ДИПЛОМНА РОБОТА
БАКАЛАВРА

ДРБ. ІП - 24.00.00.000 ПЗ

Група ІП-20-2

Омельченко Анатолій

2024

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

УДК **004.4**

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: “ Реалізація програмного забезпечення платформи завантаження та перегляду відео файлів ”

(назва згідно з наказом ректора)

спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІП - 24.00.00.000 ПЗ

(позначення)

Консультанти:

з питань нормоконтролю

асист. **/М.М. Піх/**
(посада) (підпис) (дата)(розшифровка підпису)

Студент

ІП-20-2 **/А.А.Омельченко/**
(шифр групи) (підпис) (дата) (розшифровка підпису)

Керівник роботи

д.т.н., проф. **/В.М.Юрчишин/**
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту
Завідувач кафедри

к.т.н., доц. **/В.В.Бандура/**
(посада) (підпис) (дата) (розшифровка підпису)

2024
(рік)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2024 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Омельченку Анатолію Андрійовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) “ Реалізація програмного забезпечення платформи завантаження та перегляду відео файлів”

керівник проекту (роботи) Юрчишин Володимир Миколайович, проф., д.т.н.

затверджені наказом закладу вищої освіти від **“** ____ ” травня 2024 р. № _____

2. Строк подання студентом проекту (роботи) 17 червня 2024 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

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

1. Аналіз інформаційних технологій для побудови платформи перегляду відео файлів
2. Проектування структури і бази даних платформи завантаження та перегляду відео файлів
3. Моделювання архітектури проекту
4. Програмна реалізація платформи завантаження та перегляду відео файлів
5. Розробка інтерфейсу платформи

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Зростання ринку стрімінгових послуг 2021 – 2030 роки (рис. 1.1)
2. Платформа YouTube (рис. 1.2)
3. Головна сторінка Vimeo (рис. 1.3)
4. Платформа Netflix (рис. 1.4)
5. Головна сторінка сервісу SweetTV (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **22 травня 2024 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз інформаційних технологій для побудови платформи перегляду відео файлів	26.05.2024	виконано
2	Проектування структури і бази даних платформи завантаження та перегляду відео файлів	29.05.2024	виконано
3	Моделювання архітектури проекту	30.05.2024	виконано
4	Програмна реалізація платформи завантаження та перегляду відео файлів	04.06.2024	виконано
5	Розробка інтерфейсу платформи	10.06.2024	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	17.06.2024	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Дана бакалаврська робота присвячена розробці програмного забезпечення для платформи завантаження та перегляду відео файлів. Основною метою є створення зручного та ефективного інструменту, який дозволить користувачам завантажувати відео файли на платформу, організовувати їх у зручну структуру, а також легко інтегрувати можливості перегляду та керування відтворенням.

У роботі розглядаються такі аспекти, як проектування інтерфейсу користувача, архітектура програмного забезпечення, розробка функціональних можливостей для завантаження, організації та керування віде офайлами, а також реалізація інструментів для зручного перегляду контенту.

КЛЮЧОВІ СЛОВА: ВІДЕОХОСТИНГ, ВЕБ-ДОДАТОК, СТРИМІНГ, МОДЕЛЮВАННЯ СТРУКТУРИ ПРОЕКТУ, ПРОЕКТУВАННЯ МОДЕЛІ БАЗИ ДАНИХ, ВІДЕО КОНТЕНТ

ANNOTATION

This bachelor's thesis is devoted to the development of software for a platform for downloading and viewing video files. The main goal is to create a convenient and effective tool that will allow users to upload video files to the platform, organize them in a convenient structure, and also easily integrate viewing and playback control capabilities.

The work considers such aspects as the design of the user interface, software architecture, the development of functionality for downloading, organizing and managing video files, as well as the implementation of tools for convenient viewing of content.

KEYWORDS: VIDEO HOSTING, WEB APPLICATION, STREAMING, PROJECT STRUCTURE MODELING, DATABASE MODEL DESIGN, VIDEO CONTENT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9

1. АНАЛІЗ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ПОБУДОВИ ПЛАТФОРМИ ПЕРЕГЛЯДУ ВІДЕО ФАЙЛІВ 11
1.1. Опис предметної області 11
1.2. Основні функції та переваги стрімінгових платформ 13
1.3. Аналіз існуючих платформ перегляду, збереження та розповсюдження відео файлів 15
1.3.1. Опис платформи YouTube 15
1.3.2 Веб-платформа Vimeo 18
1.3.3. Опис платформи Netflix 20
1.3.4. Опис платформи SweetTV 22
1.3.5 Веб-застосунок Tik Tok 23
1.4. Характеристика потенційної аудиторії клієнтів розроблюваної платформи 25

2. ПРОЕКТУВАННЯ СТРУКТУРИ І БАЗИ ДАНИХ ПЛАТФОРМИ ЗАВАНТАЖЕННЯ ТА ПЕРЕГЛЯДУ ВІДЕО ФАЙЛІВ 29
2.1. Опис процесу розробки платформи завантаження та перегляду відео файлів 29
2.2. Розробка алгоритму функціонування платформи 30
2.3. Алгоритм додавання нового відео файлу до платформи 32
2.4. Розробка структури бази даних 35
2.5. Моделювання архітектури проекту 37

2.6. Реалізація класів та моделей для роботи з базою даних 38
2.7. Архітектура веб-платформи 43
2.8. Основні аспекти щодо призначення розроблюваної платформи 48
2.9. Буферизація та потоки в JS. JWT-токен..... 50
2.10. Опис мови Javascript та фреймворків для розробки..... 54
2.10.1. Express.js..... 57
2.10.2. Vue.js 58

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПЛАТФОРМИ ЗАВАНТАЖЕННЯ ТА ПЕРЕГЛЯДУ ВІДЕО ФАЙЛІВ 61
3.1. Розробка функціональних вимог до платформи 61
3.2. Розробка структури back-end частини платформи..... 63
3.3. Розробка структури front-end частини платформи..... 65
3.4. Розробка інтерфейсу платформи перегляду відео файлів 68

ВИСНОВКИ 73
ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ..... 74
ДОДАТКИ
БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

BSD – Berkeley Software Distribution;

FK – Foreign key, Зовнішній ключ;

2FA - Dual Factor Authentication

TLS - Transport Layer Security

ВСТУП

Відеохостинг є сайтом, який дозволяє завантажувати, конвертувати, зберігати, відтворювати та переглядати відео в браузері, за допомогою спеціального програвача. Більшість подібних сервісів не надає власного відео, але дотримується принципу створення вмісту користувачем. Зазвичай відео вміст завантажується через веб-сайт хостинг-сервісу, мобільний або настільний застосунок або API.

Відеохостинг може зберігати будь-який тип відео вмісту, від коротких роликів до повнометражних фільмів та телевізійних шоу. Сервіс зберігає відео на своєму сервері та надає користувачам можливість вбудовувати різні типи кодів або посилань для перегляду відео іншими користувачами.

Веб-сайт, який переважно використовується для хостингу відео, зазвичай називають веб-сайтом для обміну відео.

Розробники мають створити функціональний та повноцінний продукт, який має зрозумілий інтерфейс, зручний пошук та завантаження нових відео, а також роботу з інформацією на сервері.

Найбільшою перевагою веб-додатку є можливість запуску на різних платформах - відео можна переглядати на будь-якому пристрої з доступом до Інтернету, незалежно від операційної системи.

Історія відеохостингів почалася досить давно, перші з них з’явилися ще у 2004 році, коли навіть Інтернет не був доступним для усіх користувачів. Функції таких сервісів були досить обмеженими, вони надавали користувачам можливість завантажувати відео невеликого розміру та середньої якості.

Найбільший розквіт відеохостингів прийшовся на початок 2010-их років, коли більшість контенту вже перейшла з телевізору на екрани телефонів та комп’ютерів: блоги, кліпи зірок, смішні нарізки, відео з

котиками. Сервіси відеохостингу стали справжнім проривом саме в той момент, коли стали популярні і більш корисні, ніж у перші роки.

Не зважаючи на різноманіття існуючих рішень, наразі проблема відеохостингів все ще існує, тому що немає достатньо простих у використанні відеохостингів з низькими потребами у апаратному забезпеченні, що не були б перевантажені не дуже корисними функціями, які ускладнюють взаємодію з основним функціоналом хостингу.

1. АНАЛІЗ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ПОБУДОВИ ПЛАТФОРМИ ПЕРЕГЛЯДУ ВІДЕО ФАЙЛІВ

1.1. Опис предметної області

Стрімінгова платформа (платформа перегляду відео контенту) – це онлайн-джерело розваг, завдяки якому кожен може влаштувати театр, кінотеатр чи подивитися інші віртуальні розваги онлайн, не виходячи з дому.

Ці послуги є альтернативою кабельним і супутниковим послугам, але часто за нижчою ціною та більшим спектром послуг. Замість того, щоб зберігатися на жорсткому диску пристрою, відео або медіа файл надсилається у вигляді стисненого відеоматеріалу через Інтернет для миттєвого відтворення. Замість того, щоб чекати, поки фільм буде завантажено на пристрій, а потім переглядати його, це можна зробити у реальному часі.

Будь-який пристрій, який має підключення до Інтернету та програми, які можуть розпакувати вміст, може використовувати служби відео стрімінгу. Але іноді послуги можуть бути орієнтовані на обмежений тип пристроїв, наприклад тільки для девайсів певного бренду, подібно до Apple TV або Apple Fitness+, які працюють тільки на певних пристроях Apple. Інші додатки можуть підтримуватися рекламою, працювати за безкоштовною моделлю, а також пропонувати придбати контент за окрему плату.

Однією з основних причин, що сприятливо впливає на галузь, є розширення використання Інтернету та зростання продажів мобільних пристроїв. На додаток до цього, спостерігається помітне зростання попиту на послуги потокового відео в коледжах, університетах та установах по всьому світу. Це можна віднести до його переваг, які включають покращення процесів навчання завдяки візуальним вебінарам і записам курсу. Крім того, компанії широко використовують прямі трансляції, оскільки це допомагає їм

просувати свої товари та послуги, розвивати свої бренди та покращувати взаємодію з клієнтами.

Останнім часом спостерігаються певні фактори росту. Попит на послуг стрімінгу медіа файлів за запитом (Video on demand) зростає, що сприяє значному розширенню ринку.

За даними звіту Motion Picture Association за 2020 рік, кількість користувачів онлайн VoD зросла приблизно до 1,11 мільярда під час епідемії COVID-19, а до 2025 року, за прогнозами, досягне 2 мільярдів користувачів.

Обсяг світового ринку потокового відео в 2021 році оцінювався в 375,1 мільярда доларів США, а до 2030 року очікується, що він досягне приблизно 1721,4 мільярда доларів США.



Рисунок 1.1 – Зростання ринку стрімінгових послуг 2021 – 2030 роки

Ключові ринкові можливості наступні:

- Вплив потокового відео в освіті. Відео може допомогти учням запам’ятати матеріал ефективніше, що може мати величезний позитивний вплив на освіту.

До цієї категорії входять: вебінари, відеолекції. Освітні організації, включаючи університети, школи та коледжі, тепер створюють інтерактивну інформацію та поширюють її за допомогою відеопрезентацій. Таким чином

вони ефективно використовують технології для поширення знань. Обов'язки вчителів суттєво змінюються внаслідок зростання попиту на онлайн формат.

Останні тепер можуть викладати за допомогою відео у віртуальних класах по всьому світу та демонструвати добірку навчальних фільмів у класі, що сприяє навчальному процесу.

- Вплив потокового відео у кіноіндустрії. Стрімінгові сервіси надають широкий спектр контенту, не виходячи з дому. Деякі служби безкоштовні, приклад BBC Player, для використання інших платформ вам доведеться платити, будь то підписка, приклад Netflix, чи індивідуальна оренда, як-от SweetTV. Деякі сервіси потокового передавання пропонують суміш безкоштовного перегляду, з можливістю подивитися певний контент — на основі прокату чи купівлі, приклад Amazon Prime. Деякі послуги надійшли від уже існуючих виробничих компаній, прикладом може слугувати Disney+, тоді як інші створили власні виробничі компанії після створення своєї стрімінгової платформи, як приклад - Netflix.

1.2. Основні функції та переваги стрімінгових платформ

Користувачі вимогливі до перегляду улюбленого контенту. Часи, коли глядачі сідали перед телевізором в певний час і день, давно минули. Зараз, коли світ швидко змінюється, мінливе робоче середовище та мобільний спосіб життя, глядачі хочуть мати контроль над тим, як вони дивляться та сприймають контент.

Основні функції які шукають користувачі в стімінгових платформах:

- Зручний інтерфейс. Користувачі хочуть мати легкий доступ до своєї платформи. Зручний інтерфейс повинен бути важливою особливістю кожного додатку. Перед реалізацією додатку варто опрацювати усі візуальні аспекти та провести UX та UI тести, щоб переконатися, що дизайн максимально зручний для користувача;

- Безпека та конфіденційність. Сьогодні доступ до візуального та аудіоконтенту є простим і необмеженим. Але це також сприяє зовнішнім загрозам. Тому важливо дбати про стандарти шифрування, сертифікати та протоколи безпеки;

- Управління відеоконтентом. Система керування вмістом (CMS) допомагає класифікувати весь вміст і створити організоване середовище для всіх фільмів, а також полегшує глядачам навігацію в системі;

- Особливості налаштування. Створення унікального досвіду і надання користувачам контролю над платформою;

- Сумісність. Платформа має бути сумісна з іншими пристроями.

- Батьківський контроль. Завдяки цій функції батьки можуть контролювати контент, доступний дітям;

- Регулярні оновлення контенту.

Більше людей дивляться свої улюблені шоу онлайн, ніж будь-коли раніше. Але багато сервісів потокового передавання все ще відстають, коли йдеться про те, щоб зробити послуги та шоу доступними для людей з обмеженими можливостями.

Функції стімінгових платформ з перспективи веб доступності:

• Наявність якісних субтитрів. Це функція не лише для людей із втратою слуху. Багатьом людям так, таким чином легше стежити за тим, що відбувається на екрані;

• Легка навігація;

• Тригерні попередження;

• Звуковий опису. Він забезпечує паралельну розповідь під час перегляду фільму чи серіалу на Netflix. Користувач можете почути про все, що відбувається на екрані, включаючи середовище, в якому перебувають персонажі, опис міміки та рухів персонажа, костюми, зміни сцени, декорація.;

• Зображення контенту без миготінь;

• Поганий колірний контраст;

• Наявність підпису у блоків;

• Голосові команди. Голосові елементи керування можуть допомогти користувачу шукати, переглядати та керувати медіа файлом, який він переглядаєте (старт, пауза, перемотування тощо). Вони корисні людям, які мають проблеми з пересуванням, і людям із вадами зору. Набагато простіше, не кажучи вже про те, що швидше, сказати те, що ви хочете, ніж шукати потрібні елементи керування.

1.3. Аналіз існуючих платформ перегляду, збереження та розповсюдження відео файлів

1.3.1. Опис платформи YouTube

YouTube - це платформа що належить компанії Google, надає змогу поширювати відео підписникам, та стежити за творчою діяльністю автора. Запустили даний сервіс 14 лютого 2005 року Чад Херлі, Стів Чен та Джавед Карім. Цей ресурс посідає друге місце за кількістю відвідування веб-сайтів відразу після пошукового сервісу Google. На відео сервіс YouTube заходять понад мільярд користувачів щомісяца, які в свою чергу переглядаю дуже велику кількість контенту щодня. У жовтні 2006 року Google купив право власності на YouTube. Після цього сервіс змінив свою бізнес модель, він більше не приносить дохід тільки з реклами, тепер пропонується платний контент наприклад фільми та ексклюзивний контент. YouTube і схвалені автори беруть участь у програмі Google AdSense, яка приносить більше доходу для обох сторін. З тих пір він перетворився з невеликої платформи для потокового відео до великого сервісу із заявленим доходом у 19,8 мільярдів доларів у 2020 році.

З моменту придбання компанією Google YouTube вийшов за межі веб-сайту на мобільні додатки, мережене телебачення та можливість

підключення до інших служб. Категорії відео на сервісі включають музичні відео, відеокліпи, новини, художні фільми, короткометражні фільми, документальні фільми, тизери, аудіозаписи, трейлери фільмів, прямі трансляції, відеоблоги тощо. Більшість контенту створюють окремі особи. Це включає співпрацю між ютуберами та корпоративними спонсорами.

З 2015 року створені медіа-корпорації, такі як Disney, ViacomCBS та WarnerMedia, створили та розширили свої корпоративні канали YouTube, щоб рекламувати ширшу аудиторію.

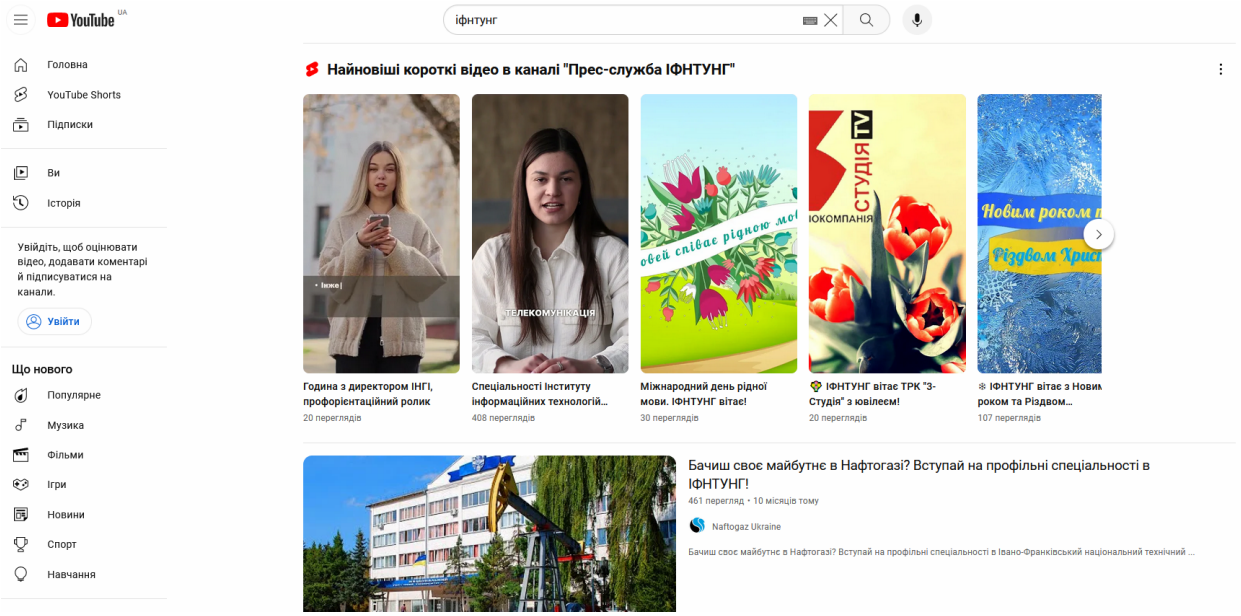


Рисунок 1.2 – Платформа YouTube

YouTube надає безліч можливостей для розповсюдження відеоконтенту. Власники каналів можуть завантажувати відео будь-якої тематики і жанру, що дає їм великий простір для самовираження та творчості.

YouTube є потужним інструментом для популяризації бренду, продукту або послуги. Багато компаній та бізнесів використовують YouTube для створення відеооглядів, рекламних роликів та навчального контенту з метою залучення нових клієнтів та збільшення своєї аудиторії. Завдяки можливості

монетизації, власники каналів можуть отримувати дохід від рекламних показів на своїх відео.

Варто зазначити, що YouTube дозволяє взаємодіяти з аудиторією. Глядачі можуть залишати коментарі, ставити лайки та ділитися відео з іншими.

YouTube, будучи однією з найбільших платформ для обміну відео, безсумнівно, революціонізував спосіб споживання та обміну відео. Однак, як і будь-яка інша платформа, вона не позбавлена недоліків і недоліків.

Одна з проблем - поширеність оманливого вмісту або вмісту, що вводить в оману. Деякі творці контенту вдаються до сенсаційних назв, ескізів і оманливих описів, щоб залучити більше переглядів і зацікавленості. Це підриває довіру до платформи та може призвести до дезінформації або розчарування серед глядачів, які шукають справжній і цінний вміст.

Система монетизації YouTube також зазнала критики. Хоча він пропонує творцям можливість отримувати дохід за допомогою реклами та спонсорства, алгоритмічний характер монетизації може бути непослідовним і несправедливим. Багатьом творцям важко відповідати суворим критеріям монетизації або на їхні прибутки суттєво впливають зміни в політиках чи алгоритмах розміщення реклами.

YouTube збирає величезну кількість даних користувачів для цільової реклами та персоналізації. Це підняло питання про те, як дані користувачів зберігаються, передаються та потенційно доступні третім особам.

Незважаючи на те, що YouTube запровадив функції конфіденційності та керування користувачами, забезпечення повного захисту даних і прозорості залишається постійною проблемою.

Інше занепокоєння пов’язане з алгоритмом рекомендацій платформи. Хоча він спрямований на персоналізацію вмісту та підтримку зацікавленості користувачів, його критикували за просування суперечливого або екстремального вмісту, який може увічнити упередженість, дезінформацію.

1.3.2 Веб-платформа Vimeo

Vimeo - це популярна веб-платформа для спільного використання та перегляду відео. Заснована у 2004 році, ця платформа надає багато функцій та можливостей для завантаження, перегляду та взаємодії з відео (рис 1.3).

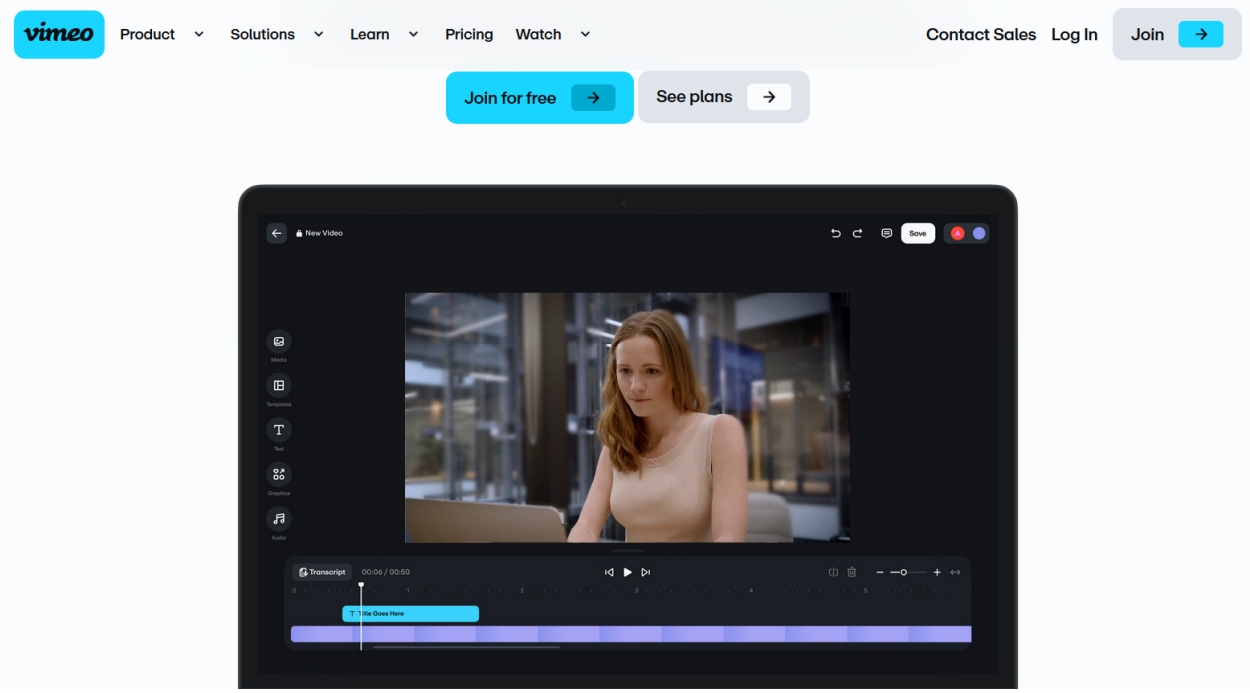


Рисунок 1.3 – Головна сторінка Vimeo

Vimeo є платформою, яка пропонує багато інструментів для творчості. Ці інструменти дозволяють користувачам редагувати свої відео безпосередньо на сайті, додавати підписи, водяні знаки та синхронізувати відео з аудіофайлами. Такі можливості значно полегшують роботу творцям вмісту, оскільки вони можуть створювати професійно виглядаючі відео, не вдаючись до використання зовнішніх програм для редагування.

Проте, слід відзначити, що Vimeo має обмежену аудиторію порівняно з платформами з великими платформами. Це може впливати на розповсюдження вашого вмісту та досягнення широкої аудиторії. Тому, якщ метою є максимальне поширення і залучення аудиторії, варто врахувати це обмеження.

Незважаючи на це, Vimeo є ідеальною платформою для професіоналів та творців вмісту, які цінують якість інструментів. Вона надає багато корисних можливостей для завантаження, перегляду, редагування та спілкування. Завдяки цим інструментам, ви можете покращити якість свого вмісту, виразити свою унікальність та поділитися своїми творчими ідеями зі світом.

Також варто зазначити, що Vimeo ставить на перший план спілкування з аудиторією. Ви можете створювати спеціальні канали, коментувати відео, обмінюватися думками з іншими користувачами та взаємодіяти зі своїми фанатами та прихильниками.

Одним з недоліків Vimeo є його обмежена безкоштовна версія. В порівнянні з іншими платформами для завантаження відео, які пропонують безкоштовні плани з більш широким функціоналом, Vimeo має обмежені можливості для безкоштовних користувачів.

Наприклад, у безкоштовній версії Vimeo можуть бути обмеження щодо кількості відео, які можна завантажити, або обмеження щодо розміру файлу. Це може бути недоцільним для тих, хто потребує безкоштовного простору для завантаження великого обсягу відео.

Ще один з недоліків Vimeo є обмежена аудиторія порівняно зі широко популярними платформами, Vimeo не має такої широкої бази активних користувачів, тому розповсюдження вашого вмісту тут може бути менш ефективним, особливо якщо мета - привернення максимальної кількості переглядів і популярність.

Завдяки меншій аудиторії, вміст може мати меншу кількість вподобань, коментарів та розповсюджень порівняно з платформами, які мають широку базу активних користувачів.

Іншим недоліком є обмежена підтримка форматів відео. Vimeo має свої власні вимоги до формату відео, що може створювати проблеми для тих, хто має велику кількість відео в інших форматах.

Наприклад, якщо відео мають формат, який не підтримується Vimeo, вам доведеться конвертувати їх у підтримуваний формат перед завантаженням. Це може забирати більше часу та зусиль.

Додатковим недоліком Vimeo є обмежені можливості соціального взаємодії та спільноти. Хоча платформа має функцію коментування та взаємодії з іншими користувачами, воно не так широко розповсюджене та активне. Також варто зазначити, що Vimeo має меншу спільноту активних користувачів порівняно з більшими платформами, такими як YouTube.
Це може вплинути на розповсюдження вашого вмісту та взаємодію з іншими користувачами. Якщо вам потрібно широке поширення вмісту та більш активну спільноту, то Vimeo може не задовольнити ваші очікування.

Vimeo спрямований більше на професіональну спільноту та якісний вміст. Це може бути перевагою для деяких творців, які хочуть зосередитися на якості, але може бути обмеженням для тих, хто прагне досягти максимальної популярності та розповсюдження свого вмісту.

1.3.3. Опис платформи Netflix

Netflix, Inc. - американська компанія, що займається продюсуванням та розповсюдженням відео контенту, створена 29 серпня 1997 року, пропонує бібліотеку фільмів і телесеріалів. Також розроблює власну продукцію, відому як Netflix Originals.

Станом на жовтень 2021 року Netflix має понад 214 мільйонів передплатників. Сервіс доступний у всьому світі, за винятком материкового Китаю (через місцеві обмеження), Сирії, Північної Кореї та Криму (через санкції США). Netflix відіграє визначну роль у незалежному кінопрокаті та є членом Асоціації кіно.

Netflix є провайдером стрімінгового відео, який дозволяє користувачам дивитися фільми, серіали та телешоу через Інтернет. Netflix надає доступ до широкого вибору фільмів, серіалів та телешоу, включаючи оригінальний

контент. Воно також використовує алгоритми для рекомендацій персоналізованого контенту.

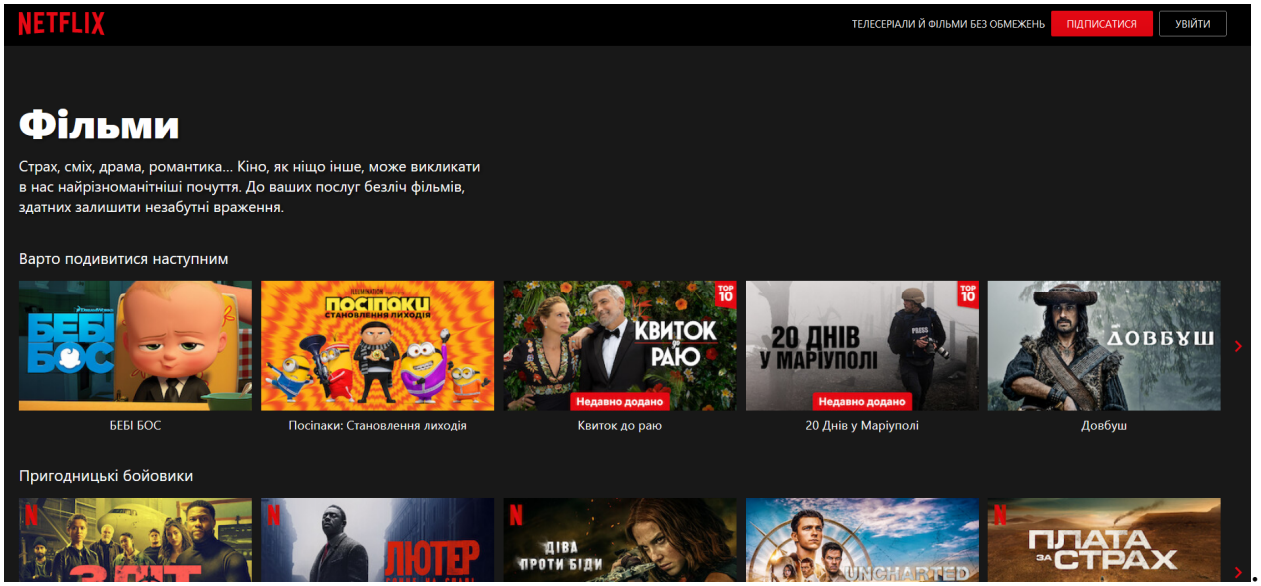


Рисунок 1.4 – Платформа Netflix

Переваги Netflix:

- Наявність власних шоу, фільмів та серіалів;
- Без реклами;
- Широкий асортимент фільмів;
- Наявність Kids mode;
- Сімейна підписка

Недоліки Netflix:

- Не має можливості дивитися ТВ онлайн;
- Відносно малий вибір озвучки та субтитрів для Українського ринку.

Можливості веб доступності на Netflix:

- Аудіоопис;
- Закриті субтитри;
- Голосові команди;
- Швидкість відтворення.

1.3.4. Опис платформи SweetTV

SweetTV – це український провайдер відео на замовлення (VOD), який також надає послуги інтернет-телебачення за технологією OTT. SWEET.TV був заснований у 2017 р. У 2020–2021 рр. були підписані прямі угоди з кінокомпаніями Disney, Paramount, Universal, Sony, MGM та телеканалом CBS, за умовами яких на платформі з'явився іноземний контент з легальним українським дубляжем. Його запис виконується за власні кошти компанії на базі проєкту «Hollywood українською». Крім того, на сервісі представлені фільми від українських кіностудій: Так Треба Продакшн, Film.ua Group тощо.

Пряма комунікація з аудиторією відбувається шляхом інструментів месенджер-маркетингу та розсилки смс-, email-, web push-, push-повідомлень.

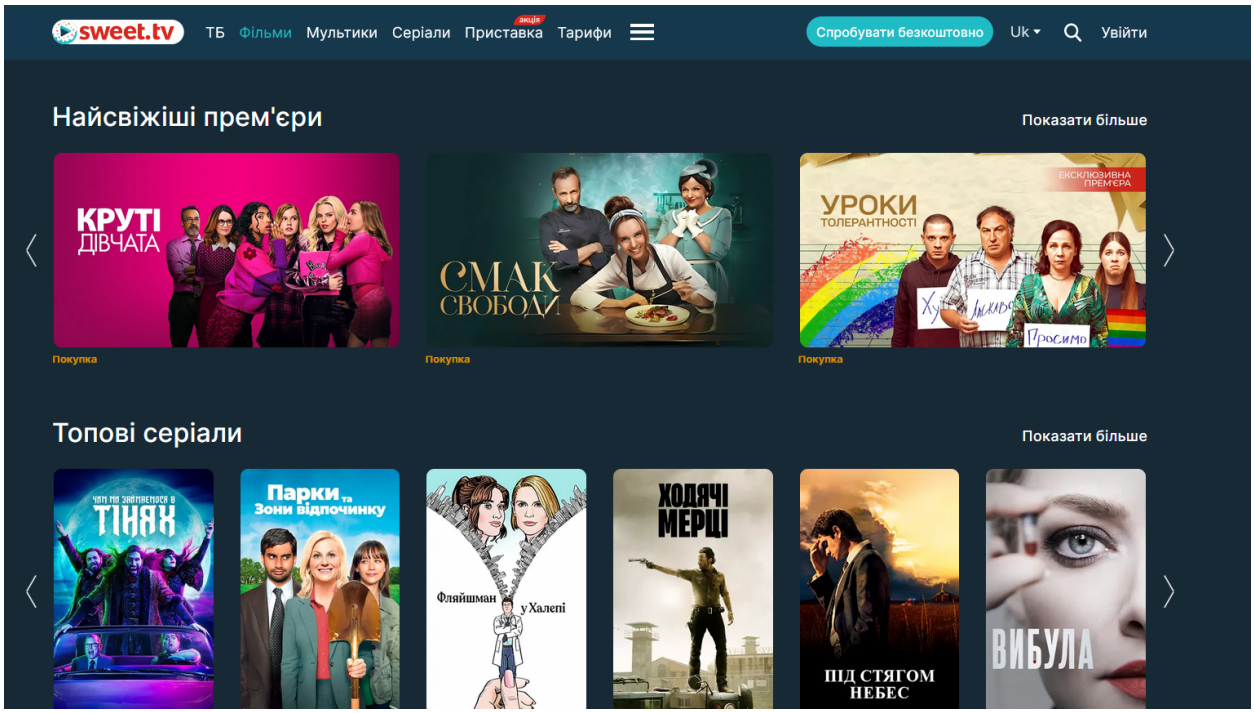


Рисунок 1.5 – Головна сторінка сервісу SweetTV

Переваги SweetTV:

- Можливість дивитися ТВ онлайн
- Без реклами;
- Широкий асортимент фільмів;

- Батьківський контроль

Недоліки SweetTV:

- Відсутність власних шоу, фільмів та серіалів;
- Відсутність сімейної підписки;

Можливості веб доступності на SweetTV:

- Закриті субтитри;
- Швидкість відтворення;
- Онлайн підтримка.

1.3.5 Веб-застосунок Tik Tok

TikTok — популярний веб-застосунок для відео-контенту, який захопив весь світ. TikTok, запущений у 2016 році, дозволяє користувачам створювати та ділитися короткими відеороликами під музику, Він набув величезної популярності, особливо серед молодшої демографії, і став культурним явищем із величезною базою користувачів по всьому світу (рис 1.6).

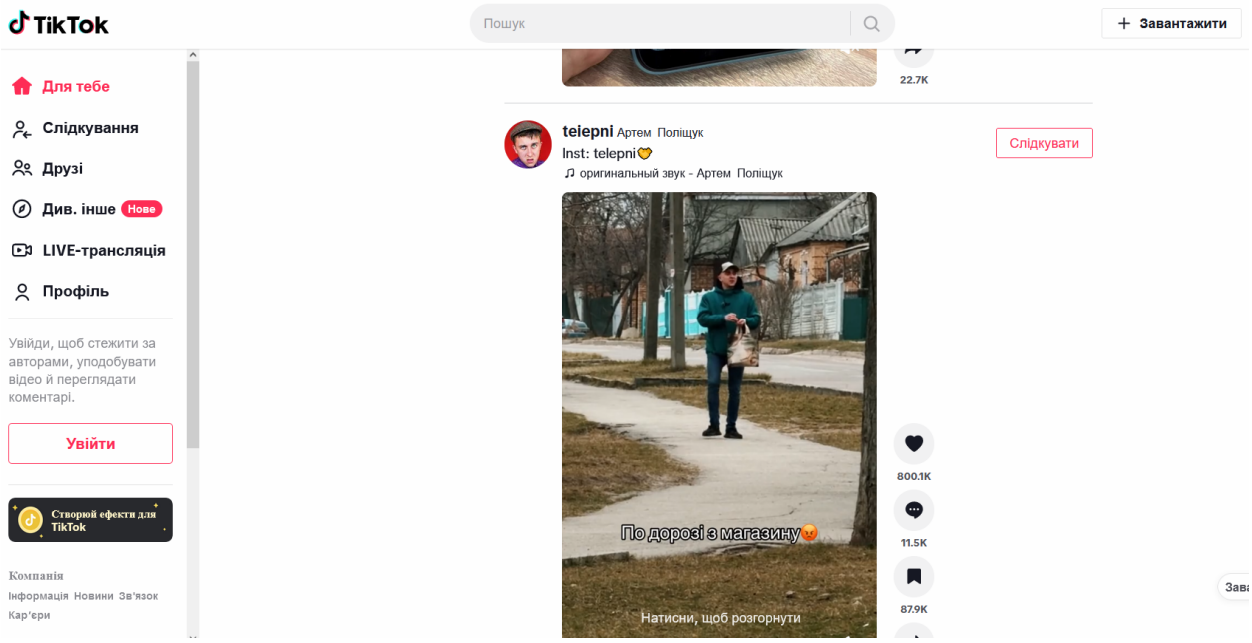


Рисунок 1.6 - Веб-застосунок для відео-контенту Tik Tok

Однією з ключових особливостей TikTok є його простота та доступність. Програма надає зручний інтерфейс, який дозволяє будь-кому легко створювати, редагувати та ділитися відео. Завдяки широкому спектру творчих інструментів, фільтрів, ефектів і музичних параметрів TikTok пропонує користувачам можливість створювати візуально привабливий і захоплюючий вміст. Функції редагування дозволяють користувачам додавати текст, наклейки та переходи, покращуючи загальне враження від відео.

Соціальні функції TikTok є невід'ємною частиною програми. Користувачі можуть стежити за іншими, ставити лайки та коментувати відео, а також брати участь у дуетах і співпраці з іншими творцями.

Ця інтерактивна природа заохочує зв'язок і взаємодію між користувачами, сприяючи почуттю причетності та соціальної залученості.

Одна з ключових проблем, пов'язаних з TikTok, стосується питань конфіденційності та безпеки даних користувачів. У зв'язку з тим, що компанія-власник TikTok, ByteDance, має штаб-квартиру в Китаї, виникали побоювання стосовно збору даних і можливого доступу до них з боку китайських владних органів.

Хоча TikTok заявляє, що дані користувачів, зокрема їхні відео та особиста інформація, зберігаються на серверах, розташованих за межами Китаю, все ж є питання щодо рівня конфіденційності та можливості неконтрольованого доступу до даних.

Це питання привернуло увагу різних країн, і були запроваджені різні регуляторні заходи, які спрямовані на забезпечення захисту даних користувачів на платформі.

- Проблеми контролю вмісту: TikTok був об'єктом скарг на неприйнятний або шкідливий вміст, включаючи насильство, булінг і експлуатацію

- Проблеми алгоритмічного контролю: Однією з особливостей TikTok є його алгоритмічна система рекомендацій, яка курує вміст, який користувачі бачать у своєму стрічці.

Це може призводити до обмеженої свободи вибору контенту та може бути обтяжливим для творців, які хочуть більшу контроль над своїм вмістом.

- Проблеми залежності від алгоритму: За своєю природою TikTok спрямований на максимальне залучення користувачів та збільшення часу, проведеного на платформі. Це може стати проблемою для користувачів, які хочуть керувати своїм часом та не допустити відволікання.

- Проблеми з аудієнцією та розповсюдженням: Хоча TikTok має велику аудиторію, конкуренція на платформі також є великою.

Крім того, використання TikTok може мати вплив на психічне здоров'я та самопочуття користувачів. Завдяки безперервному засвоєнню контенту та його алгоритмам, користувачі можуть відчувати тиск створювати власний "популярний" вміст або відчувати неадекватну самооцінку порівняно з іншими користувачами.

Такі фактори можуть призводити до стресу, тривоги та незадоволення своїм власним життям. Ці аспекти психологічного впливу TikTok стають предметом досліджень та обговорень.

1.4. Характеристика потенційної аудиторії клієнтів розроблюваної платформи

Характеристика потенційної аудиторії клієнтів платформи завантаження та перегляду відео файлів може включати наступні аспекти (рис. 1.7).

– Вікова група: аудиторія може бути різноманітною віковою групою, від підлітків до дорослих. Залежно від специфіки контенту, сервіс може привертати підлітків, молодь або дорослих користувачів.



Рисунок 1.7 – Основі аспекти потенційної аудиторії клієнтів

– Інтереси та хобі: аудиторія може мати різні інтереси та хобі, такі як музика, фільми, спорт, мода, геймінг, кулінарія, подорожі, наука, освіта тощо. Сервіс може надавати різноманітний контент, щоб задовольнити різні інтереси аудиторії.

– Географічна приналежність: аудиторія може бути розподілена по різних регіонах світу або бути орієнтована на певну географічну область. Мова та культурні впливи можуть грати роль у виборі контенту та взаємодії з сервісом.

– Професійні інтереси: деякі відеохостинги можуть спеціалізуватися на конкретних професійних групах, таких як музиканти, відеографи, блогери, викладачі тощо. Аудиторія може складатися з фахівців та аматорів відповідної галузі, які шукають платформу для спільного обміну своїм вмінням та взаємодії з однодумцями.

– Технічна готовність: аудиторія може мати різний рівень технічних навичок та здатність використовувати різноманітні функції сервісу. Деякі користувачі можуть бути більш технічно обізнані і зручно володіти функціями завантаження, відтворення та взаємодії з відео, тоді як інші можуть виявляти певні труднощі і бажати більш простого та інтуїтивного інтерфейсу.

– Мобільність: зростання використання смартфонів і планшетів призводить до збільшення аудиторії, яка споживає відеоконтент на мобільних пристроях. Така аудиторія може цінувати оптимізовані додатки та адаптивний дизайн для зручного перегляду відео на різних пристроях.

– Соціальні зв'язки: деякі користувачі можуть шукати можливості взаємодії та спілкування з іншими користувачами. Платформа завантаження та перегляду відео файлів може надавати функції коментування, обміну повідомленнями, створення спільнот та можливість підписатися на улюблені канали.

– Зацікавленість у контенті: аудиторія може бути зацікавлена в різних типах контенту, таких як розважальний, освітній, спортивний, музичний, науковий, політичний тощо. Сервіс може надавати різноманітні категорії та рекомендації для того, щоб відповідати потребам різних користувачів.

Ці характеристики можуть варіюватися в залежності від конкретної платформи розповсюдження відео файлів та її цільової аудиторії. Комбінація цих факторів допомагає розуміти, яка аудиторія має бути залучена та яким контентом вона може бути зацікавлена.

Важливо розуміти свою потенційну аудиторію, аналізувати її потреби, інтереси та звички споживання відеоконтенту. Це допоможе розробити та просувати відповідний контент, забезпечуючи задоволення потреб клієнтів та залучення нових користувачів до платформи завантаження та перегляду відео файлів.

Ціль і призначення проекту розробки платформи завантаження та перегляду відео полягають у створенні і наданні платформи, де користувачі можуть завантажувати, зберігати, відтворювати та обмінюватися відеоконтентом.

2. ПРОЕКТУВАННЯ СТРУКТУРИ І БАЗИ ДАНИХ ПЛАТФОРМИ ЗАВАНТАЖЕННЯ ТА ПЕРЕГЛЯДУ ВІДЕО ФАЙЛІВ

2.1. Опис процесу розробки платформи завантаження та перегляду відео файлів

Процес розробки системи може бути поділений на кілька етапів.

Першим етапом є аналіз специфікацій і потреб, де визначаються можливості системи, обмеження, вимоги до якості та безпеки, а також функціональні та нефункціональні вимоги.

Після аналізу переходимо до етапу проектування архітектури системи, де встановлюється структура системи, розподіл на частини та опис того, як вони взаємодіють між собою. На цьому етапі розробники також обирають необхідні технології, інструменти та фреймворки для впровадження системи.

Наступним етапом є створення бази даних, де збирається інформація про користувачів, коментарі, відео та інші дані, необхідні для системи. Розробка структури бази даних та зв’язків має гарантувати ефективний доступ до інформації і захист від небажаного доступу.

На етапі розробки інтерфейсу користувача дизайнери та розробники працюють над створенням зручного та привабливого інтерфейсу, що відповідає потребам користувачів.

При розробці функціональності системи розробники створюють програмну логіку, яка забезпечує працездатність системи, включаючи управління користувачами, завантаження, зберігання, перегляд та коментування відео.

Після завершення розробки система піддається налагодженню та тестуванню. Цей етап включає перевірку системи на відповідність вимогам, виявлення помилок, проведення функціональних, навантажувальних,

безпекових тестів та інших видів тестувань. Це дозволяє виявити помилки та недоліки системи та виправити їх перед запуском.

Коли тестування проведено успішно, система готова до запуску. Розробники випускають її на ринок і починають надавати підтримку, оновлювати та покращувати систему на основі відгуків користувачів та змін у вимогах та технологіях.

Весь процес розробки системи включає аналіз потреб та специфікацій, проектування архітектури, створення бази даних, розробку інтерфейсу користувача, зростання функціональності, налагодження та тестування, а також запуск та підтримку. Це ітеративний процес, де кожен етап взаємозв'язаний з попереднім і наступним, що дозволяє створити функціональну та якісну систему.

2.2. Розробка алгоритму функціонування платформи

Алгоритм функціонування розроблюваної платформи повинен бути побудований за принципами взаємодії ПЗ з користувачем, що будуть визначати алгоритм виконання програми.

Спочатку користувачу буде показуватися домашня сторінка, на якій будуть найактуальніші на даний момент відео, і на якій він зможе обрати відео, яке хоче подивитися, натиснувши на нього.

Крім того, він зможе за допомогою бокового меню вибрати інші опції, що будуть йому доступні, наприклад перехід до сторінки авторизації, де користувач зможе авторизуватися і мати доступ до тих сценаріїв, які доступні авторизованому користувачу, або ж перейти на сторінку реєстрації, де неавторизований користувач, що немає облікового запису зможе створити новий, заповнивши основну інформацію про себе.

Спрощена схема роботи платформи представлена на рис. 2.1.

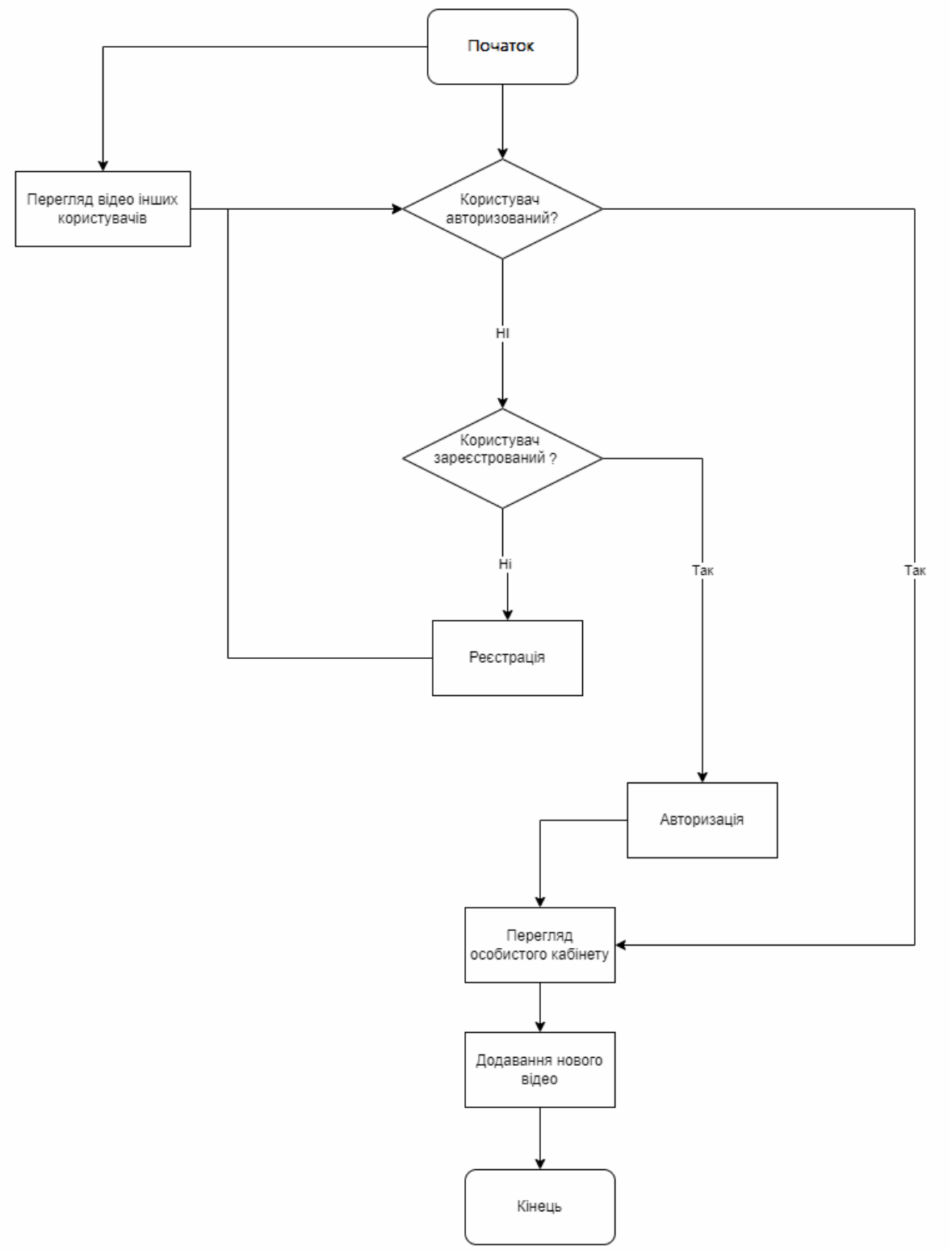


Рисунок 2.1 – Спрощений алгоритм роботи платформи

Пройшовши реєстрацію, або ж авторизувавшись, користувач буде мати можливість перейти до таких пунктів як:

− сторінка власного профілю, це користувач отримає загальну інформацію про те, що саме має на даний момент, тобто дату реєстрації, ім’я особистого каналу, email та кількість його відео;

− сторінка особистих відео, на сторінці будуть зображені особисті відео користувача, що будуть мати, як і інші, такі параметри, як дата створення, кількість переглядів, а також відображення самого відео.

Окрім того, користувач буде мати можливість опублікувати відео через сторінку особистих відео, натиснувши на відповідну кнопку, після чого відкриється модальне вікно для завантаження відео, а також отримання основної інформації про нього.

2.3. Алгоритм додавання нового відео файлу до платформи

Програмна система розповсюдження відео файлів будується на основі клієнт-серверної архітектури, яка включає два компоненти: клієнт і сервер. В даний час клієнт-серверна архітектура як метод розробки додатків та інформаційних систем корпоративного рівня, набула визнання, і стало широко поширюватися. Збільшення безпеки даних у розглянутій архітектурі пов'язане з тим, що виконання запитів всіх клієнтів реалізується єдиною програмою, яка розміщена на сервері.

Клієнт – це комп’ютер на стороні користувача, який відправляє запит до сервера для отримання інформації або виконання певних дій. Сервер – це потужний комп’ютер або обладнання, яке вирішує певні завдання, виконує програмний код, сервісні функції та забезпечує доступ користувачів до ресурсів та зберігання інформації та баз даних. Сервер забезпечує загальні всім користувачів правила використання баз даних, може керувати режимами доступу клієнтів до інформації, не дозволяючи, зокрема, паралельне виправлення однієї записи кількома користувачами. Скорочується складність

клієнтських додатків, оскільки в них відсутній код, який пов'язаний з контролем баз даних та розмежованим доступом до неї.

Відмінною рисою клієнт-серверної архітектури є існування окремих серверів баз даних, які приймають запити на мові SQL, і які проводять сортування даних, пошук даних та агрегування даних. Характерна властивість для серверів баз даних – це наявність сховища метаданих. У цьому сховищі записуються обмеження цілісності інформації, конструкція баз даних, серверні та формати процедур, які обробляють дані за подіями або викликом у програмі. Об'єктами реалізації в таких додатках, за винятком логіки та діалогу обробки, є спочатку модель реляційної баз даних і набір SQL-операторів, що відноситься з ними, для типових запитів до бази даних.

Для роботи зі складними інформаційними програмами, в яких безліч користувачів та заплутана логіка, в більшості випадків не використовуються дворівневі схеми архітектури клієнт-сервера, оскільки вони можуть призвести до проблем. У разі застосовується багаторівнева архітектура.

Для успішної роботи з такою моделлю проекту необхідно розуміти весь стек інформаційних технологій, що доступні для розробки на клієнтській та серверній сторонах, а також протоколи та методи обміну даними між клієнтом та сервером. Для роботи з клієнтською стороною сайту найкраще використовувати мову програмування JavaScript.

Було розроблено алгоритм додавання нового відео до платформи включає в себе наступні етапи:

1. Розпочати ініціалізацію додавання нового відео;

2. Заповнення необхідних полів, такі як: назва, опис, прев’ю, відео;

3. Надіслати запит до серверу про завантаження нового відео;

4. Якщо всі поля заповнено коректними даними, надати повідомлення користувачу;

5. Додати внесене відео до бібліотеки.

На рисунку 2.2 зображено алгоритм роботи завантаження нового відео до платформи.

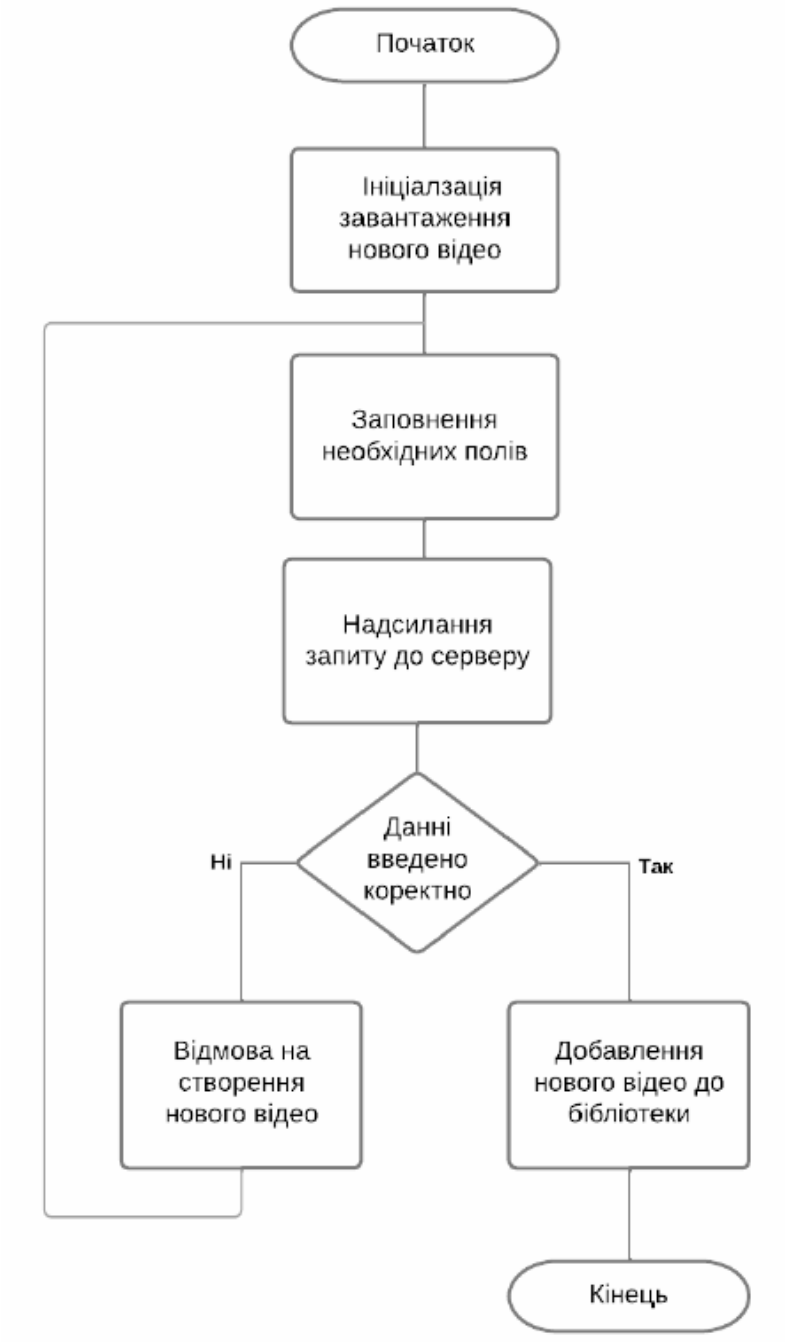


Рисунок 2.2 - Алгоритм роботи завантаження нового відео до платформи

Таким чином використання даного алгоритму підвищує надійність веб-платформи, адже при перевірці буде знайдено некоректні дані, які призводять до некоректної роботи сервісу.

2.4. Розробка структури бази даних

Сутність бази даних - це абстрактне поняття, яке визначає, що база даних складається з набору сутностей, які представляють об'єкти в реальному світі, що зберігаються у базі даних. Сутності мають атрибути, які описують характеристики цих об'єктів, і взаємозв'язки з іншими сутностями.

Сутність бази даних — це об'єкт або концепт, що має значення для домену, який ви моделюєте, і про який потрібно зберігати дані. Це може бути що завгодно, від особи, місця до події чи ідеї. В контексті структурування баз даних, кожна сутність представлена в таблиці, де рядки (записи) містять екземпляри сутності, а стовпці (поля) визначають атрибути або характеристики цієї сутності.

Враховуючи, що розроблювана платформа буде постійно працювати з великою кількістю інформації, проводити її вибірку, обробку та інші можливі операції, необхідно визначити використовувані моделі даних, сутності та їх призначення, провести опис та моделювання логічної структури даних.

Логічна модель даних використовується у таких випадках, коли необхідно детально описати дані та взаємозв’язки на дуже високому рівні. Дані описуються на дуже абстрактному рівні без їх фізичної демонстрації в БД. В основному він включає сутності та взаємозв’язки між ними, а також атрибути кожної сутності.

Логічна модель даних включає первинні ключі кожної сутності, а також зовнішні ключі. При її створенні перші сутності та їх взаємозв’язки ідентифікуються за допомогою ключів. Потім ідентифікуються атрибути кожної сутності. Логічна модель даних не залежить від системи управління базами даних, оскільки не описує фізичну структуру реальної бази даних. База даних програми повинна мати наступні сутності задля реалізації визначених функцій:

– сутність User, що представляє собою користувача, який авторизується на сайті з усіма основними його даними;

– сутність Video, що представляє собою загальну інформацію про відео, яке було викладено користувачем.

Структура БД зображена на рис. 2.4.

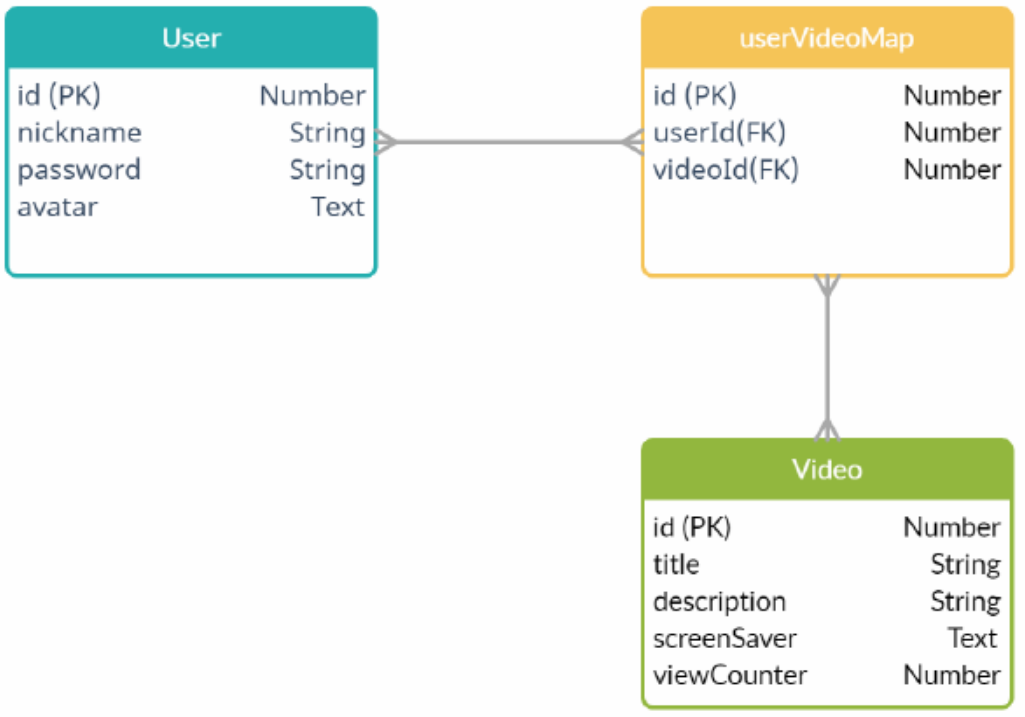


Рисунок 2.4 – Модель бази даних

Таблиця User буде мати наступні поля:

– поле id – первинний ключ користувача;

– поле password – пароль користувача у системі;

– поле email – імейл облікового запису користувача;

– поле avatar – закодований у строку аватар користувача.

Таблиця Video буде мати наступні поля:

– поле id – первинний ключ відео;

– поле description – опис відео;

– поле title – назва відео;

– поле fileHashNumber – шлях до відео, що знаходиться на сервері;

− поле viewCounter – лічильник переглядів;

− поле screenSaver – заставка відео, яку бачить користувач перед переглядом.

Кожне відео повинно мати зв’язок з користувачем у базі даних, цю функцію виконує таблиця userVideoMap, яке буде мати наступні поля, що описані нижче:

− id – первинний ключ;

− userId – зовнішній ключ, що зв’язує з таблицею User;

− videoId – зовнішній ключ, що зв’язує з таблицею Video.

2.5. Моделювання архітектури проекту

Згідно структури проекту та отриманих результатів в ході визначення ключових компонентів його можна поділити за типом трирівневої архітектури, модель якої зображено на рисунку 2.5.

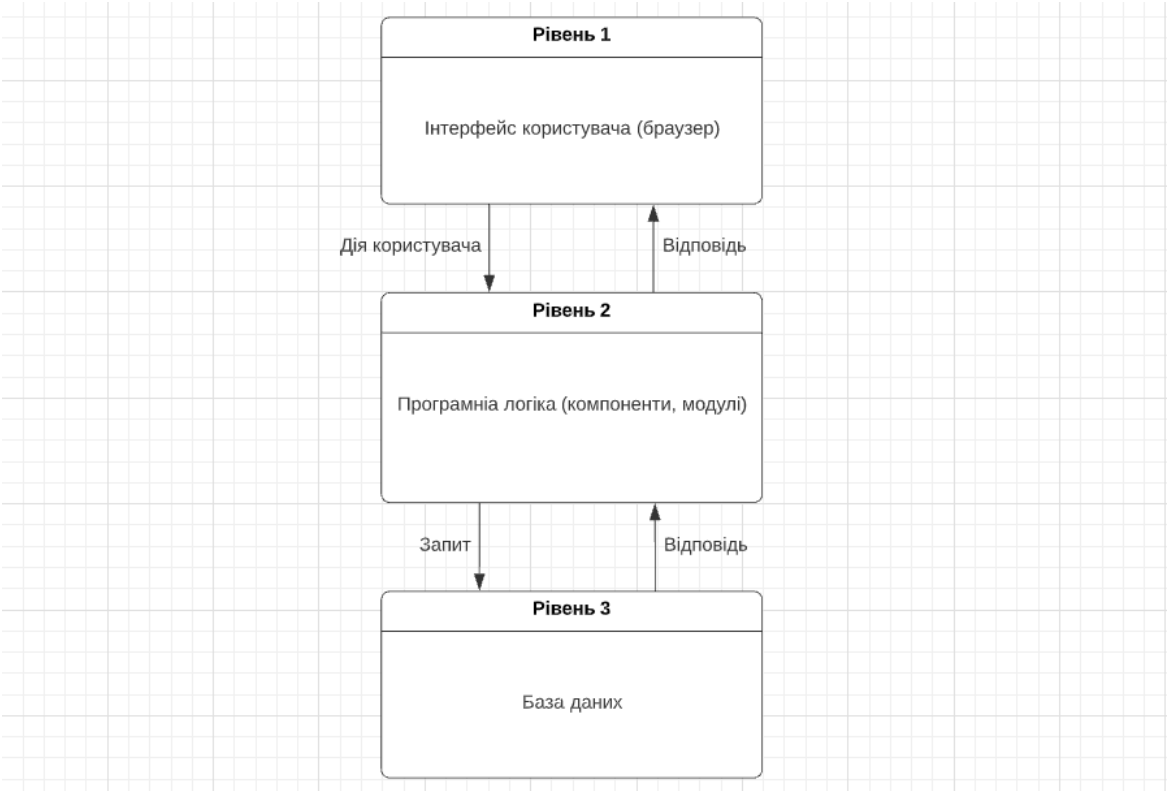


Рисунок 2.5 – Архітектура платформи

Трирівнева архітектура – це тип архітектури «клієнт-сервер», за умовами якої функціональна логіка процесу, доступ до даних, зберігання даних у комп'ютері та користувацький інтерфейс розробляються та підтримуються як незалежні модулі на окремих платформах.

На першому рівні розміщений інтерфейс, в якому проводиться відображення інформації та вся взаємодія користувача веб-сторінки з її елементами. Цей рівень слугує простором між людиною та програмною логікою сервісу.

В свою чергу, рівень логіки виконує обробку виконуваних користувачем дій з інтерфейсом. В залежності від результату обробки програмний модуль надсилає відповідь для відображення.

Третій рівень визначається, як рівень даних, тобто БД. На цьому рівні відбувається зберігання інформації, яку відправляє рівень логіки, в розділах даних та її адміністрування в залежності від запитів рівня логіки.

Метод використання багаторівневої архітектури дозволяє підвищити безпеку та надійність сервісу, спростити роботу для розробників при підтримці працездатності проекту, його вдосконаленні, та пошуку несправностей.

2.6. Реалізація класів та моделей для роботи з базою даних

Як описано вище, для таблиць Videos, User, UserVideosMap були створені та описані відповідні моделі в Express.js, з описом їх хуків, зв'язків, полей та інших даних. Вони повторюють все, що повинно включатися у зазначеній таблиці. За допомогою наступного фрагменту коду було зроблено імпорт даних моделей БД для доступу та подальшого використання:

```
const  sequelize = new  Sequelize(
process.env.MYSQL_DB,
process.env.MYSQL_USER,
```

```
process.env.MYSQL_PASSWORD,
{
host: process.env.MYSQL_HOST,
port: process.env.MYSQL_PORT,
dialect: "mysql",
}
);
const models = {
User: User.init(sequelize, Sequelize),
Video: Video.init(sequelize, Sequelize),
userVideosMap: userVideosMap.init(sequelize, Sequelize)
};
```

Клас моделі User виглядає наступним чином:

```
class User extends Model {
static init(sequelize, DataTypes) {
return super.init(
{
nickname: DataTypes.STRING,
password: DataTypes.STRING,
email: {
type: DataTypes.STRING,
validate: { isEmail: true },
},
avatar: DataTypes.TEXT,
},
{
sequelize,
modelName: "User",
}
);
}
static associate(models) {
this.hasOne(models.userVideosMap, {
foreignKey: "userId",
```

```
targetKey: "id",
as: "userLink",
});
}
}
```

Клас, як і інші класи моделей, складається з двох частин: конструктора, у якому описані поля, та методу «associate», через який описані зв’язки з іншими таблицями. У випадку таблиці Video – зв’язок типу один-до-одного з таблицею UserVideosMap.

Клас UserVideos, що описує таблицю UserVideosMap, має таку ж саму структуру:

```
class UserVideos extends Model {
static init(sequelize, DataTypes) {
return super.init(
{
userId: DataTypes.INTEGER,
videoId: DataTypes.INTEGER,
},
{
timestamps: false,
sequelize,
modelName: "userVideosMap",
}
);
}
static associate(models) {
this.belongsTo(models.User, {
foreignKey: "userId",
targetKey: "id",
as: "userLink",
});
}
}
```

Цей клас має два FK (foreign keys) у вигляді полів userId та videoId. FK userId асоціюється з полем id у таблиці User, тобто є описаним у вигляді userLink під час видобутку даних при використанні скопів, що описані також у класі User.

Клас Video, описуючий таблицю Videos має дещо іншу структуру, бо додатково описує хуки, які повинні виконатися під час створення нового запису, а також скопи, завдяки яким ми можемо достати дані з цієї таблиці «ланцюжком», де так звані скопи виглядають наступним чином:

```
const scopes = {
withUser(userId) {
return { include: [{
association: "links",
where: { userId },
attributes: [],} ],},
};
```

Для взаємодії з моделями було організовано сервіси, чиєю ціллю є взаємодія з таблицями БД для отримання інформації.

У сервісі user.service.js створено наступні методи:

− userCreate = async function (userData) – метод сервісу, що здійснює запит для створення нового облікового запису користувача у таблицю;

− userFind = async function (email) – метод сервісу, що здійснює запит для пошуку, чи існує обліковий запис з заданою поштою;

− userUpdate = async function (body, id) – метод сервісу, що здійснює оновлення облікового запису за номером.

У сервісі video.service.js створено наступні методи:

− create = async function (videoData, userId) – метод сервісу, що здійснює запит для створення нового запису у таблиці про інформацію про відео;

− update = async function (body, id) – метод сервісу, що здійснює запит для оновлення існуючої інформації у таблиці;

− findAll = async function () – метод сервісу, що повертає список існуючих відео з таблиці;

− getMy = async function (email) – метод сервісу, що повертає список власних відео користувача;

− findOne = async function (id) – метод сервісу, що повертає інформацію про відео за номером.

Наступним кроком є контролери, що виконують більш складну логіку і завдяки сервісам взаємодіють з БД.

У контролері video.controller.js створено наступні поля та методи:

− async function onVideoUpload(req, res) – метод контролера, що виконується під час завантаження відео на сервер;

− const MAX_SIZE – змінна, що зберігає інформацію про максимальний розмір відео;

− const videoStorage – змінна, що створює відео сховище з вказаним локальним шляхом;

− const videoUpload – зміна, що зберігає загальну інформацію, що потрібна під час завантаження відео, в тому числі, інформацію про обробку помилок;

− async function createVideo(req, res) – метод контролера, що виконує необхідні операції для створення відео;

− async function getVideoData(req, res) – метод контролера, що відсилає на клієнтську частину необхідну інформацію про конкретне відео;

− async function getVideoList(req, res) – метод контролера, що відсилає на клієнтську частину список відео з основною інформацією;

− async function getMyVideoList(req, res) – метод контролера, що відсилає на клієнтську частину список відео конкретного користувача;

− async function updateViews(req, res) – метод контролера, що слугує для відновлення рахунку переглядів;

Окрім того, є ще два методи, що описані у самому запиті на серверну частину, зокрема метод для стрімінгу безпосередньо відео, а також його завантаження по частинам на самий сервер.

2.7. Архітектура веб-платформи

Архітектура веб-платформи – це взаємозв'язки і взаємодія між компонентами програми, такими як системи проміжного програмного забезпечення, інтерфейси користувача та бази даних. Загальна концепція архітектури веб-додатків відповідає концепції користувача браузера, який запускає програму, яка може працювати на кількох веб-сайтах. В загальному архітектуру веб-платформи можна визначити наступним процесом:

• Користувач шукає певну URL-адресу, яку браузер знаходить і надсилає запит.

• По мережі дані надсилаються з сервера до браузера, а потім виконуються браузером, щоб він міг відобразити запитану сторінку.

• Користувач переглядає сторінку та взаємодіє з нею.

Оскільки технологічний світ продовжує розвиватися, додатки вважаються головною частиною цього процесу трансформації. Сучасна архітектура додатків та її розвиток постійно вдосконалюються як у front-end, так і в back-end можливостях. Зокрема, на стороні back-end або сервера з'являються численні підходи до архітектури розробки додатків, які з'являються, щоб впоратися з поточними потребами розробки, такими як мікросервіси, без серверні (Serverless) архітектури та односторінкові програми (SPA).

Архітектура веб-додатків складається з кількох компонентів, які допомагають створити його цифрову структуру. Ці компоненти можна

розділити на дві категорії: компоненти програми користувальницького інтерфейсу та структурні компоненти. Компоненти програми інтерфейсу користувача відносяться до веб-сторінок, на яких відображаються інформаційні панелі, журнали, сповіщення, налаштування конфігурації тощо. Вони не мають відношення до структурної розробки програми і більше орієнтовані на інтерфейс користувача. Структурними компонентами, які є справжньою основою процесу розробки програми, є:

• Веб-браузер або клієнт.

• Сервер веб-додатків.

• Сервер бази даних.

Веб-браузер або клієнт — це інтерфейс, що відображає функціональність веб-програми, з якою взаємодіє користувач. Цей контент, який надається клієнту, може бути розроблений за допомогою HTML, JavaScript і CSS і не потребує адаптацій, пов’язаних з операційною системою. По суті, веб-браузер або клієнт керують тим, як кінцеві користувачі взаємодіють із програмою. Сервер веб-додатків керує бізнес-логікою та збереженням даних і може бути створений за допомогою PHP, Python, Java, Ruby, .NET, Node.js та інших мов програмування. Сервер бази даних надає та зберігає відповідні дані для програми.

Крім того, він також може надавати бізнес-логіку та іншу інформацію, якою керує сервер веб-додатків.

Така взаємодія відображена на рисунку 2.6.

Запити клієнта обробляються і на виході отримуємо відповідь від сервера.

Існує три типи найбільш вживаних архітектури веб-додатків:

1. Single Page Applications (SPA), або односторінкова архітектура

2. Мікросервісна архітектура

3. Serverless архітектура, або безсерверна

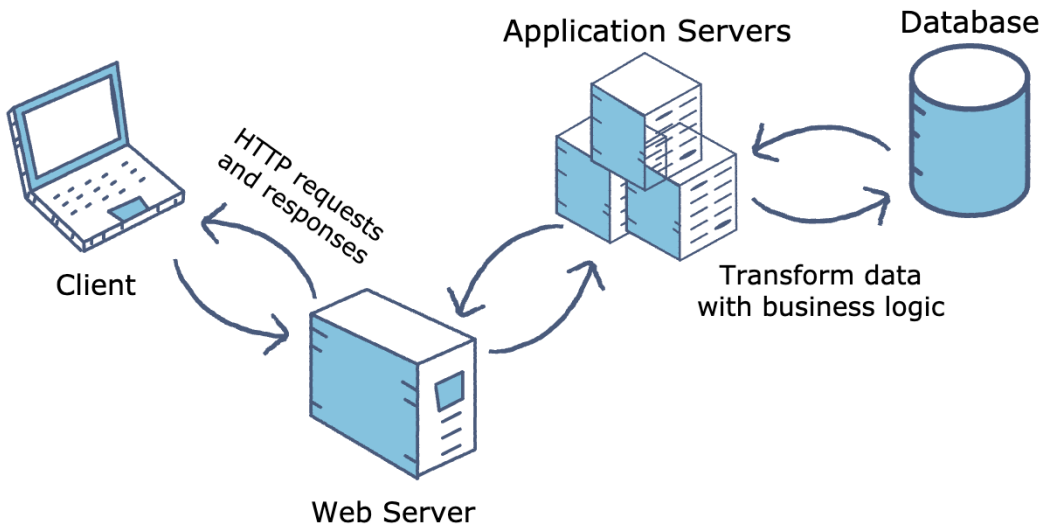


Рисунок 2.6 – Алгоритм взаємодії компонентів архітектури веб-платформи

Вибір правильної архітектури впливає на безпосередньо майбутню взаємодію користувача з веб-додатком, швидкість розробки, масштабованість, тестування, підтримку, життєвий цикл продукту та інше. Особливості вищенаведених типів архітектури веб-додатків описано у таблиці 2.1.

Таблиця 2.1 – Типи архітектури веб-додатків

Тип	**Опис**
Single Page Application	Односторінкові веб-додатки взаємодіють з користувачем більш динамічно, надаючи оновлений вміст на поточній сторінці, замість того, щоб завантажувати повністю нові сторінки із сервера з кожною новою дією користувача. Це допомагає запобігти перерви в роботі користувача, перетворюючи поведінку програми так, щоб вона нагадувала традиційну настільну програму. Основними технологіями є AJAX, скорочення від Asynchronous JavaScript і XML , який є основою для комунікації сторінок, що робить можливими SPA.

Мікросервісна	Мікросервіси — це невеликі легкі служби, які виконують певну єдину функціональність. Структура архітектури мікросервісів пропонує широкий набір переваг, які допомагають розробникам бути більш продуктивними та прискорювати процес розгортання програмних додатків. Оскільки компоненти програми безпосередньо не залежать один від одного, їх не потрібно розробляти однією мовою програмування; розробники мають вільний діапазон для роботи з технологічним стеком на свій вибір, який найкраще відповідає потребам.
Serverless	Цей підхід використовує сторонні хмарні інфраструктурні послуги для аутсорсингу серверів та управління інфраструктурою.

Мікросервіси – це архітектурне рішення, яке базується на розподілі модулів на окремі системи, які спілкуються між собою за допомогою повідомлень. Побачити приклад архітектури можна на рисунку 2.7.

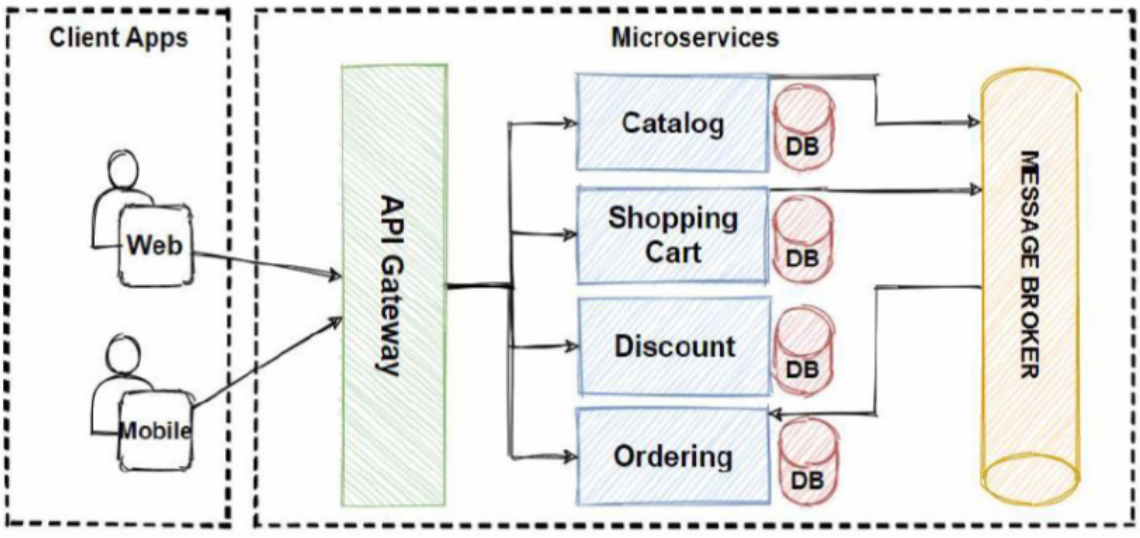


Рисунок 2.7 – Мікросервісна архітектура

Мікросервісна архітектура не зовсім підходить під середні і тим більше маленькі проекти.

Серверлес – це архітектурне рішення, яке фокусується на розробці замість розгортання та взаємодії між сервісами. Це альтернатива мікросервісам, яка автоматизує все розгортання завдяки хмарним технологіям (рисунок 2.8).

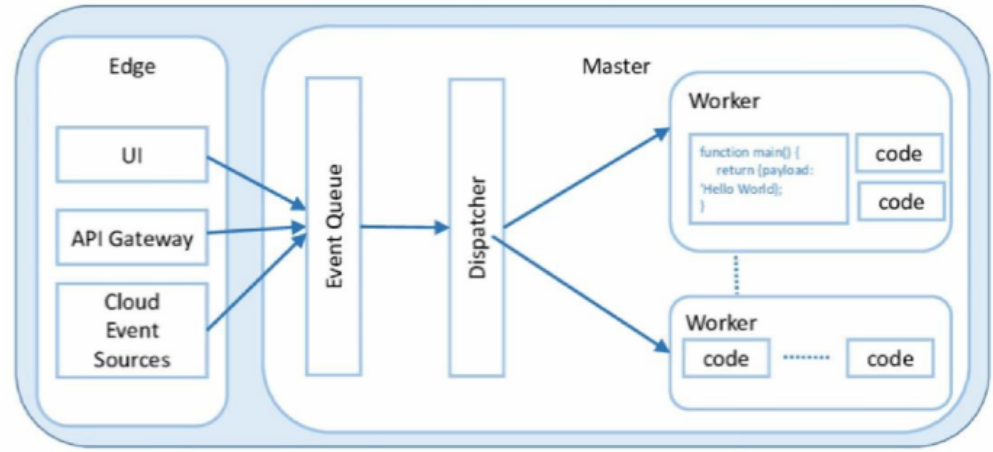


Рисунок 2.8 – Serverless архітектура

Переваги:

- гнучкість. Серверлес дуже гнучкий за рахунок відокремленості одного модуля від іншого;

- абстракція від операційної системи. Хмара сама вирішить, яка ОС потрібна та як її потрібно налаштувати;

- легкий поріг входу;

- надійність. При правильній побудові, надійність така ж як у мікросервісів.

Недоліки:

- гнучкість. Це як перевага, так і недолік, тому що розробник обмежений вплив на масштабування, все вирішує хмара;

- налагодження ускладнене взаємодією між компонентами;

- Vendor Lock. Кожна хмара працює за власними правилами, тому перехід з однієї хмари на іншу майже неможливий.

2.8. Основні аспекти щодо призначення розроблюваної платформи

Основні аспекти цілей та призначення проекту включають наступне (рис. 2.9):



Рисунок 2.9 – Основні аспекти цілей та призначення проекту

– Забезпечення простоти та зручності: однією з головних метою розробки сервісу розповсюдження відео файлів є створення зручного та простого у використанні інтерфейсу для користувачів. Це включає у себе легке завантаження відео, швидке відтворення, просту навігацію та зручну систему керування вмістом.

– Забезпечення надійності та безпеки: проект має на меті забезпечити безпеку відеоконтенту, запобігаючи несанкціонованому доступу до відеофайлів або порушенню авторських прав. Надійна система зберігання та резервне копіювання даних також є важливими аспектами для забезпечення безпеки відеоконтенту.

– Підтримка широкого спектру форматів та якостей відео: сервіс повинен підтримувати різні формати відео (наприклад, MP4, AVI, MKV) та якості відтворення (SD, HD, 4K), щоб задовольнити потреби різних користувачів і їхні можливості з пристроїв та інтернет-з'єднання.

– Забезпечення функціональних можливостей: проект повинен надавати різноманітні функції, які полегшують споживання та взаємодію з відео. Це може включати можливість коментувати, оцінювати, підписуватися на канали, рекомендації на основі особистих інтересів та інші соціальні функції.

– Розширення аудиторії та залучення: проект повинен привертати як можна більше користувачів до своєї платформи. Це може включати просування сервісу, створення партнерських програм, співпрацю з відомими відеоблогерами та інші маркетингові заходи, що сприяють залученню нових користувачів.

– Аналітика та звітність: проект повинен забезпечувати інструменти аналізу та звітності для контент-творців, щоб вони могли вимірювати успішність свого відеоконтенту, розуміти поведінку аудиторії та приймати на цій основі відповідні рішення.

– Забезпечення доступу до різноманітного вмісту: сервіс завантаження та перегляду відео файлів дозволяє користувачам отримати доступ до широкого спектру відео – від розважального контенту до навчальних матеріалів. Це допомагає розширити знання, збагатити розваги та забезпечити різноманіття вмісту для користувачів.

2.9. Буферизація та потоки в JS. JWT-токен

Буфер - це область пам'яті. Більшість розробників JS знайомі набагато менше ця концепція, у порівнянні з програмістами, які використовують системні мови програмування, які щодня безпосередньо взаємодіють з пам'яттю. Він являє собою шматок пам'яті фіксованого розміру (розмір не може бути змінений), виділений за межами рушія JavaScript. Можна уявити буфер як масив цілих чисел, кожне з яких представляє байт даних.

Буфери були введені, щоб допомогти розробникам працювати з двійковими даними в екосистемі, яка традиційно розглядаються лише рядки, а не двійкові файли. Буфери в js не пов’язані з концепцією буферизації даних. Ось що відбувається, коли потоковий процесор отримує дані швидше, ніж вони можуть переварити.

Приклад принципу роботи буферу подано на рисунку 2.10.

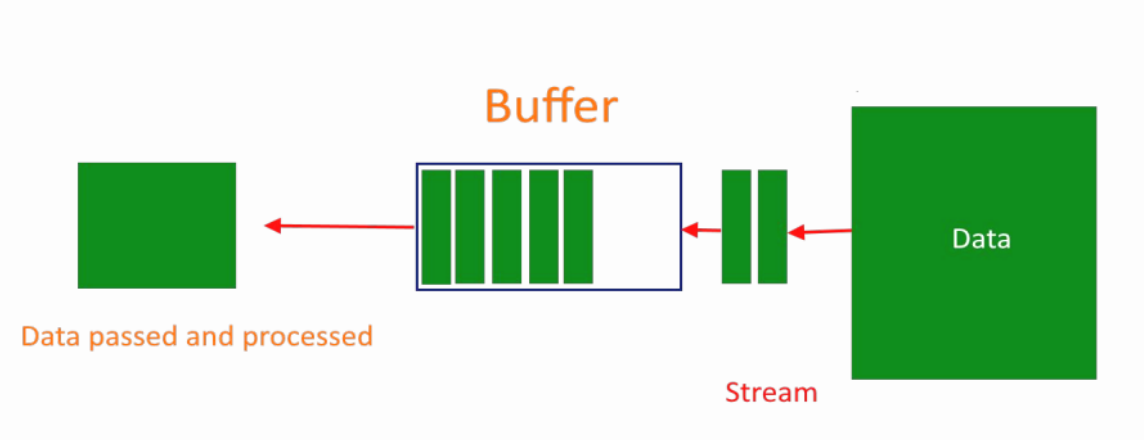


Рисунок 2.10 – Буфер в JS

Потоки є однією з основоположних концепцій, які забезпечують додатки JS. Вони є методом обробки даних і використовуються для послідовного читання або запису вхідних даних у вихідні дані. Потоки – це спосіб обробки читання/запису файлів, мережевого зв’язку тощо вид наскрізного обміну інформацією ефективним способом. Що робить потоки унікальними, так це те, що замість програми читає файл у пам'ять все

одночасно, як традиційно, потоки зчитують шматки даних по частинах, обробляти його вміст без збереження всього цього в пам'яті.

Використання потоків це:

• Ефективність пам’яті: вам не потрібно завантажувати великі обсяги даних у пам’ять перш ніж ви зможете його обробити

• Ефективність часу: потрібно значно менше часу, щоб почати обробку даних якнайшвидше як у вас, замість того, щоб чекати з обробкою до повного використання було передано Чому потоки

Типи потоків:

• З можливістю запису: потоки, в які ми можемо записувати дані. Наприклад, fs.createWriteStream() дозволяє нам записувати дані у файл за допомогою потоків.

• Читається: потоки, з яких можна зчитувати дані. Наприклад: fs.createReadStream() дозволяє нам читати вміст файлу.

• Дуплекс: потоки, які можна як читати, так і записувати. Наприклад, net.Socket

• Трансформація: потоки, які можуть змінювати або трансформувати дані під час їх запису та читати. Наприклад, у випадку стиснення файлів ви можете написати стислі дані та читання розпакованих даних у файл та з файлу.

Приклад принципу роботи потоку зчитування (рис. 2.11).

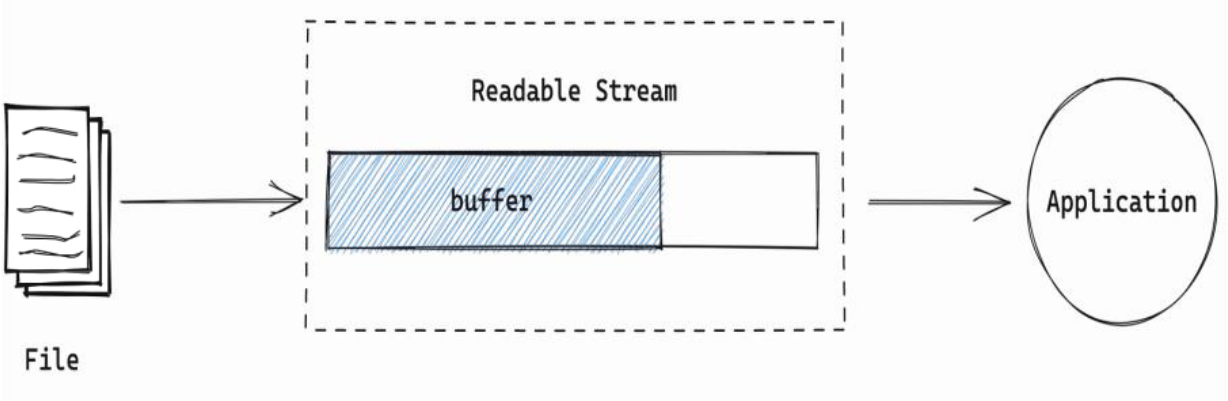


Рисунок 2.11 – Потік зчитування

Приклад роботи потоку запису подано на рисунку 2.12.

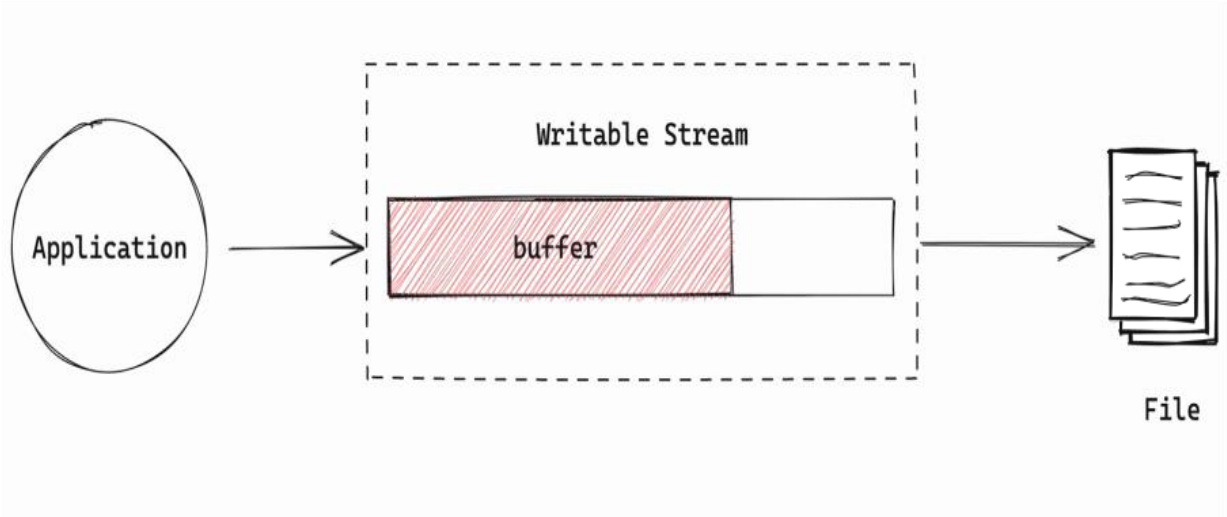


Рисунок 2.12 – Потік запису

Приклад роботи потоку зчитування і запису представлено на рисунку 2.13.

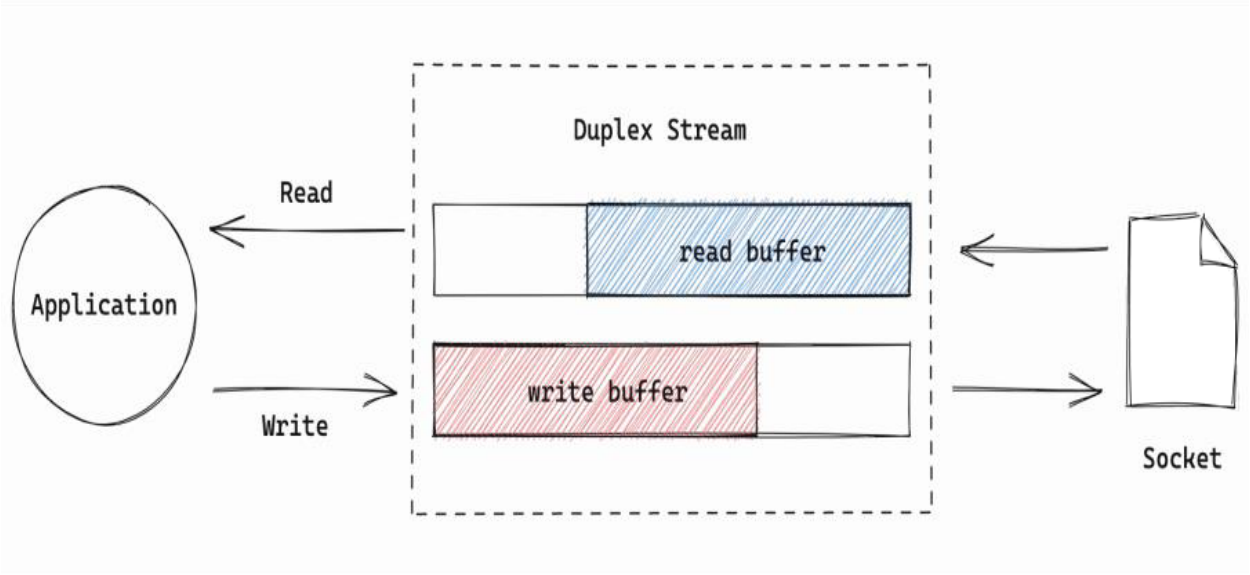


Рисунок 2.13 – Потік зчитування і запису

Приклад роботи потоку мутуючого потоку представлено нижче на рисунку 2.14.

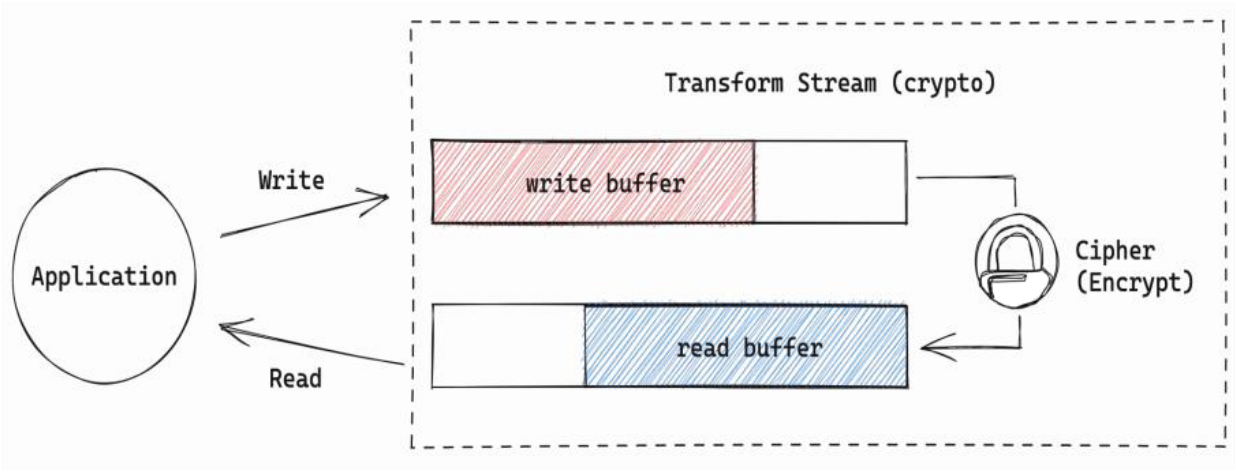


Рисунок 2.14 – Потік з мутацією

JSON Web Token - це об'єкт в JSON, який був описаний у відкритому стандарті RFC 7519. Він виступає як один з варіантів безпечного спілкування між користувачами. Для створення необхідно визначити заголовок із загальною інформацією по токену, корисні дані, такі як унікальний ідентифікатор користувача, його роль тощо. та підписи. Приклад роботи JWT токену подано на рисунку 2.15.

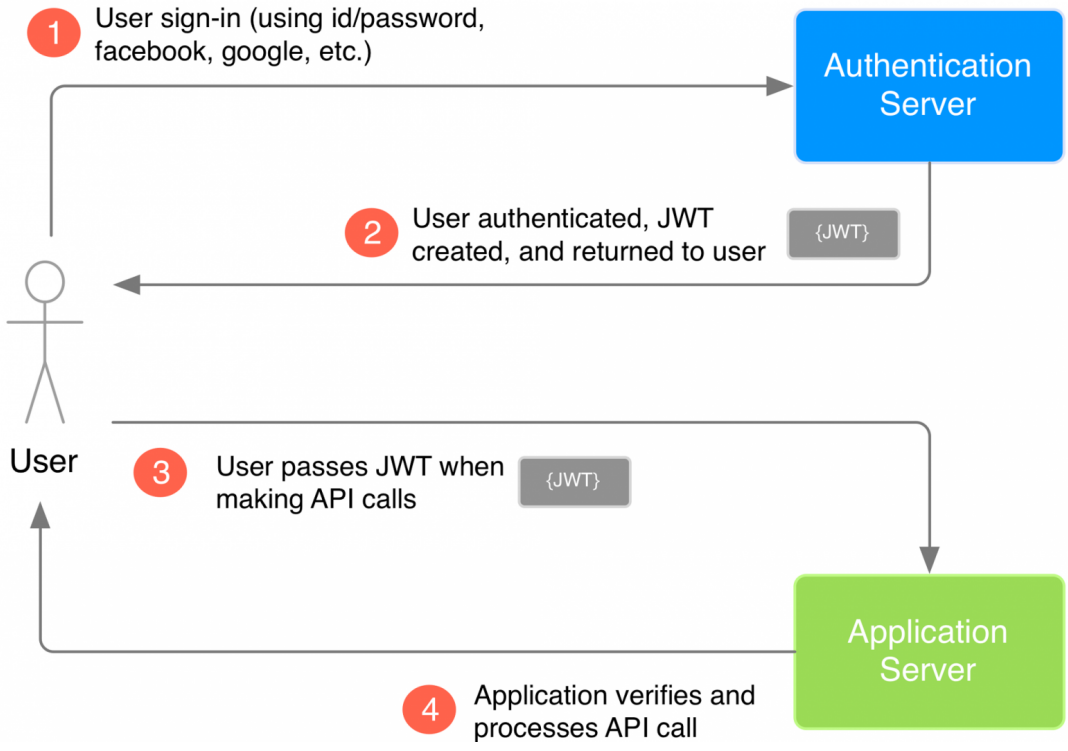


Рисунок 2.15 – Принцип роботі JWT

Додаток використовує JWT для перевірки аутентифікації користувача таким чином:

1. Спочатку користувач заходить на сервер аутентифікації за допомогою аутентифікаційного ключа (це може бути пара логін/пароль, або Facebook ключ, або Google ключ, або ключ від іншого обліку).

2. Потім сервер аутентифікації створює JWT та відправляє його користувачеві.

3. Коли користувач робить запит до програми API, він додає до нього отриманий раніше JWT.

4. Коли користувач робить API запит, програма може перевірити за переданим із запитом JWT чи є користувач тим, за кого себе видає. У цій схемі сервер програми налаштований так, що зможе перевірити, чи є вхідний JWT саме тим, що був створений сервером аутентифікації.

Приклад генерації JWT під час реєстрації подано на рисунку 2.16.

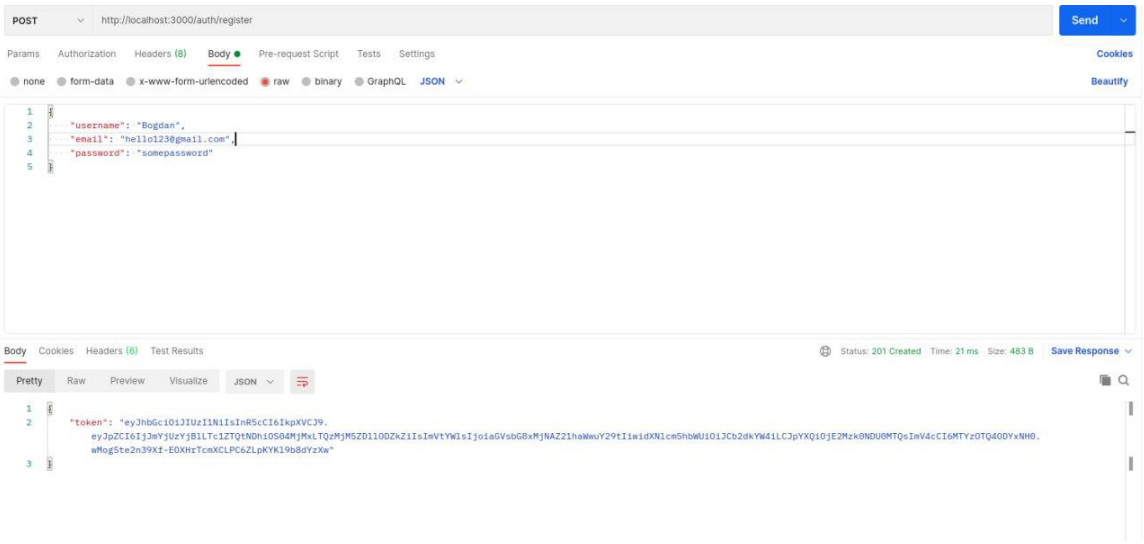


Рисунок 2.16 – Реєстрація користувача в системі

2.10. Опис мови Javascript та фреймворків для розробки

JavaScript може виконуватися не тільки засобами браузеру, але і на сервері або на пристоях, що підтримують JavaScript engine. Браузер має

вбудований механізм, який іноді називають «віртуальною машиною JavaScript». Двигуни складні. Але основи прості. Двигун (вбудований, якщо це браузер) читає («розбирає») сценарій. Потім він перетворює («компілює») сценарій у машинний код. А потім машинний код виконується досить швидко.

Механізм застосовує оптимізацію на кожному кроці процесу. Він навіть стежить за скомпільованим сценарієм під час його виконання, аналізує дані, що проходять через нього, і додатково оптимізує машинний код на основі цих знань. Можливості JavaScript значною мірою залежать від середовища, у якому він працює.

Безпека JavaScript є головним пріоритетом. Існує величезна кількість речей, які можуть піти не так, від програмних помилок і незахищених даних користувача до зловмисних атак. Хоча моніторинг помилок JavaScript може допомогти і виявити багато з цих проблем, розуміння загальних ризиків безпеки JavaScript і дотримання найкращих практик є не менш важливим.

Щоб контролювати вразливості JavaScript сторонніх розробників потрібно відстежувати всі пакети, які використовуються на веб-сайті. Це можна зробити за допомогою менеджера пакетів, наприклад npm, Yarn або pnpm.

Найкращі методи безпеки досягаються використанням лінтер JavaScript, аудіт залежностей за допомогою менеджера пакетів та перевірку цілісності підресурсу для зовнішніх сценаріїв.

Розробка сайтів на JavaScript відрізняється від інших мов програмування. У типових веб-додатках ми маємо справу з 3–20 елементами одночасно, тому продуктивність незначна. Навіть у Node.js ми розбиваємо дані на сторінки й дозволяємо своїй базі даних виконувати важку роботу. Зважаючи на те, як рідко можна побачити більше кількох елементів.

Продуктивність, пам’ять, розмір вихідного коду – все важливо. Ефективність у програмі значною мірою залежить від інтерпретатора

JavaScript. Здебільшого багато речей, які ви написали в минулому, працюватимуть швидше з кожним майбутнім оновленням інтерпретатора але може бути і навпаки (рис 2.17).

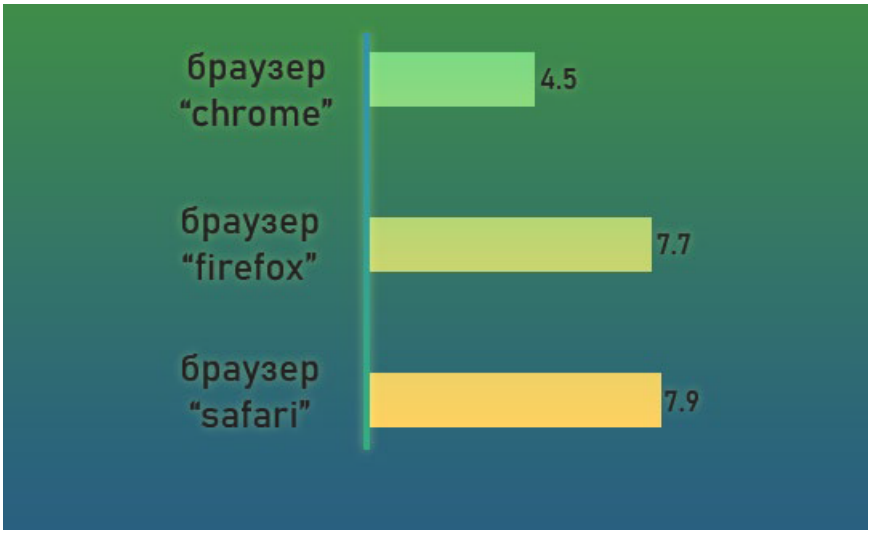


Рисунок 2.17 - Порівняння швидкості коду Javascript у браузерах

Завжди є деякі аспекти продуктивності інтерфейсу, які знаходяться поза контролем особливо при використанні JavaScript – найефективнішими будуть сайти з невеликим або відсутнім JavaScript (хоча це нереально з урахуванням вимог багатьох веб-сайтів).

Ресурси сайту ще повинні бути завантажені клієнтом. Незважаючи на те, що ми можемо пом’якшити це, об’єднавши програму в пакет, продуктивність зрештою залежить від швидкості мережі. JavaScript часто взаємодіє з різноманітними API. Хоча швидкість мережі тут також є фактором, продуктивність також залежить від того, скільки часу потрібно API для обробки запиту та надсилання відповіді.

Одним із найбільш впливових факторів продуктивності веб-сайту є час, який потрібно для завантаження початкових ресурсів. Як правило, що складніша програма, то більші ресурси потрібно завантажити.

Продуктивність веб-додатків на основі JavaScript постійно покращується завдяки прогресу у розвитку браузерних технологій,

оптимізаціям і підходам до розробки. На шляху до досягнення високої продуктивності важливо постійно оновлюватись з новими можливостями, використовувати ефективні практики програмування та уважно відстежувати зміни веб-стандартів. Це дозволить розробникам створювати швидкі та ефективні веб-застосунки.

2.10.1. Express.js

Express.js є гнучким та мінімалістичним фреймворком додатків. Він не побудований навколо конкретних компонентів, що дає розробникам можливість експериментувати. Вони отримують блискавичне налаштування та чистий досвід роботи з JS.

Express може бути використаний, для: багатосторінкових, односторінкових, гібридних додатків.

Характеристики Express.js:

- дозволяє розробникам швидше створювати RESTful API;
- швидка розробка на стороні сервера;
- підтримка NoSQL баз даних з коробки;
- підтримує архітектуру MVC;
- дозволяє динамічно відображати HTML-сторінки на основі передачі аргументів до шаблонів.

Серед його основних переваг можна чітко визначити наступні:

1. Express є вільним із відкритим вихідним кодом, пристосованим до ліцензії MIT, що робить його доступним для розробників безкоштовно. Це забезпечує економію фінансових ресурсів для інструментів та стимулює глобальну співпрацю у розвитку, прикладами якої є проекти, такі як SaaS Boilerplate та Builder Book.

2. Основою Express є мінімалістичний фреймворк, який сприяє високій продуктивності веб-додатків, що є критичним для забезпечення найкращого користувацького досвіду. Його експертно написаний код спрощує операції та

надає можливість подальшої оптимізації, такої як оптимізація зображень для поліпшення продуктивності та швидкості завантаження.

3. Виділяється його мінімалістичний та простий у використанні підхід, подібний до Node.js, який пропонує простоту та ефективність, роблячи процес розробки більш зручним та ефективним, що веде до заощадження часу розробників.

4. Express також відомий своєю здатністю ефективно створювати API, що важливо для розробників серверної частини, навіть тих, хто не володіє великим досвідом у JavaScript. Це забезпечує швидке нарощування та ефективну розробку серверної частини.

5. Express прискорює розробку додатків, надаючи готові функціональні можливості, бібліотеки та проміжне програмне забезпечення. Це допомагає у виконанні рутинних завдань, зменшує складність кодування та надає сучасні можливості для створення потужних, адаптивних додатків. Підтримка спільноти розробників надає безліч ресурсів для навчання та вирішення проблем у процесі розробки.

Надзвичайно важливим для сучасних сайтів є швидкість їхньої роботи, загрузки даних, опрацювання та вивід даних для користувача. Надсилання запитів до серверної частини через REST API виконується занадто довго, тому існують різні пошукові рушії, які спрямовані на пришвидшений пошук даних. Дані в собі вони містять за принципом нереляційних баз даних.

Express.js ідеально підходить для швидкого створення веб-додатків та сервісів, оскільки має доступні інструменти генерації API.

2.10.2. Vue.js

Vue.js – це прогресивний, легкий фреймворк для створення реактивних веб-інтерфейсів користувача. Vue розширює стандартні HTML і CSS, створюючи набір потужних інструментів для створення зовнішнього вигляду інтерактивних веб-додатків. Цей фреймворк, який підходить для створення

односторінкових додатків (Single-Page Applications) будь якої складності. Хоча Vue не пов'язаний із патерном MVVM (Model-View-ViewModel), його дизайн був частково натхненний ним. У Vue робота зосереджена на рівні ViewModel, забезпечуючи реактивну синхронізацію між моделлю та шарами перегляду через двостороннє зв’язування даних.

В основі Vue.js лежить реактивна система зв’язування даних, яка надзвичайно спрощує синхронізацію даних і DOM. Під час використання jQuery для ручного керування DOM код, який ми пишемо, часто є імперативним, повторюваним і схильним до помилок. Vue.js використовує концепцію представлення, керованого даними. Простими словами, це означає, що ми використовуємо спеціальний синтаксис у наших звичайних шаблонах HTML, щоб «прив’язати» DOM до базових даних.

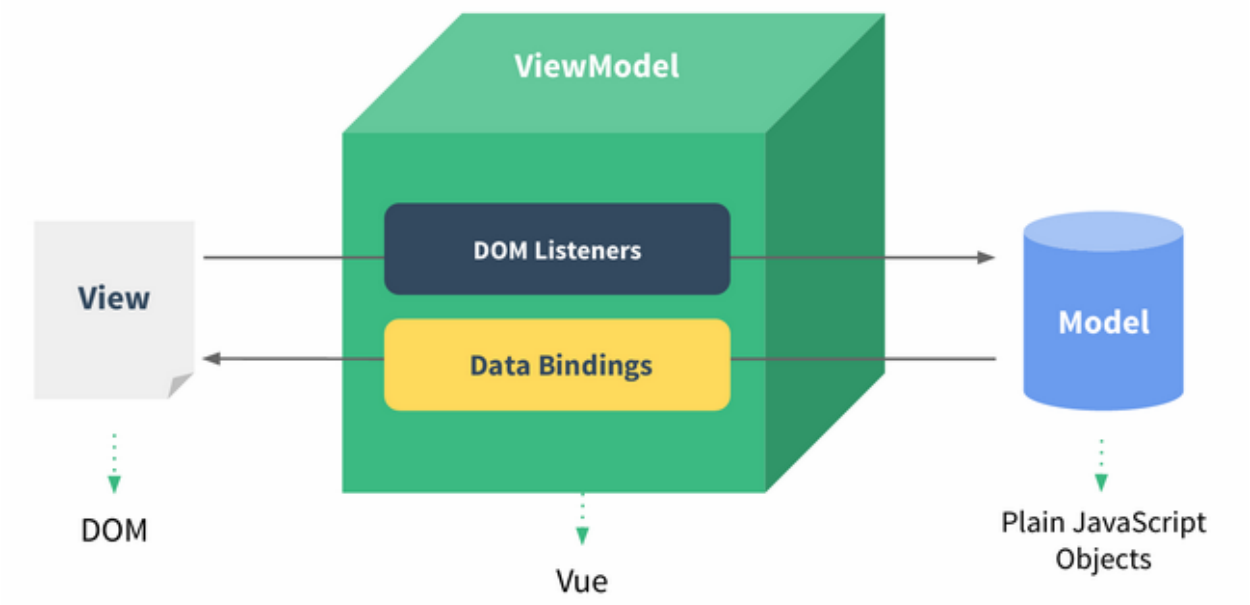


Рисунок 2.18 – Концепція Vue

Vue JS має 5 ключових переваг:

1. Vue JS, крім прогресивного підходу, відомий як "поступово адаптований". Це означає, що як сам фреймворк, так і програми, які його використовують, розробляються від початку. Однією з переваг цього підходу є легкість початку роботи. Основна бібліотека Vue JS базується на CSS,

HTML і JavaScript - основних складових будь-якого веб-проєкту веб-розробки.

2. Функціональні можливості, які можуть знадобитись у Vue JS для додаткового функціоналу, вже охоплені офіційними бібліотеками фреймворку. Після встановлення основ, ймовірно, буде необхідно налаштувати маршрутизацію на сайті і управління даними. Для досягнення цих цілей використовуються дві бібліотеки, які майже завжди присутні на сайтах, що використовують Vue JS: Vue Router і Vuex.

3. Легкий пакет Vue JS, розмір якого складає лише 21Кб, дозволяє йому працювати швидше, ніж його конкуренти. Це можливо завдяки його віртуальному DOM, який ефективно прискорює процес відображення сайту. Об'єктна модель документа (DOM) є інтерфейсом програмування, що моделює структуру тексту у HTML і XML. Віртуальний DOM, як реальне відображення, але із можливістю синхронізації змін, дозволяє оптимізувати роботу з ним.

4. Оволодіння Vue JS не потребує глибоких знань в області бібліотек та варіантів JavaScript. Тут ідеально підходять класичні CSS, HTML і JavaScript.

5. Vue JS, як і більшість мов та фреймворків з відкритим вихідним кодом, користується активною підтримкою та відданою спільнотою. Понад 132 000 учасників на GitHub використовують Vue JS.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ПЛАТФОРМИ ЗАВАНТАЖЕННЯ ТА ПЕРЕГЛЯДУ ВІДЕО ФАЙЛІВ

3.1. Розробка функціональних вимог до платформи

Розроблювана платформа повинна виконувати наступні функціональні вимоги:

− визначення інформації про відео, що було завантажено користувачем;

− редагування інформації про відео;

− відображення всіх відео користувача;

− завантаження на відеохостинг відео та метаданих, включаючи опис, заголовок, інформацію про автора;

− можливість користувача авторизуватися на сайті;

− можливість користувача реєструватися на сайті;

− можливість потокового перегляду відео;

− можливість користувача переглядати відео інших користувачів без авторизації;

− можливість користувача ставити оцінку відео;

− можливість рахування кількості переглядів відео;

− можливість користувача оновлювати інформацію з власного профілю.

Користувачі програми поділяються лише на дві ролі: авторизовані та неавторизовані.

Діаграми прецедентів (use case diagrams) є частиною методології UML (Unified Modeling Language) і використовуються під час розробки програмного забезпечення для візуалізації функціональних вимог до системи.

Діаграми прецедентів допомагають у плануванні та розробці ПЗ, надаючи чітке уявлення про те, як користувачі будуть взаємодіяти з

системою, і дозволяють розробникам зосередитися на задоволенні цих взаємодій.

Сценарії використання програми авторизованим користувачем представлені на рис. 3.1.

Сценарії використання програми авторизованим користувачем, що включають сценарії реєстрації користувача, відображення особистого профілю, відображення основної інформації про відео, а також перегляд списку власних відео, є альтернативними.

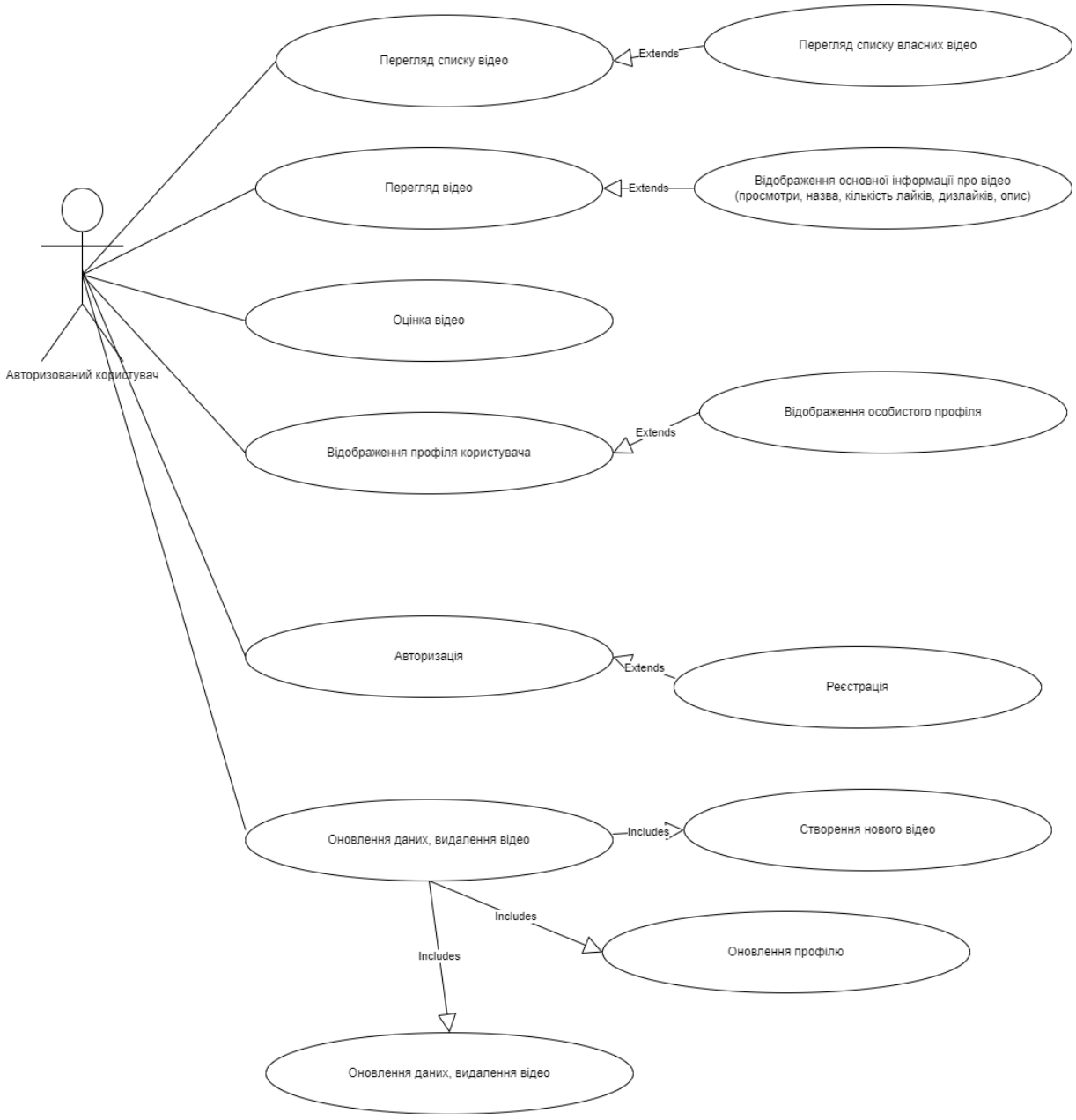


Рисунок 3.1 – Діаграма прецедентів (для авторизованого користувача)

Неавторизований користувач має схожі, але більш обмежені можливості, тобто обмежений набір сценаріїв. Це сценарії перегляду відео, перегляду списку відео, а також переходу до особистого профілю користувача.

Сценарії використання програми неавторизованим користувачем представлені на рис. 3.2.

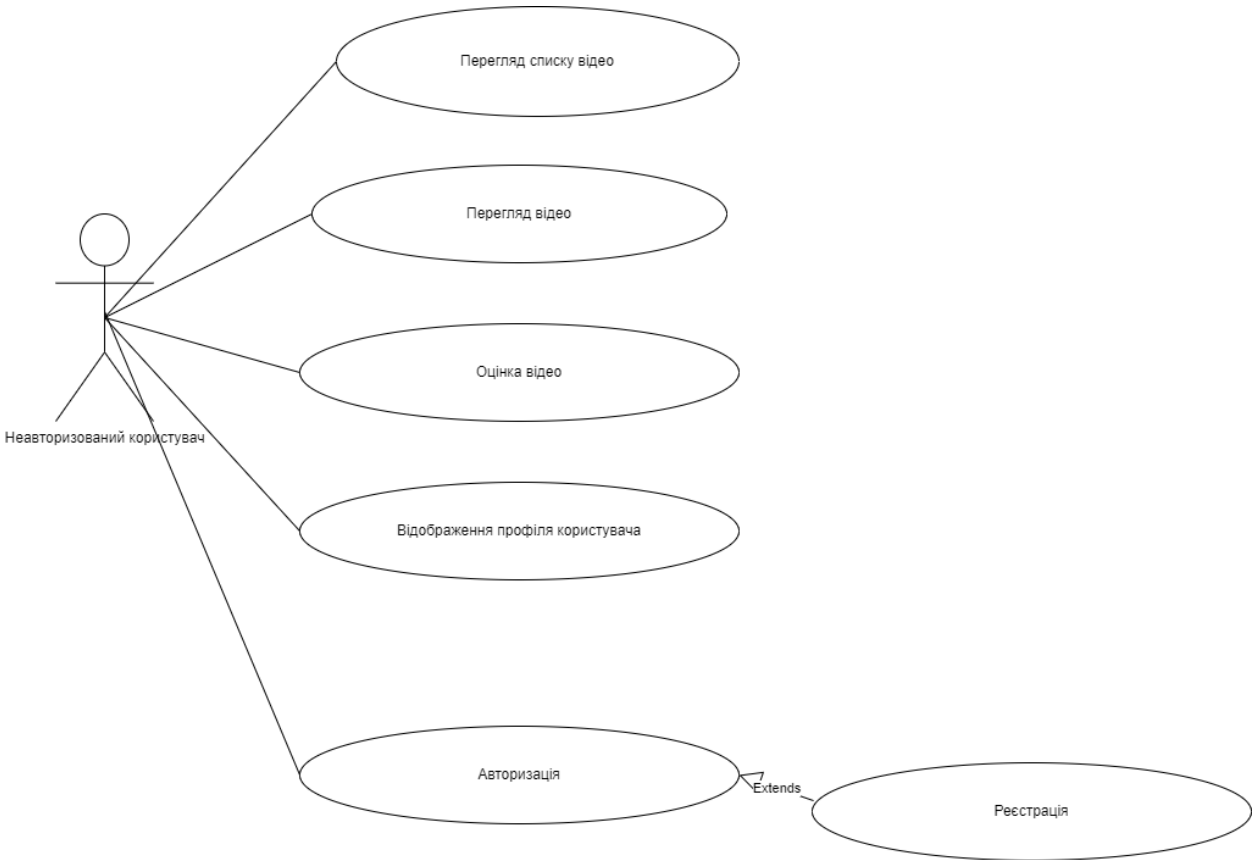


Рисунок 3.2 – Діаграма прецедентів (для неавторизованого користувача)

3.2. Розробка структури back-end частини платформи

Слід зазначити, що програма ділиться на клієнтську частину – front-end, та серверну частину – back-end.

До структури серверної частини додатку відносяться:

− сервіси – частини додатку, задачею яких є пряма взаємодія з СКБД, через які будуть отримані дані з відповідних до сервісів таблиць;

− моделі – описані у коді сутності, що містять основні їх характеристики й колонки таблиць;

− контролери – частини додатку, що представляють собою основний функціонал і виконують складні операції, зв’язані з логікою, при взаємодії сервісів;

− міддлвар – як і у випадку з клієнтською логікою відповідає за логіку у випадку, якщо користувач не є авторизованим.

Структура серверної частини додатку представлена відповідною схемою на рис. 3.3.

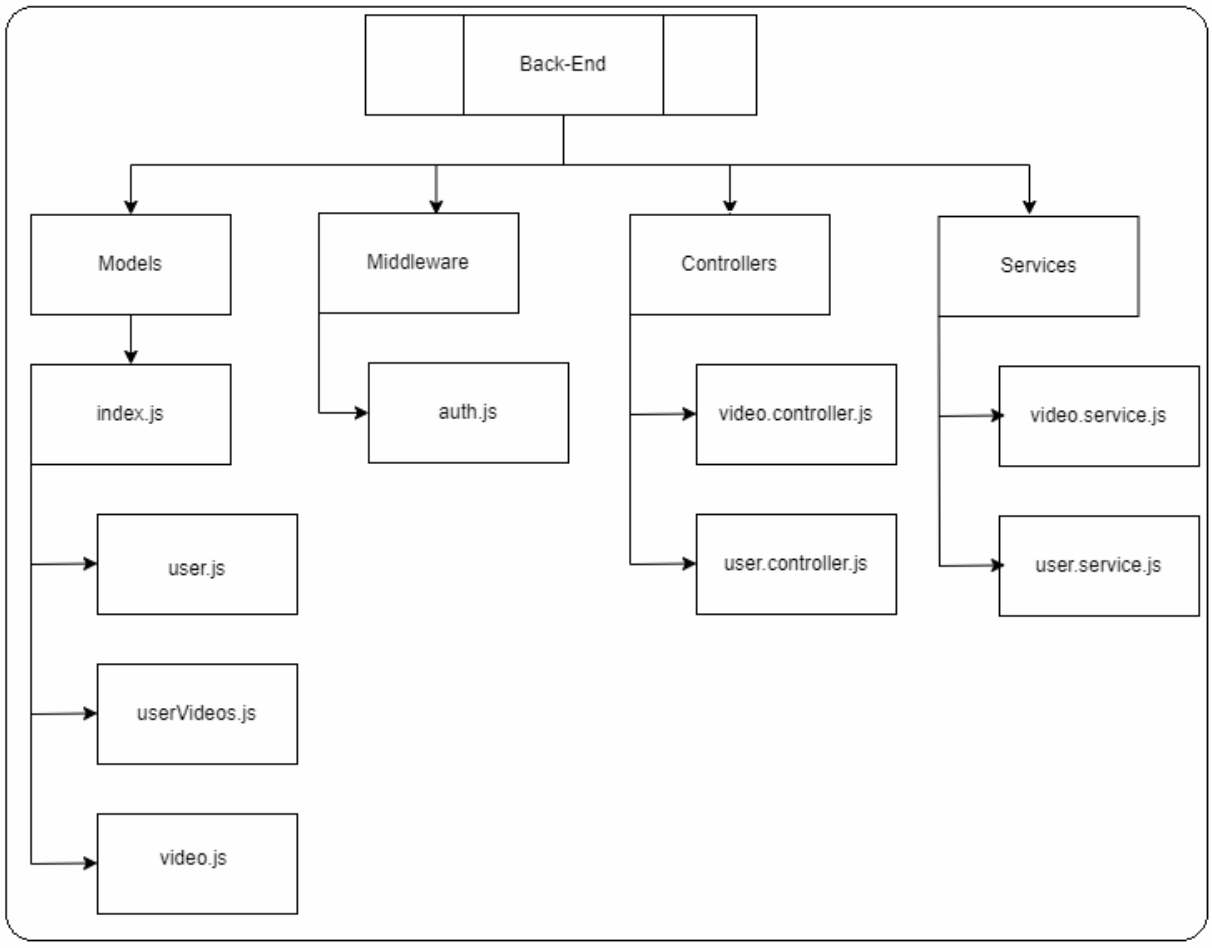


Рисунок 3.3 – Структура back-end частини платформи

До сервісів відносяться:

− video.service.js – сервіс, що відповідає за взаємодію з моделлю Video;

− user.service.js – сервіс, що відповідає за взаємодію з моделлю User.

До моделей відносяться:

− video.js – файл, в якому описана модель Video, а також допоміжні хуки при утворенні;

− user.js – файл, у якому описана модель User;

− userVideos.js – файл, у якому описана модель таблиці UserVideoMap;

− index.js – головний файл, що несе за мету імпорт моделей до основної частини.

До контролерів відносяться:

− video.controller.js – контролер, що відповідає за логічні операції, пов’язані з відео файлами;

− user.controller.js – контролер, що відповідає за логічні операції, пов’язані з користувачами.

Міддлваром виступає файл auth.js, завдання якого є обробка запитів, коли користувач не є авторизованим

3.3. Розробка структури front-end частини платформи

До структури front-end (клієнтської) частини відносяться:

− представлення – динамічні веб-сторінки, що носять в собі основну розмітку, стилізацію, а також деяку взаємодію з бізнес-логікою;

− файли сховища – файли, що є частинами глобального сховища Vuex, які виконують бізнес-логіку і відповідають за збереження загальної інформації про сайт, яку бачить користувач, що користується сайтом, тобто зберігають дані про користувача, відео, списки відео та іншу інформацію, що потрібна під час сесії. Файли сховища є частиною підходу SPA та Flux;

− компоненти – частини платформи, призначені для повторного використання, що винесені у окремий елемент задля оптимізації сайту та скорочення написаного коду у процесі;

− міддлвар – файл, що відповідає за перенаправлення користувача з тих чи інших сторінок, якщо він не є авторизованим.

Структура front-end частини додатку представлена відповідною схемою на рис. 3.4.

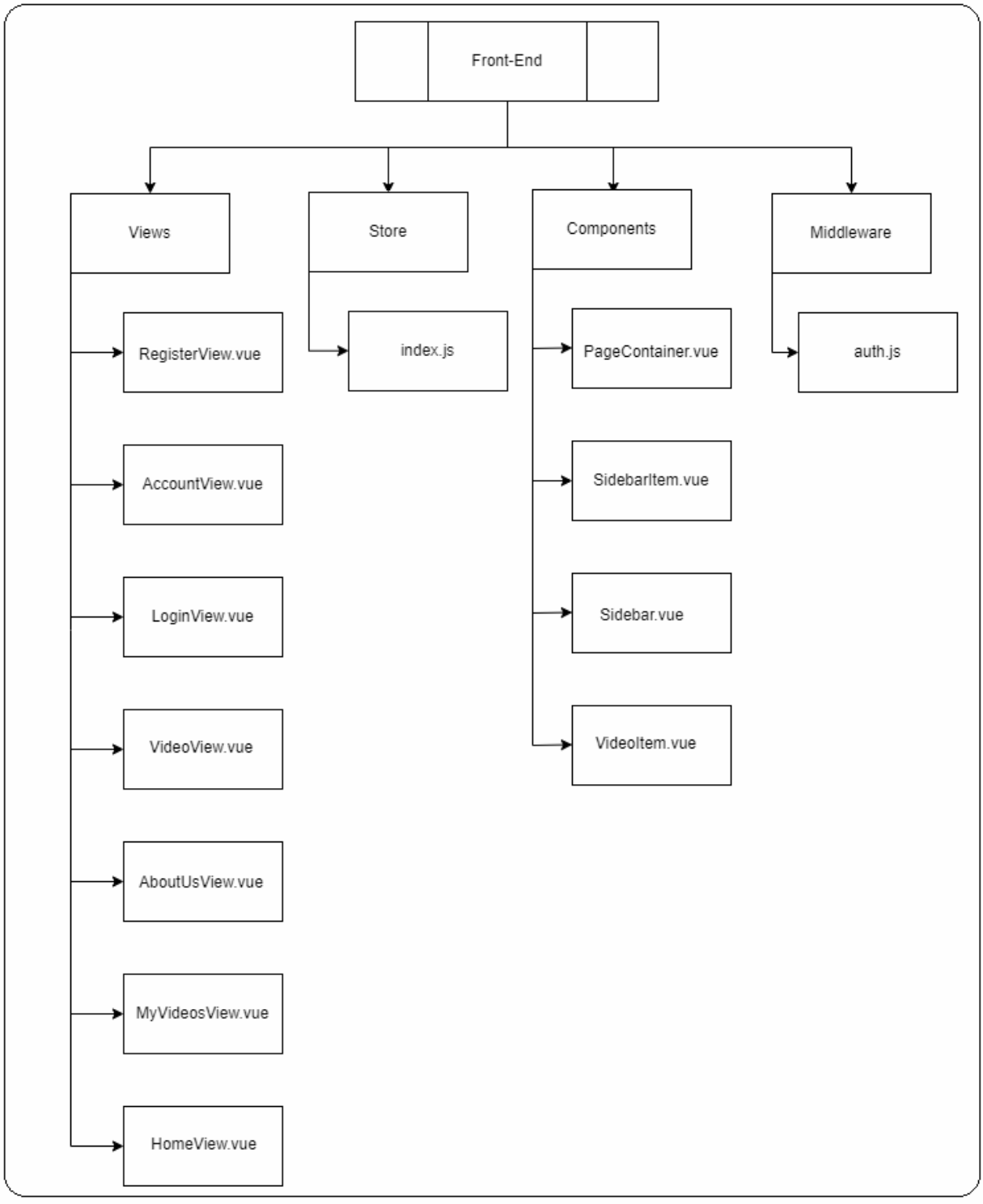


Рисунок 3.4 – Структура front-end частини розроблюваної платформи

До представлень, що були створені під час розробки платформи у папці views відносяться:

− RegisterView.vue – сторінка, на якій користувач має можливість зареєструватися, якщо не має особистого облікового запису;

− AboutUsView.vue – сторінка, що має загальний опис призначення платформи;

− AccountView.vue – сторінка, що має основну інформацію про обліковий запис користувача;

− VideoView.vue – сторінка, на якій користувач може подивитися відео, а також загальну інформацію про нього.

− HomeView.vue – сторінка, що відображається першою, тобто є домашньою, і показує користувачу, як авторизованому так і ні, список актуальних відео;

− LoginView.vue – сторінка, через яку користувач може пройти авторизацію, а також перейти до реєстрації;

− MyVideosView.vue – сторінка, на якій користувач з обліковим записом може побачити свої відео, а також завантажити своє нове відео;

До файлів сховища у папці store відноситься тільки index.js, що містить основні функції бізнес-логіки, пов’язані з відео та користувачем.

До компонентів у папці components відносяться:

− Sidebar.vue – компонент бокового меню, що містить основні кнопки для навігації по сайту;

− SidebarItem.vue – компонент кнопки бокового меню;

− PageContainer.vue – контейнер сторінки, який додає до нього основні стилі;

− VideoItem.vue – компонент, що містить плеєр, через який відтворюється відео.

Мідлваром, що зберігається у папці middleware, є файл auth.js, що відповідає за поведінку додатку, якщо користувач не є авторизованим.

3.4. Розробка інтерфейсу платформи перегляду відео файлів

Програмна платформа має один інтерфейс, реалізований у вигляді сайту, на який можна зайти, написавши адресу сайту у браузері, в нашому випадку, сайт працює локально і має адресу localhost:3000.

Першим, що бачить користувач є домашня сторінка, що зображена на рисунку 3.5.

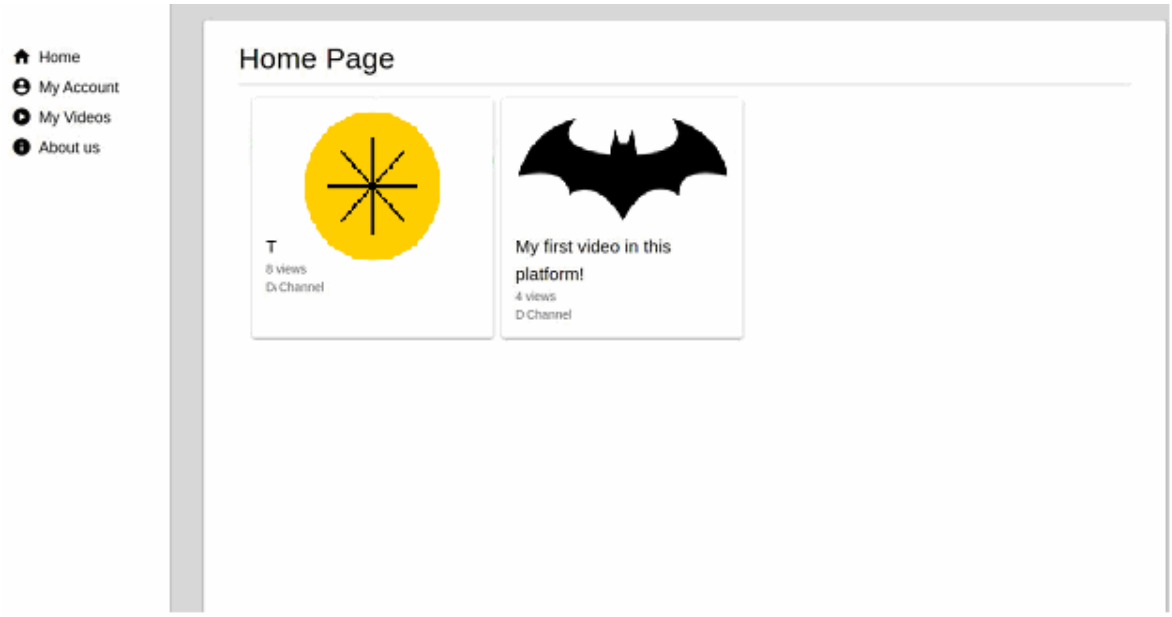


Рисунок 3.5 – Головна (домашня) сторінка розробленої платформи

Звідти, завдяки боковому меню (рис. 3.6) у користувача є вибір, куди саме піти у наступному, серед варіантів є:

− залишитися на сторінці, тобто на вкладці “Home”;

− перейти до свого облікового запису через вкладку “My account”, або ж перейти через неї на сторінку логіну;

− перейти до своїх особистих відео через вкладку “My videos”. У разі, якщо користувач неавторизований, алгоритм є таким же, як і у минулому пункті;

− почитати про основні цілі і ціль роботи у вкладці “About us”.

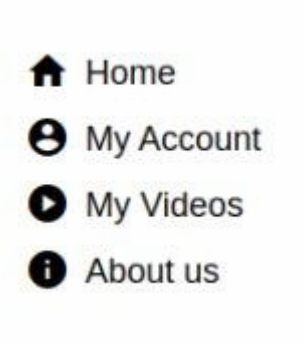


Рисунок 3.6 – Елементи бокового меню

Для подальшої взаємодії з основним функціоналом розробленої платформи користувач має скористуватись сторінкою авторизації, яку зображено на рисунку 3.7. На цій сторінці користувач буде мати можливість авторизуватися, тобто ввести свою електрону пошту та свій пароль, якщо він вже має обліковий запис. У разі, якщо користувач ще не має свого акаунту, то він може перейти до сторінки реєстрації через сіру кнопку “Create new account”.

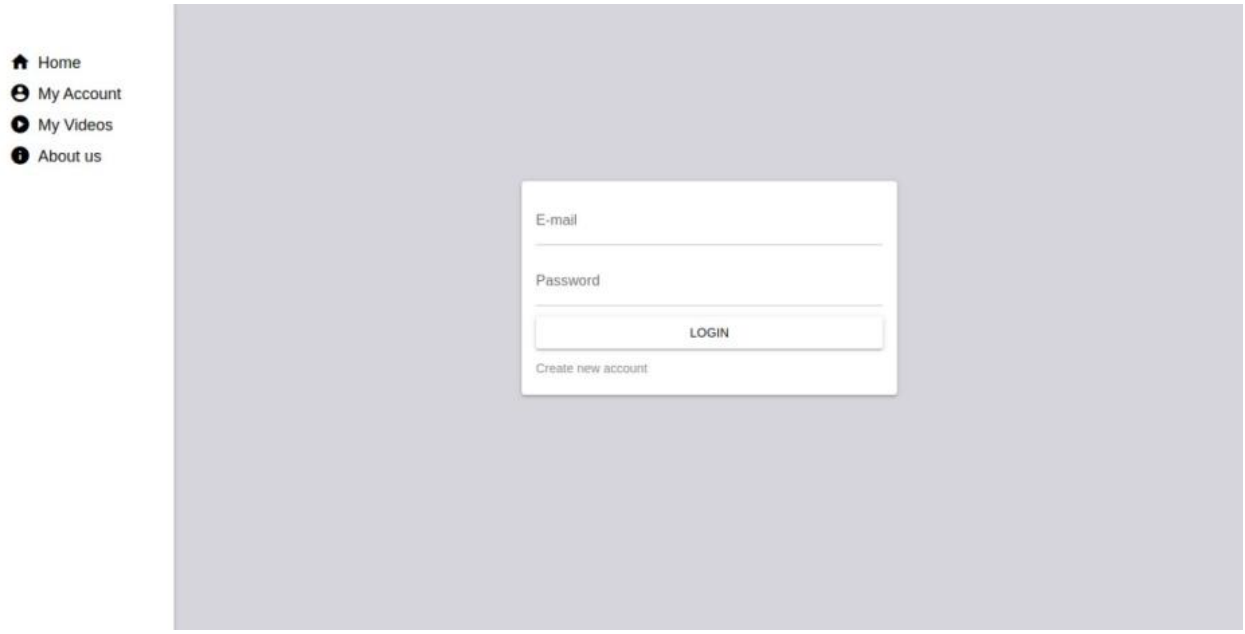


Рисунок 3.7 – Вигляд сторінки авторизації

Сторінку реєстрації зображено на рисунку 3.8.

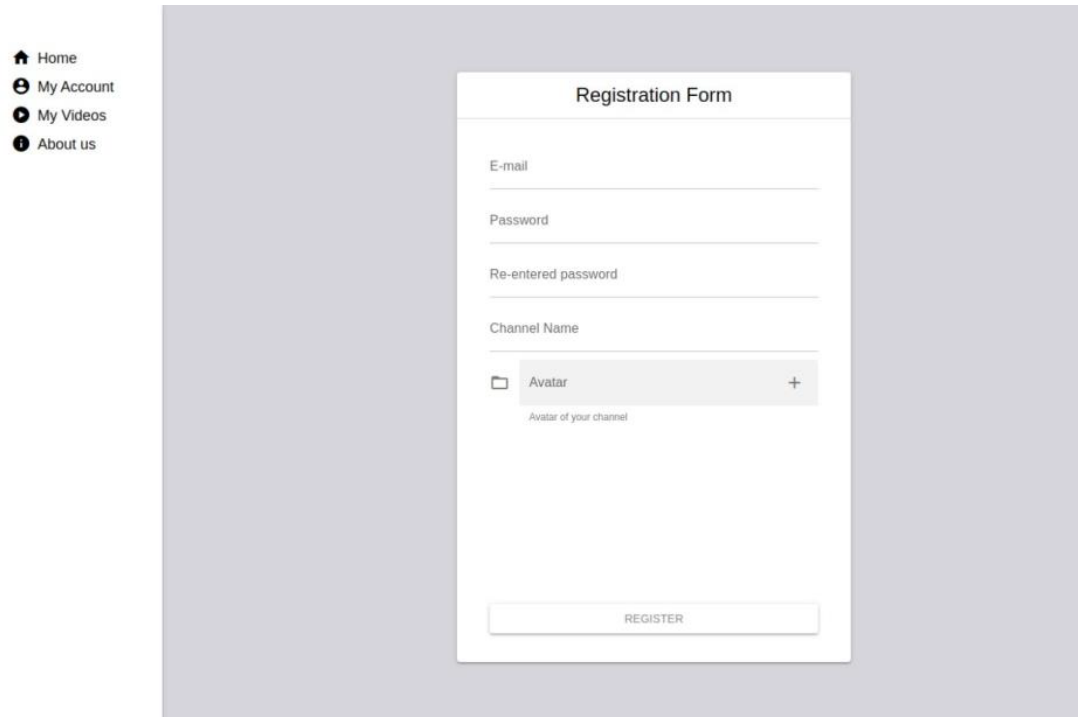


Рисунок 3.8 – Вигляд сторінки реєстрації

На сторінці реєстрації користувач повинен ввести свій пароль, підтвердити його, а також написати майбутню назву особистого каналу, завантажити аватар, що буде зображений у його профілі і свою пошту. Після реєстрації користувача буде перенаправлено на сторінку особистого акаунту, що зображена на рис. 3.9.

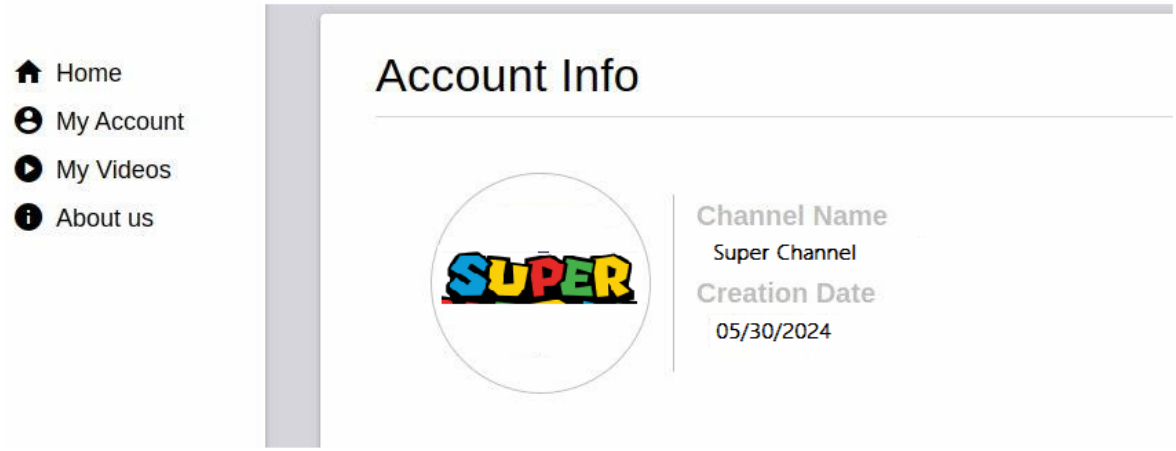


Рисунок 3.9 – Інформація про користувача (акаунт)

Тепер, коли користувач авторизований, то він може перейти на сторінку особистих відео, що буде пуста, якщо користувач ще не має своїх відео. Приклад сторінки відео зображено на рис. 3.10.

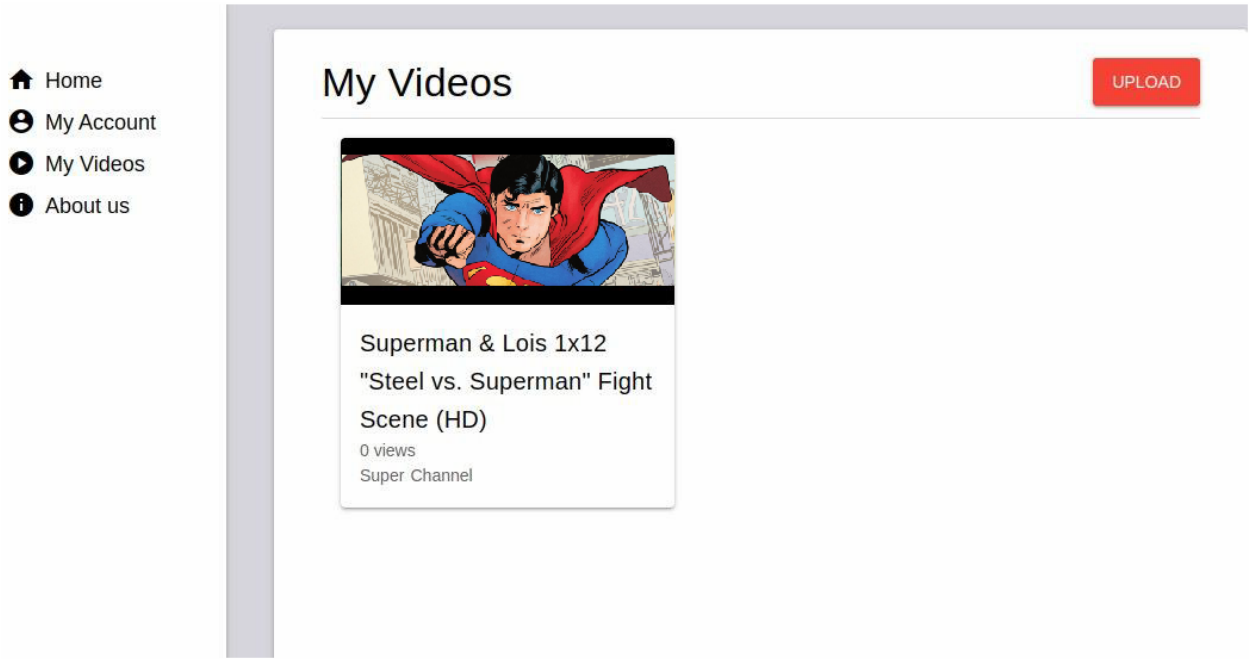


Рисунок 3.10 – Вигляд сторінки із завантаженими відео користувача

Окрім того, якщо користувач хоче завантажити нове власне відео, то він буде мати таку можливість, натиснувши на кнопку “Upload” (рис. 3.11).



Рисунок 3.11 – Кнопка для завантаження відео файлів

Користувач побачить модальне вікно, що зображено на рис. 3.11, де він повинен заповнити усі дані про відео, тобто назву, опис, заставку, а також завантажити саме відео.

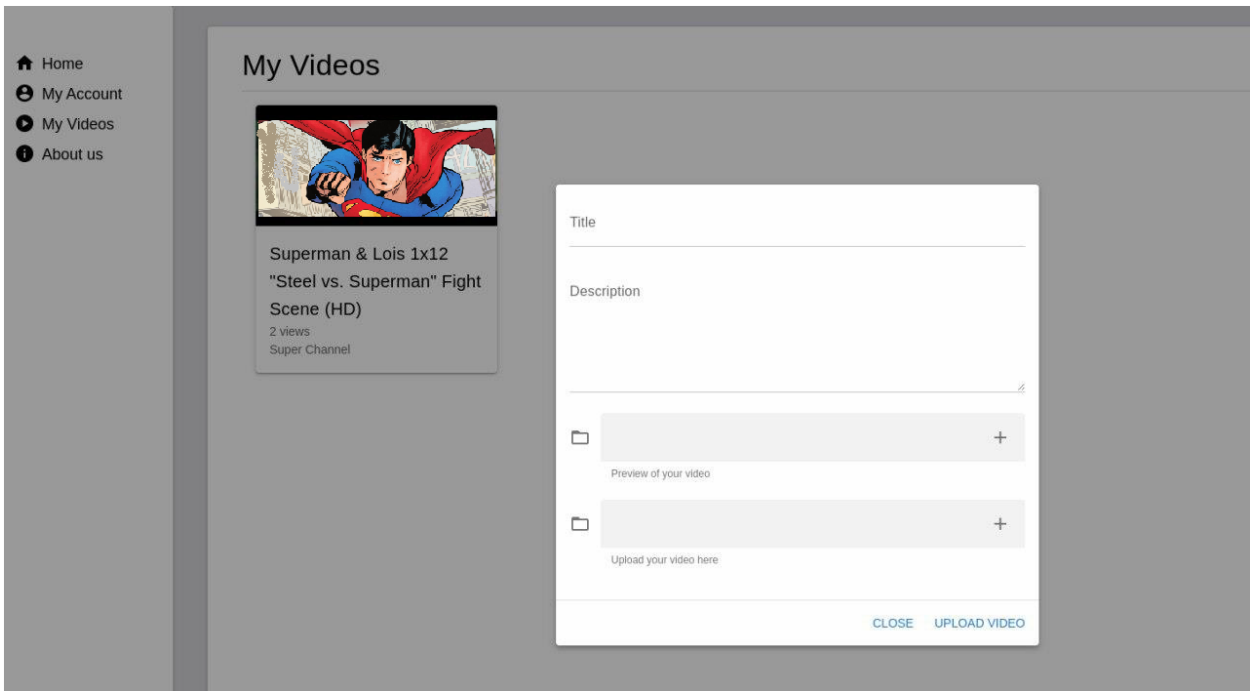


Рисунок 3.12 – Вікно для завантаження відео

Користувач, як авторизований, так і ні, має можливість переглядати відео, для чого він повинен натиснути на прев'ю. При натисканні на прев'ю відкриється сторінка перегляду, що зображена на рис. 3.13.

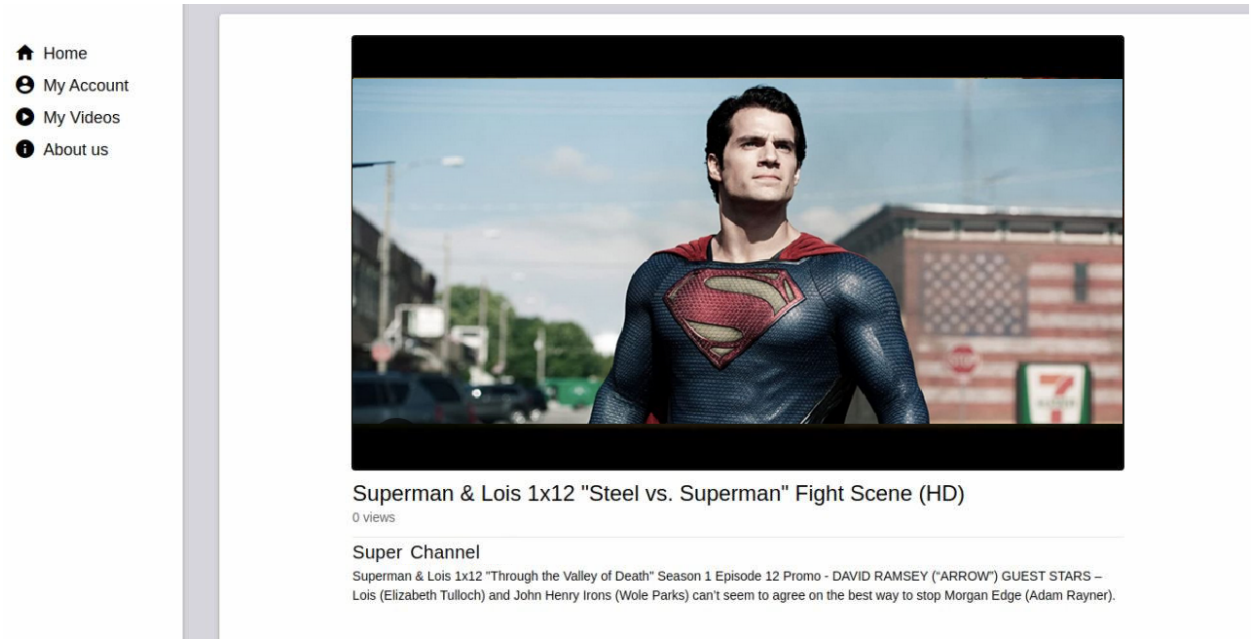


Рисунок 3.13 – Вигляд сторінки перегляду завантаженого відео

ВИСНОВКИ

В дипломній роботі виконана реалізація програмного забезпечення платформи завантаження та перегляду відео файлів. Для розробки було застосовано структурне, функціональне та об’єктно-орієнтоване програмування, фреймворки Vue.js і Express.js, мова програмування JavaScript.

Розроблено програмне забезпечення для завантаження та перегляду відео в мережі Інтернет та реалізована база даних для зберігання метаданих відео. Було розглянуто можливість побудови архітектури веб-додатків та взаємодію структурних компонентів між собою. Обрано та обгрунтовано вибір платформи для розробки.

Розроблено зручний веб-сервіс, що надає можливість користувачеві переглядати знайдене на сайті відео без додаткового завантаження будь-яких додатків. Досліджено особливості функціонування клієнт-серверної архітектури, що дозволило обґрунтувати доцільність використання для розробки бібліотек мови програмування JavaScript, що забезпечило високу швидкість, простоту та масштабованість проекту. Проаналізовано типи клієнт серверних архітектури, виявлено їх переваги та недоліки. Розроблено алгоритм додавання нових відео до платформи.

Розроблене програмне забезпечення забезпечує користувачам можливість перегляду списків всіх наявних на платформі відео, перегляду відеофайлу безпосередньо в браузері.

Розроблена платформа перегляду відео файлів має безліч переваг:

− адаптивність;

− використовуються практики використовуваності;

− використовуються практики веб-доступності;

– наповненість.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Tripathi, A., & Vasudeva, M. (2020). "TikTok App: A Review of Content, Features, and Privacy Issues." In 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE.
2. Video Streaming Market [Електронний ресурс]: https://www.precedenceresearch.com/video-streaming-market
3. Davidson, D. (2020). "Vimeo and TikTok: Exploring Alternative Video-Sharing Platforms." In "Social Media and the Transformation of Interaction in Society" (pp. 113-135). IGI Global.
4. Netflix accessibility features URL: https://www.makeuseof.com/how-to-use-netflix-accessibility-features/
5. Martindale, M., & Durham, M. G. (2018). "The Vimeo Toolbox: Strategies for Creating Quality Videos." Journal of Electronic Resources in Medical Libraries, 15(3-4), 180-189
6. Hesselberth, P. (2019). "Beyond YouTube: Vimeo as a Platform for Video Activism." In "Vimeo Video School: A Storytelling Guide to Filmmaking" (pp. 131-142). Routledge.
7. Burgess, J., & Green, J. (2013). "YouTube and the Mainstream Media." In "YouTube: Online Video and Participatory Culture" (pp. 39-58). John Wiley & Sons.
8. Gill, P., & Kaur, S. (2020). "A comparative study of video-sharing platforms: YouTube, Vimeo, and Dailymotion." In 2020 International Conference on Intelligent Sustainable Systems (ICISS) (pp. 1252-1257). IEEE.
9. Zhu, C. Streaming Media Architectures, Techniques, and Applications: Recent Advances: Recent Advances / C. Zhu, Y. Li, X. Niu. – 1st edition – IGI Global, 2010. – 502 p.

10. Cannell, S. YouTube Secrets: The Ultimate Guide to Growing Your Following and Making Money as a Video Influencer / S. Cannell, B. Travis. – Lioncrest Publishing, 2018. – 272 p. 10. Prensa. Youtube [Electronic resource]. – Access mode: https://about.youtube/.
11. Основні переваги СУБД MySQL [Електронний ресурс] // НТУ. – 2018: https://studfile.net/preview/5607354/page:3/.
12. Москаленко А. В. Програмне середовище для веб розробки. [Електронний ресурс] / А. В. Москаленко // OSPanel.io. – 2022: https://ospanel.io/.
13. 5 social media videos to help you build a loyal audience [Electronic resource]. – Access mode: https://vimeo.com/blog/post/social-media-videos-build-audience/.
14. 9GAG: why age is more than just a number when it comes to staying relevant [Electronic resource]. – Access mode: https://www.scmp.com/business/companies/article/2052601/9gag-why-age-more-just-number-when-it-comes-staying-relevant.
15. AMV Codec Tools [Electronic resource]. – Access mode: https://code.google.com/archive/p/amv-codec-tools/wikis/AmvDocumentation.wiki.
16. QuickTime File Format [Electronic resource]. – Access mode: https://www.loc.gov/preservation/digital/formats/fdd/fdd000052.shtml.
17. Fernando C. N. Pereira. The MPEG-4 Book / F. Pereira. – Prentice Hall, 2002. – 392p.
18. MPEG standards [Electronic resource]. – Access mode: https://web.archive.org/web/20100420192552/http://mpeg.chiariglione.org/standards.htm.
19. Programme of Work [Electronic resource]. – Access mode: https://web.archive.org/web/20131231054907/http://kikaku.itscj.ipsj.or.jp/sc29/29w42911.htm.

20. About WebM [Electronic resource]. – Access mode: https://www.webmproject.org/about/.
21. Що таке трирівнева архітектура? / Розвиток. – 2022.: https://uk.theastrologypage.com/three-tier-architecture.
22. Павлов А. В. Огляд підходів до організації інтелектуального інтерфейсу користувача / А. В. Павлов // МАН. – 2019: http://www.mgua.irtc.org.ua/attach/IMCS/2019/10.pdf.
23. SPA (Single-page application) [Electronic resource]. – Access mode: https://developer.mozilla.org/en-US/docs/Glossary/SPA.
24. Single Page Application implementation [Electronic resource]. – Access mode: https://experienceleague.adobe.com/docs/target/using/implement-target/client-side/at-js-implementation/deploy-at-js/target-atjs-single-page-application.html?lang=en.
25. Express [Electronic resource]. – Access mode: https://expressjs.com/.
26. Getting Started with MySQL [Electronic resource]. – Access mode: https://dev.mysql.com/doc/mysql-getting-started/en/.
27. Redis [Electronic resource]. – Access mode: https://aws.amazon.com/ru/redis/.
28. What is Flux? [Electronic resource]. – Access mode: http://fluxxor.com/what-is-flux.html.

ДОДАТКИ

Додаток А

Фрагменти лістингів коду

User.js

```
"use strict";
const { Model } = require("sequelize");
class User extends Model {
static init(sequelize, DataTypes) {
return super.init(
{
nickname: DataTypes.STRING,
password: DataTypes.STRING,
email: {
type: DataTypes.STRING,
validate: { isEmail: true },
},
avatar: DataTypes.TEXT,
},
{
sequelize,
modelName: "User",
}
);
}
static associate(models) {
this.hasOne(models.userVideosMap, {
foreignKey: "userId",
targetKey: "id",
as: "userLink",
});
}
}
module.exports = User;
```

userVideos.js

```
"use strict";
const { Model } = require("sequelize");
class UserVideos extends Model {
static init(sequelize, DataTypes) {
return super.init(
{
userId: DataTypes.INTEGER,
videoId: DataTypes.INTEGER,
},
{
timestamps: false,
sequelize,
modelName: "userVideosMap",
}
);
}
static associate(models) {
this.belongsTo(models.User, {
foreignKey: "userId",
targetKey: "id",
as: "userLink",
});
}
}
module.exports = UserVideos;
```

user.controller.js

```
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const router = require("express").Router();
const asyncHandler = require("express-async-handler");
const UserService = require("../services/user.service");
const { Op } = require("sequelize");
const redis_Client = require("../redis_connect");
```

```
const auth = require("../middleware/auth.middleware");
process.env.SECRET_KEY = "secret";
process.env.REFRESH_KEY = "refresh_diploma";
create = async (req, res) => {
const userData = {
nickname: req.body.nickname,
password: req.body.password,
email: req.body.email,
avatar: req.body.avatar,
};
const user = await UserService.userFind(userData.email);
if (!user) {
bcrypt.hash(req.body.password, 11, async (err, hash) => {
userData.password = hash;
const data = await UserService.userCreate(userData);
if (data) {
res.status(201).json({ message: "User is successfully created"
});
} else {
res.status(500).json({ error: "Can't create object" });
}
});
} else res.status(400).send({ error: "User is already exists"
});
};
login = async (req, res) => {
const user = await UserService.userFind(req.body.email);
if (user) {
if (await bcrypt.compare(req.body.password, user.password)) {
let token = jwt.sign(user.dataValues, process.env.SECRET_KEY, {
expiresIn: 300,
});
let refreshToken = jwt.sign(user.dataValues,
process.env.REFRESH_KEY, {
expiresIn: 2592000,
```

```
});
redis_client.rpush(
["UserID" + user.dataValues.id, refreshToken],
function (err) {
if (err) {
console.log(err);
} else {
console.log("Token is succesfully added!");
}
}
);
res.status(201).json({ token, refreshToken });
} else {
res.status(403).json({ error: "Incorrect password" });
}
} else {
res.status(404).json({ error: "User not found" });
}
};
getNewToken = async (req, res) => {
if (req.headers.refreshtoken) {
const refreshToken = req.headers.refreshtoken;
const decoded = await jwt.verify(refreshToken,
"refresh_diploma");
if (Date.now < decoded.exp * 1000) {
res.status(406).json({ message: "Token is expired!" });
redis_client.lrem("UserID" + decoded.id, 0, refreshToken,
function (err) {
if (err) {
console.log(err);
} else {
console.log("Token is succesfully removed!");
}
});
}
```

```
redis_client.lrange(
"UserID" + decoded.id,
0,
100,
function (err, response) {
if (response.includes(refreshToken)) {
["exp", "iat"].forEach((e) => delete decoded[e]);
let token = jwt.sign(decoded, process.env.SECRET_KEY, {
expiresIn: 300,
});
res.status(201).json({ token });
} else {
res.status(410).json({ message: "Token is not exist in store"
});
}
}
);
} else {
res.status(403).json({ error: "Forbidden refresh" });
}
};
find = async (req, res) => {
const where = {};
if (req.user.email) {
where.email = { [Op.substring]: req.query.email };
}
const data = await UserService.findByEmail(where);
res.status(200).json(req.user);
};
logout = async (req, res) => {
const refreshToken = req.headers.refreshtoken;
const decoded = await jwt.verify(refreshToken,
"refresh_diploma");
redis_client.lrem("UserID" + decoded.id, 0, refreshToken,
function (err) {
```

```
if (err) {
console.log(err);
} else {
console.log("Token is succesfully removed!");
}
});
res.status(200).json({ message: "Success" });
};
module.exports = (app) => {
router.post("/signup", asyncHandler(create));
router.post("/login", asyncHandler(login));
router.get("/get/", auth, asyncHandler(find));
router.get("/token/", asyncHandler(getNewToken));
router.get("/logout/", asyncHandler(logout));
app.use("/api/user", router);
};
```

video.controller.js

```
const asyncHandler = require("express-async-handler");
const router = require("express").Router();
const auth = require("../middleware/auth.middleware");
const VideoService = require("../services/video.service");
const multer = require("multer");
const fs = require("fs");
async function createVideo(req, res) {
const video = {
title: req.body.title,
description: req.body.description,
viewCounter: 0,
screenSaver: req.body.screenSaver,
};
const newVideo = await VideoService.create(video, req.user.id);
res.status(200).json(newVideo);
}
async function onVideoUpload(req, res) {
```

```
const id = req.params.id;
const body = { fileHashNumber: req.file.filename };
await VideoService.update(body, id);
res.status(201);
}
const MAX_SIZE = 200000000;
const videoStorage = multer.diskStorage({
destination: "./videos/",
filename: (req, file, cb) => {
cb(null, file.fieldname + "_" + Date.now() + ".mp4");
},
});
const videoUpload = multer({
storage: videoStorage,
limits: {
fileSize: MAX_SIZE,
},
fileFilter(req, file, cb) {
if (!file.originalname.match(/\.(mp4|MPEG-4|mkv)$/)) {
return cb(new Error("Please upload a video"));
}
cb(undefined, true);
},
});
async function getVideoData(req, res) {
const data = await VideoService.findOne(req.params.id);
return res.status(200).json(data);
}
async function getVideoList(req, res) {
const data = await VideoService.findAll();
return res.status(200).json(data);
}
async function getMyVideoList(req, res) {
const data = await VideoService.getMy(req.user.email);
return res.status(200).json(data);
```

```
}
async function updateViews(req, res) {
await VideoService.update({ viewCounter: req.body.views },
req.params.id);
return res.status(200);
}
module.exports = (app) => {
router.post("/", auth, asyncHandler(createVideo));
router.post(
"/upload/:id",
auth,
videoUpload.single("video"),
onVideoUpload,
(error, req, res, next) => {
res.status(400).send({ error: error.message });
}
);
router.get("/video/:id", (req, res) => {
const videoPath = `./videos/${req.params.id}.mp4`;
const videoStat = fs.statSync(videoPath);
const fileSize = videoStat.size;
const videoRange = req.headers.range;
if (videoRange) {
const parts = videoRange.replace(/bytes=/, "").split("-");
const start = parseInt(parts[0], 10);
const end = parts[1] ? parseInt(parts[1], 10) : fileSize – 1;
const chunksize = end – start + 1;
const file = fs.createReadStream(videoPath, { start, end });
const head = {
"Content-Range": `bytes ${start}-${end}/${fileSize}`,
"Accept-Ranges": "bytes",
"Content-Length": chunksize,
"Content-Type": "video/mp4",
};
res.writeHead(206, head);
```

```
file.pipe(res);
} else {
const head = {
"Content-Length": fileSize,
"Content-Type": "video/mp4",
};
res.writeHead(200, head);
fs.createReadStream(videoPath).pipe(res);
}
});
router.get("/data/:id", asyncHandler(getVideoData));
router.get("/list", asyncHandler(getVideoList));
router.get("/my-list", auth, asyncHandler(getMyVideoList));
router.patch("/update-views/:id", asyncHandler(updateViews));
app.use("/api/video", router);
};
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “ Реалізація програмного забезпечення платформи завантаження та перегляду відео файлів ”

Обсяг пояснювальної записки: 76 аркушів.

Дата закінчення роботи: 17 червня 2024 р.

Підпис студента _____