МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ДОДАТКУ ОБМІНУ ВІДЕО ДЛЯ ГЕЙМЕРІВ ЗАСОБАМИ ANGULAR ТА FIREBASE»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Г.Є. Жутовський
(ініціали та прізвище)
Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Жутовському Глібу Євгеновичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка додатку обміну відео для геймерів засобами Angular та Firebase

керівник роботи Лісняк Андрій Олександрович, к.ф.-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	29.01.2024	
2.	Збір вихідних даних.	19.02.2024	
3.	Обробка методичних та теоретичних	11.03.2024	
	джерел.		
4.	Розробка першого та другого розділу.	13.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ Г.Є. Жутовський
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка додатку обміну відео для геймерів засобами Angular та Firebase»: 67 с., 36 рис., 1 табл., 15 джерел, 4 додатки.

ANGULAR, API, FIREBASE, FRAMEWORK, REALTIME DATABASE, RXJS, TYPESCRIPT, UML.

Об’єкт дослідження – додаток, інструменти для взаємодії Angular та Firebase, інструменти для роботи з відео.

Мета роботи – розробити додаток обміну відео для геймерів.

Методи дослідження – моделювання, проєктування, програмний, аналітичний.

У роботі розглянуто проєктування та реалізацію вебдодатку для обміну відео з використанням Angular та Firebase. Застосовано UML для візуалізації системи, зокрема діаграми варіантів використання, діяльності, послідовності та розгортання.

Система надає можливість завантажувати відео, встановлювати налаштування, керувати завантаженими відео. Реалізовано реєстрацію/автентифікацію користувачів, зберігання відео та метаданих у Firebase. Впроваджено RxJS для реактивного програмування та бібліотеку ffmpeg.wasm для обробки відео. Проведено юніт- та інтеграційне тестування.

Таким чином, розроблений вебдодаток забезпечує зручний інтерфейс для обміну відеоматеріалами, дозволяючи користувачам легко завантажувати, переглядати та керувати своїми відео. Застосування Angular, Firebase та сучасних підходів до розробки програмного забезпечення сприяло створенню ефективної та масштабованої системи.

SUMMARY

Bachelor’s qualifying paper «Development of a Video Sharing Application for Gamers Using Angular and Firebase»: 67 pages, 36 figures, 1 table, 15 references, 4 supplements.

ANGULAR, API, FIREBASE, FRAMEWORK, REALTIME DATABASE, RXJS, TYPESCRIPT, UML.

The object of the study is the application, tools for Angular and Firebase interaction, tools for working with video.

The aim of the study is to develop a video sharing application for gamers.

The methods of research are modeling, design, programming, analytical.

The paper presents a discussion of the design and implementation of a web application for video sharing, developed using Angular and Firebase. UML is employed to provide a visual representation of the system, including use case diagrams, activities, sequence, and deployment.

The system enables users to upload videos, set settings, and manage uploaded videos. User registration/authentication, video and metadata storage in Firebase were implemented. RxJS was implemented for reactive programming, and the ffmpeg.wasm library was employed for video processing. Unit and integration testing were conducted.

As a result, the developed web application provides a user-friendly interface for video sharing, allowing users to easily upload, view, and manage their videos. The use of Angular, Firebase, and modern approaches to software development contributed to the creation of an efficient and scalable system.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Технічне завдання 10
1.1 Терміни та визначення 10
1.1.1 Загальні терміни 10
1.1.2 Технічні терміни 10
1.2 Функціональні вимоги 11
1.2.1 Призначення і цілі створення системи 11
1.2.2 Загальні функціональні можливості системи 11
1.3 Нефункціональні вимоги 12
1.3.1 Інтерфейс користувача 12
1.3.2 Підтримка браузерів 12
1.3.3 Вимоги до продуктивності 12
1.3.4 Вимоги до безпеки 13
1.4 Опис предметної області 13
1.5 Опис системи 14
1.6 Огляд аналогів 14
2 Проєктування 17
2.1 Використання UML під час розробки системи 17
2.2 Діаграма варіантів використання 18
2.2.1 Опис варіантів використання 21
2.3 Діаграма діяльності 27
2.4 Діаграма послідовності 30
2.5 Діаграма розгортання 31
3 Реалізація та тестування 34

3.1 Опис інструментів розробки 34
3.2 Основні класи системи 34
3.3 Налаштування Firebase 35
3.4 Тестування проєкту 38
3.5 Керівництво користувача 40
3.5.1 Рівень підготовки користувача 40
3.5.2 Реєстрація в системі 40
3.5.3 Вхід/Вихід до/з системи 43
3.5.4 Завантаження відео 44
3.5.5 Керування відео 47
Висновки 51
Перелік посилань 52
Додаток А UploadComponent 54
Додаток Б ClipService 59
Додаток В FfmpegService 63
Додаток Г Angular-тест 66

ВСТУП

Відеоігри стали невід'ємною частиною сучасної культури та дозвілля. Щоденно мільйони людей у всьому світі захоплюються іграми, демонструючи свої навички та досягнення. Проте, часто геймери стикаються з проблемою обміну відеороликами зі своїми друзями та спільнотою. Існуючі платформи для обміну відео не завжди задовольняють специфічні потреби гравців, що створює незручності та обмежує можливості для спілкування та співпраці.

Геймери шукають зручні та орієнтовані на їхні потреби платформи для обміну відеоматеріалами, пов'язаними з іграми. Наявність такого додатку дозволить задовольнити цю потребу та забезпечити комфортне середовище для спільноти гравців.

Додаток обміну відео дозволить геймерам легко ділитися своїми досягненнями, цікавими моментами гри та стратегіями, сприятиме розвитку спільноти, обміну досвідом та навчанню нових навичок. Можливість інтегрувати додаток з популярними соціальними платформами дозволить геймерам легко поширювати свої відео серед друзів та послідовників, збільшуючи залученість та популярність.

Виходячи з цього, було вирішено створити додаток, який був би простий у використанні та в той час безпечний та функціональний.

Актуальність дослідження: актуальність теми зумовлена потребою створення спеціалізованої платформи обміну відео на ігрову тематику.

З огляду на це, можна виділити наступні цілі і задачі нашого дослідження:

Мета: розробити додаток обміну відео для геймерів.

Задачі:

- сформувати вимоги до додатку;
- спроєктувати та побудувати архітектуру додатку;
- реалізувати додаток обміну відео для геймерів;
- протестувати роботу додатку.

Об’єкт дослідження: процес розробки додатку обміну відео для геймерів, інструменти для роботи з відео.

Предмет дослідження: фреймворк Angular та сервіс Firebase.

Методи дослідження: моделювання, проєктування, програмний, аналітичний.

Перший розділ присвячено збору та аналізуванню вимог до додатку, огляду подібних систем і опису додатку.

У другому розділі розглянуто етапи проєктування додатку, наведено детальний опис прецедентів та побудовано діаграми.

Третій розділ присвячено реалізації та тестуванню роботи додатку, наведено керівництво користувача, яке описує процес роботи з додатком обміну відео для геймерів.

1 ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Терміни та визначення

1.1.1 Загальні терміни

Система – додаток обміну відео для геймерів, створений засобами Angular та Firebase.

Angular – це платформа і фреймворк для розробки односторінкових вебзастосунків.

Firebase – це комплексна хмарна платформа для розробки мобільних та вебдодатків, яку розробляє і підтримує Google. Вона пропонує широкий спектр інструментів та сервісів для полегшення розробки, розгортання та масштабування додатків.

ДВІ – Діаграма Варіантів Використання чи Use Case Diagram.

ДД – Діаграма Діяльності.

ДП – Діаграма Послідовності.

ДР – Діаграма Розгортання.

Гість – людина, яка не зареєстрована або неавторизована в системі.

Користувач – людина, яка зареєстрована в системі та може взаємодіяти з функціоналом.

1.1.2 Технічні терміни

Cloud Firestore – хмарна NoSQL база даних для зберігання та синхронізації даних додатків у реальному часі.

Cloud Storage – об’єктне сховище для зберігання файлів, таких як

зображення, аудіо, відео тощо.

Framework – це програмна платформа або конкретна реалізація, що визначає структуру програмного забезпечення. Фреймворк надає базову функціональність і набір інструментів та бібліотек для розробки додатків в межах певної системи.

Компонент – структурна одиниця Angular.

1.2 Функціональні вимоги

1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість обміну відео засобами Angular та Firebase.

Експлуатаційне призначення системи – система може експлуатуватися користувачами та гостями системи.

Мета створення системи – розробка додатку обміну відео.

1.2.2 Загальні функціональні можливості системи

Система має надавати гостям такі можливості:

- переглядати список завантажених відео;
- переглядати відео;
- вхід до системи;
- реєстрація в системі.

Система має надавати користувачам такі можливості:

- додати/обирати/видаляти/переглядати відео;
- задавати назву/мініатюру відео;

- редагувати назву відео;
- копіювати посилання на відео;
- сортувати відео;
- вихід з системи;
- усі права гостя.

1.3 Нефункціональні вимоги

1.3.1 Інтерфейс користувача

Інтерфейс користувача системи повинен бути адаптивним та коректно відображатися на будь-яких пристроях, забезпечуючи зручність використання для різних категорій користувачів.

1.3.2 Підтримка браузерів

Система має підтримувати останні версії популярних веббраузерів, таких як Mozilla Firefox, Google Chrome, Safari, Microsoft Edge та Opera, гарантуючи сумісність та доступність для широкого кола користувачів.

1.3.3 Вимоги до продуктивності

Система повинна завантажувати сторінки не довше ніж за 1 секунду, а завантаження та обробка відео не повинна впливати на загальну продуктивність системи.

1.3.4 Вимоги до безпеки

Система повинна обмежувати доступ до своїх функцій лише для авторизованих користувачів, захищаючи дані та функціональність від несанкціонованого використання.

Система не повинна надавати можливість завантажувати відео інших форматів, крім вказаних, та які важать більше 20 Мб.

1.4 Опис предметної області

Предметною областю є розробка додатку обміну відео для геймерів. Дана система повинна надавати користувачам можливість завантажувати власні відео та взаємодіяти з ними.

Процес взаємодії з системою залежить від прав користувача. Для гостя все обмежується переглядом завантажених відео та можливістю створити профіль. Після цього він, як користувач, може завантажувати відео та взаємодіяти з ними та з відео інших користувачів.

Процес завантаження відео відбувається наступним чином. Користувач реєструється або авторизується в системі, та натискає на пункт меню «Upload». Система відображає інтерфейс завантаження відео.

Користувач обирає відео на пристрої та натискає кнопку «Вибрати», система опрацьовує відео та пропонує користувачу вибрати мініатюру і назву відео.

Після вибору запропонованих налаштувань, користувач натискає на кнопку «Publish», система завантажує відео в Cloud Storage та зберігає дані до Cloud Firestore.

Після успішного завантаження, перенаправляє користувача на сторінку перегляду завантаженого відео.

1.5 Опис системи

Сьогодні індустрія відеоігор переживає справжній бум, залучаючи мільйони нових прихильників з усього світу. Геймери активно діляться захопливими моментами з улюблених ігор, своїми досягненнями та враженнями, створюючи величезний пласт відеоконтенту.

Задовольнити зростаючий попит на зручну платформу для обміну такими відео покликана наша система. Вона надає можливість переглядати список завантажених відео, переглядати самі відеоролики, реєструватися та входити в обліковий запис. Зареєстровані користувачі також можуть завантажувати, видаляти та сортувати відео, задавати їм назви та мініатюри, редагувати назви, копіювати посилання для поширення контенту.

В перспективі планується створення спільнот геймерів та інструментів монетизації. Розширені функції максимально задовольнять потреби цільової аудиторії та забезпечать зручний обмін високоякісним ігровим контентом на єдиній спеціалізованій платформі.

1.6 Огляд аналогів

Існує декілька відомих платформ, які можна розглядати як аналоги для системи обміну відео для геймерів:

- YouTube Gaming – окремий розділ популярного відеохостингу YouTube, присвячений виключно ігровому контенту (дозволяє стрімити ігровий процес, завантажувати відео, підписуватись на канали) [15];
- Twitch – одна з найбільших платформ для стримінгу відеоігор (має великий функціонал для прямих трансляцій, запису відео, спілкування зі спільнотою) [12];
- Facebook Gaming – ігровий розділ соціальної мережі Facebook,

інтегрований з основною платформою (пропонує стриминг, завантаження відео, створення спільнот) [7];

- Mixer – колишня ігрова платформа від Microsoft, закрита у 2020 році (мала фокус на прямих трансляціях від геймерів та інтерактивних функціях) [11];
- DLive – блокчейн-платформа для стрімінгу відеоігор з можливістю монетизації контенту через внутрішню криптовалюту [6].

На основі опису аналогів, відобразимо порівняння з нашою системою (табл. 1.1).

Таблиця 1.1 – Порівняння аналогів

Функція	**Наша система**	**YouTube Gaming**	**Twitch**	**Facebook Gaming**	**DLive**
Завантаження відео	Так	Так	Так	Так	Так
Прямі трансляції	Ні	Так	Так	Так	Так
Створення спільнот	Базове	Повноцінні спільноти	Повноцінні спільноти	Інтеграція з групами Facebook	Ні
Монетизація контенту	Ні	Реклама, членські внески, суперчати	Біти, підписки, реклама	Реклама, зірки	Внутрішня крипто-валюта
Редагування відео	Ні	Просте редагування	Просте редагування	Ні	Ні
Інтеграція з іграми	Базова	Ні	Повна (overlay, захоплення екрану)	Ні	Ні

Продовження табл. 1.1

Функція	**Наша система**	**YouTube Gaming**	**Twitch**	**Facebook Gaming**	**DLive**
Пошук і рекомендації	Базовий	Розвинений	Розвинений	Базований на Facebook	Базовий

Порівнюючи нашу систему з існуючими аналогами, можна виділити такі її переваги:

- простота і зрозумілий інтерфейс для тих, хто просто хоче завантажувати відео, без зайвих функцій;
- базова інтеграція зі створенням спільнот – спрощений варіант порівняно з повноцінними Communities;
- потенційна можливість тісної інтеграції з популярними іграми для прямого захоплення контенту;
- зосередженість виключно на відеоконтенті геймерів без відволікання на інші формати;
- відсутність реклами та підписок на ранній стадії для залучення користувачів.

Водночас, розширюючи функціонал у майбутньому, наша система могла б запозичити передову монетизацію, редагування та пошук інших провідних платформ, але з фокусом саме на ігровій аудиторії.

2 ПРОЄКТУВАННЯ

2.1 Використання UML під час розробки системи

Під час розробки програмного забезпечення уніфікована мова моделювання (UML) відіграє важливу роль, оскільки дозволяє візуально представляти різні аспекти системи на різних рівнях абстракції. Використання UML допомагає зробити процес розробки більш структурованим, зрозумілим і керованим.

За допомогою UML можна створювати діаграми вимог, що дозволяє візуалізувати і документувати функціональні вимоги до системи, забезпечуючи спільне розуміння вимог між замовниками, аналітиками та розробниками. Крім того, UML пропонує різноманітні діаграми для моделювання архітектури системи на різних рівнях абстракції, спрощуючи комунікацію між учасниками проєкту та забезпечуючи узгодженість дизайну. Діаграми поведінки UML, такі як діаграми послідовності, діаграми комунікації та діаграми станів, дозволяють візуально представляти динамічну поведінку системи, послідовності взаємодій між об’єктами та переходи між станами [13].

UML сприяє повторному використанню коду, оскільки дозволяє моделювати компоненти та їх взаємозв’язки, полегшуючи виявлення можливостей для повторного використання коду та проєктування модульної архітектури. Діаграми UML є стандартизованими і зрозумілими для всіх учасників проєкту, що полегшує документування та передачу знань про систему, особливо корисно для підтримки та розширення системи в майбутньому.

Існує безліч інструментів, що підтримують UML, таких як Visual Paradigm, StarUML, ArgoUML та інші, які полегшують створення, редагування та підтримку діаграм UML, а також можуть генерувати код або документацію на основі моделей. Однак слід пам’ятати, що UML є лише засобом візуалізації та

документування, і його використання не гарантує створення якісного програмного забезпечення. Важливо поєднувати UML з належними практиками проєктування, кодування та тестування, щоб забезпечити успішну розробку системи.

2.2 Діаграма варіантів використання

Діаграма варіантів використання (use case diagram) є одним з найважливіших артефактів UML, який використовується для візуалізації та документування функціональних вимог системи з точки зору користувачів та акторів, які взаємодіють з нею. Ця діаграма допомагає зрозуміти основні функції системи та їх взаємозв’язки [13].

На діаграмі варіантів використання зображуються актори (люди, системи або зовнішні сутності, які взаємодіють з системою) та самі варіанти використання (дії або послідовності дій, які виконуються системою для надання певної функціональності). Варіанти використання зазвичай представлені у вигляді овальних фігур, а актори – у вигляді фігурок людини або іншого графічного елемента.

Діаграма варіантів використання дозволяє виконувати такі завдання: визначати межі системи, описувати функціональні вимоги з точки зору користувача, ідентифікувати акторів та їх ролі, візуалізувати взаємозв’язки між варіантами використання, а також полегшувати комунікацію між зацікавленими сторонами та розробниками.

При створенні діаграми варіантів використання важливо дотримуватися певних принципів та правил, таких як: використання простої та зрозумілої термінології, чітке визначення меж системи, уникнення надмірної деталізації або надмірної абстракції, відображення основних потоків подій та альтернативних сценаріїв [13].

Діаграма варіантів використання є корисним інструментом на початкових

етапах розробки програмного забезпечення, оскільки вона допомагає зрозуміти та узгодити вимоги з усіма зацікавленими сторонами. Проте вона не замінює детальної документації вимог та технічних специфікацій, а радше слугує візуальним доповненням для кращого розуміння загальної картини системи.

На рисунку 2.1 представлена діаграма варіантів використання додатку обміну відео.

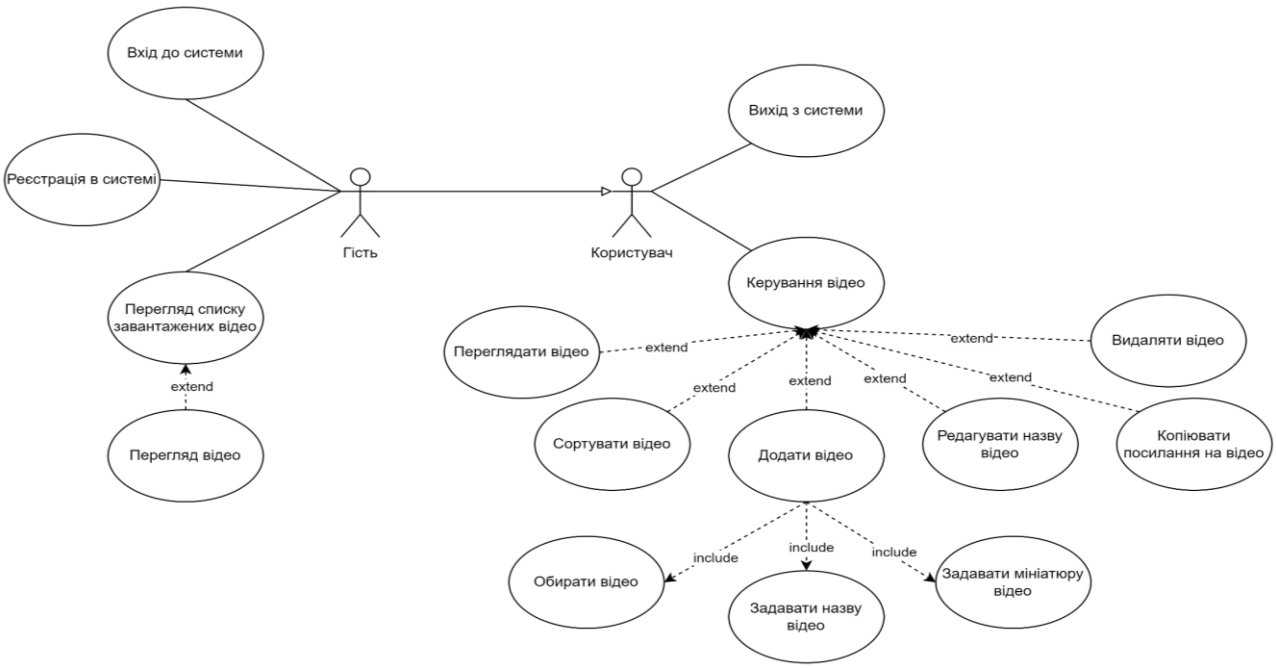

Рисунок 2.1 – Діаграма варіантів використання

На діаграмі представлено акторів «Гість» та «Користувач», кожен з яких може взаємодіяти з системою на різних рівнях. Також актор «Користувач» включає в себе прецеденти актора «Гість».

Виділено 1 основний варіант використання – «Додати відео». Після того, як користувач переходить до пункту меню «Upload», система відображає інтерфейс завантаження нового відео. Користувач може додати відео або шляхом перетягування відеофайлу в відповідне поле, або натиснувши на кнопку «Вибрати», система відобразить оглядач пристрою користувача, в якому він зможе вибрати необхідне відео. Система обробляє відео та надає можливість користувачу задати мініатюру і назву відео. Після цих налаштувань користувач

натискає на кнопку «Publish», система завантажує відео в Cloud Storage та зберігає дані до Cloud Firestore. Після успішного завантаження система перенаправляє користувача на сторінку перегляду завантаженого відео.

Варіанти використання визначають функціональні можливості системи та способи їх використання кінцевими користувачами або зовнішніми акторами (див. рис. 2.2, 2.3). Кожен варіант використання представляє певну послідовність дій, яку може виконати актор для досягнення конкретної мети або результату в рамках взаємодії з системою. Варіанти використання є важливим інструментом для визначення та документування вимог до функціональних можливостей програмної системи. Вони описують різні способи взаємодії користувачів або зовнішніх сутностей (акторів) з системою для досягнення конкретних цілей чи результатів.

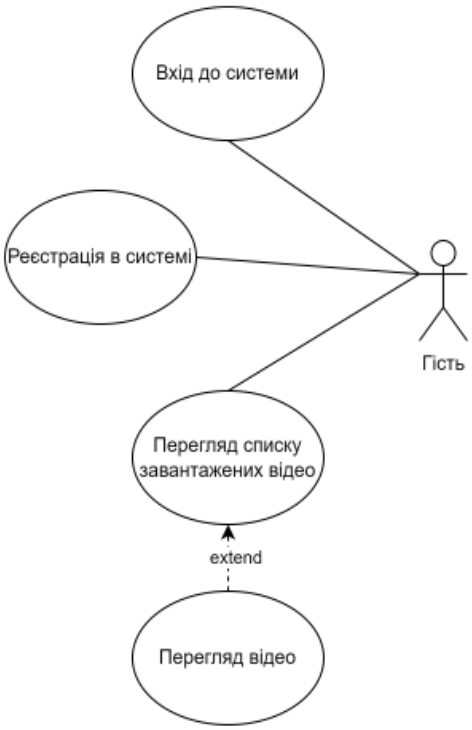

Рисунок 2.2 – Діаграма варіантів використання «Гість»

Кожен варіант використання представляє собою послідовність кроків або дій, які актор може виконати в межах системи для реалізації певного сценарію. Ці сценарії відображають різні способи застосування функціоналу системи

акторами для вирішення їхніх завдань або задоволення потреб. Варіанти використання забезпечують зручний спосіб моделювання та документування вимог до поведінки системи з точки зору її користувачів або зовнішніх сутностей. Таким чином, варіанти використання відображають різні сценарії використання функціоналу системи, забезпечуючи засоби для моделювання та документування вимог до поведінки системи з точки зору акторів, які нею оперують.

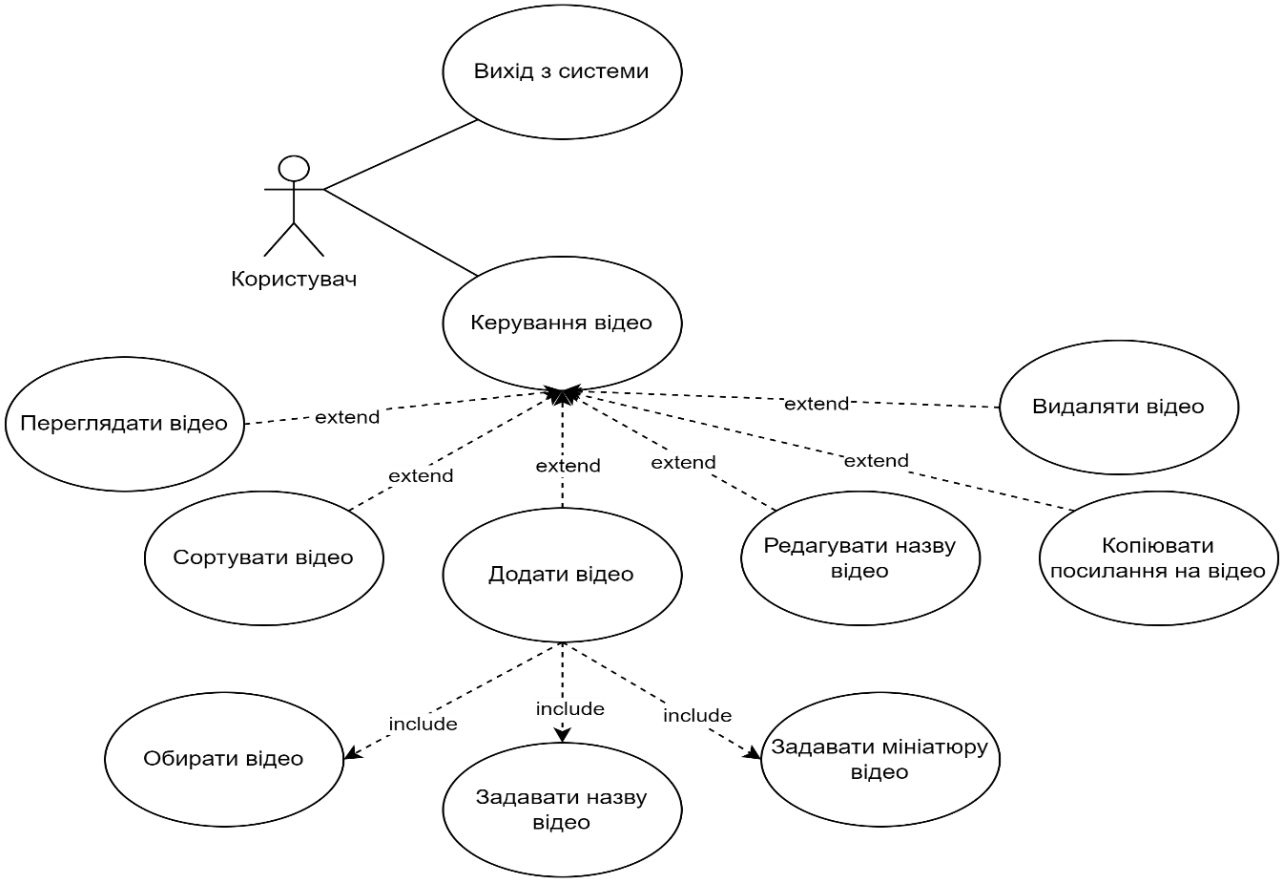

Рисунок 2.3 – Діаграма варіантів використання «Користувач»

2.2.1 Опис варіантів використання

Прецедент «Реєстрація в системі».

Призначення: даний варіант використання надає можливість гостю зареєструватися в системі.

Основний потік подій: даний варіант використання починає виконуватися, коли гість натискає на пункт меню «Login/Register» та обирає пункт «Register». Система відображає форму реєстрації. Після введення необхідних даних, гість натискає на кнопку «Submit», система створює нового користувач та зберігає його параметри.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, гість може повторно ввести дані.

Виняткова ситуація 2: такий користувач вже зареєстрований – система відображає повідомлення про помилку, гість може змінити дані.

Виняткова ситуація 3: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Вхід до системи».

Призначення: даний варіант використання надає можливість гостю входити до системи.

Основний потік подій: даний варіант використання починає виконуватися, коли гість натискає на пункт меню «Login/Register» та обирає пункт «Login». Система відображає форму авторизації. Після введення пошти та паролю, гість натискає на кнопку «Submit», система відображає додаткові функції та інструментарій.

Передумова: перед початком виконання даного варіанта використання гість повинен зареєструватися в системі.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, гість може повторно ввести дані.

Виняткова ситуація 2: такого користувача не існує – система відображає повідомлення про помилку, гість може змінити дані.

Виняткова ситуація 3: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Вихід з системи».

Призначення: даний варіант використання надає можливість користувачу виходити з системи.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на пункт меню «Logout». Система видаляє дані про сесію користувача та відображає головну сторінку.

Прецедент «Керування відео».

Призначення: даний варіант використання надає можливість користувачу керувати власними відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на пункт меню «Manage». Система відображає інтерфейс керування власними відео.

Передумова: перед початком виконання даного варіанта використання користувач повинен увійти до системи.

Виняткова ситуація 1: сесія користувача вичерпана – система відображає головну сторінку.

Прецедент «Додати відео».

Призначення: даний варіант використання надає можливість користувачу додавати нові відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач або натискає на кнопку «Upload» на сторінці керування відео, або натискає на пункт меню «Upload» навігаційної панелі. Система відображає форму завантаження нового відео. Користувач може додати відео або шляхом перетягування відеофайлу в поле завантаження файлів, або натиснувши на кнопку «Вибрати», система відобразить оглядач пристрою користувача, в якому він зможе вибрати відео. Після вибору, система обробляє відео та надає можливість користувачу задати мініатюру і назву відео. Після цих налаштувань користувач натискає на кнопку «Publish», система завантажує відео в Cloud Storage та зберігає дані до Cloud Firestore. Після успішного завантаження система перенаправляє користувача на сторінку перегляду завантаженого відео.

Передумова: перед початком виконання даного варіанта використання користувач повинен увійти до системи.

Виняткова ситуація 1: неприпустимий формат – система відображає

повідомлення про помилку, користувач може обрати відео знов.

Виняткова ситуація 2: неприпустимий розмір – система відображає повідомлення про помилку, користувач може обрати відео знов.

Виняткова ситуація 3: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Вибрати відео».

Призначення: даний варіант використання надає можливість користувачу вибрати відео для завантаження.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає кнопку «Вибрати» на сторінці завантаження відео. Система відображає оглядач файлів на пристрої користувача. Після вибору файлу, користувач натискає кнопку «Вибрати», система завантажує на оброблює відео.

Передумова: перед початком виконання даного варіанта використання користувач повинен знаходитися на сторінці завантаження відео.

Виняткова ситуація 1: неприпустимий формат – система відображає повідомлення про помилку, користувач може обрати відео знов.

Виняткова ситуація 2: неприпустимий розмір – система відображає повідомлення про помилку, користувач може обрати відео знов.

Виняткова ситуація 3: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Задати назву відео».

Призначення: даний варіант використання надає можливість користувачу задати назву відео при завантажені.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вибрав відеофайл з пристрою. Система відображає інтерфейс налаштування параметрів відео. Користувач може або залишити поточну назву відео, або змінити.

Передумова: перед початком виконання даного варіанта використання користувач повинен обрати відео з пристрою.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Прецедент «Задати мініатюру відео».

Призначення: даний варіант використання надає можливість користувачу задати мініатюру відео при завантажені.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вибрав відеофайл з пристрою. Система відображає інтерфейс налаштування параметрів відео. Користувач може обрати одну з трьох мініатюр.

Передумова: перед початком виконання даного варіанта використання користувач повинен обрати відео з пристрою.

Прецедент «Переглядати відео».

Призначення: даний варіант використання надає можливість користувачу переглянути власне відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на назву відео на сторінці керування відео. Система відображає плеєр для відтворення відео та його артефакти. Користувач може подивитися відео.

Передумова: перед початком виконання даного варіанта використання користувач повинен завантажити принаймні одне відео.

Прецедент «Сортувати відео».

Призначення: даний варіант використання надає можливість користувачу сортувати відео за датою завантаження.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вибирає або фільтр «Oldest Uploads», або фільтр «Recent Uploads». Система сортує відео за вказаними параметрами.

Передумова: перед початком виконання даного варіанта використання користувач повинен завантажити принаймні два відео.

Прецедент «Редагувати назву відео».

Призначення: даний варіант використання надає можливість користувачу редагувати назву завантаженого відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на піктрограму олівця відповідного відео. Система відображає форму зміни назви відео. Після зміни назви, користувач натискає на кнопку «Update». Система оновлює назву відео.

Передумова: перед початком виконання даного варіанта використання користувач повинен завантажити принаймні одне відео.

Виняткова ситуація 1: невалідні дані – система відображає повідомлення про помилку, користувач може повторно ввести дані.

Виняткова ситуація 2: помилка сервера – система відображає відповідне повідомлення.

Прецедент «Копіювати посилання на відео».

Призначення: даний варіант використання надає можливість користувачу скопіювати посилання на відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на кнопку «Copy link» відповідного відео. Система додає посилання до буфера обміну. Користувач може поділитися посиланням.

Передумова: перед початком виконання даного варіанта використання користувач повинен завантажити принаймні одне відео.

Прецедент «Видалити відео».

Призначення: даний варіант використання надає можливість користувачу видалити власне відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає на піктограму кошика відповідного відео. Система видаляє відео.

Передумова: перед початком виконання даного варіанта використання повинно бути створено принаймні один сертифікат.

Виняткова ситуація 1: помилка сервера – система сповіщає про це, і відео не видаляється.

Прецедент «Перегляд списку завантажених відео».

Призначення: даний варіант використання надає можливість користувачу

або гостю переглянути список усіх завантажених відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач або гість переходить на головну сторінку. Система відображає список завантажених відео.

Виняткова ситуація 1: помилка сервера – система сповіщає про це, і список відео не відображається.

Прецедент «Перегляд відео».

Призначення: даний варіант використання надає можливість користувачу або гостю переглянути обране відео.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач або гість натискають на назву відео. Система відображає плеєр відтворення відео та його артефакти.

Передумова: перед початком виконання даного варіанта використання повинно бути завантажено принаймні одне відео.

2.3 Діаграма діяльності

Діаграма діяльності в UML є потужним інструментом для моделювання бізнес-процесів, робочих потоків і алгоритмів. Вона дозволяє візуально представити послідовність дій або діяльностей, що виконуються в рамках певного процесу, а також точки прийняття рішень, паралельні потоки, цикли та винятки. Основними перевагами використання діаграм діяльності є можливість чітко візуалізувати логіку процесу, покращити комунікацію між зацікавленими сторонами та розробниками, виявити потенційні проблеми або вузькі місця у процесі на ранніх стадіях розробки.

На діаграмі діяльності використовуються різні графічні елементи, такі як активності, переходи, вирішувачі, початкові та кінцеві вузли, потоки керування та примітки. Активності представляють окремі дії або підпроцеси, а переходи показують порядок їх виконання. Вирішувачі використовуються для

моделювання точок прийняття рішень, де процес може розгалужуватися залежно від певних умов. Початкові та кінцеві вузли позначають відповідно початок та завершення процесу. Потоки керування демонструють послідовність переходів між активностями та вузлами, показуючи порядок виконання процесу. Примітки можуть бути додані для надання додаткової інформації або пояснень [13].

Діаграми діяльності можуть бути використані на різних етапах розробки програмного забезпечення, включаючи аналіз вимог, проєктування та реалізацію. На етапі аналізу вимог вони допомагають візуалізувати та узгодити бізнес-процеси з зацікавленими сторонами. На етапі проєктування вони використовуються для моделювання поведінки компонентів системи та алгоритмів. Під час реалізації діаграми діяльності можуть слугувати основою для розробки коду та тестування.

На рисунках 2.4 – 2.5 наведено діаграму діяльності прецеденту «Додати відео».

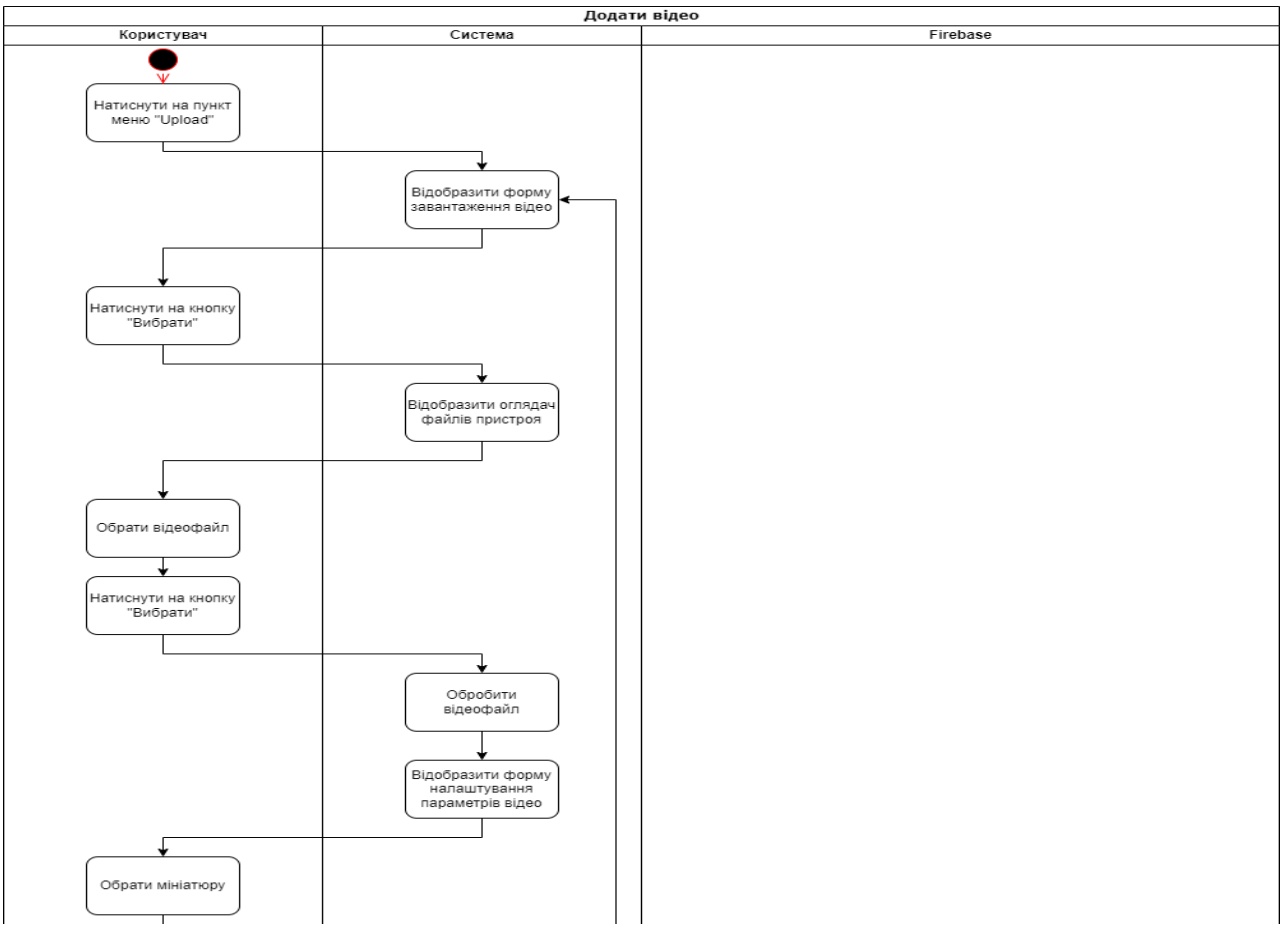

Рисунок 2.4 – Діаграма діяльності (1 частина)

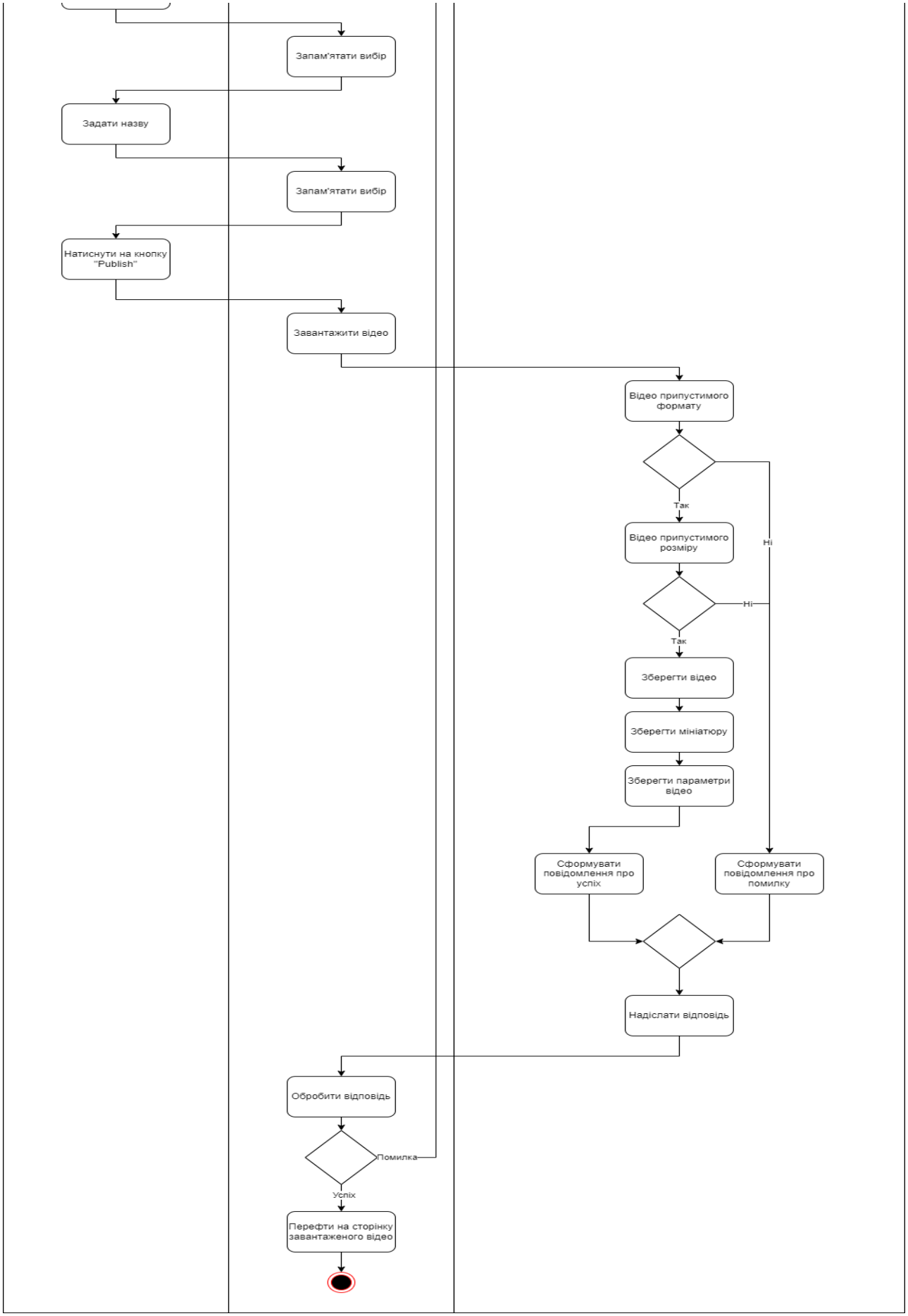

Рисунок 2.5 – Діаграма діяльності (2 частина)

2.4 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) в UML є одним з типів діаграм взаємодії, яка використовується для візуалізації послідовності обміну повідомленнями між об’єктами або компонентами системи з плином часу. Вона дозволяє детально продемонструвати, як об’єкти взаємодіють один з одним для реалізації певної функціональності або сценарію використання.

На діаграмі послідовності об’єкти, або учасники взаємодії, зображуються у вигляді прямокутників з підписами, розташованими горизонтально вздовж верхньої частини діаграми. Під кожним об’єктом розташована вертикальна лінія, яка називається лінією життя (lifeline) і представляє час існування об’єкта протягом взаємодії. Повідомлення, які обмінюються між об’єктами, зображуються стрілками, що з’єднують відповідні лінії життя [13].

Діаграма послідовності читається зверху вниз у хронологічному порядку. Вона дозволяє візуалізувати такі аспекти:

- порядок повідомлень, які надсилаються між об’єктами;
- об’єкти, які беруть участь у взаємодії, та їх ролі;
- створення та знищення об’єктів під час взаємодії;
- потоки керування, такі як ітерації, альтернативні сценарії та паралельні потоки;
- часові відношення між повідомленнями.

Діаграми послідовності є корисними для документування та пояснення складної взаємодії між компонентами системи, особливо в об’єктно-орієнтованому програмуванні. Вони допомагають розробникам візуалізувати потік повідомлень і зрозуміти, як різні об’єкти співпрацюють для досягнення певної мети.

Під час проєктування програмного забезпечення діаграми послідовності часто використовуються разом з іншими діаграмами UML, такими як діаграми класів та діаграми станів, для створення повної моделі системи. Вони є особливо

корисними для моделювання динамічної поведінки системи, тестування та документування.

На рисунку 2.6 описана діаграма послідовності прецеденту «Додати відео».

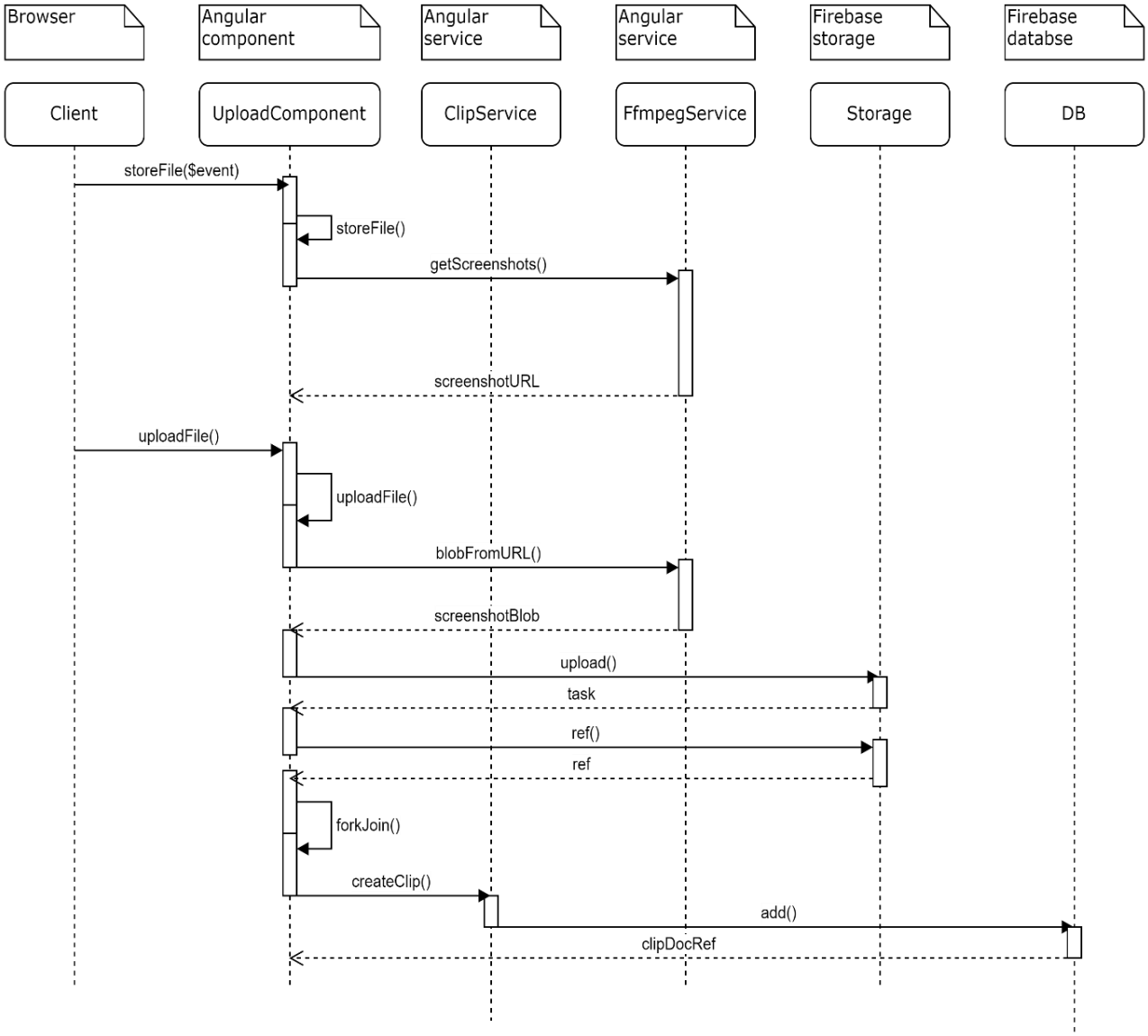

Рисунок 2.6 – Діаграма послідовності

2.5 Діаграма розгортання

Діаграма розгортання (Deployment Diagram) в UML використовується для візуалізації фізичної архітектури системи та розміщення її програмних

компонентів на різних вузлах або апаратних пристроях. Вона показує зв’язки між програмними артефактами, такими як виконувані файли, бібліотеки чи компоненти, та фізичними обчислювальними вузлами, на яких вони розміщені.

Розглянемо основні елементи діаграми розгортання:

- вузол (Node) – це фізичний ресурс, який може представляти різноманітні пристрої, такі як сервери, робочі станції, мобільні пристрої тощо (на діаграмі вузли зображуються у вигляді вузлових 3D-кубиків);
- компонент (Component) – це програмний артефакт, який може бути виконуваним файлом, бібліотекою, вебсервісом чи іншим програмним модулем (компоненти зображуються у вигляді прямокутників із вкладеними формами);
- з’єднання (Connector) – це зв’язок між компонентами або вузлами, який представляє взаємодію або комунікацію між ними (з’єднання можуть бути фізичними (наприклад, кабелі) або логічними (наприклад, мережеві з’єднання)).

Діаграма розгортання допомагає візуалізувати розподіл компонентів системи на різних вузлах, показуючи їх фізичне розміщення та взаємозв’язки. Вона корисна для планування та документування архітектури системи, визначення вимог до інфраструктури та розподілених середовищ [13].

Крім того, діаграма розгортання дозволяє:

- моделювати різні сценарії розгортання системи;
- ідентифікувати потенційні вузькі місця та проблеми масштабування;
- візуалізувати зв’язки між компонентами та їх залежності від фізичних ресурсів;
- полегшити комунікацію між командами розробки та операційними командами.

Діаграма розгортання часто використовується разом з іншими видами діаграм UML, такими як діаграми компонентів та діаграми пакетів, для створення повної моделі архітектури системи.

На рисунку 2.7 наведено діаграму розгортання системи.

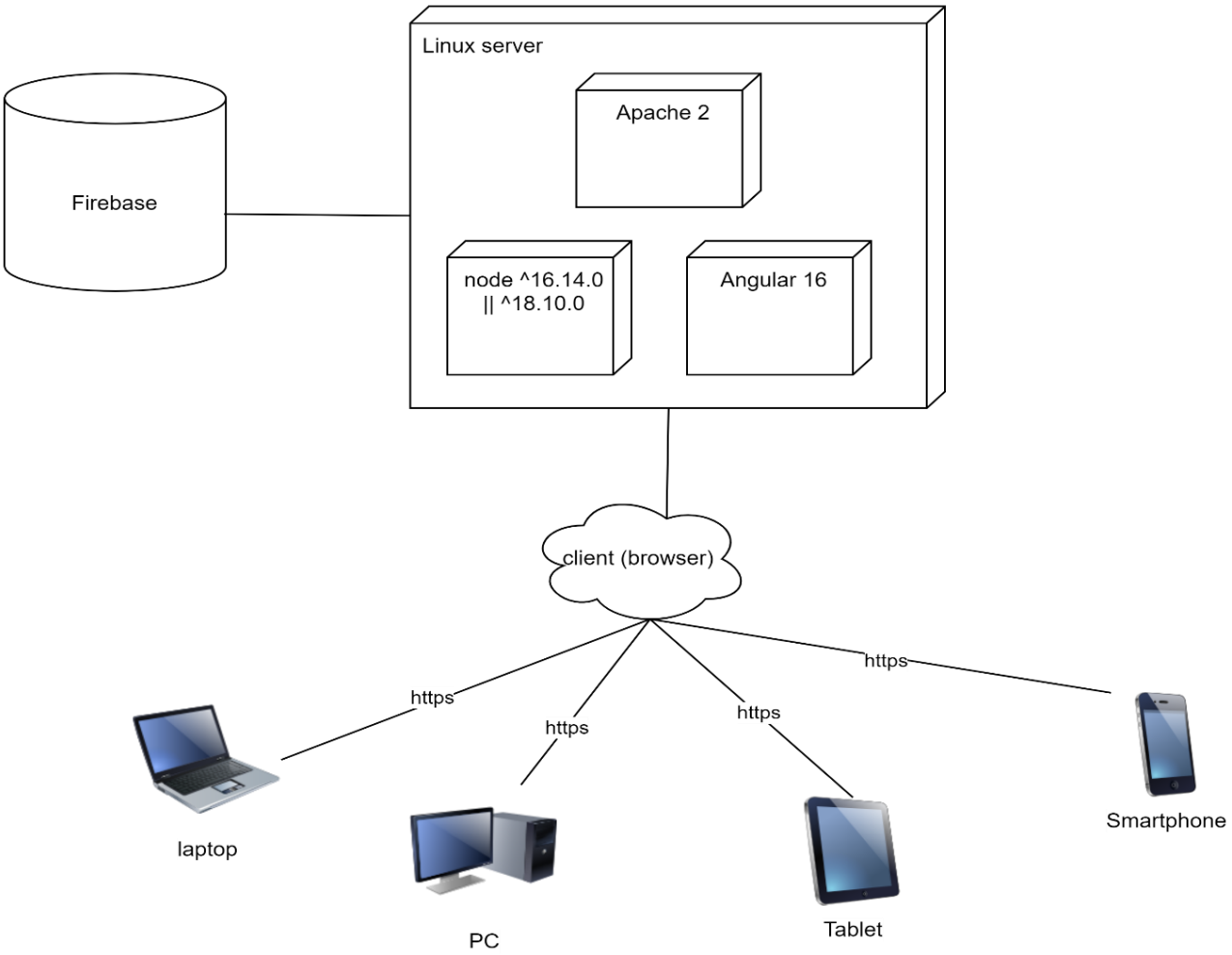

Рисунок 2.7 – Діаграма розгортання

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Опис інструментів розробки

Для реалізації було використано фреймворк Angular та платформу розробки веб застосунків Firebase.

RxJS – це бібліотека, що реалізує принципи реактивного програмування JavaScript. Заснована на об’єктах типу Observable, вона спрощує написання та контроль асинхронного та подійного коду [4].

Gsutil – інструмент, який дозволяє отримати доступ до Cloud Storage з командного рядка за протоколом HTTPS.

Ffmpeg.wasm – це чистий Webassembly / Javascript порт FFmpeg. Він дозволяє записувати, конвертувати та транслювати відео та аудіо прямо в браузері [9].

3.2 Основні класи системи

Оскільки основним прецедентом системи є «Додати відео», то основними класами системи будуть ті, що надають можливість обирати відеофайл, задавати його налаштування та завантажувати до хмарного сховища.

Головними класами системи є:

- UploadComponent – відповідає за завантаження відеофайлів користувачем, взаємодію з основними сервісами системи і за завантаження файлів у Firebase Storage;
- ClipService – відповідає за взаємодію зі Firestore database, а саме за CRUD-операції;
- FfmpegService – відповідає за отримання скріншотів для мініатюр, а також їх конвертації в Blob [9].

Нижче приведено приклад функції попередньої обробки файлу (рис. 3.1).

```
async storeFile($event: Event) {
  if (this.ffmpegService.isRunning) {
   return;
  }

  this.isDragover = false;

  this.file = ($event as DragEvent).dataTransfer
   ? ($event as DragEvent).dataTransfer?.files.item(0) ?? null
   : ($event.target as HTMLInputElement).files?.item(0) ?? null;

  if (!this.file || this.file.type !== 'video/mp4') {
   return;
  }

  this.screenshots = await this.ffmpegService.getScreenshots(this.file);

  this.selectedScreenshot = this.screenshots[0];

  this.title.setValue(this.file.name.replace(/\.[^/.]+$/, ''));
  this.nextStep = true;
 }
```

Рисунок 3.1 – Приклад функції

Детально ознайомитися з кодом компонента можна в додатку А, сервісів – в додатках Б і В.

3.3 Налаштування Firebase

Для зберігання даних, файлів та процесу реєстрації/аутентифікації використовуються відповідні інструменти Firebase. Це дозволяє пришвидшити роботу системи, а також забезпечує безпеку даних та файлів користувачів.

Для реєстрації/авторизації використано інструмент «Authentication» з sign-in методом «Email/Password», який є одним із базових методів аутентифікації в Firebase (рис. 3.2) [5, 10].

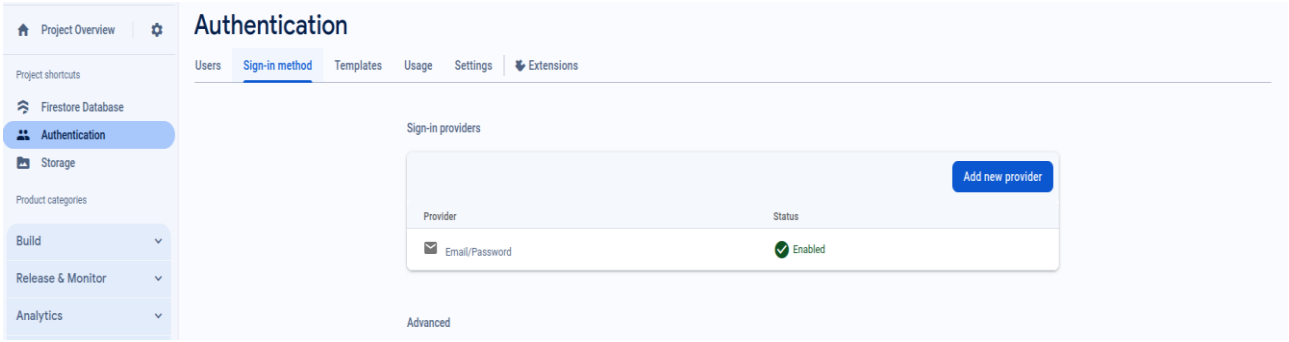

Рисунок 3.2 – Інструмент «Authentication»

Для зберігання відео та мініатюр до них використано інструмент «Storage» (рис. 3.3) за наступними налаштуваннями правил доступу (див. рис. 3.4) [5, 14]:

– читання (перегляд) будь-яких файлів у сховищі дозволено для всіх користувачів, включно з неавторизованими;
– дозвіл на запис файлів у сховище є тільки для авторизованих користувачів, якщо тип файлу mp4 і його розмір не перевищує 25Mb;
– видаляти файли зі сховища дозволяється лише авторизованим користувачам.

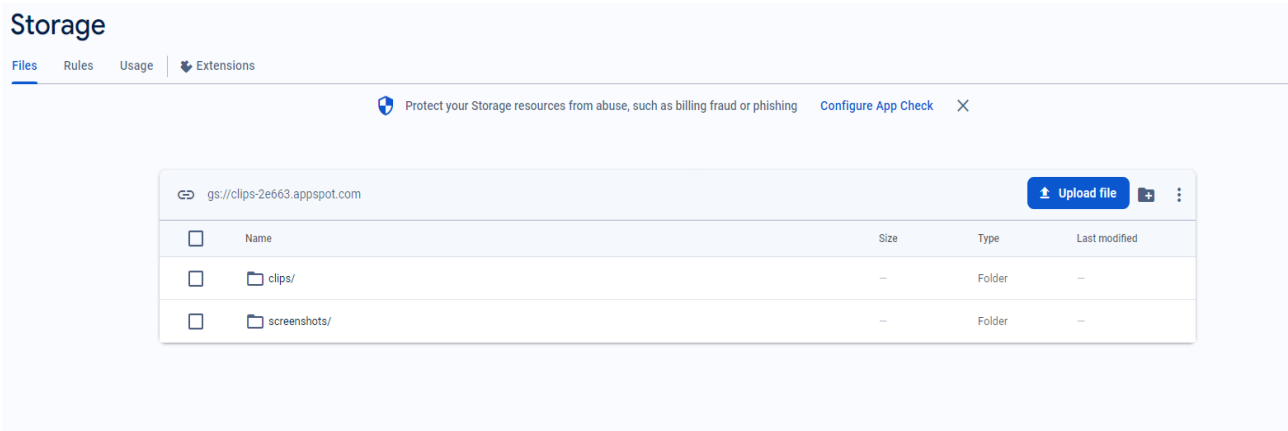

Рисунок 3.3 – Інструмент «Storage»

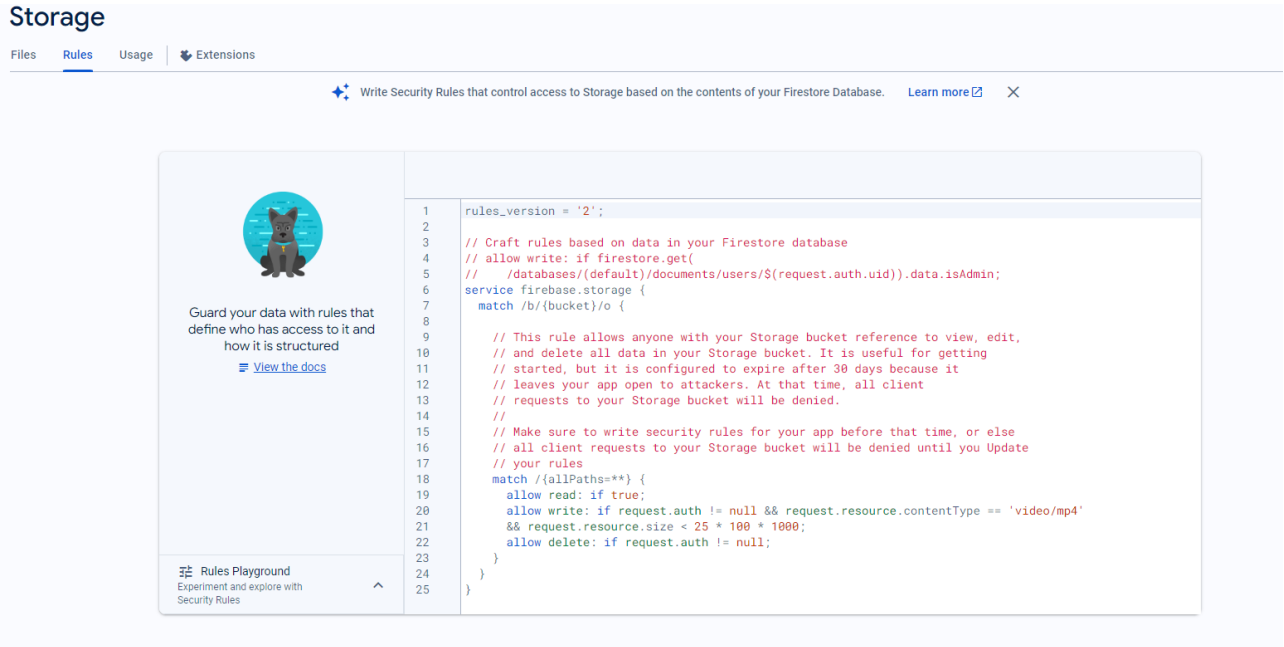

Рисунок 3.4 – Правила взаємодії зі «Storage»

Для зберігання даних користувачів та відео, а також їх зв’язку використано інструмент «Cloud Firestore» (рис. 3.5), а також за аналогією зі «Storage» задані налаштування правил, які дозволяють читати всім користувачам, а дозвіл на запис є тільки в авторизованих користувачів (див. рис. 3.6) [5, 10].

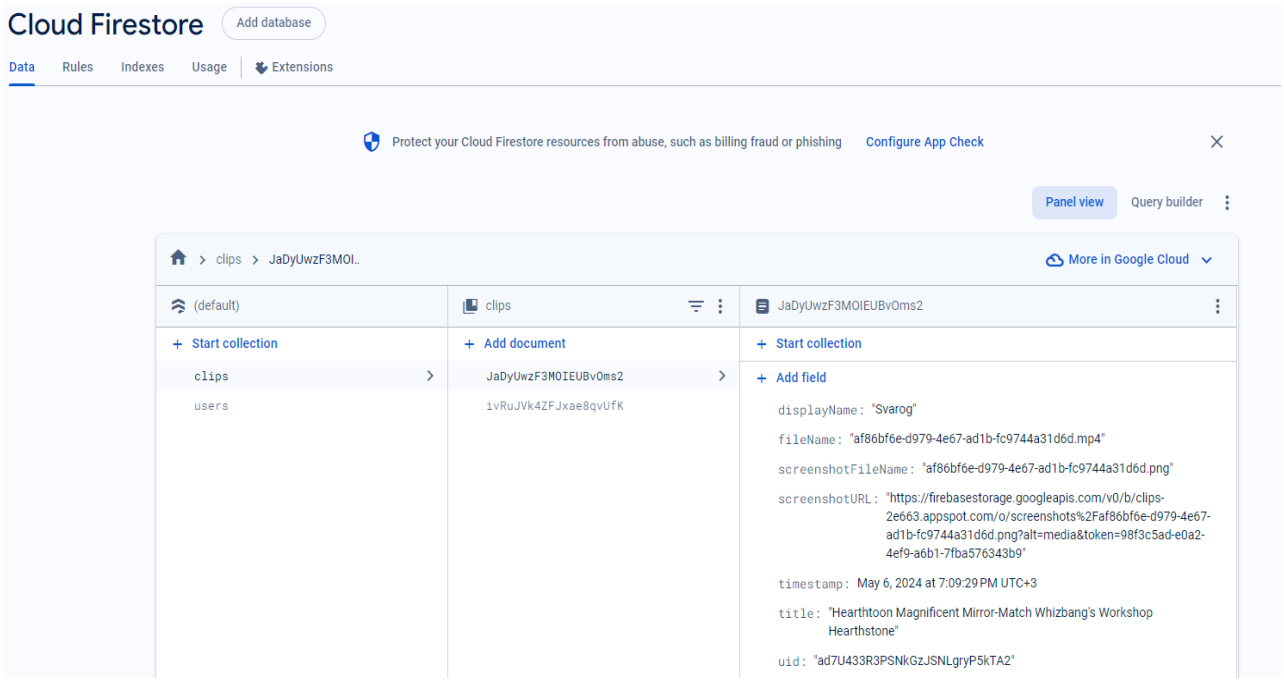

Рисунок 3.5 – Інструмент «Cloud Firestore»

Рисунок 3.6 – Правила взаємодії з «Cloud Firestore»

Так як система дозволяє сортувати відео за датою створення, то доречно також буде створити індекси за зростанням та спаданням (рис. 3.7) [5, 14].

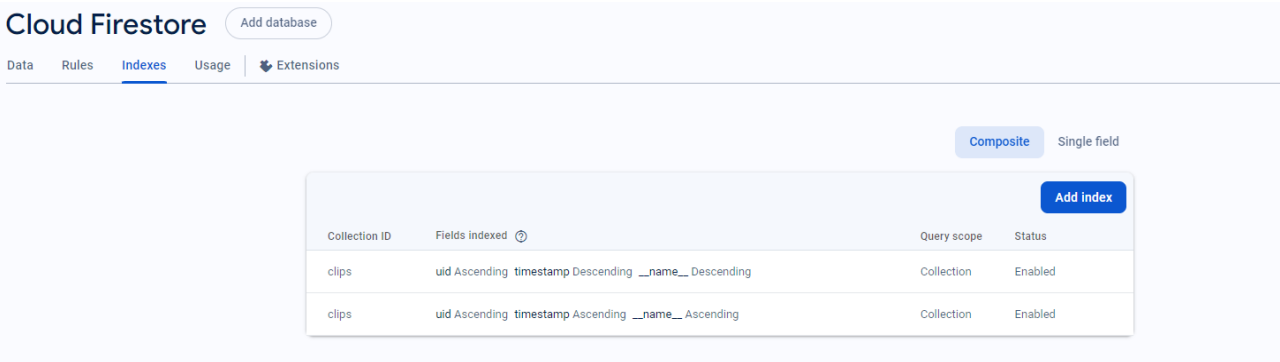

Рисунок 3.7 – Індекси

3.4 Тестування проєкту

Unit-тести в Angular використовуються для тестування окремих компонентів, сервісів, пайпів, директив або інших ізольованих одиниць коду. Основною метою unit-тестування є перевірка того, що кожна одиниця працює належним чином незалежно, без залежності від зовнішніх залежностей. Angular надає вбудовані утиліти та інструменти для написання та запуску unit-тестів.

Для написання unit-тестів в Angular зазвичай використовуються такі інструменти та фреймворки як Jasmine, Karma або Angular Testing Utilities [1, 4].

Наведемо приклад unit-тесту для компонента «Nav» (рис. 3.8).

```
describe('NavComponent', () => {
  let component: NavComponent;
  let fixture: ComponentFixture<NavComponent>;
  const mockedAuthService = jasmine.createSpyObj(
    'AuthService',
    ['createUser', 'logout'],
    {
      isAuthenticated$: of(true),
    }
  );

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [NavComponent],
      imports: [RouterTestingModule],
                providers:  [{  provide:  AuthService,  useValue:
mockedAuthService }],
    }).compileComponents();
  });
```

Рисунок 3.8 – Unit-тест компоненту Nav

Інтеграційні тести в Angular використовуються для тестування взаємодії між кількома компонентами, сервісами чи іншими одиницями коду, які працюють разом, щоб забезпечити певну функціональність. Основна мета інтеграційного тестування – перевірити, що різні частини додатку працюють належним чином, коли вони інтегровані та взаємодіють між собою згідно очікувань [2, 3].

Angular не надає спеціальних інструментів чи фреймворків для інтеграційного тестування з коробки. Натомість можна використовувати сторонні інструменти, такі як Cypress, Protractor (для end-to-end тестування) або Angular Testing Library (набір утиліт для тестування Angular компонентів) [1, 8].

Наведемо приклад інтеграційного тесту з використанням Cypress (див. рис. 3.9).

```
describe('Clip', () => {
  it('should play clip', () => {
    cy.visit('/');
    cy.get('app-clips-list > .grid a:first').click();
    cy.get('.video-js').click();
    cy.wait(3000);
    cy.get('.video-js').click();
        cy.get('.vjs-play-progress').invoke('width').should('gte',
0);
  });
});
```

Рисунок 3.9 – Інтеграційний тест

Як unit-тестування, так і інтеграційне тестування є важливими для підтримки високої якості Angular додатку. Unit-тести забезпечують коректну роботу окремих одиниць ізольовано, тоді як інтеграційні тести перевіряють, що різні частини додатку працюють разом, як очікується.

Детально ознайомитися з тестами можна в додатку Г.

3.5 Керівництво користувача

3.5.1 Рівень підготовки користувача

Для зручної роботи з додатком від користувача вимагаються певні знання та вміння. Насамперед, необхідно володіти основними навичками роботи з комп’ютером та вебпереглядачем. Ретельне ознайомлення з інструкціями для користувачів допоможе краще зрозуміти і опанувати функціональні можливості сайту.

3.5.2 Реєстрація в системі

Перед початком використання додатку, користувачу необхідно зареєструватися в системі. Для цього потрібно натиснути на пункт меню «Login/Register», що знаходиться на панелі меню в шапці додатку (рис. 3.10).

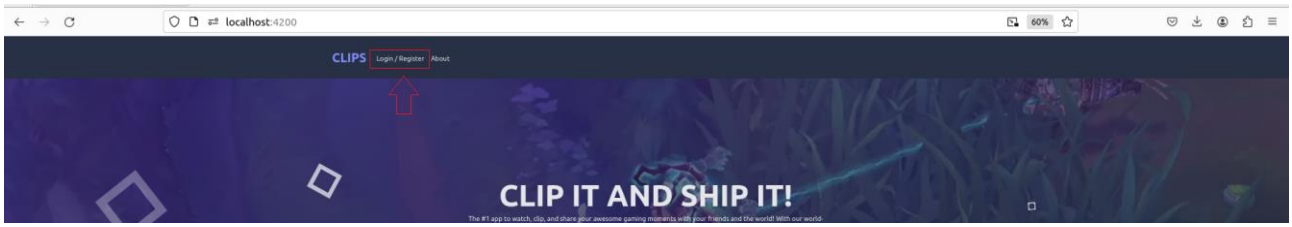

Рисунок 3.10 – Пункт меню «Login/Register»

Система відображає форму з можливістю вибрати один з двох режимів: вхід або реєстрація. Для створення нового облікового запису необхідно обрати пункт «Register». Система відображає форму реєстрації (рис. 3.11).

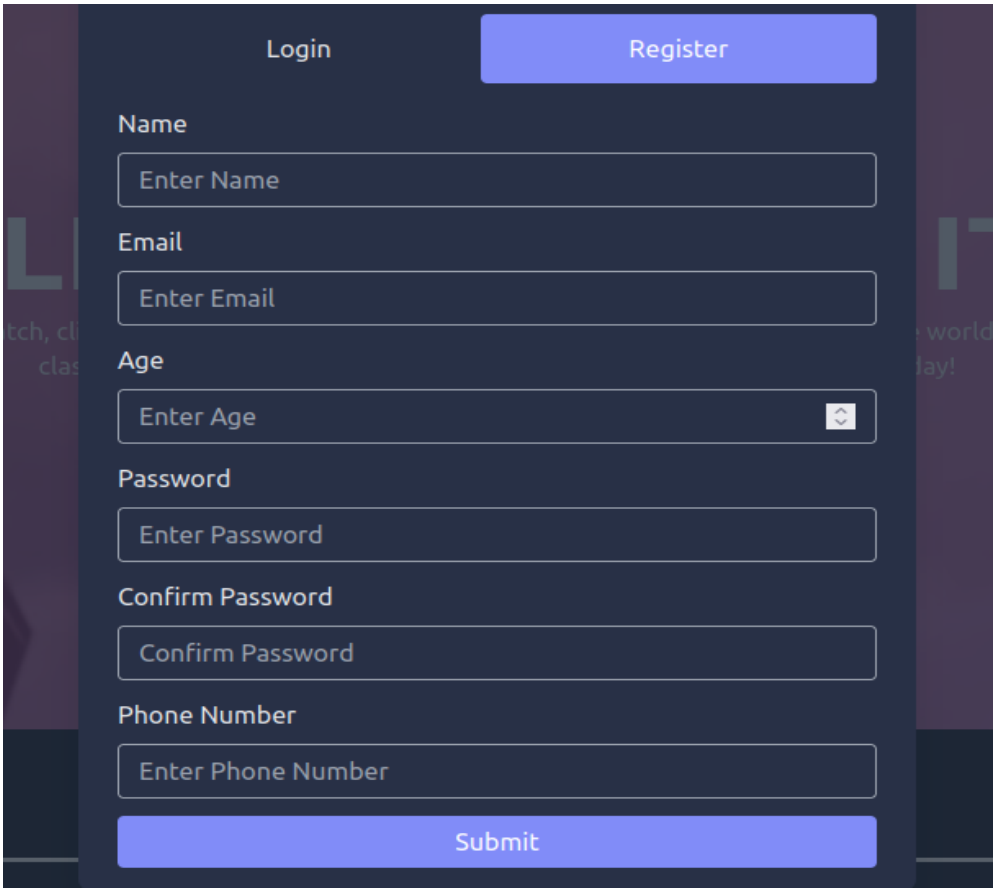

Рисунок 3.11 – Форма реєстрації

Користувачу необхідно заповнити всі поля відповідними даними, якщо, при заповненні, користувач допускає помилку, тобто вводить невалідні дані, то форма про це сповіщає у вигляді відповідного повідомлення під полем, де було введено невалідні дані (див. рис. 3.12 – 3.13).

Помилка може виникнути навіть у тому випадку, коли користувач ввів валідні дані. Це може бути помилка із-за проблем з сервером або дубляжом даних, наприклад при реєстрації використовується email, який вже використовує інший користувач (рис. 3.14).

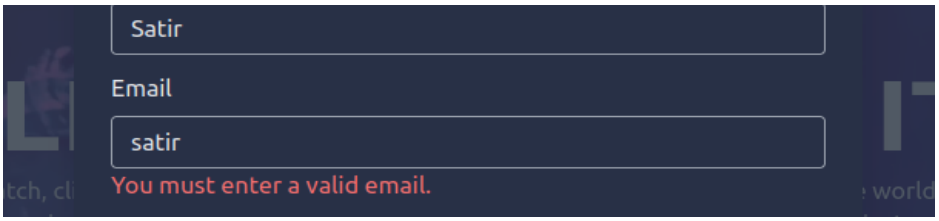

Рисунок 3.12 – Невалідний email

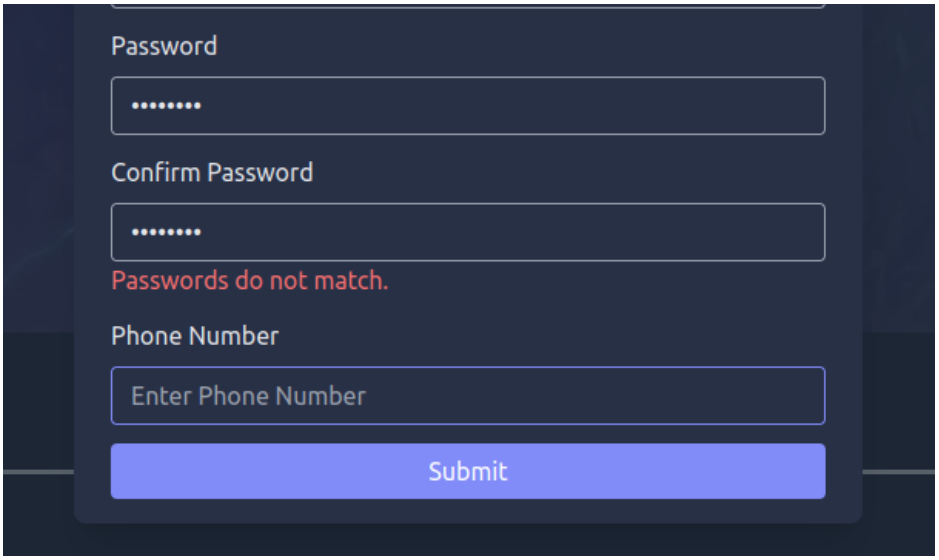

Рисунок 3.13 – Паролі не співпадають

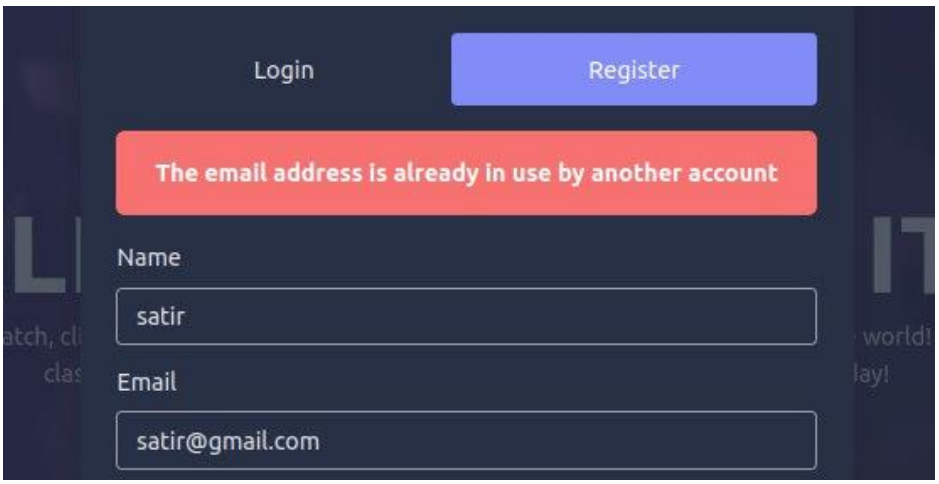

Рисунок 3.14 – Email вже використовується

Якщо реєстрація успішна, то система відображає відповідне повідомлення та переадресовує користувача на головну сторінку.

3.5.3 Вхід/Вихід до/з системи

Якщо в користувача вже є логін і пароль, то необхідно в формі входу/реєстрації обрати пункт «Login» та заповнити відповідні поля (рис. 3.15).

Якщо, при вході до системи, користувач вніс помилкові дані, не відповідає сервер або такого користувача не існує, то система відображає відповідне повідомлення (рис. 3.16).

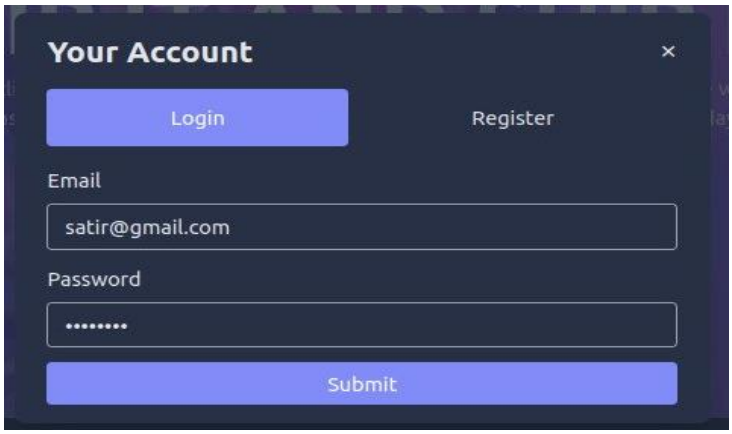

Рисунок 3.15 – Форма входу

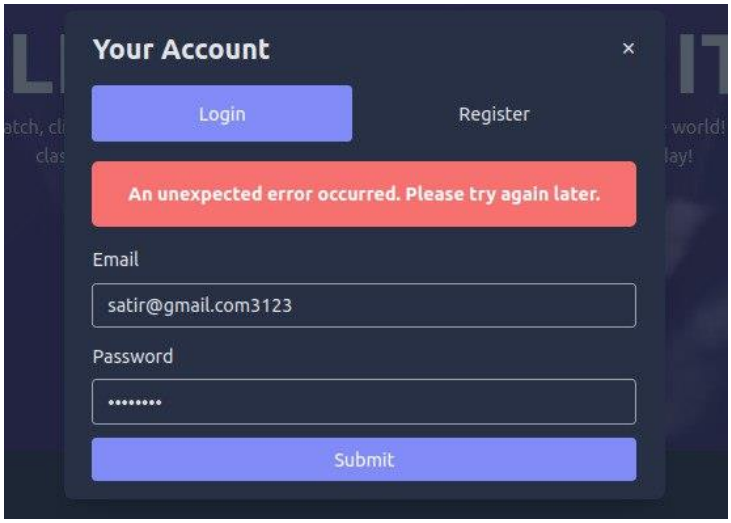

Рисунок 3.16 – Помилка серверу

Якщо авторизація успішна, то система відображає головну сторінку з оновленими пунктами меню. Для того щоб вийти з системи, необхідно обрати пункт меню «Logout» (рис. 3.17).

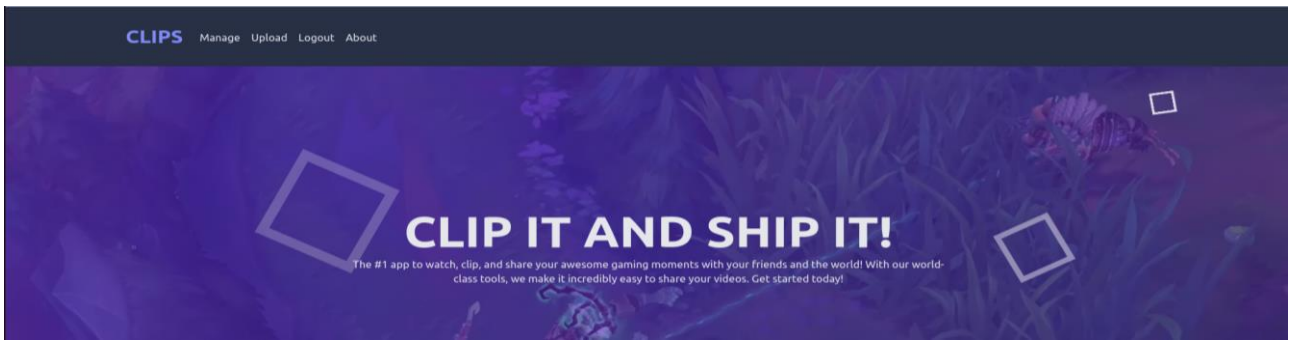

Рисунок 3.17 – Вигляд головної сторінки після авторизації

3.5.4 Завантаження відео

Для того щоб завантажити відео необхідно обрати пункт меню «Upload». Система відображає форму (рис. 3.18) яка дозволяє додати відеофайл або за допомогою перетягування, або через кнопку «Вибрати», яка викликає стандартний файловий оглядач операційної системи. де користувач може вибрати відеофайл (рис. 3.19).

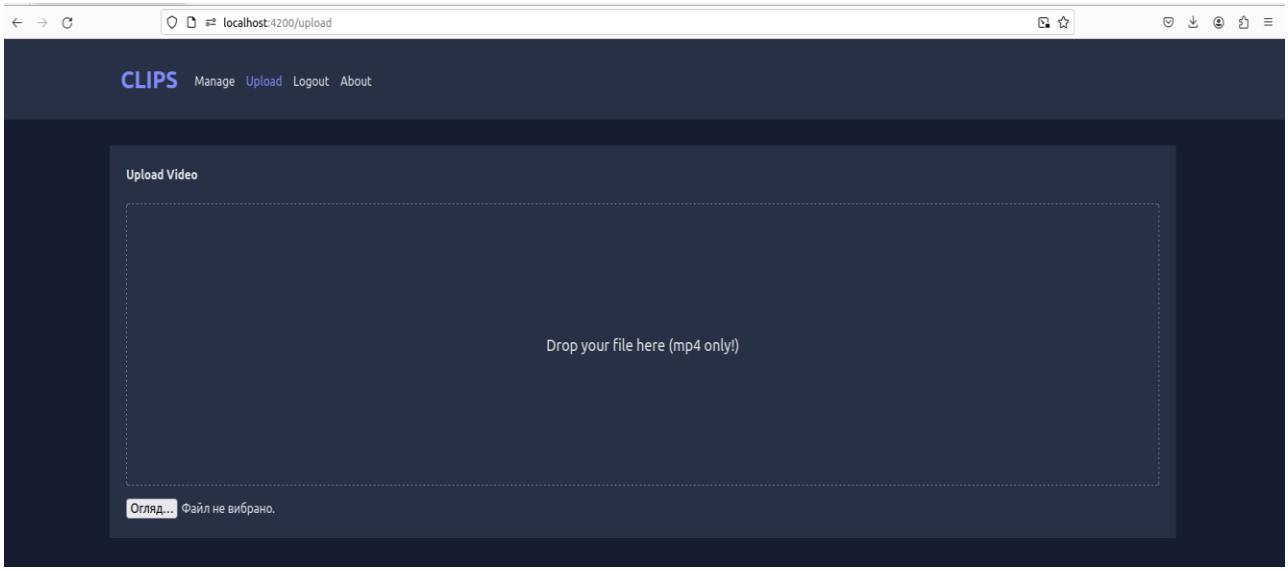

Рисунок 3.18 – Форма завантаження відеофайлу

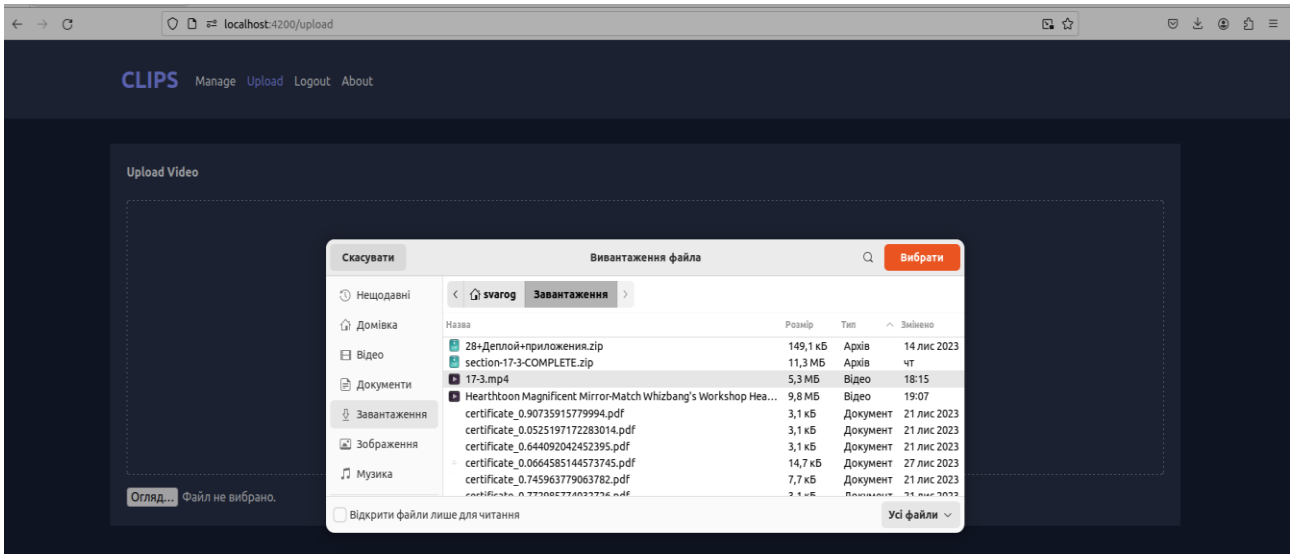

Рисунок 3.19 – Вибір файлу засобами оглядача

Після вибору файлу, система кадрує перші 3 його секунди для створення мініатюр та пропонує користувачу вибрати, яка буде використовуватися для відео (рис. 3.20).

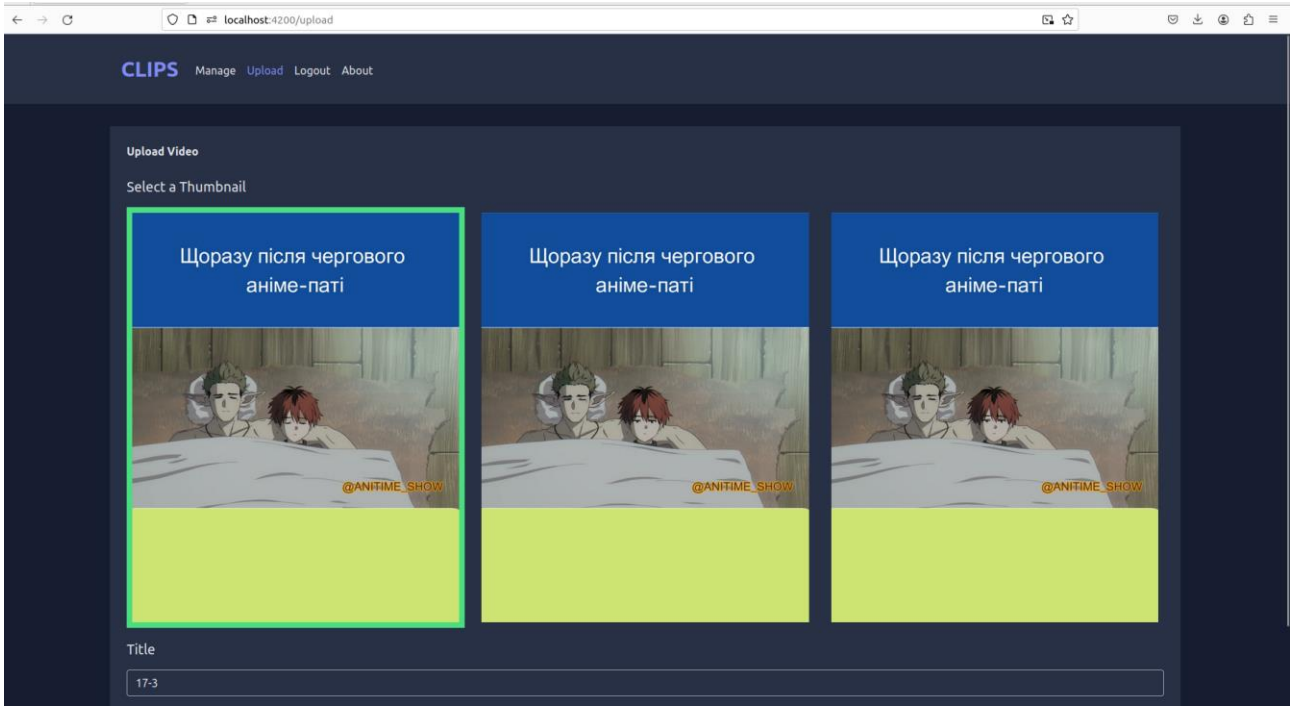

Рисунок 3.20 – Вибір мініатюри

Також цей інтерфейс надає можливість змінити поточну назву відеофайлу. Для цього необхідно змінити назву в полі «Title» (рис. 3.21).

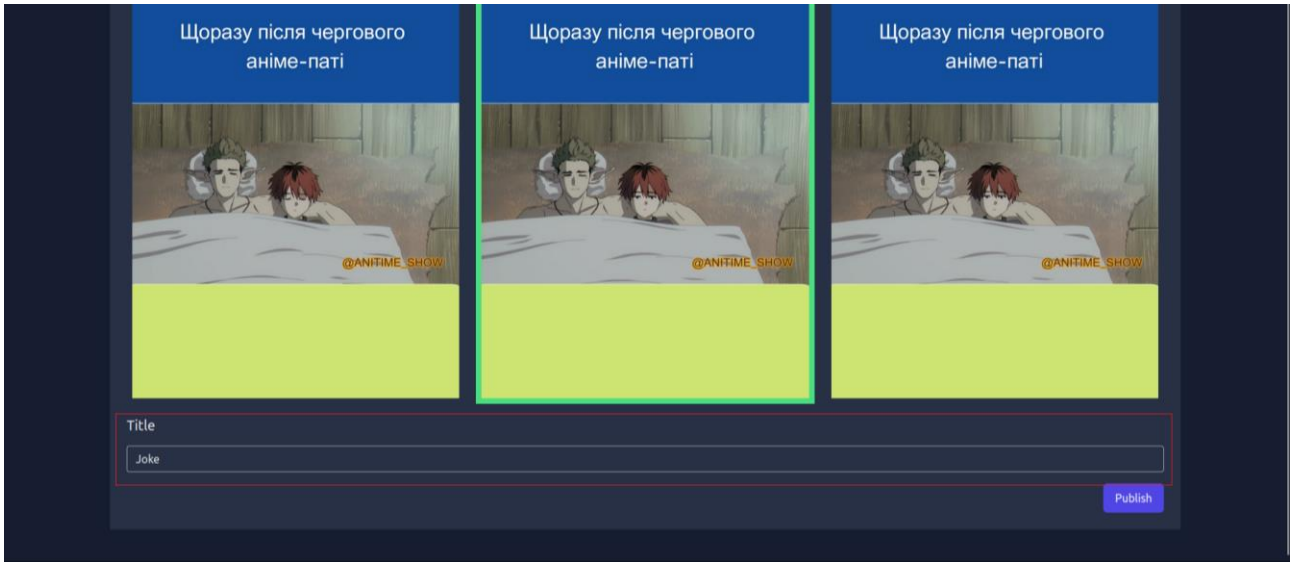

Рисунок 3.21 – Зміна назви

Для збереження відео користувачу необхідно натиснути на кнопку «Publish». Система відображає процес завантаження відео над секцією вибору мініатюри (рис. 3.22).

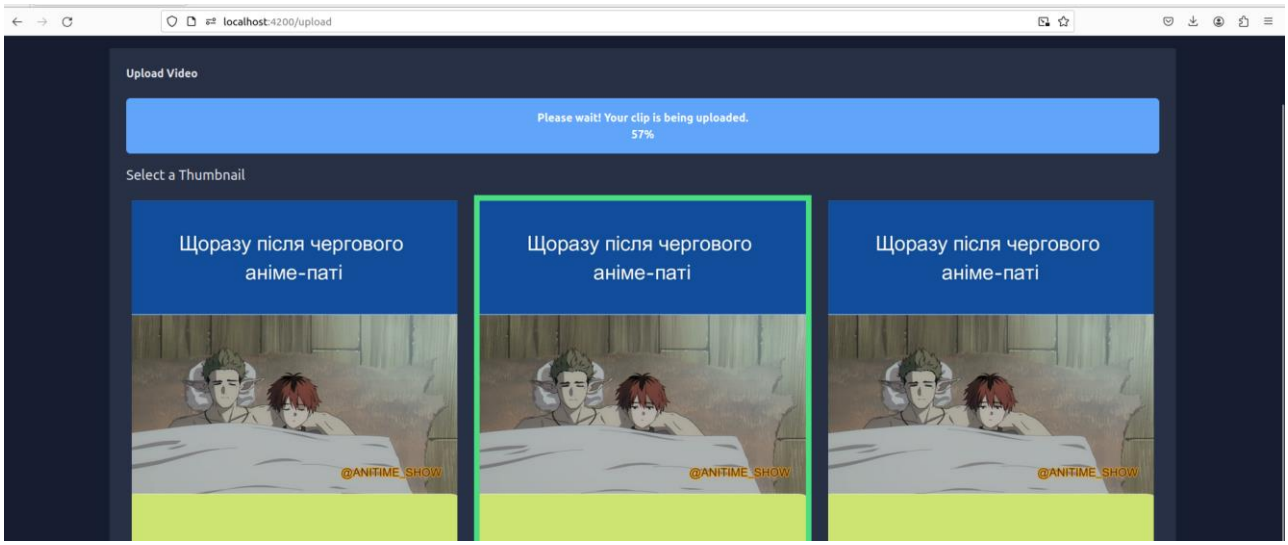

Рисунок 3.22 – Процес завантаження відео

Коли система закінчить завантаження відео, то ідникатор процесу зміниться на напис про успішне завантаження відео (рис. 3.23).

Після завантаження система автоматично переадресовує користувача на сторінку перегляду завантаженого відео (рис. 3.24).

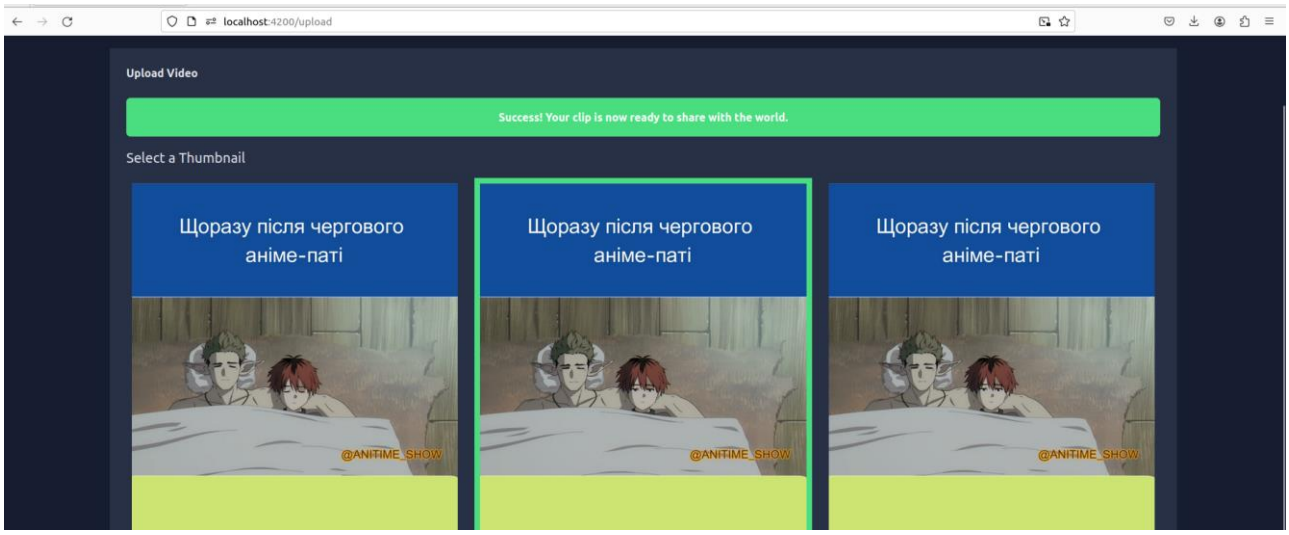

Рисунок 3.23 – Успішне завантаження відео

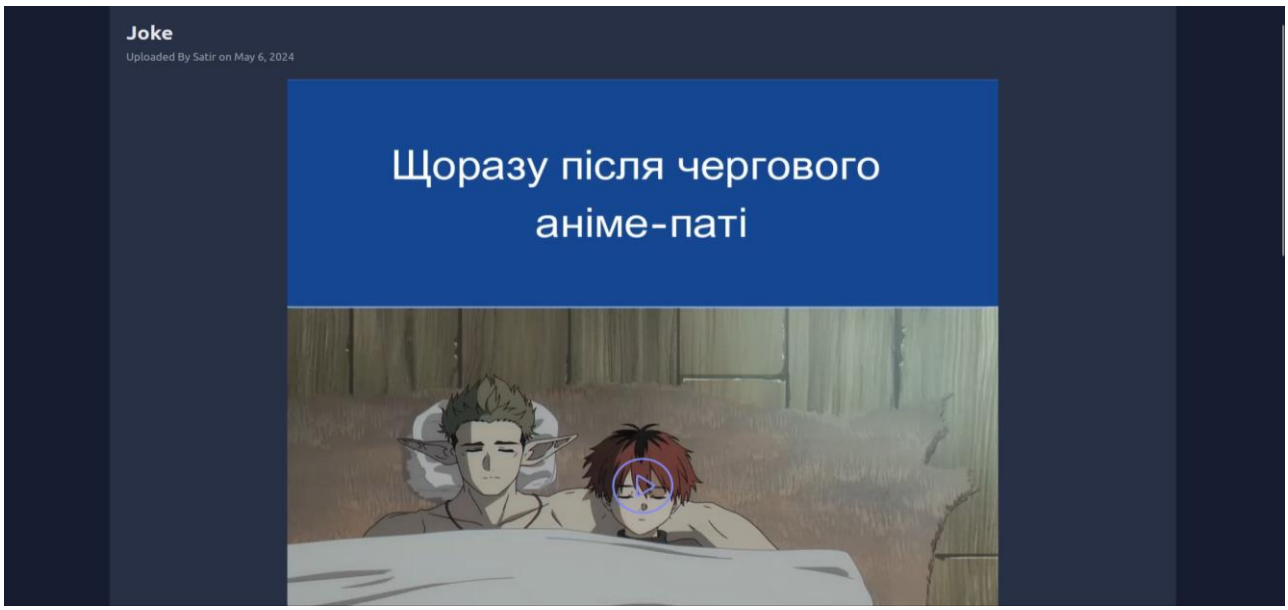

Рисунок 3.24 – Перегляд відео

3.5.5 Керування відео

Кожен авторизований користувач має можливість керувати власними відео. Для цього необхідно натиснути на пункт меню «Manage». При переході на цю сторінку, система підтягує та відображає завантажені користувачем відео (рис. 3.25).

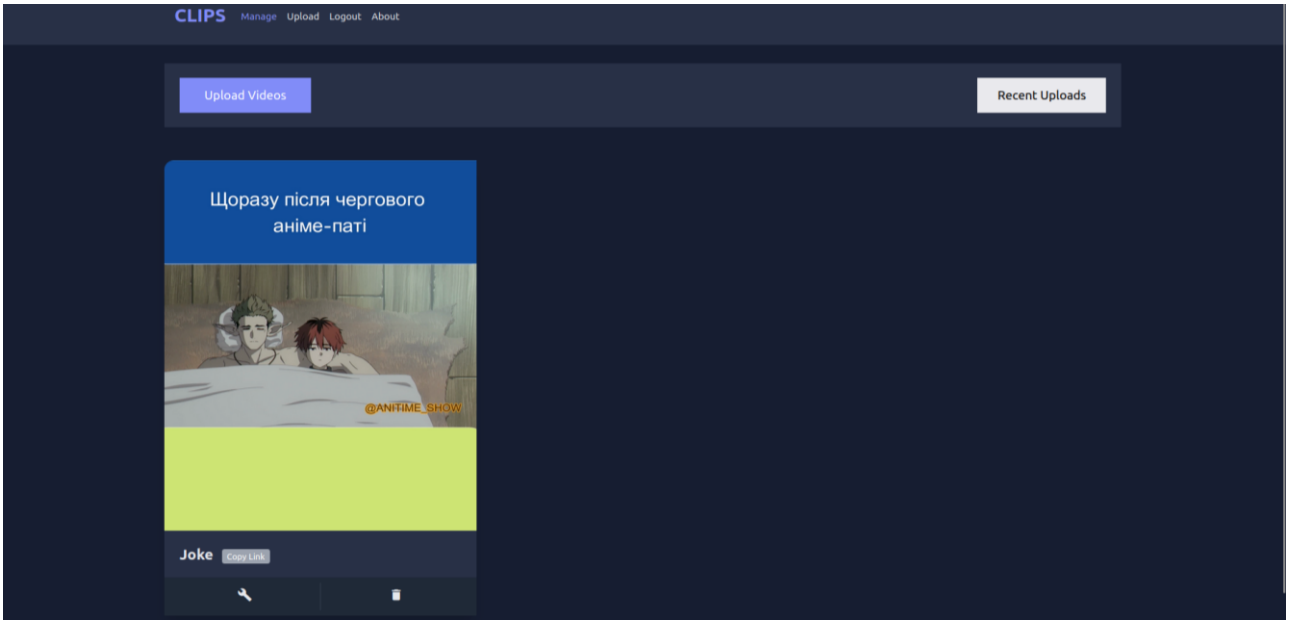

Рисунок 3.25 – Сторінка керування відео

З цієї сторінки також можна перейди до форми завантаження відео, для цього необхідно натиснути на кнопку «Upload Videos». Відео, що вже додані, можна перейменовувати. Для цього необхідно натиснути на піктограму ключа в нижній частині картки відео. Система відображає форму, де користувач має можливість оновити назву (рис. 3.26).

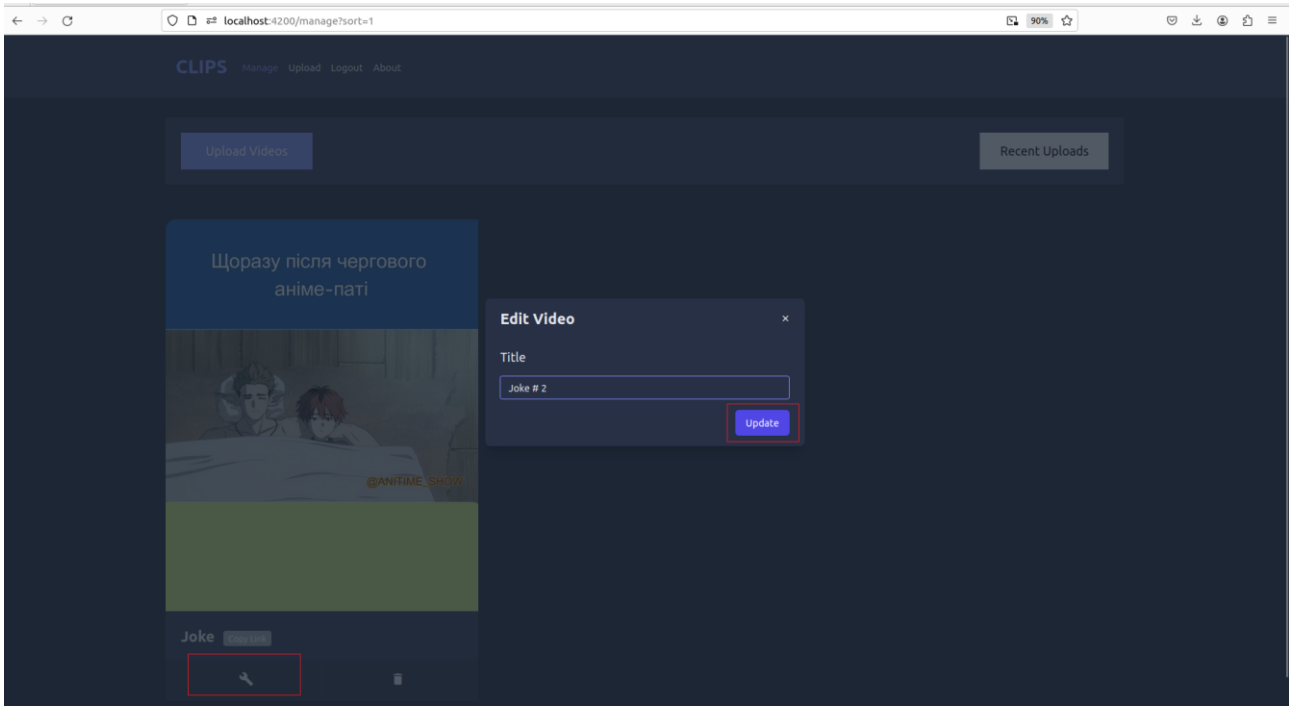

Рисунок 3.26 – Редагування назви відео

Також користувач може поділитися посиланням на відео. Для цього необхідно натиснути на кнопку «Copy link», яка знаходиться біля назви відео. В свою чергу при натисканні на назву, система відображає сторінку перегляду вибраного відео.

Якщо відео більше не потрібно, то його можна видалити, натиснувши на піктограму кошика в нижній частині картки відео.

Для того щоб користувач міг бачити спочатку нові відео, встановлено значення сортування «Recent Uploads» (рис. 3.27).

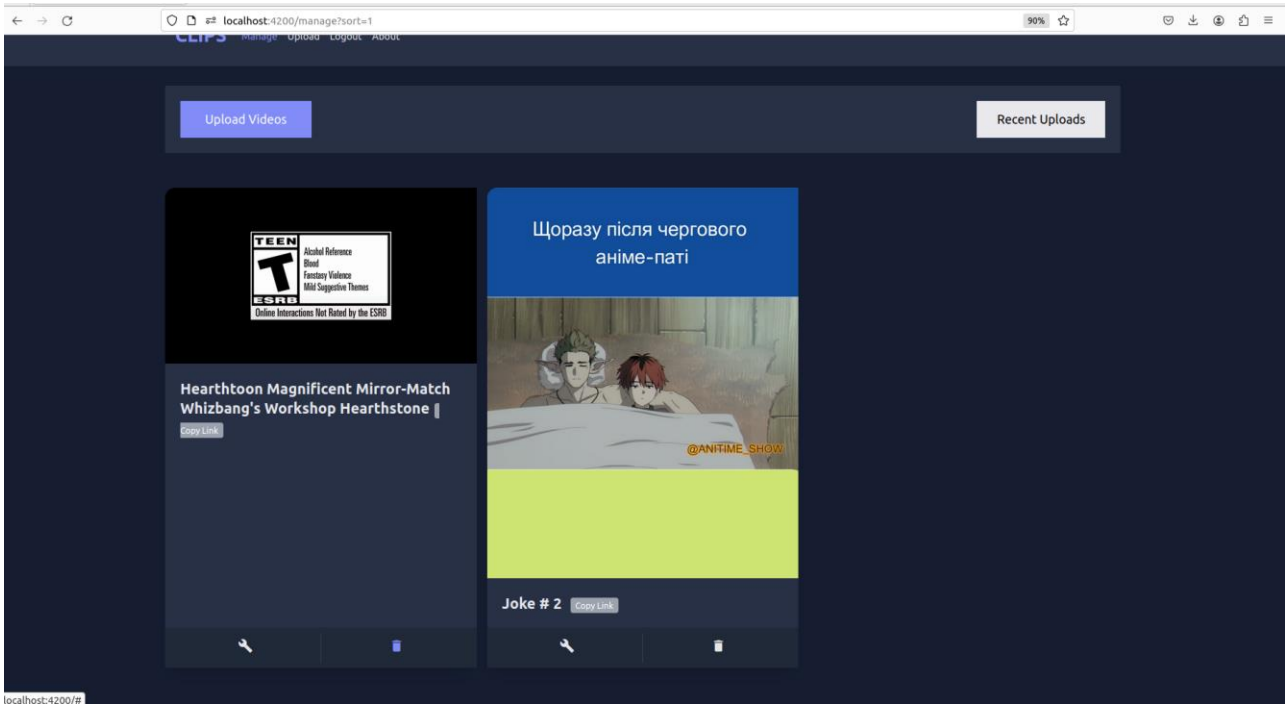

Рисунок 3.27 – Сортування від нових до старих

Якщо ж користувачу, навпаки, спочатку необхідно подивитися старі відео, то необхідно обрати значення сортування «Oldest Uploads» (див. рис. 3.28).

Також користувач може побачити власні відео і відео інших користувачів на головні сторінці. Для їх перегляду авторизація в системі не потрібна (див. рис. 3.29).

Сортування відео на головні сторінці йде виключно за принципом «від нових до старих».

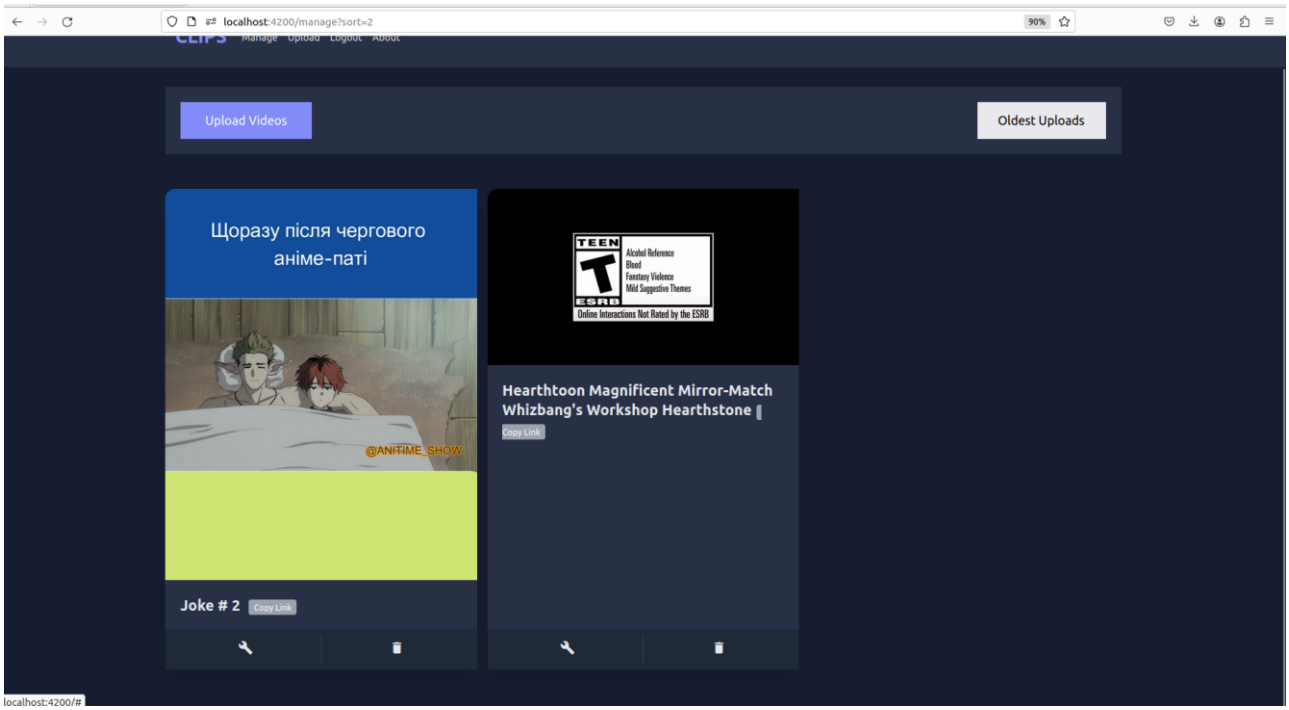

Рисунок 3.28 – Сортування від старих до нових

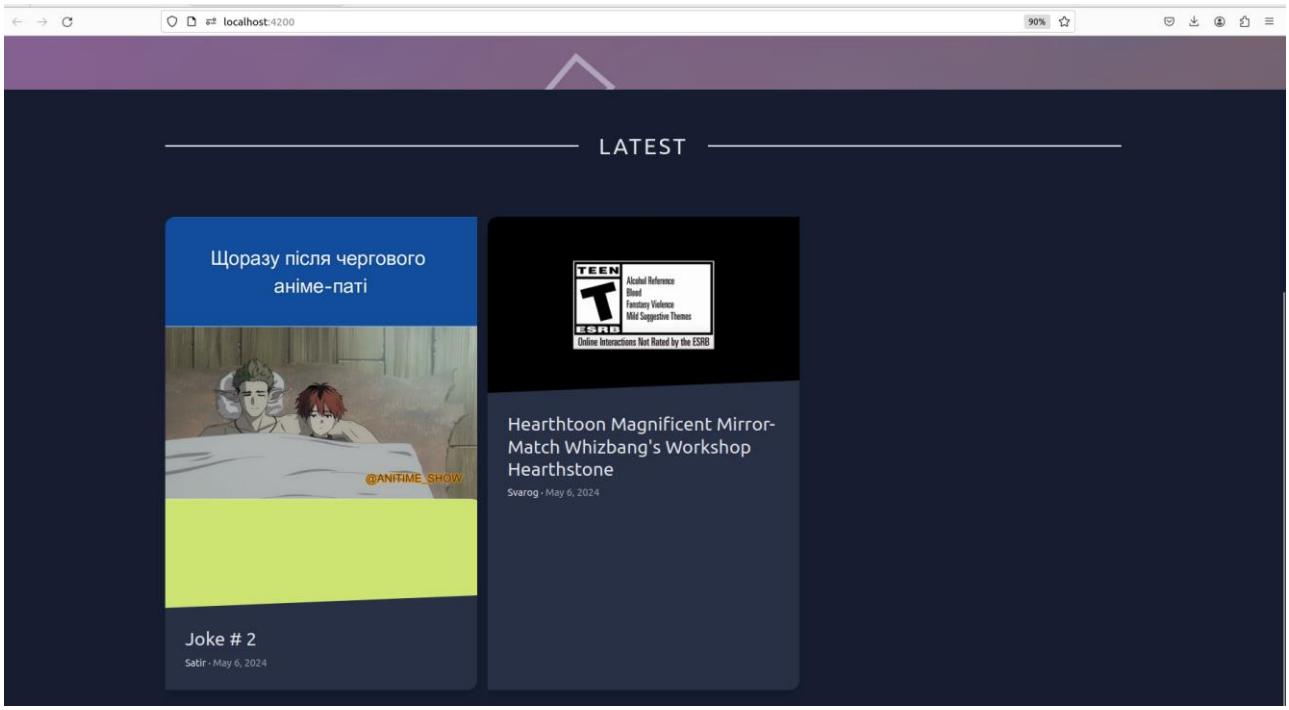

Рисунок 3.29 – Головна сторінка

ВИСНОВКИ

У ході виконання даної роботи було розроблено систему онлайн обміну відео, яка реалізована за допомогою фреймворку Angular та платформи Firebase. Основні функціональні можливості системи включають перегляд відео, завантаження нового контенту, реєстрацію та авторизацію користувачів, редагування та видалення відео. Система відповідає сучасним вимогам до вебдодатків, включаючи адаптивний інтерфейс, сумісність з популярними веббраузерами, високу продуктивність та забезпечення безпеки даних користувачів.

У відповідності з метою кваліфікаційної роботи було розроблено додаток обміну відео для геймерів з використаннями наступних технологій:

- Firebase для авторизації та зберігання даних;
- Angular для реалізації клієнтської частини, а також відправки запитів до сервісу Firebase.

У відповідності з поставленими задачами були виконані наступні етапи створення додатку:

- сформовані вимоги до додатку (функціональні та нефункціональні (інтерфейс, кросбраузерність, безпека, продуктивність)), а також проведено огляд аналогів та предметної області;
- спроєктована та побудована структура додатку (побудовані діаграми прецедентів, діяльності, послідовності та розгортання; надано детальний опис кожного з прецедентів);
- реалізовано додаток обміну відео для геймерів (наведена інструкція по налаштування Firebase, надано керівництво користувача та структура проєкту);
- протестована робота системи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 21.03.2024).

2. Bampakos A. Angular Projects: Build modern web apps by exploring Angular 12 with 10 different projects and cutting-edge technologies. Birmingham : Packt Publishing, 2021. 344 p.

3. Bampakos A, Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15. Birmingham : Packt Publishing, 2023. 446 p.

4. Chebbi L. Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application’s data reactively and efficiently using RxJS 7. Birmingham : Packt Publishing, 2022. 224 p.

5. Developer documentation for Firebase. URL: https://firebase.google.com/docs (дата звернення: 23.03.2024).

6. DLive. URL: https://dlive.tv/ (дата звернення: 02.03.2024).

7. Facebook Gaming. URL: https://www.facebook.com/gaming/video/ (дата звернення: 02.03.2024).

8. Fain Y., Moiseev A. Angular Development with TypeScript. New York : Manning Publications, 2019. 560 p.

9. Ffmpeg Documentation. URL: https://ffmpeg.org/ffmpeg.html (дата звернення: 06.04.2024).

10. Hussain F., Hussain K. Mastering Firebase: The Complete Guide to Building and Scaling Apps. Michigan : Independently published, 2024. 228 p.

11. Mixer. URL: https://news.xbox.com/en-us/2017/05/25/xbox-one-windows-10-introducing-mixer/ (дата звернення: 02.03.2024).

12. Twitch. URL: https://www.twitch.tv/ (дата звернення: 02.03.2024).

13. Unhelkar B. Software Engineering with UML. Boca Raton : Auerbach Publications, 2020. 426 p.

14. Wieruch R. The Road to Firebase: Your journey to master Firebase in JavaScript. Michigan : Independently published, 2021. 199 p.

15. YouTube Gaming. URL: https://www.youtube.com/gaming (дата звернення: 02.03.2024).

ДОДАТОК А

UploadComponent

```
import { Component, OnDestroy } from '@angular/core';
import { UntypedFormControl, FormGroup, Validators } from '@angular/forms';
import {
  AngularFireStorage,
  AngularFireUploadTask,
} from '@angular/fire/compat/storage';
import { v4 as uuid } from 'uuid';
import { switchMap } from 'rxjs/operators';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import firebase from 'firebase/compat/app';
import { ClipService } from 'src/app/services/clip.service';
import { Router } from '@angular/router';
import { FfmpegService } from 'src/app/services/ffmpeg.service';
import { combineLatest, forkJoin } from 'rxjs';

@Component({
  selector: 'app-upload',
  templateUrl: './upload.component.html',
  styleUrls: ['./upload.component.css'],
})
export class UploadComponent implements OnDestroy {
  isDragover = false;
  file: File | null = null;
  nextStep = false;
  showAlert = false;
  alertColor = 'blue';
  alertMsg = 'Please wait! Your clip is being uploaded.';
  inSubmission = false;
  percentage = 0;
  showPercentage = false;
  user: firebase.User | null = null;
  task?: AngularFireUploadTask;
  screenshots: string[] = [];
```

```
selectedScreenshot = '';
screenshotTask?: AngularFireUploadTask;

title = new UntypedFormControl('', [
  Validators.required,
  Validators.minLength(3),
]);
uploadForm = new FormGroup({
  title: this.title,
});

constructor(
  private storage: AngularFireStorage,
  private auth: AngularFireAuth,
  private clipsService: ClipService,
  private router: Router,
  public ffmpegService: FfmpegService
) {
  auth.user.subscribe((user) => (this.user = user));
  this.ffmpegService.init();
}

ngOnDestroy(): void {
  this.task?.cancel();
}

async storeFile($event: Event) {
  if (this.ffmpegService.isRunning) {
    return;
  }

  this.isDragover = false;

  this.file = ($event as DragEvent).dataTransfer
    ? ($event as DragEvent).dataTransfer?.files.item(0) ?? null
    : ($event.target as HTMLInputElement).files?.item(0) ?? null;

  if (!this.file || this.file.type !== 'video/mp4') {
```

```
    return;
  }

  this.screenshots = await this.ffmpegService.getScreenshots(this.file);

  this.selectedScreenshot = this.screenshots[0];

  this.title.setValue(this.file.name.replace(/\.[^/.]+$/, ''));
  this.nextStep = true;
}

async uploadFile() {
  this.uploadForm.disable();

  this.showAlert = true;
  this.alertColor = 'blue';
  this.alertMsg = 'Please wait! Your clip is being uploaded.';
  this.inSubmission = true;
  this.showPercentage = true;

  const clipFileName = uuid();
  const clipPath = `clips/${clipFileName}.mp4`;

  const screenshotBlob = await this.ffmpegService.blobFromURL(
    this.selectedScreenshot
  );
  const screenshotPath = `screenshots/${clipFileName}.png`;

  this.task = this.storage.upload(clipPath, this.file);
  const clipRef = this.storage.ref(clipPath);

  this.screenshotTask = this.storage.upload(screenshotPath, screenshotBlob);
  const screenshotRef = this.storage.ref(screenshotPath);

  combineLatest([
    this.task.percentageChanges(),
    this.screenshotTask.percentageChanges(),
  ]).subscribe((progress) => {
```

```
  const [clipProgress, screenshotProgress] = progress;

  if (!clipProgress || !screenshotProgress) {
    return;
  }

  const total = clipProgress + screenshotProgress;

  this.percentage = (total as number) / 200;
});

forkJoin([
  this.task.snapshotChanges(),
  this.screenshotTask.snapshotChanges(),
])
  .pipe(
    switchMap(() =>
      forkJoin([clipRef.getDownloadURL(), screenshotRef.getDownloadURL()])
    )
  )
  .subscribe({
    next: async (urls) => {
      const [clipURL, screenshotURL] = urls;

      const clip = {
        uid: this.user?.uid as string,
        displayName: this.user?.displayName as string,
        title: this.title.value,
        fileName: `${clipFileName}.mp4`,
        url: clipURL,
        screenshotURL,
        screenshotFileName: `${clipFileName}.png`,
        timestamp: firebase.firestore.FieldValue.serverTimestamp(),
      };

      const clipDocRef = await this.clipsService.createClip(clip);

      console.log(clip);
```

```
        this.alertColor = 'green';
        this.alertMsg =
          'Success! Your clip is now ready to share with the world.';
        this.showPercentage = false;

        setTimeout(() => {
          this.router.navigate(['clip', clipDocRef.id]);
        }, 1000);
      },
      error: (error) => {
        this.uploadForm.enable();

        this.alertColor = 'red';
        this.alertMsg = 'Upload failed! Please try again later.';
        this.inSubmission = true;
        this.showPercentage = false;
        console.error(error);
      },
    });
  }
}
```

ДОДАТОК Б

ClipService

```
import { Injectable } from '@angular/core';
import {
  AngularFirestore,
  AngularFirestoreCollection,
  DocumentReference,
  QuerySnapshot,
} from '@angular/fire/compat/firestore';
import IClip from '../models/clip.model';
import { AngularFireAuth } from '@angular/fire/compat/auth';
import { switchMap, map } from 'rxjs/operators';
import { of, BehaviorSubject, combineLatest } from 'rxjs';
import { AngularFireStorage } from '@angular/fire/compat/storage';
import { ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';

@Injectable({
  providedIn: 'root',
})
export class ClipService  {
  public clipsCollection: AngularFirestoreCollection<IClip>;
  pageClips: IClip[] = [];
  pendingReq = false;

  constructor(
    private db: AngularFirestore,
    private auth: AngularFireAuth,
    private storage: AngularFireStorage,
    private router: Router
  ) {
    this.clipsCollection = db.collection('clips');
  }

  createClip(data: IClip): Promise<DocumentReference<IClip>> {
    return this.clipsCollection.add(data);
```

```
}

getUserClips(sort$: BehaviorSubject<string>) {
  return combineLatest([this.auth.user, sort$]).pipe(
    switchMap((values) => {
      const [user, sort] = values;

      if (!user) {
        return of([]);
      }

      const query = this.clipsCollection.ref
        .where('uid', '==', user.uid)
        .orderBy('timestamp', sort === '1' ? 'desc' : 'asc');

      return query.get();
    }),
    map((snapshot) => (snapshot as QuerySnapshot<IClip>).docs)
  );
}

updateClip(id: string, title: string) {
  return this.clipsCollection.doc(id).update({
    title,
  });
}

async deleteClip(clip: IClip) {
  const clipRef = this.storage.ref(`clips/${clip.fileName}`);
  const screenshotRef = this.storage.ref(
    `screenshots/${clip.screenshotFileName}`
  );

  await clipRef.delete();
  await screenshotRef.delete();

  await this.clipsCollection.doc(clip.docID).delete();
}
```

```
async getClips() {
  if (this.pendingReq) {
    return;
  }

  this.pendingReq = true;
  let query = this.clipsCollection.ref.orderBy('timestamp', 'desc').limit(6);

  const { length } = this.pageClips;

  if (length) {
    const lastDocID = this.pageClips[length - 1].docID;
    const lastDoc = await this.clipsCollection
      .doc(lastDocID)
      .get()
      .toPromise();

    query = query.startAfter(lastDoc);
  }

  const snapshot = await query.get();

  snapshot.forEach((doc) => {
    this.pageClips.push({
      docID: doc.id,
      ...doc.data(),
    });
  });

  this.pendingReq = false;
}

resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
  return this.clipsCollection
    .doc(route.params.id)
    .get()
    .pipe(
      map((snapshot) => {
```

```
        const data = snapshot.data();

        if (!data) {
          this.router.navigate(['/']);
          return null;
        }

        return data;
      })
    );
  }
}
```

ДОДАТОК В

FfmpegService

```
import { Injectable } from '@angular/core';
import { createFFmpeg, fetchFile } from '@ffmpeg/ffmpeg';

@Injectable({
  providedIn: 'root'
})
export class FfmpegService {
  isRunning = false
  isReady = false
  private ffmpeg

  constructor() {
    this.ffmpeg = createFFmpeg({ log: true })
  }

  async init() {
    if(this.isReady) {
      return
    }

    await this.ffmpeg.load()

    this.isReady = true
  }

  async getScreenshots(file: File) {
    this.isRunning = true

    const data = await fetchFile(file)

    this.ffmpeg.FS('writeFile', file.name, data)

    const seconds = [1,2,3]
```

```
const commands: string[] = []

seconds.forEach(second => {
  commands.push(
    // Input
    '-i', file.name,
    // Output Options
    '-ss', `00:00:0${second}`,
    '-frames:v', '1',
    '-filter:v', 'scale=510:-1',
    // Output
    `output_0${second}.png`
  )
})

await this.ffmpeg.run(
  ...commands
)

const screenshots: string[] = []

seconds.forEach(second => {
  const screenshotFile = this.ffmpeg.FS(
    'readFile', `output_0${second}.png`
  )
  const screenshotBlob = new Blob(
    [screenshotFile.buffer], {
      type: 'image/png'
    }
  )

  const screenshotURL = URL.createObjectURL(screenshotBlob)

  screenshots.push(screenshotURL)
})

this.isRunning = false
```

```
    return screenshots
  }

  async blobFromURL(url: string) {
    const response = await fetch(url)
    const blob = await response.blob()

    return blob
  }
}
```

ДОДАТОК Г

Angular-тест

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { of } from 'rxjs';
import { RouterTestingModule } from '@angular/router/testing';
import { By } from '@angular/platform-browser';

import { NavComponent } from './nav.component';
import { AuthService } from '../services/auth.service';

describe('NavComponent', () => {
 let component: NavComponent;
 let fixture: ComponentFixture<NavComponent>;
 const mockedAuthService = jasmine.createSpyObj(
  'AuthService',
  ['createUser', 'logout'],
  {
   isAuthenticated$: of(true),
  }
 );

 beforeEach(async () => {
  await TestBed.configureTestingModule({
   declarations: [NavComponent],
   imports: [RouterTestingModule],
   providers: [{ provide: AuthService, useValue: mockedAuthService }],
  }).compileComponents();
 });

 beforeEach(() => {
  fixture = TestBed.createComponent(NavComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
 });
```

```
  it('should create', () => {
    expect(component).toBeTruthy();
  });

  it('should logout', () => {
    const logoutLink = fixture.debugElement.query(By.css('li:nth-child(3) a'));

    expect(logoutLink).withContext('Not logged in').toBeTruthy();

    logoutLink.triggerEventHandler('click');

    const service = TestBed.inject(AuthService);

    expect(service.logout)
      .withContext('Could not click logout link')
      .toHaveBeenCalledTimes(1);
  });
});
```