

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ СЕРВІСУ ВІДЕОХОСТИНГУ
SOFTWARE IMPLEMENTATION OF THE VIDEO HOSTING SERVICE

Виконав: студент 4 курсу, групи КНТ-118
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Мінібаєв Д.О.

(прізвище та ініціали)

Керівник Колпакова Т.О.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Мінібаєва Даміра Олеговича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація сервісу відеохостингу
Software Implementation of the Video Hosting Service

керівник проєкту (роботи) Колпакова Тетяна Олексіївна, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “27” квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) 07 червня 2022 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне
завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Опис програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Колпакова Т.О., доцент		
Нормоконтроль	Липовець М.В., асистент		

7. Дата видачі завдання « 04 » квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)


(підпис)

Мінібаєв Д.О.

(прізвище та ініціали)

Керівник проєкту (роботи)


(підпис)

Колпакова Т.О.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 80 с., 31 рис., 6 табл., 3 дод., 29 джерел.

ВІДЕХОСТИНГ, ВІДЕО, СТРИМІНГ, VUE.JS, EXPRESS.JS, JAVASCRIPT.

Об'єкт дослідження – процес розповсюдження відео в мережі Інтернет.

Предмет дослідження – програмне забезпечення для завантаження та перегляду відео.

Мета роботи – створення програмного забезпечення для хостингу відео, що забезпечує завантаження, перегляд та взаємодію користувачів з відео файлами.

Матеріали, методи та технічні засоби: структурне, функціональне та об'єктно-орієнтоване програмування, фреймворки Vue.js і Express.js, мови програмування JavaScript, персональний комп'ютер з процесором Ryzen 5 3500x під управлінням операційної системи Microsoft Windows 10.

Результати. Розроблено програмне забезпечення для завантаження та перегляду відео в мережі Інтернет. Створено програмний вебзастосунок, реалізований на платформах Vue.js та Express.js. Реалізована база даних для зберігання метаданих відео.

Висновки. Розроблене програмне забезпечення забезпечує користувачам можливість авторизації і реєстрації, перегляду списків всіх наявних на платформі відео, перегляду відеофайлу безпосередньо в браузері, та створення власних відео.

Галузь використання – подібну програму можна використовувати звичайним користувачам для обміну відео контентом.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 80 pages, 31 figures, 6 tables, 3 appendixes, 29 sources.

VIDEO HOSTING SITE, VIDEO, STREAMING, VUE.JS, EXPRESS.JS, JAVASCRIPT.

Object of study is the process of video distribution on the Internet

Subject of study are video download and viewing software..

The purpose of the work is creation of software for video hosting, which provides downloading, viewing and interaction of users with video files.

Materials, methods, and tools: structural, functional and object-oriented programming, Vue.js and Express.js frameworks, JavaScript programming languages, personal computer with Ryzen 5 3500x processor running Microsoft Windows 10 operating system.

Results. The web-software for downloading and watching videos on the Internet has been developed. A software web application implemented on the Vue.js and Express.js platforms has been created. Implemented database for storing video metadata.

Conclusions. The developed software provides users with the ability to authorize and register, view lists of all available videos on the platform, view the video file directly in the browser, and create their own videos.

The field of use. Such a program can be used by ordinary users to share video content.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Стрімінг відео.....	9
1.2 Огляд та порівняння відеохостингів.....	10
1.3 Існуючі формати відео	15
1.4 Висновки за розділом 1	23
2 Розробка архітектури програми.....	24
2.1 Функціональні вимоги до програмного забезпечення.....	24
2.2 Вибір засобів розробки програмного забезпечення.....	26
2.3 Проєктування бази даних	31
2.4 Висновки за розділом 2	32
3 Розробка програми	33
3.1 Алгоритм функціонування програмного забезпечення.....	33
3.2 Структура реалізованого програмного забезпечення.....	35
3.3 Опис прийнятих рішень під час розробки ПЗ	39
3.4 Висновки за розділом 3	44
4 Експлуатація, тестування та експериментальне дослідження програми	45
4.1 Призначення програми.....	45
4.2 Умови виконання програми.....	45
4.3 Виконання програми	45
4.4 Висновки за розділом 4	50
Висновки	51
Перелік джерел посилання	52
Додаток А Технічне завдання	55
Додаток Б Текст програми	59
Додаток В Слайди презентації.....	74

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

BSD – Berkeley Software Distribution;

FK – Foreign key, Зовнішній ключ;

SEO – Search Engine Optimization;

SPA – Single Page Application;

БД – база даних;

ПЗ – програмне забезпечення;

СКБД – система керування базами даних.

ВСТУП

Наразі людей, які не знають про сайти відеохостингу, майже не залишилося. До аудиторії відеохостингів входять різні вікові та соціальні групи. Ними користуються практично всі: про канал на сервісах відеохостингу замислюються і компанії, які хочуть розкрутити свої проекти, і блогери, які бажають набрати аудиторію. Наразі це не просто проекти для обміну відео. Сьогодні відеохостинги – це ще й потужні канали просування власного бренду.

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 kbit 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 ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Стрімінг відео

Стрімінг відео, тобто його завантаження і розповсюдження, реалізується у вигляді відеохостингів, завданням яких є насамперед зберігання та перегляд відео. На даний момент найбільш актуальними відеохостингами є стрімінг-сайти або сайти відеохостингу, що дозволяють не тільки зберігати та передивлятися відеофайли, а й шукати нову аудиторію.

Відеохостинг [1] – це система перегляду і роботи з відеоматеріалом, який дозволяє користувачам в інтернеті зберігати, продивлятися, конвертувати та відтворювати відео через деякий застосунок, наприклад, браузер. Серед вже згаданих функцій є й інші, що притаманні не усім сервісам, але є досить актуальними:

- монетизація відео. Після того, як користувач завантажив відео на хостинг, завдяки рекламі, кількості переглядів, інтеграційним засобам, автор може заробити деяку суму грошей, що при певних обставинах може стати його видом заробітку;

- стрімінг у реальному часі [2]. Користувачі мають можливість займатися стрімінгом у реальному часі не тільки з використанням сервісів відеохостингу, а також і у соцмережах, як наприклад: Instagram [3], Facebook [4], Periscope [5] та інших, що можуть мати інше первинне призначення і пропонувати стрімінг у реальному часі як додаткову опцію.

Також слід відзначити, що існують певні різновиди відеохостингів, такі як:

- швидкі відео [6] – це застосунки, що мають за мету створення «швидкого» контенту, тобто такого, через який користувач не витрачає багато часу, частіше за все це відео-фрагменти від 10 секунд до 2 хвилин загального хронометражу;

- стрімінгові сервіси [7] мають за основну мету реалізацію грамотного процесу стрімінгу у реальному часі, який і буде більшою частиною функціоналу проєкту;

- класичні відеохостинги [8] дозволяють створення досить великого за загальним хронометражем відео, зазвичай від 7 хвилин до 3 годин. Такими сервісами часто користуються ТВ-канали для цифровізації свого контенту.

У даній роботі передбачається розробка проєкту, що реалізує основні функції класичного відеохостингу.

1.2 Огляд та порівняння відеохостингів

Існує безліч сайтів, що підтримують, або ж мають головною ціллю або реалізацію класичного сервісу відеохостингу, або стрімінгу.

1.2.1 YouTube

YouTube [9] (рис. 1.1) – це найпопулярніший, на даний момент, відеохостинг і соціальна платформа, заснована у 2005 році компанією Google. Станом на 2022 рік, YouTube є другим за популярністю сайтом у світі. Мільйони користувачів кожен день переглядають контент, створений для YouTube блогерами, або телеканалами, які буквально щосекунди завантажують його туди. Згідно зі статистикою, сервісом користуються 2.6 мільярди користувачів на місяць [10].

Сайт дає своїм користувачам такі можливості:

- перегляд відео;
- монетизація контенту і створення рекламного контенту для багатьох світових брендів;
- соціальна система, в якій користувачі можуть ставити лайки, коментувати відео, а також приймати участь у модерації;
- створення власного контенту без обмежень, окрім реєстрації;

– інтеграція в інші соціальні мережі.

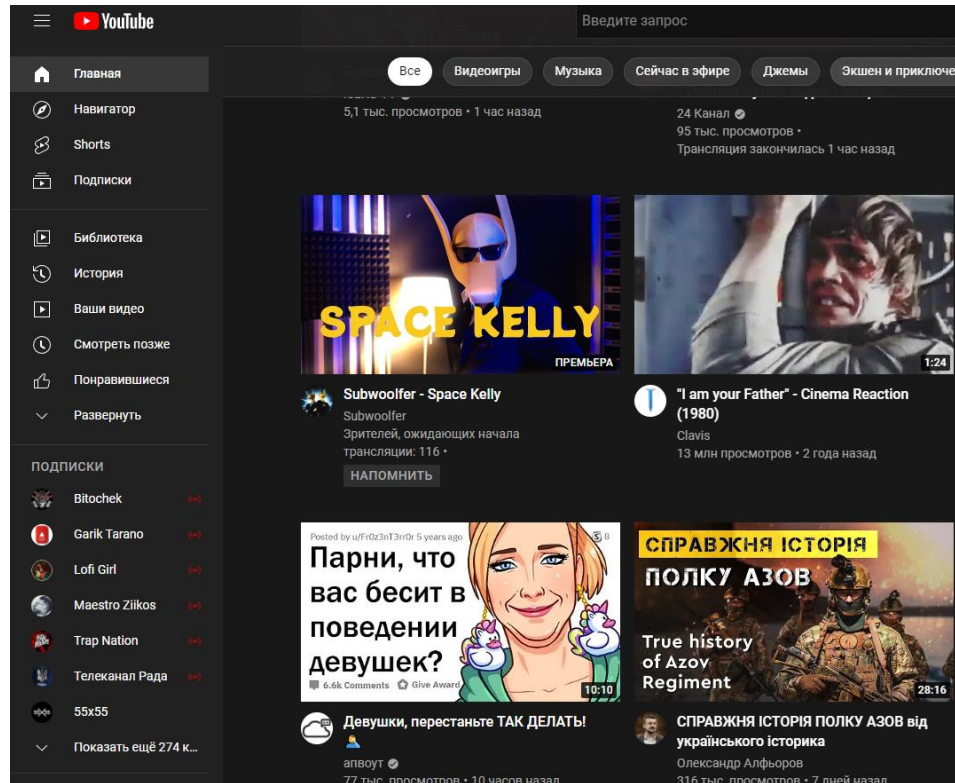


Рисунок 1.1 – Головна сторінка сайту-відеохостингу YouTube

1.2.2 Vimeo

Vimeo [11] (рис. 1.2) – як і YouTube є сервісом відеохостингу, що надає послуги зі збереження та трансляції відеоматеріалів. Сайт є також популярним у США, але у країнах Східної Європи навпаки, попитом не користується. Vimeo з’явився навіть раніше, ніж його найближчий конкурент, а саме у 2004 році і має декілька суттєвих переваг над ним, наприклад:

- вищий бітрейт у відео, яке завантажує користувач;
- краща якість звуку.

Але також цей сервіс має і суттєвий мінус у вигляді меншої кількості контенту, через обмежену популярність платформи, у порівнянні з деякими аналогами.

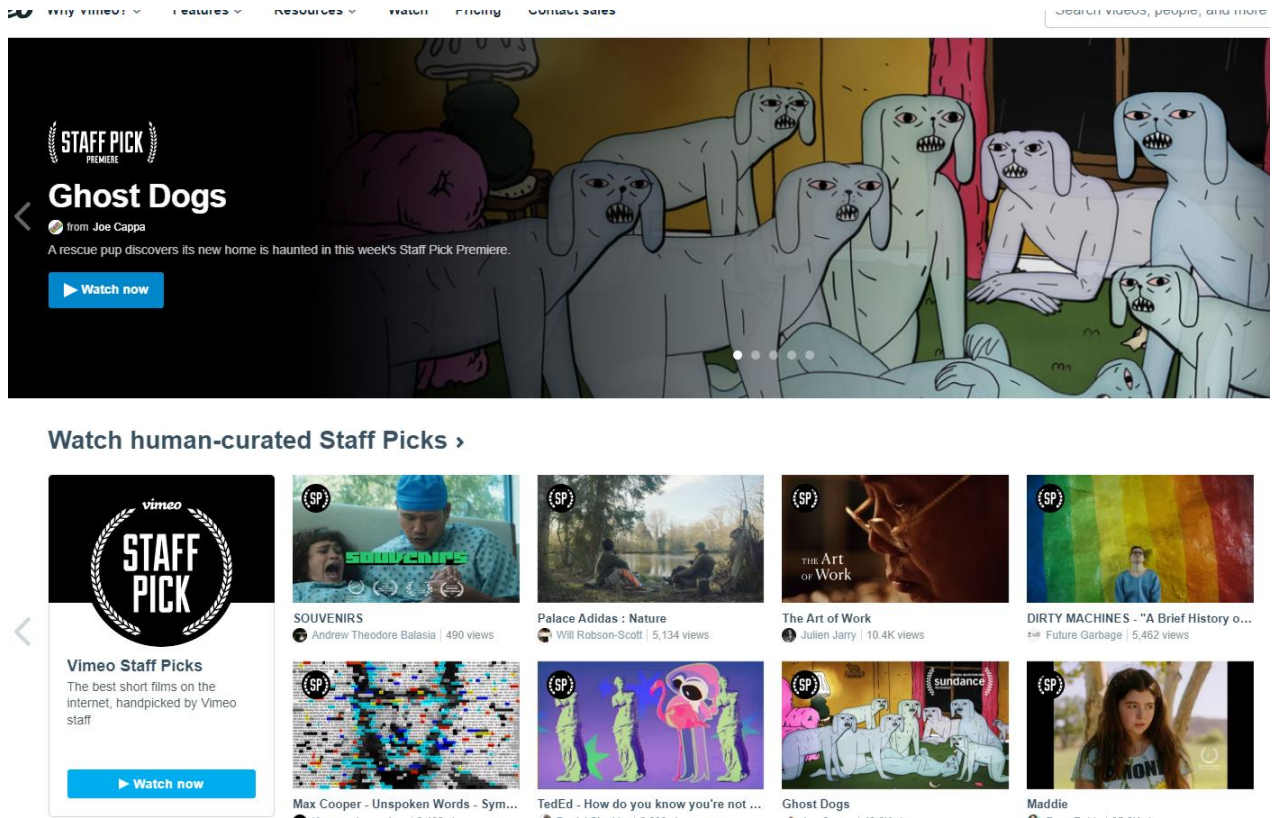


Рисунок 1.2 – Платформа Vimeo

1.2.3 9GAG

9GAG [12] (рис. 1.3) – розважальна платформа, що, як і попередні, надає послуги в збереженні відео, але на відміну від класичних сервісів має трохи інше призначення. Платформа робить ставку на соціальну складову сайту, а саме взаємодію з користувачами, показуючи їм контент незначного розміру у вигляді смішних відео. Платформа розташовує контент по вкладкам та тегам, що мають певні ключові слова, за якими і будуть знаходитися потрібні відео. Тобто сама платформа:

- не дозволяє зберігати великі за об’ємом відео;
- робить розважальний контент для користувачів від користувачів;
- на відміну від інших не монетизується;
- має розгалужену структуру, схожу на форум.

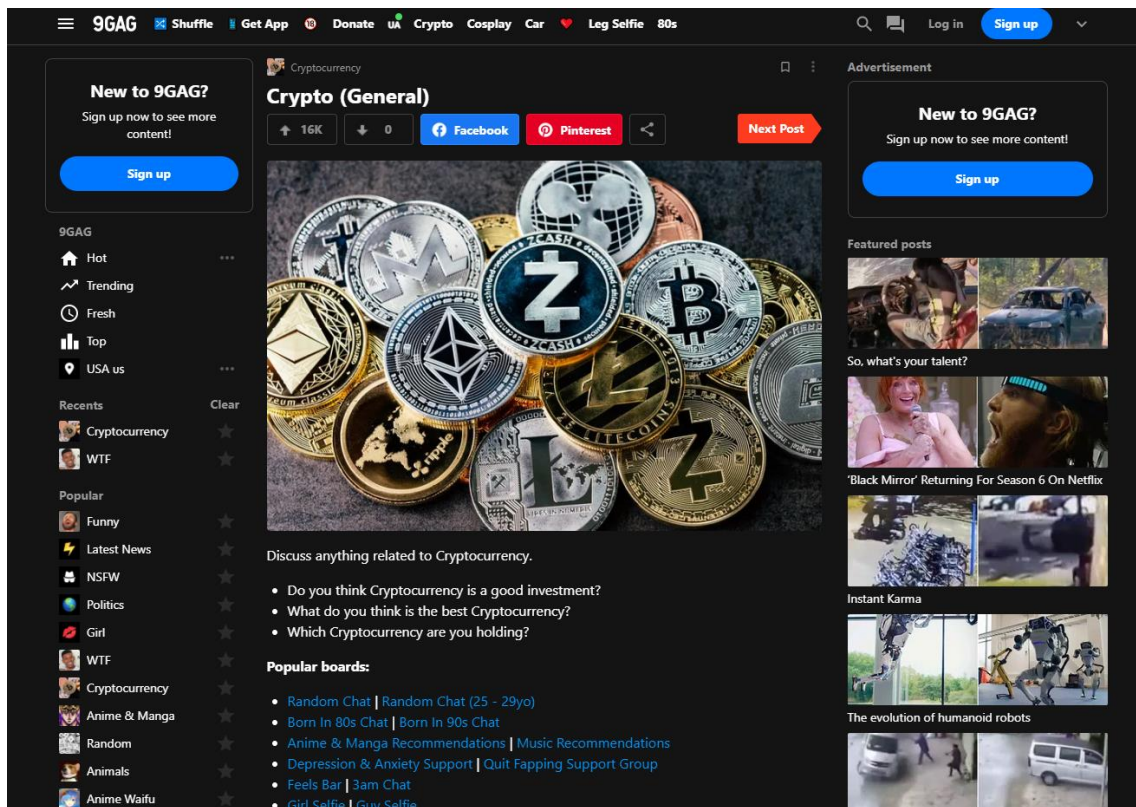


Рисунок 1.3 – Платформа 9GAG

1.2.4 TikTok

TikTok [13] (рис. 1.4) – молода розважальна та стрімінгова платформа швидких відео, що була заснована у 2016 році компанією ByteDance [14] у Китаї. Застосунок дозволяє користувачам завантажувати, а також створювати різні за тематикою відео (танцювальні, комедійні, скетчі та інші), з тривалістю від 5 секунд до 3 хвилин, а кількість користувачів у платформі вже перевищила кількість у 2 мільярди [15].

Платформа пропонує користувачам такі функції як:

- перегляд відео за спеціальним алгоритмом, згідно інтересів користувача з залученням штучного інтелекту;
- пошук відео за тегоми, а також каналами інших користувачів;
- досить зрозуміла завдяки іншим платформам система розповсюдження вподобаних відео серед друзів, а також коментарів;

– можливість записувати відео всередині платформи, використовуючи вбудовані інструменти.

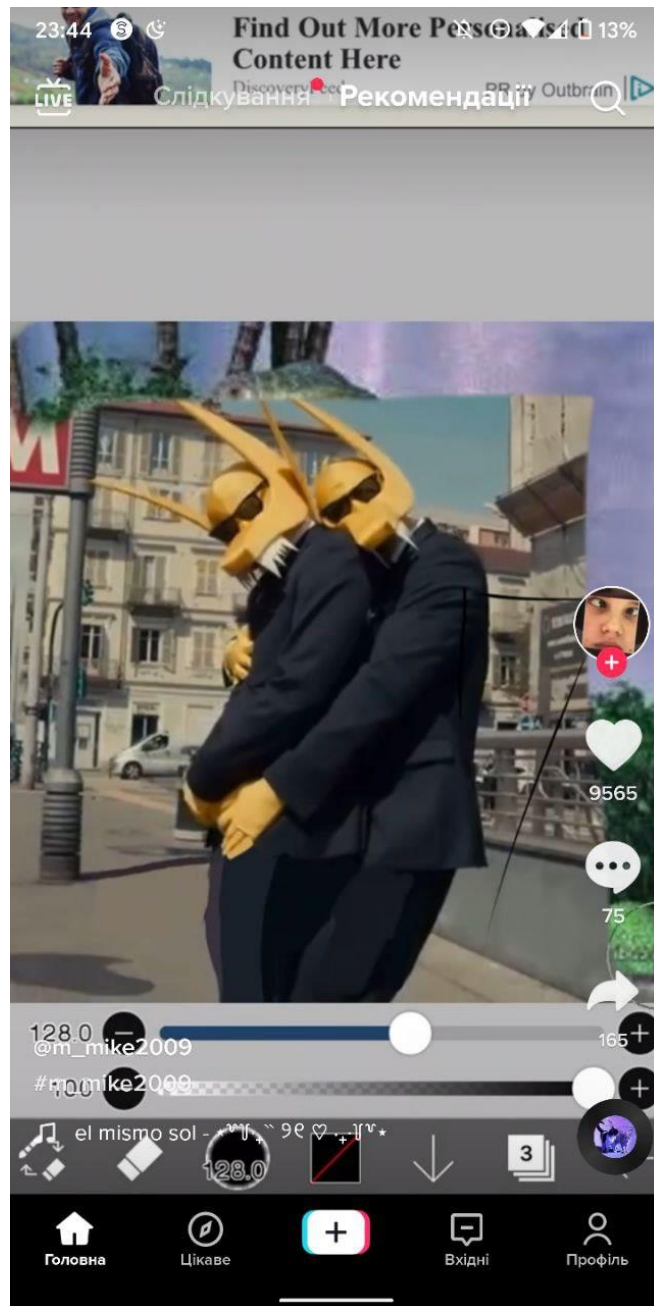


Рисунок 1.4 – Платформа TikTok

У таблиці 1.1 наведено основні ключові відмінності та подібності між приведеними сервісами.

Таблиця 1.1 – Порівняльна таблиця відеохостингів

Назва сервісу	Середня тривалість відео	Фільтри, або інші додатки для відео	Можливість реакцій, коментарів	Можливість користувачів вести живі трансляції (стріми)
YouTube	15хв	-	+	+
TikTok	30с	+	+	+
Vimeo	7-9хв	-	+	-
9GAG	10-30с	-	+	-

1.3 Існуючі формати відео

Формат відео – формат файлу цифрових відеоданих у комп’ютерній системі, що зберігається за допомогою стиснення з втратами для мінімізації розміру самого файлу.

Відеофайл зазвичай складається з контейнера (рис. 1.5), що містить у собі дані у форматі кодування відео (наприклад, AV1) разом із аудіоданими у форматі аудіокодування (наприклад, FLV). Контейнер також містить інформацію про синхронізацію, субтитри та метадані, наприклад дата створення [16]. Стандартний формат відео, такий як .webm – це профіль, визначений обмеженням щодо формату контейнера та дозволених форматів стиснення даних.

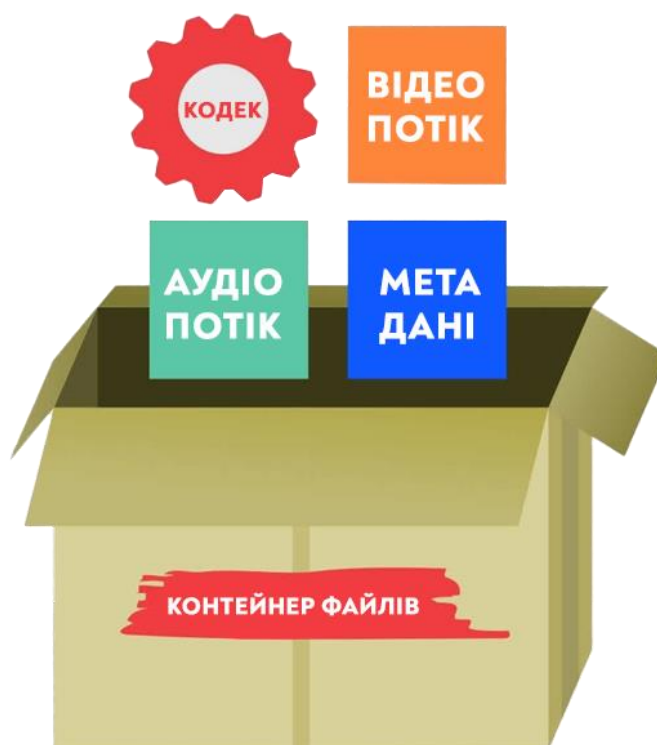


Рисунок 1.5 – Складові контейнера

Закодовані відео та аудіо дані всередині контейнера відеофайлів (тобто не заголовки, колонтитули та метадані) називають сутністю.

Програмне або апаратне забезпечення, яке може займатися декодуванням стиснутого відео чи аудіо, називають кодеком. Відтворення або кодування відеофайлу іноді вимагає від користувача встановлення бібліотеки кодеків, що відповідає типу кодування відео та звуку, що використовується у файлі.

Хороший дизайн передбачає, що формат файлу дозволяє користувачеві визначити, яка програма відкриє файл із розширення файлу. Це стосується деяких форматів відеофайлів, таких як WebM, Windows Media Video, Flash Video і Ogg Video, кожен з яких може містити лише кілька визначених підтипів форматів кодування відео та аудіо, що дозволяє відносно легко дізнатися, який кодек буде відтворювати файл. На відміну від цього, деякі типи контейнерів загального призначення, такі як AVI [17] і QuickTime [18], можуть містити відео та аудіо майже в будь-якому форматі та мати

розширення файлів, названі за типом контейнера, що не дозволяє кінцевому користувачеві перевірити тільки розширення файлу, щоб визначити, який кодек або програму використовувати для відтворення файлів.

Найпоширеніші формати відео представлені у таблиці 1.2.

Таблиця 1.2 – Порівняльна таблиця найпоширеніших відеоформатів

Назва формату	Розширення	Контейнер	Кодування відео	Кодування аудіо
1	2	3	4	5
Flash Video	.flv	FLV	VP6, Sorenson Spark, Екранне відео, Екранне відео 2, H.264	MP3, ADPCM, Nellymoser, Speex, AAC
WEBM	.webm	Matroska	VP8,VP9, AV1	Vorbis, Opus
MPEG-4	.mp4,.m4p, .m4v	MPEG-4, частина 12	H.264, MPEG-4, частина 2, MPEG-2, MPEG-1	Немає значення
GIF	.gif	Немає	Немає	Відсутнє аудіо
MATROSKA	.mkv	Matroska	Немає значення	Немає значення
AVI	.avi	AVI	Немає значення	Немає значення

Продовження таблиці 1.2

1	2	3	4	5
3GPP	.3gp	MPEG-4, частина 12	MPEG-4, частина 2, H.263, H.264	AMR-NB, AMR-WB, AMR-WB +, AAC-LC, HE- AAC v1 або розширений aacPlus (HE- AAC v2)
Windows Media Video	.wmv	WMV	Windows Media Video, Windows Media Video Screen, Windows Media Video Image	Windows Media Audio, Sipro ACELP.net

Слід зазначити, що найбільш популярними і актуальними форматами відео для програмного забезпечення (ПЗ) відеохостингу є такі формати, як MPEG-4, WEBM, а в минулому – формати Flash Video та 3GPP. Тому ці формати слід розібрати детальніше, ніж інші.

1.3.1 MPEG-4

MPEG-4 є методом визначення стиснення цифрових аудіо та візуальних даних, що був розроблений Тураджем Ебрахімі, що у майбутньому стане президентом JPEG, а також Фернандо Перейрою [19]. Він був впроваджений у кінці 1998 року і призначений стандартом для групи форматів

кодування аудіо та відео та відповідних технологій, а також узгоджений групою експертів із рухомого зображення ISO/IEC (MPEG) (ISO/IEC JTC1/SC29/WG11) відповідно до офіційного стандарту. ISO/IEC 14496 – стандарт кодування аудіовізуальних об'єктів [20]. Застосування MPEG-4 включають стиснення AV-даних для розповсюдження Інтернет-відео і компакт-дисків, голосу (телефон, відеотелефон) і телевізійних програм.

Спочатку MPEG-4 був призначений насамперед для відеозв'язку з низькою бітовою швидкістю, однак згодом його сфера застосування як стандарту мультимедійного кодування була розширена. MPEG-4 є ефективним для різноманітних швидкостей передачі даних від кількох кілобіт на секунду до десятків мегабіт на секунду.

MPEG-4 надає такі функції:

- покращена ефективність кодування порівняно з MPEG-2;
- можливість кодування змішаних медіа даних (відео, аудіо, мовлення);
- стійкість до помилок для забезпечення надійної передачі;
- можливість взаємодії з аудіо-візуальною сценою, створеною на приймачі.

Сам формат насправді складається з кількох стандартів, або ж, як їх називають, «частин». Кожна «частина» охоплює певний аспект усієї специфікації. В таблиці 1.3 зазначені стандарти MPEG-4 [21]-[22].

Таблиця 1.3 – Список стандартів MPEG-4

Номер «частини»	Номер стандарту	Рік появи	Назва (Сфера відповідальності)
1	2	3	4
Part 1	ISO/IEC 14496-1	1999	Systems
Part 2	ISO/IEC 14496-2	1999	Visual
Part 3	ISO/IEC 14496-3	1999	Audio

Продовження таблиці 1.3

1	2	3	4
Part 4	ISO/IEC 14496-4	2000	Conformance testing
Part 5	ISO/IEC 14496-5	2000	Reference software
Part 6	ISO/IEC 14496-6	1999	Delivery Multimedia Integration Framework
Part 7	ISO/IEC 14496-7	2002	Optimized reference software for coding of audio-visual objects
Part 8	ISO/IEC 14496-8	2004	Carriage of ISO/IEC 14496 contents over IP networks
Part 9	ISO/IEC 14496-9	2004	Reference hardware description
Part 10	ISO/IEC 14496-10	2003	Advanced Video Coding
Part 11	ISO/IEC 14496-11	2005	Scene description and application engine
Part 12	ISO/IEC 14496-12	2004	ISO base media file format
Part 13	ISO/IEC 14496-13	2004	Intellectual Property Management and Protection (IPMP) Extensions
Part 14	ISO/IEC 14496-14	2003	MP4 file format
Part 15	ISO/IEC 14496-15	2004	Carriage of network abstraction layer (NAL) unit structured video in the ISO base media file format
Part 16	ISO/IEC 14496-16	2004	Animation Framework eXtension
Part 17	ISO/IEC 14496-17	2006	Streaming text format
Part 18	ISO/IEC 14496-18	2004	Font compression and streaming

Продовження таблиці 1.3

1	2	3	4
Part 19	ISO/IEC 14496-19	2004	Synthesized texture stream
Part 20	ISO/IEC 14496-20	2006	Lightweight Scene Representation
Part 21	ISO/IEC 14496-21	2006	MPEG-J Graphics Framework eXtensions (GFX)
Part 22	ISO/IEC 14496-22	2007	Open Font Format
Part 22	ISO/IEC 14496-22	2007	Open Font Format
Part 23	ISO/IEC 14496-23	2008	Symbolic Music Representation (SMR)
Part 24	ISO/IEC 14496-24	2008	Audio and systems interaction
Part 25	ISO/IEC 14496-25	2009	3D Graphics Compression Model
Part 26	ISO/IEC 14496-26	2010	Audio Conformance
Part 27	ISO/IEC 14496-27	2009	3D Graphics conformance
Part 28	ISO/IEC 14496-28	2012	Composite font representation
Part 29	ISO/IEC 14496-29	2014	Web video coding
Part 30	ISO/IEC 14496-30	2014	Timed text and other visual overlays in ISO base media file format
Part 31	ISO/IEC 14496-31	У розробці	Video Coding for Browsers
Part 32	ISO/IEC 14496-32	У розробці	Conformance and reference software
Part 33	ISO/IEC 14496-33	У розробці	Internet video coding

1.3.2 WEBM

WebM – це аудіовізуальний формат медіафайлів [23]. Насамперед, він має на меті запропонувати безоплатну альтернативу для використання у відео та аудіоелементах HTML5. У нього є дочірній проєкт WebP для зображень. Розробка формату спонсорується Google, а відповідне програмне забезпечення розповсюджується під ліцензією BSD (Berkeley Software Distribution).

На відміну від MPEG-4, формат WebM має трохи гіршу якість звуку та відео, але слід зазначити, що і стиснення проходить до набагато менших розмірів саме у WebM.

Формат WebM спеціально призначений для потокових програм і дає хороші результати, як відносно розміру файлу, так і відносно швидкості. Але знову недостатня поширеність є проблемою: хоча багато пристроїв Android мають підтримку VP8, а виробники телевізорів, такі як LG, Samsung, Sony і Sharp, поступово оснащують свої пристрої підтримкою VP8/VP9, iPhone, iPad та інші пристрої Apple все ще не надають апаратної підтримки цього кодеку. Це означає, що можна очікувати, що їх продуктивність буде набагато нижчою.

Підтримку перегляду файлів у форматі WebM, забезпечують такі офлайн медіа плеєри:

- VLC;
- KMPlayer;
- Media Player Classic;
- Winamp;
- jetAudio Basic;
- Moovida Core;
- Perian;
- XMBC.

1.4 Висновки за розділом 1

Як показав проведений аналіз, наразі існує багато програмних реалізацій сервісів відеохостингу, що є різними як за своєю природою, так і за функціональністю.

Було розглянуто поняття відеохостинг або ж сайт стрімінгу, основні принципи їх роботи та базові функції.

В результаті досліджень було вирішено розробити відеохостинг, що більш за все буде схожий саме на класичний відеохостинг, як то Vimeo або YouTube.

Для досягнення мети роботи слід розв'язати такі задачі:

- виконати аналіз вимог до ПЗ;
- розробити архітектуру бази даних (БД), сайта в цілому, а також вибрати необхідний набір інструментів;
- провести серію досліджень щодо оптимізації програмного коду;
- розробити програмну документацію.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Функціональні вимоги до програмного забезпечення

Програма 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.

Сценарії використання програми авторизованим користувачем представлені на рис. 2.1.

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.

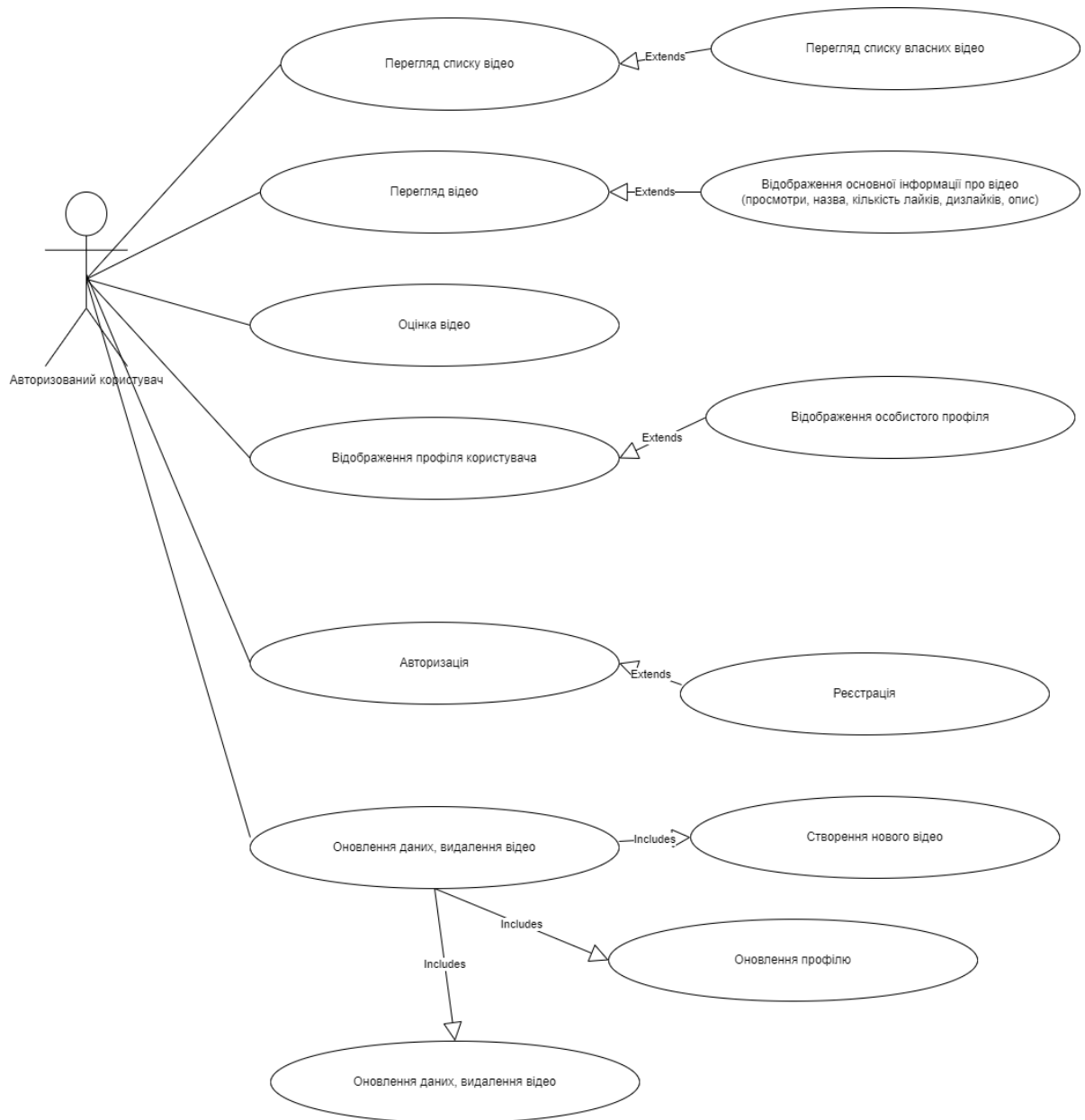


Рисунок 2.1 – Діаграма прецедентів авторизованого користувача

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

Сценарії використання програми неавторизованим користувачем представлені на рис. 2.2.

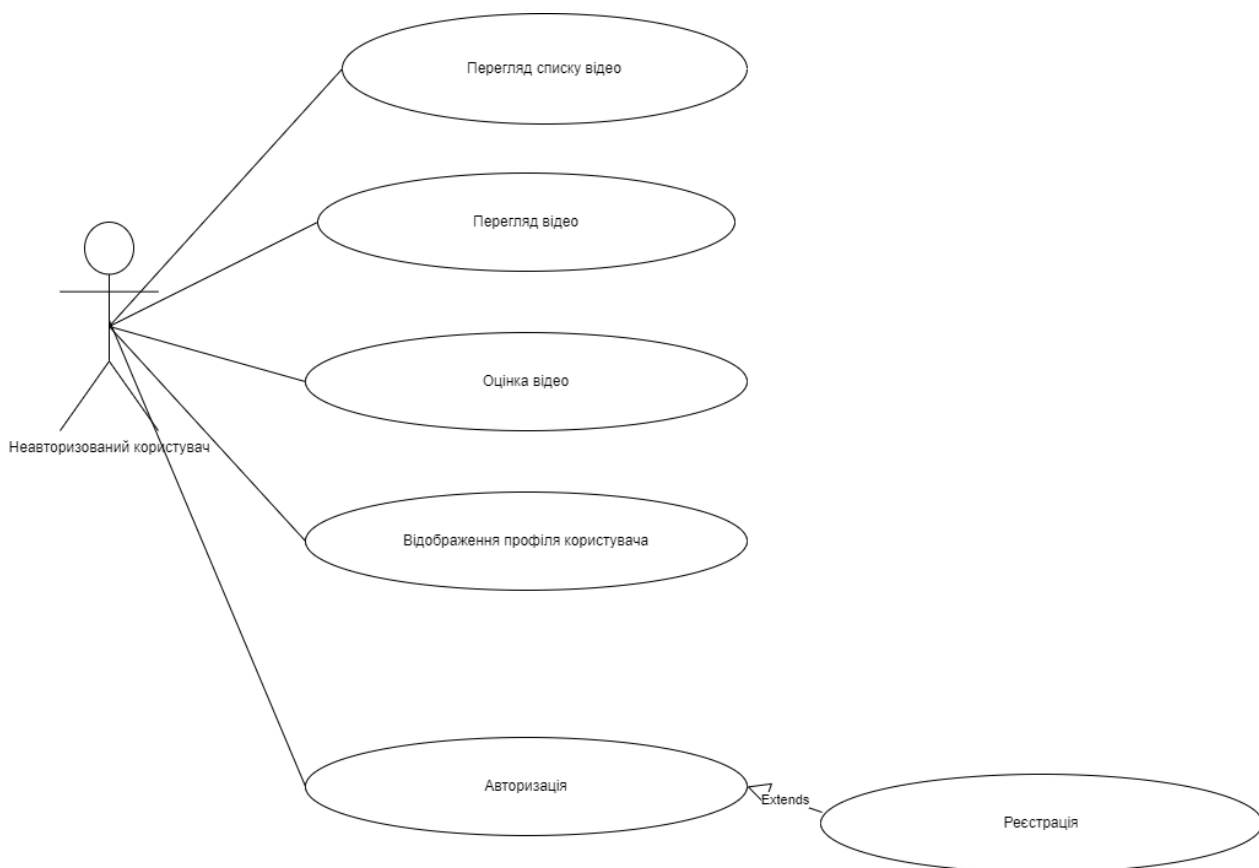


Рисунок 2.2 – Діаграма прецедентів неавторизованого користувача

2.2 Вибір засобів розробки програмного забезпечення

У даному підрозділі необхідно провести аналіз мов, що використовують для розробки подібних проєктів, та вибрати найбільш підходящу мову програмування, а також фреймворки для реалізації клієнт-серверної архітектури та розробки програмного забезпечення. Також слід зробити вибір засобів для взаємодії з БД, бо основна частина роботи програми буде пов'язана саме з роботою з даними серверу.

2.2.1 Вибір мови програмування

Клієнтську частину додатку вебдодатку можна розробити на Python або на JavaScript, використовуючи фреймворки цих мов програмування. Результати порівняння цих мов можна побачити у таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування для розробки клієнтської частини вебдодатку

Назва мови програмування	Python	JavaScript
Нативна підтримка роботи з CSS, HTML	-	+
Фреймворки для роботи з клієнтською частиною вебдодатку	Flask	Vue, Svelte, Angular
Типізація	Динамічна	Динамічна
Можливість одразу побачити результат праці над програмою	-	+

Серверну частину вебдодатку можна розробляти з використанням мов JavaScript, а саме фреймворку Node.JS, C#, а також Python. Порівняння цих трьох мов наведене у таблиці 2.2.

Таблиця 2.2 – Порівняння мов програмування для розробки серверної частини вебдодатку

Назва мови програмування	C#	Python	JavaScript
Простота в використанні	-	-	+
Фреймворки для розробки серверної частини додатку	ASP.NET	Django	Nest.js, Express.js, Koa.js
Типізація	Статична	Динамічна	Динамічна
Наявність многопоточності	+	+	+ (можливість створення додаткових допоміжних потоків)

Проаналізувавши результати порівняння, було прийнято рішення вибрати мову програмування JavaScript, як мову для реалізації як клієнтської, так і серверної частини додатку. Серед переваг мови JavaScript можна зазначити:

- нативну підтримку роботи з мовою розмітки гіпертексту HTML, а також з мовою каскадних стилів CSS;
- більшу простоту у використанні і написанні коду;
- більший вибір фреймворків для розробки клієнт-серверної архітектури;
- динамічну типізацію, завдяки якій можна запобігти великим проблемам при швидкому написанні програмного коду.

2.2.2 Вибір фреймворків для розробки програмного забезпечення

Клієнтську частину додатку мовою JavaScript можна розробити використовуючи наступні фреймворки:

- Angular;
- Svelte;
- Vue.js.

Особливістю цих фреймворків є реалізація програмного застосунку у вигляді SPA. SPA (Single Page Application) – це реалізація вебпрограми, яка завантажує лише один вебдокумент, а потім оновлює основний вміст цього окремого документа за допомогою API JavaScript, таких як XMLHttpRequest та Fetch, безпосередньо в той момент коли має бути показаний інший вміст.

Таким чином, це дозволяє користувачам використовувати вебсайти без завантаження нових сторінок із сервера, що може призвести до підвищення продуктивності та більш динамічного досвіду. Цей підхід однак відомий деякими недоліками, такими як погана SEO (Search Engine Optimization), більша кількість зусиль, необхідних для підтримки стану, впровадження навігації та значущої продуктивності [24].

На рис. 2.3 зображена різниця між традиційним підходом і SPA додатками, написаними мовою JavaScript [25].

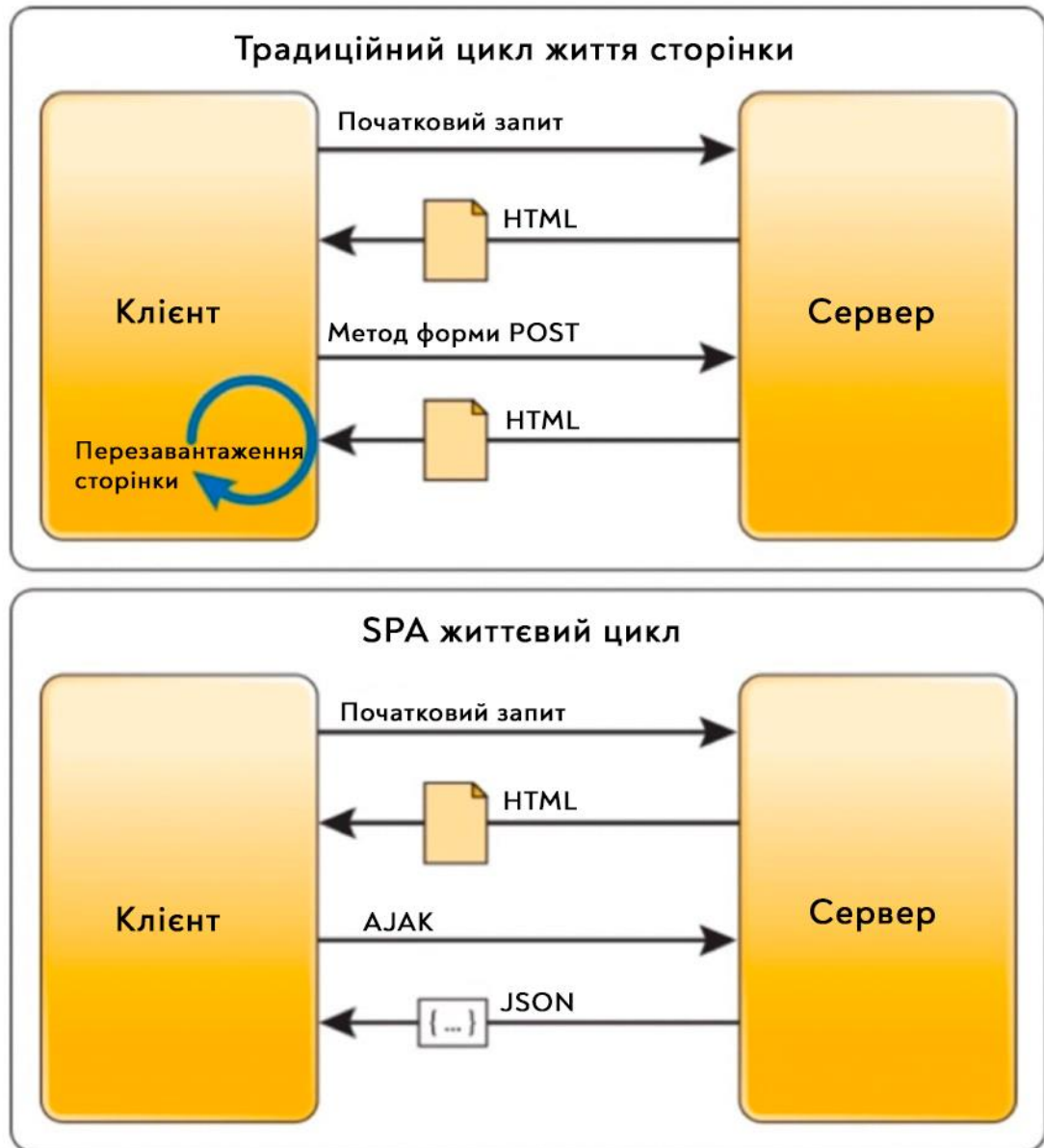


Рисунок 2.3 – Порівняння традиційного та SPA додатків

Аналіз клієнтських фреймворків мови JavaScript проведений у таблиці 2.3.

Таблиця 2.3 – Порівняння клієнтських фреймворків мови JavaScript

Назва фреймворку	Vue	Angular	Svelte
Типізація	Динамічна	Статична	Динамічна
Інтегроване глобальне сховище	+	-	+
Реактивність	+	-	+
Схожість на класичну мову JS	-	-	+
Простота використання	+	-	-

Express.js [26] – це найпопулярніший наразі фреймворк розробки серверної архітектури для Node.js, який реалізує всю функціональність, необхідну для створення ефективних додатків і API. Його використання значно допомагає у скороченні програмного коду, а отже і у зменшенні часових витрат на розробку.

За результатами порівняння можна зробити висновок, що завдяки саме простоті використання, а також реактивності при використанні пари Vue.js та Express.js у розробці програмного забезпечення є найбільш ефективним.

А у якості СКБД (система керування базами даних) було прийнято рішення використовувати MySQL.

MySQL [27] є найпопулярнішою у світі системою керування базами даних з відкритим кодом. Незважаючи на свої потужні функції, MySQL проста у налаштуванні та зручна у використанні. Вона підтримує майже усі мови програмування, тому є досить універсальним рішенням.

Також було вирішено використати ще розподілене швидке сховище Redis [28], сховище пар ключ-значення, які зберігаються у оперативній пам'яті.

Його задачею є збереження даних авторизованих користувачів, а саме токенів, завдяки яким вони будуть мати доступ до сайту.

2.3 Проектування бази даних

БД 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.

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 БД, цю функцію виконує таблиця userVideoMap, яке буде мати наступні поля:

- id – первинний ключ;
- userId – зовнішній ключ, що зв'язує з таблицею User;
- videoId – зовнішній ключ, що зв'язує з таблицею Video.

Структура БД зображена на рис. 2.4.

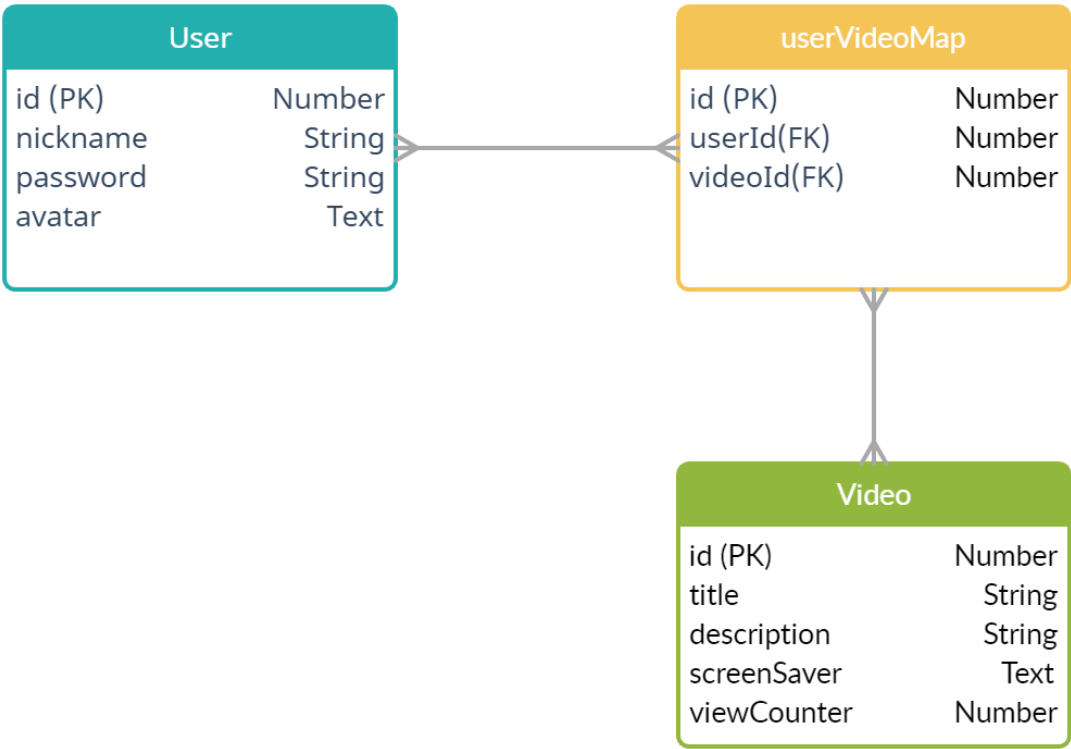


Рисунок 2.4 – Схема БД

2.4 Висновки за розділом 2

В даному розділі було досліджено основні функції і відмінності між традиційним клієнтським додатком і SPA додатком, досліджено відмінності між мовами програмування, проведене порівняння фреймворків.

Також було виконано опис функціональних вимог, побудовані діаграми прецедентів для авторизованого та неавторизованого користувачів, побудовано сценарії взаємодії, а також зроблено вибір необхідних інструментів для реалізації розробки ПЗ клієнт-серверної архітектури, а саме мови програмування JavaScript, та фреймворків Vue.js і Express.js. Окрім того було обґрунтовано вибір СКБД MySQL для взаємодії з базою даних, і спроектована структура БД.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програмного забезпечення

Алгоритм функціонування ПЗ 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.

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, імейл 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.

Спрощена схема роботи сайту представлена на рис. 3.1.

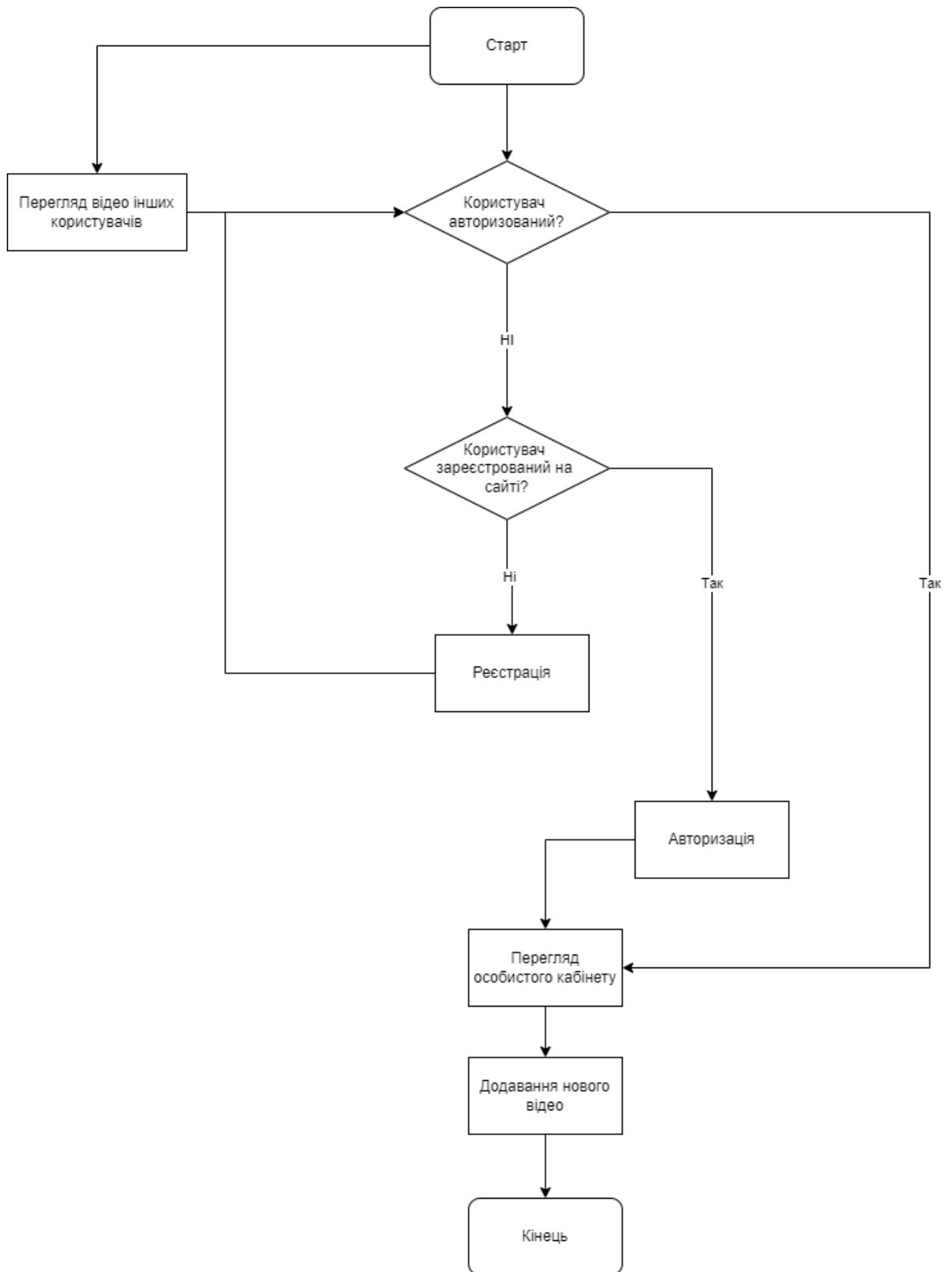


Рисунок 3.1 – Схема роботи сайту

3.2 Структура реалізованого програмного забезпечення

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

3.2.1 Структура клієнтської частини додатку

До структури клієнтської частини відносяться:

- представлення – динамічні вебсторінки, 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 [29];
- компоненти – частини сайту, призначені для повторного використання, що винесені у окремий елемент задля оптимізації сайту та скорочення написаного коду у процесі;
- міddlвар – файл, що відповідає за перенаправлення користувача з тих чи інших сторінок, якщо він не є авторизованим.

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

- AboutUsView.vue – сторінка, що має загальний опис призначення сайту;
- AccountView.vue – сторінка, що має основну інформацію про обліковий запис користувача;
- 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;
- `RegisterView.vue` – сторінка, на якій користувач має можливість зареєструватися, якщо не має особистого облікового запису;
- `VideoView.vue` – сторінка, на якій користувач може подивитися відео, а також загальну інформацію про нього.

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.

Структура клієнтської частини додатку представлена відповідною схемою на рис. 3.2.



Рисунок 3.2 – Структура клієнтської частини додатку

3.2.2 Структура серверної частини додатку

До структури серверної частини додатку відносяться:

- 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;
- сервіси – частини додатку, задачею яких є пряма взаємодія з СКБД, через які будуть отримані дані з відповідних до сервісів таблиць;
- міддлвар – як і у випадку з клієнтською логікою відповідає за логіку у випадку, якщо користувач не є авторизованим.

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.

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.

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

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

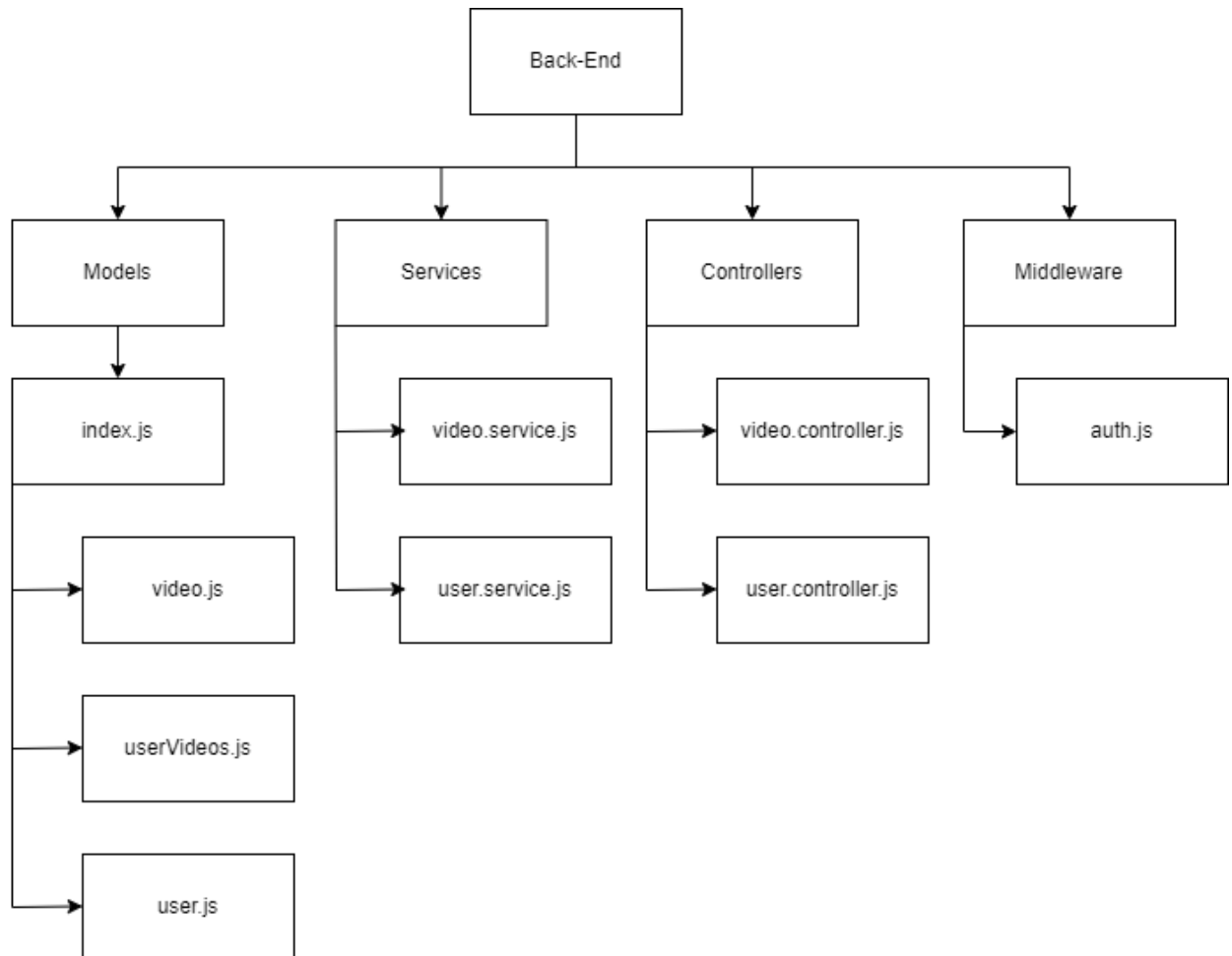


Рисунок 3.3 – Структура серверної частини додатку

3.3 Опис прийнятих рішень під час розробки ПЗ

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, чією 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` – змінна, 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.

3.4 Висновки за розділом 3

Під час написання розділу було описано структуру клієнтської та серверної частини сайту, представлені відповідні схеми, файли і їх функції, описано алгоритм роботи програми. Реалізовані основні методи програмного забезпечення, класи моделей, а також визначений зв'язок між таблицями БД, клієнтською та серверною частинами.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для надання користувачам можливості дивитися відео інших користувачів, представлення інформації про окремі відео, користувачів, а також функції поширення своїх відео. Програма дає можливість авторизації, реєстрації, завантаження відео та перегляду окремих, або всіх відео, а також дозволяє слідкувати за улюбленими авторами.

4.2 Умови виконання програми

Апаратні вимоги до серверу:

- жорсткий диск з вільним простором – 10 Гб;
- оперативна пам'ять – 8 Гб;
- процесор: 4 ядра, 3,5 ГГц частота.

Клієнт представлений браузером користувача. Для виконання вимог до нього потрібний сучасний браузер, наприклад Google Chrome, Mozilla Firefox, Opera, з доступом до мережі інтернет.

4.3 Виконання програми

Програма 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 рис. 4.1.

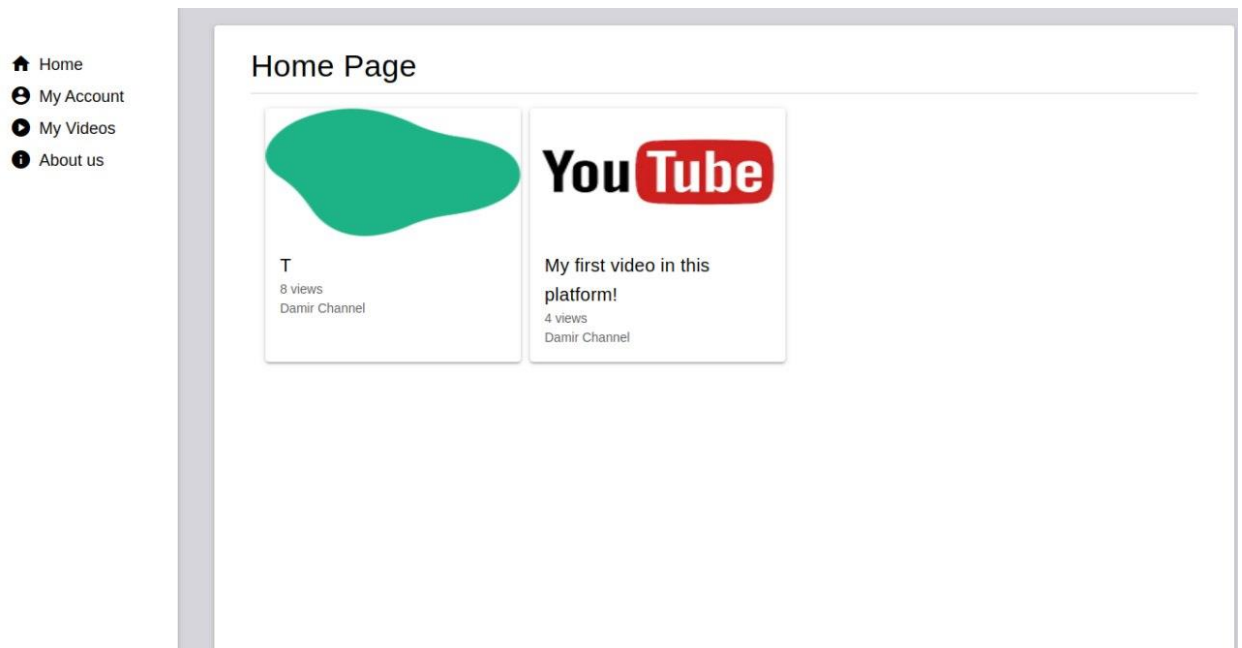
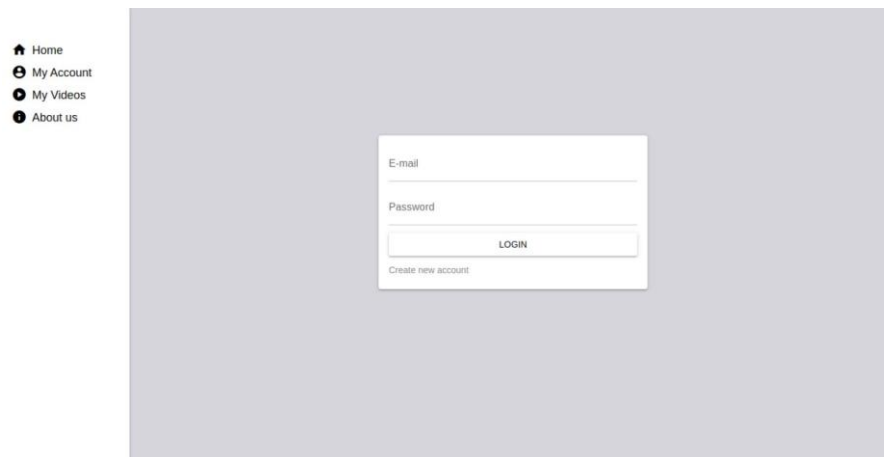


Рисунок 4.1 – Домашня сторінка сайту

Звідти, завдяки боковому меню 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”.

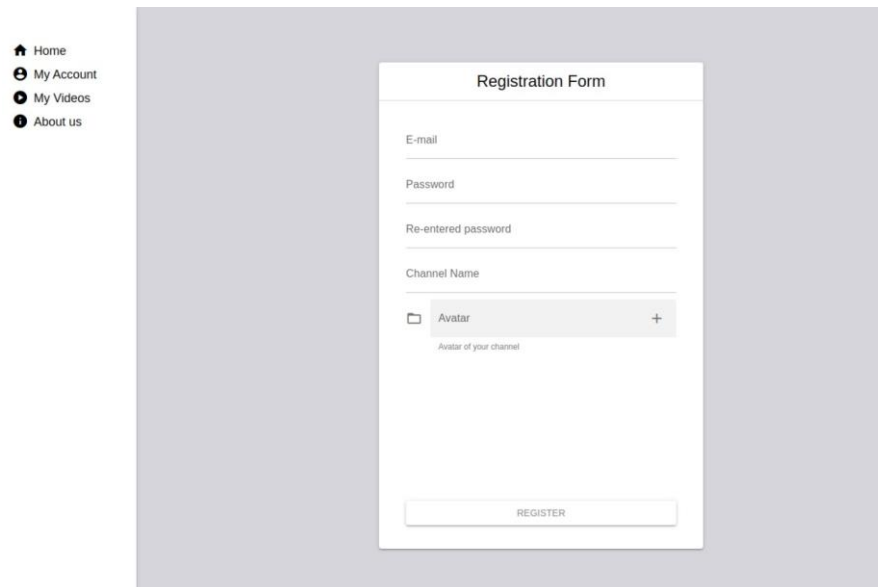
Для подальшої взаємодії з основним функціоналом сайту користувач має скористуватись сторінкою логіну, яку зображено на рис. 4.2. На цій сторінці користувач буде мати можливість авторизуватися, тобто ввести свою електронну пошту та свій пароль, якщо він вже має обліковий запис. У разі, якщо користувач ще не має свого акаунту, то він може перейти до сторінки реєстрації через сіру кнопку “Create new account”.



The image shows a login page with a light gray background. On the left, there is a vertical navigation menu with four items: 'Home' (with a house icon), 'My Account' (with a person icon), 'My Videos' (with a play button icon), and 'About us' (with an information icon). In the center, there is a white login form. It contains two input fields: 'E-mail' and 'Password'. Below these fields is a 'LOGIN' button. At the bottom of the form, there is a link that says 'Create new account'.

Рисунок 4.2 – Сторінка логіну

Сторінку реєстрації зображено на рис. 4.3.



The image shows a registration form on a light gray background. On the left, there is a vertical navigation menu with four items: 'Home' (with a house icon), 'My Account' (with a person icon), 'My Videos' (with a play button icon), and 'About us' (with an information icon). The registration form is titled 'Registration Form' and contains several input fields: 'E-mail', 'Password', 'Re-entered password', and 'Channel Name'. Below the 'Channel Name' field, there is an 'Avatar' section with a small square icon, the text 'Avatar', and a plus sign. Below this, there is a smaller text label 'Avatar of your channel'. At the bottom of the form, there is a 'REGISTER' button.

Рисунок 4.3 – Сторінка 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. 4.4.

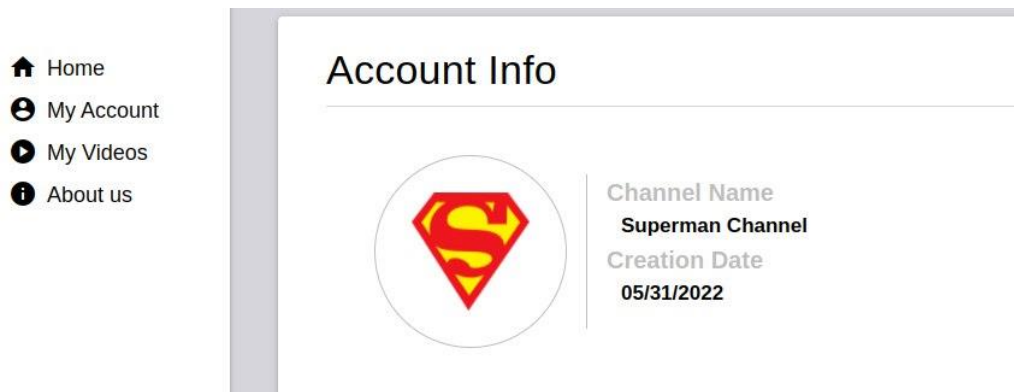


Рисунок 4.4 – Особиста сторінка користувача

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. 4.5.

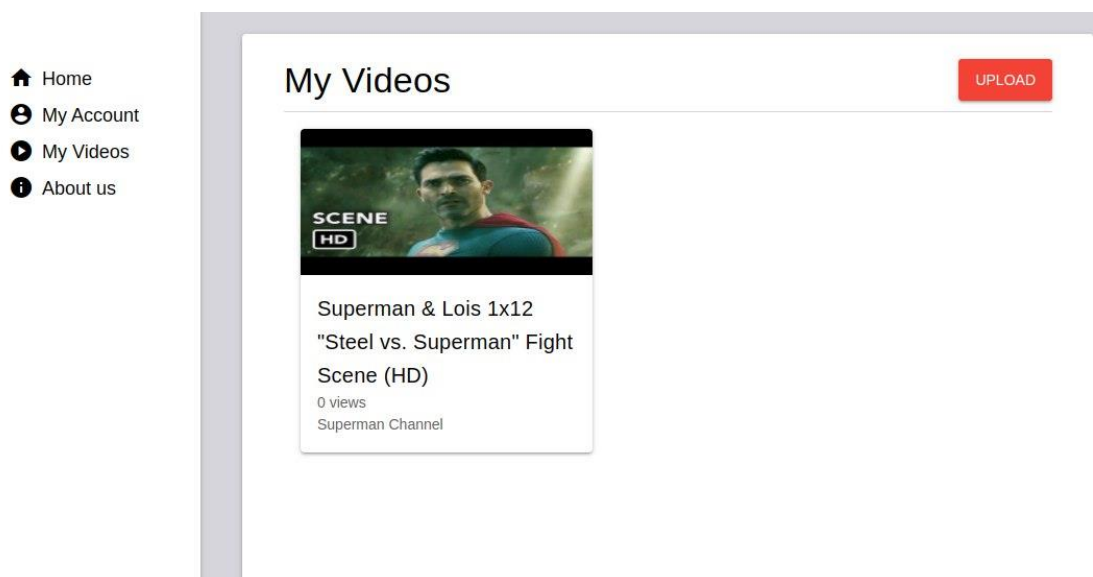


Рисунок 4.5 – Сторінка відео користувача

Окрім того, якщо користувач хоче завантажити нове власне відео, то він буде мати таку можливість, натиснувши на кнопку “Upload”. Користувач побачить модальне вікно, що зображено на рис. 4.6, де він повинен заповнити усі дані про відео, тобто назву, опис, заставку, а також завантажити саме відео.

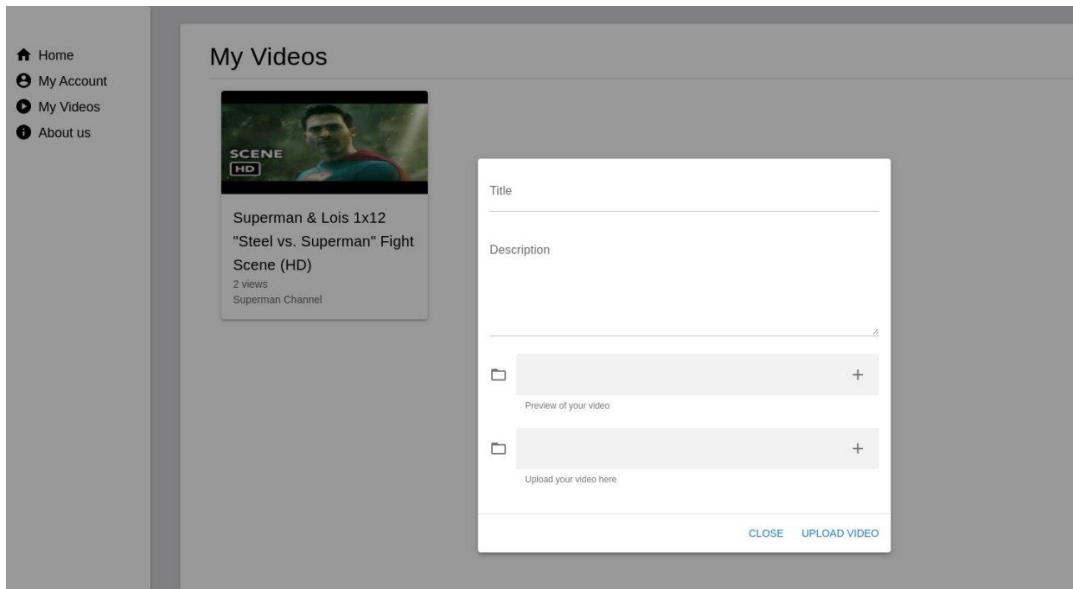


Рисунок 4.6 – Модальне вікно для завантаження відео

І найголовніше, 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. 4.7.

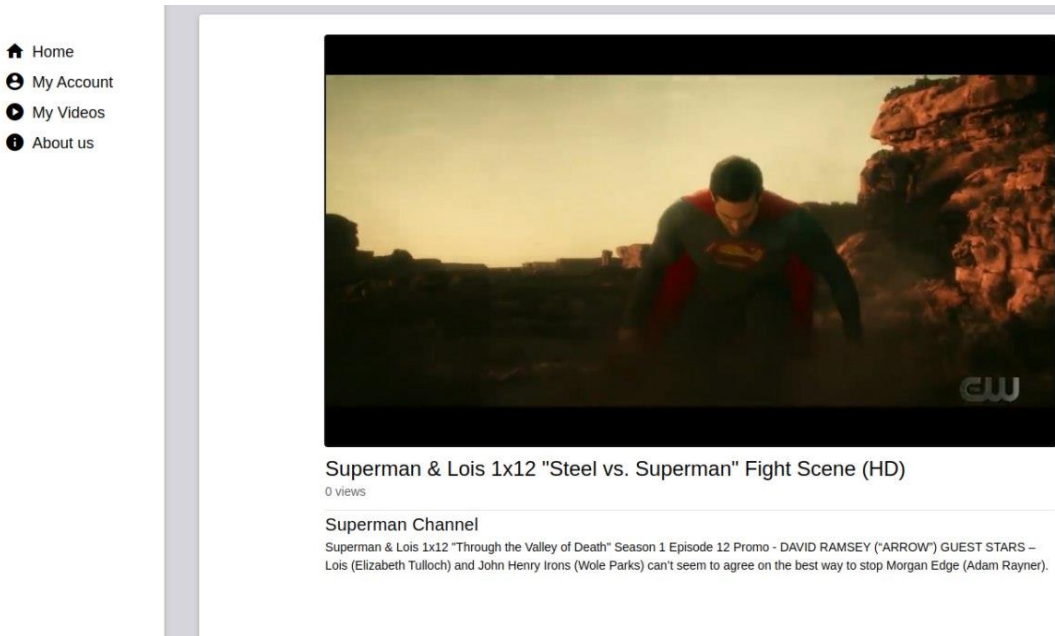


Рисунок 4.7 – Сторінка перегляду відео

4.4 Висновки за розділом

Під час написання розділу було описано призначення програми, її умова виконання, що саме потрібно для взаємодії з нею серед апаратних засобів, а також була розібрана і показана сутність роботи програми, її візуальну складову, а також послідовність дій у виконанні для отримання бажаного результату.

ВИСНОВКИ

В ході виконання дипломної кваліфікаційної роботи було розв'язано такі задачі:

- виконано аналіз проблеми сервісів відеохостингу, задля чого було проведено дослідження специфіки роботи відеохостингів, їх видів, а також проведено порівняння та огляд програмних аналогів;

- виконано розробку програмної архітектури сайту, під час якої було визначено функціональні вимоги до програмного забезпечення, змодельовано роботу з програмою на основні діаграм прецедентів, обґрунтовано вибір мови програмування JavaScript, а також технологій Vue.js та Express.js для реалізації клієнт-серверної архітектури, а також СКБД MySQL для розробки програмного забезпечення, визначено структуру БД для організації роботи;

- реалізоване програмне забезпечення, в результаті чого був описаний алгоритм роботи програми у вигляді блок-схеми, схеми структури клієнт-серверного додатку, описано моделі і таблиці, що присутні в СКБД, а також описані методи роботи ПЗ;

- проведене тестування розробленого програмного застосунку.

Створений вебзастосунок надає користувачу можливості авторизації та реєстрації, створення особистих відео, їх перегляду, заповнення особистого профілю, слідкувати за переглядами відео, їх популярністю. Згідно з результатами тестування програма відповідає усім потребам користувача.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. "Every Online Video Platform (OVP) on the Market: A Reference List" [Electronic resource]. – Access mode: <https://www.streamingmedia.com/2011/10/every-online-video-platform-ovp-on-the-market-a-reference-list>.
2. How Live-Streaming is Going to Crush it in 2016. SocialMediaToday [Electronic resource]. – Access mode: <https://www.socialmediatoday.com/social-networks/how-live-streaming-going-crush-it-2016>.
3. Instagram Tricks and Tips (December 2021 Updates). – 8th Edition. – Black Dog Media, 2021. – 65 p.
4. Levy, S. Facebook: The Inside Story / Steven Levy. – Penguin, 2021. – 592 p.
5. Periscope: Anglesey man behind video streaming app [Electronic resource]. – Access mode: <https://www.bbc.com/news/uk-wales-32164023>.
6. TikTok. Real shorts [Electronic resource]. – Access mode: <https://apps.apple.com/app/tiktok-real-short-videos/id835599320>.
7. What is Streaming? [Electronic resource]. – Access mode: <http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-Streaming-74052.aspx>.
8. 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.
9. 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. 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/>.

12. 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>.

13. Бреннан, М: TikTok. Фабрика уваги. Історія зльоту. Хронологія / Метью Бреннан. – АСТ, 2021. – 288 с.

14. Tik Tok, a Global Music Video Platform and Social Network, Launches in Indonesia [Electronic resource]. – Access mode: https://en.prnasia.com/releases/apac/Tik_Tok_a_Global_Music_Video_Platform_and_Social_Network_Launches_in_Indonesia-187963.shtml.

15. TikTok surges past 6M downloads in the US as celebrities join the app [Electronic resource]. – Access mode: <https://www.theverge.com/2018/11/15/18095446/tiktok-jimmy-fallon-tony-hawk-downloads-revenue>.

16. Полное руководство по видеоформатам [Электронный ресурс]. – Режим доступа: <https://clipchamp.com/ru/blog/ultimate-guide-video-formats/>.

17. AMV Codec Tools [Electronic resource]. – Access mode: <https://code.google.com/archive/p/amv-codec-tools/wikis/AmvDocumentation.wiki>.

18. QuickTime File Format [Electronic resource]. – Access mode: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000052.shtml>.

19. Fernando C. N. Pereira. The MPEG-4 Book / F. Pereira. – Prentice Hall, 2002. – 392p.

20. ДСТУ ISO/IEC 14496-10:2018 Інформаційні технології. Кодування аудіовізуальних об'єктів. Частина 10. Покращене відеокодування.

21. MPEG standards [Electronic resource]. – Access mode: <https://web.archive.org/web/20100420192552/http://mpeg.chiariglione.org/standards.htm>.

22. Programme of Work [Electronic resource]. – Access mode: <https://web.archive.org/web/20131231054907/http://kikaku.itscj.ipsj.or.jp/sc29/29w42911.htm>.

23. About WebM [Electronic resource]. – Access mode: <https://www.webmproject.org/about/>.

24. SPA (Single-page application) [Electronic resource]. – Access mode: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>.

25. 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>.

26. Express [Electronic resource]. – Access mode: <https://expressjs.com/>.

27. Getting Started with MySQL [Electronic resource]. – Access mode: <https://dev.mysql.com/doc/mysql-getting-started/en/>.

28. Redis [Electronic resource]. – Access mode: <https://aws.amazon.com/ru/redis/>.

29. What is Flux? [Electronic resource]. – Access mode: <http://fluxxor.com/what-is-flux.html>.

ДОДАТОК А
Технічне завдання

Вступ

Відеохостинги наразі є досить універсальними рішеннями, що мають велику функціональність, а також мають ряд плюсів та мінусів, в залежності від того, який саме використовує користувач. Більшість вебсайтів відеохостингів мають дуже велику проблему з перевантаженням контентом, а також функціоналом, через що цілі використання таким сервісом стають досить розмитими. Необхідно зробити програмне рішення реалізації відеохостингу, а також знайти функції для більш комфортного користування сервісу користувачем.

А.1 Підстави для розробки

Підставою для розробки було завдання на дипломну кваліфікаційну роботу за темою «Програмна реалізація сервісу відеохостингу» на основі наказу №102 від 27 березня 2022 р. за Національним університетом «Запорізька Політехніка».

А.2 Призначення розробки

Програма призначена для забезпечення користувачам можливість авторизації і реєстрації, перегляду списків всіх наявних на платформі відео, перегляду відеофайлу безпосередньо в браузері, та створення власних відео.

А.3 Основні вимоги для розробки

А.3.1 Вимоги до функціональних характеристик

Програма 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.

A.3.2 Вимоги до надійності

Внесення даних має відбуватися тільки через серверну частину додатку, яка буде мати інший порт, але ту ж саму адресу. Доступ користувачів, що авторизовані і ні буде реалізовуватися через вебсайт стандартно. Користувачі повинні мати можливість змінювати свої особисті записи, але ніяк не інформацію, що повинна бути доступна тільки серверній частині, як дата створення запису або відео.

A.3.3 Вимоги до складу та параметрів технічних засобів

Апаратні вимоги до серверу:

- жорсткий диск з вільним простором – 10 Гб;
- оперативна пам'ять – 8 Гб;
- процесор: 4 ядра, 3,5 ГГц частота.

Клієнт представлений браузером користувача. Для виконання вимог до нього потрібний сучасний браузер, наприклад Google Chrome, Mozilla Firefox, Opera, з доступом до мережі інтернет.

A.4 Порядок контролю та приймання

Контроль за виконанням дипломної кваліфікаційної роботи покладається на керівника. Остаточне рішення приймається екзаменаційною комісією під час захисту роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла Video.js

```

"use strict";
const User = require("./user");
const UserVideos = require("./userVideos");

const { Model } = require("sequelize");
const { Sequelize } = require("sequelize");
const { Op } = require("sequelize");
const scopes = {
  withUser(userId) {
    return {
      include: [
        {
          association: "links",
          where: { userId },
          attributes: [],
        },
      ],
    };
  },
  withOwner() {
    return {
      include: [
        {
          model: Video,
          as: "having_access",
          required: true,
          attributes: [
            [
              Sequelize.fn(
                "GROUP_CONCAT",
                Sequelize.col("`having_access->links-`" +
>userLink`.`email`)

```

```

        ),
        "email",
    ],
    [
        Sequelize.fn(
            "GROUP_CONCAT",
            Sequelize.col("`having_access->links->userLink`.`nickname`")
        ),
        "nickname",
    ],
    [
        Sequelize.fn(
            "GROUP_CONCAT",
            Sequelize.col("`having_access->links->userLink`.`avatar`")
        ),
        "avatar",
    ],
],
include: [
    {
        association: "links",
        required: true,
        attributes: [],
        include: [
            {
                required: true,
                association: "userLink",
                attributes: [],
            },
        ],
    },
],
},
],

```

```

    },
  ],
  group: ["`having_access`.`id`"],
};
},
};

```

```

class Video extends Model {
  static init(sequelize, DataTypes) {
    return super.init(
      {
        title: DataTypes.STRING,
        description: DataTypes.STRING,
        fileHashNumber: DataTypes.STRING,
        viewCounter: DataTypes.INTEGER,
        screenSaver: DataTypes.TEXT,
      },
      {
        sequelize,
        hooks: {
          async afterCreate(attributes, options) {
            let userVideos = [];
            userVideos.push({ userId: options.userId, videoId:
attributes.id });

            await UserVideos.bulkCreate(userVideos, {
              updateOnDuplicate: ["userId", "videoId"],
            });
          },
        },
        scopes,
        modelName: "Video",
      }
    );
  }
}

```

```

    }
    static associate(models) {
      this.belongsTo(models.userVideosMap, {
        foreignKey: "id",
        targetKey: "videoId",
        as: "links",
      });
      this.belongsTo(models.Video, {
        foreignKey: "id",
        targetKey: "id",
        as: "having_access",
      });
    }
  }
}

module.exports = Video;

```

Б.2 Текст файла 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;

```

Б.3 Текст файла 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;

```

Б.4 Текст файла 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);
};

```

Б.5 Текст файлы 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);
};

```

Б.6 Текст файла auth.middleware.js

```

const jwt = require("jsonwebtoken");

module.exports = (req, res, next) => {
  if (req.method === "OPTIONS") {
    return next();
  }

```

```
}  
try {  
  const token = req.headers.token;  
  if (!token) {  
    return res.status(401).json({ message: "Unauthorized" });  
  }  
  const decoded = jwt.verify(token, "secret");  
  req.user = decoded;  
  next();  
} catch (e) {  
  res.status(401).json({ message: "Unauthorized" });  
}  
};
```

ДОДАТОК В
Слайди презентації

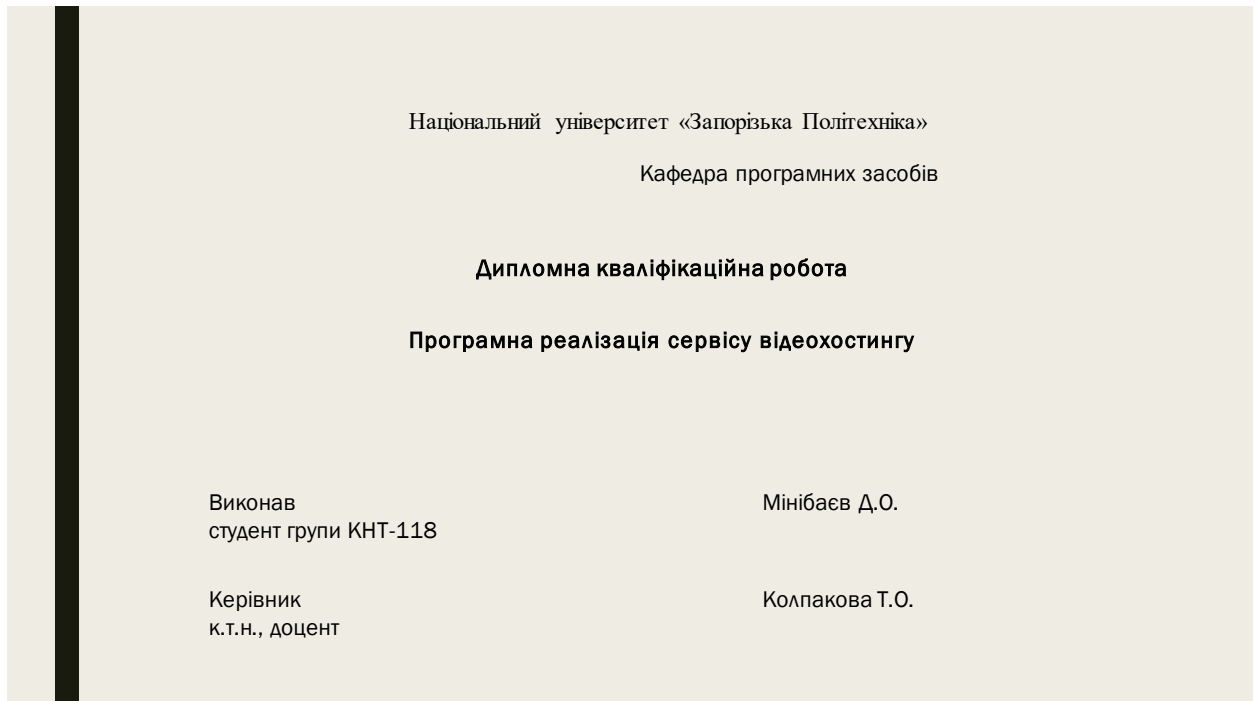


Рисунок В.1 – Слайд № 1

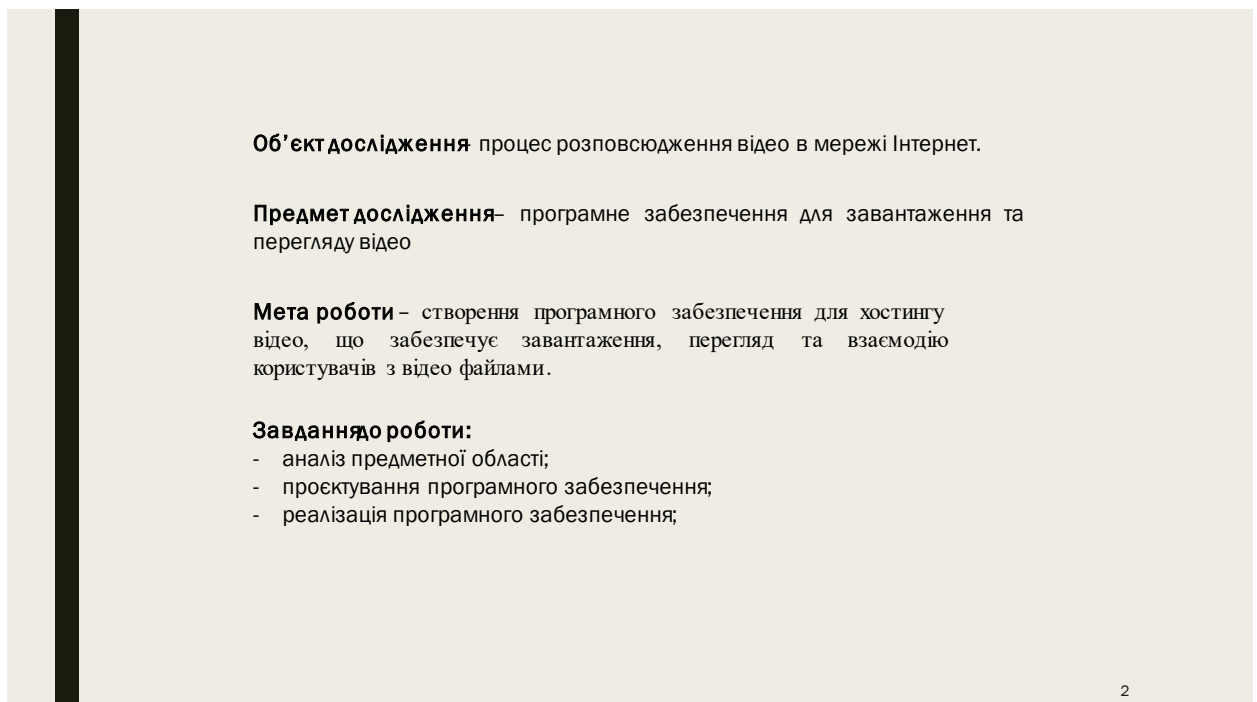
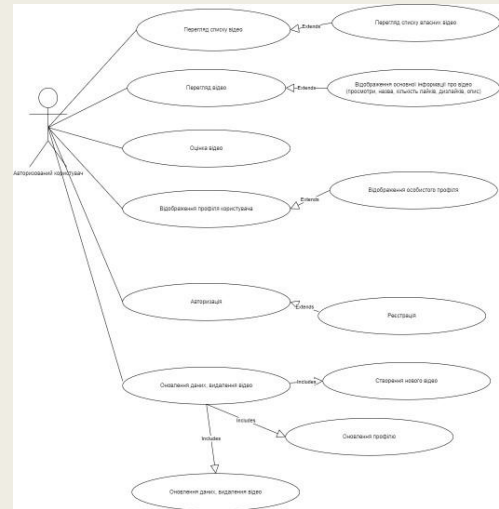
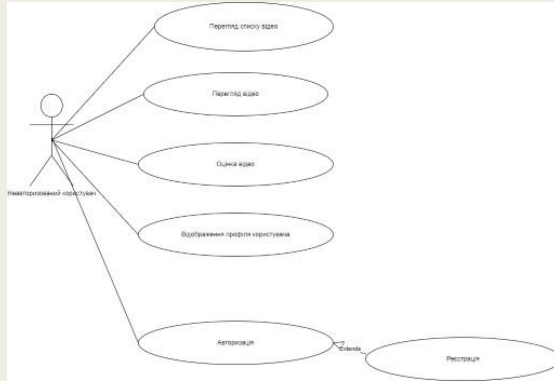


Рисунок В.2 – Слайд № 2

Діаграма прецедентів авторизованого та неавторизованого користувачів



3

Рисунок В.3 – Слайд № 3

Вибір мови програмування і фреймворків

Назва мови програмування	Python	JavaScript
Нативна підтримка роботи з CSS HTML	-	+
Фреймворки для роботи з клієнтською частиною веб додатку	Flask	Vue, Svelte, Angular
Типізація	Динамічна	Динамічна
Можливість одразу побачити результат праці над програмою	-	+

4

Рисунок В.4 – Слайд № 4

Назва мови програмування	C#	Python	JavaScript
Простота в використанні	-	-	+
Фреймворки для розробки серверної частини додатку	ASP.NET	Django	Nest.js, Express.js, Koa.js
Типізація	Статична	Динамічна	Динамічна
Наявність многопоточності	+	+	+ (можливість створення додаткових допоміжних потоків)

5

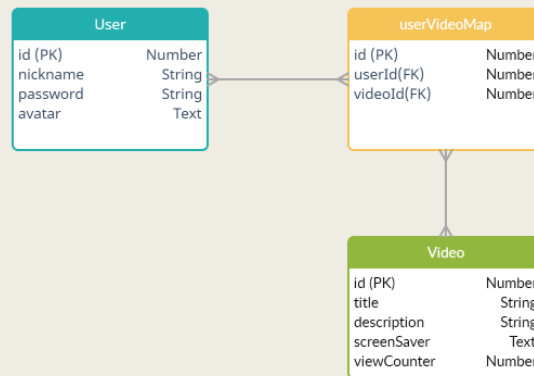
Рисунок В.5 – Слайд № 5

Назва фреймворку	Vue	Angular	Svelte
Типізація	Динамічна	Статична	Динамічна
Інтегроване глобальне сховище	+	-	+
Реактивність	+	-	+
Схожість на класичну мову JS	-	-	+
Простота використання	+	-	-

6

Рисунок В.6 – Слайд № 6

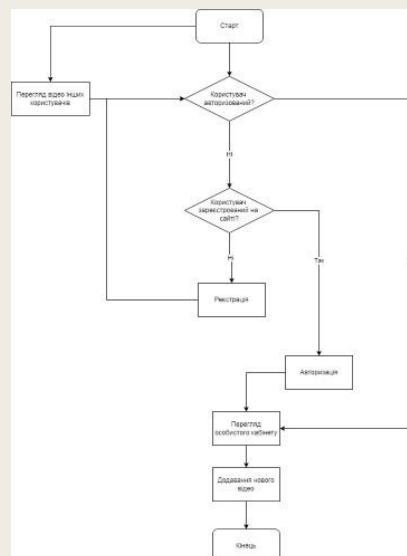
Схема БД



7

Рисунок В.7 – Слайд № 7

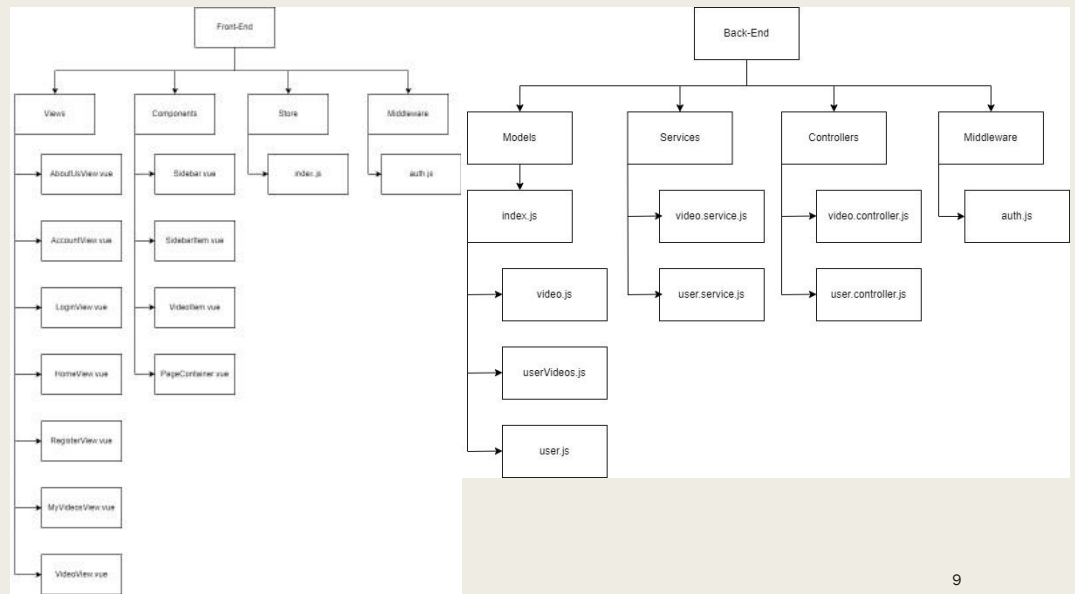
Алгоритм функціонування програми



8

Рисунок В.8 – Слайд № 8

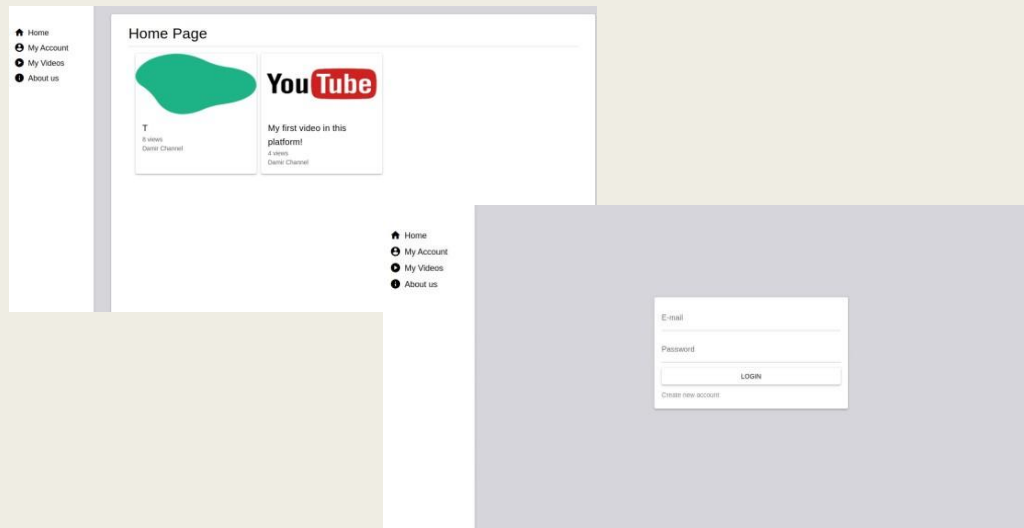
Структура клієнтської та серверної частин



9

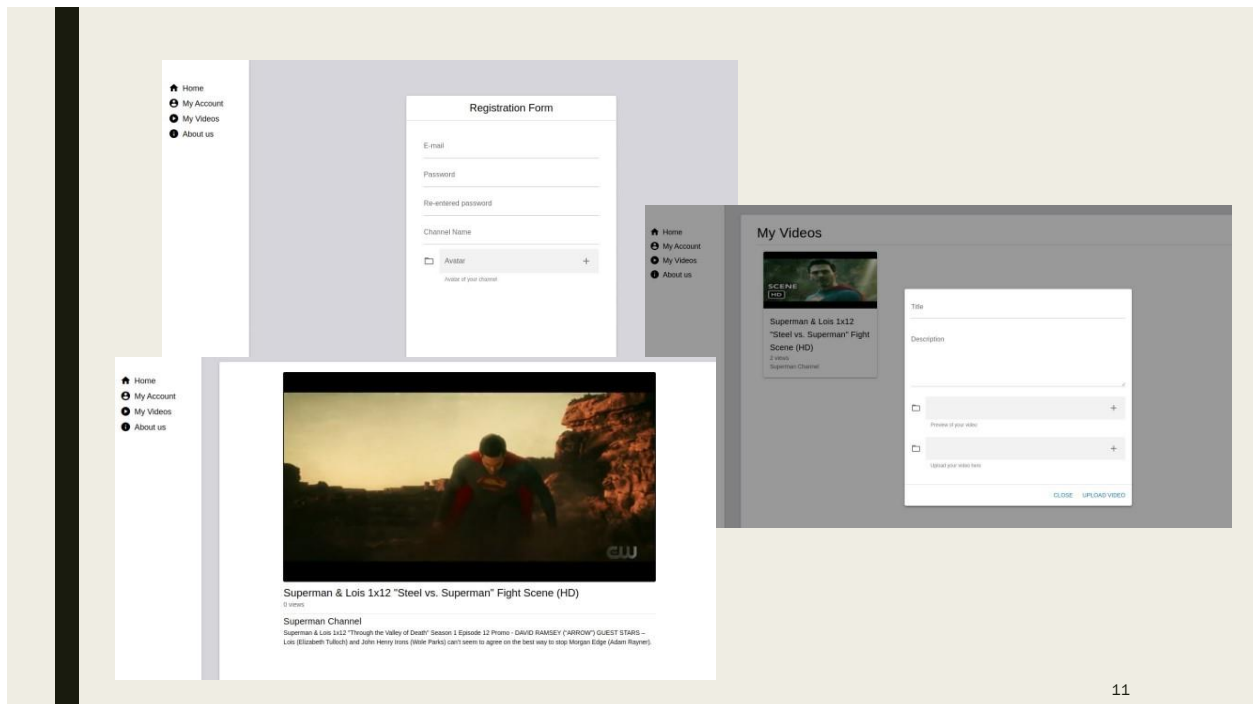
Рисунок В.9 – Слайд № 9

ІНТЕРФЕЙСИ ПРОГРАМИ



10

Рисунок В.10 – Слайд № 10



11

Рисунок В.11 – Слайд № 11

Висновки

Виконання дипломної кваліфікаційної роботи призвело до отримання наступних результатів:

- виконано аналіз проблеми сервісів відеохостингу, задля чого було проведено дослідження специфіки роботи відеохостингів, їх видів, а також проведено порівняння та огляд програмних аналогів;
- виконано розробку програмної архітектури сайту, під час якої було визначено функціональні вимоги до програмного забезпечення, змодельовано роботу з програмою на основні діаграм прецедентів, обґрунтовано вибір мови програмування JavaScript, а також технологій Vue.js та Express.js для реалізації клієнт-серверної архітектури, а також СКБД MySQL для розробки програмного забезпечення, визначено структуру БД для організації роботи;
- реалізоване програмне забезпечення, в результаті чого був описаний алгоритм роботи програми у вигляді блок-схеми, схеми структури клієнт-серверного додатку, описано моделі і таблиці, що присутні в СКБД, а також описані методи роботи ПЗ;
- проведено тестування розробленого програмного застосунку.

12

Рисунок В.12 – Слайд № 12