Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ КОРИСТУВАЧІВ ТА ЇХ КОМУНІКАЦІЇ ДЛЯ ВІДЕО СЕРВІСУ

Виконав: студент II курсу
групи 1ПІ-22М спеціальності
121 – Інженерія програмного забезпечення
Збитківський Владислав Сергійович

Керівник: к.т.н., доцент каф. ПЗ
Черноволик Г. О.
«08» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ
Кожемяко А. В.
«08» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«08» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«_19_» _вересня_ 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Збитківському Владиславу Сергійовичу

1. Тема роботи – розробка методів і програмних засобів системи користувачів та їх комунікації для відео сервісу.

Керівник роботи: Черноволик Галина Олександрівна, к.т.н., затверджені наказом вищого навчального закладу від «_18_» _вересня_ 2023 р. № 247.

2. Строк подання студентом роботи
_____5 грудня _2023 р._____

3. Вихідні дані до роботи: Angular framework, Mongo DB, Express.js, HTML, CSS, JavaScript, TypeScript, SCSS, BCrypt, Unit tests, Integration tests, E2E tests, Heroku, Jasmine, Karma, Cypress; часові обмеження відповіді системи – 2000 ms, кількість користувачів – до 100, вага фото користувача – не більше 1 mb, формат фото користувача – .jpg or .png, інтернет підключення – від 500 kb/s.

4. Зміст текстової частини: Аналіз предметної галузі системи користувачів та їх комунікації. Проектування структури та компонентів системи. Розробка програмного забезпечення та підготовка середовища. Тестування програмного забезпечення.

5. Перелік ілюстративного матеріалу: модель роботи програмного модуля; Діаграми класів та компонентів проекту; Загальний вигляд інтерфейсу користувача; Приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Черноволик Г.О., к.т.н., доцент каф.ПЗ	19.09.2023	03.12.2023
5	Причепа І. В. к.е.н., доцент каф. ЕПВМ	22.11.2023	27.11.2023

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної галузі системи користувачів	20.09.2023 - 27.09.2023	Вик.
2	Постановка основних задач роботи	28.09.2023 - 10.10.2023	Вик.
3	Проектування структури та компонентів системи	11.10.2023 - 27.10.2023	Вик.
4	Реалізація системи користувачів та їх комунікації за допомогою Angular framework, Express.js та Mongo DB	28.10.2023 - 15.11.2023	Вик.
5	Тестування програмного забезпечення	16.11.2023 - 21.11.2023	Вик.
6	Економічний розділ	22.11.2023 - 27.11.2023	Вик.
7	Оформлення пояснювальної записки таграфічного матеріалу	28.11.2023 - 8.12.2023	Вик.

Студент (підпис) **Збитківський В. С.** (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи (підпис) **Черноволик Г. О.** (прізвище та ініціали)

АНОТАЦІЯ

Збитківський В.С. Розробка системи користувачів та їх комунікації для відео сервісу: магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 99 с.

На укр. мові. Бібліогр. : 31 назв ; рис. : 27; табл. 8.

У даній магістерській кваліфікаційній роботі розроблено систему користувачів та їх комунікації для відео-стримінгового сервісу із передачею відео контенту з використанням YouTube Api, Angular framework та Mongo DB. Дана система знаходиться у відкритому доступі, вона є проста в користуванні, модульна та дозволяє доробку та підтримує можливість масштабування. Система використовує наступні технології/компоненти: NgRx, який застосовується для забезпечення передбачуваного стану контейнера, Angular компоненти для констроювання системи з окремих модулів, Mongo DB для збереження даних та контенту, YouTube Api для відтворення контенту, HTML, CSS, JavaScript, TypeScript для юзер інтерфейсу.

Отримані в магістерській кваліфікаційній роботі результати можна використати для побудови системи користувачів та їх комунікації для відео сервісу.

Ключові слова: Angular framework, YouTube, API, Mongo DB, Express.js, HTML, CSS, JavaScript, TypeScript, SCSS, BCrypt, Unit tests, Integration tests, E2E tests, Heroku, Jasmine, Karma, Cypress, фреймворк, препроцесор, сховище, гіпертекстова розмітка, таблиця каскадних стилів, веб інтерфейс.

ANNOTATION

Zbytkivskyi V.S. Development of a user system and their communication for a video service: master's qualification thesis on the specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 99 p.

In Ukrainian language. Bibliographer: 31 titles ; fig. : 27; tabl. 8.

In this master 's work, a video streaming service system for the transfer of video content using YouTube Api, Angular framework and Mongo DB has been developed. This system is available and can be used free, and it is easy in u se, modular and scalable. The components of the system include: NgRx, which is used to ensure the predicted state of the container, Angular components for building the system from individual modules, Mongo DB for saving data and content, YouTube Api for playing content, HTML, CSS, JavaScript, TypeScript for user interface.

The results obtained in the master's qualification work can be used to build a system of users and their communication for the video service.

Keywords: Angular framework, YouTube, API, Mongo DB, Express.js, HTML, CSS, JavaScript, TypeScript, SCSS, BCrypt, Unit tests, Integration tests, E2E tests, Heroku, Jasmine, Karma, Cypress, framework, preprocessor, storage, hypertext markup, cascading stylesheet, web interface.

ЗМІСТ

ВСТУП

Актуальність роботи. На даний час більш актуальним стає отримання інформації через інтернет за допомогою медіа ресурсів. Саме тому відео платформа є не заміним джерелом інформації. За допомогою відео платформ людина має змогу дізнатись всі останні новини у світі, розважатись, розслаблятись, не відходячи від ПК або телефону.

Інформація у сучасному світі є надзвичайно важливою, вона є основою всіх сфер людської діяльності. Люди отримують та обробляють інформацію різними способами, в більшості газети, журнали, телевізор, інтернет.

Сучасні технології дозволяють самостійно розробити систему користувачів і комунікації для відео платформи з використанням безкоштовних компонентів. Одним з популярних на даний час є безкоштовний і простий для використання фреймворк Angular, який може забезпечити більш простіший та масштабований процес розробки системи.

Зростання популярності відео контенту призводить до того, що відео стає все більш популярним способом споживання інформації та розваг. Платформи, такі як YouTube, Netflix, Amazon Prime, та інші, залучають мільйони користувачів щодня. Розробка ефективних систем користувачів і комунікації для таких платформ допомагає залучати та утримувати аудиторію.

Наразі існує велика конкуренція серед відео сервісів. Актуальним є вдосконалення конкурентних платформ, які повинні пропонувати користувачам зручну і персоналізовану взаємодію, що робить системи користувачів і комунікації стратегічно важливими. Обов’язково потрібно враховувати, що за останні роки користувачі стали більше взаємодіяти між собою на відео платформах через коментарі, рецензії, спільні перегляди, тому розробка системи комунікації допомагає підтримувати активну спільноту, потрібну користувачам.

При розробці систем потрібно розуміти потреби користувачів і пропонувати їм індивідуально налаштований контент. Споживачі очікують доступу до відео з будь-якого пристрою, зручності в користуванні та можливості

спілкування з іншими користувачами. Розробка ефективної системи комунікації може задовольнити ці очікування. Також захист особистих даних користувачів стає все важливішим питанням, та при розробці системи користувачів потрібно включати заходи для забезпечення безпеки і приватності даних.

Таким чином *актуальним є* розробити систему користувачів та їх комунікації для відео сервісу, що дозволить розширити функціонал існуючих систем, враховуючи їх недоліки, та легко модифікувати системи користувачів та їх комункацій враховуючи динаміку ринку та потреби користувачів.

Розробка такої системи також допоможе платформам покращувати конкурентоспроможність, залучати та утримувати аудиторію, а також збільшувати конкурентоспроможність та задоволення зростаючих потреб споживачів у цьому сегменті ринку. через ефективну систему комунікацій користувачів.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета магістерської роботи полягає в розширенні функціональних можливостей відео сервісів для передачі, збереження та відтворення відео-контенту шляхом розширення функціональних можливостей та поєднання сучасних технологій, що на відміну від існуючих систем дозволять легко та швидко модифікувати систему, архітектуру та складові, відповідно до потреб ринку. Також запропонований підхід забезпечує можливість впровадження системи на різних додатках, в тому числі на Web, Android та iOS системах.

Для досягнення поставленої мети необхідно **розв'язати наступні задачі:**

1. Провести аналіз існуючих засобів і методів комунікації користувачів та існуючих аналогів системи користувачів для відео сервісу та програмних засобів і бібліотек та обрати методи розв’язання задачі.

2. Покращити метод підвищення модульності та динамічності системи користувачів відео сервісу.

3. Удосконалити метод поєдання технологій для покращення комунікацій системи користувачів.

4. Розробити програмне забезпечення, в тому числі розробити компоненти для обробки API та веб-інтерфейс.

5. Протестувати систему на відповідність до початкових вимог та порівняти з аналогами.

Об'єктом дослідження – процес комунікації користувачів у відео сервісі.

Предметом дослідження –методи та засоби комунікації користувачів під час перегляду відео контенту та доступу до функціоналу з різних пристроїв.

Методи дослідження. У процесі роботи над системою використовуються методи системного аналізу для візуалізації інтерфейсу користувача, методи обробки інформації для визначення взаємодій між користувачами та методи розробки та тестування програмного забезпечення для розробки Api та інстерфейсу системи.

Наукова новизна отриманих результатів:

1. Удосконалено метод розробки програмного забезпечення, що вирішує питання комунікації користувачів і на відміну від існуючих систем дозволяє легко та швидко модифікувати систему, архітектуру та складові за рахунок модульності та динамічності системи, що дає можливість підтримувати систему актуальною.

2. Подальшого розвитку отримав метод поєдання технологій, а саме NgRx, який застосовується для забезпечення передбачуваного стану контейнера, Angular компоненти для констроювання системи з окремих модулів, Mongo DB для збереження даних та контенту, YouTube Api для відтворення контенту, HTML, CSS, JavaScript, TypeScript для юзер інтерфейсу, що дозволяє покращити якість систем користувачів та їх комунікацій для відео сервісу.

Практична цінність отриманих результатів полягає в розробці алгоритмічних засобів та програмного забезпечення відео сервісу для комунікації користувачів з можливістю легкої модифікації за рахунок модульності системи, інформування користувача щодо змін у системі.

Розроблена система забезпечує безпеку і приватність даних користувача та підтримує використання у різних додатках.

У роботі використовується Angular framework для створення модульної та масштабованої структури проекту. За допомогою NgRx забезпечується легкий контроль за станом проекту, а використання Mongo DB дозволяє збереження контенту та інформації користувача.

Особистий внесок здобувача. усі результати, що наведені у магістерській кваліфікаційній роботі, отримані самостійно.

Апробація результатів та публікації. Результати роботи було апробовано на Всеукраїнській науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2022)» [1, 2] та на LII науково-технічній конференції факультету інформаційних технологій та комп'ютерної інженерії (2023)» [3].

Впровадження результатів роботи

Основний модуль роботи прийнятий *на отримання свідоцтва авторського права на твір* (С202307451 від 20.10.2023).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ СИСТЕМИ КОРИСТУВАЧІВ ТА ЇХ КОМУНІКАЦІЇ ДЛЯ ВІДЕО СЕРВІСУ

1.1 Огляд існуючих систем користувачів та комунікації для відео сервісу

1.1.1 Система «YouTube»

"YouTube" - це популярний відеохостинг, який пропонує послуги для розміщення відеоматеріалів. Ця платформа була заснована 14 лютого 2005 року трьома співробітниками компанії PayPal: Чадом Герлі, Стівеном Чені і Джаведом Карімом. Вона є частиною корпорації Google. На момент серпня 2019 року YouTube був другим за популярністю відвідуванням веб-сайтом в Інтернеті. [4]

Користувачі мають можливість додавати, переглядати і залишати коментарі під відеозаписами на YouTube. Завдяки своїй простоті та легкості використання, YouTube став однією з найбільш популярних платформ для розміщення відеофайлів. На цій платформі представлені як професійні, так і аматорські відеозаписи, включаючи відеоблоги.

YouTube є одним з найбільш відвідуваних веб-сайтів у світі, і щотижня він привертає мільйони коментарів від своїх користувачів. Усі ці коментарі мають бути пов’язані з інформацією, представленою у відеоролику, до якого вони спрямовані. Однак, багато з коментарів є суто соціальними, містять рекламу, нецензурну лексику та не мають відношення до змісту відео. Тому вони не є корисними для багатьох людей, яким може бути цікаво знайти цікаві думки про переглянуте відео, прочитавши коментарі. [4]

Під час створення та підтримки системи використовують такі компоненти: Apache, Python, Linux (SuSe), MySQL, а також замість Apache використовують lighttpd і psyco (динамічний компілятор Python у C).

Оцінивши проблему та оцінивши сучасний рівень техніки в галузі, YouTube дійшли висновку, що найкращим підходом для досягнення

оптимального рішення було б використовувати систему з архітектурою, що являє собою поєднання бібліотек із відкритим кодом і зовнішніх ресурсів, об’єднаних у систему, розроблену для створення хорошого класифікатора релевантності коментарів. Основною характеристикою системи є поділ обробки на два основних етапи:

- Етап попередньої обробки для видалення спаму та інших типів коментарів, які вважаються нерелевантними для будь-якого відео.
- Етап рейтингу релевантності (називається класифікатором релевантності), який призначає оцінку кожному коментарю, що не був відфільтрований на першому етапі.

Класифікація тексту не є тривіальною проблемою, особливо при роботі з невеликими фрагментами тексту, такими як коментарі користувачів. Підхід YouTube для ранжування коментарів відповідно до їхньої відповідності відео є хорошим, хоча він має свої обмеження. Наприклад, він застосовний лише до коментарів, написаних англійською мовою, і погано працює з менш відомими піснями/виконавцями. [4]

Двоетапний метод вирішення проблеми можна використовувати для створення хорошого інструменту рейтингу релевантності для коментарів у системі. Перший крок складається з попередньої фільтрації вхідних даних, а другий крок поєднує вилучення теми з етапом обчислення релевантності коментаря. Для зберігання метаданих (користувачі, описи) використовують MySQL [4].

Архітектура YouTube включає в себе розподілену систему, яка складається з великої кількості мікросервісів, що відповідають за різні аспекти функціоналу платформи. Мікросервіси реалізовані для обробки завдань, таких як керування відео, збереження даних користувачів, обробка рекламних запитів, та інші. Архітектурний стек включає в себе використання технологій, таких як Java, Python, Go, та баз даних, таких як Bigtable та MySQL. YouTube використовує контентну доставку через CDN для забезпечення високої швидкості

завантаження відео для користувачів у різних регіонах. Також, архітектура підтримує стрімінг відео в реальному часі та інші функціональні можливості [4].

Загальна архітектура системи YouTube зображена на рисунку 1.1

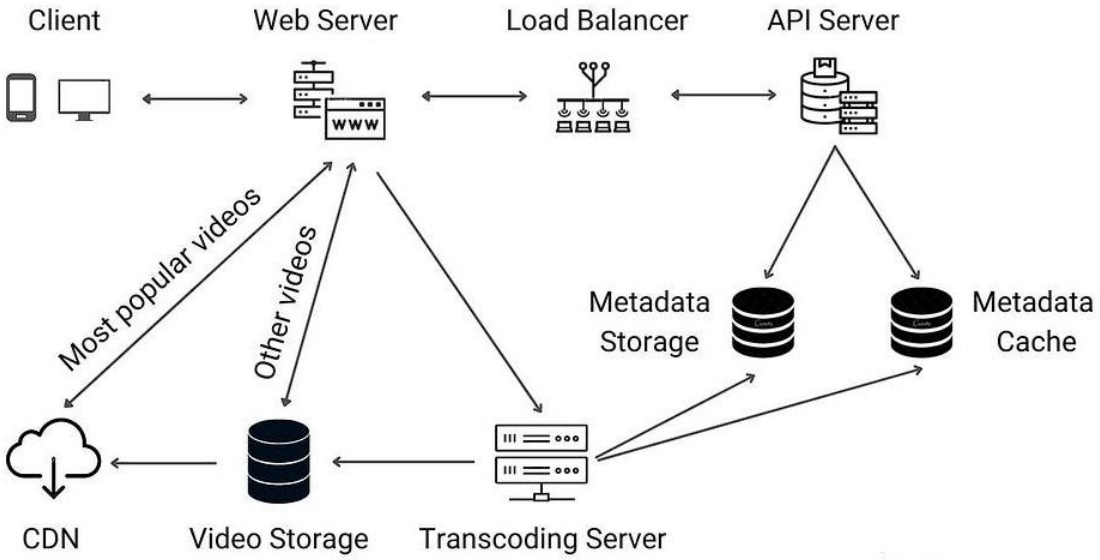

Рисунок 1.1 – Схема архітектури системи YouTube

Client: комп’ютер, мобільний телефон тощо.

Video Storage: це система зберігання BLOB для зберігання транскодованих відео. Великий двійковий об’єкт (BLOB) – це колекція двійкових даних, що зберігаються як єдине ціле в системі керування базою даних. [4]

Transcoding Server: використовується для перекодування або кодування відео, яке перетворює відео на кілька форматів із довільною роздільною здатністю (144p, 240p, 360p, 480p, 720p, 1080p і 4K), щоб забезпечити найкраще потокове відео для різних пристрої та вимоги до пропускної здатності.

API server: використовується для обробки всіх запитів, крім потокового відео. Це включає рекомендації щодо кормів, генерування URL-адреси для завантаження відео, оновлення бази даних метаданих і кешу, реєстрації користувачів тощо. Цей сервер спілкується з різними базами даних, щоб отримати всі дані. [4]

Web Server: використовується для обробки вхідних запитів від клієнта. Залежно від типу запиту, він направляє запит на сервер додатків або сервер транскодування.

CDN: закодовані відео також зберігаються в CDN для швидкої потокової передачі. Коли відтворюється відео, це популярне відео транслюється з CDN.

Load Balancer: використовується для рівномірного розподілу запитів між серверами API [4].

Metadata Storage: використовується для зберігання всіх метаданих відео, таких як назва, URL-адреса відео, мініатюри, інформація про користувача, кількість переглядів, оцінки "подобається", коментарі, розмір, роздільна здатність, формат тощо. Він сегментується та реплікується для забезпечення продуктивності. [4]

Metadata Cache: використовується для кешування метаданих відео, інформації про користувача та мініатюр для кращої продуктивності читання.

Оскільки YouTube є дуже завантаженим сервісом, він має різні API для безперебійного виконання операцій. Тому для реалізації системи використовується архітектура SOAP або REST.

1.1.2 Система «Twitch»

"Twitch" - це платформа для онлайн-відеотрансляцій, що належить компанії Twitch Interactive, дочірній компанії Amazon.com. Вона була запущена в червні 2011 року як відгалуження від загальної платформи відеотрансляцій, відомої як Justin.tv. Основний акцент сайту робиться на трансляціях відеоігор, включаючи ефіри кіберспортивних змагань, творчий контент, розділ "реальне життя" та останнім часом музичні шоу. [5]

Система комунікацій користувачів Twitch є розподілено. системою реального часу з високою масштабованістю, написаною на Go. Вона доставляє сотні мільярдів повідомлень на день користувачам, які переглядають відео через наші власні протоколи, а також підтримує IRC як протокол зв’язку, що полегшує

розробникам створення ботів IRC для додавання власних функцій чату. Основні компоненти чату включають:

Edge — отримує та розповсюджує повідомлення між клієнтами та серверними службами. Edge передає протокол IRC через необроблений TCP і WebSockets.

Pubsub — внутрішній розподіл повідомлень між вузлами Edge. Pubsub і Edge об'єднуються, щоб утворити ієрархічну систему розповсюдження повідомлень, яка виконує масове розповсюдження. [5]

Clue — Виконує аналіз дій користувачів і надсилає їх через конвеєр бізнес-логіки. Clue збирає дані з багатьох джерел, включаючи бази даних, внутрішні API та кеші.

Room — Відповідає за список глядачів. Room збирає, зберігає та запитує дані членства в усій системі, щоб отримати списки глядачів для кімнати чату кожного каналу.

На сайті можна переглядати вміст в режимі прямого ефіру або скористатися відеозапитами. Відеосистема відповідає за передачу відео від телевізійної компанії до глядачів і включає наступні ключові компоненти:

Приймання відео - відео отримується в форматі RTMP та подальше транспортується до системи перекодування. [5]

Система перекодування - вона приймає вхідний потік RTMP від джерела і транскодує його в кілька потоків HLS. Цей процес реалізований за допомогою комбінації мов програмування C/C++ та Go.

Розповсюдження та розподіл - це розсилання потоків HLS на різні географічно розташовані точки доступу (POP), щоб забезпечити найвищу якість потокового відео. Цей процес також в основному реалізований за допомогою мови програмування Go. [5]

Відео за запитом (VOD) - всі вхідні відеосистеми використовуються та архівуються для системи відео за запитом (VOD).

Крім послуг у режимі реального часу, таких як відео та чат, команда Twitch також надає різноманітні інші сервіси, включаючи, але не обмежуючись:

Веб-API, які дозволяють користувачам управляти своїми профілями, підписками та налаштовувати їх.

Сервіси пошуку та виявлення, що допомагають знайти потрібний відеоконтент.

Системи монетизації - це інструменти, які дозволяють керувати рекламою та підписками, а також слідкувати за тим, щоб партнери Twitch мали можливість отримувати дохід. [5]

Адміністративні інструменти призначені для вирішення проблем і задач користувачів через службу підтримки. Ця система побудована на поєднанні фреймворка Rails, мови програмування Go та різноманітних програм з відкритим вихідним кодом, які використовуються для маршрутизації, кешування та зберігання даних. Крім цього, було розроблено численні вертикальні та горизонтальні розділи даних і API даних, щоб покращити масштабованість системи Twitch на технічному та організаційному рівнях. [5]

Всі ці сервіси є важливими, але вони стають повноцінними та корисними тільки завдяки простим у використанні клієнтським додаткам, які забезпечують зручний користувацький досвід. Зокрема, наявність настільного веб-додатка, що спочатку був запуском як звичайний додаток на основі Rails, але з часом переріс у додаток Ember, а також наявність нативних програм для основних мобільних і планшетних платформ (iOS і Android), грає важливу роль у створенні задоволення користувачів [5].

Science Engineering розробляє системи для збору даних та інструменти для аналізу, які допомагають вивчати, як користувачі взаємодіють з продуктом Twitch. Серед цих систем наступні:

Конвеєр даних, який виконує збір, очищення та завантаження понад мільярда подій щоденно до сховища даних.

Потокові конвертери, які узагальнюють показники майже в реальному часі для користувачів [5].

Веб-сайт має дуже велику базу лояльних користувачів. Спільноти нараховуються сотнями чи тисячами і зазвичай зосереджені навколо певних

каналів. Багато користувачів знайшли нових друзів через ці спільноти та регулярно використовують їх для спілкування з іншими геймерами та улюбленими стримерами. Відчуття спільності разом із роздачами продуктів, веселими подіями для кількох стримерів, благодійними заходами та TwitchCon (конференцією для стримерів і глядачів) — усе це відрізняє веб-сайт від конкурентів і робить його дуже популярним. [5]

Підсумовуючи, можна сказати що, що Twitch досить добре виконує більшість евристик в інформаційній архітектурі. Зрозуміло, що ці принципи враховуються командами дизайну та розробки сайту. У свою чергу, цей факт частково відповідає за успіх Twitch в індустрії. Це тому, що якби вони не дотримувалися цих основних евристичних принципів, їхній веб-сайт і продукт у цілому були б набагато менш інтуїтивно зрозумілими та приємними у використанні.

1.1.3 Система «Netflix»

"Netflix" - це американська компанія, яка надає медійні послуги та є виробником контенту, із штаб-квартирою в Лос-Гатос, Каліфорнія. Заснована у 1997 році Рідом Гастінгсом і Марком Рендолфом у Скотс-Валлі, Каліфорнія. Основним напрямком діяльності компанії є надання передплатних послуг із потокового відтворення великої бібліотеки фільмів і телесеріалів через Інтернет. [6]

Netflix працює на двох хмара AWS і Open Connect. Ці дві хмари працюють разом як основа Netflix, і обидві несуть велику відповідальність за надання користувачам найкращого відео. [6]

Додаток має в основному 3 компоненти:

Client: пристрій (інтерфейс користувача), який використовується для перегляду та відтворення відео Netflix. Телевізор, XBOX, ноутбук або мобільний телефон.

OC (Open Connect) або Netflix CDN: CDN — це мережа розподілених серверів у різних географічних місцях, а Open Connect — це власна глобальна CDN (мережа доставки вмісту) Netflix. Він обробляє все, що включає потокове відео. Розповсюджується в різних місцях, і коли ви натискаєте кнопку відтворення, відеопотік із цього компонента відображається на вашому пристрої. Отже, якщо ви намагаєтеся відтворити відео в Північній Америці, відео буде подано з найближчого відкритого підключення (або сервера) замість вихідного сервера (швидша відповідь від найближчого сервера). [6]

Backend (база даних): ця частина обробляє все, що не передбачає потокового відео (до того, як натискається кнопка відтворення), як-от завантаження нового вмісту, обробка відео, їх розповсюдження на сервери, розташовані в різних частинах світу, і керування мережевим трафіком. Більшість процесів виконує Amazon Web Services. [6]

Інтерфейс Netflix написаний на ReactJS головним чином з трьох причин: швидкість запуску, продуктивність під час виконання та модульність. Загальна архітектура системи Netflix зображена на рисунку 1.2.

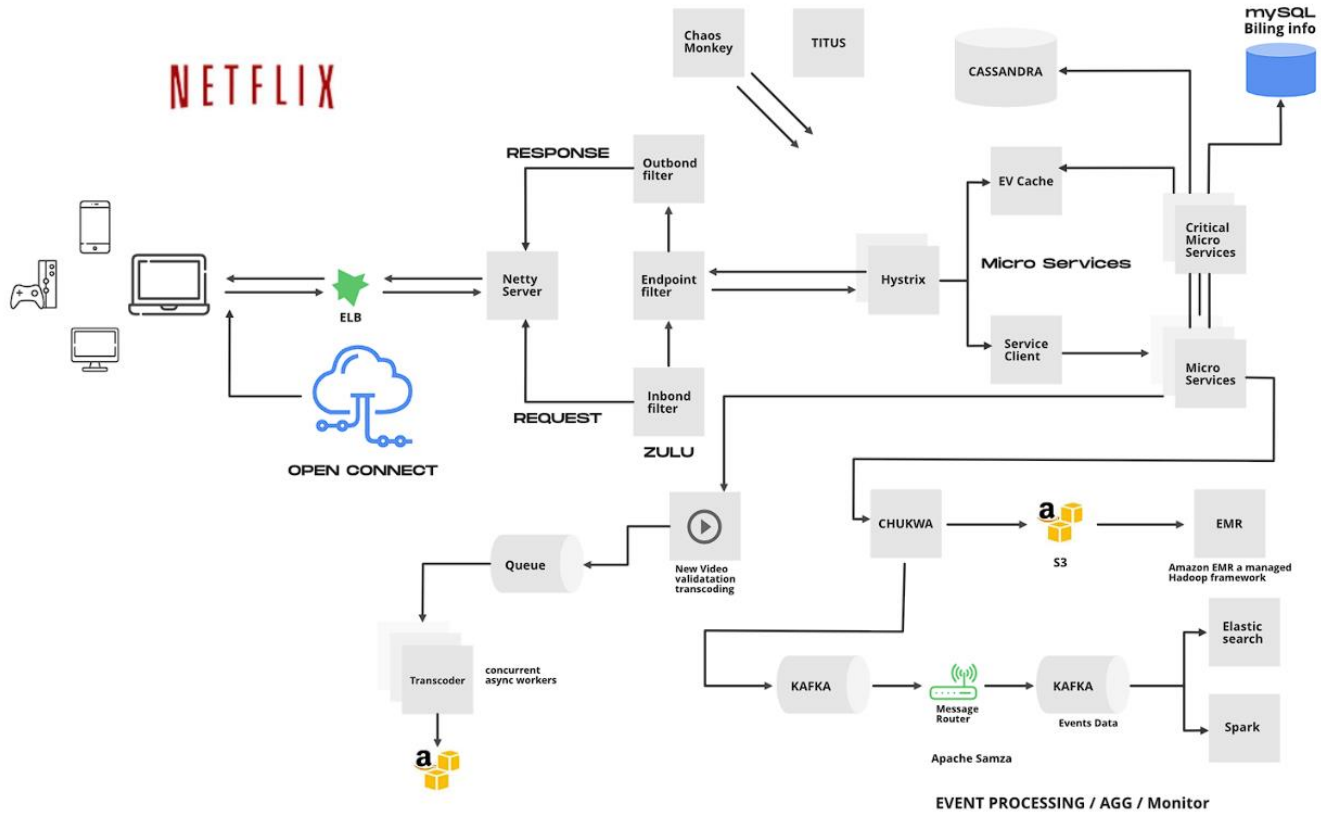

Рисунок 1.2 – Схема архітектури системи Netflix

Архітектура Netflix побудована як набір послуг. Цей тип архітектури відомий як архітектура мікросервісів, і вона забезпечує всі API, необхідні для додатків і веб-додатків. Коли запит надходить до кінцевої точки, він викликає інші мікросервіси для отримання необхідних даних, і ці мікросервіси також можуть запитувати дані з різних мікросервісів. Після цього повна відповідь на запит API надсилається назад до кінцевої точки.

В архітектурі мікросервісів служби мають бути незалежними одна від одної, наприклад, служба зберігання відео буде відокремлена від служби, відповідальної за перекодування відео. [6]

Окремі критичні мікросервіси можна відокремити у критично важливі сервіси (або кінцеві точки чи API) і зробити їх менш залежними або незалежними від інших сервісів. Також зробивши деякі важливі служби залежними лише від інших надійних служб та вибираючи критичні мікросервіси, можливо включити всі основні функції, як-от пошук відео, навігація до відео, натискання та відтворення відео тощо. Таким чином кінцеві точки є високодоступними та навіть у найгіршому випадку користувач зможе робити основні речі. [6]

Ідея полягає в тому, щоб розробити службу таким чином, що якщо одна з кінцевих точок видає помилку або якщо вона не обслуговує запит своєчасно, потрібно переключитися на інший сервер і виконати роботу. Замість того, щоб покладатися на певний сервер і зберігати стан на цьому сервері, потрібно направити запит до іншого екземпляра служби та автоматично створити новий вузол, щоб замінити його. Якщо сервер перестає працювати, його замінить інший. [6]

Netflix використовує два різні алгоритми для створення системи рекомендацій:

1. Спільна фільтрація: ідея цієї фільтрації полягає в тому, що якщо два користувачі мають подібну історію оцінок, то в майбутньому вони поводитимуться однаково. Наприклад, розглянемо наявність двох осіб. Одній людині фільм сподобався, і він поставив фільм на гарну оцінку. Тепер існує

велика ймовірність, що інша особа також матиме подібний шаблон і він/вона зробить те саме, що зробила перша особа.

2. Фільтрування на основі вмісту: ідея полягає в тому, щоб відфільтрувати ті відео, які схожі на відео, яке користувачеві сподобалося раніше. Фільтрація на основі вмісту сильно залежить від інформації з продуктів, такої як назва фільму, рік випуску, актори, жанр. Отже, щоб реалізувати цю фільтрацію, важливо знати інформацію, що описує кожен елемент, а також бажано мати якийсь профіль користувача, який описує, що подобається користувачеві.

1.2 Огляд існуючих програмних засобів та бібліотек

Система повинна виконувати певні завдання, включаючи наступне: при отриманні даних у рядку пошуку, система повинна викликати API YouTube та отримувати результати, які відповідають цьому запиту. [7]

Після отримання результатів, ці дані повинні бути збережені у списку або масиві. Метадані кожного відео повинні бути структуровані у словнику або об'єкті в цьому масиві. Тобто, для кожного відео потрібно створити окремий об'єкт та додати його до масиву. Після створення такого масиву відеооб'єктів, ми можемо переглянути його та створити компоненти, які будуть відображатися у правій частині нашого інтерфейсу. Все це необхідно для отримання результатів пошуку. [7]

Після того, як список буде відображено у нашому браузері, ми можемо додати простий ефект, який реагує на клік на відео або його назву. Функція onClick оновить атрибут src тегу iFrame, замінюючи ідентифікатор відео відтворюваного відео на ідентифікатор відео, яке ми обрали зі списку.

Також необхідно мати можливість створити користувача та надати йому певні права та можливості в системі.

Фреймворк - це фундаментальна основа програми, складаючись із набору бібліотек. Ці компоненти разом дозволяють користувачеві будувати власну функціональність на встановленому фундаменті. Фреймворк встановлює

структуру та підхід до архітектури вашої системи, і він визначає, як ви повинні писати свій код:

Angular - це фреймворк для створення односторінкових додатків (SPA), що включає інструменти для розробки і тестування, і реалізує архітектури MVC і MVVM, і він має відкритий вихідний код. [7]

Vue.js - також є фреймворком для створення SPA, який реалізує шаблон MVVM і теж має відкритий вихідний код. Цей фреймворк відомий своєю простотою та легкістю в освоєнні, що дозволяє розробникам швидко створювати ефективні інтерфейси. Vue.js базується на двухосновних концепціях: "директиви" для обробки DOM та "реактивність" для автоматичного оновлення відображення при зміні даних [7].

Однією з ключових особливостей Vue.js є можливість компонентного підходу до розробки, де веб-сторінку можна розглядати як дерево вкладених компонентів, кожен з яких відповідає за свою частину інтерфейсу.

Також активно використовується для створення односторінкових додатків (SPA), де весь контент завантажується один раз, а зміни на сторінці відбуваються динамічно без повторного завантаження сторінки.

Ember.js - це ще один відкритий фреймворк для розробки SPA, який використовує шаблон архітектури MVC. Він використовує парадигму конвенції понад конфігурацією, надаючи стандартизовану структуру та підходи до розробки [7].

Фреймворк надає набір готових рішень і концепцій, що дозволяє розробникам швидше створювати складні додатки. Він включає в себе систему управління маршрутами, об'єктно-орієнтовану модель даних, а також систему шаблонів для легкого відображення інтерфейсу користувача [7].

Однією з ключових особливостей Ember.js є Ember CLI, інструмент командного рядка, який допомагає автоматизувати багато аспектів розробки, включаючи створення компонентів, моделей та тестів. Фреймворк також активно підтримує ідеї конвенції над конфігурацією, що сприяє швидкому розгортанню та зменшує витрати на прийняття рішень розробниками [7].

У 2010 році з'явилася бібліотека Backbone.js, яка реалізує паттерн MPV (Model-Presenter-View). За допомогою Backbone.js можна було створювати SPA із клієнтським рутингом. Раніше робота з клієнтським рутингом була досить складною задачею, і тому більшість проектів використовували серверний рутинг для віддачі шаблонів.

На базі Backbone.js було реалізовано багато додатків, і деякі з них продовжують підтримуватися досі. Однією з особливостей додатків на Backbone.js є використання CoffeeScript, що було розроблено тим же автором. [8]

У 2011 році з'явився Ember.js, який є ще одним фреймворком для створення односторінкових додатків (SPA) і має вбудований шаблонізатор Handlebars. Сам фреймворк ґрунтується на класичній архітектурі MVC (Model-View-Controller). Ember має активне та дружнє спільнота, яка розвиває цей інструмент і продовжує вдосконалювати його, навіть у 2022 році у фреймворку є роадмап та цілі щодо подальшого розвитку. [7]

Angular - це повноцінний фреймворк, який був розроблений для спрощення створення односторінкових додатків (SPA). Однією з головних особливостей цього фреймворку є можливість взаємодії між моделлю і відображенням, де зміна моделі призводить до автоматичної зміни відображення і навпаки. Це необхідно для забезпечення зв'язку дій користувача. Наприклад, якщо користувач вводить дані в поле, то модель оновлюється, потім відправляється запит на сервер, отримується відповідь, яка також змінює модель, і, як результат, оновлюється відображення. Цей принцип називається "two-way data binding" (двонаправлене зв'язування даних). [7]

Цей механізм дозволяє легко підключати необхідні залежності для модулів. Можна уявити використання залежностей, як команду, в якій є розробник (контролер), дизайнер (сервіс) та контент-менеджер (константа), всі разом в одному офісі (DI контейнері). Коли розробник потребує дизайн для створення інтерфейсу, він звертається до дизайнера, який, в свою чергу, отримує необхідну інформацію від контент-менеджера. Це забезпечує швидку і

ефективну роботу, оскільки всі необхідні ресурси знаходяться в одному місці. На виході розробник отримує всі необхідні компоненти для реалізації проекту. [8]

Angular включає у себе численні додаткові інструменти:

- Модуль анімацій (Animations) для створення анімаційних ефектів.
- Тест-раннер Karma для виконання unit-тестів.
- Фреймворк для unit-тестів Jasmine.
- Фреймворк Protractor для проведення end-to-end (e2e) тестів.
- Підтримка локалізації та інтернаціоналізації (інтернаціоналізація – це процес адаптації програми під різні мови та регіони).
- REST-клієнт, що включає в себе модуль або бібліотеку для клієнт-серверної взаємодії (аналогічно до fetch або XHR).
- Бібліотека Router для роботи з браузерною історією, що дозволяє виконувати переходи на різні адреси всередині програми.

Компонентний підхід полягає у використанні та перевикористанні окремих модулів, які включають в себе логіку для роботи з представленням, розміткою, стилями, подіями і, можливо, навіть моделями. При цьому кожен компонент вирішує конкретне завдання і виконує лише те, для чого він призначений. Наприклад, навігаційний компонент на простому веб-сайті може бути перенесений в іншу програму, змінивши лише конфігурацію. [7]

1.3 Концепція системи користувачів та їх комунікації для відео сервісу

Необхідно створити систему відеострімінгового сервісу подібну до YouTube, оскільки всі відеодані надходять безпосередньо з YouTube. У системі повинен бути присутній рядок пошуку, який виконує офіційний пошук на YouTube і відображає результати. Кожен раз, коли користувач натискає на відеокарту, він або вона буде перенаправлено на сторінку відео. Було також створено сервер і базу даних для управління користувачами, зберігання їх даних

та додавання функціоналу, такого як підписка, лайк, історія переглядів, налаштування, зміна профілю користувача, коментування і багато іншого. [9].

Angular - це фреймворк на основі JavaScript з відкритим вихідним кодом, який надає всі необхідні засоби для розробки клієнтської логіки вашої системи. Цей фреймворк використовує концепцію модулів, які є окремими частинами коду, кожна з яких має власну специфічну функцію. Angular в основному складається з таких модулів, які можуть бути ізольованими або взаємодіяти один з одним. [9]

Модулі в додатку можна організувати різними способами, і вивчаючи різні архітектурні структури, можна визначити, як додаток буде масштабуватися з розвитком, а також навчитися ізолювати код та знижувати взаємозалежність.

Однією з основних характеристик цього фреймворку була двостороння прив'язка. Проте з часом такий підхід став призводити до проблем з продуктивністю. Виявилось, що двостороння прив'язка не є необхідною у всіх випадках. Angular 2+ надає можливість використовувати її, але тепер це потрібно робити явно, що дозволяє розробнику керувати потоками даних та їхнім напрямком. Це також надає додатку більшу гнучкість у роботі з даними, оскільки можна вказати, як вони передаються. [9]

Директиви - це спосіб розширення HTML за допомогою додаткових елементів. Атрибутивні директиви дозволяють змінювати властивості елемента. Структурні директиви, з свого боку, впливають на макет, додаючи або видаляючи елементи DOM. [9]

Наприклад, ngSwitch і ngIf - це структурні директиви, які оцінюють параметри і визначають, чи існують конкретні частини DOM. Атрибутивні директиви визначають поведінку, яка прикріплюється до елемента, компонента або інших директив. [9]

Освоївши використання обох видів директив, можна розширити можливості власного програмного продукту та зменшити повторення коду в проекті. Атрибутивні директиви також сприяють централізації певної поведінки, яка використовується в різних частинах програми. [9]

У кожній програмі існує життєвий цикл, який визначає, як об'єкти створюються, обробляються і потім видаляються. У Angular є можливість обробляти ключові моменти цього циклу та спрямовуватися на певні моменти часу чи події. Це дозволяє створювати відповідні реакції та налаштовувати поведінку відповідно до різних етапів життєвого циклу компонента.

Роль компонента в загальному контексті програми визначається тим, чи він є "простим" чи "розумним". "Прості" компоненти, зазвичай, не відслідковують стан, їхню поведінку легко передбачити і зрозуміти - і в деяких випадках саме такими повинні бути компоненти. [9]

"Розумні" компоненти більш складні в розумінні, оскільки вони взаємодіють з входами та виходами. Для ефективного використання можливостей Angular, важливо зрозуміти архітектуру "розумних" та "простих" компонентів, що допоможе писати більш продуктивний код та забезпечити правильну взаємодію всередині програми.

Стан програми в кінцевому рахунку визначає дані, які користувач бачить на екрані. Angular має свою систему управління станами, але NgRx справляється з централізацією станів і зв'язаних з ними даних набагато більш ефективно. У ланцюжку відносин між батьківськими та дочірніми елементами дані можуть бути втрачені, а NgRx створює для них централізований сховище, що вирішує цю проблему. [9]

У фреймворку Angular програми будуються на основі компонентів. Проте не всі компоненти є статичними - деякі з них створюються динамічно під час виконання програми, а не завчасно під час компіляції. Компоненти, які створюються динамічно в процесі роботи програми, називаються динамічними компонентами. Статичні компоненти припускають певну незмінність: вони очікують певних значень на вході та виході та демонструють передбачувану поведінку.

Динамічні компоненти, навпаки, створюються за потребою під час роботи програми. Їх зручно використовувати у випадках, коли програма повинна

слухати зовнішні події, реагувати на зміни стану або реагувати на події, що відбуваються на сторінці. [9]

У разі, якщо програма має доступ до облікового запису, то необхідно використовувати механізми захисту маршрутів. Це необхідно для забезпечення захисту конкретних сторінок від несанкціонованого доступу, що часто є обов'язковою вимогою. Захисники маршрутів функціонують як посередники між маршрутизатором і запитуваним маршрутом, визначаючи, чи слід дозволяти доступ до певного маршруту. [9]

Для прискорення завантаження програми і зроблення роботи з нею більш зручною також корисними є попереднє та відкладене завантаження. Важливо відзначити, що ці інструменти не тільки дозволяють визначити, чи слід завантажувати конкретний набір зображень, але й сприяють поліпшенню архітектури програми та завантаженню різних її частин, які можуть розташовуватися в різних доменах і областях визначення. [9]

1.4 Висновки

У першому розділі розглянуто основні аспекти роботи, включаючи огляд та аналіз існуючих користувацьких систем відео-сервісів. Також представлені ключові поняття, пов'язані з цією темою, і наведено переваги розробки системи для користувачів і контенту, використовуючи фреймворк Angular.

Актуальність полягає в створенні повноцінної і самодостатньої системи відео-стрімінгу на базі безкоштовного фреймворку Angular та додаткових бібліотек і програмного забезпечення. Ця система відрізняється від існуючих можливістю легкої розширюваності і оновлення складових та компонентної архітектури відповідно до нових технологій і розробних методів. Крім того, вона може застосовуватися на різних платформах, включаючи веб, Android і iOS.

РОЗДІЛ 2
ПРОЕКТУВАННЯ СТРУКТУРИ ТА КОМПОНЕНТІВ СИСТЕМИ

2.1 Планування реалізації сховища за допомогою «NgRx»

Організація втілення сховища за допомогою NgRx в Angular передбачає створення централізованого резервуару для ефективного управління станом програмного забезпечення. NgRx використовує концепції Redux та забезпечує засоби для ефективного управління станом програми. Для належного організування структури сховища необхідно чітко визначити основні об'єкти стану, які будуть зберігатися в цьому сховищі. Рекомендується розділити ці об'єкти на модулі залежно від функціональності додатку. [8]

Також важливо визначити дії (actions), які будуть відповідати за зміну стану, та редуктори (reducers), що будуть обробляти ці дії та вносити зміни в стан.

Для створення ефектів рекомендується використовувати NgRx Effects з метою обробки асинхронних операцій, таких як виклики HTTP-запитів або обробка подій.

Крім того, слід визначити ефекти, які будуть реагувати на дії та виконувати необхідні операції. З'єднання з Angular зазвичай відбувається за допомогою використання NgRx Store у системі, часто через включення StoreModule.forRoot() в NgModule. Взаємодія з сховищем рекомендується здійснювати в компонентах та сервісах додатку за допомогою використання дій, редукторів і селекторів.

Для видобутку необхідних даних із сховища необхідно визначити селектори (selectors). Ці селектори надають можливість компонентам отримувати доступ до стану безпосередньо [8].

Для здійснення тестування необхідно створити юніт-тести, які перевірять правильність функціонування дій, редукторів, ефектів і селекторів.

Для тестування NgRx-коду рекомендується використовувати інструменти, такі як Jasmine і TestBed. Для покращення ефективності додатку рекомендується

розглянути можливості оптимізації, такі як використання бібліотеки reselect для селекторів або застосування мемоізації для зберігання обчислених значень.

Для впровадження системи використовується серверний сервіс, що гарантує коректну роботу сховища в продуктивному середовищі.

Для забезпечення підтримки та моніторингу слід досліджувати логи та проводити моніторинг із метою виявлення можливих проблем у роботі сховища в продуктивному середовищі.

У єдиному екземплярі сховища зберігається весь стан додатка. NgRx Store формується, об'єднуючи стани, які повертаються кожним з редукторів (навіть найпростіший Angular додаток може використовувати всього один редуктор). [8]

NgRx Store включає у себе такі основні функції:

1. Зберігання та надання доступу до стану програми;
2. Дозвіл на оновлення стану шляхом використання методу dispatch() через визначені дії;
3. Реєстрація функцій, що будуть викликані при будь-якій зміні стану, за допомогою методу subscribe() [8]

Використання NgRx для реалізації сховища сприятиме створенню додатка з централізованим управлінням станом, що спрощує процес розробки та підтримки.

NgRx, як бібліотека, реалізує принципи роботи Redux для додатків Angular. Головна мета NgRx полягає в централізації та максимальному упорядкуванні управління всіма станами програми.

Досягнення цієї мети в NgRx досягається завдяки декільком фундаментальним принципам, які закладені в бібліотеці:

1. Єдиний принцип джерела даних про стан реалізується через наявність сховища (store);
2. Доступність стану обмежена лише можливістю читання;
3. Стан системи змінюється виключно через дії (actions), які обробляються чистими функціями - редукторами (reducers). [8]

Унікальні події відображаються за допомогою дій, що мають місце в додатку Angular, і використовуються для передачі даних у сховищі. NgRx Actions функціонують як єдина джерело даних для сховища.

Створення дій реалізується через використання інтерфейсу Action, який включає єдине обов'язкове текстове поле "type". У випадку, коли дії описують події, які трапилися в додатку, редуктори визначають, як необхідно змінити стан програми Angular у відповідь на виниклу подію [8].

Редуктори NgRx додаються до сховища та повинні представляти собою чисті функції. Редуктор отримує початковий стан і подію в якості аргументів, проте для внесення змін він має повернути новий стан і строго утримуватися від прямих змін вхідного стану. У випадку, якщо редуктор не передбачає обробки дії, він має просто повернути вхідний стан [8].

Реєстрацію всіх NgRx Effects у програмі слід проводити за допомогою модуля EffectsModule. При визначенні ефектів на рівні кореневого модуля, слід використовувати метод forRoot(); у випадку визначення на рівні вторинного - forFeature(). Масив ефектів передається як параметр обом методам. Для реалізації побічного ефекту необов'язково створювати нову дію; варто передати декоратору @Effect() об'єкт з вказівкою значення false для властивості dispatch. [8]

NgRx також дозволяє контролювати життєвий цикл ефекту за допомогою реалізації відповідних інтерфейсів:

1. OnInitEffects - повертає дію негайно після реєстрації ефекту у додатку;
2. OnRunEffects – надає можливість налаштовувати початок та завершення виконання ефекту; за замовчуванням це відбувається одночасно з роботою програм;
3. OnIdentifyEffects - реєструє NgRx Effects кілька разів, ефект реєструється в додатку Angular один раз, за замовчуванням, незалежно від кількості завантажень самого класу ефекту.

Селектори у NgRx використовуються для отримання конкретних частин глобального стану і є чистими функціями. Унікальні характеристики селекторів включають в себе мобільність, мемоізацію, можливість композиції селекторів та легкість тестування.

Функції createSelector() у NgRx використовуються для створення селекторів і можуть приймати необмежену кількість функцій, данні функції повертають конкретну частину стану системи. В останню функцію, яка повертає кінцевий результат, передаються результати виконання перших функцій як аргументи. Однією із важливих властивостей є мемоізація даних. Властивість мемоізації дозволяє уникнути повторного обчислення при виклику функції, яка вже була викликана раніше. Під час першого виклику функції запам'ятовується її повернуте значення. Це значення буде перераховане та оновлене при будь-якій зміні у наборі параметрів. У випадку відсутності змін у параметрах, повертатиметься збережене значення [8].

Побічні ефекти, використовують бібліотеку RxJS та взаємодіють з сховищем. NgRx Effects можуть генерувати нові дії, наприклад, на основі результатів виконання HTTP-запитів або повідомлень, отриманих через Web Sockets. [8]

NgRx Effects мають такі основні цілі та функції:

1. Розширити функціональність компонента, виокремивши управління станом та виконання побічних ефектів, і зосередити його роботу на отриманні станів та генерації дій;
2. Моніторинг та фільтрація потоку дій для виклику побічного ефекту при виникненні конкретної дії;
3. Здійснення побічних ефектів, як синхронних, так і асинхронних.

NgRx Effect створюються починаючи з відстеження потоку подій, який надходить через сервіс Actions та передується декоратором @Effect(). Після цього, використовуючи оператор ofType(), визначається тип дії, при виникненні якого буде виконаний побічний ефект. Цей ефект, у свою чергу, повинен

породжувати нову дію, яка передається далі у сховище. Також важливо обробляти помилки. [8]

Метод StoreRouterConnectingModule.forRoot() приймає об'єкт з налаштуваннями NgRx Router Store як необов'язковий параметр. Об'єкт, який передається, повинен реалізовувати інтерфейс StoreRouterConfig з трьома необов'язковими властивостями:

1. stateKey - це частина глобального стану, в якій зберігаються всі дані маршрутизації, за замовчуванням називається "router". Крім рядка, це також може бути селектор;
2. serializer - надає можливість налаштовувати структуру даних стану, яка передається у кожній дії;
3. navigationActionTiming - має тип Navigation Action Timing.PreActivation або Navigation Action Timing.PostActivation та визначає момент генерації дії ROUTER_NAVIGATION.

Navigation Action Timing.PreActivation означає, що ROUTER_NAVIGATION буде згенеровано до виклику всіх Guards і Resolvers, а Navigation Action Timing.PostActivation - після.

NgRx Router State використовується для зв'язування модуля маршрутизації зі сховищем, дозволяючи відслідковувати процес зміни URL в програмі Angular за допомогою дій. У NgRx Router Store передбачено п'ять дій, кожна з яких представляє одну з етапів процесу зміни маршруту:

1. ROUTER_REQUEST – початок переходу на інший URL;
2. ROUTER_NAVIGATION - сам процес переходу, що викликається до виконання всіх Guards і Resolvers;
3. ROUTER_NAVIGATED – успішний перехід на заданий URL;
4. ROUTER_CANCEL – генерується, якщо зміна URL була заблокована Guard або Resolver;
5. ROUTER_ERROR – генерується, якщо в процесі переходу виникає помилка

Інформація про тип помилки чи скасування дії в маршруті з якого здійснюється перехід міститься в ROUTER_CANCEL та ROUTER_ERROR. [8]

Потрібно правильно організувати та структурувати код, що формує сховище (Store) програми задля того щоб використання бібліотеки Angular NgRx було максимально ефективним. Схема структури сховища зображена на рисунку 2.1.

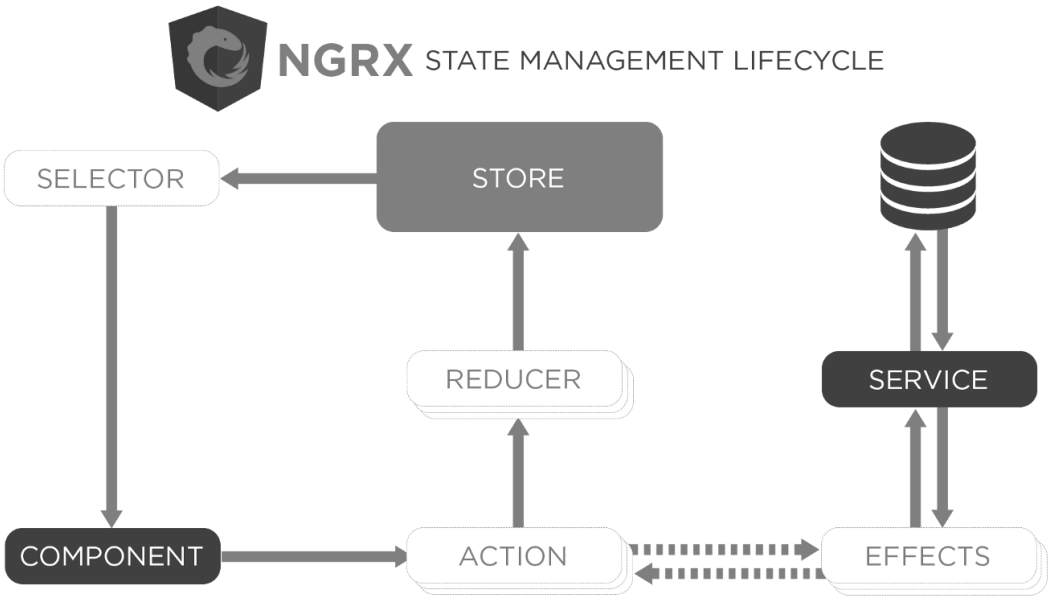

Рисунок 2.1 - Схема архітектури «NgRx»

Директорія store створюється у корені модуля, яка включає наступні піддиректорії: actions, effects, models, reducers, та index.ts..

У каталозі actions розташовані файли, що містять опис всіх дій (NgRx Actions). У кожному файлі слід описувати лише взаємопов'язані дії, причому назва файлу повинна відповідати його змісту. Наприклад, якщо впроваджується відеокаталог, то файл з діями може мати назву videos.actions.ts [8].

У каталозі effects розташовуються всі побічні ефекти (NgRx Effects) відеокаталогу. Директорія models включає в себе опис моделей всіх сутностей, які використовуються під час розробки. Для кожної сутності має бути створений окремий файл.

В директорії reducers розташовані всі редуктори (NgRx Reducers) для обробки змін стану сховища. Кожен редуктор має містити опис тієї частини

стану, за обробку якої він відповідальний. Також в даному контексті створюється Entity Adapter для керування масивом сутностей. [8]

У файлі index.ts імпортуються всі створені редуктори та формується повна модель сховища, що визначає структуру каталогу store. Після цього ця модель передається як параметр методу StoreModule.forRoot(). Крім того, в цьому файлі описуються та експортуються всі селектори сховища [8].

2.2 Метод розробки системи користувачів та комунікації відео сервісу

Компонент в рамках Angular представляє собою ізольовану частину функціоналу з власною логікою, HTML-шаблоном та CSS-стилями. Щоб клас став Angular-компонентом, його необхідно оголосити та додати декоратор @Component() з відповідними налаштуваннями. Кожен Angular-компонент відповідає за виконання конкретного завдання і не має виконувати зайвих функцій. Рекомендується уникати створення складних компонентів та дотримуватися принципу розділення функціоналу [10].

Проектування компонентів для веб-додатку на Angular охоплює процес створення окремих частин інтерфейсу користувача, що взаємодіють з користувачем. Цей процес включає аналіз вимог, розробку компонентів, визначення їх логіки, структури та дизайну, а також оптимізацію та тестування для забезпечення якості та продуктивності веб-додатку. Компоненти можуть включати навігаційні панелі, форми, таблиці, карточки, кнопки та інші елементи. [10]

Зазвичай, розробляються компоненти з мінімальним набором функцій для простої підтримки та тестування. Проте вони повинні бути гнучкими для перевикористання та уникнення копіювання коду для подібних компонентів. Не рекомендується надмірно завантажувати компоненти параметрами, які рідко використовуються. [10]

В Angular багато роботи виконується у компонентах. Кожен Angular-додаток має принаймні один кореневий компонент. Наприклад, можна створити

два компоненти в додатку: app.component.ts (який є кореневим) і all-videos.component.ts. Кореневий компонент визначається у файлі app.module.ts за допомогою декоратора @NgModule. [10]

Можна створити всього один компонент і розміщувати весь код в ньому, у середніх і великих проектах рекомендується розділяти функціональність на різні компоненти. Кожен з них відповідає за конкретний фрагмент функціоналу. В такій системі існує низка компонентів, які будуть організовані в різні структури, залежно від поточної сторінки або функціональності системи.

Компонент починається як звичайний клас, але стає Angular компонентом після застосування декоратора @Component() з метаінформацією. Цей декоратор містить наступні важливі властивості:

1. Template (Шаблон): Ця властивість дозволяє вам визначити шаблон компонента без потреби створювати окремий файл шаблону. Однак для великих шаблонів це може бути незручним, коли код стає дуже великим (близько 15-20 рядків).

2. Styles (Стилі): Ця властивість дозволяє визначити стилі компонента прямо в файлі компонента.

3. Providers (Постачальники): Ця властивість дозволяє ініціалізувати сервіси, які будуть використовуватися у цьому компоненті.

4. Animations (Анімації): Ця властивість дозволяє визначити анімації для даного компонента.

Компонентний вигляд формується через шаблон-компаньйон. Цей шаблон представляє собою HTML-структуру, яка визначає, як сам компонент буде представлений на веб-сторінці [10].

Темплейти мають ієрархічну структуру, що надає можливість управляти цілими розділами або сторінками інтерфейсу користувача як єдиним цілим, змінюючи або відображаючи їх за потребою. Шаблон, що прямо пов'язаний з компонентом, визначає, як саме цей компонент буде виглядати на сторінці. У додаток до цього, компонент може включати ієрархію темплейтів, що може

включати вбудовані шаблони, розташовані в інших компонентах. [11] Структура прикладу компонента показана на рисунку 2.2.

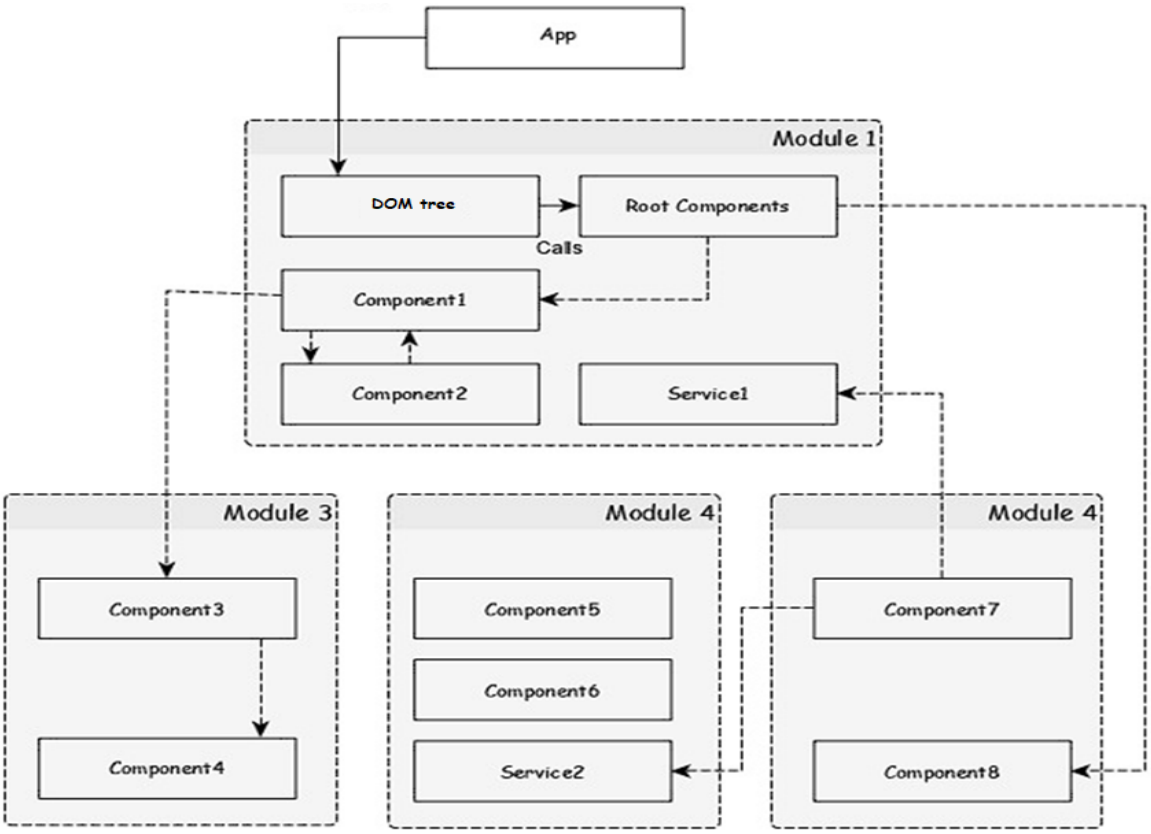

Рисунок 2.2 - Схема архітектури компонента системи користувачів

Ієрархія уявлень може включати уявлення від компонентів, які належать до одного і того ж NgModule, але також може (і часто включає) уявлення від компонентів, що визначені в різних NgModules [11].

Шаблон подібний до звичайного HTML, але з внесенням синтаксису шаблону Angular, який адаптує HTML відповідно до логіки програми, стану програми та даних DOM. Синтаксис надає можливість використовувати прив'язку даних для зіставлення даних програми із DOM, конвейери для обробки даних перед їх відображенням, а також директиви для застосування логіки програми до відображуваних даних. [11]

У шаблоні використовуються стандартні HTML-елементи, такі як <h2> і <p>, а також елементи синтаксису Angular-шаблону, такі як *ngFor, {{video.name}}, (click), [video] та <app-video-detail>. Елементи синтаксису

шаблону дозволяють Angular визначити, як візуалізувати HTML-код на екрані, використовуючи програмну логіку та дані.

Angular відповідає за передачу значень даних в HTML-контролі та обробку відповідей користувачів, які впливають на дії та оновлення значень [12].

Angular підтримує двосторонню прив'язку даних, яка представляє собою механізм для встановлення взаємозв'язку між частинами шаблону та частинами компонента. Додатково, ви можете включити розмітку прив'язки в HTML-шаблон, щоб інструктувати Angular, як з'єднати обидві сторони. [12] На рисунку 2.3 показані чотири приклади розмітки прив'язки даних що використовувались у розробці системи.

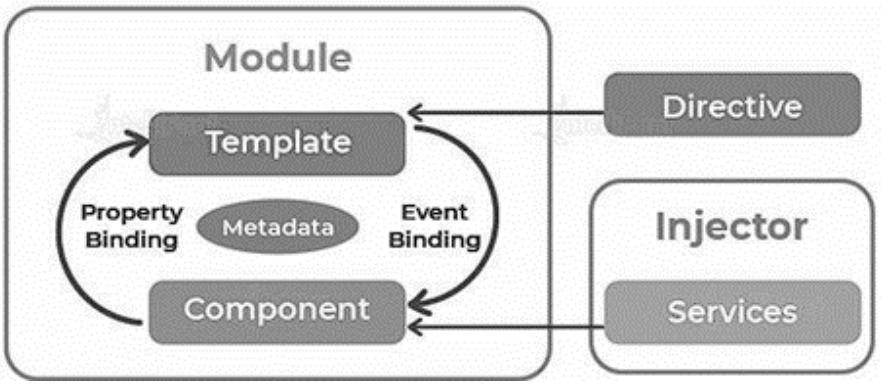

Рисунок 2.3 - Схема форм розмітки прив'язки даних в компоненті системи

Кутові шаблони є динамічними. Коли Angular відображає їх, він перетворює DOM відповідно до інструкцій, які надаються директивами. Директива - це клас із декоратором @Directive(). [12]

Технічно кажучи, компонент також є однією з форм директив. Проте компоненти мають таку важливість і особливості в Angular програмах, що фреймворк визначає окремий декоратор @Component(), який розширює функціональність директиви @Directive() та додає до неї функції, спеціалізовані на роботі з шаблонами.

Окрім компонентів, в Angular існують ще два типи директив: структурні та атрибутні. Angular визначає кілька вбудованих директив обох типів, і ви також можете створювати власні директиви, використовуючи декоратор @Directive().

Як і для компонентів, метадані для директиви визначаються через декоратор та пов'язують декорований клас з елементом, який вибирається за допомогою селектора. У шаблонах, директиви зазвичай відображаються у вигляді атрибутів елемента, або використовуються як імена атрибутів, цілі призначення або прив'язки [12].

Структурні директиви змінюють макет, додаючи, видаляючи або замінюючи елементи в DOM. Наприклад, дві вбудовані структурні директиви, які визначають спосіб візуалізації представлення:

- ngFor: Ця ітеративна структурна директива повідомляє Angular відображати по одному компоненту для кожного елемента в масиві даних.
- ngIf: Ця умовна структурна директива відображає компонент лише тоді, коли задане вираз повертає true. [12]

Директиви атрибутів змінюють зовнішній вигляд або поведінку існуючого елемента в DOM. У шаблонах вони виглядають як звичайні атрибути HTML, тому і названі директивами атрибутів.

Наприклад, ngModel - це директива атрибута, яка реалізує двосторонню прив'язку даних. Вона змінює поведінку існуючого елемента, зазвичай <input>, встановлюючи його властивість відображуваного значення та реагуючи на події зміни.

2.3 Метод поєднання технологій системи користувачів та комунікації відео сервісу

Розробка дизайну інтерфейсу є важливим етапом в процесі створення цифрового рішення, проте це не є початковим етапом. Для створення високоякісного веб-інтерфейсу необхідно спочатку провести прототипування системи. Прототипи можуть включати мокапи, блок-схеми та діаграми, які враховують бізнес-логіку та комунікацію в системі. [13]

Однією з переваг веб-інтерфейсу є те, що не потрібно встановлювати клієнтські додатки на комп'ютери користувачів. Все відбувається в браузері, при

цьому стандарти підтримуються більшістю браузерів. Оновлення веб-додатку здійснюються централізовано, і користувачам не потрібно встановлювати їх на кожному своєму комп'ютері.

З точки зору безпеки даних, вони зберігаються на сервері, регулярно резервуються та захищаються. Використання протоколу HTTPS та формату JSON забезпечує безпеку та зручність передачі даних. У моделі SaaS (Software as a Service), користувачі отримують доступ до програми через мережу, і нелегальне використання програмного забезпечення практично виключено.. [13]

Мобільні програми та веб-додатки, незалежно від їх призначення, зазвичай складаються з кількох компонентів. Це:

Front-end (Клієнтська частина): Візуальна частина програми, яка призначена для взаємодії з користувачами через графічний інтерфейс. Front-end відображає інформацію та дозволяє користувачам взаємодіяти з програмою.

Back-end (Серверна частина): Ця складова програми відповідає за обчислення та логіку, яка зазвичай відбувається на сервері. Вона обробляє запити від клієнтської частини та взаємодіє з базою даних [13].

Оглянуті ключові аспекти розвитку веб-інтерфейсів відображають сучасні тенденції у світі веб-розробки. Нижче наведені основні аспекти, що підтримуються в передових мережних додатках:

База даних використовується для зберігання необхідних програмі даних та організації інформації. Вона є основою для роботи програми, де зберігають та структурують дані. [13]

Принцип мінімалізму та простоти стає основою дизайну, зробивши його більш зрозумілим та доступним для користувача. Зменшення кількості кроків і виборів спрощує досягнення бажаного результату, сприяючи простоті та зрозумілості для користувача. Цей підхід полегшує використання і поліпшує користувацький досвід [13].

Інтерактивність та асинхронність стають ключовими характеристиками веб-інтерфейсів. Динамічні та інтерактивні елементи забезпечують більш ефективну взаємодію з програмою. Замість повного перезавантаження сторінок,

взаємодія з додатком відбувається асинхронно, що сприяє швидкій та зручній роботі з додатком..

Контекстність: Інформація та контролі з'являються на екрані тільки в той момент, коли це дійсно потрібно, спрощуючи взаємодію користувача та зменшуючи завантаження екрана [13].

Синхронізація: Програми здатні генерувати події на сервері та автоматично синхронізувати екран користувача з актуальним станом даних. Це надає можливість користувачам спостерігати за оновленнями в реальному часі.

Повноекранний лейаут: Дизайн сторінок орієнтований на максимальне використання екранного простору з урахуванням роздільної здатності.Спрощений мобільний інтерфейс: З огляду на поширення мобільних пристроїв, розробники активно створюють інтерфейси, призначені для малих дозволів та використання тачскріну [13].

Підтримка стандартів: впровадження нових можливостей та специфікацій включає механізми фелбеку для застарілих технологій, що гарантує сумісність та доступність для різних браузерів та пристроїв. [14]

Мокапи веб-інтерфейсу системи відео-стрімінгового сервісу були створені у веб-редакторі інтерфейсів Figma.

На рисунку 2.4 показана головна сторінка системи. Інтерфейс сторінки перегляду відео та трансляцій представлено на рисунку 2.5, а сторінку користувача можна побачити на рисунку 2.6.

Крім цього, в системі буде розроблений адаптивний дизайн для підтримки різних типів екранів, таких як мобільні пристрої та планшети, які використовують сенсорний жест і скролінг замість миші та клавіатури. Адаптивний дизайн дозволяє оптимально відображати веб-інтерфейс на різних пристроях [13].

Рисунок адаптивного веб-інтерфейсу головної сторінки та сторінки перегляду відео і трансляцій представлено на рисунку 2.7.

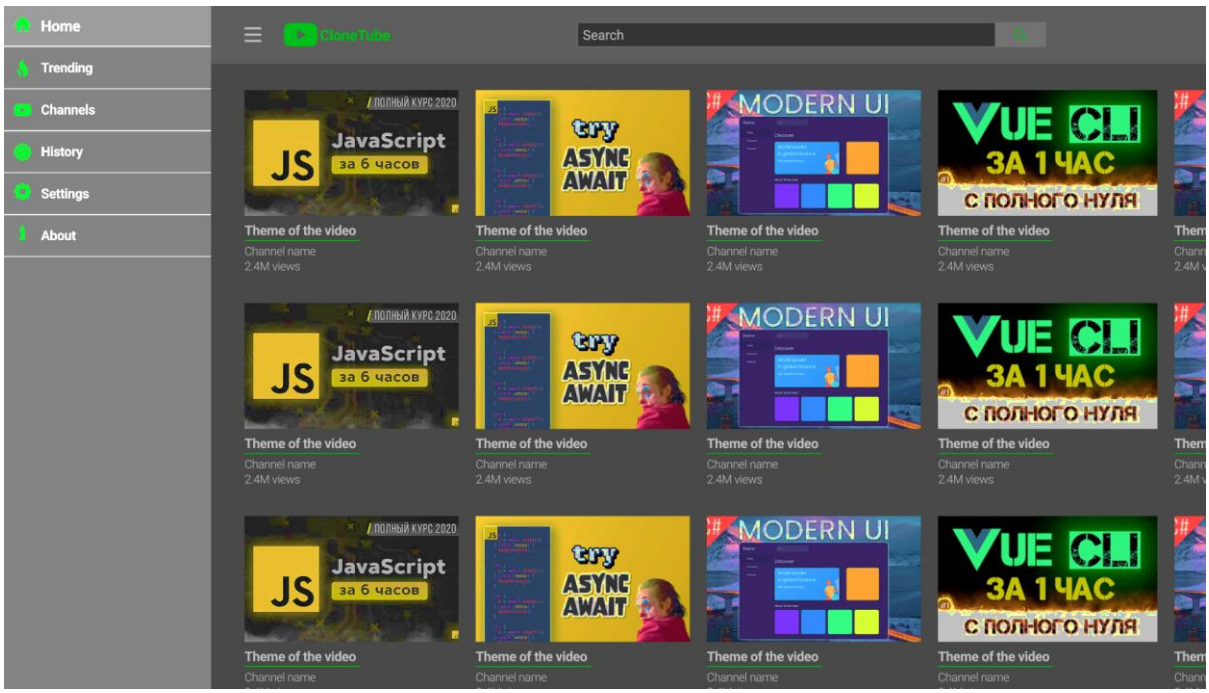

Рисунок 2.4 – Інтерфейс головної сторінки системи розроблений у «Figma»

Рисунок 2.5 – Інтерфейс сторінки перегляду відео та трансляцій системи розроблений у «Figma»

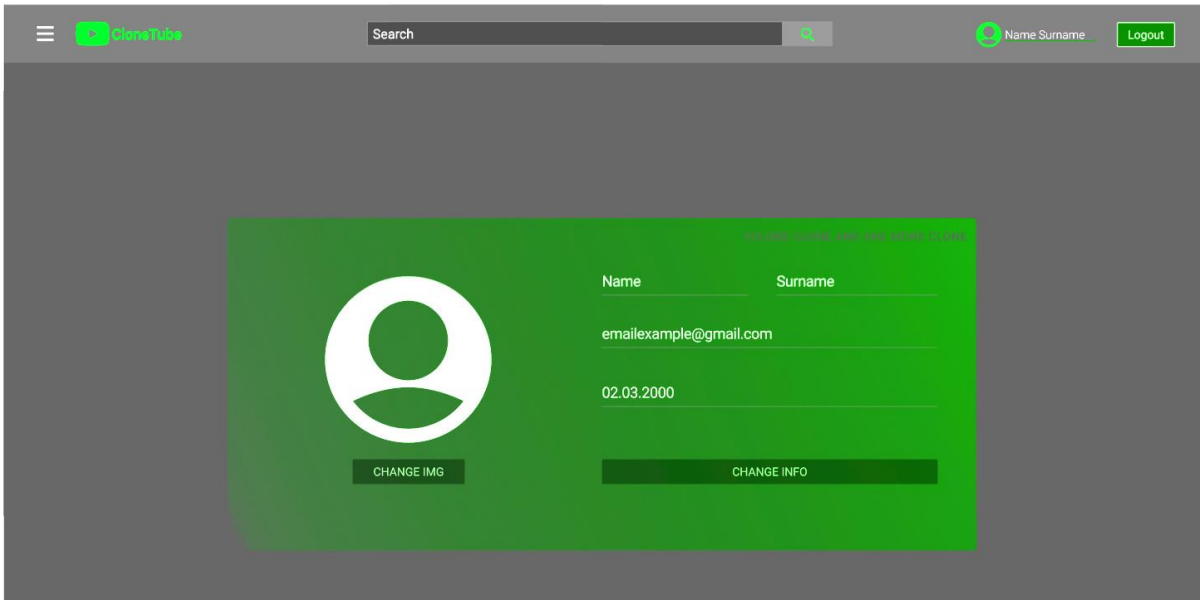

Рисунок 2.6 – Інтерфейс сторінки користувача системи розроблений у «Figma»

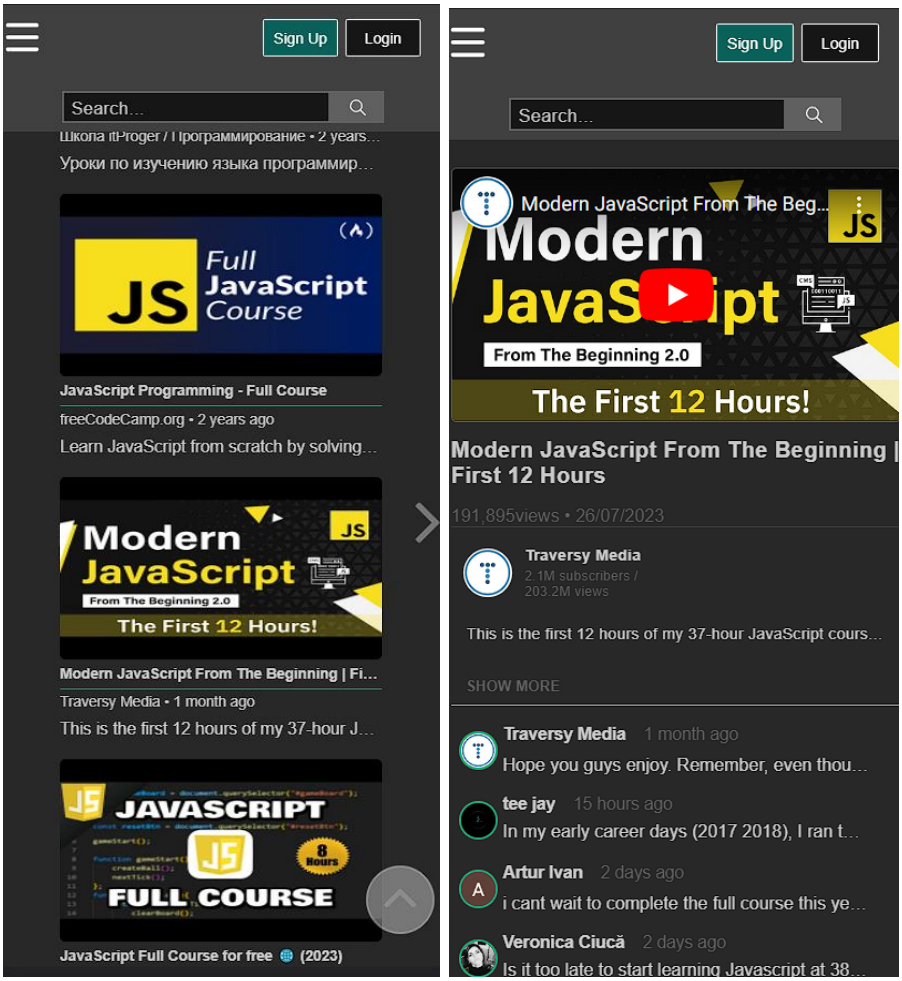

Рисунок 2.7 – Адаптивний інтерфейс головної та сторінки перегляду відео системи розроблений у «Figma»

Кожен елемент, доступний для використання у розробленій системі веб-додатку, може бути налаштований індивідуально. Розробнику не потрібно витрачати ресурси на створення та програмування окремого рішення для веб-інтерфейсу, щоб підключити соціальну мережу, модуль управління профілями користувачів, інструменти перевірки контенту або блок авторизації користувача. При цьому компоненти UX / UI повністю оптимізовані для використання в веб-додатках. [13]

Отже, можна виділити такі ключові аспекти проектування та розробки інтерфейсу системи користувачів: [13]

Аналіз потреб користувача: процес розпочинається з ретельного вивчення вимог і очікувань користувачів. Розуміння їх потреб та функціональних вимог допомагає визначити необхідні компоненти інтерфейсу.

Створення структури і архітектури: увага звертається на структурні аспекти додатку і визначаються, компоненти які потрібно створити та як вони будуть взаємодіяти між собою. Використовується Angular Router з метою організації навігації між різними сторінками [13].

Розробка компонентів: створюються окремі компоненти для різних частин інтерфейсу, такі як заголовок, бічна панель, форми, таблиці, списки та інші.

Шаблони і розмітка: потрібно визначити структуру та макет кожного компонента в HTML-шаблоні. Використовуються директиви Angular для вставки динамічних даних та обробки подій.

Логіка компонентів: розробляється бізнес-логіка та процес обробки подій для кожного компонента у TypeScript-коді [13].

Стилізація інтерфейсу: використовується CSS або передпроцесори стилів (наприклад, Sass або Less) для надання зовнішнього вигляду компонентам. Важливо дотримуватися одного стильового підходу та використовувати найкращі практики в дизайні.

Робота з даними: розробляється процес взаємодії з сервером та зберігання стану додатку в сервісах. Для цього використовуються HTTP-запити для отримання та передачі даних [13].

Тестування і відлагодження: проводиться тестування компонентів для виявлення та виправлення помилок. Для цього використовуються інструменти розробника Angular для відлагодження коду [13].

Оптимізація і продуктивність: час завантаження стрінок зменшується використовуючи підзавантаження та асинхронне завантаження даних. Потрібно дотримуватись рекомендацій з щодо підвищення продуктивності Angular.

Документація: розроляється документація для кожного компонента та функціональності інтерфейсу. Текст розробляється зрозумілим та корисним для інших розробників [13].

Тестування з користувачами: залучення реальних користувачів для тестування інтерфейсу та отримання їхнього відгуку.

Підтримка і вдосконалення: після впровадження системи користувачів потрібно стежити за її роботою, вносити поліпшення та виправляти помилки.

Для відображення поєднання компонентів системи наведенна компонентна діаграма на рисунку 2.8.

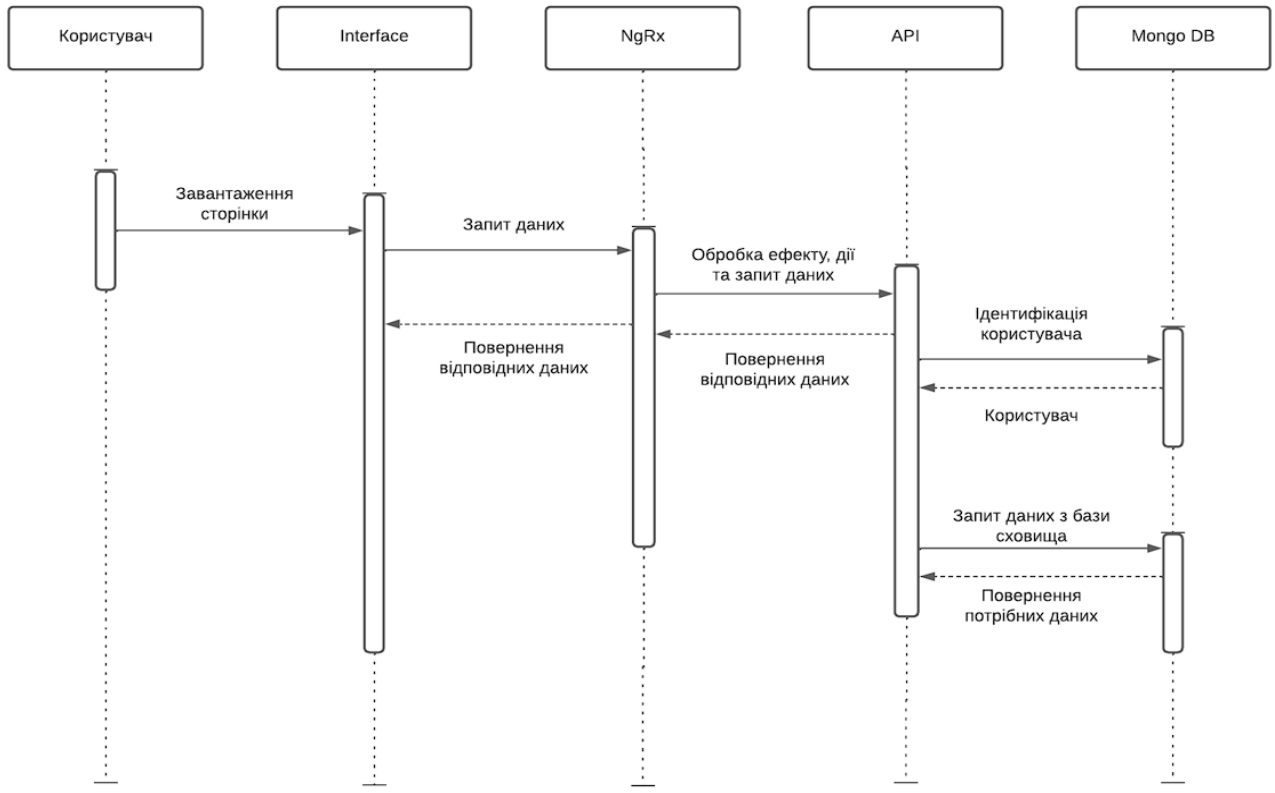

Рисунок 2.8 – Компонента діаграма системи користувачів та їх комінікацій

Проектування веб-інтерфейсу для системи користувачів на Angular - це поетапний процес, який вимагає уважності до деталей, відповідності вимогам користувачів і використання передових практик розробки. Цей процес призводить до створення зручного та функціонального веб-додатку, який відповідає потребам користувачів. [19]

2.4 Висновки

В даному розділі розглядаються основні елементи, які складаються з розробленого веб-інтерфейсу відповідно до актуальних вимог і стандартів, а також пояснюється принципи роботи системи.

Аналіз потреб користувачів становить ключовий етап на початку проектування. Уважне вивчення та розуміння цих потреб сприяє визначенню функціональності та вимог до системи, які утворюють основу для розробки компонентів. Розбиття інтерфейсу на менші компоненти, або декомпозиція, спрощує процес розробки, тестування та підтримки коду, а також підвищує можливість перевикористовування і покращує читабельність коду. Крім того, використання Angular CLI прискорює процес розробки, надаючи інструменти для швидкого створення та налаштування компонентів, що сприяє ефективності розробників [19].

Розміщення логіки у TypeScript-файлах та відокремлення шаблону у HTML допомагає відділити програмну логіку від візуального представлення, сприяючи спрощенню розробки та обслуговування. Щодо стилізації інтерфейсу, застосування CSS для оформлення компонентів сприяє зручності і візуальній привабливості інтерфейсу. У контексті тестування і оптимізації, детальна перевірка компонентів допомагає виявити та усунути помилки, а процес оптимізації сприяє підвищенню продуктивності додатку.

Детальна документація компонентів та відповідність найкращим практикам розробки сприяють забезпеченню якості та легкості обслуговування коду. Ефективне проектування компонентів у середовищі Angular є ключовою

складовою розробки веб-додатків, що сприяє створенню інтерфейсів, які є продуктивними, функціональними і зручними для користувачів. [19]

Для забезпечення надійності системи користувачів використовується формат обміну даними JSON та шифрування інформації криптографічними методами. У проекті використовується фреймворк Angular для створення модульної та масштабованої структури проекту. За допомогою NgRx забезпечуться легкий контроль за станом проекту і також використовується MongoDB для збереження контенту та інформації користувача.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Підготовка середовища для розробки

Для керування версіями файлів програмного коду системи та створення правильної структури та архітектури репозиторія була використана система контролю версій "GitHub". Git - це консольна утиліта, яка використовується для відстеження та ведення історії змін файлів у проекті. Git надає можливість повертати свій проект до попередніх версій, порівнювати, аналізувати та зливати свої зміни до репозиторію [17].

Репозиторій - це термін, який використовується для позначення сховища вашого коду та історії його змін. Git працює локально, і всі ваші репозиторії зберігаються у певних папках на жорсткому диску.

Система контролю версій - це програмне забезпечення, яке допомагає розробникам відстежувати та контролювати стан вихідного коду на протязі усього процесу розробки. Іншими словами, це система, яка фіксує зміни у файлах і, за потреби, дозволяє повернутися до попередніх версій проекту [17, 18].

Розподілена система контролю версій - це система, при якій кожен розробник може мати свою копію репозиторію, що значно зменшує ризик втрати результатів роботи над проектом. Прикладами таких систем є Git, Mercurial, Bazaar.

Git є однією з розподілених систем контролю версій, яка була розроблена Лінусом Торвальдсом для керування розробкою ядра Linux. На сьогоднішній день Git завоював величезну популярність у IT-середовищі і, як наслідок, часто використовується у стеку технологій різних компаній [18].

Для ініціалізації репозиторію системи використовувались наступні команди:

– git init;
– git add;
– git clone Vlad-Zbutkovsky/YouTubeClone_Pet_Project.git;

– git commit -m "first commit";

– git push -u origin master.

Для додавання додаткових пакетів і бібліотек використовувався пакетний менеджер npm. Це можна здійснити, виконавши команду npm install "назва пакету". Структура репозиторія системи повинна мати такий вигляд.:

```
Project/
├── back-end
│   ├── config
│   ├── html-pages
│   ├── logs
│   ├── middleware
│   ├── models
│   ├── node_modules
│   ├── routes
│   ├── services
│   ├── logger.js
│   ├── package.json
│   ├── package-lock.json
│   └── server.js
├── front-end
│   ├── coverage
│   ├── cypress
│   ├── node_modules
│   ├── src
│   ├── angular.json
│   ├── cypress.json
│   ├── karma.conf.js
│   ├── package.json
│   ├── package-lock.json
│   ├── README.md
```

```
│   ├── tsconfig.app.json
│   ├── tsconfig.json
│   └── tsconfig.spec.json
├── package-lock.json
└── README.md
```

Усі файли репозиторія можуть знаходитись у декількох станах, ці стани зображені на рисунку 3.1.

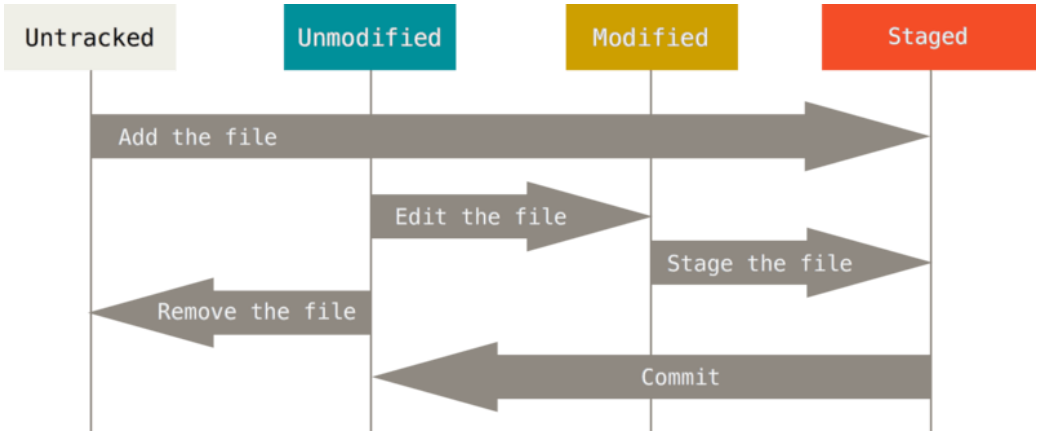

Рисунок 3.1 – Життєвий цикл фалів при використані системи Git

Для редагування коду використовувалась інтегрована середовище розробки Visual Studio Code (VS Code). VS Code — це безкоштовний та дуже популярний редактор коду, розроблений компанією Microsoft. Цей редактор є ідеальним інструментом для веб-розробників на будь-якому рівні володіння. З одного боку, його інтерфейс дуже простий і легкий для освоєння, що корисно для початківців. З іншого боку, VS Code володіє багатьма потужними функціями, які стануть у пригоді професійним розробникам. Крім того, у VS Code є вбудована підтримка системи контролю версій Git, що спрощує роботу зі змінами в коді та прискорює процес розробки. [19]

Інтерфейс VS Code включає в себе велике вікно для редагування текстового вмісту файлу, яке займає основну частину екрана. У лівій частині екрана розташовані вкладки головного меню, де доступні основні функції

редактора. Вкладка "Провідник" (або "Explorer") відкривається за замовчуванням при запуску програми і відображає список відкритих файлів і структуру каталогу відкритої папки. [21]

Значок "Source Control" на панелі активності ліворуч завжди вказуватиме на наявність змін у репозиторії. Вибір цієї піктограми дозволить переглянути деталі поточних змін у репозиторії, такі як "CHANGES", "STAGED CHANGES" та "MERGE CHANGES".

При кліку на кожен з цих елементів, ви зможете детально переглянути текстові зміни у кожному файлі. Навіть для непоетапних змін редактор праворуч все ще дозволятиме редагувати файли [21].

Також, індикатори стану репозиторія розташовані в нижньому лівому куті VS Code. Вони включають в себе інформацію про поточну гілку, статус змінених файлів ("брудні індикатори") та кількість відправлених і отриманих комітів у поточній гілці. Це дозволяє перевірити статус будь-якої гілки у репозиторії, натиснувши на цей індикатор стану і вибравши опцію "Git" зі списку [21].

VS Code виділяється серед інших редакторів наявністю вбудованого інструмента відладки коду. Після невеликих налаштувань ви зможете виявляти помилки в коді прямо з редактора. Наприклад, ви можете встановити точку зупинки та стежити за виконанням конкретного фрагмента коду. Крім цього, редактор має вбудовану консоль, в якій ви можете переглядати результати виконання коду або повідомлення про помилки. Відладчик може бути налаштований для різних мов програмування та завдань. [19]

У розділі "File" розташовані команди, які дозволяють працювати з файлами у вашій робочій папці. Ви можете зберігати та відкривати файли, обирати папки та виконувати інші дії через цей розділ. У розділі "Edit" ви можете працювати з відкритим файлом, скасовувати останні дії та виконувати пошук по файлу. У розділі "Selection" ви знайдете команди для виділення потрібного фрагмента коду [19].

Для доступу до різних додатків, які вбудовані у редактор коду, є можливість скористатися розділом "View". Тут доступні повторювані пункти

іншого активного меню, а також можливість відкрити термінал і налаштувати елементи інтерфейсу.

Розділ "Go" дозволяє переміщатися між відкритими файлами та шукати в документі. Також ви можете поділити робочу область на сегменти, що корисно при роботі на великому моніторі для збереження розмітки та файлів стилів в одному місці. У редакторі також є вбудований термінал, який може виконувати окремі завдання і відображати командний рядок у інтерфейсі редактора. [19]

VS Code використовується розробниками у компаніях різних розмірів і є дуже зручним для навчання при розробці власних проектів і систем. У ньому доступні всі необхідні функції для написання та тестування проектів.

3.2 Розробка API системи користувачів та комунікації відео сервісу

Для забезпечення повноцінної роботи системи, було впроваджено декілька API. Ці API відповідають за взаємодію системи з даними, пов'язаними з відео та користувачами. Реалізація цих API виконана за допомогою мови програмування JavaScript та платформи Node.js. Для спілкування з клієнтом, система надсилає додаткові параметри до реалізованих API та очікує на відповідь. Кожен запит супроводжується ключем, який ідентифікує конкретного користувача, що відправляє запит до API. Це забезпечує безпеку системи, запобігаючи доступу до даних інших користувачів, яким ці дані не належать, і захищає від можливості вторгнення та крадіжки даних користувача.[20] Нижче наведений список реалізованих API та їх функціонал.:

1. app.use("/api/users", cors(corsOptions), require("./routes/users")) – отримати список користувачів системи;
2. app.use("/api/auth", cors(corsOptions), require("./routes/auth")) – аутентифікація та авторизація користувача;
3. app.use("/api/user-change-info", cors(corsOptions), require("./routes/change-info")) – зміна даних користувача;

4. app.use("/api/subscribe", cors(corsOptions), require("./routes/subscribe")) – підписатись на відео канал;
5. app.use("/api/unsubscribe", cors(corsOptions), require("./routes/ unsubscribe")) – відписатись від відео каналу;
6. app.use("/api/like", cors(corsOptions), require("./routes/like-video")) – додати відео до списку вподобаних відеозаписів;
7. app.use("/api/un-like", cors(corsOptions), require("./routes/un-like-video")) – видалити відео з списку вподобаних відеозаписів;
8. app.use("/api/dislike", cors(corsOptions), require("./routes/dislike-video")) – поставити відео дизлайк;
9. app.use("/api/un-dislike", cors(corsOptions), require("./routes/un-dislike-video")) – видалити дизлайк з відео;
10. app.use("/api/history", cors(corsOptions), require("./routes/history")) – отримати історію переглядів користувача;
11. app.use("/api/settings", cors(corsOptions), require("./routes/update-settings")) – отримати налаштування системи користувача;
12. app.use("/api/img-upload", cors(), require("./routes/img-upload")) – завантажити фото профілю користувача;
13. app.use("/api/videos", cors(corsOptions), require("./routes/videos")) – отримати список відео;
14. app.use("/confirmation", cors(), require("./routes/email-confirmation")) – підтвердити імейл користувача;
15. app.use("/resend", cors(), require("./routes/email-token-resend")) – змінити імейл користувача;
16. app.use("/password-reset", cors(), require("./routes/ password-reset-token")) – змінити пароль користувача;
17. app.use("/password-reset-confirm", cors(), require("./routes/pass-change-form")) – відправити лист підтвердження імейлу користувача;
18. app.use("/password-change", cors(), require("./routes/change-user-password")) – змінити пароль доступу користувача;

Сервіси, які мають доступ до наведених вище API, занесені до списку дозволених користувачів. Це заходить на користь збільшенню безпеки даних та унеможливлює запити з інших джерел до поточних API. Наразі список дозволених адрес виглядає так::

```
const whitelist = [
  "https://my-clone-tube.herokuapp.com",
  "https://clone-tube-backend.herokuapp.com",
];
```

Для передачі даних використовується протокол HTTPS, що є розширенням протоколу HTTP і підтримує шифрування за допомогою криптографічних протоколів SSL і TLS. HTTPS не є самостійним протоколом передачі даних, а представляє собою розширений варіант протоколу HTTP з додатковим шифруванням [20]. У випадку передачі даних через HTTP, інформація не захищена, тоді як HTTPS забезпечує конфіденційність даних завдяки шифруванню. За допомогою шифрування таких даних ускладнюється їх перехоплення та незаконне використання третіми особами. [20]

Для забезпечення передачі даних за допомогою HTTPS на веб-сервері, який обробляє запити від клієнтів, необхідно встановити спеціальний SSL-сертифікат. Існують різні типи сертифікатів. Деякі сертифікати призначені для захисту лише одного конкретного доменного імені, в той час як інші, такі як Wildcard SSL, забезпечують захист для доменного імені та всіх його субдоменів. TrueBusinessID with EV сертифікат є прикладом, який забезпечує вищий рівень безпеки для вашого домену та додає зелений рядок у вікні браузера. Цей процес шифрування застосовується як до даних, які клієнт отримує, так і до даних, які він надсилає на сервер. [22]

Маючи SSL-сертифікат, можна покращити рейтинг у пошуковій системі Google, оскільки вона враховує цей фактор при ранжируванні сторінок. На сьогоднішній день вплив цього фактору не є вирішальним, проте в майбутньому його вагомість у пошукових результатах може зростати. [21]

Компоненти інтерфейсу взаємодіють з API, передаючи необхідні параметри та ідентифікуючи себе в системі. Якщо сервер підтверджує існування користувача, то надається необхідна інформація, і запит вважається успішним [22]. Нижче наведено приклад коду, який відображає процес створення запиту для отримання інформації про користувача:

```
getUserInfoById(id: string, token: string): Observable<UserInfo> {
  const headers = new HttpHeaders({
    'Content-Type': 'application/json',
    'x-auth-token': token,
  });
  return this.http.get<UserInfo>(`${GET_USER_INFO_BY_ID}/${id}`, {
    headers: headers,
  });
}
```

Для розробки API було використано різноманітні засоби що підвищують їх надійність та покращують роботу із ними.

3.3 Розробка веб-інтерфейсу системи користувачів та комунікації відео сервісу

Веб-інтерфейс системи був розроблений власноруч, без використання зовнішніх бібліотек та інструментів, з використанням гіпертекстової розмітки та каскадних стилів. Цей підхід до створення інтерфейсу надає найвищу масштабованість і можливість повторного використання компонентів системи [23].

Для забезпечення взаємодії між компонентами інтерфейсу та передачі даних між базою даних та шаблонами веб-сайту, використовувалися додаткові інструменти, які надає фреймворк Angular. Зокрема, застосовувалися такі методи:

- Двостороння та одностороння прив'язка даних для зв'язку компонентів та обміну даними між ними.
- Використання структурної директиви *ngFor для відображення великої кількості однакових елементів.
- Використання *ngIf для умовного відображення компонентів на основі певних умов.
- Використання ng-template для динамічної заміни компонентів.
- Використання ng-routerOutlet для динамічної заміни компонентів відповідно до поточного роуту системи.

Ці інструменти та підходи допомагали створювати інтерактивний та динамічний веб-інтерфейс системи. [16]

Структура компонента складається із кількох файлів. Перший файл представляє собою скрипт компонента, в якому реалізовані всі необхідні операції з даними, що повинен виконувати даний компонент, і він також підключає інші файли компонента [16]. Цей файл має характерний формат .ts, оскільки TypeScript є мовою для розробки скриптів у фреймворку Angular. Переваги та недоліки використання TypeScript порівняно з JavaScript наведені в таблиці 3.1.

Таблиця 3.1 - Переваги та недоліки TypeScript

Доступні можливості	*TypeScript*	*JavaScript*
Використовуються такі поняття, як типи та інтерфейси для опису даних.	+	–
Підтримка типів даних та кастомної типізації	+	–
Код потрібно компілювати перед тим як його прочитає браузер	+	–
Полегшує використання сторонніх бібліотек,	+	–
Тяжкий у освоєні	+	–

Продовженя таблиці 3.1

Маленька спільнота розробників та користувачів	+	–
Безпека коду гарантується типізацією	+	–

Приклад коду скрипта для компонента селекту наведений нижче:

```
@Component({
  selector: 'app-select',
  templateUrl: './select.component.html',
  styleUrls: ['./select.component.scss'],
  providers: [COUNTRY_CONTROL_VALUE_ACCESSOR],
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class SelectComponent implements ControlValueAccessor {
  @Input() options: Array<Option>;
  @Input() innerValue: Option;
  selectedOption: string;
  showList: boolean;
  disabled = false;
  private onTouched: () => void;
  private onChanged: (id: string) => void;
  constructor(private eRef: ElementRef, private ref: ChangeDetectorRef) {}
  @Input()
  get value(): Option {
    return this.innerValue;
  }
  set value(v: Option) {
    if (v !== this.innerValue) {
```

```
    this.innerValue = v;
   }
  }
  @HostListener('document:click', ['$event'])
  clickOut(event: Event): void {
   if (!this.eRef.nativeElement.contains(event.target) && this.showList) {
    this.showList = false; }
 }
```

Другий файл представляє собою шаблон компонента, який містить структуру та розмітку елементів інтерфейсу цього компонента. Він зазвичай має формат файлу .html, який використовує гіпертекстову розмітку.[24] Приведено нижче приклад коду, який відображає структуру та розмітку елементів інтерфейсу для компонента селекту:

```
<div class="select noselect">
 <div>
  <button (click)="showListOfOptions($event)" [disabled]="disabled">
   {{ innerValue?.name }}
  </button>
  <i class="fa fa-angle-down" aria-hidden="true"></i>
 </div>
 <div *ngIf="showList" class="list">
  <p (click)="selectOption(option.id)" *ngFor="let option of options">
   {{ option.name }}
  </p>
 </div>
</div>
```

Третім обов'язковим файлом є файл каскадних стилів, який містить інформацію про кольори елементів гіпертекстової розмітки та всі їхні стилі, відповідно до єдиного стилю інтерфейсу системи. Цей файл, як правило, має формат .scss, оскільки для розробки стилів інтерфейсу системи був обраний

препроцесор SCSS. Даний препроцесор стилів є стандартним при розробці на фреймворку Angular і має низку переваг порівняно зі звичайним CSS.[22] Нижче наведено приклад коду, який визначає стилі для компонента селекту:

```
@import "/src/styles";
 .list::-webkit-scrollbar {
  width: 6px;
  border-radius: 5px;
  background-color: transparent;
 }
 .list::-webkit-scrollbar-thumb {
  border-radius: 50px;
  background-color: var(--text_basic_color);
 }
 p {
  display: flex;
  align-items: center;
  padding: 0.3em 0;
  border-bottom: 1px dashed var(--text_basic_color);
  font-size: 14px;
  &:hover {
   transform: scale(1.05);
   background-color: var(--main_color);
   box-shadow: 0 0 11px var(--main_color);
   color: var(--text_light_color);
  }
 i:hover {
   transform: scale(1.05);
  }
 }
}
```

Третім файлом, який є необов'язковим, є файл з тестами для даного компонента системи. Формат файлу з тестами - .spec.ts, і він визначає його призначення, зокрема, у випадку даної системи, це тести на рівні юніт-тестів та інтеграційних тестів [22].

Інтерфейс системи підтримує всі можливості, які були визначені на етапі початкового проектування системи, і має наступний вигляд. На рисунку 3.2 показаний компонент реєстрації та входу для користувачів системи.

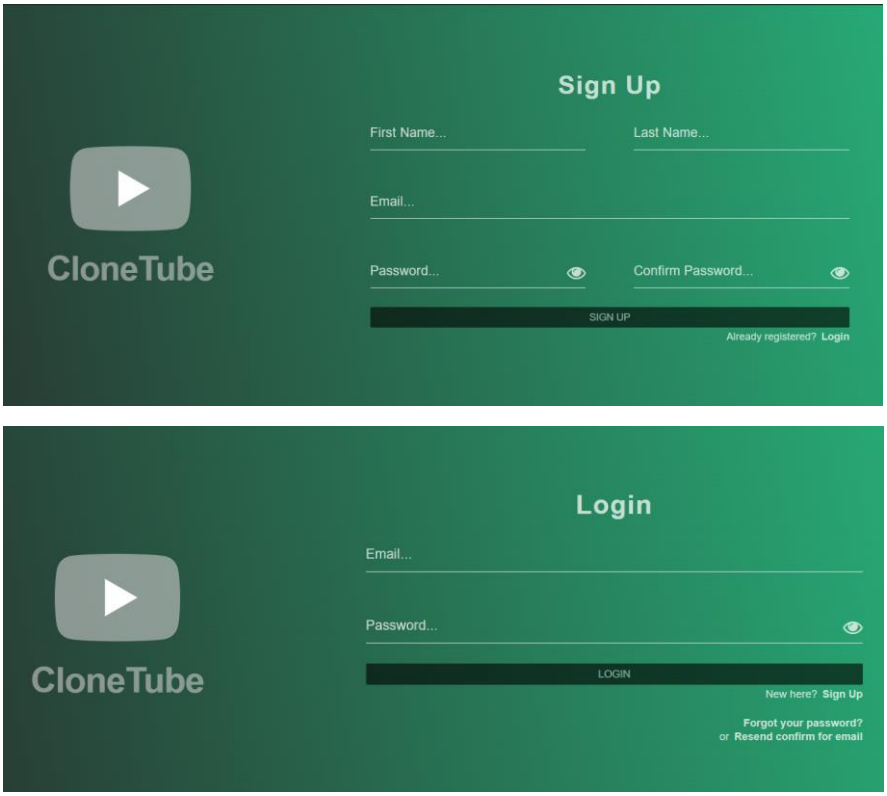

Рисунок 3.2 - Компоненти реєстрації та логіна користувача

На рисунку 3.3 зображено компонент профіля з даними користувача системи.

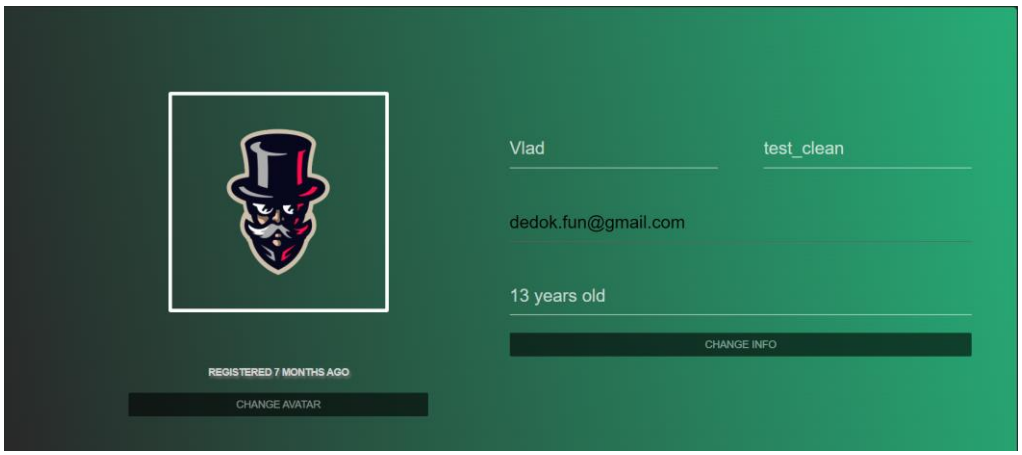

Рисунок 3.3 - Компоненти профіля з даними користувача

На рисунку 3.4 зображено компонент з посиланнями на сторінки системи для авторизованого та неавторизованого користувача.

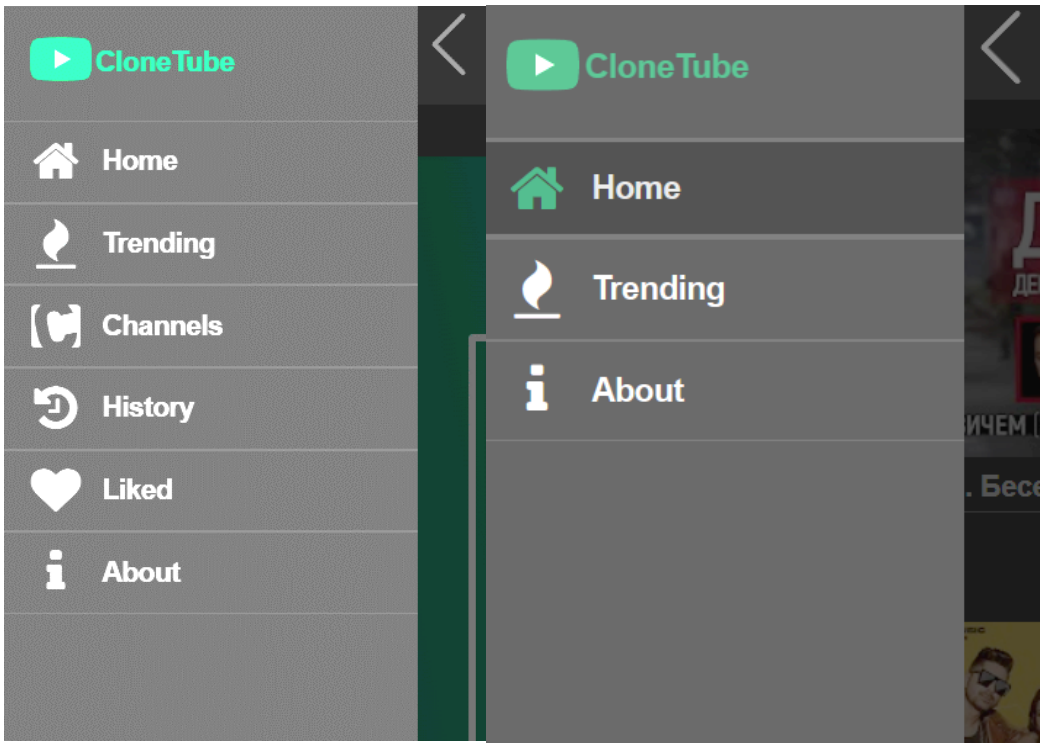

Рисунок 3.4 - Компонент з посиланнями на сторінки системи

На рисунку 3.5 зображено інтерфейс сторінки із списком каналів користувача.

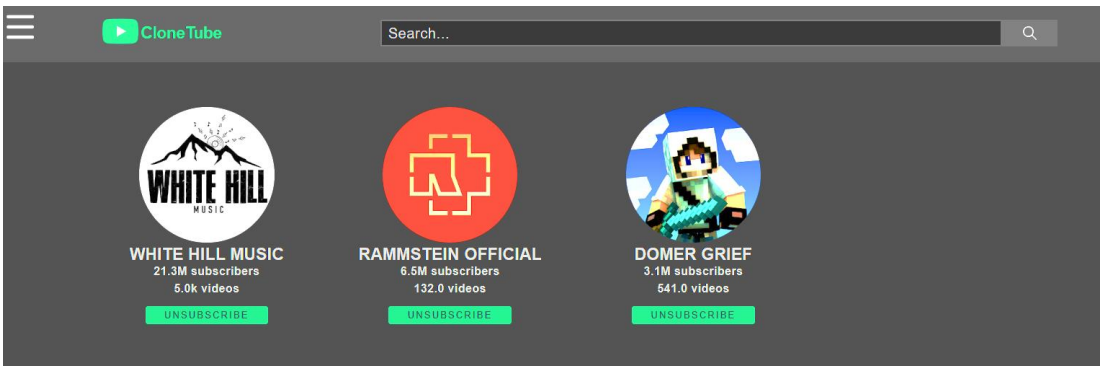

Рисунок 3.5 - Інтерфейс сторінки із списком каналів користувача

На рисунку 3.6 зображено інтерфейс сторінки для перегляду відео контенту.

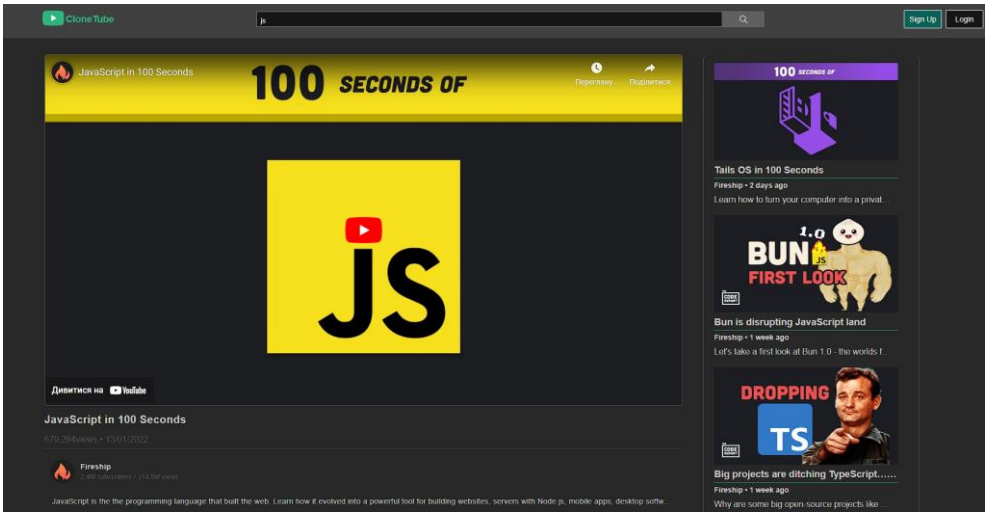

Рисунок 3.6 - Інтерфейс сторінки для перегляду відео контенту

На рисунку 3.7 зображено інтерфейс сторінки з налаштуваннями системи користувача.

Рисунок 3.7 - Інтерфейс сторінки з налаштуваннями системи користувача

На рисунку 3.8 зображено інтерфейс сторінки з інформацією про розроблену систему.

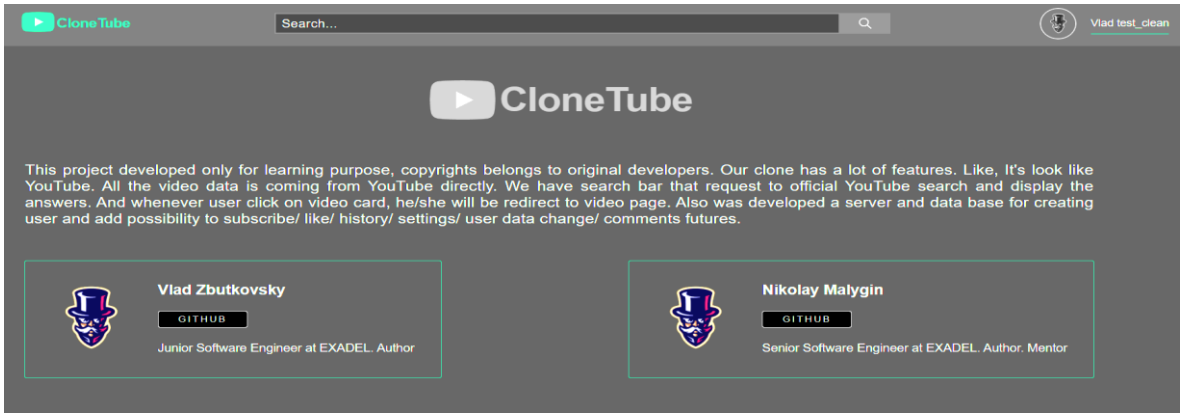

Рисунок 3.8 - Інтерфейс сторінки з налаштуваннями системи користувача

3.4 Розробка мікро сервісної архітектури для поєднання технологій системи користувачів та комунікації відео сервісу

Мікросервісна архітектура - це підхід до розробки програмного забезпечення, в рамках якого система розбивається на дрібні і автономні компоненти, відомі як мікросервіси, кожен з яких відповідає за конкретну функціональну частину. Кожен мікросервіс відповідає за конкретну функціональність та може бути розгорнутий, масштабований та оновлюваний окремо від інших [23].

Мікросервісна архітектура сприяє гнучкості та розширюваності системи, оскільки окремі частини можуть бути розроблятися, тестуватися та вдосконалюватися незалежно. Це також полегшує впровадження нових технологій та зменшує вплив змін в одній частині системи на інші [23].

За допомогою мікросервісної архітектури, розробники можуть створювати горизонтально масштабовані системи, де кожен мікросервіс може бути розгорнутий на окремих серверах або контейнерах. Однак ця модель також вносить виклики, пов'язані з управлінням консистентністю даних, моніторингом та управлінням версіями, які вимагають ретельного планування та керування.

Для реалізації системи користувачів на Angular з використанням мікросервісної архітектури, розгляньте наступні компоненти:

User Interface (UI): Angular відповідає за створення та відображення інтерфейсу для користувачів, включаючи розробку компонентів, шаблонів та налаштування роутингу [23].

Authentication Service: Це мікросервіс, який забезпечує обробку аутентифікації та авторизації користувачів.

User Service: Він відповідає за управління користувачами, зокрема за створення, редагування та видалення облікових записів користувачів.

Profile Service: Цей сервіс може використовуватися для керування профілями користувачів, включаючи можливість зміни паролю, завантаження фотографії профілю та інші операції.

Notification Service: Він використовується для відправки сповіщень або повідомлень користувачам [23].

Кожен мікросервіс може використовувати власну базу даних для збереження інформації, що стосується його функціональності.

Необхідно мати централізований шлюз, який керує обробкою запитів до різних мікросервісів і надає доступ до них через HTTP або інші протоколи. Він може виконувати функції, такі як маршрутизація, розподілення навантаження і забезпечення безпеки [23].

Для забезпечення взаємодії між мікросервісами можна використовувати різні протоколи, такі як HTTP / REST або GRPC. Крім того, розглядається можливість використання механізмів для асинхронного обміну даними, таких як RabbitMQ або Apache Kafka. [23]

Для моніторингу та забезпечення надійності системи рекомендується використовувати інструменти моніторингу, такі як Prometheus, і інструменти для логування, наприклад, ELK Stack (Elasticsearch, Logstash, Kibana).

Для спрощення процесу розгортання та управління мікросервісами можна використовувати контейнеризацію (наприклад, Docker) та оркестратори, такі як Kubernetes. [23]

Крім того, можна впроваджувати системи кешування, такі як Redis, для покращення продуктивності системи, а також системи управління сесіями для збереження стану сесій між запитами користувачів.

Забезпечення безпеки мікросервісів, включаючи аутентифікацію, авторизацію та захист від атак, становить критичний аспект архітектури.

Ця архітектура дозволяє розробляти, масштабувати і підтримувати систему користувачів на Angular у більш гнучкий та ефективний спосіб, розділяючи її на незалежні компоненти, які можуть бути розроблені та підтримувані окремо. [23]

Під час розробки системи відео-стрімінгового сервісу було створено 4 модулі, використовуючи фреймворк Angular, компоненти Node.js, Express.js та базу даних Mongo DB:

– NgRx стейт для зберігання даних
– Веб інтерфейс системи
– Компонент для обробки запитів на отримання даних
– Модуль для хостингу системи
– База даних Mongo DB

Модулем для хостингу системи був обраний Heroku.

Heroku - це хмарний сервіс, який дозволяє розгортати ваші застосунки. У порівнянні з Microsoft Azure або Amazon Web Services, Heroku має простий інтерфейс, що дозволяє зосередитися на розробці вашого програмного забезпечення. Ще однією важливою перевагою є можливість безкоштовного розгортання застосунків на Heroku. [23]

Heroku надає 550 безкоштовних годин роботи вашого застосунку, і ще 450 годин, якщо ви додаєте кредитну картку на місяць. Щоб уникнути закінчення безкоштовного періоду, Heroku переводить неактивні застосунки в режим сну.

Хмарні платформи володіють еластичністю і функцією обліку споживання ресурсів. [23]

Якщо на ваш сервіс приходить багато користувачів, платформа автоматично масштабується вгору або вниз. Облік споживання означає, що ви платите лише за ті ресурси, які ви використовуєте. [23]

NgRx стейт та веб-інтерфейс системи розміщені на одному сервері за адресою "https://my-clone-tube.herokuapp.com".

Компонент для обробки запитів клієнта розташований на іншому сервері за адресою "https://my-clone-tube.herokuapp.com", на який клієнт відправляє запити для отримання та збереження даних.

Кожен модуль системи можна представити як окремий блок, як показано на рисунку 3.9:

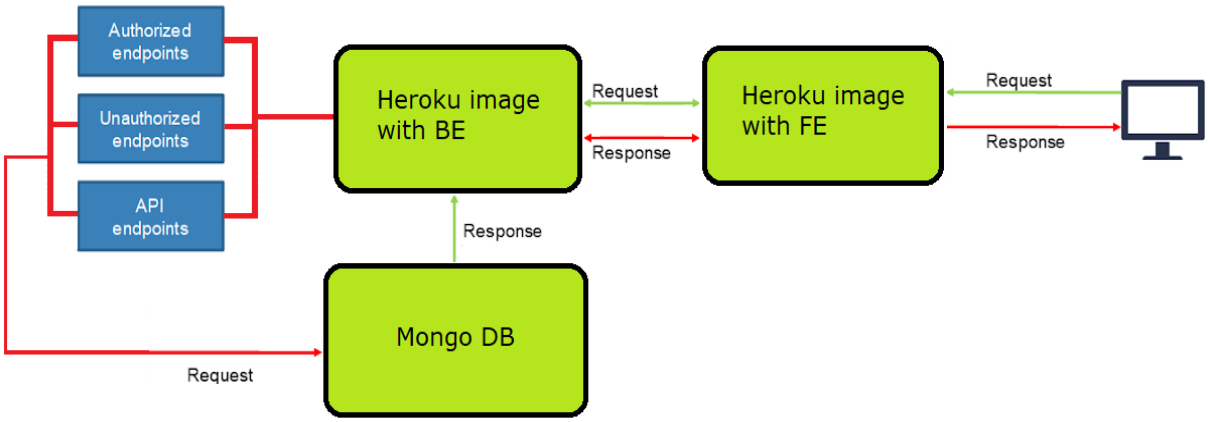

Рисунок 3.8 - Діаграма модулів системи

3.5 Висновки

У цьому розділі було створено робоче середовище для розробки системи, розроблено компоненти для обробки API, створено адаптивний веб-інтерфейс, який відповідає актуальним вимогам і стандартам. Також була розроблена мікросервісна архітектура системи, і наведено діаграму цієї архітектури.

РОЗДІЛ 4
ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Опис методик, що використовуються для тестування системи

Тестування - це не просто пошук помилок, а в першу чергу перевірка результатів роботи програми. Помилка в цьому контексті - це розбіжність між очікуваним і фактичним результатом. Для проведення тестування створеної системи використовувалися три види тестів: Unit-тестування, Інтеграційне тестування і тестування "від початку до кінця" (E2E).

Unit-тестування - це ізольоване тестування окремих методів класу. Воно полягає в перевірці роботи методу, подаючи на його вхід різні комбінації параметрів і порівнюючи отриманий результат з очікуваним результатом. [24]

Основна ідея полягає в тому, щоб перевірити, чи працює конкретна одиниця коду (одиниця може бути функцією, методом чи класом) як очікується.

У процесі unit-тестування розробники створюють невеликі тести, які перевіряють конкретні аспекти функціональності коду. Це допомагає виявити помилки та дефекти на ранніх етапах розробки і забезпечити стабільність окремих частин програми. Unit-тести є автоматизованими, що дозволяє їх запускати повторно при кожній зміні коду для перевірки стабільності і правильності роботи.

Цей вид тестування важливий для підтримки кодової бази, забезпечення рефакторингу та зменшення можливості виникнення непередбачених проблем у великих програмних проектах. Unit-тестування є ключовою частиною практики Test-Driven Development (TDD), коли тести створюються перед реалізацією функціональності, щоб визначити очікувану поведінку.

Інтеграційне тестування - це перевірка взаємодії кількох компонентів програми. Під час цього виду тестування перевіряється не лише робота окремих методів класу, але і їхнє взаємодія з іншими компонентами, включаючи їхнє відображення в шаблонах [24]. Основна мета полягає в тому, щоб переконатися,

що різні компоненти програми працюють разом як цілісна система і взаємодіють правильно.

У процесі інтеграційного тестування тести фокусуються на перевірці точок з'єднання між компонентами, а також на взаємодії між системними елементами. Це може включати тестування API, баз даних, інтерфейсів користувача та інших аспектів системи.

Інтеграційне тестування дозволяє виявляти проблеми, які можуть виникнути при взаємодії різних частин програми, такі як невірні дані, проблеми синхронізації чи некоректні взаємодії. Важливо переконатися, що програма може ефективно інтегруватися та співпрацювати з усіма своїми складовими для забезпечення надійної та стабільної роботи системи в цілому [24].

Тестування "від початку до кінця" (E2E) - це спосіб тестування програми в цілому, який призначений для виявлення проблем, які можуть уникнути у процесі інших видів тестування. У цьому виді тестування створюються тестові сценарії, які відтворюють дії користувача на сторінці, перевіряючи, чи працюють всі компоненти та сервіси програми разом правильно. [24]

У процесі E2E тестування використовуються автоматизовані тести, які моделюють та перевіряють взаємодію користувача з програмою від початку до кінця, включаючи всі можливі сценарії взаємодії та можливі варіанти введення.

Основна мета E2E тестування - це забезпечити, що всі частини системи працюють разом правильно, і визначити можливі проблеми на рівні інтеграції та взаємодії між компонентами. Цей вид тестування важливий для виявлення дефектів, які можуть виникнути в реальних умовах використання програми користувачем [24].

E2E тестування допомагає забезпечити, що весь додаток працює так, як очікується, та що він здатний ефективно взаємодіяти з іншими системами та службами. Цей підхід сприяє забезпеченню якості продукту і підвищенню довіри користувачів до програмного забезпечення.

Утиліти тестування в Angular представляють собою набір класів і функцій, які необхідні для створення тестового середовища для Angular. Одним з

найважливіших класів є TestBed, який налаштовує модуль Angular аналогічно тому, як це робиться за допомогою декоратора @NgModule (за винятком того, що модуль налаштовується для тестування). У класі TestBed доступна функція configureTestingModule, в якій вказуються всі необхідні залежності для роботи компонента в тестовому середовищі. [25]

Фреймворк Jasmine використовується для написання тестів в Angular і базується на behavior-driven нотації. Кожен тест в Jasmine складається як мінімум з двох елементів: функції describe (яка представляє собою сьют тестів) і функції it (яка представляє собою окремий тестовий випадок). Зазвичай функція describe використовується для опису функції, яка підлягає тестуванню, наприклад, "createCustomer()". У межах сюїта створюються різні тести за допомогою функції it. Кожен окремий тест перевіряє цільову функцію на очікувану поведінку в різних умовах. [26]

Jasmine дозволяє розробникам створювати невеликі, легкі та зрозумілі тести, які можуть бути легко виконані під час розробки або під час автоматизованого тестування. Основні елементи Jasmine включають в себе функції describe та it, які дозволяють створювати блоки тестів та окремі тести відповідно [26].

Також, Jasmine надає можливість використовувати різні функції для перевірки (assertions), такі як expect, що робить код тестів більш зрозумілим та ефективним. Фреймворк також підтримує використання спеціальних функцій для асинхронного коду, що робить його досить гнучким та пристосованим для різних сценаріїв тестування веб-додатків [26].

Тестування системи користувачів на Angular включає в себе створення тестових сценаріїв для перевірки роботи компонентів, сервісів і інших частин програми. Angular надає засоби для спрощення цього процесу. Ось основні кроки підходу до тестування системи користувачів на Angular:

Налаштування тестового середовища: Впевніться, що встановлені необхідні залежності для тестування, включаючи фреймворк Jasmine (для написання тестів) і інструмент Karma (для виконання тестів).

Створення тестових специфікацій: Створіть тестові специфікації (файли .spec.ts) для компонентів, сервісів і інших частин системи користувачів. Наприклад, для тестування компонента user.component.ts створіть файл user.component.spec.ts. У цих специфікаціях описуйте різні тести для перевірки функціональності. [27]

Імітація дій та подій: Під час тестування може знадобитися імітувати HTTP запити або інші дії, щоб не залежати від реального сервера або даних. Angular надає інструменти для створення фейкових сервісів для імітації цих дій. Потрібно перевірити, як компоненти реагують на події, такі як клік миші або введення даних. Для цього використовується TestBed та функція triggerEventHandler. [28]

Необхідно позбутися ізольованості компонентів та провести тестування їх взаємодії. Для цього використовують функції TestBed.createComponent і fixture.detectChanges(). Крім того, важливо перевірити, як система обробляє помилки, включаючи обробку помилок, які можуть виникнути на сервері.

4.2 Проведення тестування веб додатку

Проведення тестування веб-додатку є ключовим етапом в розробці програмного забезпечення. Цей процес включає в себе ряд дій та стратегій для перевірки різних аспектів додатку. Функціональне тестування спрямоване на перевірку основних функцій та взаємодії з користувачем.

Одночасно нефункціональне тестування охоплює аспекти продуктивності, безпеки та сумісності. Важливо також тестувати на різних браузерах та пристроях для забезпечення широкої сумісності [29].

Тестування відмовостійкості дозволяє переконатися, що додаток може відновлюватися після збоїв. Тестування користувацького інтерфейсу оцінює зручність та інтуїтивність взаємодії з додатком [29].

Важливо враховувати аспекти безпеки, які включають в себе виявлення та усунення потенційних вразливостей та заходи для захисту від можливих атак.

Тестування на різних рівнях навантаження допомагає оцінити, як добре додаток справляється з великою кількістю користувачів або запитів [29].

У процесі тестування важливо також взаємодіяти з розробниками для виправлення виявлених дефектів та оптимізації коду. Крім того, автоматизовані засоби тестування дозволяють ефективно виконувати повторювані тести та заощаджувати час [29].

Прозорість і документація грають важливу роль, допомагаючи всім учасникам проекту розуміти результати тестування, виявлені проблеми та кроки для їх вирішення. Цей процес допомагає забезпечити якість веб-додатку та зменшити ризики пов'язані з його впровадженням.

Важливим елементом є також тестування баз даних для перевірки цілісності та правильного зберігання даних. Застосування автоматизованих тестів допомагає в ефективному виконанні тестових сценаріїв та швидкому виявленні помилок. Всі ці аспекти спільно сприяють забезпеченню якості веб-додатку та задоволенню потреб користувачів [29].

Для автоматичного тестування під час розгортання системи можна використовувати системи Continuous Integration/Continuous Deployment (CI/CD). Його основна мета полягає в автоматизації процесу тестування для ефективного виявлення помилок, забезпечення стабільності та збільшення продуктивності розробників.

Автоматичне тестування дозволяє створювати тести, які можуть бути легко повторюваними, виконувати швидше і виявляти проблеми на ранніх етапах розробки. Це включає в себе тести для функціональності, інтеграції, продуктивності та інших аспектів програмного забезпечення [29].

Інструменти для автоматичного тестування дозволяють автоматизовано створювати, виконувати та аналізувати результати тестів. Це допомагає забезпечити швидку зворотню зв'язок та сприяє швидкому виявленню та виправленню помилок у коді програми. Автоматичне тестування є важливою складовою практик розробки програмного забезпечення, таких як Continuous Integration та Continuous Delivery (CI/CD). Для тестування взаємодії з реальним

сервером використовують HttpClientTestingModule та фейки для HTTP-запитів. [29]

Щоб перевірити, як працюють маршрутизатори, використовують RouterTestingModule. Також важливо використовувати інструменти, такі як Istanbul, для вимірювання покриття коду системи тестами.

Приклад роботи фреймворку зображено на рисунку 4.1.

Рисунок 4.1 Результат роботи Unit тестування для компонента системи

Для написання юніт-тестів використовувалися інструменти компіляції Karma і фреймворк Jasmine. На рисунку 4.2 наведено приклад коду юніт-тесту.

З метою забезпечення надійності та виключення можливості вплинути на реальні дані користувача, юніт-тести використовують фейкові дані. Для цього створюються спеціальні мокапи даних, які використовуються під час тестування

системи. На рисунку 4.3 наведено приклад мокапу даних для розробленої системи [29].

Результатом юніт-тестування є визначення покриття коду тестами. Це покриття вказує на якість тестів і генерується автоматично [29]. Також це може включати в себе перевірку правильності виконання конкретної функції, методу чи класу без взаємодії з іншими частинами програми.

Успішне юніт-тестування свідчить про те, що конкретна одиниця коду працює відповідно до очікувань і виконує визначені функції без помилок. В разі, якщо тест не пройдений, це може вказувати на наявність дефектів у функціоналі або несправності коду, що вимагає виправлення перед інтеграцією з іншими частинами програми. Його можна відстежувати за допомогою спеціальних інструментів фреймворку. На рисунку 4.4 показано приклад покриття для розробленої системи.

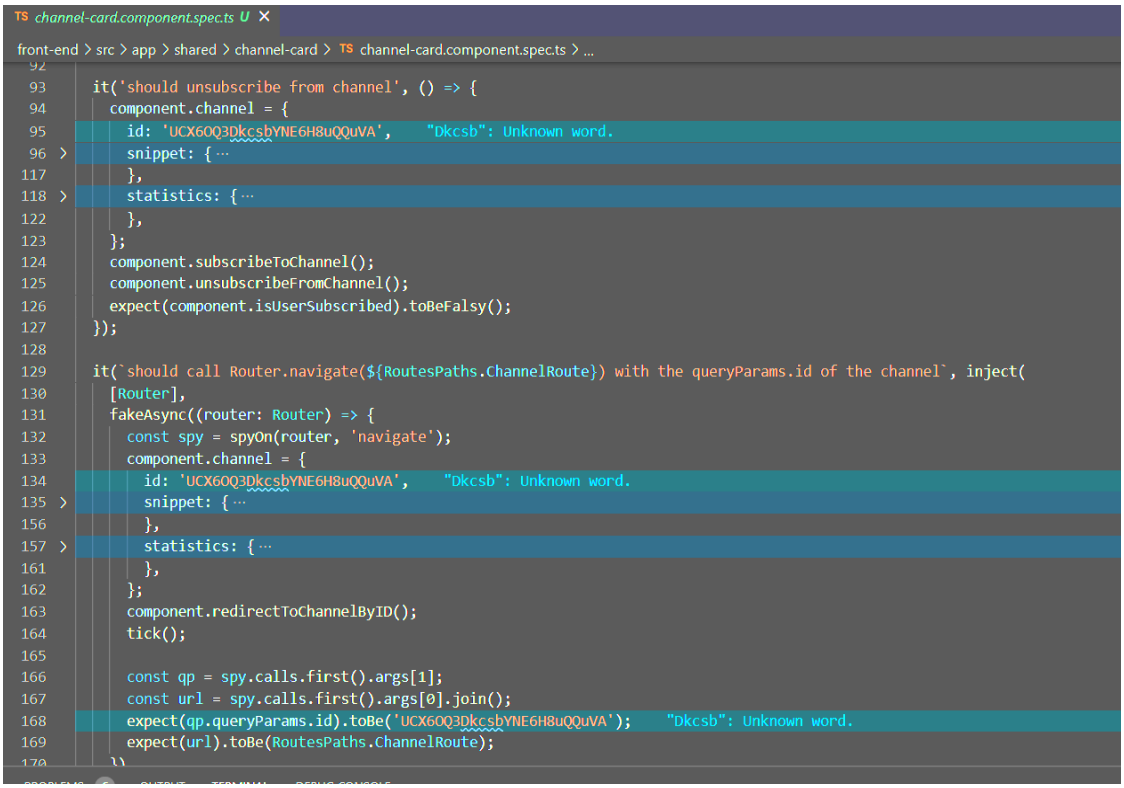

Рисунок 4.2 - Код Unit тесту для компонента системи

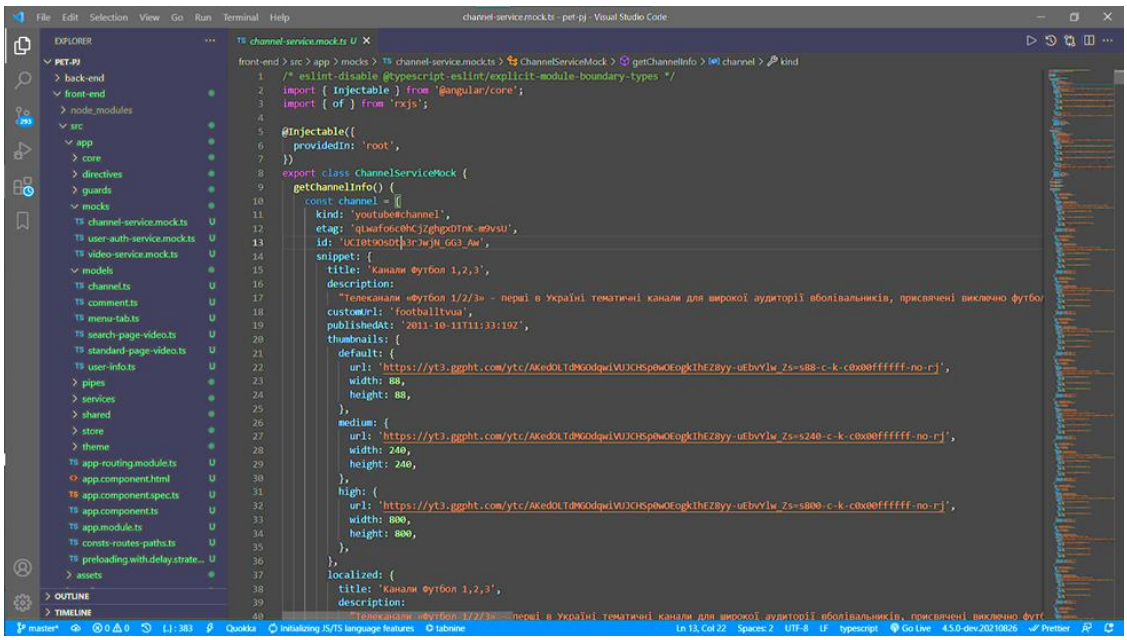

Рисунок 4.3 - Код мокапу даних для Unit тесту

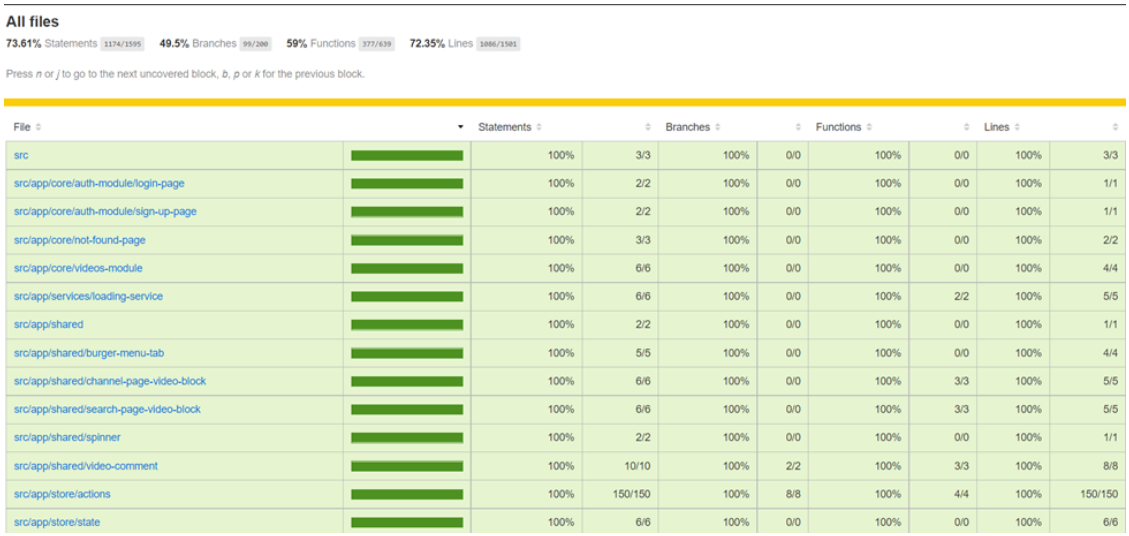

Рисунок 4.4 - Приклад покриття розробленої системи Unit тестами

Для створення інтеграційних тестів використовувалися інструменти компіляції Karma та фреймворк Jasmine. Приклад коду інтеграційних тестів можна побачити на рисунку 4.5, а результати їх виконання представлені на рисунку 4.6. [30]

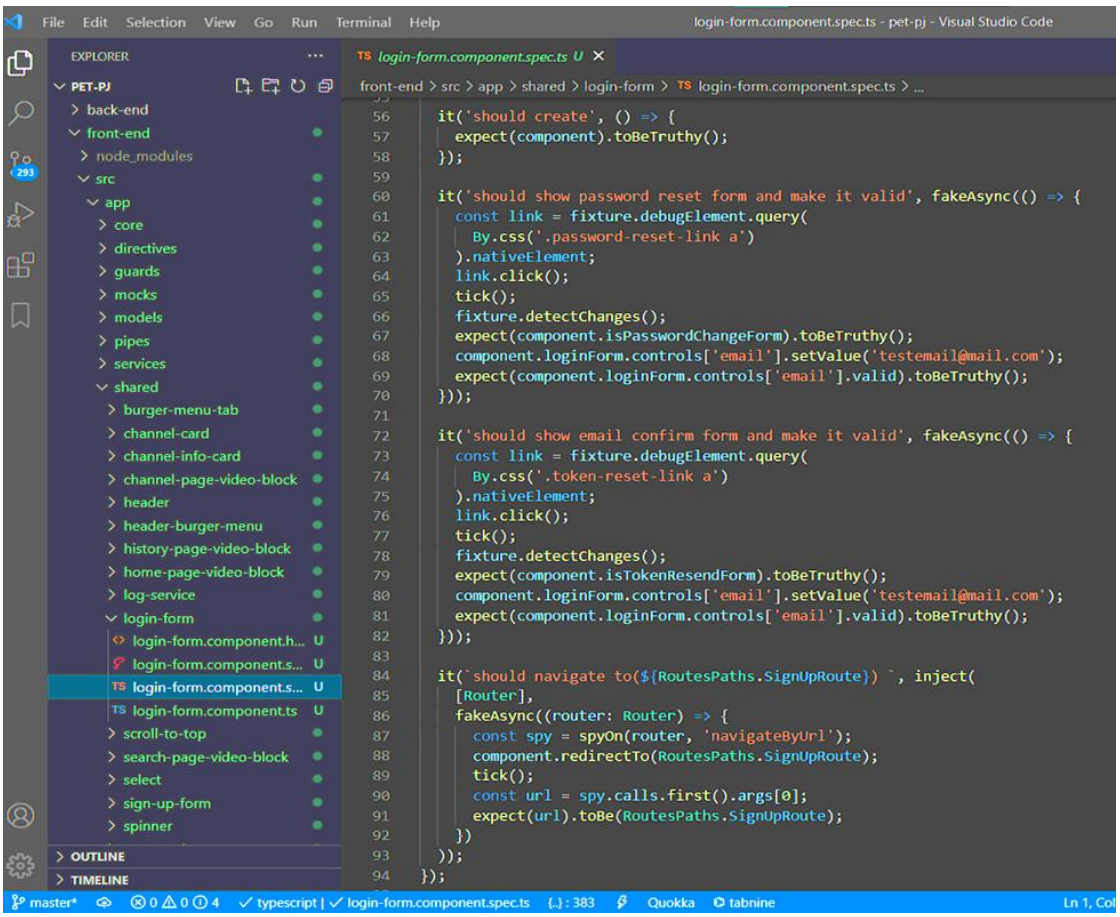

Рисунок 4.5 - Приклад коду інтеграційних тестів

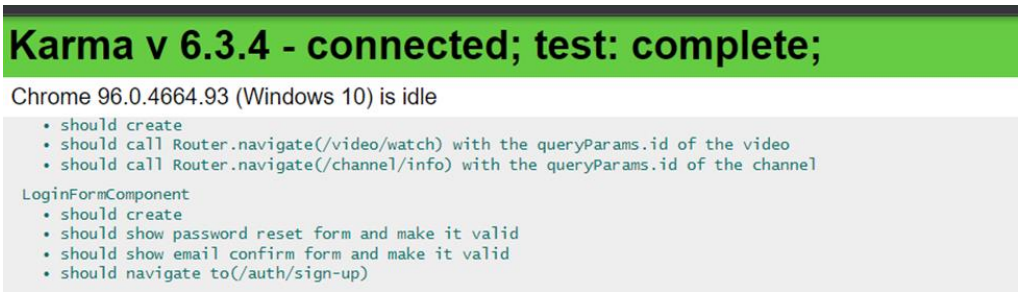

Рисунок 4.6 - Приклад результату роботи інтеграційних тестів

Для створення e2e тестів використовувався інструмент Cypress [30]. За допомогою Cypress можна протестувати систему, оцінюючи її роботу в повному обсязі. Приклад коду e2e тестів можна побачити на рисунку 4.7, а результати роботи цих тестів представлені на рисунку 4.8.

Рисунок 4.7 - Приклад коду e2e тестів

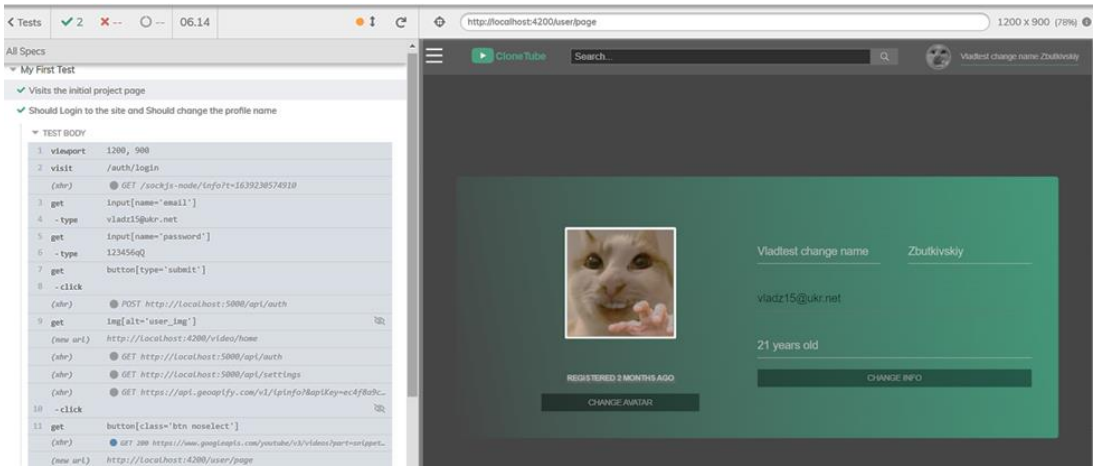

Рисунок 4.8 - Приклад результату роботи e2e тестів

4.3 Проведення аналізу розробленої системи порівняно із аналогом

Для порівняння розробленої системи користувачів та їх комунікації для відео сервісу використовувалась система YouTube. Данна система є актуальним конкурентом розробленої системи та має подібний функціонал що покращить якість аналізу розробленої системи [30].

Недоліки системи YouTube, як відеосервісу, включають в себе можливість появи неприязних або нецензурних коментарів, високу ймовірність поширення дезінформації через відеоконтент, проблеми із збереженням приватності користувачів у зв'язку зі збором та використанням персональних даних, а також проблеми із контентом, який порушує правила та стандарти спільноти.

В деяких випадках алгоритми рекомендацій можуть сприяти формуванню ізольованих бульбашок та підкріплювати користувачів обмеженими точками зору. Також, виникають сумніви стосовно адекватного виявлення та видалення неприпустимого контенту, а також питання щодо впливу алгоритмів рекомендацій на сприйняття інформації та формування публічної думки [30].

Розроблена система користувачів та їх комунікації для відео сервісу використовує NgRx, який застосовується для забезпечення передбачуваного стану контейнера, Angular компоненти для констроювання системи з окремих модулів, Mongo DB для збереження даних коритсувача, HTML, CSS, JavaScript, TypeScript для створення юзер інтерфейсу [30].

Переваги динамічної структури веб-проекту включають гнучкість та зручність розширення, спрощене управління великим обсягом контенту, ефективну роботу з даними в режимі реального часу, спрощену інтеграцію з різними сервісами та API, здатність до швидкого впровадження змін та оновлень без значного впливу на загальну стабільність системи. Така структура також сприяє покращенню користувацького досвіду через динамічне оновлення контенту, взаємодію з користувачем без перезавантаження сторінок та адаптацію до різних типів пристроїв та екранних розмірів [30].

Інтерфейс системи підтримує всі можливості, які були визначені на етапі початкового проектування системи, і був розроблений власноруч, без використання зовнішніх бібліотек та інструментів, з використанням гіпертекстової розмітки та каскадних стилів. Цей підхід до створення інтерфейсу надає найвищу масштабованість і підвищує легкість повторного використання компонентів системи [30].

Переваги розробки API для веб-сервісу включають забезпечення гнучкості та можливості взаємодії з іншими системами та додатками, спрощення розробки для різних платформ та пристроїв, створення стандартизованого інтерфейсу для обміну даними, забезпечення відокремленості між фронтендом та бекендом, що полегшує масштабування та підтримку кодової бази, а також сприяє обміну даними з іншими розробниками та партнерами.

API також дозволяє створювати розширені функціональні можливості та впроваджувати оновлення без необхідності внутрішньої модифікації додатку, що полегшує процес розробки та забезпечує високий рівень відкритості та доступності для користувачів [30].

Розроблені API відповідають за взаємодію системи з даними, пов'язаними з відео та користувачами. Реалізація цих API виконана за допомогою мови програмування JavaScript та платформи Node.js.

Для спілкування з клієнтом, система надсилає додаткові параметри до реалізованих API та очікує на відповідь. Для підвищення рівня захисту даних кожен запит супроводжується ключем, який ідентифікує конкретного користувача, що відправляє запит до API. Це забезпечує безпеку системи, запобігаючи доступу до даних інших користувачів, яким ці дані не належать, і захищає від можливості вторгнення та крадіжки даних у системі [30].

Порівняння розробленої системи користувачів та їх комунікації із системою користувачів YouTube наведенно у таблиці 4.1.

Таблиця 4.1 Порівняння розробленої системи користувачів та їх комунікації із системою користувачів YouTube

Характеристики системи	Розроблена система користувачів та їх комунікації	Система користувачів та їх комунікації YouTube
Маштабованість системи	+	-
Рівень контролю даних користувачем	+	+
Можливості фільтрації контенту	+	+

Продовження таблиці 4.1

Використання ресурсів інтернет з’єднання	+	-
Використання ресурсів пристрою	+	-
Наявний функціонал системи профілю користувачів	+	-
Оповіщення користувачів системи	+	+

У результаті порівняння системи користувачів та їх комунікації із системою користувачів YouTube можна зробити висновок, що у результаті розробки було досянуто переваг над існуючою системою по наведеним вище критеріям. Система працює стабільно, ефективно та виконує функції згідно визначених вимог.

4.4 Висновки

Після проведених тестів можна зробити висновок, що основна ціль проекту була досягнута. Розроблена система користувачів та їх комунікації розширює функціональні можливостей відео сервісів для передачі, збереження та відтворення відео-контенту, використавши поєднання актуальних технологій, дозволило легко розширювати систему і обновляти її складові і компонентну архітектуру відповідно до появи нових технологій та методів розробки.

Також забезпечено можливість впровадження системи на будь якій платформі, в тому числі для Web, Android та iOS системах. Програмне забезпечення працює стабільно, ефективно та виконує функції згідно визначених вимог.

РОЗДІЛ 5
ЕКОНОМІЧНА ЧАСТИНА

5.1 Проведення комерційного та технологічного аудиту

Нині існує багато методик, за якими можна здійснювати оцінювання ефективності науково-дослідних робіт. Під час виконання технічних спеціальностей магістерської кваліфікаційної роботи на тему «Розробка методів і програмних засобів системи користувачів та їх комунікації для відео сервісу». був обраний напрям у якому робота та її результат призначаються для виведення на ринок, тобто коли відбувається комерціалізація науковотехнічної розробки.[31]

Для обраного напряму мають бути виконані такі етапи роботи:

а) проведення комерційного аудиту науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

б) розрахунок витрат на здійснення науково-технічної розробки;

в) розрахунок економічної ефективності науково-технічної розробки у випадку її впровадження та комерціалізації потенційним інвестором і обґрунтування економічної доцільності комерціалізації потенційним інвестором розробленої у магістерській кваліфікаційній роботі науково-технічної розробки. [31]

Метою проведення комерційного та комерційного аудиту є оцінювання комерційного потенціалу розробки метод і засобів веб-системи користувачів та їх комунікації для відео сервісу. Для проведення комерційного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету: Яровий Ігор Олексійович, Нестерук Віталій Андрійович, Поліщук Микола Олегович. Для проведення комерційного аудиту було використано таблицю 5.1 в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерії	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою

Продовження таблиці 5.1

7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військовопромисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти р.	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Продовження таблиці 5.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 5.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 5.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 5.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали експерта		
	Яровий І. О.	Нестерук В. А.	Поліщук М. О.
	Бали, виставлені експертами:		
1	4	4	4

Продовження таблиці 5.3

2	3	3	3
3	4	3	4
4	3	3	3
5	4	3	4
6	4	3	3
7	3	3	4
8	4	4	4
9	3	3	4
10	3	3	4
11	4	4	4
12	4	4	4
Сума балів	43	44	45
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{43+40+45}{3} = 40$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 40 балів, що згідно таблиці 5.2 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Методи і програмі засоби системи користувачів та їх комунікації для відео сервісу, що розробляються в рамках магістерської кваліфікаційної роботи будуть цікаві інвесторам як потенційно прибутковий стартап [31].

Порівняємо нову розробку, що розробляється в магістерській кваліфікаційній роботі з аналогом, який існує на ринку.

Аналогом і даному випадку було обрано систему YouTube. Основними недоліками аналога є реклама, втручання в приватність та відсутність достатнього контролю над даними.

У розробці данні проблеми вирішується за допомогою функціоналу що надає повний контроль над своїми даними із можливістю їх зміни та видалення із системи також у розробленій системі відсутня додаткова реклама.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 5.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 5.4 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
Рівень контролю даних	5	6	0.83	50%
Точність визначення геолокації	5	5	1	10%
Використання ресурсів інтернет з’єднання, %	80	10	8	20%
Використання ресурсів пристрою, %	50	80	0.625	20%

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (5.1) та (5.2) і занесемо їх у відповідну колонку табл. 5.5.

$$q_i = \frac{P_{\text{Ні}}}{P_{\text{Бі}}} \tag{5.1}$$

або

$$q_i = \frac{P_{\text{Бі}}}{P_{\text{Нi}}} \tag{5.2}$$

де $P_{\text{Н}i}$, $P_{\text{Б}i}$ – числові значення i-го параметру відповідно нового і базового виробів

$$q_1 = \frac{5}{6} = 0.83;$$

$$q_2 = \frac{5}{5} = 1;$$

$$q_3 = \frac{80}{10} = 8;$$

$$q_4 = \frac{50}{80} = 0.625;$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$K_{\text{Я.В.}} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \tag{5.3}$$

$$K_{\text{Я.В.}} = 0.83 \cdot 0{,}5 + 1 \cdot 0{,}1 + 8 \cdot 0{,}2 + 0.625 \cdot 0{,}2 = 2{,}24$$

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{5.4}$$

де $I_{m.n.}$ – індекс технічних параметрів; $I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (5.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{\text{Б}ei}} \tag{5.5}$$

де $P_{Hei}, P_{\text{Бе}i}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

Відповідно тематиці роботи це можлива ціна використання ресурсу за одиницю часу. Аналоги розробленої системи за одиницю часу використовують один місяць, тому під час розрахунків використовувався один календарний місяць.

$$I_{e.n.} = \frac{150}{200} = 0{,}75$$

$$K = \frac{2.24}{0.75} = 2{,}98$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде більш конкурентоспроможною, ніж конкурентний товар.

5.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, спецустаткування для наукових робіт, програмне забезпечення для наукових робіт, витрати на роботи які виконують сторонні підприємства, установи і організації, інші витрати, накладні витрати. [31]

Основна заробітна плата кожного із дослідників О З , якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$З_0 = \frac{\text{М}}{\text{Т}_\text{р}} \cdot t(\text{грн}), \qquad (5.6)$$

де М – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$T_Р$ – число робочих днів в місяці; приблизно $T_р \approx 21 \dots 23$ дні;

t – число робочих днів роботи дослідника.

$$З_0 = \frac{25000}{22} \cdot 70 = 79520 \text{ (грн)}$$

$$З_0 = \frac{20000}{22} \cdot 66 = 59994 \text{ (грн)}$$

Зведемо сумарні розрахунки до таблиці 5.5.

Таблиця 5.5 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник	25000	1136	70	79520
Програміст	20000	909	66	59994
Всього				139514

Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата Зд всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_Д = (З_о + З_р) \cdot \frac{Н_{доп}}{100\%}, \quad (5.7)$$

$$З_Д = 0{,}11 \cdot 139514 = 15346{,}54 \text{ (грн)},$$

Нарахування на заробітну плату НЗП дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (5.8):

$$Н_{ЗП} = (З_о + З_д) \cdot \frac{\beta}{100} (грн), \quad (5.8)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = (139514 + 15346{,}54) \cdot \frac{22}{100} = 34069.31 \text{ (грн)}$$

Витрати на матеріали М та комплектуючі К, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$М = \sum_i^n H_i \cdot Ц_i \cdot К_i - \sum_i^n В_i \cdot Ц_i (грн), \quad (5.9)$$

де $Н_i$ – витрати матеріалу і-го найменування, кг;

$Ц_i$ – вартість матеріалу і-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, Кі = (1,1…1,15);

$В_i$ – маса відходів матеріалу і-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу і-го найменування, грн/кг;

n – кількість видів матеріалів.

Таблиця 5.6 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн	Витрачено	Вартість витраченого матеріалу, грн.
Папір офісний	2	300	600
Канцелярське приладдя	50	4	200
SSD накопичував	1500	2	3000
Всього			3800
З врахуванням коефіцієнта транспортування			4000

Методи і програмі засоби системи користувачів та їх комунікації для відео сервісу для наукової роботи включають витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення необхідного для проведення дослідження [31]. Для нової розробки використовувались безкоштовні програмні засоби. Безкоштовне середовище розробкиVisual Studio Code, безкоштовний інструмент для контролю версій коду проєкту Git, безкоштовне хмарне середовище для збереження коду проекту GitHub, безкоштовне середовище для розроблення дизайну системи Figma, безкоштовне хмарне середовище для розгортання системи Heroku та безкоштовне середовище для планування задач розробки Trello [31].

Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання даного етапу роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$А = \frac{Ц \cdot Т}{Т_{кор} \cdot 12} (грн), \quad (5.10)$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункту 137.3.3 Податкового кодексу амортизація нараховується на основні засоби вартістю понад 20000 грн [31]. В нашому випадку для написання магістерської роботи використовувався персональний ноутбук Acer Nitro 5 AN515-57-53P6 вартістю 25000 грн.

$$А = \frac{25000 \cdot 3}{3 \cdot 12} = 2083{,}3 \text{ (грн)}$$

До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \text{ (грн)}, \qquad (5.11)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

Для написання магістерської роботи використовується персональний комп’ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}05 \cdot 528 \cdot 7 \cdot 0{,}05}{0{,}8} = 115{,}5 \text{(грн)}$$

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення,

водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати $В_{нзв}$ можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_О + З_Р) \cdot \frac{Н_{НЗВ}}{100\%} (грн), \quad (5.12)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати»

$$В_{нзв} = 139514 \cdot \frac{100}{100\%} = 139514 \text{ (грн)}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР:

$$В = 139514 + 15346{,}54 + 34069.31 + 4000 + 2083{,}3 + 115{,}5 + 139514 = 334672{,}97 \text{ (грн)}$$

Витрати на службові відрядження, витрати на спецустаткування для наукових робіт, витрати на програмне забезпечення для наукових робіт, витрати на роботи, які виконують сторонні підприємства, установи, організації та інші витрати в нашому досліджені не враховуються оскільки їх не було [31].

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta} (грн), \quad (5.13)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт $\beta = 0{,}9$.

Звідси:

$$ЗВ = \frac{334672{,}97}{0{,}9} = 371825{,}52 \text{ (грн)}$$

5.2 Прогнозування комерційного та технологічного аудиту науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства ΔПі, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_{\text{і}} = \sum_{i}^{n}(\Delta Ц_0 \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{v}{100}) \text{ (грн)}, \qquad (5.14)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

λ – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт λ = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х – ставка податку на прибуток. У 2023 році – 18%.

Припустимо, що при впроваджені результатів наукової розробки покращується якість методів і програмних засобів системи користувачів та їх комунікації для відео сервісу. Прогнозується, що дохід від одного користувача зросте на 50 грн. Кількість одиниць реалізованої продукції (в даному випадку кількість користувачів) також збільшиться: протягом першого року на 5000,

протягом другого року – на 10000, протягом третього року на 25000. Кількість користувачів до реалізації становить 50000, а дохід від одного користувача складає 150 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [50 \cdot 50000 + (150 + 50) \cdot 5000] \cdot 0{,}833 \cdot 0{,}25 \cdot (1 + \frac{18}{100}) = 860072{,}5 \text{ (грн)}$$

$$\Delta П_2 = [50 \cdot 50000 + (150 + 50) \cdot (5000 + 10000)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 1351542{,}5 \text{ (грн)}$$

$$\Delta П_3 = [50 \cdot 50000 + (150 + 50) \cdot (5000 + 10000 + 25000)] \cdot 0{,}833 \cdot 0{,}25 \cdot (1 + \frac{18}{100}) = 2580217{,}5 \text{(грн)}$$

Таким чином, розрахунки показують, що відповідно прогнозуванню комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

5.4 Розрахунок економічної ефективності науково-технічної розробки за її можливості комерціалізації потенційним інвестором

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності [31].

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}\,, \qquad (5.15)$$

$k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науковотехнічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{інв}$= 2…5).

$$PV = 2 \cdot 371825{,}52 = 74361{,}04 \text{ (грн)}$$

Розрахуємо абсолютну ефективність вкладених інвестицій Еабс згідно наступної формули:

$$\text{Е}_{\text{абс}} = (\text{ПП} - PV)\,, \qquad (5.16)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$\text{ПП} = \sum_{\text{i}}^{\text{T}} \frac{\Delta\text{П}_{\text{i}}}{(1+\tau)^t}\,, \qquad (5.17)$$

де $\Delta\text{П}_{\text{i}}$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$\text{ПП} = \frac{860072{,}5}{(1+0{,}2)^1} + \frac{1351542{,}5}{(1+0{,}2)^2} + \frac{2580217{,}5}{(1+0{,}2)^3}$$

$$= 716727{,}08 + 93871{,}18 + 1493181{,}42 = 2303779{,}68 \text{ (грн)}$$

$$\text{Е}_{\text{абс}} = (2303779{,}68 - 74361{,}04) = 2229418{,}64 \text{ (грн)}$$

Оскільки $Е_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій . Для цього користуються формулою:

$$Е_в = \sqrt[Т_ж]{1 + \frac{Е_{абс}}{PV}} - 1 , \quad (5.18)$$

де $Т_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{1 + \frac{2229418,64}{74361,041}} - 1 = 310\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f , \quad (5.19)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{min} = 0,18 + 0,05 = 0,23$$

Так як $Е_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ОК} = \frac{1}{E_{в}}, \quad (5.20)$$

$$\mathrm{T_{ок}} = \frac{1}{3{,}10} = 0{,}32 \text{ роки}$$

Так як $\mathrm{T_{ок}} \leq 3 \ldots 5$ -ти років, то фінансування даної наукової розробки є доцільним.

5.5 Висновки

В даному розділі було здійснено оцінювання комерційного потенціалу розробки системи користувачів та їх комунікації для відео сервісу у вигляді веб додатку для платформ Windows, Linux, iOS та Android [31].

Проведено комерційного аудит з залученням трьох експертів. Аналіз експертних даних показав, що рівень комерційного потенціалу розробки вище середнього. Дослідження комерційного потенціалу розробки показав, що програмний продукт за своїми характеристиками випереджає аналогічні програмні продукти і є перспективною розробкою. Він має кращі функціональні показники, а тому є конкурентоспроможним товаром на ринку.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складає 34672,97 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 371825,52 грн.

Вкладені інвестиції в даний проект окупляться через 5 з половиною місяців при прогнозованому прибутку 2229418,64 грн. за три роки.

ВИСНОВКИ

У магістерській кваліфікаційній роботі проведено детальне дослідження систем користувачів та їх комунікацій та існуючих систем та виконано проектування та розробку методів і програмних засобів системи користувачів та їх комунікації для відео сервісу.

У результаті аналізу предметної області було виявлено, що в сучасній сфері веб застосунків існує потреба у реалізації системи що допоможе платформам покращувати конкурентоспроможність, залучати та утримувати аудиторію, а також збільшувати конкурентоспроможність та задоволення зростаючих потреб споживачів у цьому сегменті ринку. через ефективну систему комінікацій користувачів. Враховуючи ряд вимог до розробки веб застосунку було вирішено обрати архітектуру MVC та здійснити реалізацію додатку за допомогою мультиплатформенного фреймворку Angular на мові TS. Система використовує наступні технології/компоненти: NgRx, який застосовується для забезпечення передбачуваного стану контейнера, Angular компоненти для констроювання системи з окремих модулів, Mongo DB для збереження даних та контенту, YouTube Api для відтворення контенту, HTML, CSS, JavaScript, TypeScript для юзер інтерфейсу.

Проведено аналіз з такими аналогами на ринку, як YouTube, Netflix та Twitch. В процесі здійснення аналізу було визначено слабкі та сильні сторони аналогів та визначено відмінність розроблюваної системи від її конкурентів. Ця система відрізняється від існуючих можливістю легкої розширюваності і оновлення складових та компонентної архітектури відповідно до нових технологій і розробних методів.

Проведено аналіз найвідоміших засобів реалізації програмного продукту, які мають можливість одночасної розробки під різні платформи. Було налаштовано робоче середовище у вигляді Visual Studio Code для розробки системи, розроблено компоненти для обробки API, створено адаптивний веб-інтерфейс, який відповідає актуальним вимогам і стандартам. Також була

розроблена мікросервісна архітектура системи, і наведено діаграму цієї архітектури. Angular framework використовується для створення модульної та масштабованої структури проекту; NgRx забезпечує контроль за станом проекту, а Mongo DB дозволяє збереження контенту та інформації користувача.

Для забезпечення надійності системи користувачів використовується формат обміну даними JSON і шифрування інформації криптографічними методами.

В результаті роботи удосконалено методи поєднання модульного фреймворку «Angular», розроблених API та програмного забезпечення, що вирішує питання комунікації користувачів і на відміну від існуючих систем дозволяє легко та швидко модифікувати систему, архітектуру та складові за рахунок модульності та динамічності системи, що дає можливість підтримувати систему актуальною та створено веб застосунок за допомогою фреймворку Angular, який доступний для платформ Windows, Linux, iOS та Android.

Удосконалено методи розробки системи користувачів для відео сервісу результатом чого є те що система проста в користуванні, модульна та дозволяє доробку та підтримує можливість масштабування, а також має забезпечення приватності користувацьких даних за допомогою шифрування.

Для повноцінної роботи системи було імплементовано декілька API. Вони реалізуються роботу системи з даними для відео та користувача. API реалізовані за допомогою програмної мови JS та програмної платформи Node.js. Для взаємодії з клієнтом, він надсилає додаткові параметри для реалізованих API та очікує відповідь, також кожен запит супроводжується ключем що ідентифікує який саме користувач надсилає запит до API. Це захищає систему від доступу до даних іншими користувачами яким ці дані не належать та захищає її від злому та викрадання даних користувача.

Після здійснення розробки програмного забезпечення було проведено тестування системи та виправлено наявні баги у системі. Для написання юніт тестів використовувався компілятор Karma та фрейморк Jasmine. Для забезпечення надійності та виключення можливості змінити реальні данні

користувача Unit тести працюють з фейковими даними. Для написання інтеграційних тестів використовувався компілятор Karma та фрейморк Jasmine. Для написання e2e тестів використовувався Cypress. Cypress дозволяє протестувати систему за допомогою e2e тестів, що оцінює роботу системи повністю. Після виконання тестування було доведено, що система відповідає своїй головній задачі, а також виконує усі функції коректно.

Економічні розрахунки витрат та прибутків від впровадження розробки, термінів окупності та визначення її економічного ефекту показали, що в сучасній технічній ситуації розробка є конкурентоспроможною на IT-ринку.

Робота має наукову та практичну цінність, оскільки розроблені модулі системи користувачів та їх комунікації для відео сервісу можуть використовуватись для інших проєктів, а також має високу конкурентоспроможність на ринку, оскільки має мало прямих конкурентів при цьому маючи низку вагомих переваг.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Огляд фільтрації контенту стримінгового відео сервісу / В. С. Збитківський, І. В. Богач / матер. LI Науко-технічній конференції підрозділів ВНТУ 2021. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2022/author/submission/14315

2. Огляд шифрування інформації користувача стримінгового відео сервісу / В. С. Збитківський, І. В. Богач / матер. Всеукраїнській науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2022)». URL: https://conferences.vntu.edu.ua/index.php/mn/mn2022/author/ submission/15252

3. Розробка багатофакторної аутентифікації користувача відео сервісу / В. С. Збитківський, І. В. Богач / матер. LII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2023). URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2023/paper/view/17735 /14749

4. Архітектура YouTube 2023. – URL: https://www.researchgate.net/publication/251898765_Relevance-Based_Ranking_of_Video_Comments_on_ YouTube

5. Twitch Engineering: An Introduction and Overview 2023. – URL: https://blog.twitch.tv/en/2015/12/18/twitch-engineering-an-introduction-and-overview-a23917b71a25/

6. Мікросервіси та архітектура Netflix 2019 2023. – URL: https://www.geeksforgeeks.org/system-design-netflix-a-complete-architecture/

7. Компоненти Angular URL: https://web-dev.guru/angular-book/angular-components.

8. Angular архітектура NgRx 2022. URL: https://techtalks.qima.com/angular-get-started-with-ngrx/.

9. Angular компоненти та архітектура 2022. URL: https://shorturl.at/dCFIN.

10. Introduction to Angular concepts 2023. URL: https://angular.io/guide/architecture.

11. Angular Architecture – Detailed Explanation 2023. URL: https://www.interviewbit.com/blog/angular-architecture/.

12. Динамічні компоненти Angular 2023. URL: https://blog.bitsrc.io/dynamic-components-in-angular-9ddc346e2742.

13. Веб інтерфейси 2011. URL: https://diib.com/learn/web-interface/.

14. Best Practices & Guidelines for web applications using Angular 2023. URL: https://blogs.halodoc.io/angular-best-practices/

15. Angular: Building an Interface 2023. URL: https://www.linkedin.com/learning/angular-building-an-interface

16. Create User Interface with an Angular component 2023. https://www.infragistics.com/products/ignite-ui-angular/angular/components/general/wpf-to-angular-guide/create-ui-with-components

17. Основи GIT 2019. URL: https://www.atlassian.com/git.

18. GIT коротко про головне 2021. URL: https://githowto.com/uk.

19. Visual Studio Code архітектура 2021. URL: https://franz-ajit.medium.com/understanding-visual-studio-code-architecture-5fc411fca07.

20. Build a Serverless REST API with Angular, Persistence, and Security 2023. URL:https://www.freecodecamp.org/news/serverless-rest-api-with-angular-persistence-and-security-ff274f04e3d0/

21. Using Version Control in VS Code 2022. URL: https://code.visualstudio.com/docs/editor/versioncontrol.

22. Різниця між HTTP та HTTPS 2022. URL: https://hostiq.ua/wiki/ukr/http-https/.

23. Документація HEROKU 2022. URL: https://devcenter.heroku.com/.

24. Інтеграційне тестування 2023. URL: https://qalight.ua/baza-znaniy/integratsijne-testuvannya/.

25. Angular 5: Unit тести | MarkupUA 2023. URL: https://markup-ua.com/angular-5-unit-testi/.

26. Види тестування та підходи до їх застосування 2023. URL: https://qalearning.com.ua/.

27. Юніт-тести на Angular: практичний гайд з тестування директив 2023. URL:https://highload.today/uk/blogs/yunit-testi-na-angular-praktichnij-gajd-z-testuvannya-direktiv/

28. The State of end-to-end testing with Angular 2022. URL: https://blog.angular.io/the-state-of-end-to-end-testing-with-angular-d175f751cb9c.

29. Jasmine documentation 2013. URL: https://jasmine.github.io/tutorials/your_first_suite

30. How to perform End to End Testing in Angular 2022. URL: https://www.browserstack.com/guide/how-to-perform-end-to-end-testing-in-angular

31. В. О. Козловський, О. Й. Лесько, В. В. Кавецький. *Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт*. Вінниця, Україна : ВНТУ, 2021. – 42 с

99

ДОДАТОК А (обов'язковий)

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів і програмних засобів системи користувачів та їх комунікації для відео сервісу» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., Черноволик Г. О.

"19" вересня 2023 р.

Виконав:

_____ студент гр.1ПІ-22м В.С. Збитківський

"19" вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів системи користувачів та їх комунікації для відео сервісу».

Галузь застосування - системи відео-стримінгу.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є розширення функціональних можливостей відео сервісів для передачі, збереження та відтворення відео-контенту шляхом впровадження або покращення та модернізації системи користувачів та їх комунікації.

Призначення роботи – розробка системи у вигляді модернізованої системи користувачів та їх комунікації для веб-сервісу, яку можна використовувати для обміну відеоконтентом, а також спілкування з іншими користувачами. Цей прототип може служити основою для подальшого розвитку та розширення функціональності веб-відео сервісу. В роботі пропонується використання поєднання актуальних технологій, що дозволить легко розширювати систему і обновляти її складові і компонентну архітектуру відповідно до появи нових технологій та методів розробки. Також забезпечує можливість впровадження системи на будь якій платформі, в тому числі для Web, Android та iOS системах.

4 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Огляд фільтрації контенту стримінгового відео сервісу / В. С. Збитківський, І. В. Богач / матер. LI Науко-технічній конференції підрозділів ВНТУ 2021.

URL: https://conferences.vntu.edu.ua/index.php/mn/mn2022/author/submission/14315.

2. Огляд шифрування інформації користувача стримінгового відео сервісу / В. С. Збитківський, І. В. Богач / матер. Всеукраїнській науково-практичній Інтернет-конференції студентів, аспірантів та молодих науковців «Молодь в науці: дослідження, проблеми, перспективи (МН-2022)». URL: https://conferences.vntu.edu.ua/index.php/mn/mn2022/author/submission /15252.
3. Розробка багатофакторної аутентифікації користувача відео сервісу / В. С. Збитківський, І. В. Богач / матер. LII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2023/paper/view/17735/14749
4. Brian Hogan. Web Design for Developers: A Programmer's Guide to Design Tools and Techniques, 2010. 15 – 200 с.

5. Технічні вимоги

Angular framework, Mongo DB, Express.js, HTML, CSS, JavaScript, TypeScript, SCSS, BCrypt, Unit tests, Integration tests, E2E tests, Heroku, Jasmine, Karma, Cypress; вихідні дані часові обмеження відповіді системи – 2000 ms, кількість користувачів – до 100, роздільна здатність файлів – до 720 p, вага фото користувача – не більше 1 mb, формат фото користувача – .jpg or .png, інтернет підключення – від 500 kb/s.

6. Конструктивні вимоги.

Розроблена структуру системи повинна забезпечувати якість та надійність роботи. Графічний інтерфейс повинен мати підтримку різних екранів для легкого та зрозумілого користувачеві використання, відповідно до стандартів.Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз предметної галузі системи користувачів	20.09.2023 - 27.09.2023
2	Постановка основних задач роботи	28.09.2023 - 10.10.2023
3	Проектування структури та компонентів системи	11.10.2023 - 27.10.2023
4	Реалізація системи користувачів та їх комунікації за допомогою Angular framework, Express.js та Mongo DB	28.10.2023 - 15.11.2023
5	Тестування програмного забезпечення	16.11.2023 - 21.11.2023
6	Економічний розділ	22.11.2023 - 27.11.2023
7	Оформлення пояснювальної записки таграфічного матеріалу	28.11.2023 – 8.12.2023

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

103

ДОДАТОК Б (обов'язковий)

Протокол перевірки МКР на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методів і програмних засобів системи користувачів та їх комунікації для відео сервісу**

Тип роботи: кваліфікаційна робота
Підрозділ : кафедра програмного забезпечення, ФІТКІ, 1ПІ – 22м
Науковий керівник: к.т.н. доц. Черноволик Г. О.

Unicheck	
Оригінальність	**90.5%**
Схожість	9.5%

Аналіз звіту подібності

- **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлений з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Збитківський В. С.

Керівник роботи _____ Черноволик Г. О.

ДОДАТОК В (обов'язковий)

Лістинг вихідного коду

app.component.ts:

```
import { Component, OnInit } from '@angular/core';
import { Store } from '@ngrx/store';
import { take } from 'rxjs/operators';
import { AuthService } from './services/user-auth-service/user-auth-service';
import { GetUserInfo } from './store/actions/user-actions';
import { selectUserIsLogin } from './store/selectors/user-selectors';
import { AppState } from './store/state/app.state';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
})
export class AppComponent implements OnInit {
  constructor(
    private authService: AuthService,
    private store: Store<AppState>
  ) {}

  ngOnInit(): void {
    if (this.authService.isUserLogged()) {
      this.store
        .select(selectUserIsLogin)
        .pipe(take(1))
        .subscribe((res) => {
          if (res) {
            this.store.dispatch(GetUserInfo());
          }
        });
    }
  }
}
```

app.state.ts:

```
import { VideoState, initialVideoState } from './standart.videos.state';
import {
  SearchVideoState,
  initialSearchVideoState,
} from './search.videos.state';
import { ChannelState, initialChannelState } from './channel-state';
import { initialUserState, UserState } from './user-state';
import { CommentsState, initialCommentsState } from './comment.state';
import { initialMyCommentsState, MyCommentsState } from './my-comments-state';
export interface AppState {
  homePageVideos: VideoState;
  searchPageVideos: SearchVideoState;
```

```
  channel: ChannelState;
  user: UserState;
  comments: CommentsState;
  myComments: MyCommentsState;
}

export const initialAppState: AppState = {
  homePageVideos: initialVideoState,
  searchPageVideos: initialSearchVideoState,
  channel: initialChannelState,
  user: initialUserState,
  comments: initialCommentsState,
  myComments: initialMyCommentsState,
};
export function getInitialState(): AppState {
  return initialAppState;
}
```

channel-state.ts:

```
import { SearchVideo } from 'src/app/models/search-page-video';
import { Channel } from '../../models/channel';
export interface ChannelState {
  channel: Channel;
  channels: Channel[];
  videos: SearchVideo[];
  pageTokens: {
    nextPageToken: string;
    prevPageToken: string;
  };
}

export const initialChannelState: ChannelState = {
  channel: {
    id: null,
    snippet: {
      title: null,
      description: null,
      publishedAt: null,
      thumbnails: {
        default: {
          url: null,
          width: 0,
          height: 0,
        },
        medium: {
          url: null,
          width: 0,
          height: 0,
        },
        high: {
```

```
          url: null,
          width: 0,
          height: 0,
        },
      },
    },
    statistics: {
      viewCount: 0,
      subscriberCount: 0,
      videoCount: 0,
    },
  },
  channels: [],
  videos: [],
  pageTokens: {
    nextPageToken: null,
    prevPageToken: null,
  },
};
```

comment.state.ts:

```
import { Comment } from '../../models/comment';
export interface CommentsState {
  comments: Comment[];
  pageTokens: {
    nextPageToken: string;
    prevPageToken: string;
  };
}

export const initialCommentsState: CommentsState = {
  comments: [],
  pageTokens: {
    nextPageToken: null,
    prevPageToken: null,
  },
};
```

search.videos.state.ts:

```
import { Video } from 'src/app/models/standard-page-video';
import { SearchVideo } from '../../models/search-page-video';
export interface SearchVideoState {
  videos: SearchVideo[];
  video: Video;
  pageTokens: {
    nextPageToken: string;
    prevPageToken: string;
  };
  autoQuery: string[];
```

```
}

export const initialSearchVideoState: SearchVideoState = {
  videos: [],
  autoQuery: [],
  video: {
    id: null,
    snippet: {
      publishedAt: null,
      channelId: null,
      title: null,
      description: null,
      thumbnails: {
        default: {
          url: null,
          width: 0,
          height: 0,
        },
        medium: {
          url: null,
          width: 0,
          height: 0,
        },
        high: {
          url: null,
          width: 0,
          height: 0,
        },
        standard: {
          url: null,
          width: 0,
          height: 0,
        },
        maxres: {
          url: null,
          width: 0,
          height: 0,
        },
      },
      channelTitle: null,
    },
    statistics: {
      viewCount: 0,
      likeCount: 0,
      dislikeCount: 0,
    },
    contentDetails: {
      bmukkRating: null,
    },
  },
  pageTokens: {
```

```
    nextPageToken: null,
    prevPageToken: null,
  },
};
```

standart.videos.state.ts:

```
import { Video } from '../../models/standard-page-video';
export interface VideoState {
  videos: Video[];
  video: Video;
  pageTokens: {
    nextPageToken: string;
    prevPageToken: string;
  };
}

export const initialVideoState: VideoState = {
  videos: [],
  video: {
    id: null,
    snippet: {
      publishedAt: null,
      channelId: null,
      title: null,
      description: null,
      thumbnails: {
        default: {
          url: null,
          width: 0,
          height: 0,
        },
        medium: {
          url: null,
          width: 0,
          height: 0,
        },
        high: {
          url: null,
          width: 0,
          height: 0,
        },
        standard: {
          url: null,
          width: 0,
          height: 0,
        },
        maxres: {
          url: null,
          width: 0,
          height: 0,
```

```
        },
      },
      channelTitle: null,
    },
    statistics: {
      viewCount: 0,
      likeCount: 0,
      dislikeCount: 0,
    },
    contentDetails: {
      bmukkRating: null,
    },
  },
  pageTokens: {
    nextPageToken: null,
    prevPageToken: null,
  },
};
```

user-state.ts:

```
import { UserInfo, UserSettings } from '../../models/user-info';
export interface UserState {
  user: UserInfo;
  settings: UserSettings;
  login: boolean;
  isAuthFailed: boolean;
  signUp: boolean;
  isTokenSend: boolean;
  isDataChanged: boolean;
  avatarUrl: string;
  selectedUser: UserInfo;
}

export const initialUserState: UserState = {
  user: {
    _id: null,
    firstName: null,
    lastName: null,
    email: null,
    date: null,
    registrationDate: null,
    dislikedVideosIds: [],
    likedVideosIds: [],
    historyVideos: {
      VideoIds: [],
      VideoTimes: [],
    },
    subscriptions: [],
    avatarUrl: null,
    contentChild: false,
```

```
  },
  settings: {
    userId: null,
    region: '',
    theme: null,
  },
  login: false,
  isAuthFailed: false,
  signUp: false,
  isTokenSend: false,
  isDataChanged: false,
  avatarUrl: null,
  selectedUser: {
    _id: null,
    firstName: null,
    lastName: null,
    email: null,
    date: null,
    registrationDate: null,
    dislikedVideosIds: [],
    likedVideosIds: [],
    historyVideos: {
      VideoIds: [],
      VideoTimes: [],
    },
    subscriptions: [],
    avatarUrl: null,
    contentChild: false,
  },
};
```

user-actions.ts:

```
import { User, UserInfo, UserSettings } from './../../models/user-info';
import { Comment } from 'src/app/models/comment';
import { createAction, props } from '@ngrx/store';

export enum EUUserActions {
  SignUp = '[USER] SIGN_UP',
  SignUpSuccess = '[USER] SIGN_UP_SUCCESS',
  SignUpError = '[USER] SIGN_UP_ERROR',
  Login = '[USER] LOGIN',
  LoginSuccess = '[USER] LOGIN_SUCCESS',
  LoginError = '[USER] LOGIN_ERROR',
  Logout = '[USER] LOGOUT',
  LogoutSuccess = '[USER] LOGOUT_SUCCESS',
  GetUserInfo = '[USER] GET_USER_INFO',
  GetUserInfoSuccess = '[USER] GET_USER_INFO_SUCCESS',
  ChangeUserInfo = '[USER] CHANGE_USER_INFO',
  ChangeUserInfoSuccess = '[USER] CHANGE_USER_INFO_SUCCESS',
  ChangeUserInfoFailed = '[USER] CHANGE_USER_INFO_FAILED',
```

```
  LikeVideo = '[USER] LIKE_VIDEO',
  LikeVideoSuccess = '[USER] LIKE_VIDEO_SUCCESS',
  UnLikeVideo = '[USER] UN_LIKE_VIDEO',
  UnLikeVideoSuccess = '[USER] LIKE_UNLIKE_VIDEO_SUCCESS',
  DislikeVideo = '[USER] DISLIKE_VIDEO',
  DislikeVideoSuccess = '[USER] DISLIKE_VIDEO_SUCCESS',
  UnDislikeVideo = '[USER] UN_DISLIKE_VIDEO',
  UnDislikeVideoSuccess = '[USER] UN_DISLIKE_VIDEO_SUCCESS',
  AddVideoToHistory = '[USER] ADD_VIDEO_TO_HISTORY',
  AddVideoToHistorySuccess = '[USER] ADD_VIDEO_TO_HISTORY_SUCCESS',
  UpdateSettings = '[USER] UPDATE_SETTINGS',
  UpdateSettingsSuccess = '[USER] UPDATE_SETTINGS_SUCCESS',
  UpdateSettingsFailed = '[USER] UPDATE_SETTINGS_FAILED',
  ChangeAvatar = '[USER] CHANGE_AVATAR',
  ChangeAvatarSuccess = '[USER] CHANGE_AVATAR_SUCCESS',
  DeleteAvatar = '[USER] DELETE_AVATAR',
  DeleteAvatarSuccess = '[USER] DELETE_AVATAR_SUCCESS',
  PostComment = '[USER] POST_COMMENT',
  PostCommentSuccess = '[USER] POST_COMMENT_SUCCESS',
  GetMyCommentsByVideoId = '[Video] GET_MY_COMMENTS_BY_VIDEO_ID',
  GetMyCommentsByVideoIdSuccess = '[Video] GET_MY_COMMENTS_BY_VIDEO_ID_SUCCESS',
  GetMyCommentsByVideoIdError = '[Video] GET_MY_COMMENTS_BY_VIDEO_ID_ERROR',
  GetUserSettings = '[USER] GET_USER_SETTINGS',
  GetUserSettingsSuccess = '[USER] GET_USER_SETTINGS_SUCCESS',
  GetUserInfoById = '[USER] GET_USER_INFO_BY_ID',
  GetUserInfoByIdSuccess = '[USER] GET_USER_INFO_BY_ID_SUCCESS',
  SendEmailToken = '[USER] SEND_EMAIL_TOKEN',
  SendEmailTokenSuccess = '[USER] SEND_EMAIL_TOKEN_SUCCESS',
  PasswordResetToken = '[USER] PASSWORD_RESET_TOKEN',
  PasswordResetTokenSuccess = '[USER] PASSWORD_RESET_TOKEN_SUCCESS',
  PasswordChange = '[USER] PASSWORD_CHANGE_TOKEN',
  PasswordChangeSuccess = '[USER] PASSWORD_CHANGE_SUCCESS',
}

export const SignUp = createAction(
  EUUserActions.SignUp,
  props<{ payload: { user: User } }>()
);
export const SignUpSuccess = createAction(
  EUUserActions.SignUpSuccess,
  props<{ payload: boolean }>()
);
export const SignUpError = createAction(
  EUUserActions.SignUpError,
  props<{ payload: boolean }>()
);
export const Login = createAction(
  EUUserActions.Login,
  props<{ payload: { user: User } }>()
);
export const LoginSuccess = createAction(
```

```
  EUUserActions.LoginSuccess,
  props<{ payload: boolean }>()
);
export const LoginError = createAction(
  EUUserActions.LoginError,
  props<{ payload: boolean }>()
);
export const Logout = createAction(EUUserActions.Logout);
export const LogoutSuccess = createAction(
  EUUserActions.LogoutSuccess,
  props<{ payload: boolean }>()
);
export const GetUserInfo = createAction(EUUserActions.GetUserInfo);
export const ChangeUserInfo = createAction(
  EUUserActions.ChangeUserInfo,
  props<{ payload: { user: UserInfo } }>()
);
export const ChangeUserInfoSuccess = createAction(
  EUUserActions.ChangeUserInfoSuccess,
  props<{ payload: boolean }>()
);
export const ChangeUserInfoFailed = createAction(
  EUUserActions.ChangeUserInfoFailed,
  props<{ payload: boolean }>()
);
export const GetUserInfoSuccess = createAction(
  EUUserActions.GetUserInfoSuccess,
  props<{ payload: UserInfo }>()
);
export const LikeVideo = createAction(
  EUUserActions.LikeVideo,
  props<{ payload: { videoId: string } }>()
);
export const LikeVideoSuccess = createAction(
  EUUserActions.LikeVideoSuccess,
  props<{ payload: boolean }>()
);
export const UnLikeVideo = createAction(
  EUUserActions.UnLikeVideo,
  props<{ payload: { videoId: string } }>()
);
export const UnLikeVideoSuccess = createAction(
  EUUserActions.UnLikeVideoSuccess,
  props<{ payload: boolean }>()
);
export const DislikeVideo = createAction(
  EUUserActions.DislikeVideo,
  props<{ payload: { videoId: string } }>()
);
export const DislikeVideoSuccess = createAction(
  EUUserActions.DislikeVideoSuccess,
```

```
  props<{ payload: boolean }>()
);
export const UnDislikeVideo = createAction(
  EUUserActions.UnDislikeVideo,
  props<{ payload: { videoId: string } }>()
);
export const UnDislikeVideoSuccess = createAction(
  EUUserActions.UnDislikeVideoSuccess,
  props<{ payload: boolean }>()
);
export const AddVideoToHistory = createAction(
  EUUserActions.AddVideoToHistory,
  props<{ payload: { videoId: string } }>()
);
export const AddVideoToHistorySuccess = createAction(
  EUUserActions.AddVideoToHistorySuccess,
  props<{ payload: boolean }>()
);
export const UpdateSettings = createAction(
  EUUserActions.UpdateSettings,
  props<{ payload: { user: UserInfo } }>()
);
export const UpdateSettingsSuccess = createAction(
  EUUserActions.UpdateSettingsSuccess,
  props<{ payload: boolean }>()
);
export const UpdateSettingsFailed = createAction(
  EUUserActions.UpdateSettingsFailed,
  props<{ payload: boolean }>()
);
export const ChangeAvatar = createAction(
  EUUserActions.ChangeAvatar,
  props<{ payload: { img: File; name: string } }>()
);
export const ChangeAvatarSuccess = createAction(
  EUUserActions.ChangeAvatarSuccess,
  props<{ payload: string }>()
);
export const DeleteAvatar = createAction(EUUserActions.DeleteAvatar);
export const DeleteAvatarSuccess = createAction(
  EUUserActions.DeleteAvatarSuccess,
  props<{ payload: string }>()
);
export const PostComment = createAction(
  EUUserActions.PostComment,
  props<{ payload: { videoId: string; commentText: string } }>()
);
export const PostCommentSuccess = createAction(
  EUUserActions.PostCommentSuccess,
  props<{ payload: Comment[] }>()
);
```

```
export const GetMyCommentsByVideoId = createAction(
  EUUserActions.GetMyCommentsByVideoId,
  props<{ payload: { id: string } }>()
);
export const GetMyCommentsByVideoIdSuccess = createAction(
  EUUserActions.GetMyCommentsByVideoIdSuccess,
  props<{ payload: Comment[] }>()
);
export const GetMyCommentsByVideoIdError = createAction(
  EUUserActions.GetMyCommentsByVideoIdError
);
export const GetUserSettings = createAction(EUUserActions.GetUserSettings);
export const GetUserSettingsSuccess = createAction(
  EUUserActions.GetUserSettingsSuccess,
  props<{ payload: UserSettings }>()
);
export const GetUserInfoById = createAction(
  EUUserActions.GetUserInfoById,
  props<{ payload: { userId: string } }>()
);
export const GetUserInfoByIdSuccess = createAction(
  EUUserActions.GetUserInfoByIdSuccess,
  props<{ payload: UserInfo }>()
);
export const SendEmailToken = createAction(
  EUUserActions.SendEmailToken,
  props<{ payload: { userEmail: string } }>()
);
export const SendEmailTokenSuccess = createAction(
  EUUserActions.SendEmailTokenSuccess,
  props<{ payload: boolean }>()
);
export const PasswordResetToken = createAction(
  EUUserActions.PasswordResetToken,
  props<{ payload: { userEmail: string } }>()
);
export const PasswordResetTokenSuccess = createAction(
  EUUserActions.PasswordResetTokenSuccess,
  props<{ payload: boolean }>()
);
export const PasswordChange = createAction(
  EUUserActions.PasswordChange,
  props<{ payload: { userPassword: string; userId: string } }>()
);
export const PasswordChangeSuccess = createAction(
  EUUserActions.PasswordChangeSuccess,
  props<{ payload: boolean }>()
);
```

search-page-videos-actions.ts:

```
import { CommentList } from './../../models/comment';
import { Video, VideoList } from '../../models/standard-page-video';
import { createAction, props } from '@ngrx/store';

export enum EUVideoActions {
  GetVideos = '[Video] GET_VIDEOS',
  GetVideo = '[Video] GET_VIDEO',
  GetHotVideos = '[Video] GET_HOT_VIDEOS',
  GetVideosSuccess = '[Video] GET_VIDEOS_SUCCESS',
  GetVideosNextPage = '[Video] GET_VIDEOS_NEXT_PAGE',
  GetHotVideosNextPage = '[Video] GET_HOT_VIDEOS_NEXT_PAGE',
  GetVideoSuccess = '[Video] GET_VIDEO_SUCCESS',
  GetVideosByIds = '[Video] GET_VIDEOS_BY_IDS',
  GetVideosByIdsSuccess = '[Video] GET_VIDEOS_BY_IDS_SUCCESS',
  GetCommentsByVideoId = '[Video] GET_COMMENTS_BY_VIDEO_ID',
  GetCommentsByVideoIdSuccess = '[Video] GET_COMMENTS_BY_VIDEO_ID_SUCCESS',
  GetCommentsNextPage = '[Video] GET_COMMENTS_NEXT_PAGE',
}

export const GetVideos = createAction(
  EUVideoActions.GetVideos,
  props<{ payload: { videos_count_per_page: number; region: string } }>()
);
export const GetVideo = createAction(
  EUVideoActions.GetVideo,
  props<{ payload: { id: string } }>()
);
export const GetHotVideos = createAction(EUVideoActions.GetHotVideos);
export const GetVideosNextPage = createAction(
  EUVideoActions.GetVideosNextPage,
  props<{
    payload: {
      videos_count_per_page: number;
      region: string;
      pageToken: string;
    };
  }>()
);
export const GetHotVideosNextPage = createAction(
  EUVideoActions.GetHotVideosNextPage,
  props<{ payload: { pageToken: string } }>()
);
export const GetVideosSuccess = createAction(
  EUVideoActions.GetVideosSuccess,
  props<{ payload: VideoList }>()
);
export const GetVideoSuccess = createAction(
  EUVideoActions.GetVideoSuccess,
  props<{ payload: Video }>()
```

```
);
export const GetVideosByIds = createAction(
  EUVideoActions.GetVideosByIds,
  props<{ payload: { id: string[] } }>()
);
export const GetVideosByIdsSuccess = createAction(
  EUVideoActions.GetVideosByIdsSuccess,
  props<{ payload: Video[] }>()
);
export const GetCommentsByVideoId = createAction(
  EUVideoActions.GetCommentsByVideoId,
  props<{ payload: { id: string } }>()
);
export const GetCommentsByVideoIdSuccess = createAction(
  EUVideoActions.GetCommentsByVideoIdSuccess,
  props<{ payload: CommentList }>()
);
export const GetCommentsNextPage = createAction(
  EUVideoActions.GetCommentsNextPage,
  props<{ payload: { id: string; token: string } }>()
);
```

user-effects.ts:

```
import { Injectable } from '@angular/core';
import { ofType, Actions, createEffect } from '@ngrx/effects';
import {
  catchError,
  map,
  mergeMap,
  switchMap,
  withLatestFrom,
} from 'rxjs/operators';
import { EMPTY, of } from 'rxjs';
import { AuthService } from 'src/app/services/user-auth-service/user-auth-service';
import { Store } from '@ngrx/store';
import { AppState } from '../state/app.state';
import { selectUserInfo } from '../selectors/user-selectors';
import * as UserActions from '../actions/user-actions';
import { Router } from '@angular/router';
import { RoutesPaths } from 'src/app/consts-routes-paths';
import { LogServiceComponent } from 'src/app/shared/log-service/log-service';
import { UserInfo, UserSettings } from 'src/app/models/user-info';
import { Comment } from '../../models/comment';
@Injectable()
export class UserEffects {
  constructor(
    private userService: AuthService,
    private actions$: Actions,
    private store: Store<AppState>,
    private route: Router,
```

```
  private logger: LogServiceComponent
) {}

getUserInfo$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.GetUserInfo),
    switchMap(() =>
      this.userService.getUserInfo(this.userService.getCurrentToken()).pipe(
        map((user: UserInfo) =>
          UserActions.GetUserInfoSuccess({ payload: user })
        ),
        catchError((error) => {
          this.logger.error('[USER_EFFECT]getUserInfo', error);
          this.route.navigate([RoutesPaths.LoginRoute]);
          return of(UserActions.Logout());
        })
      )
    )
  );
});

getUserSettings$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.GetUserSettings),
    switchMap(() =>
      this.userService
        .getUserSettings(this.userService.getCurrentToken())
        .pipe(
          map((settings: UserSettings) =>
            UserActions.GetUserSettingsSuccess({ payload: settings })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]getUserSettings', error);
            return EMPTY;
          })
        )
    )
  );
});

SignUp$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.SignUp),
    switchMap((action) =>
      this.userService.SignUp(action.payload.user).pipe(
        map((isLogin: boolean) =>
          UserActions.SignUpSuccess({ payload: isLogin })
        ),
        catchError((error) => {
          this.logger.error('[USER_EFFECT]SignUp', error);
          return of(UserActions.SignUpError({ payload: error }));
```

```
        })
      )
    )
  );
});

Login$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.Login),
    switchMap((action) =>
      this.userService.Login(action.payload.user).pipe(
        map((isLogin: boolean) =>
          UserActions.LoginSuccess({ payload: isLogin })
        ),
        catchError((error) => {
          this.logger.error('[USER_EFFECT]Login', error);
          return of(UserActions.LoginError({ payload: error }));
        })
      )
    )
  );
});

Logout$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.Logout),
    switchMap(() =>
      this.userService.Logout().pipe(
        map((isLogin: boolean) =>
          UserActions.LogoutSuccess({ payload: isLogin })
        ),
        catchError((error) => {
          this.logger.error('[USER_EFFECT]Logout', error);
          return EMPTY;
        })
      )
    )
  );
});

changeInfo$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.ChangeUserInfo),
    switchMap((action) =>
      this.userService
        .changeUserInfo(
          action.payload.user,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: boolean) =>
```

```
            UserActions.ChangeUserInfoSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]changeInfo', error);
            return of(UserActions.ChangeUserInfoFailed({ payload: error }));
          })
        )
    )
  );
});

likeVideo$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.LikeVideo),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([action, user]) =>
      this.userService
        .likeVideo(
          user.email,
          action.payload.videoId,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: boolean) =>
            UserActions.LikeVideoSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]likeVideo', error);
            return EMPTY;
          })
        )
    )
  );
});

unLikeVideo$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.UnLikeVideo),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([action, user]) =>
      this.userService
        .unLikeVideo(
          user.email,
          action.payload.videoId,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: boolean) =>
            UserActions.UnLikeVideoSuccess({ payload: data })
          ),
          catchError((error) => {
```

```
            this.logger.error('[USER_EFFECT]unLikeVideo', error);
            return EMPTY;
          })
        )
    )
  );
});

dislikeVideo$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.DislikeVideo),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([action, user]) =>
      this.userService
        .dislikeVideo(
          user.email,
          action.payload.videoId,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: boolean) =>
            UserActions.DislikeVideoSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]dislikeVideo', error);
            return EMPTY;
          })
        )
    )
  );
});

unDislikeVideo$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.UnDislikeVideo),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([action, user]) =>
      this.userService
        .unDislikeVideo(
          user.email,
          action.payload.videoId,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: boolean) =>
            UserActions.UnDislikeVideoSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]unDislikeVideo', error);
            return EMPTY;
          })
```

```
        )
    )
  );
});

addVideoToHistory$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.AddVideoToHistory),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([action, user]) =>
      this.userService
        .addVideoToHistory(
          user.email,
          action.payload.videoId,
          this.userService.getCurrentToken(),
          new Date().toISOString()
        )
        .pipe(
          map((data: boolean) =>
            UserActions.AddVideoToHistorySuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]addVideoToHistory', error);
            return EMPTY;
          })
        )
    )
  );
});

updateSettings$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.UpdateSettings),
    switchMap((action) =>
      this.userService
        .updateSettings(
          action.payload.user,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: boolean) =>
            UserActions.UpdateSettingsSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]updateSettings', error);
            return of(UserActions.UpdateSettingsFailed({ payload: error }));
          })
        )
    )
  );
});
```

```
updateAvatar$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.ChangeAvatar),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([action, user]) =>
      this.userService
        .uploadImgToCloud(
          user.email,
          action.payload.img,
          action.payload.name,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((url: string) =>
            UserActions.ChangeAvatarSuccess({ payload: url })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]updateAvatar', error);
            return EMPTY;
          })
        )
    )
  );
});

deleteAvatar$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.DeleteAvatar),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([, user]) =>
      this.userService
        .deleteImgFromCloud(
          user.email,
          user.avatarUrl.split('/api/img-upload/file/')[1],
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: string) =>
            UserActions.DeleteAvatarSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]deleteAvatar', error);
            return EMPTY;
          })
        )
    )
  );
});

postComment$ = createEffect(() => {
```

```
  return this.actions$.pipe(
    ofType(UserActions.PostComment),
    withLatestFrom(this.store.select(selectUserInfo)),
    switchMap(([action, user]) =>
      this.userService
        .uploadComment(
          action.payload.videoId,
          user.email,
          new Date().toISOString(),
          action.payload.commentText,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: Comment[]) =>
            UserActions.PostCommentSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]postComment', error);
            return EMPTY;
          })
        )
    )
  );
});

getMyCommentsByVideoId$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.GetMyCommentsByVideoId),
    mergeMap((action) =>
      this.userService
        .getMyVideoComments(
          action.payload.id,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((data: Comment[]) =>
            UserActions.GetMyCommentsByVideoIdSuccess({ payload: data })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]getMyCommentsByVideoId', error);
            return of(UserActions.GetMyCommentsByVideoIdError());
          })
        )
    )
  );
});

getUserInfoById$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.GetUserInfoById),
    mergeMap((action) =>
```

```
      this.userService
        .getUserInfoById(
          action.payload.userId,
          this.userService.getCurrentToken()
        )
        .pipe(
          map((user: UserInfo) =>
            UserActions.GetUserInfoByIdSuccess({ payload: user })
          ),
          catchError((error) => {
            this.logger.error('[USER_EFFECT]getUserInfoById', error);
            return EMPTY;
          })
        )
    )
  );
});

sentUserEmailTokenAgain$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.SendEmailToken),
    mergeMap((action) =>
      this.userService.SentEmailTokenAgain(action.payload.userEmail).pipe(
        map((isSend: boolean) =>
          UserActions.SendEmailTokenSuccess({ payload: isSend })
        ),
        catchError((error) => {
          this.logger.error('[USER_EFFECT]sentUserEmailTokenAgain', error);
          return EMPTY;
        })
      )
    )
  );
});

sentUserPasswordResetToken$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(UserActions.PasswordResetToken),
    mergeMap((action) =>
      this.userService.SentPasswordResetToken(action.payload.userEmail).pipe(
        map((isSend: boolean) =>
          UserActions.PasswordResetTokenSuccess({ payload: isSend })
        ),
        catchError((error) => {
          this.logger.error('[USER_EFFECT]sentUserPasswordResetToken', error);
          return EMPTY;
        })
      )
    )
  );
});
```

```
  sentUserNewPassword$ = createEffect(() => {
    return this.actions$.pipe(
      ofType(UserActions.PasswordChange),
      mergeMap((action) =>
        this.userService
          .ResetPassword(action.payload.userPassword, action.payload.userId)
          .pipe(
            map((isSend: boolean) =>
              UserActions.PasswordChangeSuccess({ payload: isSend })
            ),
            catchError((error) => {
              this.logger.error('[USER_EFFECT]sentUserNewPassword', error);
              return EMPTY;
            })
          )
      )
    );
  });
}
```

home-page-videos-effects.ts:

```
import { Injectable } from '@angular/core';
import { ofType, Actions, createEffect } from '@ngrx/effects';
import { catchError, map, mergeMap, switchMap } from 'rxjs/operators';
import { VideoService } from 'src/app/services/video-service/video-service';
import { EMPTY } from 'rxjs';
import { Video, VideoList } from 'src/app/models/standard-page-video';
import * as VideoActions from '../actions/home-page-videos-actions';
import { LogServiceComponent } from 'src/app/shared/log-service/log-service';
import { CommentList } from 'src/app/models/comment';
@Injectable()
export class VideoEffects {
  constructor(
    private videoService: VideoService,
    private actions$: Actions,
    private logger: LogServiceComponent
  ) {}

  getVideos$ = createEffect(() => {
    return this.actions$.pipe(
      ofType(VideoActions.GetVideos),
      mergeMap((action) =>
        this.videoService
          .getVideosForHomePage(
            action.payload.videos_count_per_page,
            action.payload.region
          )
          .pipe(
```

```
          map((videosList: VideoList) =>
            VideoActions.GetVideosSuccess({ payload: videosList })
          ),
          catchError((error) => {
            this.logger.error('[HOME_PAGE_VIDEO_EFFECT]getVideos', error);
            return EMPTY;
          })
        )
    )
  );
});

getHotVideos$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(VideoActions.GetHotVideos),
    mergeMap(() =>
      this.videoService.getVideosForTrendingPage().pipe(
        map((videosList: VideoList) =>
          VideoActions.GetVideosSuccess({ payload: videosList })
        ),
        catchError((error) => {
          this.logger.error('[HOME_PAGE_VIDEO_EFFECT]getHotVideos', error);
          return EMPTY;
        })
      )
    )
  );
});

getVideosNextPage$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(VideoActions.GetVideosNextPage),
    mergeMap((action) =>
      this.videoService
        .getNextPageVideosForHome(
          action.payload.videos_count_per_page,
          action.payload.region,
          action.payload.pageToken
        )
        .pipe(
          map((videosList: VideoList) =>
            VideoActions.GetVideosSuccess({ payload: videosList })
          ),
          catchError((error) => {
            this.logger.error(
              '[HOME_PAGE_VIDEO_EFFECT]getVideosNextPage',
              error
            );
            return EMPTY;
          })
        )
```

```
    )
  );
});

getHotVideosNextPage$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(VideoActions.GetHotVideosNextPage),
    mergeMap((action) =>
      this.videoService
        .getNextPageVideosForTrending(action.payload.pageToken)
        .pipe(
          map((videosList: VideoList) =>
            VideoActions.GetVideosSuccess({ payload: videosList })
          ),
          catchError((error) => {
            this.logger.error(
              '[HOME_PAGE_VIDEO_EFFECT]getHotVideosNextPage',
              error
            );
            return EMPTY;
          })
        )
    )
  );
});

getVideoInfo$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(VideoActions.GetVideo),
    switchMap((action) => {
      return this.videoService.getVideoInfo(action.payload.id).pipe(
        map((video: Video) =>
          VideoActions.GetVideoSuccess({ payload: video })
        ),
        catchError((error) => {
          this.logger.error('[HOME_PAGE_VIDEO_EFFECT]getVideoInfo', error);
          return EMPTY;
        })
      );
    })
  );
});

getVideosByIds$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(VideoActions.GetVideosByIds),
    mergeMap((action) =>
      this.videoService.getVideosInfo(action.payload.id).pipe(
        map((videos: Video[]) =>
          VideoActions.GetVideosByIdsSuccess({ payload: videos })
        ),
```

```
        catchError((error) => {
          this.logger.error('[HOME_PAGE_VIDEO_EFFECT]getVideosByIds', error);
          return EMPTY;
        })
      )
    )
  );
});

getCommentsByVideoId$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(VideoActions.GetCommentsByVideoId),
    mergeMap((action) =>
      this.videoService.getVideoComments(action.payload.id).pipe(
        map((commentsList: CommentList) =>
          VideoActions.GetCommentsByVideoIdSuccess({ payload: commentsList })
        ),
        catchError((error) => {
          this.logger.error(
            '[HOME_PAGE_VIDEO_EFFECT]getCommentsByVideoId',
            error
          );
          return EMPTY;
        })
      )
    )
  );
});

getCommentsByVideoIdNextPage$ = createEffect(() => {
  return this.actions$.pipe(
    ofType(VideoActions.GetCommentsNextPage),
    mergeMap((action) =>
      this.videoService
        .getVideoCommentsNextPage(action.payload.id, action.payload.token)
        .pipe(
          map((commentsList: CommentList) =>
            VideoActions.GetCommentsByVideoIdSuccess({
              payload: commentsList,
            })
          ),
          catchError((error) => {
            this.logger.error(
              '[HOME_PAGE_VIDEO_EFFECT]getCommentsByVideoIdNextPage',
              error
            );
            return EMPTY;
          })
        )
```

home-page.component.html:

```
<div class="container">
  <div class="pagination-container">
    <button
      debounceClick
      [debounceTime]="300"
      class="btn btn-left"
      *ngIf="prevPageToken$ | async as prevToken"
      (debounceClick)="nextPage(prevToken)"
    >
      <i class="fa fa-angle-left fa-5x" aria-hidden="true"></i>
    </button>
    <button
      debounceClick
      [debounceTime]="300"
      class="btn btn-right"
      *ngIf="nextPageToken$ | async as nextToken"
      (debounceClick)="nextPage(nextToken)"
    >
      <i class="fa fa-angle-right fa-5x" aria-hidden="true"></i>
    </button>
  </div>
  <div>
    <div class="posts">
      <app-home-page-video-block
        *ngFor="let video of videos$ | async; trackBy: identify"
        [video]="video"
      >
      </app-home-page-video-block>
    </div>
  </div>
</div>
```

search-page.component.html:

```
<div class="pagination-container">
  <button
    debounceClick
    [debounceTime]="300"
    class="btn btn-left"
    *ngIf="prevPageToken$ | async as prevToken"
    (debounceClick)="nextPage(prevToken)"
  >
    <i class="fa fa-angle-left fa-5x" aria-hidden="true"></i>
  </button>
  <button
    debounceClick
    [debounceTime]="300"
    class="btn btn-right"
    *ngIf="nextPageToken$ | async as nextToken"
```

```
    (debounceClick)="nextPage(nextToken)"
  >
    <i class="fa fa-angle-right fa-5x" aria-hidden="true"></i>
  </button>
</div>
<div class="search-title-page">Search Results</div>
<div class="videos-container">
  <app-search-page-video-block
    *ngFor="let video of videos$ | async; trackBy: identify"
    [video]="video"
  >
  </app-search-page-video-block>
</div>
```

video-page.component.html:

```
<div class="container">
  <div class="video">
    <div class="player">
      <app-video-player [video]="video$ | async"></app-video-player>
      <app-video-description [video]="video$ | async"></app-video-description>
      <form
        *showWhenUserLogin="true"
        [formGroup]="commentForm"
        (ngSubmit)="onSubmit()"
        class="comment-field"
      >
        <textarea
          formControlName="commentText"
          class="comment-input"
          placeholder="left a comment..."
        ></textarea>
        <button
          *ngIf="!commentForm.invalid"
          class="btn-sent-comment"
          [disabled]="commentForm.invalid"
        >
          Sent
        </button>
      </form>
      <div class="comments-container">
        <app-video-comment
          [ifUser]="true"
          *ngFor="let comment of myComments$ | async"
          [comment]="comment"
        >
        </app-video-comment>
      </div>
      <div class="comments-container">
        <app-video-comment
          *ngFor="let comment of comments$ | async"
```

```
          [comment]="comment"
        >
        </app-video-comment>
        <div class="pagination-container">
          <button
            class="btn btn-left"
            *ngIf="prevPageToken$ | async as prevToken"
            (click)="nextPage(prevToken)"
          >
            <i class="fa fa-angle-left fa-3x" aria-hidden="true"></i>
          </button>
          <button
            class="btn btn-right"
            *ngIf="nextPageToken$ | async as nextToken"
            (click)="nextPage(nextToken)"
          >
            <i class="fa fa-angle-right fa-3x" aria-hidden="true"></i>
          </button>
        </div>
      </div>
    </div>
  </div>
  <div class="videos-list">
    <app-channel-page-video-block
      *ngFor="let video of videos$ | async; trackBy: identify"
      [video]="video"
    >
    </app-channel-page-video-block>
  </div>
</div>
```

video-page.component.scss:

```
@import '/src/styles';

:host {
  .container {
    display: flex;
    flex-direction: row;
    justify-content: center;
    margin-top: 3vh;
  }

  .comments-container {
    margin: 0 auto;
  }

  .comment-field {
    display: flex;
    flex-direction: row;
    align-items: flex-end;
```

```
  .comment-input {
    width: 100%;
    padding: 1em 0 0 1em;
    border: none;
    border-bottom: 1px solid var(--text_basic_color);
    outline: none;
    background: none;
    color: var(--text_basic_color);
    font-family: Roboto, 'Helvetica Neue', sans-serif;
    resize: none;
    transition: 0.5s;

    &::-webkit-input-placeholder {
      color: var(--bg_main-lighter_color);
      font-size: 14px;
      letter-spacing: 0.02em;
      line-height: 21px;
    }

    &:focus {
      border-bottom: 1px solid var(--main_color);

      &::-webkit-input-placeholder {
        opacity: 0.2;
      }
    }
  }
}

.videos-list {
  margin-left: 2vw;
  padding: 0 1vw;
  border-right: 1px solid var(--bg_main-lighter_color);
  border-left: 1px solid var(--bg_main-lighter_color);
  border-radius: 10px;
}

.pagination-container {
  display: flex;
  flex-direction: row-reverse;
  align-items: center;
  justify-content: space-between;

  .btn {
    border: none;
    outline: none;
    background: none;
    color: var(--text_basic_color);
    opacity: 50%;
```

```
    &:hover {
      cursor: pointer;
      opacity: 100%;
      transition: 0.6s;
    }
  }
}

.btn-sent-comment {
  height: 50%;
  margin-left: 1em;
  padding: 0.6em 1em;
  border: 1px solid black;
  border-radius: 2px;
  background: #000;
  color: var(--text_basic_color);
  color: var(--text_light_color);
  font-size: 14px;
  font-style: normal;
  font-weight: normal;
  opacity: 0.6;

  &:hover {
    border: 1px solid var(--text_basic_color);
    opacity: 1;
  }

  &:active {
    transform: translateY(3px);
    transition: ease-in-out 0.1s;
  }
}
@media (max-width: 1600px) {
  .player {
    width: 950px;
  }
}
@media (max-width: 1370px) {
  .player {
    width: 98vw;
  }

  .container {
    flex-direction: column;
    margin: 5em 0.3em 0 0.3em;
  }

  .videos-list {
    display: none;
  }
}
```

```
}
```

styles.scss:

```
@import
url("https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700&display=
swap");
@import
url("https://fonts.googleapis.com/css2?family=Open+Sans:wght@400;600&display=swap")
;

:root {
  --main_color: #27f9a3a1;
  --main-dark_color: rgb(5 135 136 / 80%);
  --main-darker_color: #0a615a;
  --text_basic_color: rgba(255, 255, 255, 0.733);
  --bg_main_color: #282828;
  --bg_main-light_color: #404040;
  --bg_main-lighter_color: #606060;
  --text_light_color: rgba(255, 255, 255, 0.733);
  --card_bg_gradient: linear-gradient(
    258.97deg,
    #27f9a3a1 0.88%,
    rgba(1, 50, 0, 0) 100%
  );
}
* {
  box-sizing: border-box;
  padding: 0;
  margin: 0;
}

html,
body {
  height: 93vh;
  margin-top: 7vh;
}
body {
  margin: 0;
  padding: 0;
  font-family: Roboto, "Helvetica Neue", sans-serif;
  background-color: var(--bg_main_color);
```

ДОДАТОК Г (обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДІВ І ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ КОРИСТУВАЧІВ ТА ЇХ КОМУНІКАЦІЇ ДЛЯ ВІДЕО СЕРВІСУ

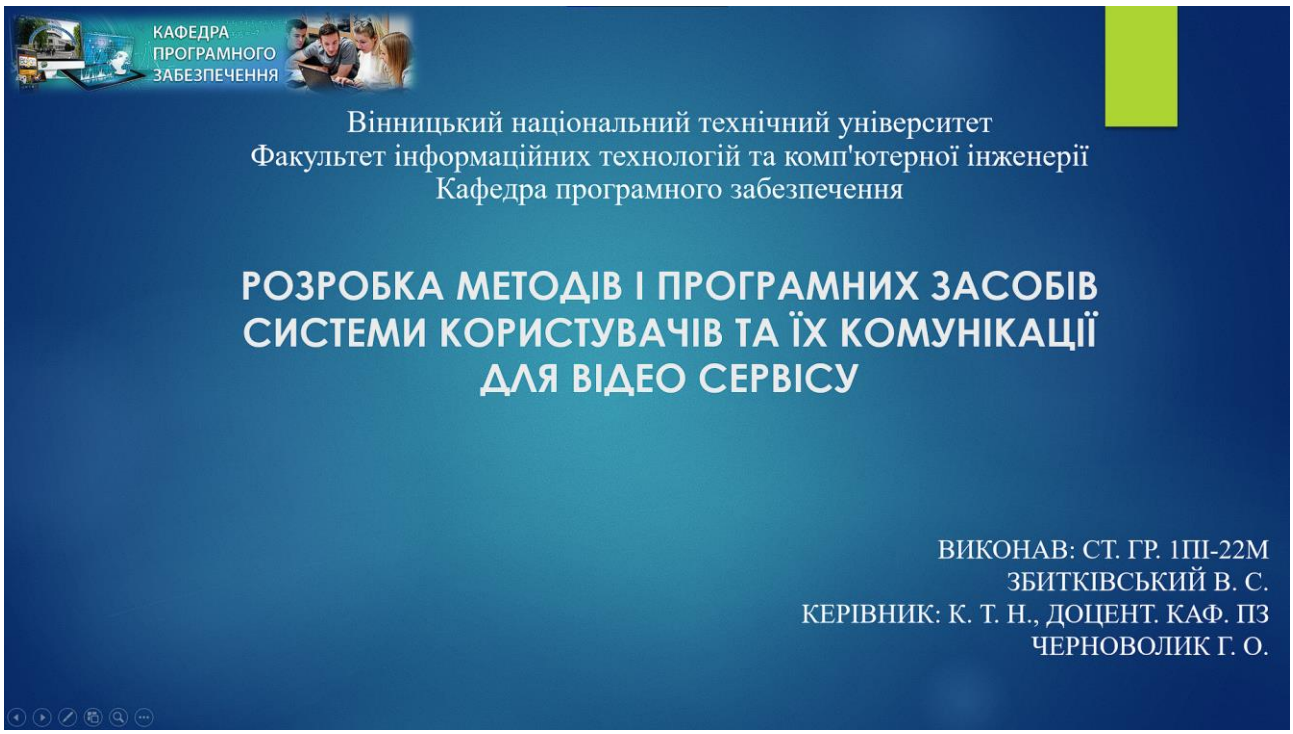

Рисунок Г.1 – Слайд презентації №1

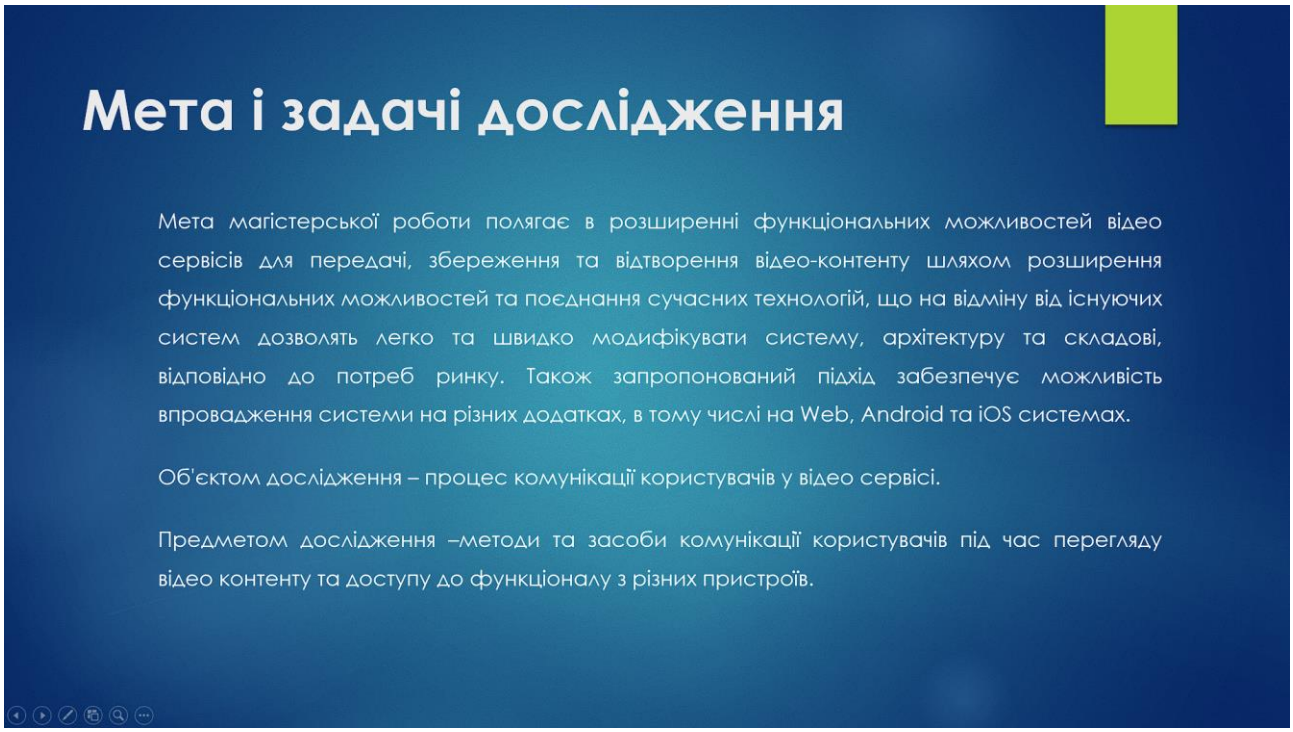

Рисунок Г.2 – Слайд презентації №2

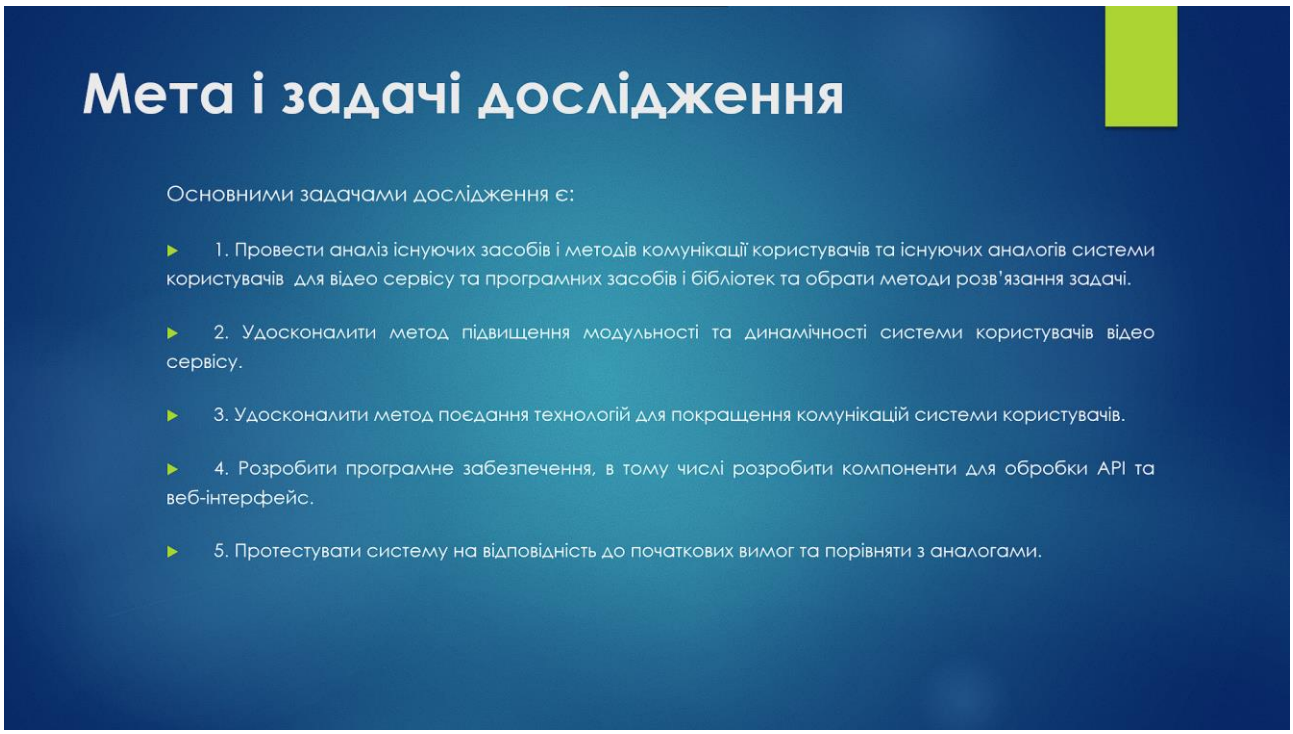

Рисунок Г.3 – Слайд презентації №3

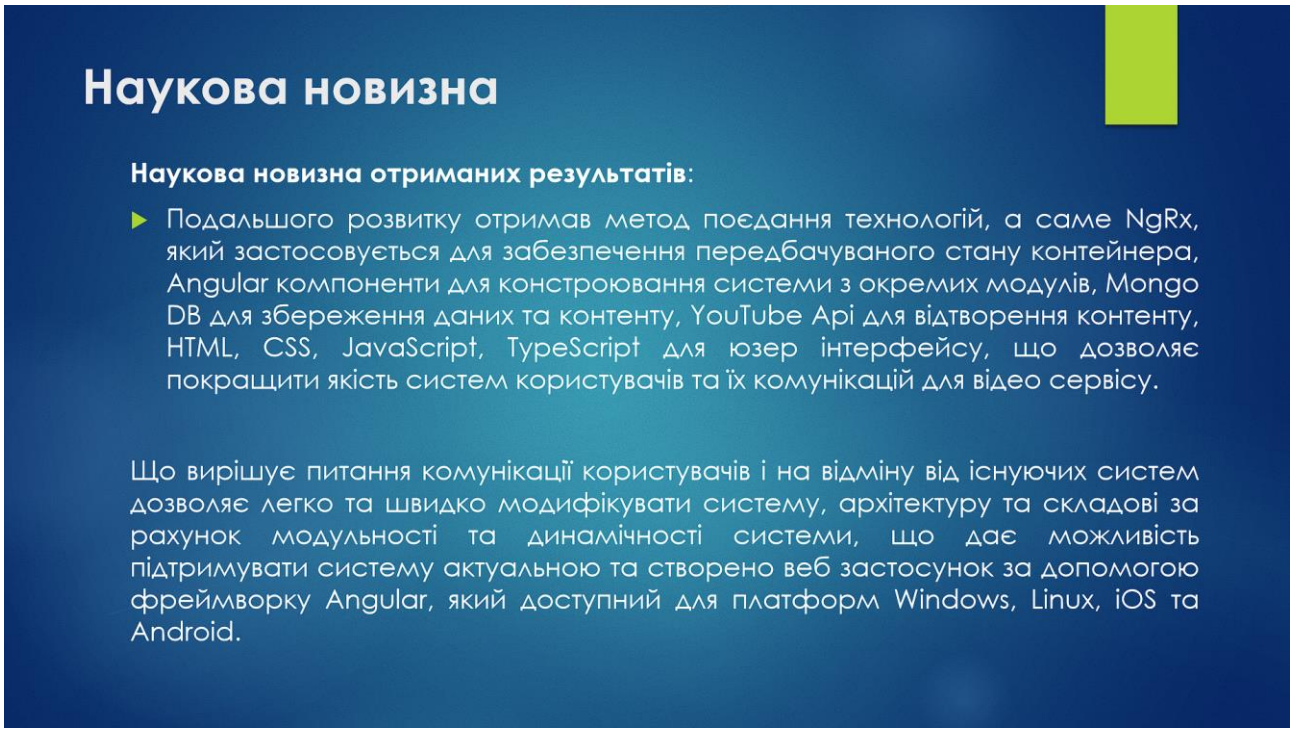

Рисунок Г.4 – Слайд презентації №4

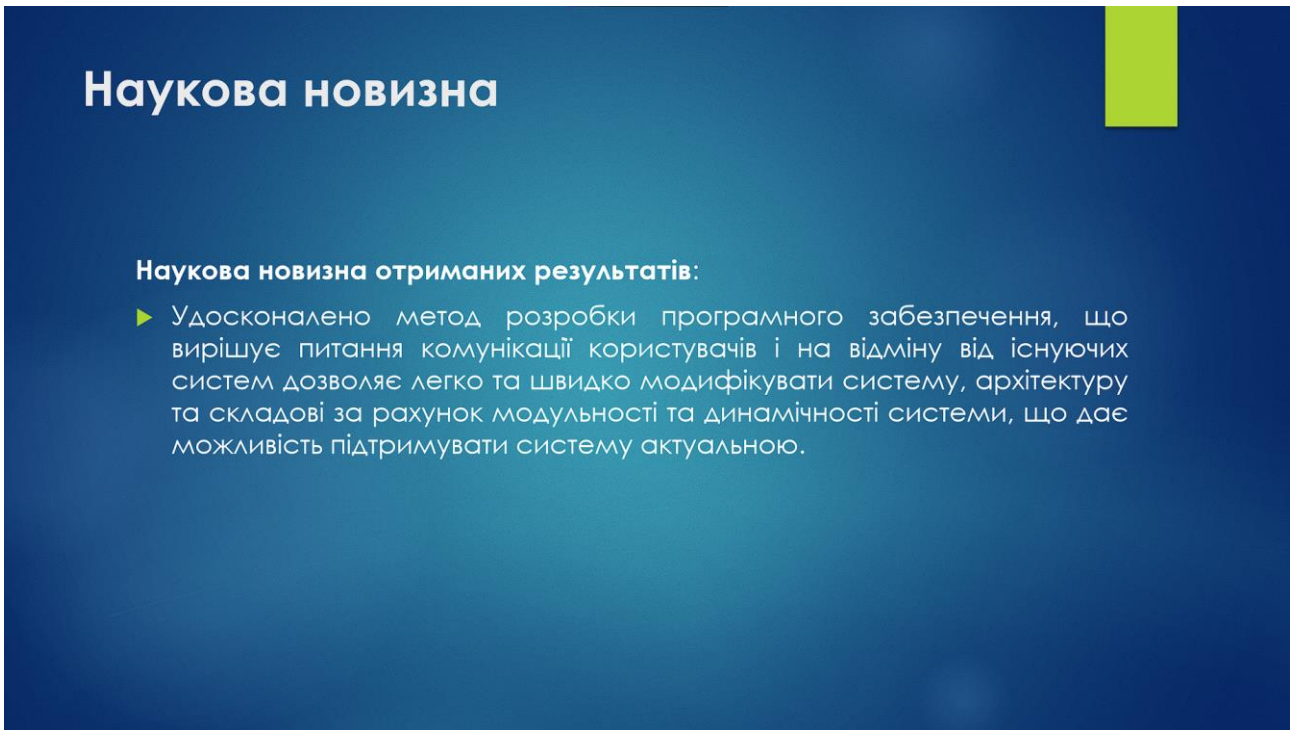

Рисунок Г.5 – Слайд презентації №5

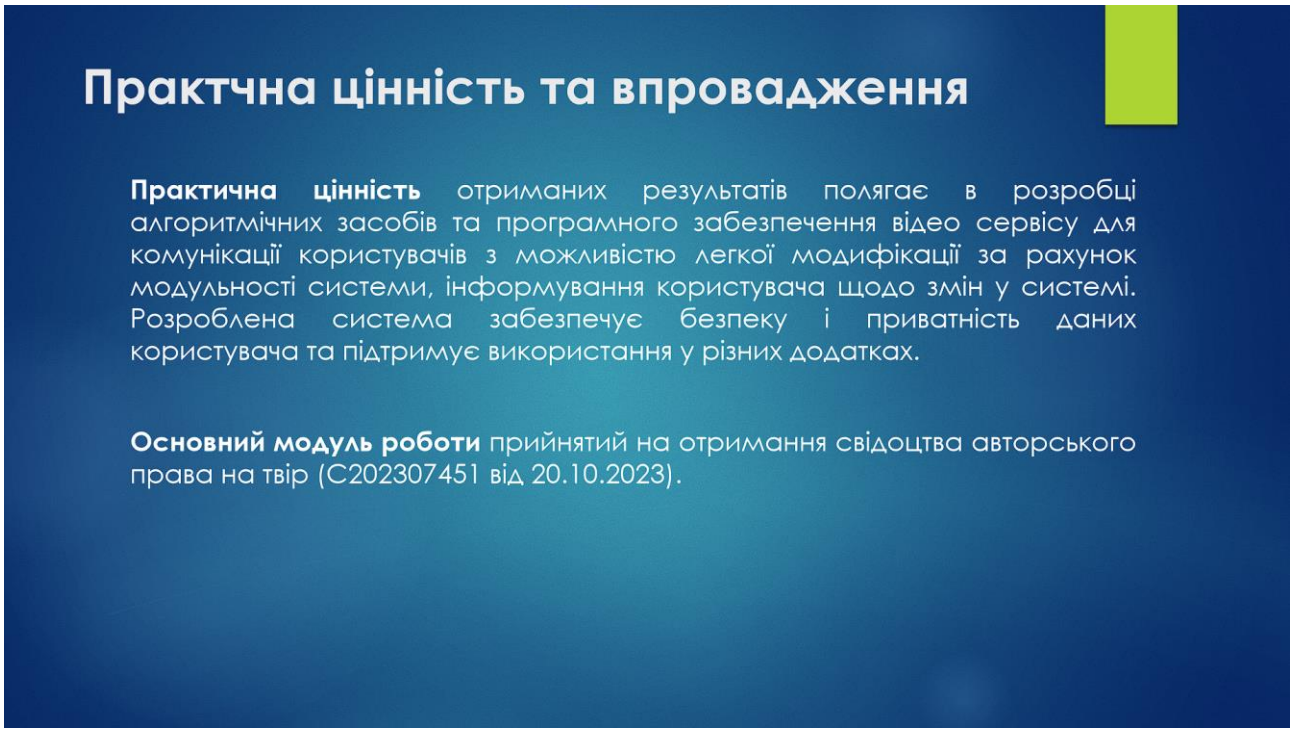

Рисунок Г.6 – Слайд презентації №6

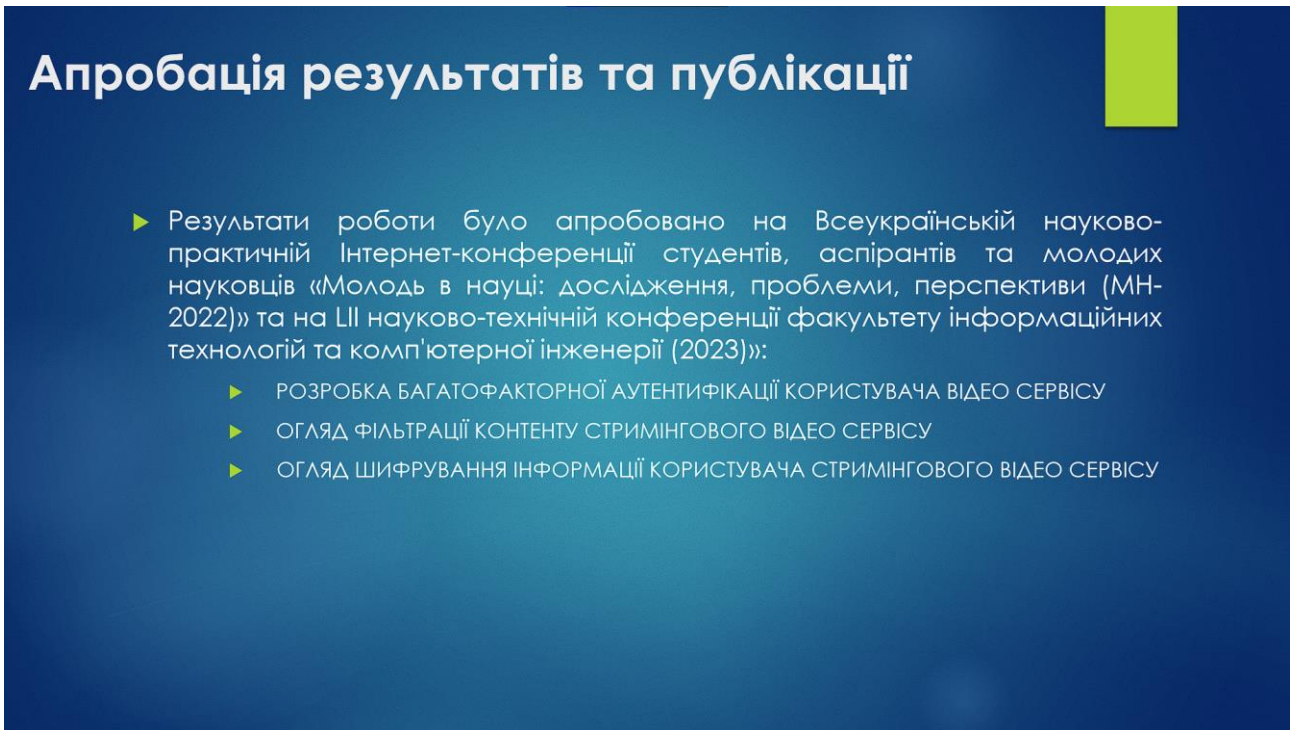

Рисунок Г.7 – Слайд презентації №7

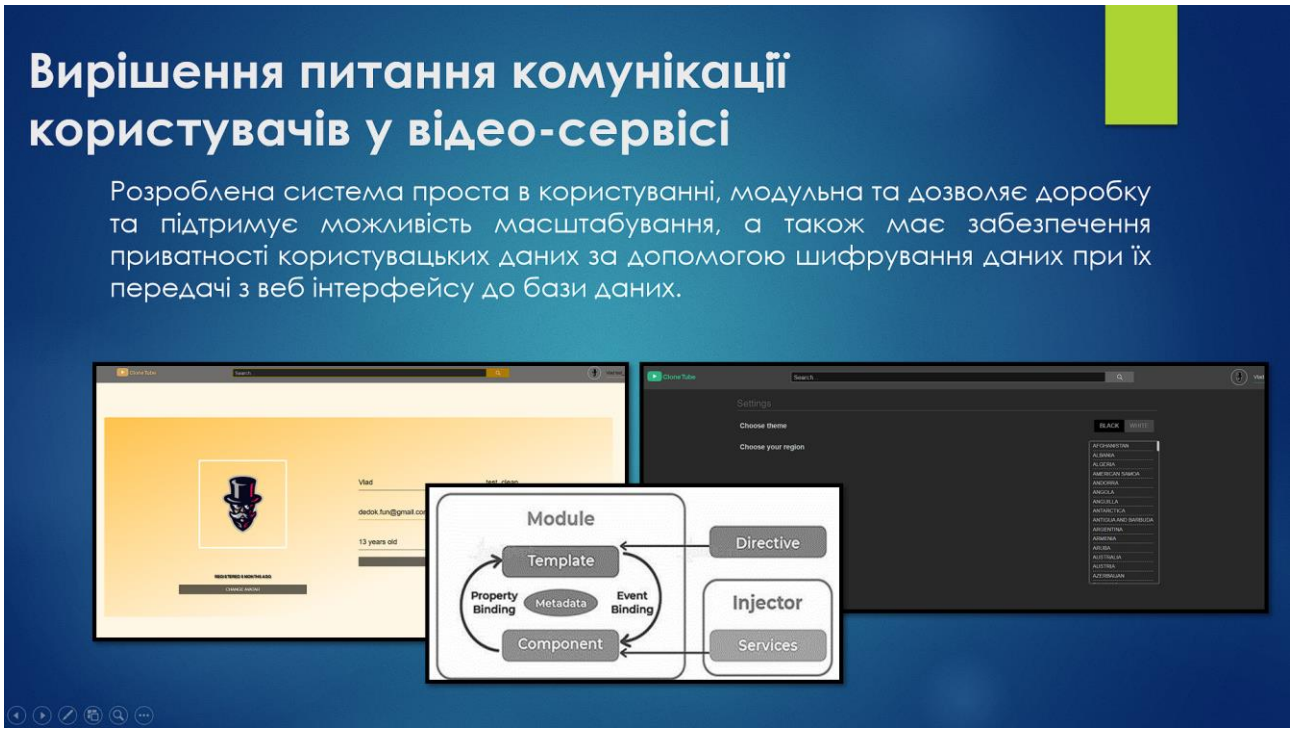

Рисунок Г.8 – Слайд презентації №8

Рисунок Г.9 – Слайд презентації №9

Рисунок Г.10 – Слайд презентації №10

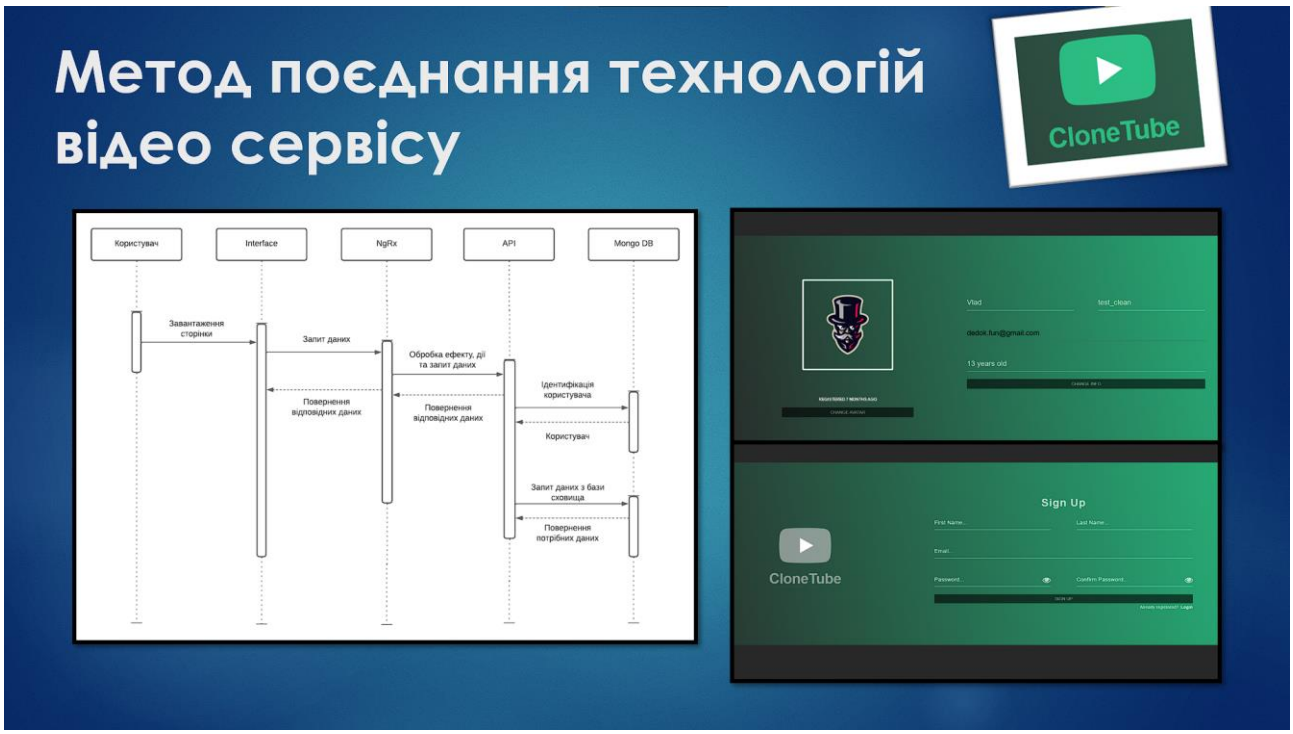

Рисунок Г.11 – Слайд презентації №11

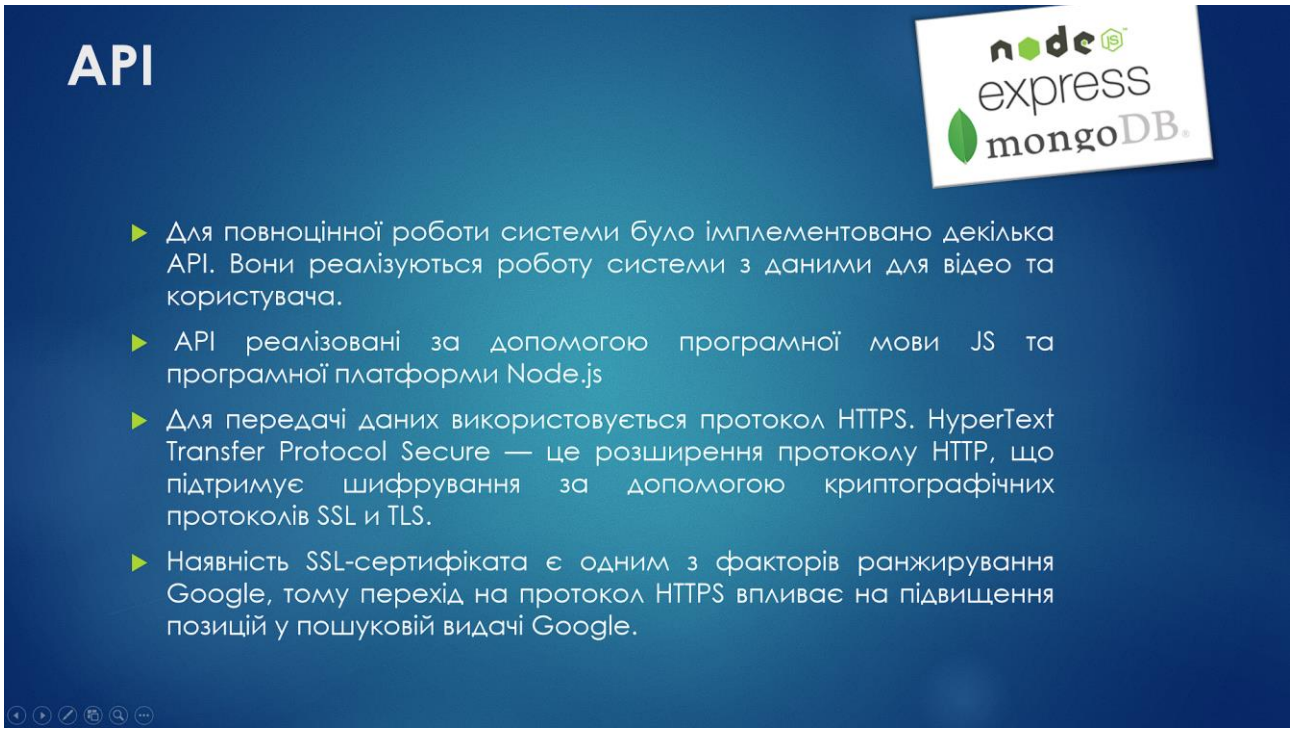

Рисунок Г.12 – Слайд презентації №12

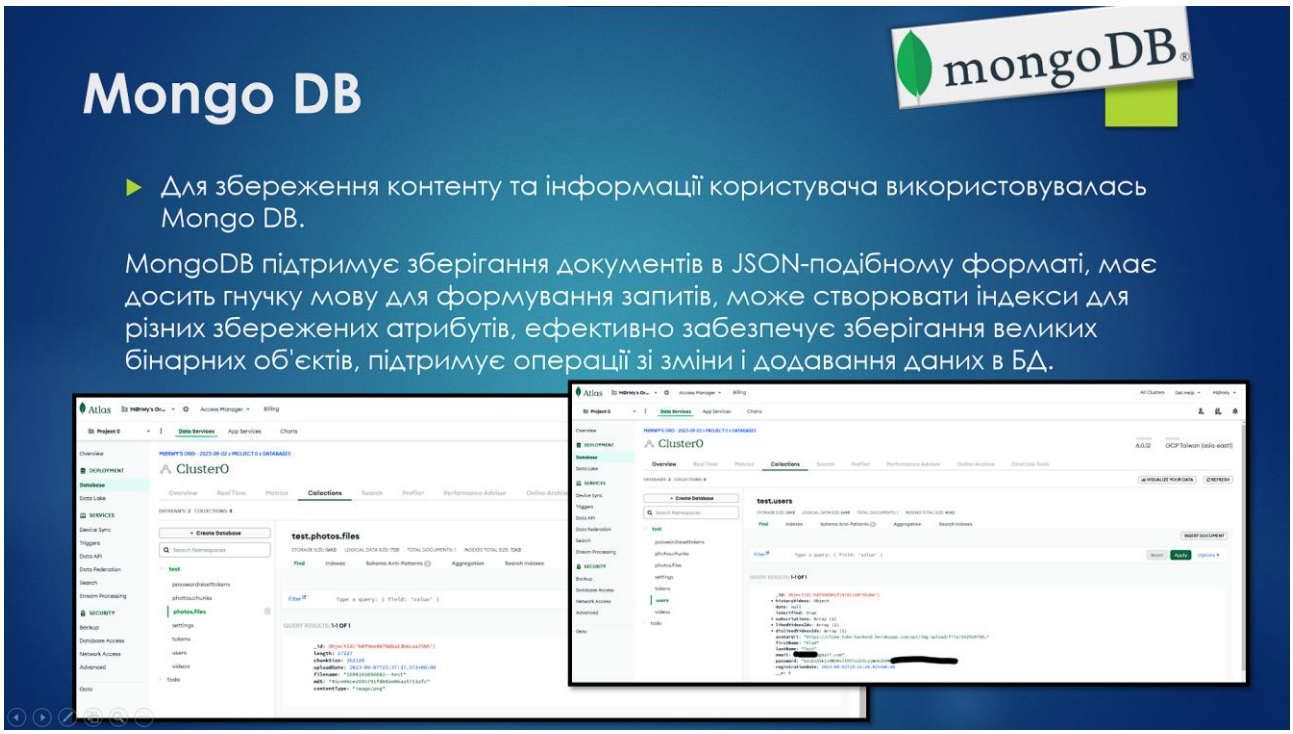

Рисунок Г.13 – Слайд презентації 13

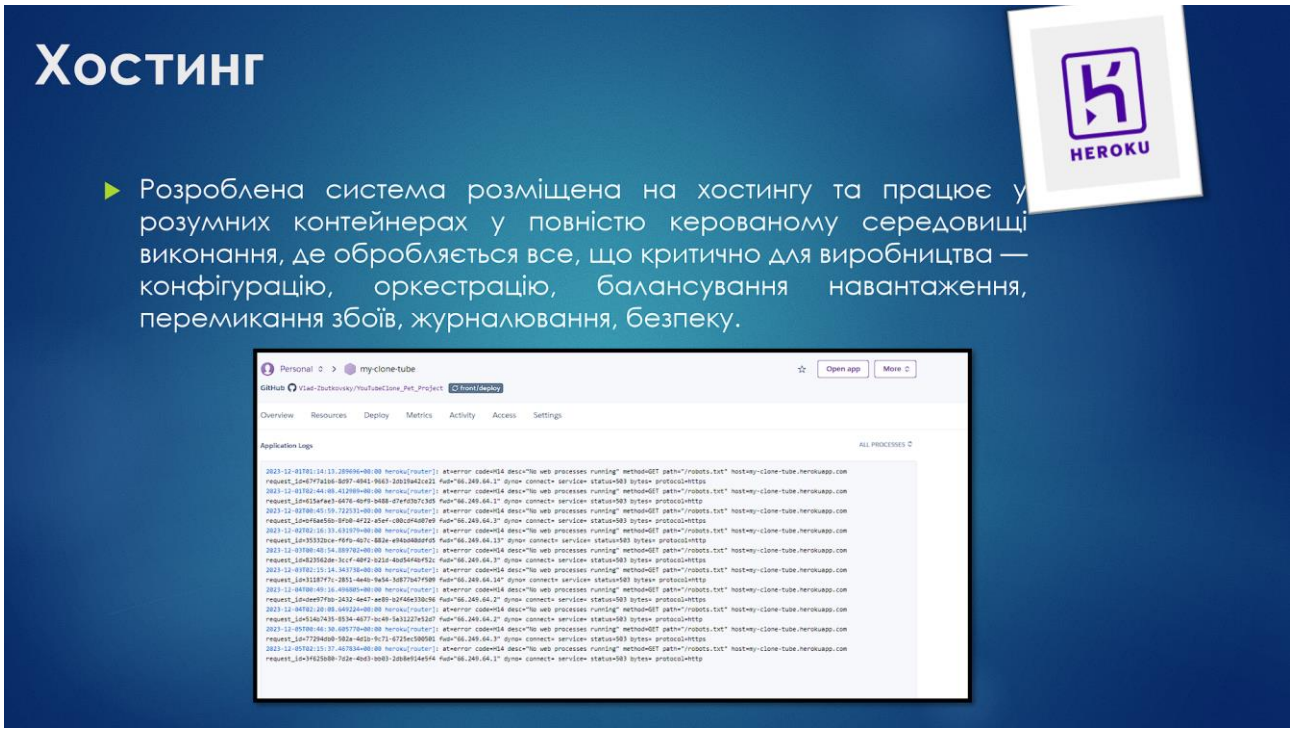

Рисунок Г.14 – Слайд презентації №14

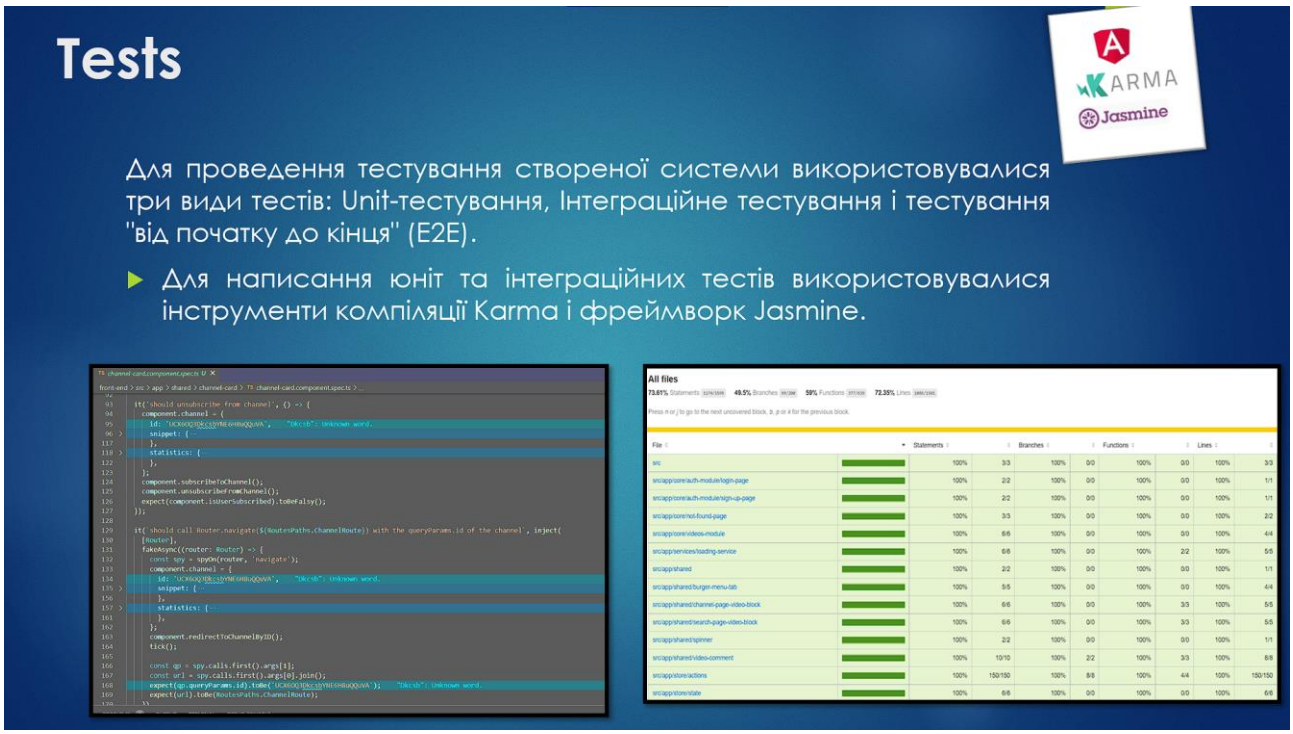

Рисунок Г.15 – Слайд презентації №15

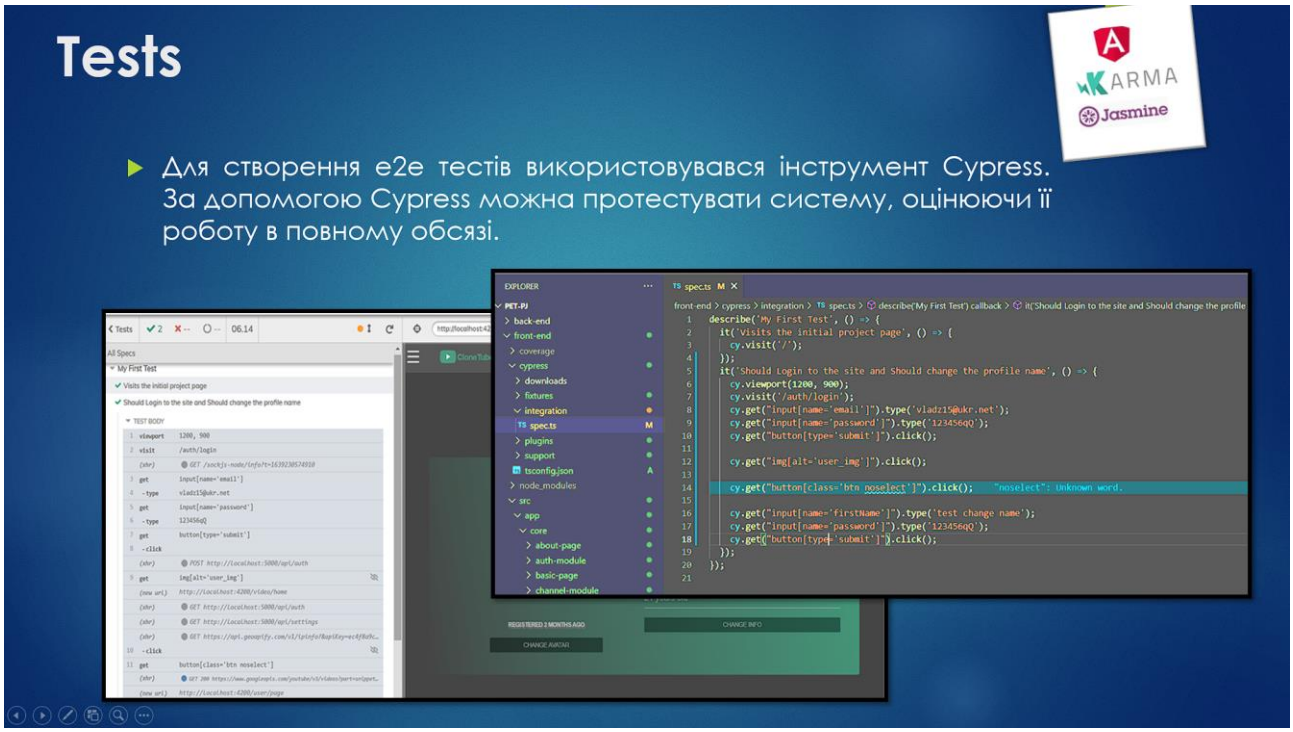

Рисунок Г.16 – Слайд презентації №16

Характеристики системи	Розроблена система користувачів та їх комунікації	Система користувачів та їх комунікації YouTube
Маштабованість системи	+	-
Рівень контролю даних користувачем	+	+
Можливості фільтрації контенту	+	+
Використання ресурсів інтернет з'єднання	+	-
Використання ресурсів пристрою	+	-
Наявний функціонал системи профілю користувачів	+	-
Оповіщення користувачів системи	+	+

Рисунок Г.17 – Слайд презентації №17

Рисунок Г.18– Слайд презентації №18

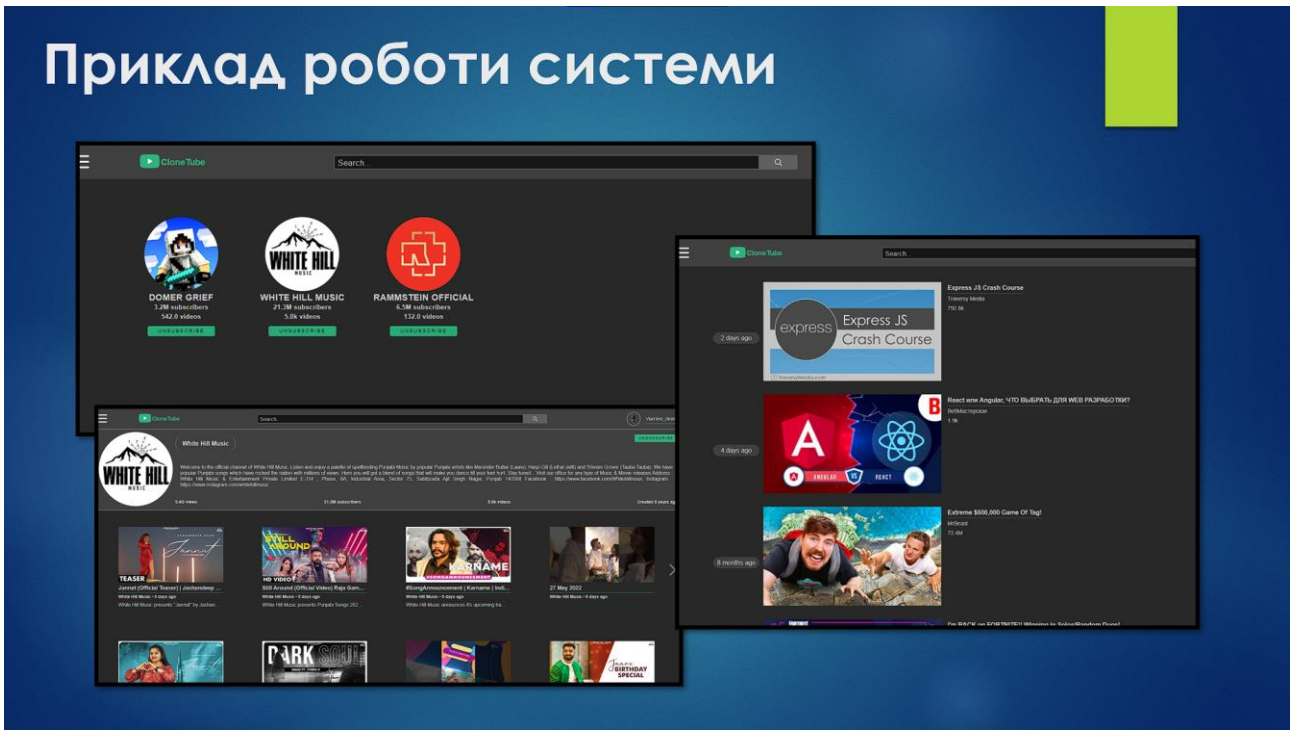

Рисунок Г.19 – Слайд презентації №19

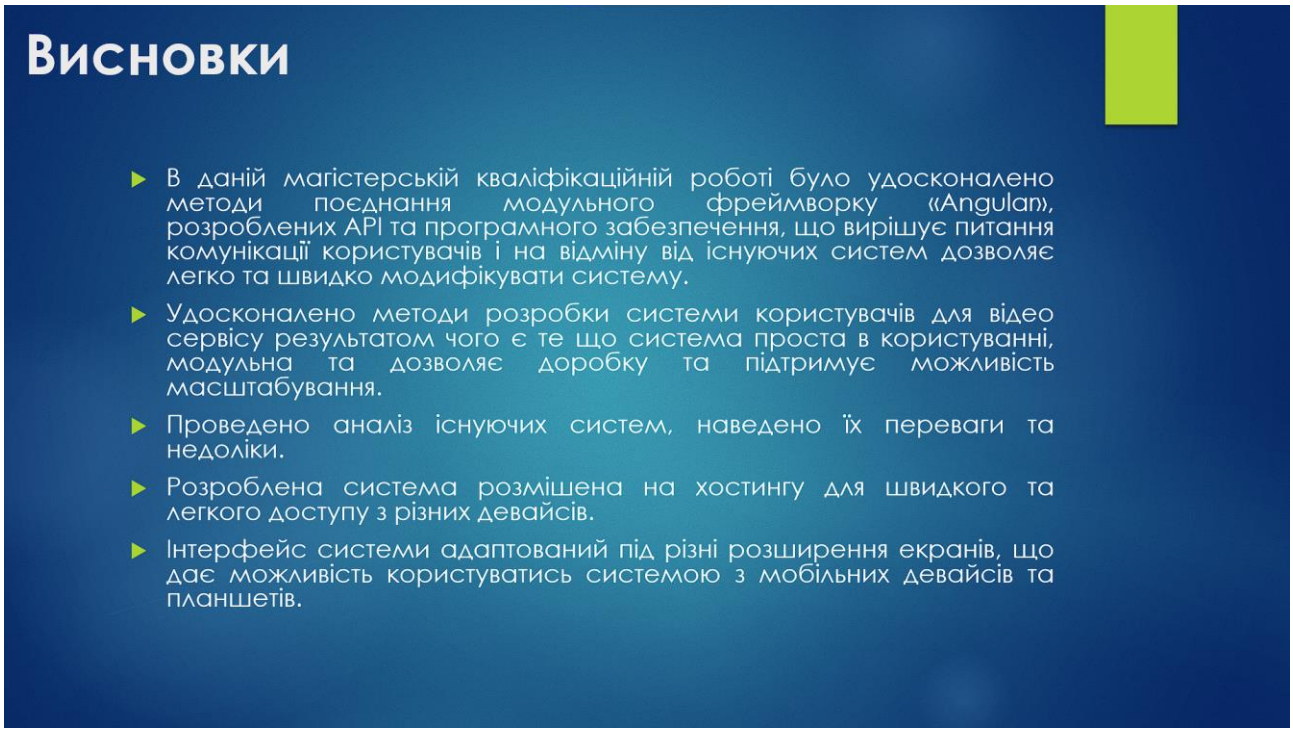

Рисунок Г.20 – Слайд презентації №20

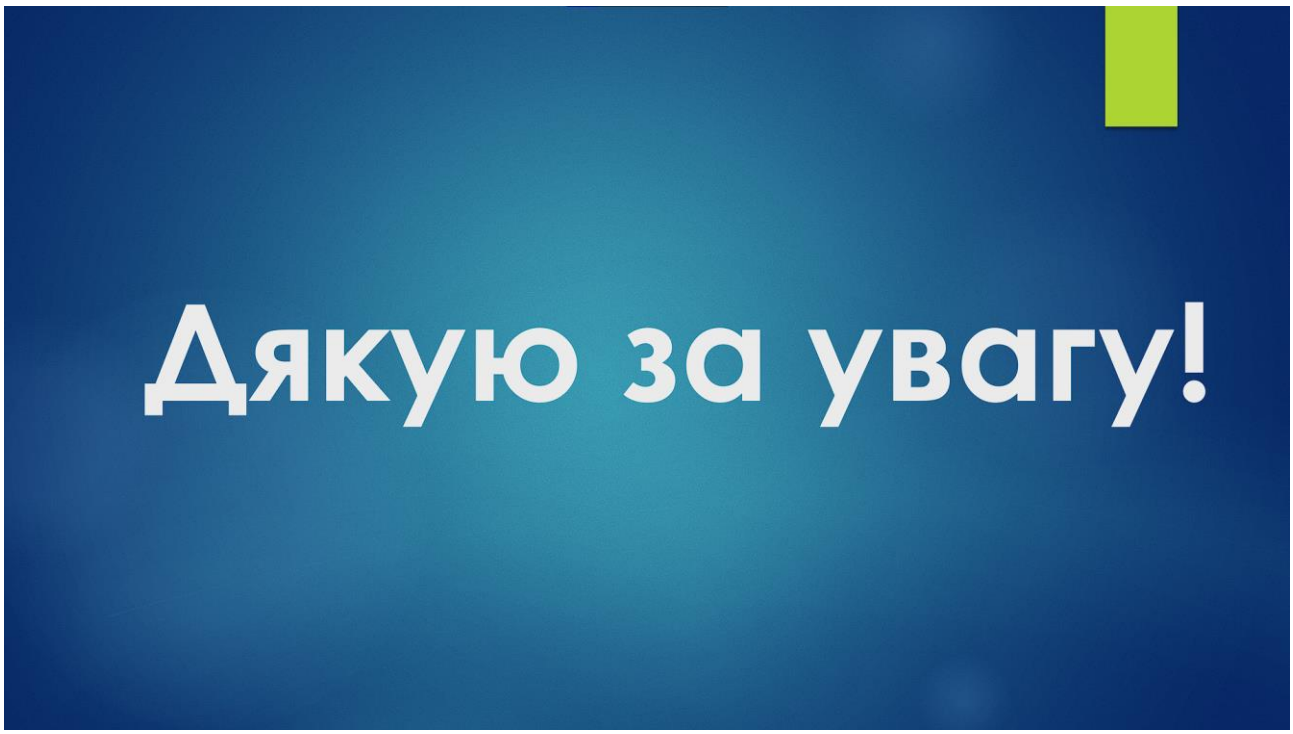

Рисунок Г.21 – Слайд презентації №21