. Структурну схему застосунку представлено в Додатку 1. Розглянемо кожну з цих складових детальніше.

3.1.1 Сервер (бекенд)

У ролі серверу було прийнято рішення використовувати Spring Boot та Java. Ця пара застосовується на найбільших сучасних ентерпрайз проектах. Їх використовують такі компанії як Netflix, Uber, Spotify тощо. Використаємо й ми. Spring Boot користується популярністю через його автоматизацію початкових конфігурацій. Це означає, що не потрібно витрачати багато часу на створення початкової архітектури та конфігурації проекту, за нас це зробить Spring Boot. Це реалізовано за допомогою стартерів, спеціальних розширень, які пропонують мінімально необхідну початкову структуру того чи іншого сервісу.

У нашому проекті були використані наступні стартери:

- Spring-boot-starter-web: надає інструменти для побудови веб додатків
- Spring-boot-starter-data-jpa: надає інструменти для ефективної та зручної роботи з базою даних, представляючи таблиці у вигляді класів
- Spring-boot-starter-security: надає інструменти для роботи та налаштування безпеки додатку
- Spring-boot-starter-mail: надає підключатись до емейл сервера та надсилати листи користувачам
- Springdoc-openapi-starter-webmvc-ui: дозволяє побудувати API документацію згідно з OpenAPI специфікації

Також були використані такі залежності:

- Mysql-connector-j: надає змогу підключатись до бази даних програмно за допомогою спеціального драйвера
- Lombok: утиліта, яка генерує код замість нас, все що треба це позначити за допомогою анотацій що саме і де потрібно генерувати
- Java-jwt: бібліотека для роботи з JWT (Json Web Token)
- Aws-java-sdk-core: надає базові інструменти для роботи із хмарним сервісом AWS (Amazon Web Services)
- Aws-java-sdk-s3: надає спеціалізовані інструменти для роботи виключно із AWS S3 (Simple Storage Service) сховищем

Незважаючи на те, що Spring Boot робить всю базову конфігурацію та налаштування за нас, нам, всетаки, потрібно додати деякі параметри для успішного старту додатку. Вони зазначаються у файлі application.properties або application.yml.

```yaml
spring:
  application:
    name: showme
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: ${DB_URL:jdbc:mysql://localhost:3306/show_me}
    username: ${DB_USERNAME:root}
    password: ${DB_PASSWORD:root}
  jpa:
    hibernate:
      ddl-auto: update
  mail:
    host: ${MAIL_HOST:localhost}
    port: ${MAIL_PORT:4025}

app:
  path: ${APP_PATH:http://localhost:8080}
  ui-path: ${APP_UI_PATH:http://localhost:5173}
  security:
    jwt:
      access:
        secret: ${JWT_SECTER:QWEASDZXCRTYFGHVBNUIOJKLM}
        expiration-ms: ${JWT_EXPIRATION_MS:900000}
  aws:
    s3:
      path: ${AWS_S3_PATH:http://localhost:4566}
      access-key: ${AWS_ACCESS_KEY:AAAAAAAAAAAAAAAAAAAA}
      secret-key: ${AWS_SECRET_KEY:KKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKKK}
      bucket-name: ${AWS_S3_BUCKET_NAME:showme}
      region: ${AWS_REGION:eu-north-1}
```

Рис. 3.1 - Файл конфігурації проекту application.yml

Необхідними параметрами для старту додатку є інформація про підключення до бази даних та до емейл сервісу. Тут ми задаємо шлях до бази даних, юзернейм, пароль, та необхідний драйвер для роботи з базою. Змінні під неймспейсом app використовує не Spring Boot, тому вони є необов’язковими для старту, але без них неможливо реалізувати необхідну функціональність.

Основним завданням застосунку є можливість завантажувати і переглядати відео. Для зберігання відео вирішено застосувати AWS S3 сервіс, який є сховищем файлів. Іншим варіантом було зберігати відео прямо в базі даних, але це є поганою практикою, тому ми будемо зберігати файли відео на S3, а шлях до них у базі даних.

Розглянемо сервіс, який займається завантаженням файлів у хмару та повернення утвореного шляху до них.

```
4 usages
@Slf4j
@Service
@RequiredArgsConstructor
public class AmazonS3Service {
    private final AmazonS3 s3Client;

    @Value("${app.aws.s3.path}")
    private String path;

    @Value("${app.aws.s3.bucket-name}")
    private String bucketName;

    4 usages
    public String uploadFile(MultipartFile multipartFile, FileType fileType, String userId) {
        try {
            String fileName = constructFileName(multipartFile, fileType, userId);
            File file = convertMultipartFileToIOFile(multipartFile);
            uploadFileToS3Bucket(fileName, file);
            file.delete();
            return path + "/" + bucketName + "/" + fileName;
        } catch (IOException e) {
            log.error("Failed to upload file to S3 bucket: {}", e.getLocalizedMessage());
            throw new RuntimeException("Failed to upload file to S3 bucket: " + e.getLocalizedMessage(), e);
        }
    }
```

Рис. 3.2 - Клас AmazonS3Service

Бачимо, що для того щоб завантажити файл нам потрібно згенерувати ім’я, перетворити файл із одного типу в інший, який підтримується клієнтом S3, який ми отримали із стартера aws-java-sdk-s3.

```
    1 usage
    private String constructFileName(MultipartFile multipartFile, FileType fileType, String userId) {
        String commonFileName = fileType.getFolderName() + "/" + userId;
        if (fileType == FileType.AVATAR) {
            return commonFileName + "." + FilenameUtils.getExtension(multipartFile.getOriginalFilename());
        }
        return commonFileName + "/" + System.currentTimeMillis() + "__" + multipartFile.getOriginalFilename();
    }

    1 usage
    private File convertMultipartFileToIOFile(MultipartFile multipartFile) throws IOException {
        String originalFilename = multipartFile.getOriginalFilename();
        File converted = new File(originalFilename == null ? UUID.randomUUID().toString() : originalFilename);
        FileOutputStream fos = new FileOutputStream(converted);
        fos.write(multipartFile.getBytes());
        fos.close();
        return converted;
    }

    1 usage
    private void uploadFileToS3Bucket(String fileName, File file) {
        if (s3Client.doesObjectExist(bucketName, fileName)) {
            s3Client.deleteObject(new DeleteObjectRequest(bucketName, fileName));
        }
        s3Client.putObject(new PutObjectRequest(bucketName, fileName, file)
                .withCannedAcl(CannedAccessControlList.PublicRead));
    }
}
```

Рис. 3.3 - методи генерації імені та перетворення файлу

Головним методом класу є uploadFileToS3Bucket. Саме він звертається до клієнта S3 із запитом на збереження файлу на хмару. Та перед цим його також треба проініціалізувати та задати параметри нашого сховища. Це було виконано у класі AppConfig. Отримавши шлях до нашого сховища, назву бакета, та ключі для авторизації із конфігураційного файлу, про який згадувалось вище, ми можемо провести налаштування клієнту.

```java
@Configuration
public class AppConfig {
    @Value("${app.aws.s3.path}")
    private String path;

    @Value("${app.aws.s3.region}")
    private String region;

    @Value("${app.aws.s3.access-key}")
    private String accessKey;

    @Value("${app.aws.s3.secret-key}")
    private String secretKey;

    @Bean
    public AmazonS3 amazonS3() {
        return AmazonS3ClientBuilder.standard()
                .withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(path, region))
                .withCredentials(new AWSStaticCredentialsProvider(new BasicAWSCredentials(accessKey, secretKey)))
                .withPathStyleAccessEnabled(true)
                .build();
    }

    @Bean
    public MultipartConfigElement multipartConfigElement() {
        MultipartConfigFactory factory = new MultipartConfigFactory();
        factory.setMaxFileSize(DataSize.ofBytes(512000000L));
        factory.setMaxRequestSize(DataSize.ofBytes(512000000L));
        return factory.createMultipartConfig();
    }
}
```

Рис. 3.4 - AppConfig

Також можемо побачити, що проводиться налаштування максимального розміру файлу, який можна завантажити. Це робиться для того щоб обійти стандартну валідацію, яка дозволяє завантажувати файли всього лише на 10МБ, що є дуже малим допустимим розміром для відеофайлів.

Далі розглянемо як саме сервер обробляє запит на збереження відео. Розпочнемо із рівня контролерів, який напряму приймає запити від фронтенду, про який буде йтись пізніше. Цей контроллер називається VideoController.

```java
@RestController
@RequiredArgsConstructor
@RequestMapping("/videos")
public class VideoController {
    private final UserService userService;
    private final VideoService videoService;

    @PostMapping
    public ResponseEntity<String> uploadVideo(@RequestParam("video") MultipartFile video,
                                              @RequestParam("preview") MultipartFile preview,
                                              @RequestParam("title") String title,
                                              @RequestParam("description") String description,
                                              Authentication authentication) {
        User user = userService.findByEmail(authentication.getName()).orElseThrow(
                () -> new UnauthorizedException("Unauthorized")
        );
        Video uploadedVideo = videoService.uploadVideo(video, preview, user, title, description);
        return ResponseEntity.ok( body: "Video uploaded: " + uploadedVideo.getTitle());
    }
```

Рис. 3.5 - VideoController.uploadVideo()

Бачимо що для того щоб завантажити відео на хмару, спочатку треба знайти користувача із сесії, який намагається це зробити. Якщо такий користувач знайдений, проводиться виклик до VideoService.uploadVideo() який в свою чергу звертається до AmazonS3Service, який вже і завантажує відео у сховище та виклик до VideoRepository.save(), який зберігає інформацію про збережене відео до бази даних.

```java
@Service
@RequiredArgsConstructor
public class DefaultVideoService implements VideoService {
    private final AmazonS3Service s3Service;
    private final VideoRepository videoRepository;
    private final UserRepository userRepository;

    1 usage
    @Override
    public Video uploadVideo(MultipartFile video, MultipartFile preview, User author, String title, String description) {
        String videoUrl = s3Service.uploadFile(video, FileType.VIDEO, author.getId());
        String previewUrl = s3Service.uploadFile(preview, FileType.PREVIEW, author.getId());
        return videoRepository.save(new Video(videoUrl, previewUrl, title, description, author));
    }
```

Рис. 3.6 - Імплементація VideoService

Розібравшись із збереженням відео, розглянемо як це відео дістається з бази даних і повертається на фронтенд. За приймання такого запиту також відповідає VideoController.

```
@GetMapping("/{videoId}")
public ResponseEntity<VideoResponseDto> getVideo(@PathVariable String videoId, Authentication authentication) {
    Video video = videoService.findVideoById(videoId).orElseThrow(
            () -> new NotFoundException("Video [%s] not found".formatted(videoId))
    );
    VideoResponseDto videoResponseDto = new VideoResponseDto(
            video.getId(),
            video.getVideoUrl(),
            video.getTitle(),
            video.getDescription(),
            video.getAuthor().getId(),
            video.getAuthor().getName(),
            video.getAuthor().getImageUrl(),
            video.getLikes().size(),
            video.getLikes().stream().map(User::getEmail).anyMatch(s -> s.equals(authentication.getName())),
            video.getDislikes().size(),
            video.getDislikes().stream().map(User::getEmail).anyMatch(s -> s.equals(authentication.getName()))
    );
    return ResponseEntity.ok(videoResponseDto);
}
```

Рис. 3.7 - метод отримання відео з бази даних

Бачимо, що пошук відео проводиться по його айді, яке нам передає фронтенд. Потім проводиться формування респонсу, до якого входять айді відео, посилання на сховище, де це відео знаходиться, назва, опис, інформація про автора цього відео, кількість лайків та дизлайків.

У проекті також присутня реалізація авторизації та аутентифікації. За це відповідає SecurityConfig, а саме бін securityFilterChain, який надає доступ до певних дій в залежності чи авторизований користувач, чи ні.

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    1 usage
    private static final String[] PUBLIC_MATCHERS = {
            "/auth/**",
            "/test/all",
            "/swagger-ui/**"
    };

    @Value("${app.ui-path}")
    private String uiPath;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http, JwtAuthFilter jwtAuthFilter) throws Exception {
        return http
                .csrf(AbstractHttpConfigurer::disable)
                .sessionManagement(config -> config.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests(config -> config
                        .requestMatchers(PUBLIC_MATCHERS).permitAll()
                        .anyRequest().authenticated()
                )
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
                .build();
    }
```

Рис. 3.8 - Налаштування безпеки у SecurityConfig

Іншу функціональність, яку пропонує нам сервер можна побачити за допомогою Swagger, який ми отримали із springdoc-openapi-starter-webmvc-ui стартера.

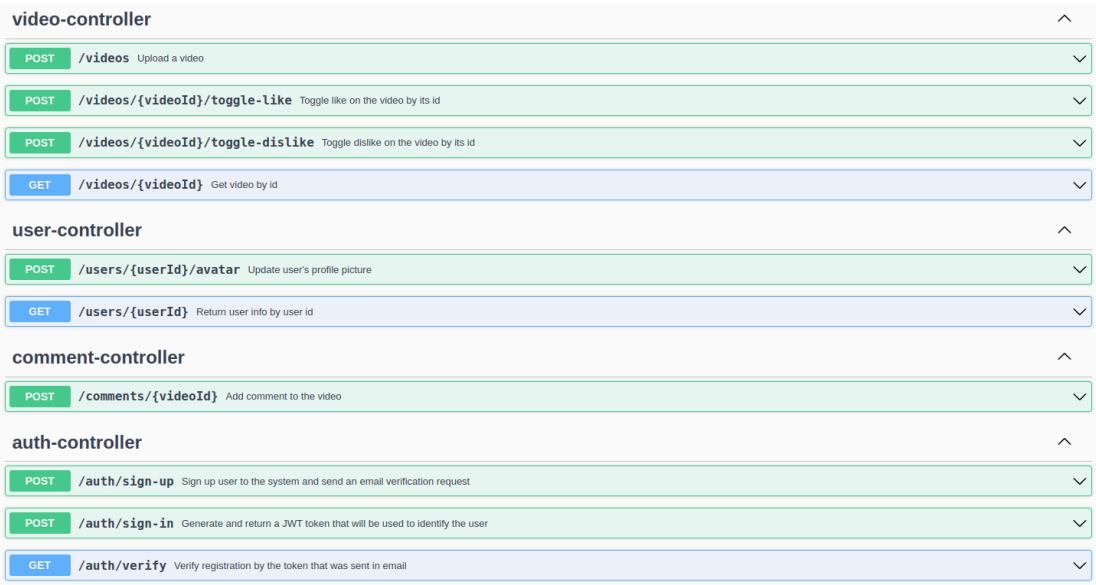


Рис. 3.9 - API документація за допомогою Swagger

3.1.2 Клієнт (фронтенд)

Клієнтська частина системи реалізована на React з допомогою Vite. Сторінки побудовані за допомогою tsx файлів. Це такий тип файлів, у якому поєднується код Typescript та верстка HTML та CSS. Такі файли є дуже зручними, оскільки логіка, стиль та представлення знаходяться в одному місці, що значно зменшує кількість файлів у проекті, що полегшує пошук на навігацію між ними.

Ось список сторінок, які були розроблені:

- Followed.tsx
- Home.tsx
- Layout.tsx
- Liked.tsx
- Login.tsx
- MyProfile.tsx
- Profile.tsx
- Register.tsx
- VideoDetails.tsx
- Watched.tsx

Кожна сторінка побудована із компонентів, які можна перевикористовувати у багатьох місцях, вони є базовими елементами в React.

Список деяких компонентів:

- BlueButton.tsx
- ChannelButton.tsx
- FilterButton.tsx
- LikedButton.tsx
- LinkButrton.tsx
- ProfileButton.tsx
- VideoPicker.tsx
- FormInput.tsx
- ProfileInput.tsx
- Search.tsx
- AddVideo.tsx
- Header.tsx
- Login.tsx
- Register.tsx
- SideBar.tsx
- UserNavigation.tsx
- Video.tsx
- VideoCopy.tsx

Для навігації по інтерфейсу було використано react-router-dom, який відповідає за маршрутизацію в React. Для регулювання доступу до сторінок були використані ProtectedRoute, які забороняють доступ неавторизованим користувачам до певних сторінок.

```
import { createBrowserRouter } from "react-router-dom";
import Layout from "../pages/Layout";
import Home from "../pages/Home";
import Profile from "../pages/Profile";
import Watched from "../pages/Watched";
import Login from "../pages/Login";
import Register from "../pages/Register";
import Liked from "../pages/Liked";
import VideoDetails from "../pages/VideoDetails";
import MyProfile from "../pages/MyProfile";

export const router = createBrowserRouter([
    {
        path: "/",
        element: <Layout />,
        children: [
            {
                index: true,
                element: <Home />
            },
            {
                path: 'profile/:userId',
                element: <Profile />
            },
            {
                path: 'history',
                element: <Watched />
            },
            {
                path: 'liked',
                element: <Liked />
            },
            {
                path: 'sign-in',
                element: <Login />
            },
            {
                path: 'sign-up',
                element: <Register />
            },
```

Рис. 3.10 - Налаштування роутера

Якщо говорити про спілкування із сервером, то тут використовується бібліотека axios, яка дозволяє зручно здійснювати запити, налаштовувати хедери, які потрібні для авторизації. Цей хедер називається Authorization і у ньому знаходиться JWT токен, який очікує сервер для перевірки доступу користувача до певного ресурсу.

3.1.3 База даних

Сутностей у проекті 4, це:

- users

users	
email	varchar(255)
enabled	bit(1)
name	varchar(255)
password	varchar(255)
image_ur	varchar(255)
id	varchar(255)

- verification_tokens

verification_tokens	
confirmed_a	datetime(6)
created_at	datetime(6)
expires_at	datetime(6)
token	varchar(255)
user_id	varchar(255)
id	varchar(255)

- videos

videos	
created_at	datetime(6)
description	varchar(500)
preview_ur	varchar(255)
title	varchar(255)
video_ur	varchar(255)
user_id	varchar(255)
id	varchar(255)

- comments

comments	
created_at	datetime(6)
text	varchar(255)
user_id	varchar(255)
video_id	varchar(255)
id	varchar(255)

Та ще 2 таблиці, які слугують для реалізації відношення багато-до-багатьох:

- liked_videos
- disliked_videos

Структурна схема бази даних, яка показує відношення між таблицями наведена у Додатку 2.

ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було розглянуто архітектуру розробленого застосунку. Було розглянуто як саме побудовані всі частини проекту, та як вони взаємодіють між собою. Описано які технології використовує сервер та клієнт. Бекенд побудовано на Java з використанням фреймворку Spring Boot, Spring Data Jpa, Spring Security, Hibernate тощо. Фронтенд розроблений за допомогою React, ReactRouterDom, Vite, Axios, JWT та інші. Спілкування між цими двома компонентами відбувається через HTTP/HTTPS запити. До бази даних бекенд звертається за допомогою JDBC, під капотом якого знаходиться драйвер, спеціально підібраний під певну базу даних. Також було розглянуто налаштування підключення до хмарного сховища AWS S3 та реалізовано завантаження чи читання файлів. Всі ці компоненти забезпечують ефективність, масштабованість та підтримку даної системи.

РОЗДІЛ 4. ПРЕДСТАВЛЕННЯ ТА АНАЛІЗ ВЕБ-ЗАСТОСУНКУ

4.1 Демонстрація функціональності

У цьому розділі продемонстровано роботу розробленого веб-додатку. Всю функціональність додатку уявно можна поділити на декілька етапів:

- Реєстрація та авторизація користувачів;
- Редагування особистих даних користувачем;
- Завантаження нових відео;
- Можливість переглядати, та поширювати відео, ставити уподобання та підписуватись на інших користувачів;
- Можливість фільтрувати відео за темами, та здійснювати пошук.

Точка входу розробленого застосунку - головна сторінка, на якій розміщено шапку сторінки, бокове меню, меню для сортування, та самі відео.

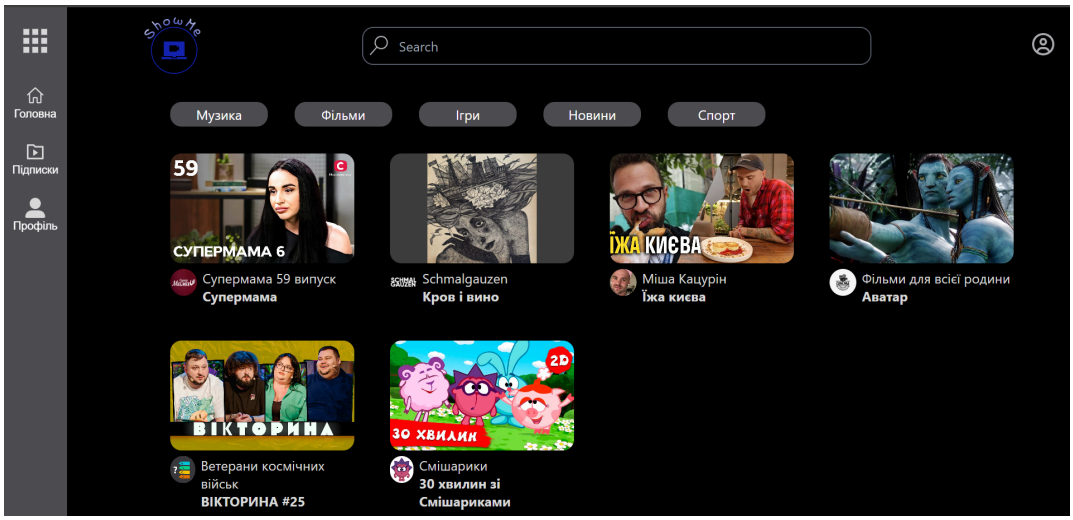


Рис. 4.1 - Головна сторінка

Для фільтрування відео по типам, достатньо вибрати та натиснути кнопку з відповідним типом під шапкою сторінки.

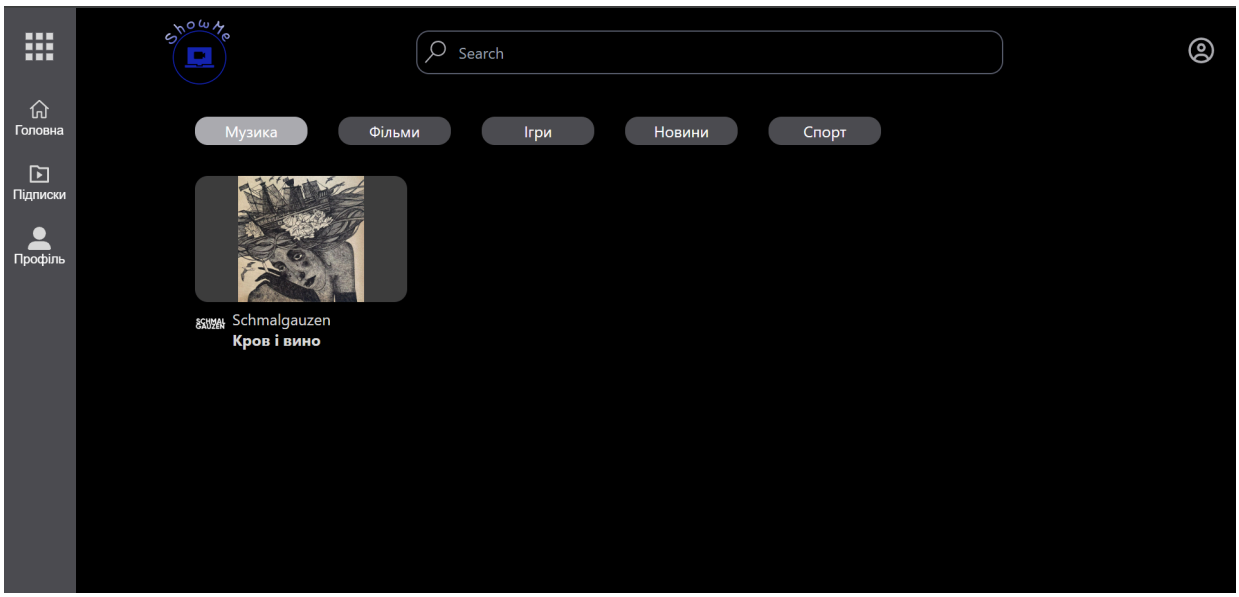


Рис. 4.2 - Сторінка відфільтрованих за типом відео

Для здійснення пошуку відео достатньо вписати запит у поле в шапці, та натиснути іконку лупи.

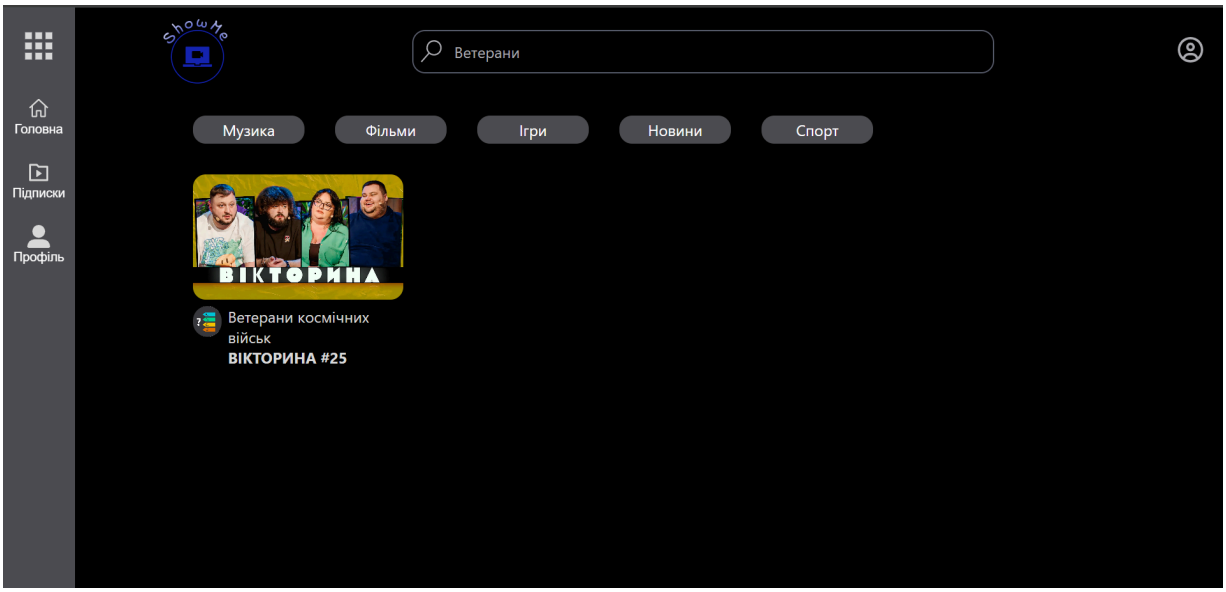


Рис. 4.3 - Сторінка знайдених за пошуком відео

Далі спробуємо зареєструватись. Перейдемо до сторінки реєстрації.

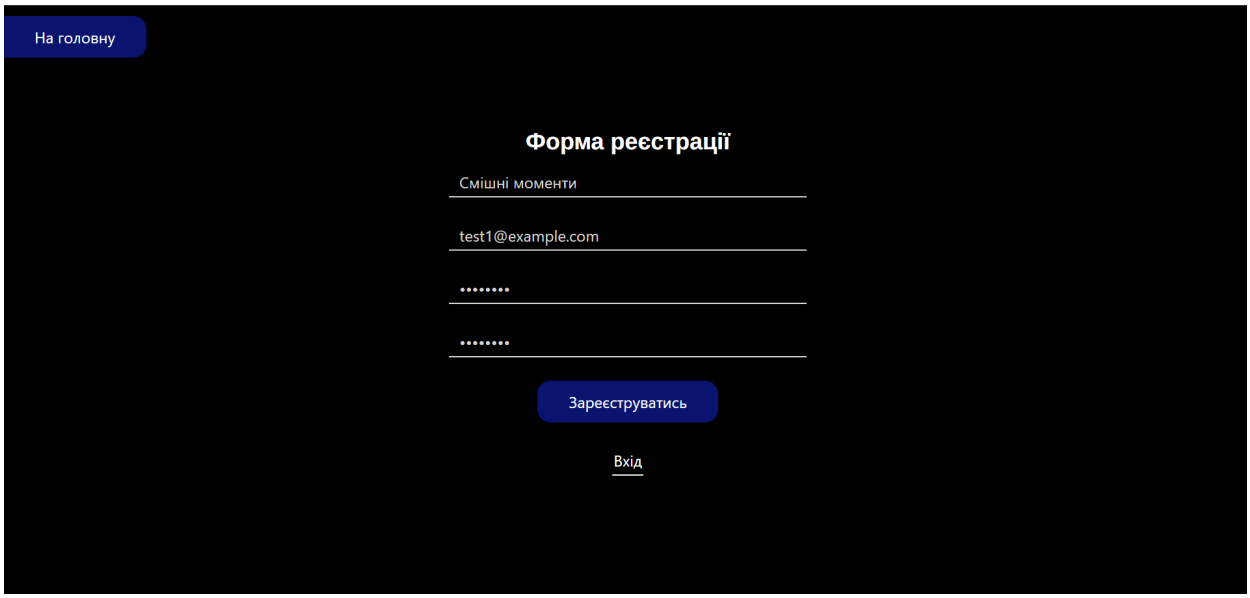


Рис. 4.4 - Сторінка реєстрації

Після спроби зареєструватись, переходимо на пошту, задля підтвердження активації аккаунта.

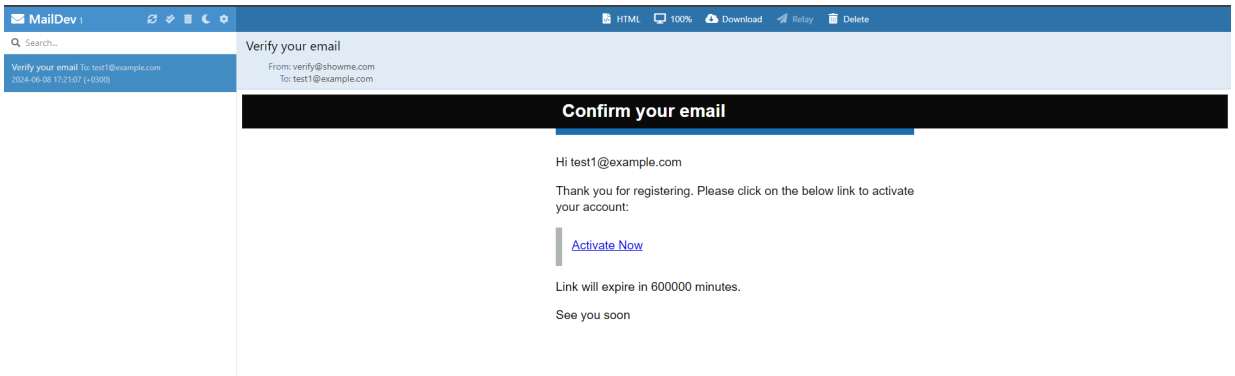


Рис. 4.5 - Підтвердження активації аккаунта

Після успішної активації аккаунта, спробуємо авторизуватись.

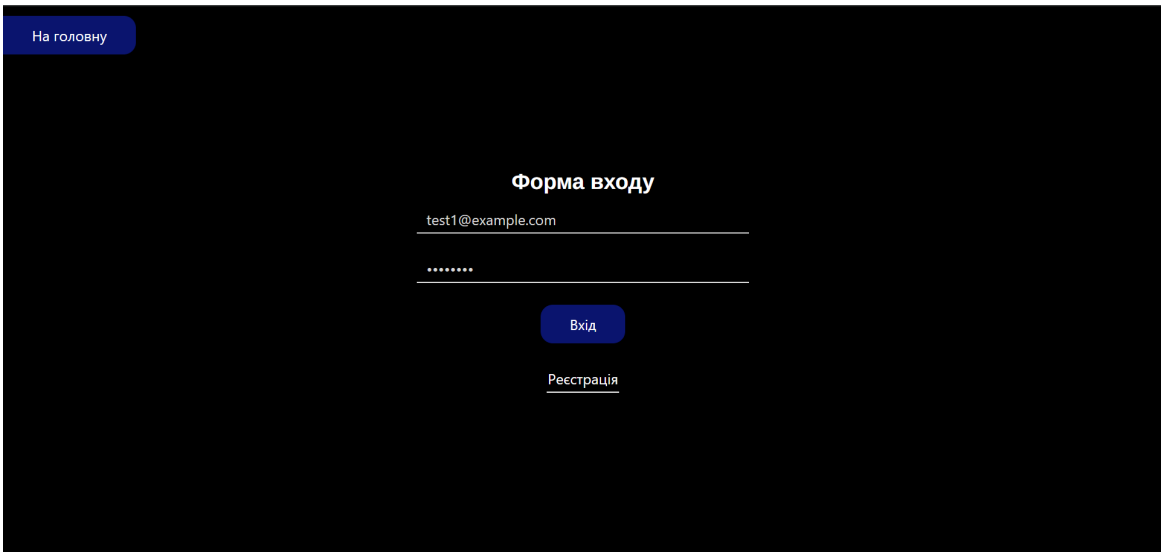


Рис. 4.6 - Сторінка авторизації користувача

Авторизовавшись, нам відкрився доступ до сторінки профіля. Там ми можемо змінити особисті дані, а також змінити зображення. Відповідно можемо додати нове відео. Щоб змінити дані, достатньо натиснути кнопку - “Редагувати”, а потім відповідно зберегти.

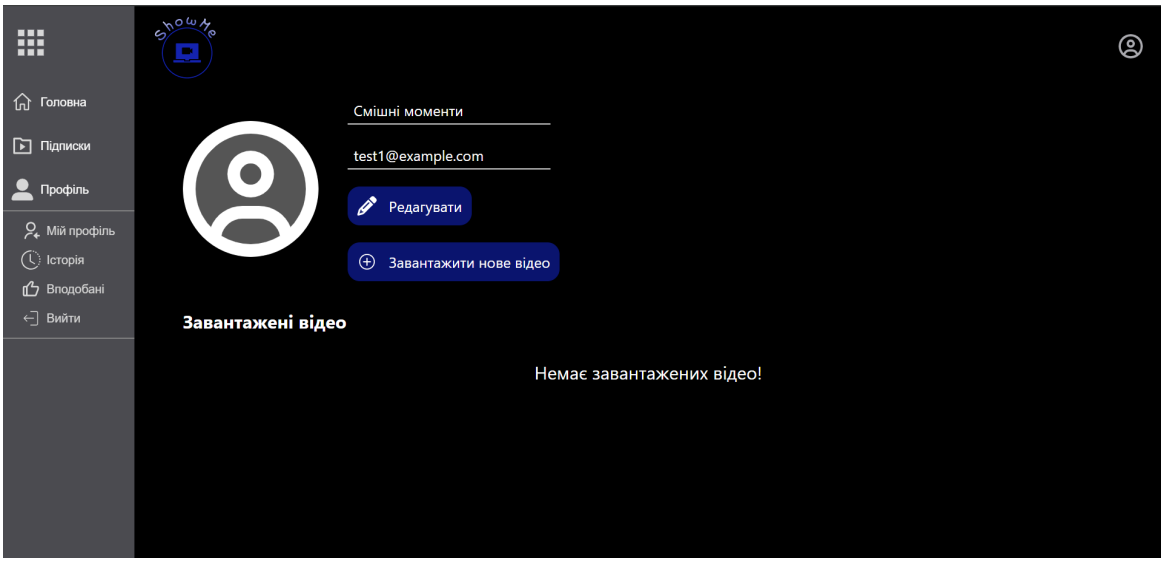


Рис. 4.7 - Сторінка профіля користувача

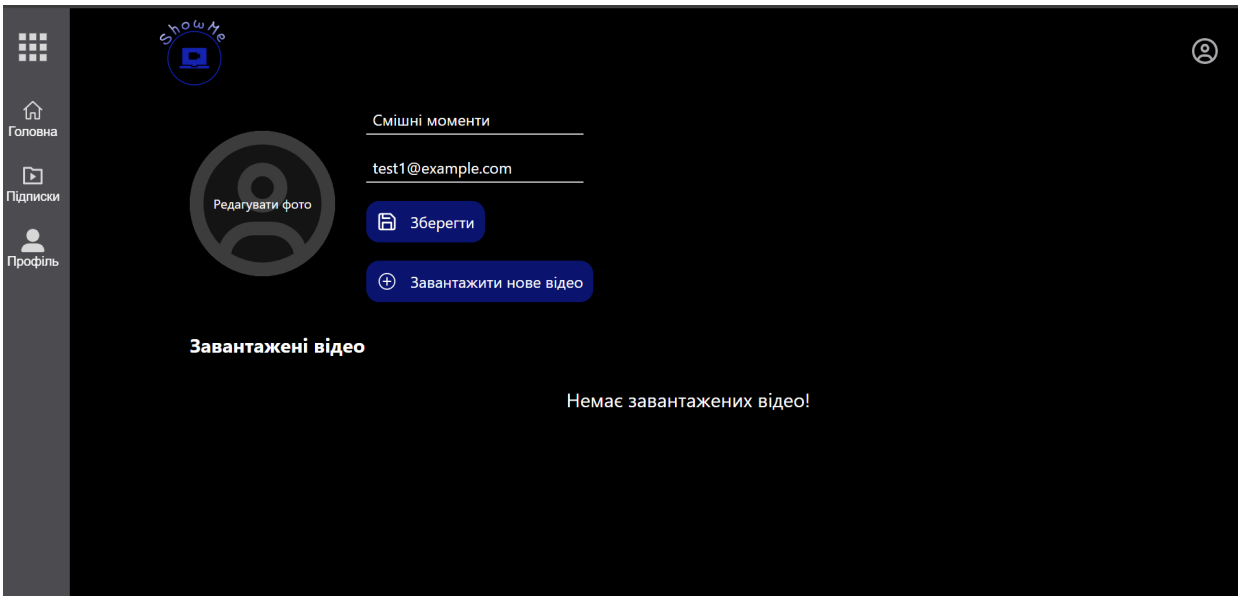


Рис. 4.8 - Сторінка профіля користувача з можливістю змінювати поля та редагувати фото

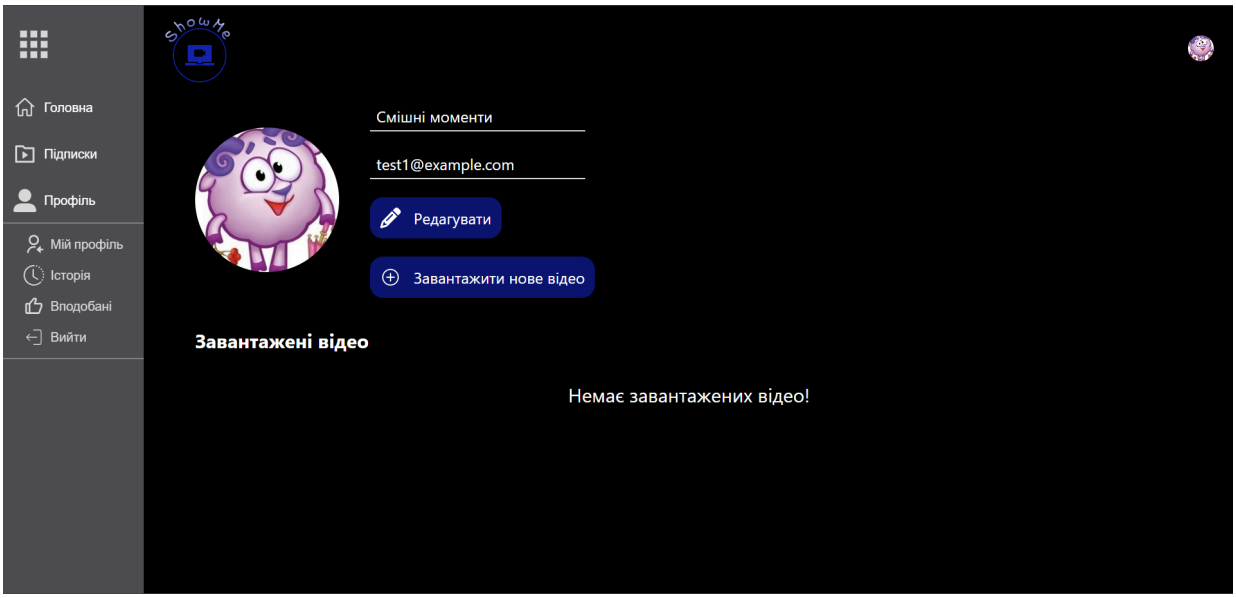


Рис. 4.9 - Сторінка профіля користувача з редагованими даними

Спробуємо додати нове відео. Для цього натиснемо кнопку - “Завантажити відео”, після чого з’явиться потрібна форма.

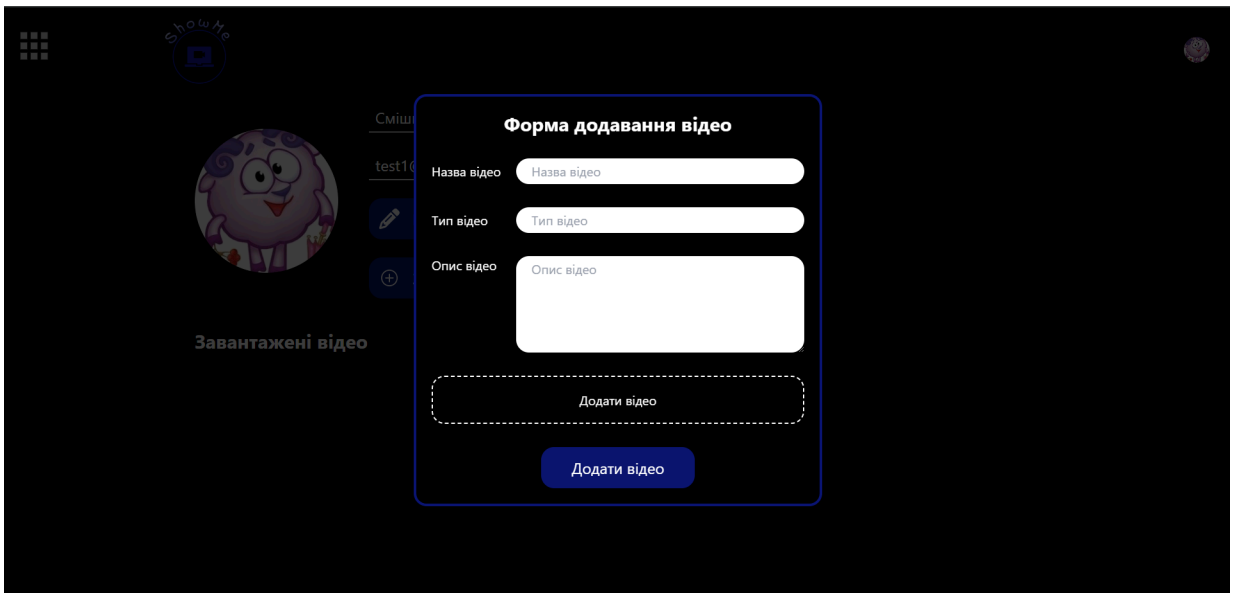


Рис. 4.10 - Форма завантаження відео

Після введення потрібних для створення відео, та натиснення кнопки додати відео, нове відео одразу з’явиться у профілі.

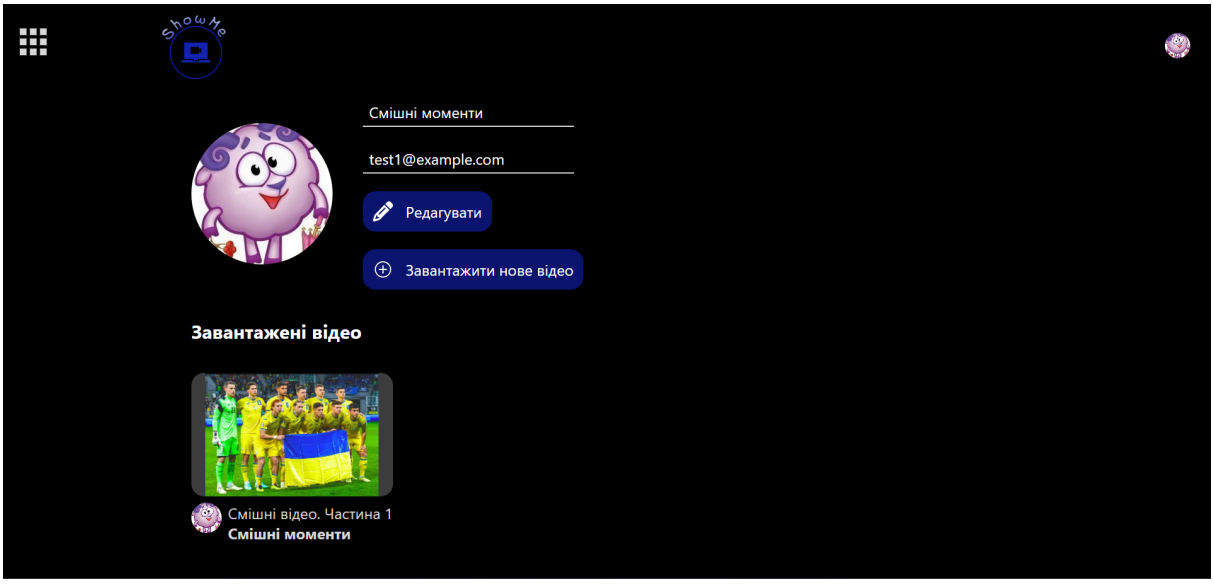


Рис. 4.11 - Профіль з доданим відео

Також, увійшовши в свій аккаунт, ми можемо переглядати історію переглядів, та сторінку своїх уподобань.

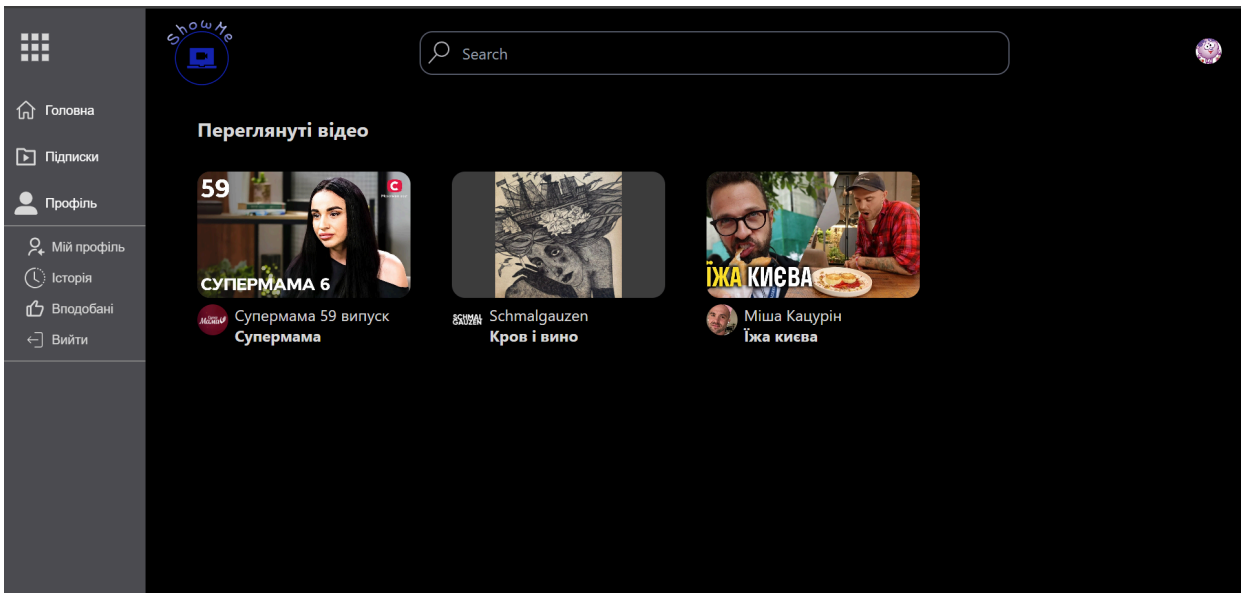


Рис. 4.12 - Сторінка переглянутих відео

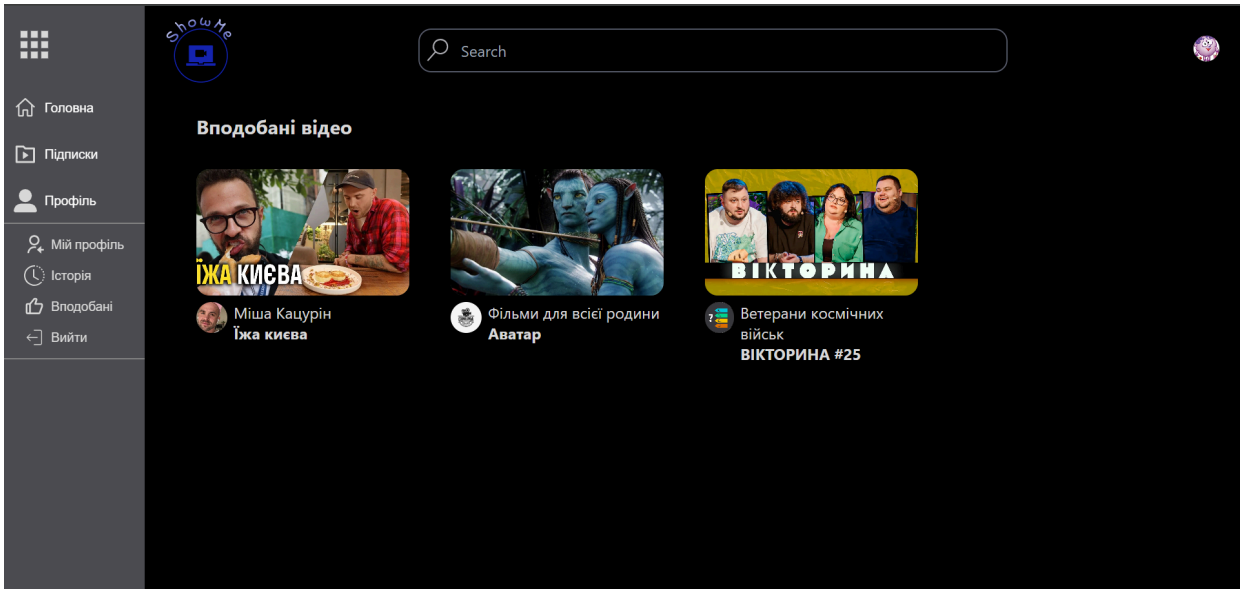


Рис. 4.13 - Сторінка вподобаних відео

Спробуємо переглянути якесь існуюче в додатку відео. Для цього оберемо яке саме відео, та перейдемо на сторінку перегляду відео.

Рис. 4.14 - Сторінка перегляду конкретного відео

У вікні перегляду відео, можна побачити саме відео, його назву, опис, посилання на канал, кнопки для лайка та дизлайка. Також є кнопки для підписки та поширення відео. Спробуємо підписатися.

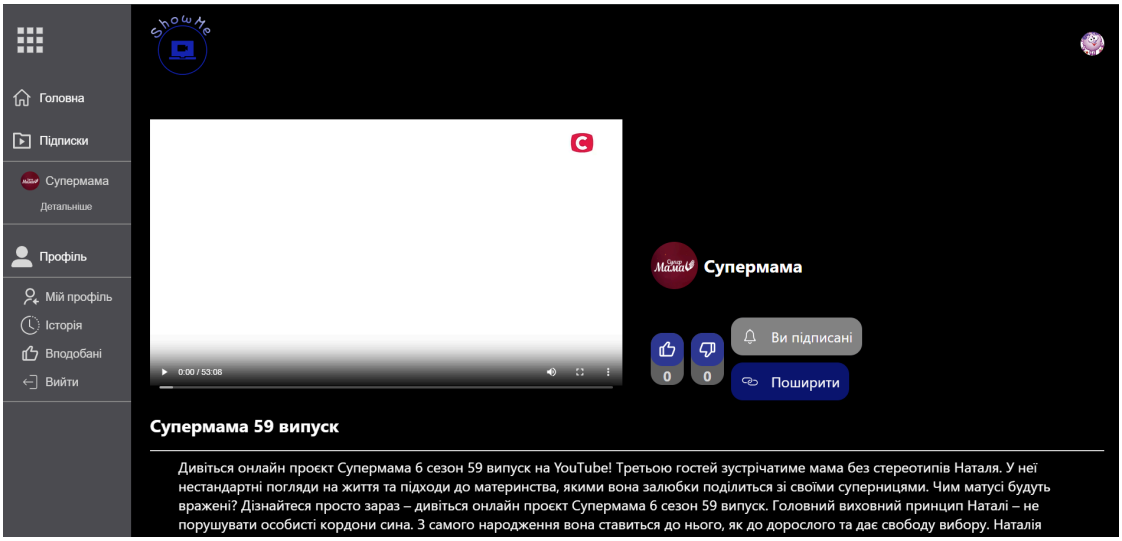


Рис. 4.15 - Сторінка перегляду відео після нажаття кнопки “Підписатися”

Спробуємо поставити лайк.

Рис. 4.16 - Сторінка перегляду відео після нажаття кнопки “Лайк”

Спробуємо поставити дизлайк.

Рис. 4.17 - Сторінка перегляду відео після нажаття кнопки “Дізлайк”

Перевіримо роботу кнопки - Поширити. Повідомлення зверху, сигналізує що посилання на відео було скопійовано.

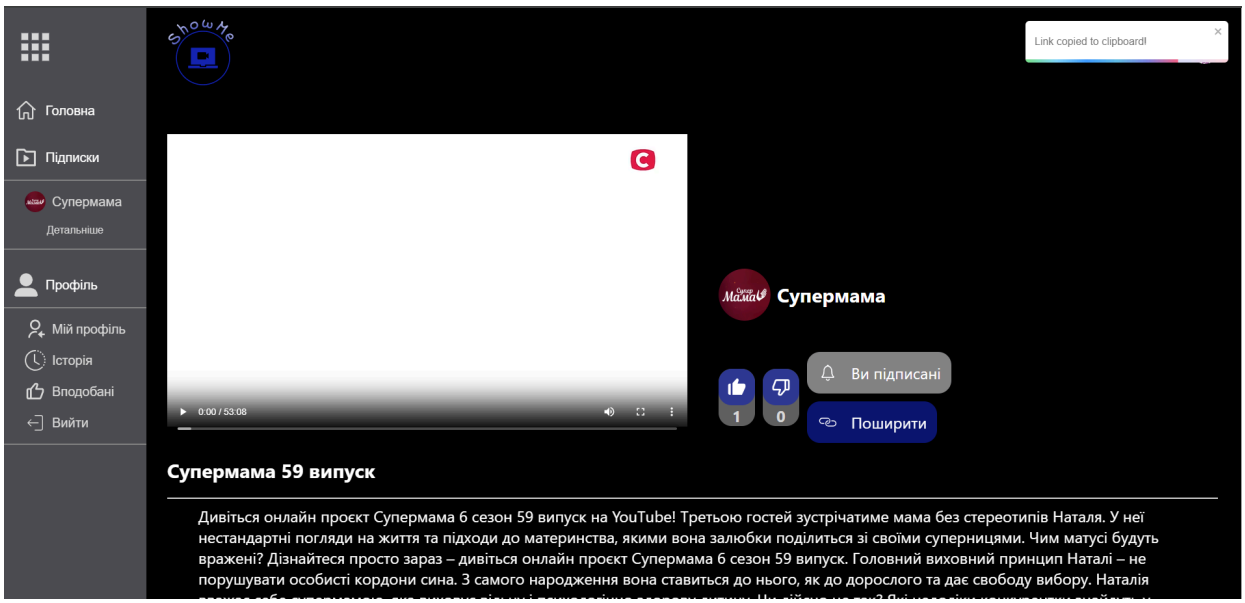


Рис. 4.18 - Сторінка перегляду відео після нажаття кнопки “Поширити”

Спробуємо перейти на сторінку іншого користувача.

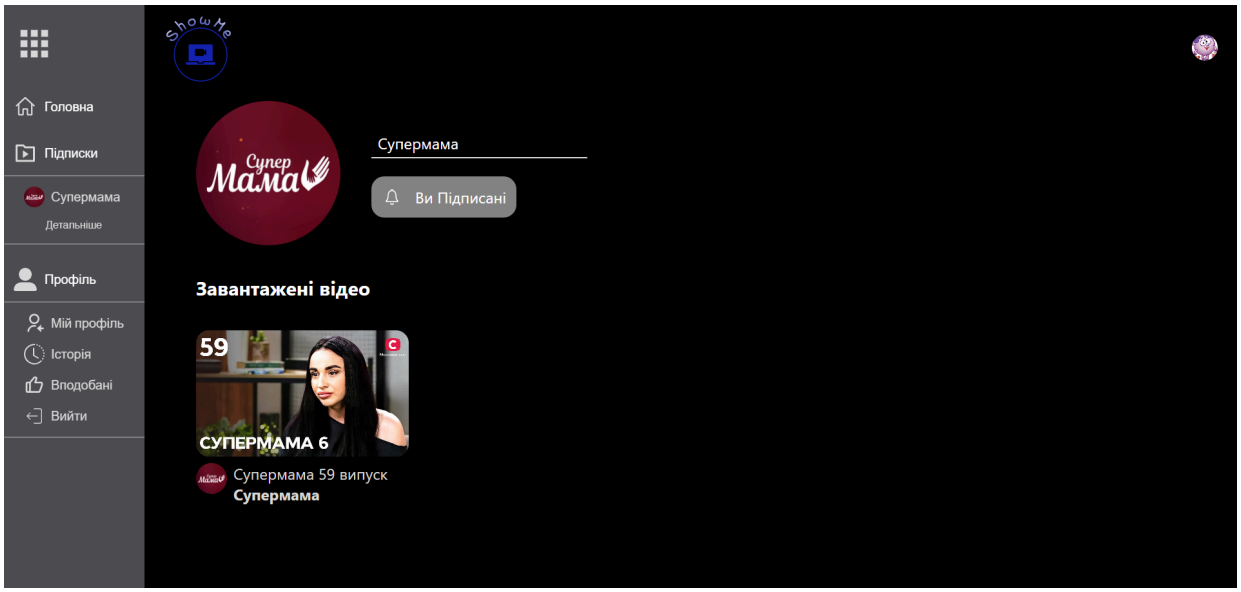


Рис. 4.19 - Сторінка профілю іншого користувача

Також можемо переглянути сторінку всіх підписок. Для цього достатньо на боковому меню натиснути кнопку детальніше.

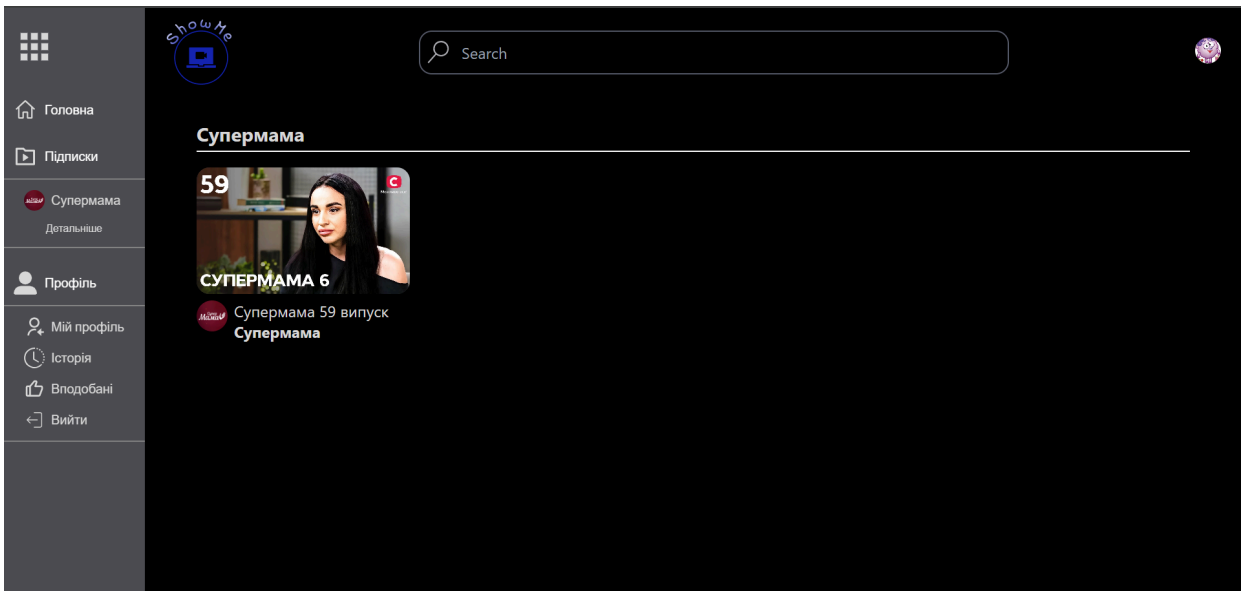


Рис. 4.20 - Сторінка підписок

Щоб вийти з акаунта, потрібно натиснути кнопку Вийти, після чого користувача перекине на головну сторінку.

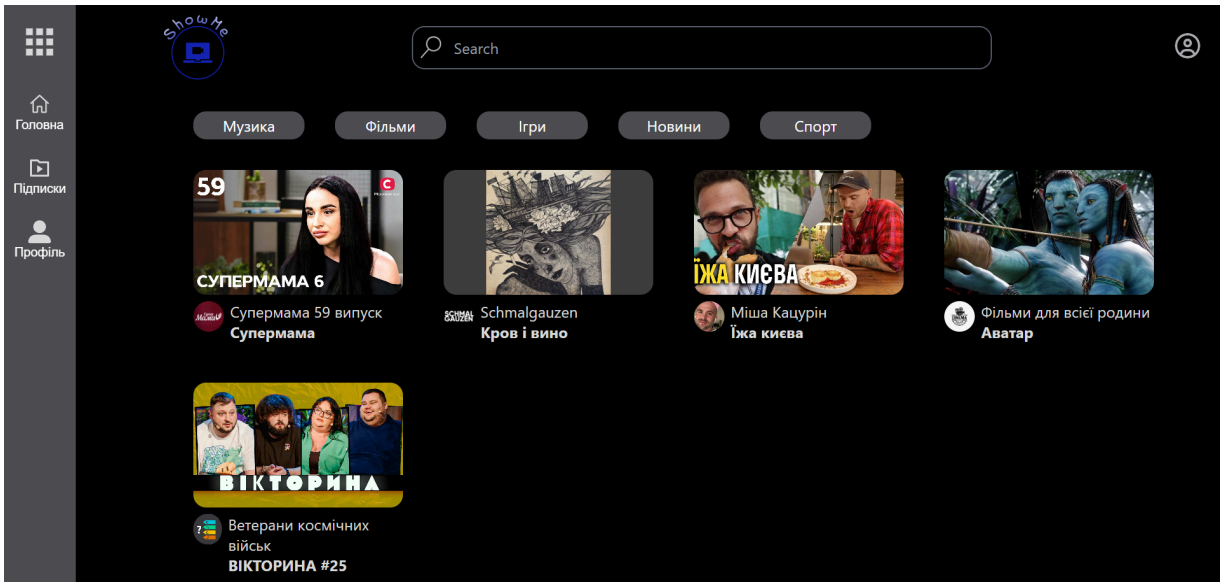


Рис. 4.21 - Головна сторінка, після нажаття кнопки вийти

4.2 Розвиток проекту

Для подальшого розвитку проекту "ShowMe" планується розширення функціоналу, включаючи можливості живих трансляцій, інтерактивних опитувань та інтеграції з соціальними мережами. Також буде вдосконалено аналітичні інструменти та впроваджено систему монетизації контенту. Поліпшення користувацького досвіду охоплює розробку мобільних додатків для iOS та Android, оптимізацію веб-інтерфейсу та забезпечення доступності для користувачів з обмеженими можливостями. Технічний розвиток зосередиться на масштабованості, безпеці та інтеграції з іншими сервісами.

ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі було наочно представлено функціонал платформи "ShowMe" та її основні можливості. В результаті демонстрації було показано, як користувачі можуть взаємодіяти з системою, завантажувати та переглядати відео, та оцінювати контент. Детально розглянуто користувацький інтерфейс та його основні компоненти, що забезпечують зручність та інтуїтивність використання платформи.

Під час демонстрації було підтверджено, що обрані технології та архітектурні рішення забезпечують високу продуктивність та надійність системи. Веб-застосунок успішно виконує свої функції, надаючи користувачам стабільний та приємний досвід взаємодії.

ВИСНОВКИ

Під час роботи над дипломним проектом було досягнуто кількох важливих результатів, які підтверджують успішність реалізації поставлених завдань.

Було проведено детальний аналіз існуючих платформ, що дозволило визначити ключові вимоги та функціональні можливості, які необхідно реалізувати в "ShowMe". Вивчення переваг та недоліків таких платформ як YouTube, Vimeo та Dailymotion допомогло сформувати оптимальну стратегію розвитку власного проекту.

На основі аналізу було обрано сучасні та надійні технології для реалізації проекту. Використання Java з Spring Boot для бекенду, React для фронтенду, MySQL для бази даних, а також Docker для контейнеризації і AWS для хмарного розгортання забезпечило високу продуктивність, масштабованість та безпеку системи.

У процесі розробки було створено повноцінну платформу для відеохостингу з можливістю завантаження, перегляду та оцінки відео. Інтерфейс користувача був розроблений з урахуванням принципів зручності та інтуїтивності, що забезпечує приємний досвід взаємодії з системою.

Проведена демонстрація веб-застосунку підтвердила функціональність та надійність розробленої платформи. Система успішно виконує всі заплановані функції та забезпечує стабільну роботу.

В ході роботи були визначені напрямки для подальшого розвитку проекту, включаючи розширення функціоналу, поліпшення користувацького досвіду, підвищення безпеки та інтеграцію з іншими сервісами. Це дозволить платформі "ShowMe" залишатися конкурентоспроможною та актуальною в умовах швидко змінюваного ринку технологій.

Узагальнюючи, можна стверджувати, що мета дипломного проекту була успішно досягнута. Розроблена платформа "ShowMe" є повнофункціональним рішенням для відеохостингу, яке відповідає сучасним вимогам та стандартам, забезпечуючи зручність, продуктивність та надійність для користувачів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

[1] – Java Tutorials (Oracle) [Електронний ресурс] – Режим доступу до ресурсу https://docs.oracle.com/javase/tutorial/

[2] – Spring Boot Documentation (Spring) [Електронний ресурс] – Режим доступу до ресурсу https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[3] – React Documentation (Facebook) [Електронний ресурс] – Режим доступу до ресурсу https://reactjs.org/docs/getting-started.html

[4] – Docker Documentation (Docker) [Електронний ресурс] – Режим доступу до ресурсу https://docs.docker.com/

[5] – MySQL Reference Manual (MySQL) [Електронний ресурс] – Режим доступу до ресурсу https://dev.mysql.com/doc/refman/8.0/en/

[6] – AWS Documentation (Amazon Web Services) [Електронний ресурс] – Режим доступу до ресурсу https://docs.aws.amazon.com/

[7] – Maven Documentation (Apache Maven) [Електронний ресурс] – Режим доступу до ресурсу https://maven.apache.org/guides/index.html

[8] – Vimeo Developer API (Vimeo) [Електронний ресурс] – Режим доступу до ресурсу https://developer.vimeo.com/

[9] – Baeldung (Spring and Java Tutorials) [Електронний ресурс] – Режим доступу до ресурсу https://www.baeldung.com/

[10] – YouTube [Електронний ресурс] – Режим доступу до ресурсу https://www.youtube.com/

[11] – Wikipedia - Monolithic architecture: https://en.wikipedia.org/wiki/Monolith

[12] – Wikipedia - Мікросервісна архітектура: https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%81%D0%B5%D1%80%D0%B2%D0%B8%D1%81%D0%BD%D0%B0%D1%8F_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0

[13] – Microservices.io: https://microservices.io/patterns/microservices

[14] – Martin Fowler - Microservices: https://martinfowler.com/microservices/

[15] – Spring Framework Guru (Spring Tutorials and Articles) [Електронний ресурс] – Режим доступу до ресурсу https://springframework.guru/

[16] – Jakarta EE Documentation [Електронний ресурс] – Режим доступу до ресурсу https://jakarta.ee/specifications/

[17] – Hibernate ORM Documentation [Електронний ресурс] – Режим доступу до ресурсу https://hibernate.org/orm/documentation/

[18] – Robert C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall, 2008.

[19] – Martin Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2002.

[20] – Eric Evans, Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2003.

Додаток 1

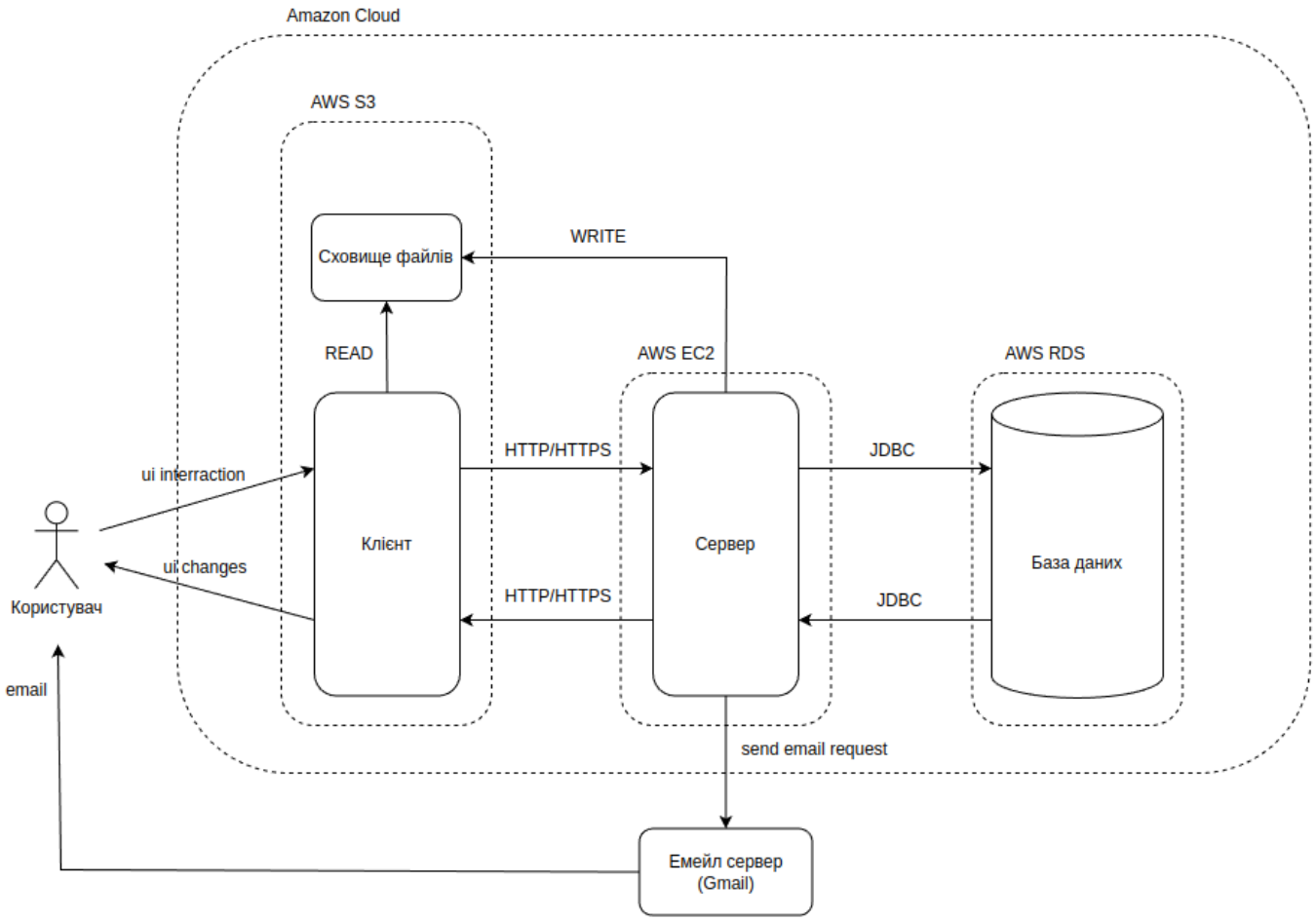
Платформа для відеохостингу ShowMe

Схема принципова (архітектурна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.004 Д1		
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*			
Розроб		Бондаренко Р. І.			*Платформа для відеохостингу ShowMe*	Літ.	Аркуш	Аркушів
Перев		Череватенко О. В.					1	1
Резценз					Схема принципова (архітектурна схема)	**КПІ ім. І. Сікорського, ФІОТ, Група ІО-03**	
Н. контр.		Іваніщев Б. В.					
Затверд							

Додаток 2

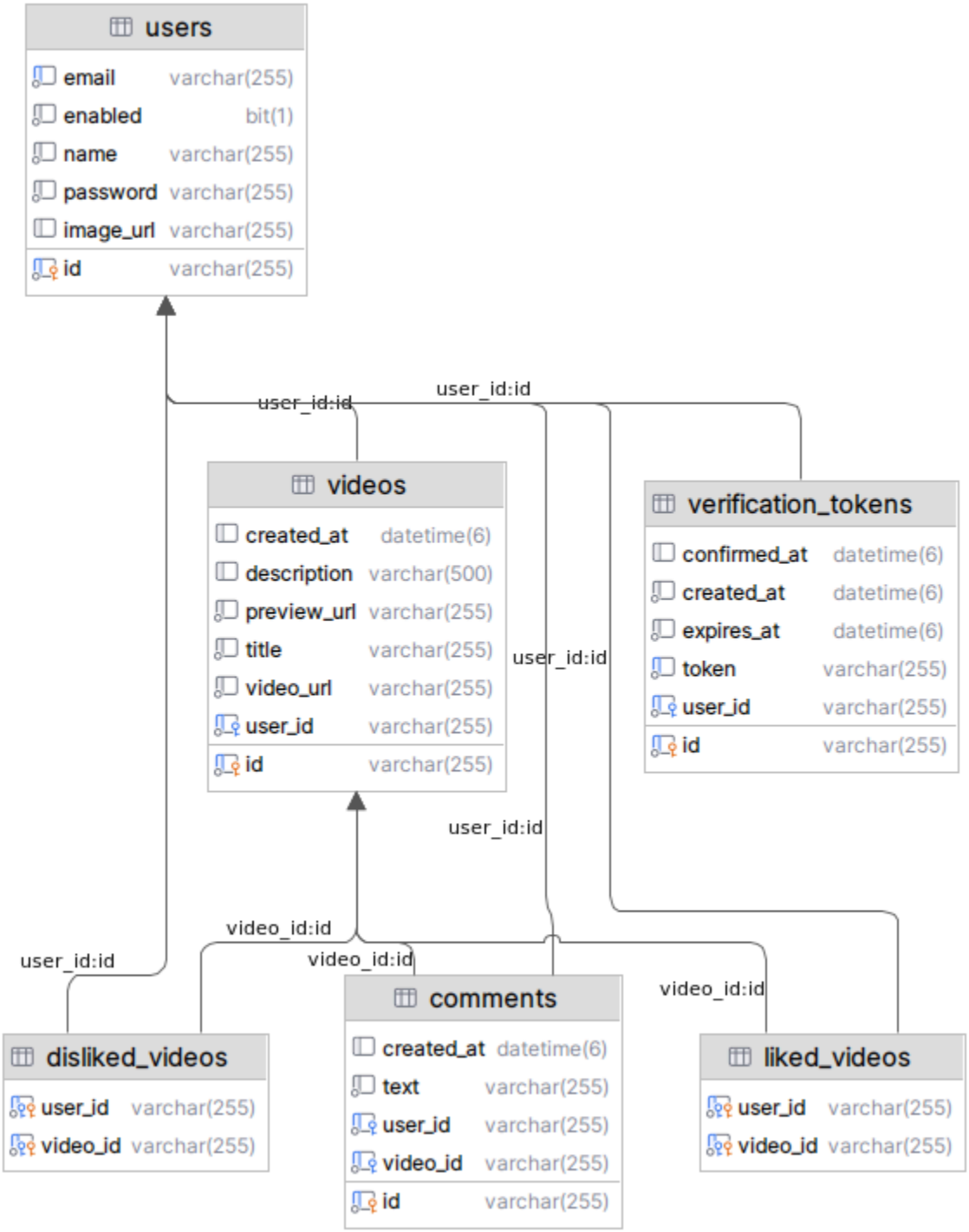
Платформа для відеохостингу ShowMe

Схема структурна (схема бази даних)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.005 Д2
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*	
Розроб		Бондаренко Р. І.			*Платформа для відеохостингу ShowMe*
Перев		Череватенко О. В.			
Резценз					Схема структурна (схема бази даних)
Н. контр.		Іваніщев Б. В.			
Затверд					

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. І. Сікорського, ФІОТ, Група ІО-03

Додаток 3

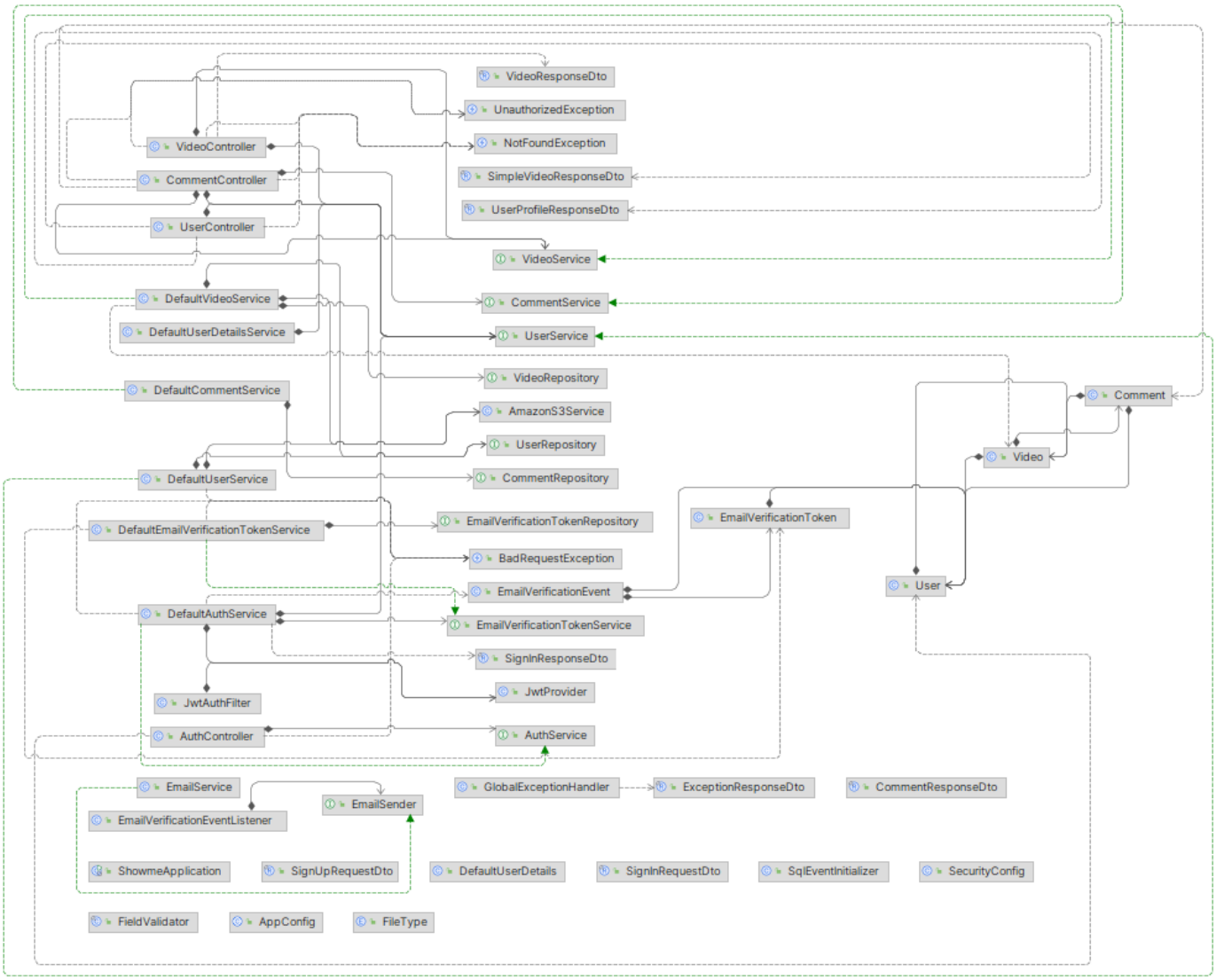
Платформа для відеохостингу ShowMe

Схема функціональна (діаграма класів)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.006 ДЗ			
Зм	*Лист*	*№ докум.*	*Підп*	*Дата*				
Розроб		Бондаренко Р. І.			*Платформа для відеохостингу ShowMe*	Літ.	Аркуш	Аркушів
Перев		Череватенко О. В.					1	1
Резценз					Схема функціональна (діаграма класів)	*КПІ ім. І. Сікорського, ФІОТ, Група ІО-03*		
Н. контр.		Іваніщев Б. В.						
Затверд								

Додаток 4

Платформа для відеохостингу ShowMe

Лістинг програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 39

Київ 2024 р

AppConfig.java

```
package com.itzroma.showme.config;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.client.builder.AwsClientBuilder;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import jakarta.servlet.MultipartConfigElement;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.web.servlet.MultipartConfigFactory;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.util.unit.DataSize;

@Configuration
public class AppConfig {
    @Value("${app.aws.s3.path}")
    private String path;

    @Value("${app.aws.s3.region}")
    private String region;

    @Value("${app.aws.s3.access-key}")
    private String accessKey;

    @Value("${app.aws.s3.secret-key}")
    private String secretKey;

    @Bean
    public AmazonS3 amazonS3() {
        return AmazonS3ClientBuilder.standard()
                .withEndpointConfiguration(new AwsClientBuilder.EndpointConfiguration(path,
region))
                .withCredentials(new AWSStaticCredentialsProvider(new
BasicAWSCredentials(accessKey, secretKey)))
                .withPathStyleAccessEnabled(true)
                .build();
    }

    @Bean
    public MultipartConfigElement multipartConfigElement() {
        MultipartConfigFactory factory = new MultipartConfigFactory();
        factory.setMaxFileSize(DataSize.ofBytes(512000000L));
        factory.setMaxRequestSize(DataSize.ofBytes(512000000L));
        return factory.createMultipartConfig();
    }
}
```

SecurityConfig.java

```
package com.itzroma.showme.config;

import com.itzroma.showme.security.JwtAuthFilter;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.ProviderManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebSecurity
public class SecurityConfig {
    private static final String[] PUBLIC_MATCHERS = {
            "/auth/**",
            "/test/all",
            "/swagger-ui/**",
            "/v3/api-docs/**"
    };

    @Value("${app.ui-path}")
    private String uiPath;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http, JwtAuthFilter
jwtAuthFilter) throws Exception {
        return http
                .csrf(AbstractHttpConfigurer::disable)
                .sessionManagement(config ->
config.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests(config -> config
                        .requestMatchers(PUBLIC_MATCHERS).permitAll()
                        .anyRequest().authenticated()
                )
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
                .build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager daoAuthenticationProvider(UserDetailsService
userDetailsService) {
        DaoAuthenticationProvider daoProvider = new DaoAuthenticationProvider();
        daoProvider.setUserDetailsService(userDetailsService);
        daoProvider.setPasswordEncoder(passwordEncoder());
        return new ProviderManager(daoProvider);
    }
```

```
    @Bean
    public WebMvcConfigurer corsConfigurer() {
        return new WebMvcConfigurer() {
            @Override
            public void addCorsMappings(CorsRegistry registry) {
                registry.addMapping("/**")
                        .allowedOrigins(uiPath)
                        .allowedHeaders(
                                HttpHeaders.AUTHORIZATION,
                                HttpHeaders.ACCESS_CONTROL_ALLOW_ORIGIN,
                                HttpHeaders.CONTENT_TYPE
                        )
                        .allowedMethods(
                                HttpMethod.GET.name(),
                                HttpMethod.POST.name(),
                                HttpMethod.PUT.name(),
                                HttpMethod.DELETE.name()
                        );
            }
        };
    }
}
```

AuthController.java

```
package com.itzroma.showme.controller;

import com.itzroma.showme.domain.dto.request.SignInRequestDto;
import com.itzroma.showme.domain.dto.request.SignUpRequestDto;
import com.itzroma.showme.domain.dto.response.SignInResponseDto;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.exception.BadRequestException;
import com.itzroma.showme.service.AuthService;
import io.swagger.v3.oas.annotations.Operation;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Objects;

@RestController
@RequiredArgsConstructor
@RequestMapping("/auth")
public class AuthController {
    private final AuthService authService;

    @Operation(summary = "Sign up user to the system and send an email verification request")
    @PostMapping("/sign-up")
    public ResponseEntity<String> signUp(@RequestBody SignUpRequestDto dto) {
        if (!Objects.equals(dto.password(), dto.confirmPassword())) {
            throw new BadRequestException("Passwords do not match");
        }
        User user = authService.signUpUser(new User(dto.name(), dto.email(), dto.password()));
        return ResponseEntity.ok("Sign up successful. Go to " + user.getEmail() + " to
activate an account.");
    }

    @Operation(summary = "Verify registration by the token that was sent in email")
    @GetMapping("/verify")
    public ResponseEntity<String> verify(@RequestParam("token") String token) {
        if (authService.verifyRegistration(token)) {
            return ResponseEntity.ok("User is verified. You can close this tab.");
        }
        return new ResponseEntity<>("Invalid email verification link.",
HttpStatus.BAD_REQUEST);
    }
```

```
    @Operation(summary = "Generate and return a JWT token that will be used to identify the
user")
    @PostMapping("/sign-in")
    public ResponseEntity<SignInResponseDto> signIn(@RequestBody SignInRequestDto dto) {
        return ResponseEntity.ok(authService.signInUser(dto.email(), dto.password()));
    }
}
```

CommentController.java

```
package com.itzroma.showme.controller;

import com.itzroma.showme.domain.entity.Comment;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.domain.entity.Video;
import com.itzroma.showme.exception.NotFoundException;
import com.itzroma.showme.exception.UnauthorizedException;
import com.itzroma.showme.service.CommentService;
import com.itzroma.showme.service.UserService;
import com.itzroma.showme.service.VideoService;
import io.swagger.v3.oas.annotations.Operation;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

@RestController
@RequiredArgsConstructor
@RequestMapping("/comments")
public class CommentController {
    private final CommentService commentService;
    private final UserService userService;
    private final VideoService videoService;

    @Operation(summary = "Add comment to the video")
    @PostMapping("/{videoId}")
    public ResponseEntity<String> addComment(@PathVariable("videoId") String videoId,
                                             @RequestBody String text,
                                             Authentication authentication) {
        User user = userService.findByEmail(authentication.getName()).orElseThrow(
                () -> new UnauthorizedException("Unauthorized")
        );
        Video video = videoService.findVideoById(videoId).orElseThrow(
                () -> new NotFoundException("Video [%s] not found")
        );
        commentService.addComment(new Comment(text, user, video));
        return ResponseEntity.ok("Comment added successfully");
    }
}
```

Userontroller.java

```
package com.itzroma.showme.controller;

import com.itzroma.showme.domain.dto.response.SimpleVideoResponseDto;
import com.itzroma.showme.domain.dto.response.UserProfileResponseDto;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.exception.NotFoundException;
import com.itzroma.showme.service.UserService;
import io.swagger.v3.oas.annotations.Operation;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;
```

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/users")
public class UserController {
    private final UserService userService;

    @Operation(summary = "Return user info by user id")
    @GetMapping("/{userId}")
    public ResponseEntity<UserProfileResponseDto> profile(@PathVariable String userId) {
        User user = userService.findById(userId).orElseThrow(
                () -> new NotFoundException("User [%s] not found".formatted(userId))
        );
        List<SimpleVideoResponseDto> videos = user.getMyVideos().stream()
                .map(video -> new SimpleVideoResponseDto(
                        video.getId(),
                        video.getPreviewUrl(),
                        video.getTitle(),
                        video.getAuthor().getId(),
                        video.getAuthor().getName()
                ))
                .toList();
        return ResponseEntity.ok(new UserProfileResponseDto(user.getId(), user.getName(),
user.getEmail(), user.getImageUrl(), videos));
    }

    @Operation(summary = "Update user's profile picture")
    @PostMapping("/{userId}/avatar")
    public ResponseEntity<String> uploadAvatar(@PathVariable String userId,
@RequestParam("file") MultipartFile file) {
        String newImageUrl = userService.updateImage(userId, file);
        return ResponseEntity.ok(newImageUrl);
    }
}
```

VideoController.java

```
package com.itzroma.showme.controller;

import com.itzroma.showme.domain.dto.response.VideoResponseDto;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.domain.entity.Video;
import com.itzroma.showme.exception.NotFoundException;
import com.itzroma.showme.exception.UnauthorizedException;
import com.itzroma.showme.service.UserService;
import com.itzroma.showme.service.VideoService;
import io.swagger.v3.oas.annotations.Operation;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

@RestController
@RequiredArgsConstructor
@RequestMapping("/videos")
public class VideoController {
    private final UserService userService;
    private final VideoService videoService;

    @Operation(summary = "Upload a video")
    @PostMapping
    public ResponseEntity<String> uploadVideo(@RequestParam("video") MultipartFile video,
                                              @RequestParam("preview") MultipartFile preview,
                                              @RequestParam("title") String title,
                                              @RequestParam("description") String description,
                                              Authentication authentication) {
        User user = userService.findByEmail(authentication.getName()).orElseThrow(
                () -> new UnauthorizedException("Unauthorized")
        );
```

```
        Video uploadedVideo = videoService.uploadVideo(video, preview, user, title,
description);
        return ResponseEntity.ok("Video uploaded: " + uploadedVideo.getTitle());
    }

    @Operation(summary = "Get video by id")
    @GetMapping("/{videoId}")
    public ResponseEntity<VideoResponseDto> getVideo(@PathVariable String videoId,
Authentication authentication) {
        Video video = videoService.findVideoById(videoId).orElseThrow(
                () -> new NotFoundException("Video [%s] not found".formatted(videoId))
        );
        VideoResponseDto videoResponseDto = new VideoResponseDto(
                video.getId(),
                video.getVideoUrl(),
                video.getTitle(),
                video.getDescription(),
                video.getAuthor().getId(),
                video.getAuthor().getName(),
                video.getAuthor().getImageUrl(),
                video.getLikes().size(),
                video.getLikes().stream().map(User::getEmail).anyMatch(s ->
s.equals(authentication.getName())),
                video.getDislikes().size(),
                video.getDislikes().stream().map(User::getEmail).anyMatch(s ->
s.equals(authentication.getName()))
        );
        return ResponseEntity.ok(videoResponseDto);
    }

    @Operation(summary = "Toggle like on the video by its id")
    @PostMapping("/{videoId}/toggle-like")
    public ResponseEntity<String> toggleLike(@PathVariable("videoId") String videoId,
Authentication authentication) {
        User user = userService.findByEmail(authentication.getName()).orElseThrow(
                () -> new UnauthorizedException("Unauthorized")
        );
        Video video = videoService.findVideoById(videoId).orElseThrow(
                () -> new NotFoundException("Video [%s] not found".formatted(videoId))
        );
        videoService.toggleLike(user, video);
        return ResponseEntity.ok("Like toggled on video: " + video.getTitle());
    }

    @Operation(summary = "Toggle dislike on the video by its id")
    @PostMapping("/{videoId}/toggle-dislike")
    public ResponseEntity<String> toggleDislike(@PathVariable("videoId") String videoId,
Authentication authentication) {
        User user = userService.findByEmail(authentication.getName()).orElseThrow(
                () -> new UnauthorizedException("Unauthorized")
        );
        Video video = videoService.findVideoById(videoId).orElseThrow(
                () -> new NotFoundException("Video [%s] not found".formatted(videoId))
        );
        videoService.toggleDislike(user, video);
        return ResponseEntity.ok("Dislike toggled on video: " + video.getTitle());
    }
}
```

Comment.java

```
package com.itzroma.showme.domain.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;

import java.time.LocalDateTime;
```

```
@Getter
@Entity
@Table(name = "comments")
@NoArgsConstructor
@AllArgsConstructor
public class Comment {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private String id;

    @Column(nullable = false)
    private String text;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User author;

    @ManyToOne
    @JoinColumn(name = "video_id", nullable = false)
    private Video video;

    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime createdAt;

    public Comment(String text, User author, Video video) {
        this.text = text;
        this.author = author;
        this.video = video;
    }

    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
    }
}
```

VerificationToke.java

```
package com.itzroma.showme.domain.entity;

import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;
import java.time.temporal.ChronoUnit;

@Getter
@Entity
@Table(name = "verification_tokens")
@NoArgsConstructor
public class VerificationToken {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private String id;

    @Column(unique = true, nullable = false)
    private String token;

    @Column(nullable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime createdAt;

    @Column(nullable = false)
    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime expiresAt;

    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime confirmedAt;
```

```
    @OneToOne(optional = false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    public VerificationToken(String token, User user) {
        this.token = token;
        this.user = user;
        createdAt = LocalDateTime.now();
        expiresAt = createdAt.plus(900_000, ChronoUnit.MILLIS);
    }
}
```

Video.java

```
package com.itzroma.showme.domain.entity;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import java.time.LocalDateTime;
import java.util.Set;

@Getter
@Entity
@Table(name = "videos")
@NoArgsConstructor
@AllArgsConstructor
public class Video {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private String id;
    @Column(nullable = false)
    private String videoUrl;
    @Column(nullable = false)
    private String previewUrl;
    @Column(nullable = false)
    private String title;
    @Column(length = 500)
    private String description;
    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User author;
    @ManyToMany(mappedBy = "likedVideos")
    private Set<User> likes;
    @ManyToMany(mappedBy = "dislikedVideos")
    private Set<User> dislikes;
    @OneToMany(mappedBy = "video")
    private Set<Comment> comments;
    @Temporal(TemporalType.TIMESTAMP)
    private LocalDateTime createdAt;

    public Video(String videoUrl, String previewUrl, String title, String description, User
author) {
        this.videoUrl = videoUrl;
        this.previewUrl = previewUrl;
        this.title = title;
        this.description = description;
        this.author = author;
    }

    @PrePersist
    protected void onCreate() {
        this.createdAt = LocalDateTime.now();
    }
}
```

JwtAuthFilter.java

```
package com.itzroma.showme.security;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import lombok.extern.log4j.Log4j2;
import org.apache.http.HttpHeaders;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Log4j2
@Component
@RequiredArgsConstructor
public class JwtAuthFilter extends OncePerRequestFilter {
    private final JwtService jwtService;
    private final UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(
            HttpServletRequest request,
            HttpServletResponse response,
            FilterChain filterChain
    ) throws ServletException, IOException {
        try {
            String jwt = getJwtFromRequest(request);

            if (StringUtils.hasText(jwt) && jwtService.validateJwt(jwt)) {
                String email = jwtService.getUserEmailFromToken(jwt);
                UserDetails userDetails = userDetailsService.loadUserByUsername(email);
                UsernamePasswordAuthenticationToken authentication =
                        new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        } catch (Exception ex) {
            log.error("Could not set user authentication in security context", ex);
        }

        filterChain.doFilter(request, response);
    }

    private String getJwtFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader(HttpHeaders.AUTHORIZATION);
        if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }
        return null;
    }
}
```

JwtService.java

```
package com.itzroma.showme.security;
```

```
import com.auth0.jwt.JWT;
import com.auth0.jwt.JWTVerifier;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.exceptions.JWTVerificationException;
import com.auth0.jwt.exceptions.SignatureVerificationException;
import com.auth0.jwt.exceptions.TokenExpiredException;
import jakarta.annotation.PostConstruct;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import java.util.Date;

@Slf4j
@Service
public class JwtService {
    @Value("${app.security.jwt.access.secret}")
    private String accessSecret;

    @Value("${app.security.jwt.access.expiration-ms}")
    private Long expirationMs;

    private Algorithm accessAlgorithm;

    @PostConstruct
    private void init() {
        accessAlgorithm = Algorithm.HMAC512(accessSecret);
    }

    public String generateAccessToken(MyUserDetails authUserDetails) {
        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + expirationMs);

        return JWT.create()
                .withSubject(authUserDetails.getUsername())
                .withIssuedAt(now)
                .withExpiresAt(expiryDate)
                .sign(accessAlgorithm);
    }

    public String getUserEmailFromToken(String token) {
        JWTVerifier verifier = JWT.require(accessAlgorithm).build();
        return verifier.verify(token).getSubject();
    }

    public boolean validateJwt(String token) {
        try {
            JWT.require(accessAlgorithm).build().verify(token);
            return true;
        } catch (SignatureVerificationException ex) {
            log.error("Invalid JWT signature: {}", ex.getMessage());
        } catch (TokenExpiredException ex) {
            log.error("JWT token is expired: {}", ex.getMessage());
        } catch (JWTVerificationException ex) {
            log.error("Invalid JWT token: {}", ex.getMessage());
        } catch (Exception ex) {
            log.error("JWT token validation failed: {}", ex.getMessage());
        }
        return false;
    }
}
```

MyUserDetails.java

```
package com.itzroma.showme.security;

import lombok.Builder;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
```

```
import java.util.Collection;
import java.util.Collections;

@Builder
public class MyUserDetails implements UserDetails {
    private String email;
    private String password;
    private Boolean enabled;
    private Collection<? extends GrantedAuthority> authorities;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities == null ? Collections.emptyList() : authorities;
    }

    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return enabled;
    }
}
```

MyUserDetailsService.java

```
package com.itzroma.showme.security;

import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;

import java.util.Collections;

@Service
@RequiredArgsConstructor
public class MyUserDetailsService implements UserDetailsService {
    private final UserService userService;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userService.findByEmail(username).orElseThrow(() -> new
```

```
UsernameNotFoundException("User not found"));
        return MyUserDetails.builder()
                .email(user.getEmail())
                .password(user.getPassword())
                .enabled(user.isEnabled())
                .authorities(Collections.emptyList())
                .build();
    }
}
```

AmazonS3Service.java

```
package com.itzroma.showme.service;

import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.model.CannedAccessControlList;
import com.amazonaws.services.s3.model.DeleteObjectRequest;
import com.amazonaws.services.s3.model.PutObjectRequest;
import com.itzroma.showme.domain.FileType;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.apache.commons.io.FilenameUtils;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.UUID;

@Slf4j
@Service
@RequiredArgsConstructor
public class AmazonS3Service {
    private final AmazonS3 s3Client;

    @Value("${app.aws.s3.path}")
    private String path;

    @Value("${app.aws.s3.bucket-name}")
    private String bucketName;

    public String uploadFile(MultipartFile multipartFile, FileType fileType, String userId) {
        try {
            String fileName = constructFileName(multipartFile, fileType, userId);
            File file = convertMultipartFileToIOFile(multipartFile);
            uploadFileToS3Bucket(fileName, file);
            file.delete();
            return path + "/" + bucketName + "/" + fileName;
        } catch (IOException e) {
            log.error("Failed to upload file to S3 bucket: {}", e.getLocalizedMessage());
            throw new RuntimeException("Failed to upload file to S3 bucket: " +
e.getLocalizedMessage(), e);
        }
    }

    private String constructFileName(MultipartFile multipartFile, FileType fileType, String
userId) {
        String commonFileName = fileType.getFolderName() + "/" + userId;
        if (fileType == FileType.AVATAR) {
            return commonFileName + "." +
FilenameUtils.getExtension(multipartFile.getOriginalFilename());
        }
        return commonFileName + "/" + System.currentTimeMillis() + "__" +
multipartFile.getOriginalFilename();
    }
```

```
    private File convertMultipartFileToIOFile(MultipartFile multipartFile) throws IOException
{
        String originalFilename = multipartFile.getOriginalFilename();
        File converted = new File(originalFilename == null ? UUID.randomUUID().toString() :
originalFilename);
        FileOutputStream fos = new FileOutputStream(converted);
        fos.write(multipartFile.getBytes());
        fos.close();
        return converted;
    }

    private void uploadFileToS3Bucket(String fileName, File file) {
        if (s3Client.doesObjectExist(bucketName, fileName)) {
            s3Client.deleteObject(new DeleteObjectRequest(bucketName, fileName));
        }
        s3Client.putObject(new PutObjectRequest(bucketName, fileName, file)
                .withCannedAcl(CannedAccessControlList.PublicRead));
    }
}
```

AuthServiceImpl.java

```
package com.itzroma.showme.service;

import com.itzroma.showme.domain.dto.response.SignInResponseDto;
import com.itzroma.showme.domain.entity.VerificationToken;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.email.event.EmailVerificationEvent;
import com.itzroma.showme.exception.BadRequestException;
import com.itzroma.showme.security.JwtService;
import com.itzroma.showme.security.MyUserDetails;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.UUID;

@Service
@RequiredArgsConstructor
public class AuthServiceImpl implements AuthService {
    private final EmailVerificationTokenService emailVerificationTokenService;
    private final AuthenticationManager authenticationManager;
    private final UserDetailsService userDetailsService;
    private final ApplicationEventPublisher publisher;
    private final PasswordEncoder passwordEncoder;
    private final UserService userService;
    private final JwtService jwtService;

    @Value("${app.path}")
    private String appPath;

    @Override
    public User signUpUser(User user) {
        User savedUser = userService.save(user);
        VerificationToken verificationToken =
                emailVerificationTokenService.generateEmailVerificationToken(user);
        sendEmailVerificationToken(savedUser, verificationToken);
        return savedUser;
    }

    @Override
    public boolean verifyRegistration(String token) {
```

```
        VerificationToken verificationToken =
emailVerificationTokenService.validateEmailVerificationToken(token);
        userService.enable(UUID.fromString(verificationToken.getUser().getId()));
        return true;
    }

    @Override
    public SignInResponseDto signInUser(String email, String password) {
        UserDetails userDetails = userDetailsService.loadUserByUsername(email);
        validateLogin(password, userDetails);
        User user = userService.findByEmail(email).orElseThrow(() -> new
BadRequestException("Bad Credentials"));
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(userDetails.getUsername(), password)
        );
        return new SignInResponseDto(user.getId(),
jwtService.generateAccessToken((MyUserDetails) authentication.getPrincipal()));
    }

    private void sendEmailVerificationToken(User forUser, VerificationToken token) {
        publisher.publishEvent(new EmailVerificationEvent(forUser, appPath + "/auth", token));
    }

    private void validateLogin(String password, UserDetails userDetails) {
        if (!passwordEncoder.matches(password, userDetails.getPassword())) {
            throw new BadRequestException("Bad Credentials");
        }
        if (!userDetails.isEnabled()) {
            throw new BadRequestException("Your account is disabled");
        }
    }
}
```

CommentServiceImpl.java

```
package com.itzroma.showme.service;

import com.itzroma.showme.domain.entity.Comment;
import com.itzroma.showme.repository.CommentRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class CommentServiceImpl implements CommentService {
    private final CommentRepository commentRepository;

    @Override
    public Comment addComment(Comment comment) {
        return commentRepository.save(comment);
    }
}
```

VerificationTokenServiceImpl.java

```
package com.itzroma.showme.service;

import com.itzroma.showme.domain.entity.VerificationToken;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.exception.BadRequestException;
import com.itzroma.showme.repository.EmailVerificationTokenRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import java.time.LocalDateTime;
import java.util.Optional;
import java.util.UUID;
```

```
@Service
@RequiredArgsConstructor
public class VerificationTokenServiceImpl implements VerificationTokenService {
    private final EmailVerificationTokenRepository emailVerificationTokenRepository;

    @Override
    public Optional<VerificationToken> findByToken(String token) {
        return emailVerificationTokenRepository.findByToken(token);
    }

    @Override
    public VerificationToken generateEmailVerificationToken(User user) {
        return emailVerificationTokenRepository.save(
                new VerificationToken(UUID.randomUUID().toString(), user)
        );
    }

    @Override
    @Transactional
    public VerificationToken validateEmailVerificationToken(String token) {
        VerificationToken verificationToken = findByToken(token).orElseThrow(() -> {
            return new BadRequestException("Email verification token [%s] not
found".formatted(token));
        });
        validateToken(verificationToken);
        emailVerificationTokenRepository.confirm(token, LocalDateTime.now());
        return verificationToken;
    }

    private static void validateToken(VerificationToken verificationToken) {
        if (verificationToken.getConfirmedAt() != null) {
            throw new BadRequestException("Email is already confirmed");
        }
        if (verificationToken.getExpiresAt().isBefore(LocalDateTime.now())) {
            throw new BadRequestException("Email verification token is expired");
        }
    }
}
```

UserServiceImpl.java

```
package com.itzroma.showme.service;

import com.itzroma.showme.domain.FileType;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.exception.BadRequestException;
import com.itzroma.showme.repository.UserRepository;
import com.itzroma.showme.util.FieldValidator;
import lombok.RequiredArgsConstructor;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import java.util.Optional;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final AmazonS3Service amazonS3Service;

    @Override
    public User save(User user) {
        validateUser(user);
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        return userRepository.save(user);
    }
```

```
    @Override
    public Optional<User> findById(String id) {
        return userRepository.findById(id);
    }

    @Override
    public Optional<User> findByEmail(String email) {
        return userRepository.findByEmail(email);
    }

    @Override
    public void enable(UUID id) {
        userRepository.enable(id.toString());
    }

    @Override
    public String updateImage(String userId, MultipartFile file) {
        String imageUrl = amazonS3Service.uploadFile(file, FileType.AVATAR, userId);
        userRepository.updateImage(userId, imageUrl);
        return imageUrl;
    }

    private void validateUser(User user) {
        if (FieldValidator.validateStringEmpty(user.getName())) {
            throw new BadRequestException("Name is required");
        }
        if (FieldValidator.validateStringEmpty(user.getEmail())) {
            throw new BadRequestException("Email is required");
        }
        if (FieldValidator.validateStringEmpty(user.getPassword())) {
            throw new BadRequestException("Password is required");
        }
        if (!FieldValidator.validateEmail(user.getEmail())) {
            throw new BadRequestException("Invalid email: " + user.getEmail());
        }
        if (userRepository.existsByEmail(user.getEmail())) {
            throw new BadRequestException("Email already in use: " + user.getEmail());
        }
        if (!FieldValidator.validatePassword(user.getPassword(), false)) {
            throw new BadRequestException("Invalid password: " + user.getPassword());
        }
    }
}
```

AmazonS3ServiceImpl.java

```
package com.itzroma.showme.service;

import com.itzroma.showme.domain.FileType;
import com.itzroma.showme.domain.entity.User;
import com.itzroma.showme.domain.entity.Video;
import com.itzroma.showme.repository.UserRepository;
import com.itzroma.showme.repository.VideoRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.web.multipart.MultipartFile;
import java.util.Optional;

@Service
@RequiredArgsConstructor
public class VideoServiceImpl implements VideoService {
    private final AmazonS3Service s3Service;
    private final VideoRepository videoRepository;
    private final UserRepository userRepository;

    @Override
    public Video uploadVideo(MultipartFile video, MultipartFile preview, User author, String
title, String description) {
```

```
        String videoUrl = s3Service.uploadFile(video, FileType.VIDEO, author.getId());
        String previewUrl = s3Service.uploadFile(preview, FileType.PREVIEW, author.getId());
        return videoRepository.save(new Video(videoUrl, previewUrl, title, description,
author));
    }

    @Override
    public Optional<Video> findVideoById(String id) {
        return videoRepository.findById(id);
    }

    @Override
    public void toggleLike(User user, Video video) {
        if (video.getDislikes().contains(user)) {
            video.getDislikes().remove(user);
            user.getDislikedVideos().remove(video);
        }

        if (video.getLikes().contains(user)) {
            video.getLikes().remove(user);
            user.getLikedVideos().remove(video);
        } else {
            video.getLikes().add(user);
            user.getLikedVideos().add(video);
        }
        videoRepository.save(video);
        userRepository.save(user);
    }

    @Override
    public void toggleDislike(User user, Video video) {
        if (video.getLikes().contains(user)) {
            video.getLikes().remove(user);
            user.getLikedVideos().remove(video);
        }

        if (video.getDislikes().contains(user)) {
            video.getDislikes().remove(user);
            user.getDislikedVideos().remove(video);
        } else {
            video.getDislikes().add(user);
            user.getDislikedVideos().add(video);
        }
        videoRepository.save(video);
        userRepository.save(user);
    }
}
```

ShowmeApplication.java

```
package com.itzroma.showme;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class ShowmeApplication {
    public static void main(String[] args) {
        SpringApplication.run(ShowmeApplication.class, args);
    }
}
```

axios.api.ts

```
import axios from "axios";
import { getTokenFromLocalStorage } from "../helpers/localstorage.helper";

export const instance = axios.create({
    baseURL: "http://localhost:8080",
    headers: {
        Authorization: 'Bearer ' + getTokenFromLocalStorage(),
    }
})
```

BlueButton.tsx

```
import { FC, ReactNode } from "react";

interface Props {
    type: "submit" | "button";
    children: ReactNode;
    onClick?: () => void;
    styles?: string
}

const BlueButton: FC<Props> = ({type, children, onClick, styles}) => {
    return (
        <div className="flex justify-center">
            <button type={type} className={`bg-dark-blue/65 hover:bg-dark-blue/100 px-12 py-4
text-2xl text-white ${styles ? styles : "rounded-20"}`} onClick={onClick}>{children}</button>
        </div>
    )
}

export default BlueButton;
```

ChannelButton.tsx

```
import { FC, ReactNode } from "react";
import { Link } from "react-router-dom";

interface Props {
    path: string
    image: string;
    children: ReactNode
}

const ChannelButton: FC<Props> = ({path, children, image}) => {
    return (
        <Link to={path} className="flex items-center gap-2.5">
            <div className="w-8 h-8 rounded-full flex justify-center items-center
overflow-hidden">
                <img src={image} alt={image} className="w-full h-full object-cover" />
            </div>
            <div className="text-xl">
                {children}
            </div>
        </Link>
    )
}

export default ChannelButton;
```

FilterButton.tsx

```
import { FC, ReactNode } from "react";

interface Props {
    children: ReactNode
}

const FilterButton: FC<Props> = ({ children }) => {
    return (
        <div className="flex">
            <button className="px-11 py-1.5 bg-dark-gray/45 text-white text-2xl rounded-20
min-w-[175px]">{children}</button>
        </div>
    )
}

export default FilterButton
```

LikedButton.tsx

```
import { FC } from "react";

interface Props {
    count: number;
    onClick: () => void;
    children: JSX.Element
}

const LikedButton: FC<Props> = ({count, onClick, children}) => {
    return (
        <div className="rounded-20 overflow-hidden bg-light-gray/45 max-w-14">
            <div className="p-3 bg-dark-blue/65 text-white rounded-20 overflow-hidden h-14">
                <button onClick={onClick}>
                    {children}
                </button>
            </div>
            <div className="text-center text-light-gray text-2xl font-bold">
                {count}
            </div>
        </div>
    )
}

export default LikedButton;
```

LinkButton.tsx

```
import { FC, ReactNode } from "react";

interface Props {
    children: ReactNode,
    type: "submit" | "button",
    onClick: () => void,
}

const LinkButton: FC<Props> = ({children, type, onClick}) => {
    return (
        <div className="flex flex-col items-center">
            <button onClick={onClick} type={type} className="py-1.5 border-b-2 border-white
px-0.5 text-2xl text-white">{children}</button>
        </div>
    )
}

export default LinkButton
```

ProfileButton.tsx

```
import { FC } from "react";
import { Link } from "react-router-dom";

interface Props {
    path?: string
    children: JSX.Element
    title: string
    onClick?: () => void
}

const ProfileButton: FC<Props> = ({path, children, title, onClick}) => {
    return (
        <>
            {path && (
                <Link to={path} className="flex items-center gap-2.5">
                    <div className="flex justify-center items-center">
                        {children}
                    </div>
                    <div className="text-xl">
                        {title}
                    </div>
                </Link>
            )}
            {!path && (
                <button className="flex items-center gap-2.5" onClick={onClick}>
                    <div className="flex justify-center items-center">
                        {children}
                    </div>
                    <div className="text-xl">
                        {title}
                    </div>
                </button>
            )}
        </>
    )
}

export default ProfileButton;
```

FormInput.tsx

```
import { FC } from "react";
import { IoIosCloseCircleOutline } from "react-icons/io";

interface Props {
    name: string;
    type: "text" | "password";
    value: string;
    onChange: (event: React.ChangeEvent<HTMLInputElement>) => void;
    errorMsg: string;
    placeholder: string;
}
const FormInput: FC<Props> = ({name, type, value, onChange, errorMsg, placeholder}) => {
    return (
        <div>
            <div className="relative">
                <input
                    name={name}
                    type={type}
                    value={value}
                    onChange={onChange}
                    className={`border-b-2 text-2xl bg-transparent px-4 min-w-[550px] py-1.5
${errorMsg ? "text-red-500 border-red-500 placeholder:text-red-500" : "text-light-gray
border-light-gray"}`}
                    placeholder={placeholder}
                />
```

```
                {errorMsg && (
                    <div className="absolute right-4 center-y text-red-500">
                        <IoIosCloseCircleOutline size={30} />
                    </div>
                )}
            </div>
            {errorMsg && (
                <div className="px-4 text-red-500 text-base mt-1.5">
                    {errorMsg}
                </div>
            )}
        </div>
    )
}

export default FormInput;
```

ProfileInput.tsx

```
import { FC } from "react";

interface Props {
    value: string;
    onChange: (e: any) => void;
    name: string;
    disabled?: boolean
}

const ProfileInput: FC<Props> = ({name, value, onChange, disabled = false}) => {
    return (
        <div className="max-w-[335px] w-full">
            <input type="text" value={value} name={name} onChange={onChange}
disabled={disabled} className="bg-transparent border-b-2 border-light-gray text-2xl px-2.5
py-1.5 w-full text-white"/>
        </div>
    )
}

export default ProfileInput;
```

Search.tsx

```
import { ChangeEvent, FC, useState } from "react";
import { VscSearch } from "react-icons/vsc";

interface Props {
    value: string;
    placeholder: string
    onSearch: (value: string) => void;
}

const Search: FC<Props> = ({value, placeholder, onSearch}) => {
    const [searchValue, setSearchValue] = useState(value);

    const handleChange = (e: ChangeEvent<HTMLInputElement>) => {
        setSearchValue(e.target.value);
    }
    const handleSearch = () => {
        onSearch(searchValue);
    }

    return (
        <div className="relative flex items-center flex-1">
            <input type="text" value={searchValue} onChange={handleChange} className="pl-16
text-dark-gray py-4 bg-transparent border-2 border-blue-gray rounded-20 text-2xl w-full"
placeholder={placeholder}/>
```

```
            <div className="absolute center-y left-3 text-dark-gray h-full flex items-center">
                <button onClick={handleSearch}>
                    <VscSearch size={35} />
                </button>
            </div>
        </div>
    )
}

export default Search;
```

AddVideo.tsx

```
import { FC } from "react";
import BlueButton from "./buttons/BlueButton";
const AddVideo: FC = () => {
    return (
        <div className="absolute top-0 left-0 w-full h-full flex justify-center items-center
bg-black/75">
            <div className="p-6 bg-black border-4 border-dark-blue rounded-20 flex flex-col
gap-9 items-center">
                <h2 className="text-white text-3xl font-bold">Форма додавання відео</h2>
                <div className="flex items-center text-xl text-white gap-6 w-full">
                    <label className="flex-1">Назва відео</label>
                    <input type="text" className="px-6 py-1.5 text-black min-w-[450px]
rounded-20" placeholder="Назва відео"/>
                </div>
                <div className="flex items-center text-xl text-white gap-6 w-full">
                    <label className="flex-1">Тип відео</label>
                    <input type="text" className="px-6 py-1.5 text-black min-w-[450px]
rounded-20" placeholder="Тип відео"/>
                </div>
                <div className="flex items-start text-xl text-white gap-6 w-full">
                    <label className="flex-1">Опис відео</label>
                    <textarea className="px-6 py-1.5 text-black min-w-[450px] rounded-20
min-h-[150px]" placeholder="Опис відео"/>
                </div>
                <div className="flex w-full h-[75px] border-2 border-white rounded-20
border-dashed">
                    <label className="flex justify-center items-center w-full cursor-pointer">
                        <span className="text-xl text-white">Додати відео</span>
                        <input type="file" className="hidden" accept="video/*"/>
                    </label>
                </div>
                <BlueButton type="submit">Додати відео</BlueButton>
            </div>
        </div>
    )
}
export default AddVideo
```

Header.tsx

```
import { FC } from "react";
interface Props {
    children: JSX.Element;
}
const Header: FC<Props> = ({children}) => {
    return (
        <>
            <header className="flex items-center justify-between gap-72">
                {children}
            </header>
        </>
    )
}
export default Header;
```

Login.tsx

```
import { FC, useEffect, useState } from "react";
import FormInput from "./inputs/FormInput";
import BlueButton from "./buttons/BlueButton";
import LinkButton from "./buttons/LinkButrton";

interface Props {
    handleCloseLogin: () => void;
    handleOpenRegister: () => void;
}

const Login: FC<Props> = ({handleCloseLogin, handleOpenRegister}) => {
    const [formValues, setFormValues] = useState({
        email: "",
        password: "",
    });
    const [formErrors, setFormErrors] = useState({
        email: "",
        password: ""
    })

    const handleInputChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = event.target;
        setFormValues({ ...formValues, [name]: value });
    };

    const handleClickForm = (event: any) => {
        event.stopPropagation()
    }

    const handleSubmit = async(e: any) => {
        e.preventDefault();
        if(formValues.email === "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                email: "Некоректна пошта",
            }));
        }
        if(formValues.password === "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                password: "Неправильний пароль",
            }));
        }

        handleCloseLogin();

    }

    useEffect(() => {
        if (formValues.email !== "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                email: ""
            }));
        }
        if(formValues.password !== "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                password: ""
            }));
        }
    }, [formValues, setFormErrors])

    return (
        <div className="absolute top-0 left-0 bg-slate-500/75 w-full h-full min-h-screen flex
justify-center items-center z-50" onClick={handleCloseLogin}>
```

```
            <div className="px-7 py-12 bg-black rounded-20 flex flex-col gap-5"
onClick={handleClickForm}>
                <div className="font-arimo font-bold text-white text-4xl text-center">
                    Форма входу
                </div>
                <form className="flex flex-col gap-5" onSubmit={handleSubmit}>
                    <FormInput errorMsg={formErrors.email} name="email" placeholder="Адреса
електроної пошти" type="text" value={formValues.email} key={"email"}
onChange={handleInputChange}/>
                    <FormInput errorMsg={formErrors.password} name="password"
placeholder="Пароль" type="password" value={formValues.password} key={"password"}
onChange={handleInputChange}/>
                    <BlueButton onClick={() => {}} type="submit"
key={"sub_btn"}>Вхід</BlueButton>
                    <LinkButton onClick={handleOpenRegister}
type="button">Реєстрація</LinkButton>
                </form>
            </div>
        </div>
    )
}

export default Login;
```

Register.tsx

```
import { FC, useEffect, useState } from "react";
import FormInput from "./inputs/FormInput";
import BlueButton from "./buttons/BlueButton";
import LinkButton from "./buttons/LinkButrton";

interface Props {
    handleCloseRegister: () => void;
    handleOpenLogin: () => void;
}


const Register: FC<Props> = ({handleCloseRegister, handleOpenLogin}) => {
    const [formValues, setFormValues] = useState({
        first_name: "",
        email: "",
        password: "",
        confirm_password: ""
    });

    const [formErrors, setFormErrors] = useState({
        first_name: "",
        email: "",
        password: "",
        confirm_password: "",
    })

    const handleInputChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = event.target;
        setFormValues({ ...formValues, [name]: value });
    };

    const handleClickForm = (event: any) => {
        event.stopPropagation()
    }
    const handleSubmit = (e: any) => {
        e.preventDefault();
        if(formValues.email === "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                email: "Некоректна пошта",
            }));
        }
```

```
        if(formValues.password === "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                password: "Неправильний пароль",
            }));
        }
        if(formValues.first_name === "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                first_name: "Некоректне ім’я",
            }));
        }
        if(formValues.confirm_password === "" || formValues.confirm_password !==
formValues.password) {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                confirm_password: "Неправильний пароль",
            }));
        }
        console.log("Register")
    }

    useEffect(() => {
        if (formValues.email !== "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                email: ""
            }));
        }

        if(formValues.password !== "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                password: ""
            }));
        }
        if(formValues.first_name !== "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                first_name: ""
            }));
        }
        if(formValues.confirm_password !== "") {
            setFormErrors(prevErrors => ({
                ...prevErrors,
                confirm_password: ""
            }));
        }
    }, [formValues, setFormErrors])

    return (
        <div className="absolute top-0 left-0 bg-slate-500/75 w-full h-full min-h-screen flex
justify-center items-center z-50" onClick={handleCloseRegister}>
            <div className="px-7 py-12 bg-black rounded-20 flex flex-col gap-5"
onClick={handleClickForm}>
                <div className="font-arimo font-bold text-white text-4xl text-center">
                    Форма реєстрації
                </div>
                <form className="flex flex-col gap-5" onSubmit={handleSubmit}>
                    <FormInput errorMsg={formErrors.first_name} name="first_name"
placeholder="Ім’я" type="text" value={formValues.first_name} key={"first_name"}
onChange={handleInputChange}/>
                    <FormInput errorMsg={formErrors.email} name="email" placeholder="Адреса
електроної пошти" type="text" value={formValues.email} key={"email"}
onChange={handleInputChange}/>
                    <FormInput errorMsg={formErrors.password} name="password"
placeholder="Пароль" type="password" value={formValues.password} key={"password"}
onChange={handleInputChange}/>
```

```
                    <FormInput errorMsg={formErrors.confirm_password} name="confirm_password"
placeholder="Повторіть пароль" type="password" value={formValues.confirm_password}
key={"confirm_password"} onChange={handleInputChange}/>
                    <BlueButton onClick={() => {}} type="submit"
key={"sub_btn"}>Зареєструватись</BlueButton>
                    <LinkButton onClick={handleOpenLogin} type="button">Вхід</LinkButton>
                </form>
            </div>
        </div>
    )
}

export default Register;
```

UserNavigation.tsx

```
import { FC } from "react";
import { FaRegCircleUser } from "react-icons/fa6";
import { useNavigate } from "react-router-dom";
import { useAuth } from "../hooks/useAuth";
import { useGetUserQuery } from "../services/user.service";

const UserNavigation: FC = () => {
    const navigate = useNavigate()
    const isAuth = useAuth();

    const userId = localStorage.getItem("userId")
    const {data: userDataResponse} = useGetUserQuery(userId || "", {
        skip: !userId
    })

    const userImg = userDataResponse?.userImageUrl

    const handleClickLogin = () => {
        navigate("/sign-in")
    }

    const handleClickProfile = () => {
        navigate("/my-profile")
    }

    return (
        <div className="flex text-dark-gray gap-5">
            <div>
                <button onClick={!isAuth ? handleClickLogin : handleClickProfile}
className="overflow-hidden rounded-20">
                    {!userImg && (
                        <img src={"user.png"} alt={"user"} className="w-[40px] h-[40px]
object-cover"/>
                    )}
                </button>
            </div>
        </div>
    )
}

export default UserNavigation;
```

Video.tsx

```
import { FC } from "react";
import { Link } from "react-router-dom";
interface Props {
    videoId: string;
    channelPath: string
    title: string;
    img: string;
```

```
    channelTitle: string
    channelImg: string
}

const Video: FC<Props> = ({videoId, channelPath, title, img, channelTitle, channelImg}) => {
    return (
        <div className="max-w-[330px]">
            <div className="max-h-[196px] max-w-[330px] w-full relative h-full rounded-20
overflow-hidden flex justify-center bg-dark-gray/35">
                <Link to={`/video/${videoId}`}>
                    <img src={img} alt={img} className="object-cover h-full w-auto"/>
                </Link>
            </div>
            <div className="flex gap-2.5 mt-2.5">
                <div className="relative w-12 h-12 rounded-full overflow-hidden">
                    <Link to={channelPath}>
                        <img src={channelImg} alt={channelImg} className="h-full w-full
object-cover"/>
                    </Link>
                </div>
                <div className="text-light-gray text-2xl">
                    <div>
                        <Link to={`/video/${videoId}`}>{title}</Link>
                    </div>
                    <Link to={channelPath} className="font-bold">{channelTitle}</Link>
                </div>
            </div>
        </div>
    )
}

export default Video;
```

userAuth.tsx

```
import { useAppSelector } from "../store/hooks"

export const useAuth = (): boolean => {
    const isAuth = useAppSelector((state) => state.user.isAuth);

    return isAuth;
}
```

Layout.tsx

```
import { Outlet } from "react-router-dom"
import SideBar from "../components/SideBar"

const Layout = () => {
    return (
        <div className="min-h-screen bg-black w-full">
            <SideBar />
            <div className="flex">
                <div className="min-w-[215px] h-full"></div>
                <div className="px-9 flex-1">
                    <Outlet />
                </div>
            </div>
        </div>
    )
}

export default Layout
```

Login.tsx

```
import { FC, useState } from "react";
import FormInput from "../components/inputs/FormInput";
import BlueButton from "../components/buttons/BlueButton";
import { useNavigate } from "react-router-dom";
import LinkButton from "../components/buttons/LinkButrton";
import { AuthService } from "../services/auth.service";
import { toast } from "react-toastify";
import { setTokenToLocalStorage } from "../helpers/localstorage.helper";
import { useAppDispatch } from "../store/hooks";
import { login } from "../store/user/userSlice";

const Login: FC = () => {
    const [formValues, setFormValues] = useState({
        email: "",
        password: "",
    });

    const navigate = useNavigate()
    const dispatch = useAppDispatch();

    const handleLoginSubmit = async(e: React.FormEvent<HTMLFormElement>) => {
        try {
            e.preventDefault();
            const data = await AuthService.login({
                email: formValues.email,
                password: formValues.password
            })

            if(data) {
                setTokenToLocalStorage("accessToken", data.accessToken);
                localStorage.setItem("userId", data.userId);
                dispatch(login(data))
                toast.success("You have been logged in!");
                navigate("/")
            }
        } catch (err: any) {
            const error = err.response?.data.message;
            toast.error(error.toString());
        }
    }

    const handleInputChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = event.target;
        setFormValues({ ...formValues, [name]: value });
    };

    const handleRegistration = () => {
        navigate("/sign-up")
    }

    const handleHome = () => {
        navigate("/");
    }

    return (
        <div className="absolute top-0 left-0 bg-black w-full h-full min-h-screen z-50">
            <div className="flex py-4">
                <BlueButton onClick={handleHome} type="button" styles="rounded-r-[20px]">На
головну</BlueButton>
            </div>
            <div className="absolute top-[50%] left-[50%] translate-x-[-50%]
translate-y-[-50%]">
                <div className="px-7 py-12 bg-black rounded-20 flex flex-col gap-5">
                    <div className="font-arimo font-bold text-white text-4xl text-center">
                        Форма входу
                    </div>
                    <form className="flex flex-col gap-9" onSubmit={handleLoginSubmit}>
```

```
                        <FormInput errorMsg={""} name="email" placeholder="Адреса електроної
пошти" type="text" value={formValues.email} key={"email"} onChange={handleInputChange}/>
                        <FormInput errorMsg={""} name="password" placeholder="Пароль"
type="password" value={formValues.password} key={"password"} onChange={handleInputChange}/>
                        <BlueButton type="submit" key={"sub_btn"}>Вхід</BlueButton>
                        <LinkButton onClick={handleRegistration}
type="button">Реєстрація</LinkButton>
                    </form>
                </div>
            </div>
        </div>
    )
}

export default Login;
```

MyProfile.tsx

```
import { FC, useEffect, useState } from "react";
import { useGetUserQuery } from "../services/user.service";
import Header from "../components/Header";
import UserNavigation from "../components/UserNavigation";
import ProfileInput from "../components/inputs/ProfileInput";
import BlueButton from "../components/buttons/BlueButton";
import { FaPencil, FaRegCircleUser } from "react-icons/fa6";
import { FiSave } from "react-icons/fi";
import { IoIosAddCircleOutline } from "react-icons/io";
import Video from "../components/Video";
import AddVideo from "../components/AddVideo";

const MyProfile: FC = () => {
    const userId = localStorage.getItem("userId")
    const {data: userDataResponse} = useGetUserQuery(userId || "", {
        skip: !userId
    })

    const [userData, setUserData] = useState({
        username: userDataResponse?.userName || "",
        email: userDataResponse?.userEmail || "",
    })
    const [userImg, setUserImg] = useState(userDataResponse?.userImageUrl)
    const [inputsState, setInputsState] = useState(true)
    const [showVideoModal, setShowVideoModal] = useState(false);

    const handleInputChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = event.target;
        setUserData({ ...userData, [name]: value });
    };

    const handleEditUserData = () => {
        setInputsState(state => !state);
    }

    const handleSubmitEdit = () => {
        setInputsState(true);
    }

    useEffect(() => {
        if(userDataResponse) {
            setUserData({
                username: userDataResponse.userName,
                email: userDataResponse.userEmail
            })
            setUserImg(userDataResponse.userImageUrl)
        }
    }, [userDataResponse])
```

```
    return (
        <div className="flex flex-col gap-7 z-0 py-4 flex-1">
            <Header>
                <>
                    <div>
                        <img src="logo.png" alt="logo" />
                    </div>
                    <UserNavigation />
                </>
            </Header>
            <div className="px-12 flex flex-col gap-12">
                <div className="flex gap-12 items-center relative">
                    <div className="relative rounded-full overflow-hidden w-56 h-56
bg-white/35 group">
                        {!userDataResponse?.userImageUrl && (
                            <img src={userImg || "user.png"} alt={userImg || ""}
className="w-full h-full object-cover"/>
                        )}
                        {/* {!userDataResponse?.userImageUrl && (
                            <div className="text-white flex justify-center items-center">
                                <FaRegCircleUser size={224}/>
                            </div>
                        )} */}
                        {!inputsState && (
                            <div className="absolute top-0 left-0 w-full h-full bg-black/75
justify-center items-center hidden group-hover:flex">
                                <label className="cursor-pointer">
                                    <span className="text-white text-xl
text-medium">Редагувати фото</span>
                                    <input type="file" className="hidden"/>
                                </label>
                            </div>
                        )}
                    </div>
                    <div className="flex-1 flex flex-col gap-7">
                        <div>
                            <form className="flex flex-col gap-7 items-start">
                                <ProfileInput name="username" onChange={handleInputChange}
value={userData.username} disabled={inputsState} />
                                <ProfileInput name="email" onChange={handleInputChange}
value={userData.email} disabled={inputsState} />
                                {inputsState && (
                                    <BlueButton type="button" onClick={handleEditUserData}
styles="px-4 rounded-20">
                                        <div className="flex gap-5 items-center">
                                            <FaPencil size={32}/>
                                            <div>Редагувати</div>
                                        </div>
                                    </BlueButton>
                                )}
                                {!inputsState && (
                                    <BlueButton type="button" onClick={handleSubmitEdit}
styles="px-4 rounded-20">
                                        <div className="flex gap-5 items-center">
                                            <FiSave size={32}/>
                                            <div>Зберегти</div>
                                        </div>
                                    </BlueButton>
                                )}
                            </form>
                        </div>
                        <div className="flex justify-start">
                            <BlueButton type="button" styles="px-4 rounded-20">
                                <div className="flex gap-5 items-center">
                                    <IoIosAddCircleOutline size={32}/>
                                    <div>Завантажити нове відео</div>
                                </div>
                            </BlueButton>
                        </div>
```

```
                    </div>
                </div>
                <div className="flex flex-col gap-12">
                    <h4 className="text-3xl text-white font-bold">Завантажені відео</h4>
                    <div className="flex flex-wrap gap gap-x-16 gap-y-16">
                        {userDataResponse?.videos && userDataResponse.videos.length !== 0 &&
userDataResponse.videos.map(video => (
                            <Video channelImg="channel_1_img.png" channelPath="/profile"
channelTitle={video.userName} img={video.videoPreviewUrl} videoId={video.videoId}
title={video.videoTitle} key={video.videoId}/>
                        ))}
                        {userDataResponse?.videos && userDataResponse.videos.length === 0 && (
                            <div className="flex justify-center w-full">
                                <p className="text-3xl text-white">Немає завантажених
відео!</p>
                            </div>
                        )}
                    </div>
                </div>
            </div>
            {showVideoModal && (
                <AddVideo />
            )}
        </div>
    )
}

export default MyProfile;
```

Register.tsx

```
import React, { FC, useState } from "react";
import FormInput from "../components/inputs/FormInput";
import BlueButton from "../components/buttons/BlueButton";
import LinkButton from "../components/buttons/LinkButrton";
import { useNavigate } from "react-router-dom";
import { AuthService } from "../services/auth.service";
import { toast } from "react-toastify";

const Register: FC = () => {
    const navigate = useNavigate();
    const [formValues, setFormValues] = useState({
        name: "",
        email: "",
        password: "",
        confirmPassword: ""
    });

    const hanldeRegisterSubmit = async(e: React.FormEvent<HTMLFormElement>) => {
        try {
            e.preventDefault();
            const data = await AuthService.registration({
                name: formValues.name,
                email: formValues.email,
                password: formValues.password,
                confirmPassword: formValues.confirmPassword
            })

            if(data) {
                toast.success(data);
                navigate("/sign-in")
            }
        } catch (err: any) {
            const error = err.response?.data.message;
            toast.error(error.toString());
        }
    }
```

```
    const handleInputChange = (event: React.ChangeEvent<HTMLInputElement>) => {
        const { name, value } = event.target;
        setFormValues({ ...formValues, [name]: value });
    };

    const handleLogin = () => {
        navigate("/sign-in");
    }

    const handleHome = () => {
        navigate("/")
    }

    return (
        <div className="absolute top-0 left-0 bg-black w-full h-full min-h-screen z-50">
            <div className="flex py-4">
                <BlueButton onClick={handleHome} type="button" styles="rounded-r-[20px]">На
головну</BlueButton>
            </div>
            <div className="absolute top-[50%] left-[50%] translate-x-[-50%]
translate-y-[-50%]">
                <div className="px-7 py-12 bg-black rounded-20 flex flex-col gap-5">
                    <div className="font-arimo font-bold text-white text-4xl text-center">
                        Форма реєстрації
                    </div>
                    <form className="flex flex-col gap-9" onSubmit={hanldeRegisterSubmit}>
                        <FormInput errorMsg={""} name="name" placeholder="Ім’я" type="text"
value={formValues.name} key={"first_name"} onChange={handleInputChange}/>
                        <FormInput errorMsg={""} name="email" placeholder="Адреса електроної
пошти" type="text" value={formValues.email} key={"email"} onChange={handleInputChange}/>
                        <FormInput errorMsg={""} name="password" placeholder="Пароль"
type="password" value={formValues.password} key={"password"} onChange={handleInputChange}/>
                        <FormInput errorMsg={""} name="confirmPassword" placeholder="Повторіть
пароль" type="password" value={formValues.confirmPassword} key={"confirm_password"}
onChange={handleInputChange}/>
                        <BlueButton type="submit">Зареєструватись</BlueButton>
                        <LinkButton onClick={handleLogin} type="button">Вхід</LinkButton>
                    </form>
                </div>
            </div>
        </div>
    )
}

export default Register;
```

VideoDetails.tsx

```
import { FC } from "react";
import Header from "../components/Header";
import UserNavigation from "../components/UserNavigation";
import { Link, useParams } from "react-router-dom";
import LikedButton from "../components/buttons/LikedButton";
import BlueButton from "../components/buttons/BlueButton";
import { IoIosNotificationsOutline } from "react-icons/io";
import { CiLink } from "react-icons/ci";
import ReactPlayer from "react-player";
import { useGetVideoQuery, useToggleDislikeMutation, useToggleLikeMutation } from
"../services/videos.service";
import { toast } from "react-toastify";
import { BiDislike, BiLike, BiSolidDislike, BiSolidLike } from "react-icons/bi";

const VideoDetails: FC = () => {
    const {videoId} = useParams();
    const userId = localStorage.getItem("userId");
    const {data: videoDetails} = useGetVideoQuery(videoId || "", {
        skip: !videoId,
        pollingInterval: 60000
    });
```

```
    const [toggleLike, {}] = useToggleLikeMutation()
    const [toggleDislike, {}] = useToggleDislikeMutation()

    const handleToggleLike = async() => {
        if(videoId) {
            await toggleLike(videoId)
        }
    }

    const handleToggleDislike = async() => {
        if(videoId) {
            await toggleDislike(videoId)
        }
    }

    const handleShare = async () => {
        const shareableLink = window.location.href;
        try {
            await navigator.clipboard.writeText(shareableLink);
            toast("Link copied to clipboard!")
        } catch (error) {
            toast.error("Failed to copy link.");
        }
    };

    return (
        <div className="flex flex-col gap-12 z-0 py-4">
            <Header>
                <>
                    <div>
                        <img src="logo.png" alt="logo" />
                    </div>
                    <UserNavigation/>
                </>
            </Header>
            <div>
                <div className="flex items-end gap-12">
                    <div>
                        <ReactPlayer controls url={videoDetails?.videoUrl} width={800}
height={500} />
                    </div>
                    <div className="flex flex-col gap-12">
                        <Link to={"/profile"} className="flex items-center gap-2.5">
                            <div className="w-20 h-20 rounded-full flex justify-center
items-center overflow-hidden">
                                <img src="channel_1_img.png" alt="channel_1_img.png" />
                            </div>
                            <div className="text-3xl text-white font-bold">
                                {videoDetails?.authorName}
                            </div>
                        </Link>
                        <div className="flex justify-start items-center gap-3">
                            <LikedButton count={videoDetails?.likes || 0}
onClick={handleToggleLike}>
                                <>
                                    {!videoDetails?.isLiked && (
                                         <BiLike size={32}/>
                                    )}
                                    {videoDetails?.isLiked && (
                                        <BiSolidLike size={32} />
                                    )}
                                </>
                            </LikedButton>
                            <LikedButton count={videoDetails?.dislikes || 0}
onClick={handleToggleDislike}>
                                <>
                                    {videoDetails?.isDisliked && (
                                         <BiSolidDislike size={32}/>
                                    )}
```

```
                                    {!videoDetails?.isDisliked && (
                                        <BiDislike size={32} />
                                    )}
                                </>
                            </LikedButton>
                            <div className="flex flex-col items-start gap-3">
                                {userId !== videoDetails?.authorId && (
                                    <BlueButton type="button" styles="px-4 rounded-20">
                                        <div className="flex gap-5 items-center">
                                            <IoIosNotificationsOutline size={32}/>
                                            <div>Підписатися</div>
                                        </div>
                                    </BlueButton>
                                )}
                                <BlueButton type="button" styles="px-4 rounded-20"
onClick={handleShare}>
                                    <div className="flex gap-5 items-center">
                                        <CiLink size={32}/>
                                        <div>Поширити</div>
                                    </div>
                                </BlueButton>
                            </div>
                        </div>
                    </div>
                </div>
                <div>
                    <div className="w-full border-b-2 border-light-gray py-6">
                        <h2 className="text-3xl text-white
font-bold">{videoDetails?.title}</h2>
                    </div>
                    <div className="py-2.5">
                        <p className="text-white text-2xl px-12">
                            {videoDetails?.description}
                        </p>
                    </div>
                </div>
            </div>
        </div>
    )
}

export default VideoDetails;
```

router.tsx

```
import { createBrowserRouter } from "react-router-dom";
import Layout from "../pages/Layout";
import Home from "../pages/Home";
import Profile from "../pages/Profile";
import Watched from "../pages/Watched";
import Login from "../pages/Login";
import Register from "../pages/Register";
import Liked from "../pages/Liked";
import VideoDetails from "../pages/VideoDetails";
import MyProfile from "../pages/MyProfile";

export const router = createBrowserRouter([
    {
        path: "/",
        element: <Layout />,
        children: [
            {
                index: true,
                element: <Home />
            },
            {
                path: 'profile/:userId',
                element: <Profile />
            },
```

```
            {
                path: 'history',
                element: <Watched />
            },
            {
                path: 'liked',
                element: <Liked />
            },
            {
                path: 'sign-in',
                element: <Login />
            },
            {
                path: 'sign-up',
                element: <Register />
            },
            {
                path: 'video/:videoId',
                element: <VideoDetails />
            },
            {
                path: 'my-profile',
                element: <MyProfile />
            }
        ]
    }
])
```

auth.service.ts

```
import { instance } from "../api/axios.api";
import { ILogin, ILoginResponse, IRegistration } from "../types";

export const AuthService = {
    async registration(userData: IRegistration): Promise<string | undefined> {
        const { data } = await instance.post<string>("auth/sign-up", userData);

        return data
    },
    async login(userData: ILogin): Promise<ILoginResponse | undefined> {
        const { data } = await instance.post<ILoginResponse>("auth/sign-in", userData);

        return data
    }
}
```

user.service.ts

```
import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";
import { getTokenFromLocalStorage } from "../helpers/localstorage.helper";
import { IUserData } from "../types";

export const userApi = createApi({
    reducerPath: "userInfo",
    baseQuery: fetchBaseQuery({
        baseUrl: import.meta.env.VITE_BASE_URL,
        prepareHeaders: (headers) => {
            const token = getTokenFromLocalStorage();
            if (token) {
                headers.set('Authorization', `Bearer ${token}`);
            }
            return headers;
        }
    }),
    tagTypes: ['User'],
    endpoints: build => ({
        getUser: build.query<IUserData, string>({
```

```
            query: (id: string) => ({
                url: `/users/${id}`
            }),
            providesTags: ['User'],
        }),
    })
})

export const {useGetUserQuery} = userApi
```

videos.service.ts

```
import { createApi, fetchBaseQuery } from "@reduxjs/toolkit/query/react";
import { getTokenFromLocalStorage } from "../helpers/localstorage.helper";
import { IVideoDetails } from "../types";

export const videosApi = createApi({
    reducerPath: "videosApi",
    baseQuery: fetchBaseQuery({
        baseUrl: import.meta.env.VITE_BASE_URL,
        prepareHeaders: (headers) => {
            const token = getTokenFromLocalStorage();
            if (token) {
                headers.set('Authorization', `Bearer ${token}`);
            }
            return headers;
        }
    }),
    tagTypes: ['Video'],
    endpoints: build => ({
        getVideo: build.query<IVideoDetails, string>({
            query: (id: string) => ({
                url: `/videos/${id}`
            }),
            providesTags: ['Video'],
        }),
        toggleLike: build.mutation({
            query: (videoId: string) => ({
                url: `/videos/${videoId}/toggle-like`,
                method: "POST"
            }),
            invalidatesTags: ['Video'],
        }),
        toggleDislike: build.mutation({
            query: (videoId: string) => ({
                url: `/videos/${videoId}/toggle-dislike`,
                method: "POST"
            }),
            invalidatesTags: ['Video'],
        }),
    })
})

export const {useGetVideoQuery, useToggleLikeMutation, useToggleDislikeMutation} = videosApi
```

userSlice.ts

```
import { createSlice } from '@reduxjs/toolkit'
import type { PayloadAction } from '@reduxjs/toolkit'
import type { RootState } from '../store'
import { ILoginResponse } from '../../types'

// Define a type for the slice state
interface UserState {
  user: ILoginResponse | null
  isAuth: boolean
}
```

```
// Define the initial state using that type
const initialState: UserState = {
  user: null,
  isAuth: false
}

export const userSlice = createSlice({
  name: 'user',
  // `createSlice` will infer the state type from the `initialState` argument
  initialState,
  reducers: {
    login: (state, action:PayloadAction<ILoginResponse>) => {
        state.user = action.payload
        state.isAuth = true
    },
    logout: (state) => {
        state.user = null,
        state.isAuth = false
    }
  },
})

export const { login, logout } = userSlice.actions

// Other code such as selectors can use the imported `RootState` type
export const selectUser = (state: RootState) => state.user

export default userSlice.reducer
```

store.ts

```
import { configureStore } from '@reduxjs/toolkit'
import userReducer from "./user/userSlice"
import { userApi } from '../services/user.service'
import { videosApi } from '../services/videos.service'

export const store = configureStore({
  reducer: {
      user: userReducer,
      [userApi.reducerPath]: userApi.reducer,
      [videosApi.reducerPath]: videosApi.reducer
  },
  middleware: getDefaultMiddleware => getDefaultMiddleware().concat(userApi.middleware,
videosApi.middleware)
})


// Infer the `RootState` and `AppDispatch` types from the store itself
export type RootState = ReturnType<typeof store.getState>
// Inferred type: {posts: PostsState, comments: CommentsState, users: UsersState}
export type AppDispatch = typeof store.dispatch
```

types/index.ts

```
export interface IRegistration {
    name: string,
    email: string,
    password: string,
    confirmPassword: string
}

export interface ILogin {
    email: string;
    password: string;
}
```

```
export interface ILoginResponse {
    userId: string;
    accessToken: string
}

export interface IVideo {
    userId: string;
    userName: string;
    videoId: string;
    videoPreviewUrl: string;
    videoTitle: string;
}

export interface IUserData {
    userEmail: string;
    userId: string;
    userImageUrl: string | null;
    userName: string;
    videos: IVideo[]
}

export interface IVideoDetails {
    id: string,
    title: string,
    description: string,
    authorId: string,
    authorName: string,
    authorImageUrl: null | string,
    likes: number,
    dislikes: number,
    videoUrl: string
    isLiked: boolean,
    isDisliked: boolean
}
```

App.tsx

```
import { RouterProvider } from "react-router-dom"
import { router } from "./router/router"
import "./index.css"
import 'react-toastify/dist/ReactToastify.css';
import { useDispatch } from "react-redux";
import { getTokenFromLocalStorage } from "./helpers/localstorage.helper";
import { useGetUserQuery } from "./services/user.service";
import { useEffect } from "react";
import { login, logout } from "./store/user/userSlice";

function App() {
  const dispatch = useDispatch();
  const token = getTokenFromLocalStorage();
  const id = localStorage.getItem("userId");

  const { data: userData } = useGetUserQuery(id || "", {
    skip: !token || !id,
  });

  useEffect(() => {
    if (token) {
      dispatch(login({
        userId: id || "",
        accessToken: token
      }));
    } else {
      dispatch(logout());
    }
  }, [userData, dispatch, token, id]);
```

```
  return (
    <RouterProvider router={router} />
  )
}

export default App
```

main.tsx

```
import React from 'react'
import ReactDOM from 'react-dom/client'
import App from './App.tsx'
import { ToastContainer } from 'react-toastify'
import { Provider } from 'react-redux'
import { store } from './store/store.ts'

ReactDOM.createRoot(document.getElementById('root')!).render(
  <React.StrictMode>
    <Provider store={store}>
      <App />
      <ToastContainer position='top-right' autoClose={7000}/>
    </Provider>
  </React.StrictMode>,
)
```