Міністерство освіти і науки України

Криворізький національний університет

Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня віщої освіти магістра

зі спеціальності 121 — Інженерія програмного забезпечення

На тему: Дослідження та розробка системи управління субтитрами з використанням штучного інтелекту

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів
без відповідних посилань.
Студент групи ІПЗ-23-1м
Гордієнко Владислав

Керівник

кваліфікаційної роботи _____ А. М. Стрюк

Завідувач кафедри _____ А. М. Стрюк

Кривий Ріг

2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: моделювання та програмного забезпечення

Ступінь вищої освіти: магістр

Спеціальність: 121 — Інженерія програмного забезпечення

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи

1. Тема: дослідження та розробка системи управління субтитрами з використанням штучного інтелекту
затверджено наказом по КНУ №TODO від «TODO» TODO 2024 р.

2. Термін подання студентом закінченої роботи: «TODO» TODO 2024 р.

3. Вихідні дані по роботі: розроблена система повинна надавати користувачам можливість автоматичного генерувати субтитри та інші можливості роботи організації субтитрів.

4. Зміст пояснювальної записки (перелік питань, що треба розробити): проаналізувати існуючі аналоги, розробити систему управління субтитрами, перевірити розроблену систему.

5. Перелік ілюстративного матеріалу: зображення аналогів, функціональна схема, блок-схеми розроблених алгоритмів, інтерфейс системи, ER-діаграма бази даних.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Ознайомлення з джерелами за темою дослідження	01.08.2024 - 14.08.2024
2	Дослідження існуючих систем	15.08.2024 - 19.08.2024
3	Аналіз розробки методів системи субтитрів	20.08.2024 - 26.08.2024
4	Аналіз предметної області та визначення вимог до системи	27.08.2024 - 04.09.2024
5	Підготовка матеріалів для першого розділу роботи	05.09.2024 - 15.09.2024
6	Розробка блок-схем та діаграм програмного забезпечення	16.09.2024 - 21.09.2024
7	Підготовка матеріалів для другого розділу роботи	22.09.2024 - 30.09.2024
8	Реалізація програмного модуля системи	01.10.2024 - 08.10.2024
9	Створення користувацького інтерфейсу	09.10.2024 - 15.10.2024
10	Підготовка матеріалів третього розділу роботи	16.10.2024 - 24.10.2024
11	Оформлення пояснювальної записки	25.10.2024 - 31.10.2024

Дата видачі завдання «TODO» TODO 2024 р.

Студент _____ / В.С. Гордієнко /

Керівник роботи _____ / А. М. Стрюк /

РЕФЕРАТ

Ключові слова: СУБТИТРИ, ШТУЧНИЙ ІНТЕЛЕКТ.

Кваліфікаційна робота складається з 122 сторінок, вона має 1 додаток і список посилань на використані джерела. У роботі є 38 рисунків і 14 таблиць.

Мета кваліфікаційної роботи – дослідження та розробка системи управління субтитрами з використанням штучного інтелекту.

В результаті аналізу і дослідження різних джерел та аналогів, були визначені задачі, які потрібно вирішити для успішної реалізації системи управління субтитрами з штучним інтелектом.

У процесі проектування були визначені різні функції системи та створені алгоритми їх реалізацій. Крім того, був розроблений інтерфейс програми, який оптимально відповідає заданому функціоналу системи.

У процесі реалізації було проведено аналіз технологій розробки веб-додатків, різних мов програмування та бібліотек. На основі цього аналізу було обрано найбільш оптимальні технології для розробки цієї системи.

Система управління субтитрами з використанням штучного інтелекту була розроблена в результаті цієї роботи.

Система дає користувачам можливість ефективно і просто керувати своєю колекцією субтитрів. Це означає, що користувач може додавати субтитри до свого улюбленого фільму або серіалу. Також він буде мати можливість видалити старі субтитри або замінити їх. Дуже цікавою функцією є пошук у субтитрах. Користувач зможе зробити пошук по тексту і знайти в яких фільмах або серіалах зустрічається вказаний текст.

Також користувач буде мати можливість автоматично генерувати субтитри за допомогою штучного інтелекту. Автоматична генерація субтитрів дозволить користувачу отримати субтитри навіть для не дуже популярних фільмів або серіалів.

ABSTRACT

Key words: SUBTITLES, ARTIFICIAL INTELLIGENCE.

The qualification work consists of 122 pages, includes 1 appendix, and has a list of references. The work contains 38 figures and 14 tables.

The goal of the qualification work is research and development of a subtitle management system using artificial intelligence.

As a result of analyzing and researching various sources, the tasks necessary for a successful subtitle management system were identified.

During the design process, the various functions of the system were defined, and algorithms for their implementation were created. Additionally, an interface was developed that was optimized for the system functionality.

During the implementation process, an analysis was conducted of web application development technologies, various programming languages and libraries. The most optimal technologies for developing this system were selected based on this analysis.

The AI-powered subtitle management system was developed as a result of this work.

The system allows users to efficiently and easily manage their subtitles. This means that the user can add subtitles to their favorite movies and TV series. Naturally, it also means that the user can delete old subtitles and replace them with new ones. A very interesting feature is the subtitle search function. The user will be able to search for a specific line and find which movies or series contain that line.

Additionally, the user will have the ability to automatically generate subtitles using the artificial intelligence feature. Automatic subtitles generation will allow users to obtain subtitles even for less popular movies or TV series.

ЗМІСТ

ВСТУП

Люди дивляться велику кількість різних фільмів та серіалів з усіх куточків світу. Існує багато різноманіття різних жанрів та видів історій, які можна знайти на екрані кінотеатру або телевізора. Майже кожна країна має власну продукцію контенту і частину інтернаціонального контенту, що прийшов з інших країн. Деякі люди надають перевагу дубляжу, а інші вважають, що субтитри є кращим варіантом.

Кількість людей, котрі надають перевагу субтитрам є настільки великою, що субтитри стали великою індустрією. Майже кожен фільм, котрий з'являться в кінотеатрах, згодом отримує субтитри на багатьох різних мовах. Існують навіть сеанси в кінотеатрах, де фільми показують не в дубляжу, а з субтитрами. Схожу долю мають і серіали. В наш час, більшість серіалів, які виходять на різних сервісах, одразу мають велику кількість субтитрів на різних мовах.

Проблема полягає в тому, що більшість програм та сервісів не мають фокусу на субтитрах. Вони мають деякий функціонал субтитрів, але він є лише додатковим функціоналом в додачу до основного функціоналу перегляду фільмів або серіалів. Головний фокус і більшість ресурсів йде на розробку функціоналу перегляду відео чи аудіо контенту, а вже потім залишки ресурсів йдуть на роботу з функціоналом субтитрів. Звісно, попит на перегляд контенту є набагато більшим ніж попит на субтитри, тому компанії та розробники витрачають свої ресурси на те, що потрібно більшій кількості людей, але це не змінює того факту, що це створює деяку нішу в медіа просторі, котра потребує більше уваги та ресурсів.

Ця робота фокусується на дослідженні та розробці системи управління субтитрами за допомогою штучного інтелекту, яка буде мати головний фокус на функціоналі субтитрів. Штучний інтелект буде використано для того, щоб дати можливість користувачам автоматично генерувати субтитри для майже будь-якого відео-файлу. Користувачі можуть зробити швидкий пошук тексту в субтитрах та знайти те, що вони шукали.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми кваліфікаційної роботи

Фільми є невід’ємною частиною сучасного життя. Кожного року люди ходять до кінотеатрів, щоб побачити свої улюблені фільми [1]. Вони обговорюють їх з друзями, діляться своїми думками в різних соціальних мережах. Це може бути історія про стародавній Єгипет або наукова фантастика про далекі галактики. Завдяки стрімкому розвитку технологій, з кожним новим роком фільми дивують глядачів інноваціями в галузі спеціальних ефективній, котрі стають все більш реалістичними.

Але інновації стосуються не лише великих екранів, а і методів дистрибуції контенту. В наш час фільми може дивитися майже на будь-якому екрані, будь то смартфон, планшет, ноутбук або комп’ютер [2]. Фільми дивляться люди з усіх куточків нашої планети.

Існує багато способів, щоб люди з різних країн розуміли, що відбувається на екрані. Багато людей полюбляють переглядати фільми в дубляжі, але також є ті, котрі надають перевагу субтитрам [3].

Субтитри дозволять переглянути фільм в оригінальній формі та отримати максимальне задоволення від перегляду, але при цьому все ж таки зрозуміти, що в ньому відбувається. Також вони є незамінним інструментом для людей з порушеннями слуху, забезпечуючи їм доступ до улюбленого контенту.

Кількість контенту зростає майже кожного року, а з ним і потреба в якісних і доступних субтитрах. Це особливо актуально в сучасному тренді глобалізації, коли фільми та серіали поширюються по всьому світу. З цих причин, система управління субтитрами з можливістю їх автоматичного генерування за допомогою штучного інтелекту є дуже важливою та актуальною в наш час.

Серіали, на відміну від фільмів, здебільшого завжди знаходили собі місце на малих екранах. Люди не ходять в кінотеатри, щоб подивитися нову серію свого улюбленого серіалу, але це не означає, що вони є менш важливими.

Серіали стали важливою частиною сучасного повсякденного життя, тому, що вони дають можливість зануритися в історії, які розгортаються плавно, поступово, дозволяючи глядачам глибше переживати події. Глядачі стежать за персонажами і розвитком подій на протязі багатьох сезонів та серій.

Вони мають особливий тип взаємодії з аудиторією, тому, що глядачі регулярно повертаються до світу серіалу та його персонажів, подій. Вони переживають разом з своїми персонажами весь спектр емоцій.

Через те, що на відміну від фільмів, серіали можуть мати багато різних сезонів і серій, знайти якийсь специфічний момент стає складніше. Фільм може тривати максимум декілька годин, а серіали можуть тривати десятки, і навіть сотні годин. Складність знаходження якогось конкретного моменту сильно зростає. Що, якщо ти хочеш показати друзям дуже цікавий момент з твого улюбленого серіалу, але ти не пам’ятаєш точну серію та час? Якщо це був би фільм, то можна було б витратити декілька годин і просто переглянути фільм, але з серіалом це зовсім інші розрахунки. Виникає актуальність пошуку конкретного тексту в субтитрах усіх серій серіалу. Система управління субтитрами з використанням штучного інтелекту буде мати такий функціонал і дозволить знайти будь-який момент з серіалу дуже швидко.

Також, не потрібно забувати про цінність цього функціоналу у процесі вивчення іноземних мов [4]. Звісно, можна просто вивчити переклад будь-якого слова в словнику, але більш ефективно буде вивчити слово у цікавому контексті. Наприклад, спробувати знайти дане слово у одній з серій свого улюбленого серіалу та вивчити його через такий знайомий контекст. Це дозволяє, побачити як використовуються різні слова або словосполучення в

реальному світі. Також, це підвищує інтерес до вивчення у людини, бо вона не просто вивчає значення, як робот, а ще має справу з мистецтвом та культурою.

Через велику кількість сезонів та серій стає складніше організовувати файли субтитрів. Ця проблема вирішується за допомогою автоматичної організації файлів в системі. Всі файли субтитрів будуть прив’язані до конкретної серії в серіалі. Це означає, що користувачу не потрібно буде постійно шукати де знаходиться той чи інший файл. Йому потрібно буде просто вибрати конкретну серію в системі, а система автоматично знайде всі файли субтитрів, котрі підходять до цієї серії цього серіалу.

В сучасному світі є велика кількість різних сервісів перегляду фільмів та серіалів, у кожного з них є свої переваги та недоліки. Однією з найбільших різниць для користувачів є різноманіття фільмів та серіалів. Один сервіс може мати одну колекцію фільмів, а інший може не мати такої колекції, але може мати іншу унікальну колекцію. Це є дуже актуальною проблемою для користувачів, які бажають не бути обмежені в колекції фільмів і серіалів, а хочуть мати доступ до максимальної кількості фільмів і серіалів.

У зв’язку з цим, система управління субтитрами, яке не має таких обмежень, стає ще більш актуальною. Система управління субтитрами дозволяє користувачам додати будь-який фільм або серіал та завантажити до нього будь-які субтитри. Користувачі отримують більше свободи у виборі контенту, який вони хочуть переглядати та зможуть легко адаптувати його для своєї зручності, додавши субтитри на потрібній мові. Такий функціонал допомагає усунути обмеження пов’язані з різними сервісами, і дозволяє користувачам створювати свою власну колекцію фільмів і серіалів.

Дуже актуальною проблемою також є стабільність цих сервісів. Навіть, якщо зараз є ті речі, які потрібно користувачу, але це не означає, що вони там будуть завжди. Система управління субтитрами буде мати можливість локально встановити її на комп’ютер, на відміну від більшості популярних сервісів в наш час. Це означає, що всі файли та дані буду знаходитися на

комп’ютері користувача, тому тільки користувач буде мати змогу видалити ці дані. Це дає користувачу впевненість в майбутньому, що його дані знаходяться в безпеці, це дає йому свободу. Користувач знає, що якщо він завантажив інформацію до системи сьогодні, то вона буде там і завтра.

Більшість сервісів в наш час беруть фокус на медійну частину контенту, будь то відео чи аудіо файли. Майже вся їхня увага та фокус йдуть на вирішення проблем з такими видами контенту, а інші види контенту здебільшого мають лише другорядну важливість. Це означає, що майже всі ресурси дизайну, проектування, реалізації та підтримки проекту йдуть на вирішення питань про роботу з найпопулярнішими видами контенту, будь то відео чи аудіо файли. Всі інші не дуже популярні типи даних отримують лише маленький залишок тих ресурсів, що залишились, а в деяких випадках взагалі не отримують ніяких ресурсів. Звісно, з одного боку це зрозуміло, що найпопулярніші типи контенту отримують найбільше уваги, адже на це є найбільший попит. Але з іншого боку це означає, що після цього всього залишається велика ніша в типах контенту на які звертають увагу розробники. Субтитри, хоча й можуть здаватися менш популярними ніж відео та аудіо файли, але це не означає, що вони не мають ніякої користі і не потребують уваги та ресурсів.

Субтитри мають багато різних застосувань. Доступність для людей з вадами слуху є важливою частиною того, що роблять субтитри. Субтитри допомагають з вивченням різних мов, розвивати інтерес до різних мов та культур. Існує велика кількість людей, які надають перевагу субтитрам, а не дубляжу. Цьому є багато причин, але однією з найголовніших є те, що це дає людям можливість спожити контент, який є більш близьким до оригінального бачення автору. Зазвичай дубляж робить інші команда в іншій країні, а не та, що створила контент. Актори дубляжу мають свої власні переваги, але при дубляжу майже завжди виникають деякі розбіжності з оригіналом, це може бути голос, інтонація, або будь-які інші речі пов'язані з дубляжем.

1.2 Критичний аналіз літературних джерел за темою

Netflix – один з найпопулярніших сайтів в Інтернеті [5].

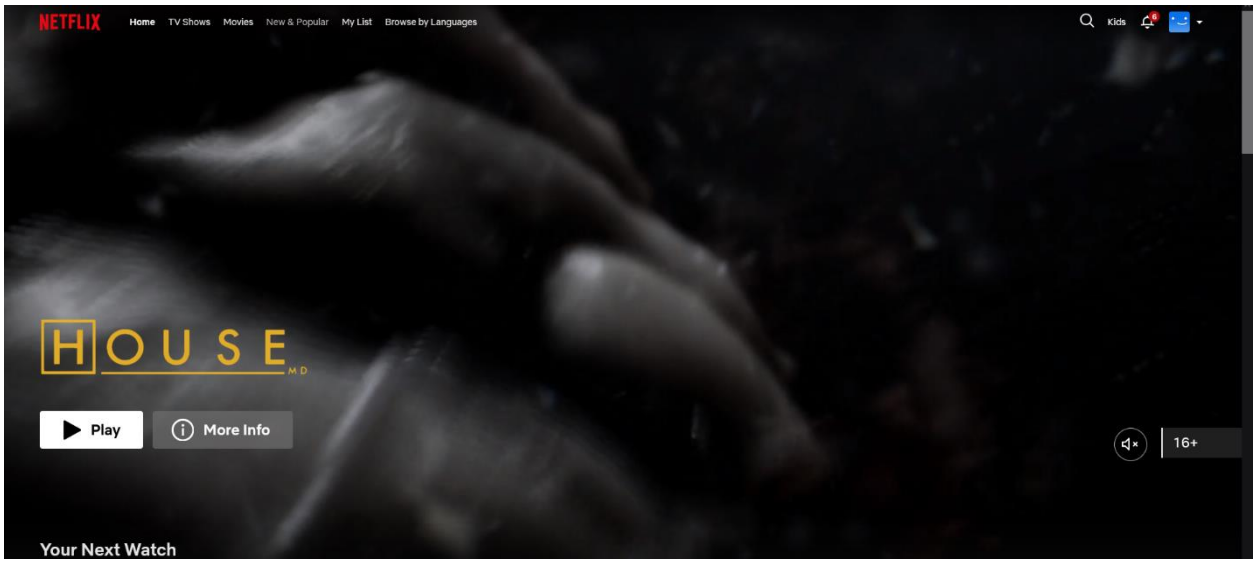

Рисунок 1.1– Головна сторінка Netflix

Netflix має великий список переваг та недоліків.

Дизайн сайту зроблений дуже ефективно, все виглядає дуже добре і зрозуміло. Технічна частина також зроблена дуже добре. Під час використання сервісу не з'являться ніяких незрозумілих технічних помилок. Все відео файли завантажуються досить швидко з серверів Netflix, тому швидкість завантаження здебільшого залежить від швидкості Інтернету у користувача, а не проміжною можливості серверів Netflix.

Netflix коштує грошей, щоб його використовувати потрібно платити кожного місяця. Багато користувачів звикли до безкоштовних програм або до програм, які ти купуєш один раз і потім можеш використовувати безмежно, але Netflix працює не так.

Чому Netflix працює так? Існує багато причин, але однією з найголовніших причин є те, що сервіс орендує велику кількість фільмів та серіалів у інших провайдерів.

Вибір контенту на сервісі обмежений тим, що Netflix орендує лише деякі фільми і серіали. Звісно, вибір контент на сервісі є дуже великим, але він не є безмежним. Існують навіть регіональні ліміти. Netflix може мати

права на показ фільму в одній країні, але не мати цих прав в інший країні, тому користувачі в тій іншій країні не зможуть переглянути цей контент.

Також через контракти про оренду, серіали та фільми можуть, як з’являтися на сервісі, так і зникати з нього, коли час оренди того чи іншого контенту на сервісі закінчується [6]. Коли час контракту на право показу контенту закінчується, Netflix вирішує долю контенту на основі багатьох різних речей. Спочатку потрібно дізнатися чи хоче інша компанія взагалі продовжувати ліцензувати цей контент для Netflix, чи може вони створили власний сервіс, який буде конкурувати в цій індустрії і їх більше не цікавлять контракти ліцензування контенту. Велика кількість популярних медіа компаній в наш мають власні сервіси перегляду контенту, тому їх не завжди цікавлять контракти ліцензування. У разі, якщо компанію цікавить продовження контракту, то Netflix також бере до увагу ціну продовження контракту та популярність контенту у регіоні. Якщо контент не є дуже популярним і його ціна є дуже великою, то скоріш за все Netflix не буде продовжувати цей контракт. Користувачі не можуть точно знати, коли їх улюблений фільми або серіалі зникнуть з Netflix. Повідомлення про те, що контент зникне з платформи зазвичай з’являються лише десь за один місяць до кінця контакту. Якщо користувач почне дивитися серіал з великою кількістю різних серій, то є шанс того, що серіал може зникнути з сервісу до того, як користувач перегляне все серії. Навіть, якщо на початку перегляду не буде повідомлення про те, що серіал зникне з платформи через певний час, то це повідомлення може з’явитися на будь-якому етапі перегляду. Користувачі не мають можливості самостійно додавати нові фільми або серіали до сервісу, таку можливість має лише адміністрація сервісу на основі різних контрактів та ліцензій.

Фокус сервісу повністю зосереджено на фільмах та серіалах. Netflix має субтитри, але вони працюють лише в додачу до відео файлів. Сервіс немає можливості додати власні субтитри, можна використовувати лише ті, які вже

є на сайті. Сервіс не має пошуку по тексту у субтитрах, або автоматичної генерації субтитрів за допомогою штучного інтелекту.

Можна сказати, що деякі речі Netflix робить добре, будь то простота використання або швидкість завантаження відео, але, якщо користувач хоче більше свободи, стабільності вибору контенту, фокусу на субтитрах, то це є не найкращим варіантом.

1.3 Цілі та завдання кваліфікаційної роботи

Метою даної кваліфікаційної роботи є дослідження та розробка системи управління субтитрами з використанням штучного інтелекту.

Система буде мати 3 основних типи користувачів: гість, авторизований користувач, адміністратор.

Для реалізації взаємодії гостів потрібно розробити:

- перегляд інформації про фільм або серіал.
- пошук тексту у субтитрах.
- перегляд субтитрів.
- реєстрація користувачів
- авторизація користувачів.

Для авторизованих користувачів буде доступний ті ж функції, що і для гостей, але в додачу ще буде:

- автоматична генерація субтитрів за допомогою штучного інтелекту.
- управління своїм обліковим записом.
- управління своїм списками контенту.
- управління інформацією про контент.

Для адміністраторів користувачів буде доступний ті ж функції, що і для авторизованих користувачів, але в додачу ще буде:

- управління обліковими записами користувачів.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Актуальність теми кваліфікаційної роботи

Розглянемо структуру схеми (рис 2.1) і функціонал. Схема складається з 3 розділів, що залежать від типу авторизації користувача.

У разі, якщо користувач не увійшов до власного акаунту у системі, він вважається гостем. Цей тип користувача має доступ до наступного функціоналу:

- перегляд інформації про фільм або серіал.
- пошук тексту у субтитрах.
- перегляд субтитрів.
- реєстрація користувачів
- авторизація користувачів.

Після реєстрації користувач отримує можливість увійти в свій аккаунт. Цей тип користувача вважається авторизованим користувач та отримує додатковий функціонал:

- управління своїм обліковим записом.
- автоматична генерація субтитрів за допомогою штучного інтелекту.
- управління своїм списками контенту.
- управління інформацією про контент.

Існує також інший користувача – адміністратор. Адміністратор має доступ до того ж функціоналу, що й авторизований користувач, але ще в додачу у нього є доступ до:

- управління обліковими записами користувачів.

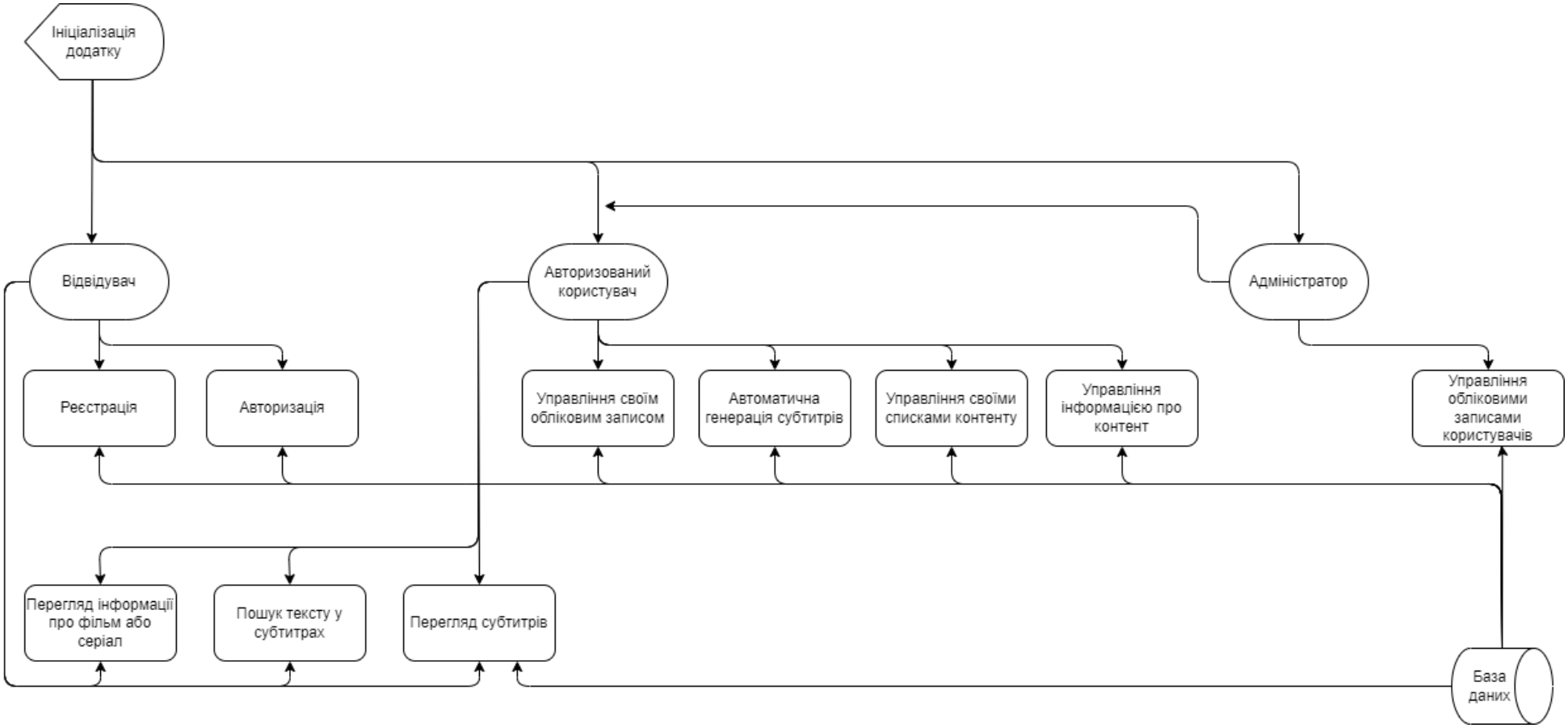

Рисунок 2.1 — Функціональна схема

2.2 Розробка та опис алгоритму роботи програми

Розглянемо алгоритм пошуку тексту у субтитрах за допомогою блок-схеми.

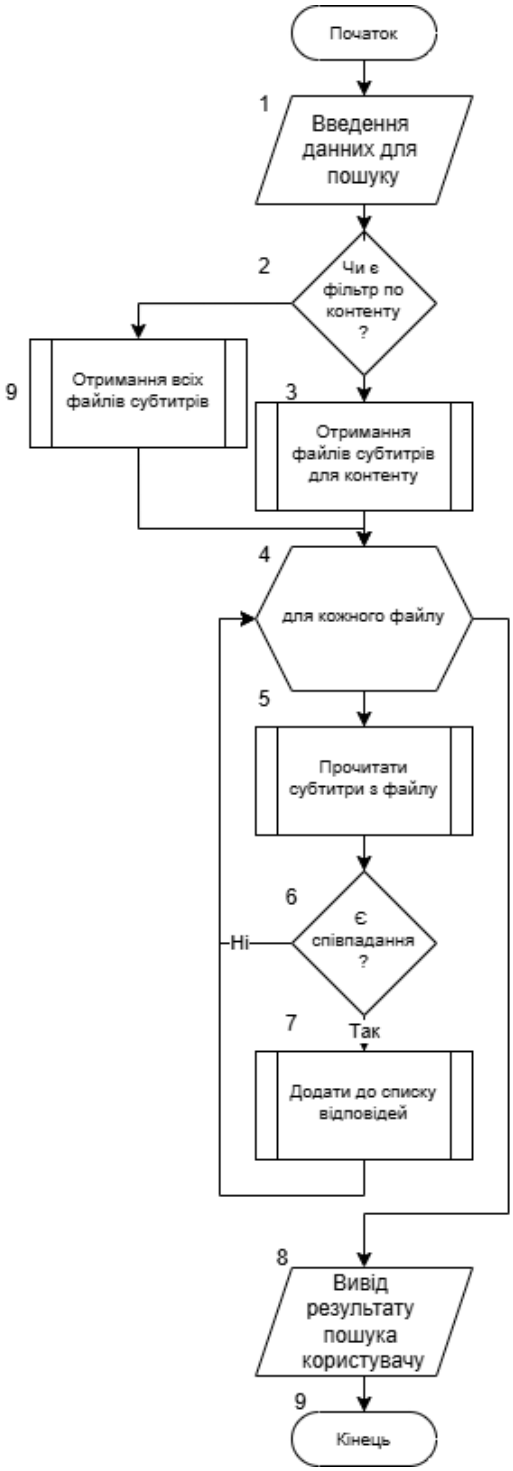

Рисунок 2.2 — Блок-схема алгоритму пошуку тексту в субтитрах

Спочатку користувач задає вхідні дані для пошуку тексту у субтитрах. Це означає, що користувачу потрібно ввести текст, який він хоче знайти в субтитрах.

Також користувач має можливість обмежити пошук лише за конкретним контентом, будь то фільм або серіал. Це може зробити пошук набагато швидшим ніж звичайний пошук тексту без фільтра контенту. Вся різниця полягає в тому, що при звичайному пошуку тексту без фільтру контенту система перевіряє кожен файл субтитрів на відповідність, що займає деякий час. Наприклад, у системі є 1000 різних файлів субтитрів з 10 серіалів фільмів, кожен серіал має по 100 субтитрів. У першому випадку користувач робить пошук по тексту без фільтра по серіалу, тому система перевіряє всі 1000 файлів. У другому випадку користувач робить пошук тексту, але також вибирає фільтр по 1 серіалу, у разі цього система перевіряє лише 100 файлів субтитрів цього серіалу. В результаті, у другому випадку система отримала той же результат, але зробила це набагато швидше, тому, що вона перевірила в 10 разів менше файлів.

Переваги пошуку тексту з фільтром по тексту є великі, але також є і недоліки. Фільтр по контенту можна лише використовувати тоді, коли користувач знає в якому контенті знаходиться потрібна йому інформація. Якщо користувач зробить пошук з фільтром на серіал №1, але інформація насправді знаходиться в субтитрах серіалу №2, то система покаже користувачу, що інформацію в серіалі №1 не знайдено.

Цей фільтр є оптимізацією пошуку, яку може здійснити користувач, але це не є обов’язковим. Користувач може власноруч вибрати фільтр по контенту та збільшити швидкість пошуку, а може й не вибрати фільтр по контенту, зменшивши роботу для себе, але збільшивши роботу для системи пошуку.

Правило використання є таким, що якщо користувач впевнений в фільтрі, то його краще використовувати, але якщо є сумніві, то можна не робити фільтр по контенту і просто трошки більше чекати результату.

Після того, як список файлів було зроблено, система проходиться по кожному файлу та перевіряє чи має він співпадіння з текстом від користувача. У разі знаходження співпадіння, вони додаються до списку відповідей. Після перевірки всіх потрібних файлів, список відповідей виводиться користувачу.

Розглянемо алгоритм роботи оновлення паролю за допомогою блок-схеми.

Рисунок 2.3 — Блок-схема алгоритму оновлення паролю

Спочатку користувач вводить вхідні дані. Це означає, що йому потрібно ввести його пароль, новий пароль та підтвердження нового паролю.

Спочатку йде перевірка нових паролів на вимоги паролів. Користувач, наприклад, не може створити аккаунт у якого не буде жодної букви в паролі.

Введення поточного паролю потрібно для безпеки користувача. Якщо користувач ввів неправильний пароль, то йому буде повідомлено про цю помилку.

Новий пароль користувач повинен написати одразу два рази підряд. Це зроблено для того, щоб захистити користувача від випадкового вибору неправильних літер під час оновлення паролю. Якщо користувач випадково натисне якусь літеру на клавіатурі без цієї перевірки, то він змінить свій пароль на неправильну комбінацію, яка йому невідома. Ця перевірка захищає від цього, і навіть якщо користувач випадково десь натиснув на клавіатурі не ту літеру, система захистить користувача від цього і повідомить його про те, що його паролі не співпадають.

Після перевірок відбувається генерація хешу пароля користувача. Генерація хешу паролю робиться заради безпеки даних користувача. В базі даних система не зберігає паролю користувача у текстовому виді, в ній зберігається лише хеш паролю. Це означає, що ніхто не зможе дізнатися паролю користувача навіть, якщо він отримає доступ для бази даних.

Після генерації хешу паролю нові дані будуть записані в базу даних і користувач буде повідомлений про успішне оновлення паролю.

Розглянемо алгоритм автоматичної генерації субтитрів за допомогою блок-схеми.

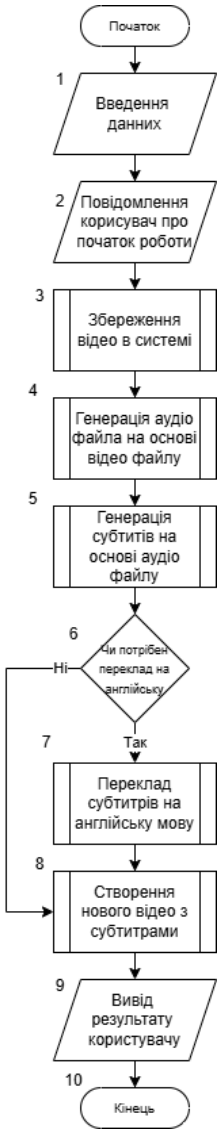

Рисунок 2.4 — Блок-схема алгоритму автоматичної генерації субтитрів

Спочатку користувач завантажує відео для якого він хоче автоматично генерувати субтитри. Після цього система повідомляє користувача про початок роботи. Повідомлення потрібно тому, що процес автоматичної генерації субтитрів займає певний час. Для генерації потрібно працювати не лише з базою даних, а і з файловою системою, різними бібліотеками, тощо. Для цього після завантаження відео користувачем, він потрапляє на сторінку перегляду процесу генерації субтитрів. На цій сторінці користувач може переглянути на якому етапі генерації зараз знаходиться система.

Першим етапом є збереження відео на сервері. Для цього система зберігає файл відео в файловій системі серверу та записує ці дані в базу даних для подальшого використання.

Другим етапом є генерація аудіо файлу на основі відео файлу. Автоматична генерація субтитрів відбувається на основі аудіо, а не відео файлів, тому потрібен аудіо файл. Звісно, не буде дуже добре, якщо користувачу самостійно потрібно буде генерувати аудіо файли. Це буде потребувати додаткових технічних навичок, часу та терпіння. Система автоматично підготовлює аудіо файли на основі тих відео файлів, що завантажив користувач. Це робить процес автоматичної генерації субтитрів значно швидшим та простішим для користувачів.

Наступним етапом є генерація субтитрів на основі аудіо файлу. Система генерує субтитри використовуючи штучний інтелект та ті параметри генерації, які вказав користувач. В формі автоматичної генерації субтитрів користувач може вибрати модель штучного інтелекту, яку він хоче використовувати. Система має звичайну модель, яка використовується для більшості робіт, але користувач може вибрати іншу за своїм бажанням. Вибір моделі значно впливає на потребу в ресурсах, швидкість та точність генерації субтитрів. Деякі моделі орієнтовані на швидкість роботи, а інші – точність генерації. Якщо користувач хоче результату швидше, то він може вибрати швидшу модель, а у разі, якщо йому потрібен більш правильний результат – він може обрати більш точну модель, яка займає більше часу для отримання результату. В результаті система зберігає у файловій системі сервера файл субтитрів, який був згенерований за допомогою штучного інтелекту.

Система може автоматично генерувати субтитри для багатьох різних мов, тому в формі автоматичних субтитрів є опція автоматичного перекладу на англійську мову. У разі, якщо вибрана ця опція, то система перекладає субтитри з оригінальної мови відео на англійську. Ця опція може бути корисною для тих користувачів, котрі хочуть почути контент на іншій мові, але ще не мають достатніх знань, щоб їм вистачило лише субтитрів на цій мові, тому вони можуть використати субтитри на англійській мові.

Потім система генерує відео файл з вбудованими субтитрами та зберігає його у файловій системі сервера. Після цього всього користувач

бачить результат. Він може відразу переглянути відео з субтитрами, скачати його на власний комп’ютер, або просто скачати файл з субтитрами.

2.3 Інтерфейс програми

Інтерфейс системи повинен бути ефективним в вирішенні задач, які хоче зробити користувач. Головною метою повинна бути функціональність. Інтерфейс повинен бути інтуїтивним, щоб користувачі могли успішно та стабільно працювати з системою та зробити ознайомлення з системою більш зрозумілим.

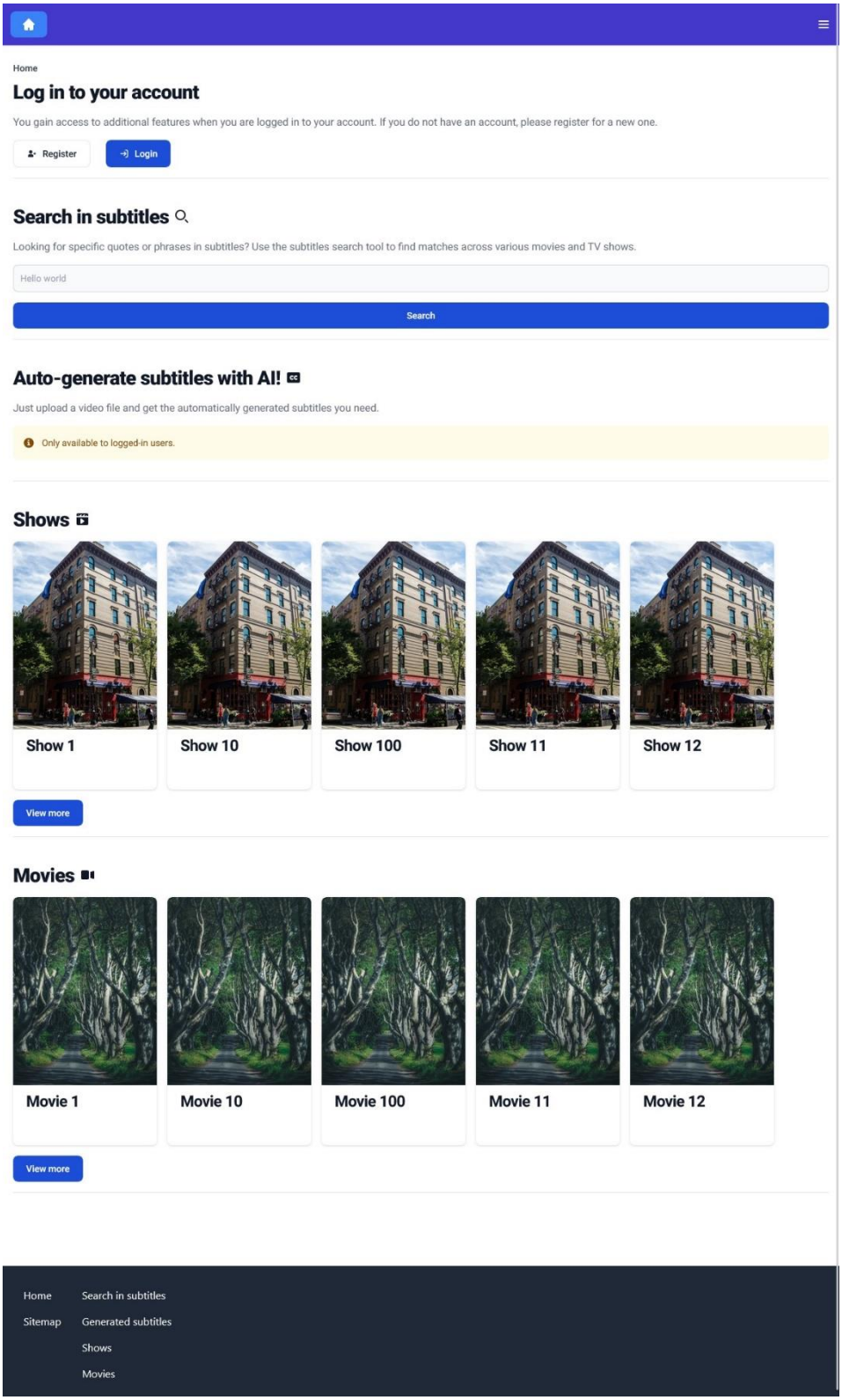

Рисунок 2.5 — Головна сторінка

Зверху сторінки ми бачимо повідомлення про те, що користувачу пропонують авторизуватися в системі для отримання додаткового

функціоналу. Якщо користувач авторизується в системі, то це повідомлення зникне.

Нижче можна побачити функціонал пошуку тексту в субтитрах. Це є одним з найголовніших типів функціоналу систему, тому він займає важливе місце на головній сторінці. Користувач може ввести текст для пошуку та натиснути на кнопку і почати пошук. Також користувач має можливість натиснути на кнопку пошуку без вводу тексту та одразу перейти на сторінку пошуку тексту і вже на новій сторінці ввести текст для пошуку.

Потім йде блок про автоматичну генерацію субтитрів. Автоматична генерація субтитрів за допомогою штучного інтелекту є дуже важливою частиною систему, тому цей блок йде відразу після блоку з пошуком. Користувач може побачити тут повідомлення про те, що для доступу до цього функціоналу йому потрібно авторизуватися в системі.

Нижче користувач може переглянути список серіалів, які є в системі. Цей блок ознайомлює користувача з тим, що в системі є інформація про різні серіали. Цей блок показує не всі серіали, які є в системі, а лише деяку обмежену вибірку. Це зроблено для того, щоб не перенавантажити користувача інформацією. Якщо ж користувач хоче детальніше ознайомитись з інформацією, він має можливість перейти по посиланню на сторінку серіалів і вже там ознайомити з значно більшим набором інформації про серіали.

Також в системі є різні фільмі, тому наступний блок має обмежену вибірку фільмів, які є в системі. Принцип роботи цього блоку є дуже схожим з блоком про серіали. Цей блок ознайомлює користувача з тим, що в системі зберігається інформація про фільмі та дає можливість перейти по посиланню, щоб дізнатись більш детальну інформацію.

Знизу сторінки є список корисних посилань для користувача. Тут можна знайти мапу сайту, посилання на головну сторінку, тощо.

Розглянемо вид головної сторінки на екрані мобільного телефону.

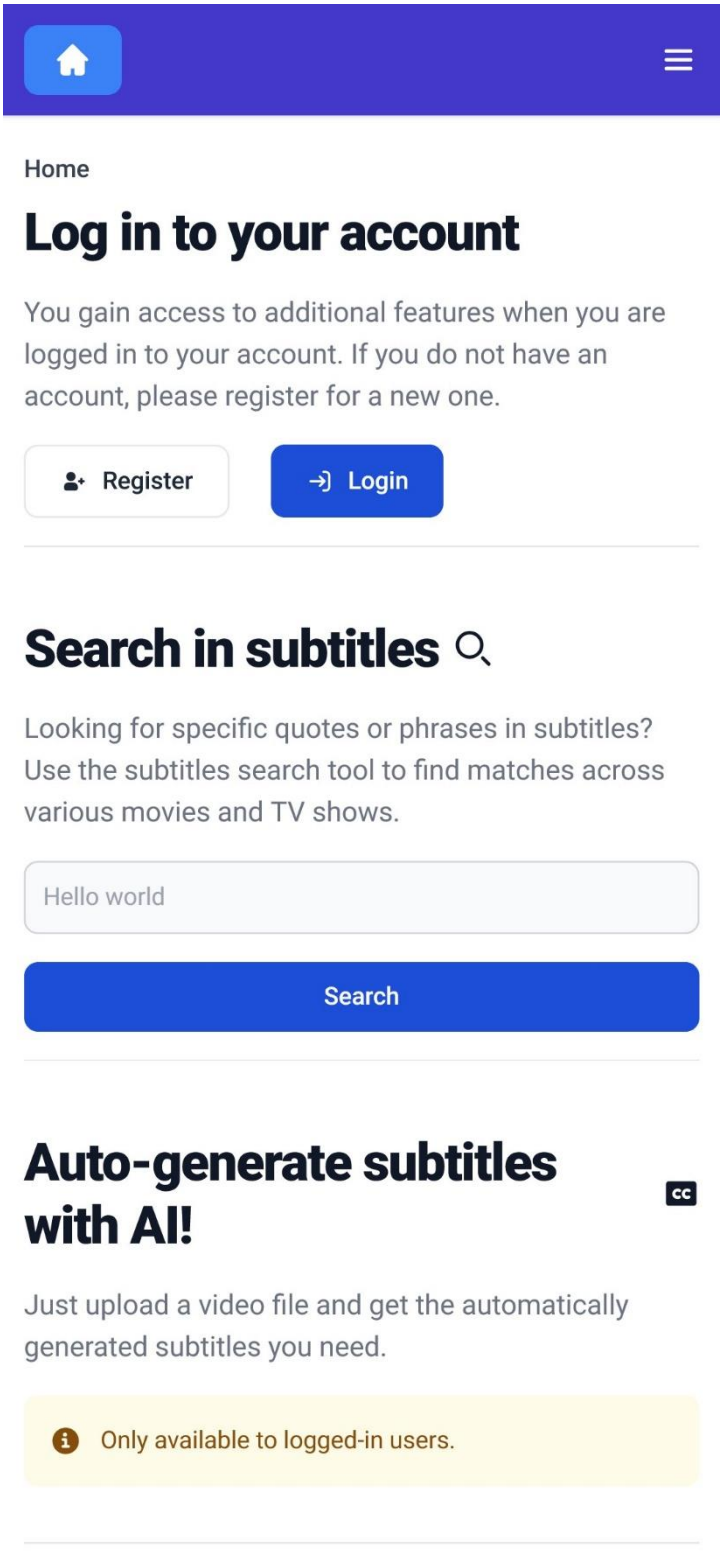

Рисунок 2.6 —Головна сторінка на екрані мобільного телефону

Головна сторінку на екрані мобільного телефону дуже схожа на версію для комп’ютера. Деякі речі трохи змінилися, але набір інформації залишився

незмінним. Ідея полягає у тому, щоб надати користувачу один і той же набір інформації як на комп’ютері так і на телефоні. Звісно, в деяких місцях може трохи змінитися спосіб подачі інформації, але інформація змінюватися не буде.

Розглянемо меню, що можна відкрити натиснувши на кнопку вгорі сторінки.

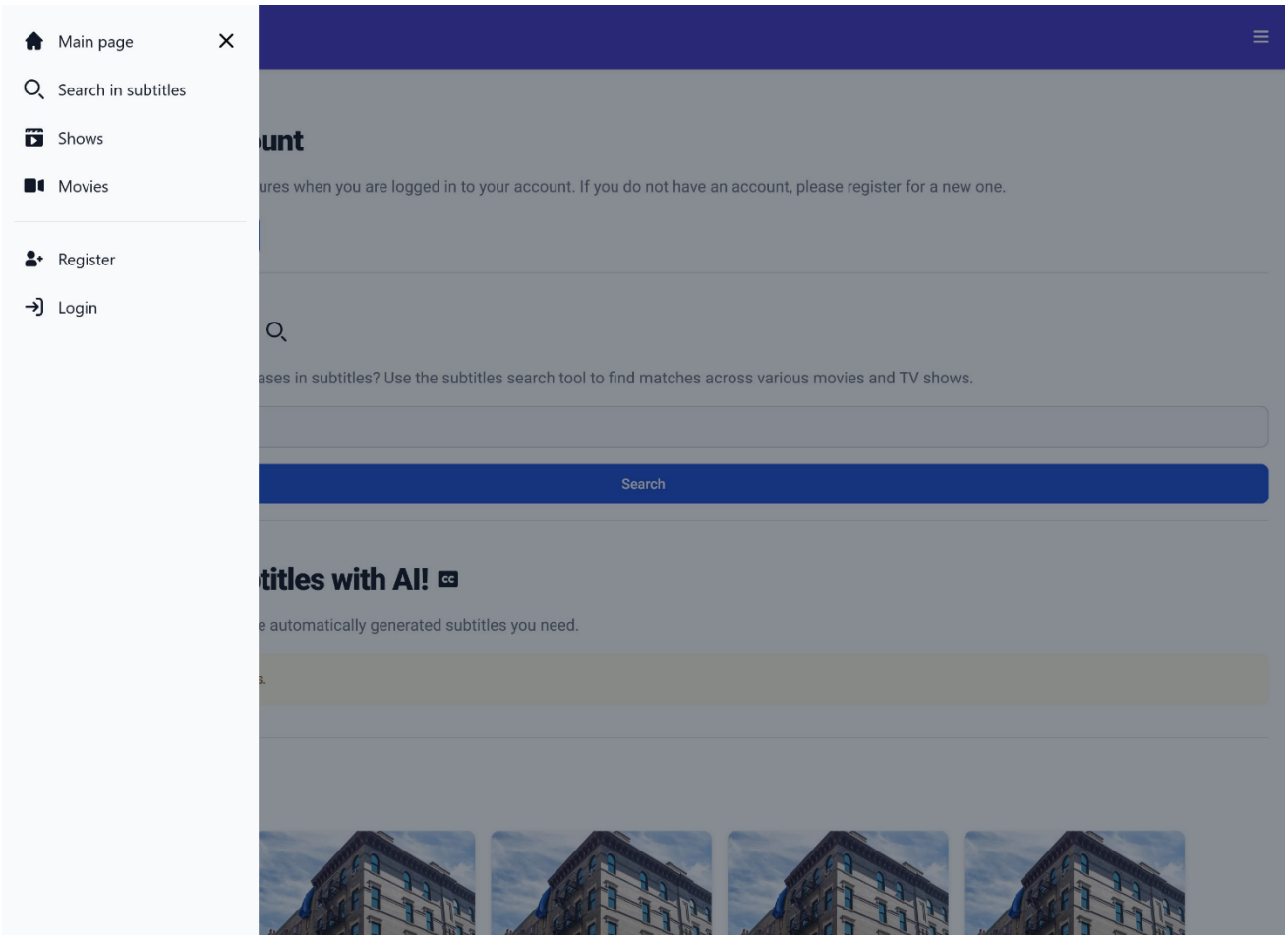

Рисунок 2.7 — Меню

В цьому меню можна побачити список корисних посилань. Перше посилання є посиланням на головну сторінку. Це меню можна відкрити майже з будь-якої сторінки, тому функціонал повернення на головну сторінку є дуже корисним. Потім є посилання на сторінку пошуку тексту в субтитрах, що також є дуже корисним, воно дає змогу користувачам швидко почати пошук. Потім йдуть посилання на серіали та фільми в системі. Після

них є посилання на реєстрацію або авторизацію в системі. Це меню має посилання на найважливіші сторінки в дуже зручному місці, що дає користувачам можливість дуже швидко та легко переходити до потрібних сторінок.

Розглянемо сторінку реєстрації користувача.

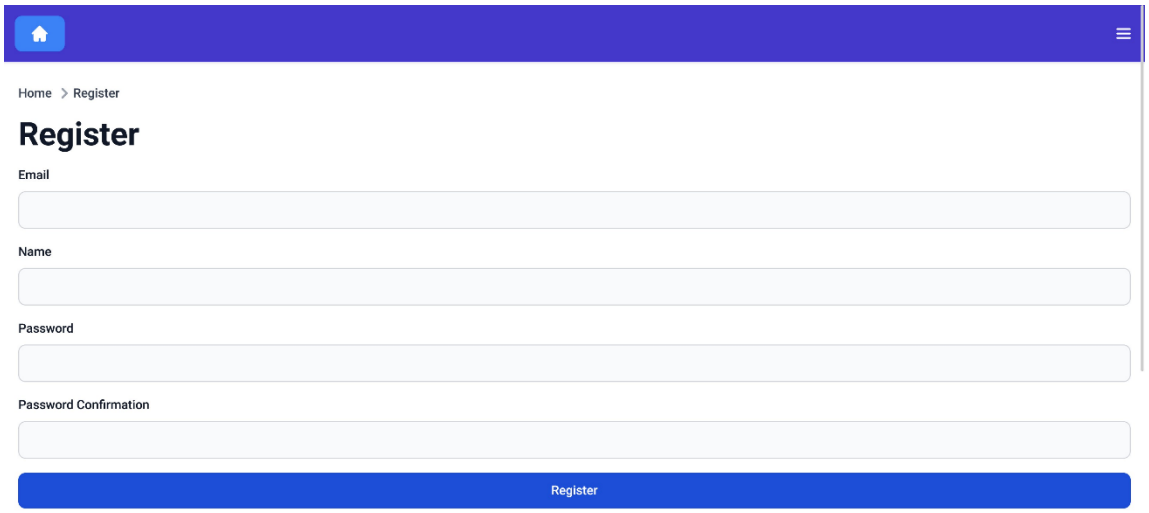

Рисунок 2.8 — Сторінка реєстрації користувача

На цій сторінці можна побачити інформацію, яку потрібно надати користувачу для того, щоб створити аккаунт в системі. Це означає, що йому потрібно ввести свою електронну пошту, ім’я, пароль та підтвердження паролю. Якщо користувач зробить якусь помилку під час заповнення форми, система повідомить його про це.

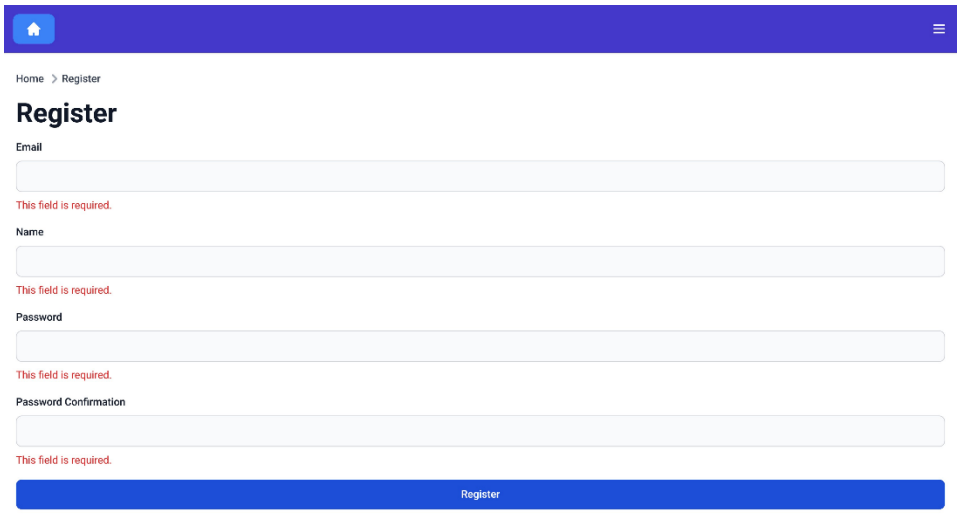

Рисунок 2.9 — Повідомлення про помилку на сторінці реєстрації користувача

Кожне поле в формі має власні правила заповнення, котрі повинен виконати користувач. Якщо користувач порушить правило заповнення поля, то під полем з’явиться повідомлення про те, що виникла помилка.

У разі успішного заповнення форми користувача буде повідомлено про створення нового акаунту.

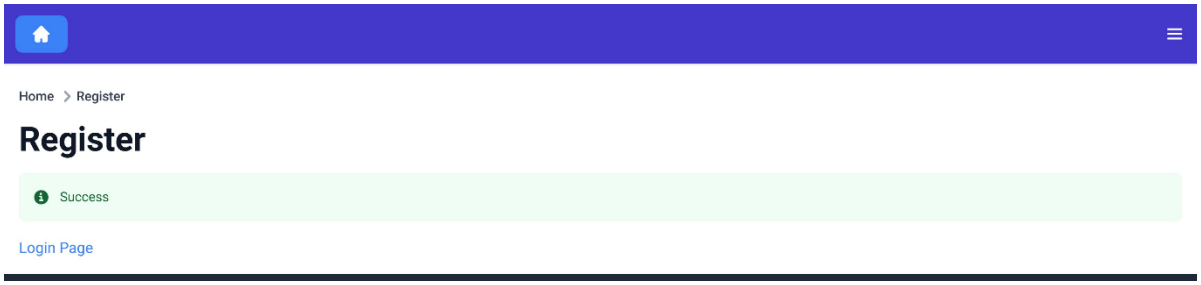

Рисунок 2.10 — Успішна реєстрація користувача

Сторінка авторизації виглядає наступним чином. Користувачу потрібно заповнити форму, щоб авторизуватися в системі. Для цього потрібно ввести електронну пошту та пароль.

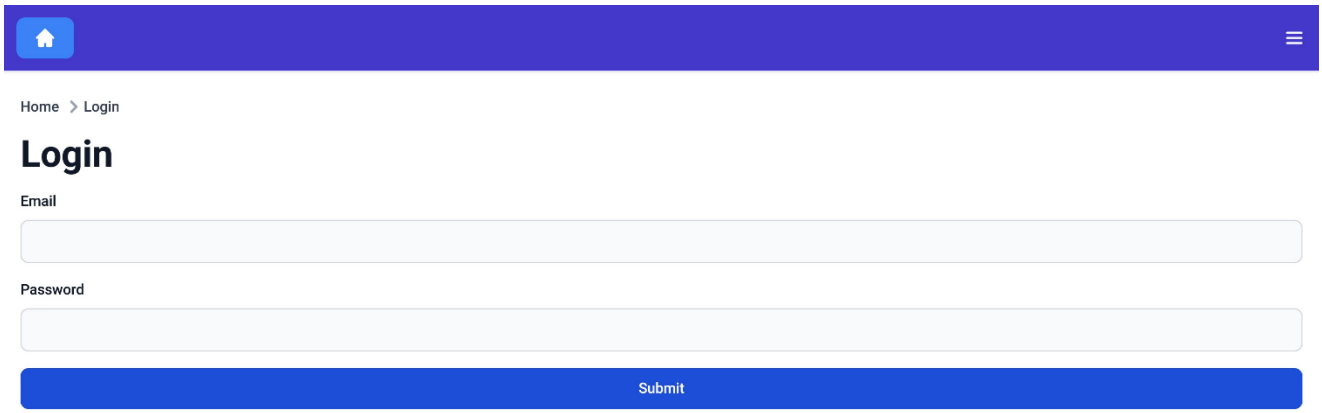

Рисунок 2.11 — Сторінка авторизації

Після успішної авторизації користувач потрапляє на головну сторінку. Розглянемо головну сторінку для авторизованого користувача.

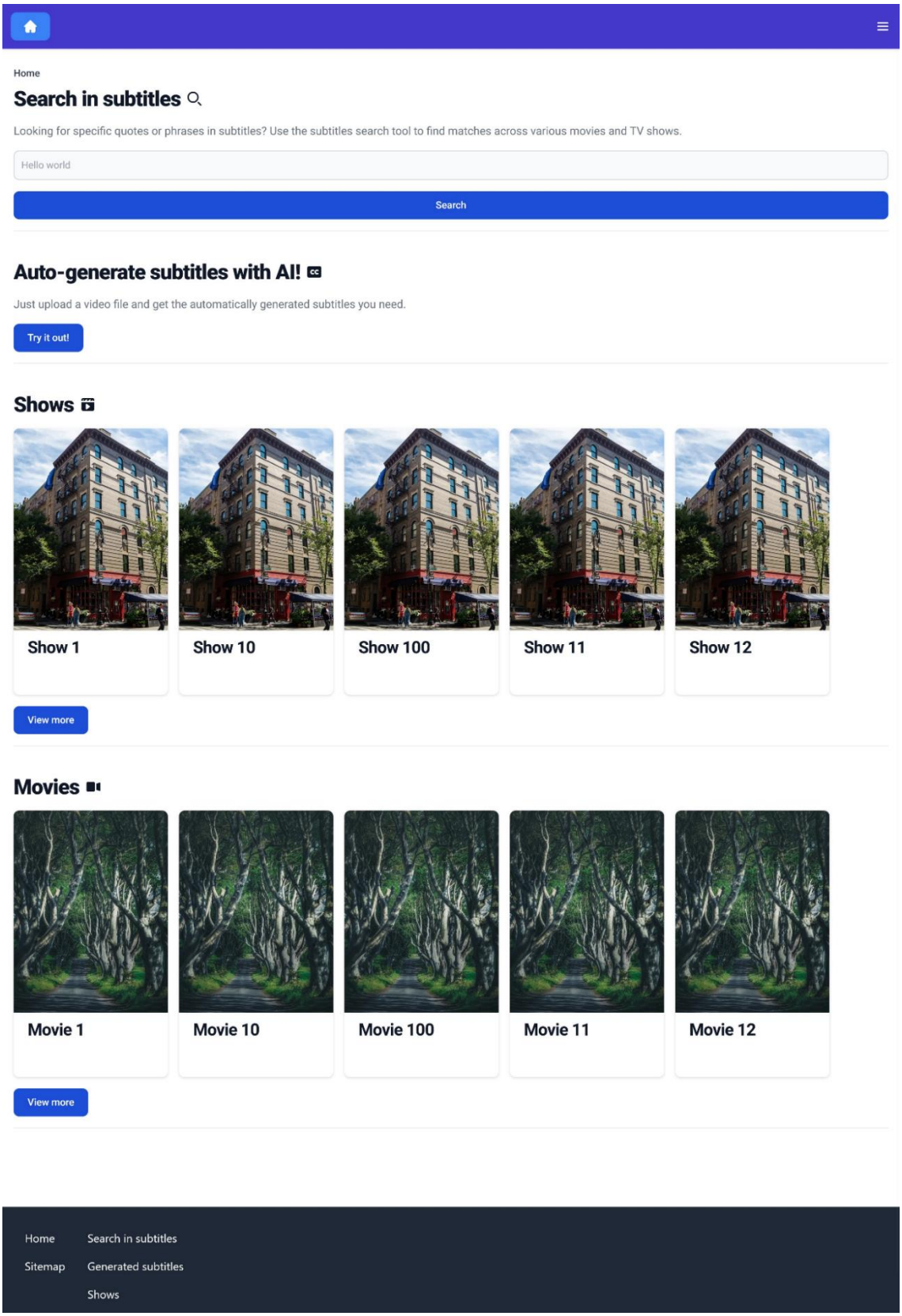

Рисунок 2.12 — Головна сторінка для авторизованого користувача

Вид головної сторінки для авторизованого користувача відрізняється від головної сторінки для неавторизованих користувачів.

Зник блок, який повідомляє користувачів про те, що їм варто авторизуватися в системі. Якщо користувач вже авторизувався в системі, то йому більше не потрібно про це нагадувати, тому цього блоку більше немає.

Також зникло повідомлення про те, що потрібно авторизуватися в системі, щоб автоматично генерувати субтитри за допомогою штучного інтелекту. Користувач вже є авторизованим, тому на старому місці повідомлення знаходиться посилання на сторінку автоматичної генерації субтитрів за допомогою штучного інтелекту.

Блоки серіалів та фільмів не мають візуальних змін, тому, що функціонал перегляду інформації про контент вже був доступний для неавторизованих користувачів.

Розглянемо меню для авторизованих користувачів.

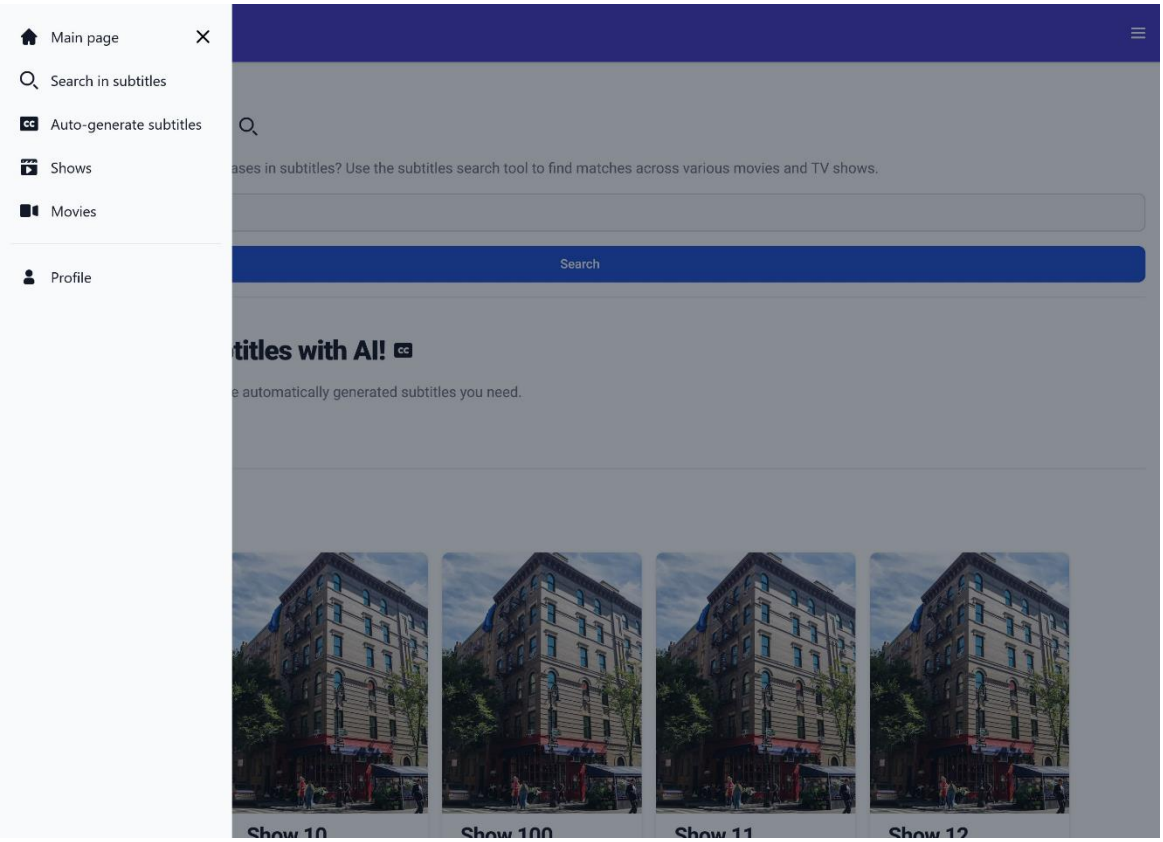

Рисунок 2.13 — Меню для авторизованого користувача

Меню для авторизованих користувачів відрізняється від меню для авторизованих в системі користувачів. Було додано посилання на сторінку

для автоматичної генерації субтитрів за допомогою штучного інтелекту. Цей функціонал не є доступним для авторизованих користувачів, тому його раніше не було в цьому меню. З меню зникли посилання на сторінки реєстрації та авторизації користувача в системі. Якщо користувач вже є авторизованим в системі, то йому не потрібні такі посилання. На місце цих посилань з'явилось посилання на сторінку профілю користувача.

Розглянемо сторінку профілю користувача.

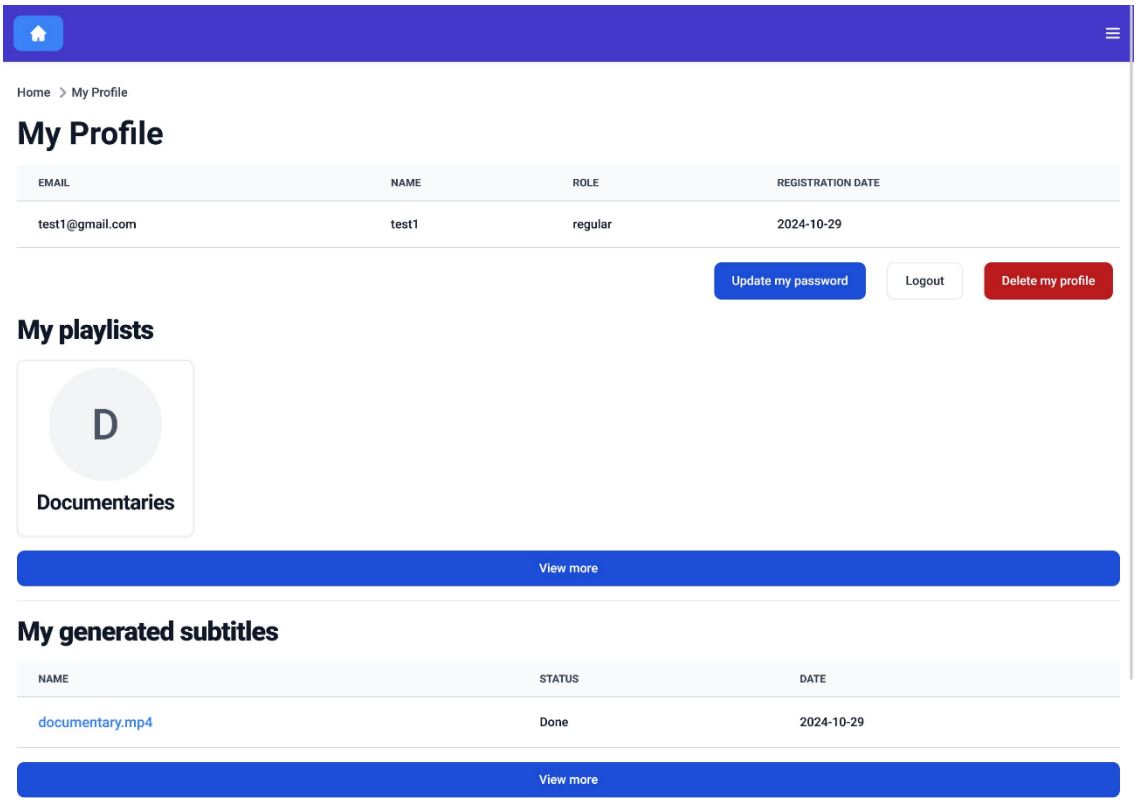

Рисунок 2.14 — Сторінка профілю користувача

Сторінка має інформацію про користувача, його ім'я, електронну пошту, дата реєстрації, роль. Також ця сторінка має функціонал оновлення паролю, виходу з аккаунту та видалення облікового запису.

Для виходу з аккаунта або видалення з облікового запису потрібно натиснути на відповідну кнопку на сторінці, після чого система запитає чи є користувач впевненим в тому, що він хоче зробити. Якщо користувач впевнений, то він має підтвердити свою дію натиснувши на кнопку

підтвердження, після чого система виконує відповідну операцію та повідомляє користувача про результат. Якщо користувач не є впевненим, то він не підтверджую свою дію та система не робить ніяких нових операцій. Система потребує підтвердження дій в цьому випадку тому, що дії є дуже важливими і потім не буде можливість їх відмінити. Якщо користувач вийде з аккаунту, то потім йому потрібно буде знову авторизуватися в системі на сторінці авторизації. У випадку видалення облікового запису користувача, його обліковий запис буде повністю видалено і система не буде мати можливості потім відмінити ці дії, тому потрібно бути дуже обережними.

Нижче можна побачити блок з вибірками контенту користувача. В цьому блоці можна побачити деякі вибірки, які раніше були створені користувачем. Також в цьому блоці є посилання на сторінку вибірок, де можна переглянути всі вибірки контенту користувача, створити нові, або видалити старі.

Внизу сторінки є блок з автоматично створеними субтитрами за допомогою штучного інтелекту. В цьому блоці можна побачити вибірку деяких згенерованих субтитрів. Також є посилання на сторінку, де можна побачити повний список згенерованих субтитрів користувача.

Розглянемо сторінку оновлення паролю.

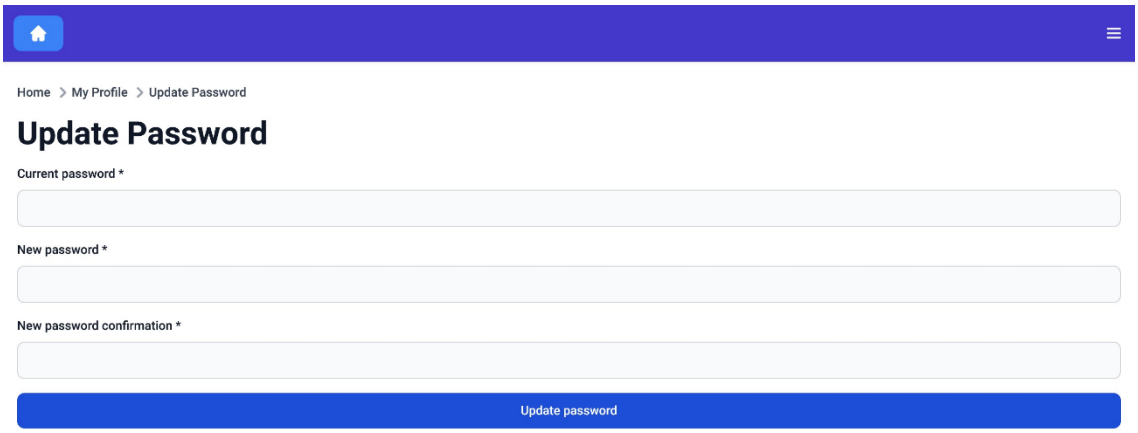

Рисунок 2.15 — Сторінка оновлення паролю

Для оновлення паролю користувачу потрібно ввести свій пароль, новий пароль та підтвердження нового паролю. У випадку, якщо користувач зробить помилку під час заповнення форми, система повідомить його про помилку. У разі успішного заповнення форми, система змінить пароль користувача.

3 РОЗРОБКА БАЗИ ДАНИХ І ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Система буде реалізована за принципами веб-додатку, що передбачає розробку серверної частини і клієнтської частини для повної реалізації всіх можливостей. Серверна частина відповідає за всі ті речі, що будуть відбуватися на сервері. Робота з базою даних, файловою системою та багато інших речей будуть відбуватися на сервері. Клієнтська частина це та частина, що працює з браузером клієнта. Наприклад, стилі та дизайн продукту є зоною відповідальності клієнтської частини. Спілкуватися між собою клієнтська та серверна частини будуть за допомогою HTTPS.

3.1 Вибір технології розробки інтерфейсу

Інтерфейс – це зона відповідальності клієнтської частини, тому потрібно використовувати технології, які добре працюють з браузерами.

Перше - HTML (HyperText Markup Language). Ця стандартна мова розмітки є стандартом розробки веб-додатків [7]. Кожен сучасний браузер дуже добре працює з цією технологією, тому користувачі можуть не перейматися за те, чи зможуть вони працювати з системою на своєму комп’ютері чи телефоні. HTML має різні теги та атрибути для створення різних елементів.

Стандарт стилізації веб-додатків – CSS (Cascading Style sheets) [8]. За допомогою HTML можна створити елемент, але щоб він гарно виглядав потрібно використовувати CSS. Ця технологія надає можливість стилізувати майже кожний HTML елемент. Стилізація зробить веб-додаток більш гарним на вигляд, збільшить ефективність та продуктивність користувачів та вирішить багато проблем презентація, які неможливо вирішити без CSS. Кожен браузер має стандартний набір стилів для різних HTML елементів, але ці стилі відрізняються від одного браузера до іншого. CSS дає можливість розробникам зменшити цю проблему і забезпечити користувачів різних браузерів стабільними та гарними HTML елементами. За допомогою CSS

веб-додаток буде мати стабільний вигляд в сучасних браузера та більш гарні та ефективні елементи для продуктивної роботи користувача.

Потрібно вирішити питання взаємодії користувача з системою. HTML та CSS не можуть самостійно забезпечити високий рівень ефективності в взаємодії користувача з системою, тому потрібно використовувати TypeScript.

Сучасні браузери не можуть напряму працювати з TypeScript, але вони можуть працювати з JavaScript. Щоб вирішити цю проблему TypeScript компілюється в JavaScript, що дає йому змогу працювати з браузером. JavaScript був створений для того, щоб зробити веб-сторінки більш інтерактивними і він добре виконую цю задачу, але в цієї мови є певні недоліки. Одним з найголовнішим недоліків JavaScript є робота з типізацією. JavaScript є дуже динамічною мовою програмування, де дуже легко зробити помилку і зробити операцію з неправильним типом даних. TypeScript вирішує цю проблему дуже добре, він значно покращує роботу з різними типами даних і інтерфейсами в мові JavaScript [9]. В результаті виходить мова, яка може дуже ефективно робити веб-сторінки інтерактивними та має потужну систему типізації. Потужна система типізації значно збільшує швидкість та продуктивність розробки програмного забезпечення, зменшуючи кількість проблем при роботі з різними типами даних.

Для того, щоб побудувати продуктивну систему потрібно вибрати фреймворк, який допоможе краще об’єднати всі ці технології разом та допомогти в розробці.

Одним з найпопулярніших сучасних фреймворків є ReactJS. Ця технологію дозволяє розробникам будувати інтерфейси використовуючи декларативний підхід [10]. Одним з основних блоків є компоненти. В цих компонентах потрібно описати те, як вид компоненту залежить від його даних. Компоненти можуть бути різними, мати різний розмір, виконувати різні задачі, тощо. Один компонент може відповідати за форму створення, а інший за показ даних у таблиці, тощо. Потім потрібно зібрати всі ці різні

компоненти в одне ціле і в результаті буде система, котра реагує на зміни даних та змінює вид відповідно до цього.

Існують і інші популярні альтернативи: AngularJS, VueJS, тощо. Вони також мають велику кількість корисного функціоналу та дозволяють будувати комплексні системи, як і з ReactJS. З недоліків можна сказати про те, що вони менше популярні ніж ReactJS, тому мають меншу екосистему для розробників. Це означає, що кількість різних допоміжних бібліотек та компонентів є меншою, що є великим мінусом. ReactJS має дуже велику екосистему, де можна знайти велику кількість різних допоміжних бібліотек та компонентів. Також до списку недоліків можна додати історичні проблеми з міграціями між різними версіями. Велика кількість різних систем мають проблеми з міграціями, але ReactJS має меншу кількість проблем в цій сфері порівняно з альтернативами.

В результаті для цієї обрати було обрано ReactJS.

3.2 Вибір технології розробки серверної частини

Робота з базою даних, файловою системою та багато чого іншого – це зона відповідальності серверної частини. Потрібно вибрати технологію, яка буде добре та ефективно виконувати таку роботу.

Гарним рішенням є NodeJS. NodeJS – дає можливість працювати з JavaScript та TypeScript не тільки в браузері, а ще й на серверній частині [11]. Це рішення дозволить використовувати одну мову програмування для роботи як з клієнтською, так і з серверною частиною. Це значно збільшить швидкість розробки тому, що можна буде використовувати деякі спільні модулі та частини коду відразу в обох частинах.

Одна з найбільших переваг NodeJS - асинхронність. Вона дає можливість ефективно працювати з великою кількістю файлів, працювати з базою даних та робити ще багато корисних речей. Найважливіші частини функціоналу на серверній частині буду використовувати асинхронність для продуктивності роботи користувача та оптимізації функцій системи.

З недоліків є те, що NodeJS не є найшвидшою технологією для серверної частини веб-додатків в плані швидкості роботи сервера. Втім, вона не є і найповільнішим варіантом, вона має оптимальну швидкість для веб-додатків. В випадку з системою субтитрів це не буде великою проблемою, тому, що ця система здебільшого орієнтована на роботу з базою даних та файловою системою, що NodeJS дуже гарно робить за допомогою асинхронних функцій. У цій системі не має потреби у найшвидших в світі розрахунках, як, наприклад, в системах торгівлі акціями. Якщо розробляти систему, яку буде займатися торгівлею акціями різних компаній, то дійсно вибір NodeJS не був би найкращим рішенням в такому випадку, але система управління субтитрами є іншим випадком.

Для розробки найшвидших серверів можна використовувати такі мови програмування як Golang або C++, але ці мови також мають недоліки. Одним з найбільших недоліків таких технологій є швидкість розробки, вона значно повільніше ніж в NodeJS і проекти займають більше час перш ніж вони досягають своєї цілі. Це компроміс між швидкістю розробки та швидкістю роботи сервера. Потрібно знайти баланс між цими речами, тому для цього проекту було обрано NodeJS для розробки серверної частини. Вона має добрий баланс швидкості розробки і швидкості роботи, що дозволить розробити цю системою в оптимальний час і отримати в результаті ефективну швидкість сервера.

Whisper від компаній OpenAI є чудовим кандидатом для автоматичної генерації субтитрів. Ця модель має підтримку багатьох різних мов, що робить її дуже корисною для багатьох різних задач. Модель має велику якість та точність роботи. Whisperer є open source, використовує MIT ліцензію, тобто використання є безкоштовним і може бути адаптовано для вимог користувача [12]. Модель може бути завантажена на комп’ютер і потім працювати навіть без доступу до інтернету. Велика кількість схожих моделей працюють лише через API запити до серверу компанії, що в даному випадку є великим недоліком. Це означає відносно великі грошові витрати порівняно

з ціною електрики, якщо запускати модель на своєму власному сервері. Також стабільність таких моделей має багато питань. Моделі можуть бути оновленні компанією на своєму сервері без оповіщення клієнтів, після чого користувачі отримують інший результат, а не той, що очікували. Також не потрібно забувати, що це також додає ризик, що сервер компанії може тимчасово вийти із ладу або компанія може стати банкротом, що також буде дуже погано для користувачів.

Модель Whisper має велику кількість різних мов з якими вона може працювати. Це буде дуже корисно для створення субтитрів для різних типів відео з різних країн. Також модель має можливість автоматично перекласти субтитри на англійську мову, якщо потрібно. Це може бути дуже корисно для користувачів, які вивчають іноземні мови і хочуть їх почути, але все ще потребують субтитрів на англійській мові.

Також в опціях можна вибрати конкретну модель для роботи. Whisper має набір моделей: tiny.en, tiny, base.en, base, small.en, small, medium.en, medium. large, turbo. Всі ці моделі мають власні характеристики, точність роботи, потребу у ресурсах, доступний набір мов. Деякі моделі можуть працювати лише з англійською мовою. Здебільшого моделі знаходяться на спектрі точності і потребу у ресурсах. Чим більш точна модель, тим більше вона потребує ресурсів. Чим швидше модель, тим менше її точність порівняно з іншими моделями. Ці опції є дуже корисними і дають користувачу змогу обрати ту модель, яка оптимально робить те, що йому потрібно.

В результаті була вибрана модель Whisperer від компанії OpenAI для автоматичної генерації субтитрів. Вона має великий набір корисного функціоналу та чудову стабільність, що робить її дуже гарним варіантом для розробки системи управління субтитрами з використанням штучного інтелекту.

3.3 Вибір середовища розробки

Для продуктивної роботи потрібно вибрати ефективну середу розробки. Гарним рішенням є Visual Studio Code (VS code).

Visual Studio Code – редактор коду, розроблений Microsoft у 2015 році. Цей редактор має відкритий код та великий набір різних додатків [13].

Велика кількість додатків є величезною перевагою, можна знайти додаток майже на будь-який смак. Це дозволяє розробникам працювати з усіма потрібними технологіями в одній програмі, що підвищує продуктивність. Розробниками не потрібно постійно переходити з однієї програму в іншу, щоб працювати з іншою технологію, згадувати які клавіші в якій програмі відповідають за який функціонал, тощо. Це дає можливість відкрити потрібні речі в одній програмі, почати ефективно працювати і не відволікатися на різні речі.

Цей редактор також має велику кількість людей, які постійно займаються його розробкою і покрашенням. Компанія Microsoft, яка займається розробкою, добре розуміє великі переваги, які має цей продукт. Тому ця компанія має команду розробників, які на постійній основі стабільно працюють над цим продуктом. Ця стабільність є дуже важливою, адже велика кількість редакторів коду розробляються волонтерами, які можуть просто не мати часу швидко додати нові технології або вирішити інші проблеми з редактором.

Найкращим варіантом для розробки цього продукту є Visual Studio Code.

3.4 Розробка бази даних

Для системи потрібно обрати технологію бази даних. Це є дуже важливим рішенням, адже від цього залежить дуже багато речей.

Гарним рішенням для цієї системи є SQLite. Це реляційна система бази даних, яка використовує SQL (Structured Query Language) [14]. Вона є дуже простою у використанні в порівнянні з альтернативами, не потребує складної

конфігурації чи налаштування. SQLite дозволяє досить просто вбудовувати її в різні додатки та має велику швидкість та ефективність роботи. Вона є перевіреною часом стабільною технологією, котра ефективно виконує свої задачі.

Іншими варіантами можуть бути Postgres, MySQL або MongoDB. Postgres та MySQL також є реляційними базами даних, як і SQLite, але вони ще в додачу мають деякий додатковий функціонал. Цей додатковий функціонал є цікавим, але система управління субтитрами не дуже сильно потребує такого комплексного функціоналу. Також ці системи набагато складніше правильно налаштувати та запустити, вони є набагато менш портативними та потребують більше ресурсів. В деяких випадках деякі системи дійсно потребують комплексного функціоналу Postgres або MySQL, але система управління субтитрами не є таким випадком.

MongoDB – база даних орієнтована на зберігання документів. В системі управління субтитрами дані здебільшого будуть реляційними, мати багато взаємозв’язків між різними таблицями, тому у цьому випадку MongoDB не є найкращим варіантом.

В результаті було обрано SQLite як технологію бази даних для системи управління субтитрами з використанням штучного інтелекту.

Розглянемо структуру бази даних.

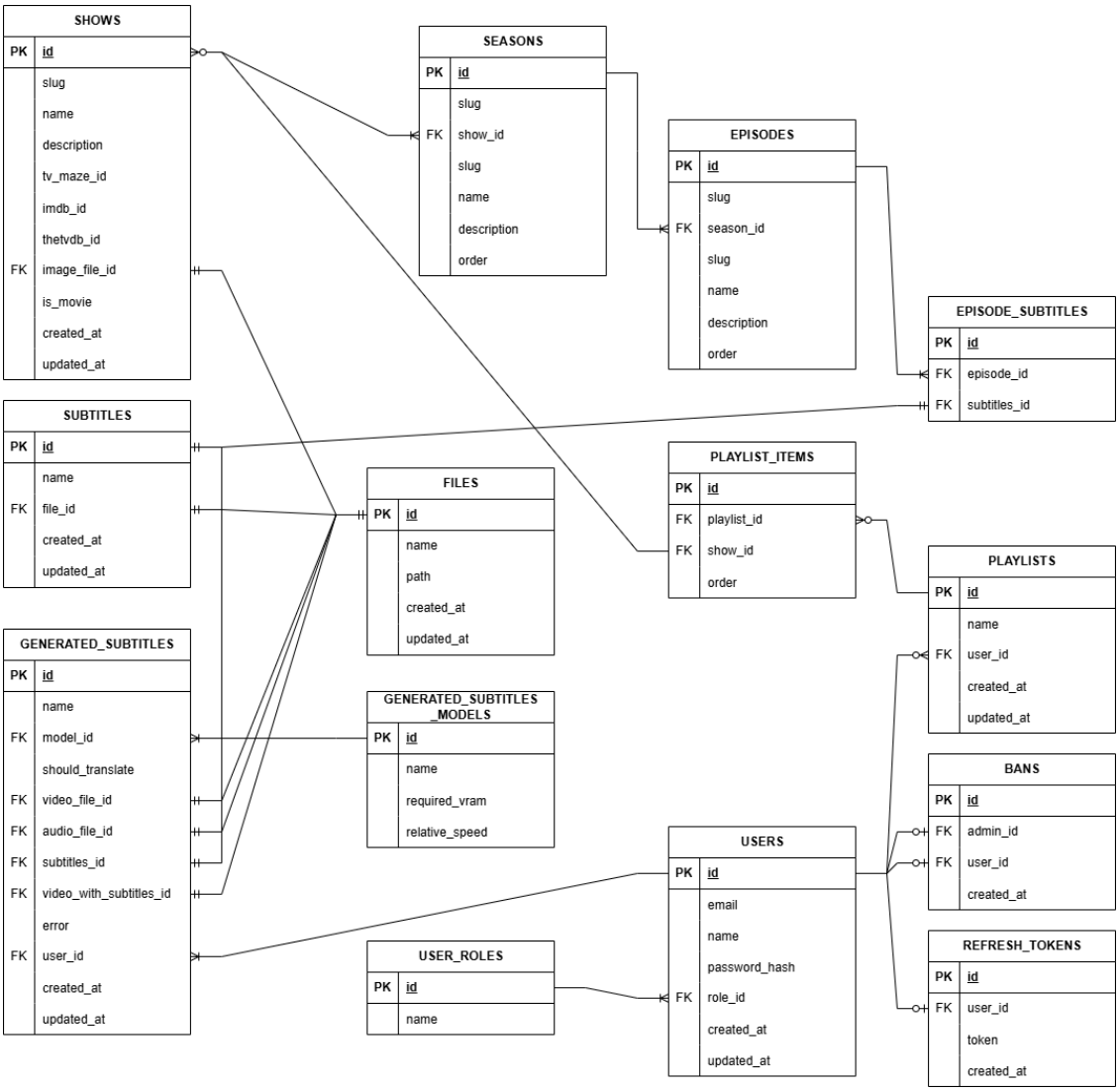

Рисунок 3.1 — Структура бази даних.

База даних складається з наступних таблиць.

Таблиця 3.1 – USER_ROLES (Ролі користувачів).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор ролі користувача
Обов’язкове	name	Назва ролі користувача

В цій таблиці зберігаються дані про різні ролі користувачів.

Таблиця 3.2 – USERS (Користувачі).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор користувача
Обов’язкове	email	Пошта користувача
Обов’язкове	name	Ім’я користувача
Обов’язкове	password_hash	Хеш паролю користувача
Обов’язкове	role_id	Роль користувача
Обов’язкове	created_at	Дата реєстрації
Обов’язкове	updated_at	Дата оновлення

В цій таблиці зберігаються дані про різних користувачів. Кожен користувач має унікальний ідентифікатор, пошту, ім’я, пароль, роль, дату створення та дату оновлення. Хешування паролю зроблено для безпеки даних користувача та системи.

Таблиця 3.3 – REFRESH_TOKENS (Токени оновлення).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор токена
Обов’язкове	user_id	Унікальний ідентифікатор користувача
Обов’язкове	token	Токен
Обов’язкове	created_at	Дата створення токену

Система використовує JWT (JSON Web Token) для авторизації користувачів, тому потрібно мати місце, де можна зберігати ці токени. Всі ці токени зберігаються в цій таблиці. У кожного токена є унікальний ідентифікатор, унікальний ідентифікатор користувача, котрому належить токен, текстовий токен та дата створення.

Таблиця 3.4– BANS (Блокування).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор
Обов’язкове	admin_id	Унікальний ідентифікатор адміністратора
Обов’язкове	user_id	Унікальний ідентифікатор користувача
Обов’язкове	created_at	Дата створення

Система має адміністраторів, котрі слідкують за її роботою. Для того, щоб адміністратори могли виконувати свою роботу їм потрібні певні можливість. Адміністратори мають можливість заблокувати користувачів, щоб підтримувати стабільну роботу системи. Кожне блокування має інформацію про користувача, якого було заблоковано, адміністратора та дату блокування.

Таблиця 3.5 – FILES (Файли).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор файлу

Обов’язкове	name	Назва файлу
Обов’язкове	path	Шлях до файлу
Обов’язкове	created_at	Дата створення
Обов’язкове	updated_at	Дата оновлення

Система працює з великою кількість різний файлів, будь то файли субтитрів або відео, тому потрібно створити таблицю для організації цих даних та створити ефективну системи роботи з файлами. У кожного файлу є свій унікальний ідентифікатор, назва, шлях до файлу у файловій системі, дата створення та дата оновлення.

Таблиця 3.6 – SUBTITLES (Субтитри).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор субтитрів
Обов’язкове	name	Назва субтитрів
Обов’язкове	file_id	Унікальний ідентифікатор файлу субтитрів
Обов’язкове	created_at	Дата створення
Обов’язкове	updated_at	Дата оновлення

Система має ефективно працювати з великою кількістю різних субтитрів, тому потрібно створити окрему таблицю для організації цієї інформації. Кожен рядок в таблиці субтитрів має унікальний ідентифікатор, назву, унікальний ідентифікатор файлу з субтитрами, дату створення та дату оновлення.

Таблиця 3.7 – GENERATED_SUBTITLES_MODEL (Моделі генерації субтитрів).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор
Обов’язкове	name	Назва
Обов’язкове	required_vram	Ресурс віртуальної пам’яті потрібний для роботи
Обов’язкове	relative_speed	Швидкість роботи відносно інших моделей

Для автоматичної генерації субтитрів потрібно вибрати модель за допомогою якої буде відбуватися генерація. Існує відносно широкий вибір моделей, в залежності від потреб користувача. Деякі моделі мають пріоритет на швидкість роботи, а інші на точність результату, а інші намагаються досягти балансу між швидкістю та точністю.

Таблиця 3.8 – GENERATED_SUBTITLES (Згенеровані субтитри).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор згенерованих субтитрів
Обов’язкове	name	Назва
Обов’язкове	model_id	Унікальний ідентифікатор моделі
Обов’язкове	should_translate	Потреба в автоматичному перекладу
Необов’язкове	video_file_id	Унікальний ідентифікатор відео файлу

Необов’язкове	audio_file_id	Унікальний ідентифікатор аудіо файлу
Необов’язкове	subtitles_id	Унікальний ідентифікатор субтитрів
Необов’язкове	video_with_subtitles_file_id	Унікальний ідентифікатор відео з субтитрами файлу
Необов’язкове	error	Текст помилки, яка могла статися під час генерації
Обов’язкове	user_id	Унікальний ідентифікатор користувача
Обов’язкове	created_at	Дата створення
Обов’язкове	updated_at	Дата оновлення

Система має функціонал автоматичної генерації субтитрів за допомогою штучного інтелекту, тому потрібно створити таблицю для зберігання та організації цієї інформації. Кожен рядок в таблиці має унікальний ідентифікатор, назву, модель, потребу в перекладі, унікальний ідентифікатор файлу з відео, з аудіо, з субтитрами, унікальний ідентифікатор користувача, дату створення та оновлення. Також є поле error, в яке можуть потрапити помилки, які можуть статися під час генерації автоматичних субтитрів.

Таблиця 3.9 – SHOWS (Серіали та фільми).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор

Обов’язкове	slug	Частина URL
Обов’язкове	name	Назва
Обов’язкове	description	Опис
Необов’язкове	tv_maze_id	Унікальний ідентифікатор в системі tv maze
Необов’язкове	imdb_id	Унікальний ідентифікатор в системі imdb
Необов’язкове	thetvdb_id	Унікальний ідентифікатор в системі thetvdb
Обов’язкове	image_file_id	Унікальний ідентифікатор файлу з фотографією контенту
Обов’язкове	is_movie	Позначення серіалу або фільму
Обов’язкове	created_at	Дата створення
Обов’язкове	updated_at	Дата оновлення

Система має функціонал роботи з різними видами контенту, будь то серіали чи фільмі, тому потрібно створити таблицю для роботи з такою інформацією. Для більш ефективної роботи та стабільної структури система технічно вважає будь-який фільм за серіал лише з однією серією, що дозволяє значно оптимізувати роботу з базою даних у цьому випадку. Кожен рядок в цій таблиці має унікальний ідентифікатор, slug, позначення фільму чи серіалу, назву, опис, дату створення та оновлення. Також в цій таблиці є необов’язкові колонки, де може зберігатися відношенню контенту до інших систем збору інформації, будь то imdb, tvmaze чи інші.

Таблиця 3.10 – SEASONS (Сезони).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор
Обов’язкове	show_id	Унікальний ідентифікатор контенту
Обов’язкове	slug	Частина URL
Обов’язкове	name	Назва
Обов’язкове	description	Опис
Обов’язкове	order	Номер

У майже кожного популярного серіалу повинно бути багато різних сезонів, тому система повинна мати змогу працювати з такою інформацією. У кожного сезону є унікальний ідентифікатор сезону, унікальний ідентифікатор серіалу, slug, назва, опис, та номер.

Таблиця 3.11 – EPISODES (Епізоди).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор
Обов’язкове	season_id	Унікальний ідентифікатор сезону
Обов’язкове	slug	Частина URL
Обов’язкове	name	Назва
Обов’язкове	description	Опис
Обов’язкове	order	Номер

Сезони не мають дуже великої користі, якщо у них немає епізодів, тому система повинна мати таблицю, де зберігається та організовується інформація про епізоди. Кожен епізод має унікальний ідентифікатор епізоду, унікальний ідентифікатор сезону, slug, назву, опис та номер.

Таблиця 3.12 – EPISODE_SUBTITLES (Субтитри епізодів).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор
Обов’язкове	episode_id	Унікальний ідентифікатор епізоду
Обов’язкове	subtitles_id	Унікальний ідентифікатор субтитрів

Епізоди можуть мати різну кількість різних субтитрів, тому система повинна мати таблицю, де можна організувати таку інформацію. Кожен рядок в таблиці має унікальний ідентифікатор рядку, унікальний ідентифікатор епізоду до якого відносяться субтитри та унікальний ідентифікатор субтитрів.

Таблиця 3.13 – PLAYLISTS (Списки відтворення контенту).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор
Обов’язкове	name	Назва
Обов’язкове	user_id	Унікальний ідентифікатор користувача
Обов’язкове	created_at	Дата створення
Обов’язкове	updated_at	Дата оновлення

Користувачі можуть створювати власні списки відтворення контенту. Ці вибірки контенту доступі лише для користувача, який їх створив. До цього списку користувач може додати будь-який контент, це може бути серіал або

фільм. У кожному рядку є інформація про унікальний ідентифікатор, назву та користувача, дату створення та оновлення.

Таблиця 3.14 – PLAYLIST_ITEMS (Елементи списків відтворення контенту).

Тип поля	Поле	Вміст
Ключове	id	Унікальний ідентифікатор
Обов’язкове	playlist_id	Назва
Обов’язкове	show_id	Унікальний ідентифікатор користувача
Обов’язкове	order	Дата створення

Списки відтворення контенту потребують набору контенту з якого вони складаються. В цій таблиці є інформація про те, який контент належить до якого списку і в якому порядку. Один серіал або фільм може належати одразу до багатьох різних списків.

3.5 Програмна реалізація основних функцій

Розглянемо реалізацію у коді основних функцій системи організації субтитрів.

Реалізація пошуку тексту в субтитрах у коді.

```
    public async search(
    page: number,
    searchRequest: ISubtitlesSearchRequestDto
  ): Promise<ISubtitlesSearchResponseDto> {
    try {
      const db = this.database.getDB();

      const subtitles = await db
        .selectFrom('subtitles')
        .innerJoin('files', 'files.id', 'subtitles.file_id')
```

```
            .innerJoin(
              'episode_subtitles',
              'episode_subtitles.subtitles_id',
              'subtitles.id'
            )
            .innerJoin('episodes', 'episodes.id',
'episode_subtitles.episode_id')
            .innerJoin('seasons', 'seasons.id',
'episodes.season_id')
            .innerJoin('shows', 'shows.id', 'seasons.show_id')
            .select([
              'subtitles.id as subtitles_id',
              'subtitles.name as subtitles_name',
              'files.path',

              'shows.name as show_name',
              'shows.slug as show_slug',
              'seasons.name as season_name',
              'seasons.slug as season_slug',
              'episodes.name as episode_name',
              'episodes.slug as episode_slug',
            ])
            //do not find anything if there is not text
            .$if(searchRequest.text.length === 0, (qb) =>
              qb.where('subtitles.id', 'is', null)
            )
            .$if(searchRequest.show_id.length > 0, (qb) =>
              qb.where('shows.id', '=', searchRequest.show_id)
            )
            .orderBy('shows.name asc')
            .orderBy('seasons.order asc')
            .orderBy('episodes.order asc')
            .orderBy('subtitles.name asc')
            .execute();

          const allItems: ISubtitlesSearchResponseDto['items'] =
[];
          for (const subtitle of subtitles) {
            const fileResult = await searchForTextInFile(
              this.logger,
              searchRequest.text,
              subtitle.path
            );
            if (fileResult.length > 0) {
              allItems.push({
                subtitles_id: subtitle.subtitles_id,
                subtitles_name: subtitle.subtitles_name,
                nodes: [...fileResult],

                name: subtitle.episode_name,
                href:
`/shows/${subtitle.show_slug}/seasons/${subtitle.season_slug}/ep
isodes/${subtitle.episode_slug}`,
```

```
            });
          }
        }

        const itemsPaginated = allItems.slice(
          getPageOffset(page),
          getPageOffset(page + 1)
        );

        return {
          items: itemsPaginated,
          pagination: {
            currentPage: page,
            totalPages: Math.ceil(allItems.length /
PAGE_SIZE),
          },
        };
      } catch (error) {
        this.logger.error(`search: ${error}`);
        throw error;
      }
    }
```

Функція пошуку тексту в субтитрах є асинхронною функцією, яка приймає два параметри: сторінку та об’єкт запиту пошуку. Сторінка це число, яке вказує на якій сторінці пошуку зараз знаходиться користувач. Об’єкт запиту пошуку містить в собі два поля: текст та унікальний ідентифікатор контенту.

Унікальний ідентифікатор контенту є необов’язковим полем, яке може бути заповнено користувачем для того, щоб збільшити швидкість пошуку. У разі заповненого поля унікального ідентифікатору контенту, пошук буде зроблено тільки на епізодах цього контенту, але якщо це поле не буде заповненим, то система буде робити пошук по всьому контенту, що є доступним у системі.

Поле тексту використовується для того, щоб знайти цей текст у файлах субтитрів. Система спочатку отримує всі файли з субтитрами епізодів. Пошук всіх файлів відбувається за допомогою бази даних SQLite. Система генерує запит до бази даних на основі параметрів пошуку користувача. В результаті система отримує масив з усіма потрібними даними. В масиві є не

тільки місце файлу в файловій системі, а ще і інформація про те, до якого серіалу чи фільму він відноситься, його номер та посилання, тощо.

Потім система асинхронно читає субтитри з кожного файлу і робить пошук тексту. Якщо є співпадіння, то система додає їх до результату.

Після пошуку в файлах система робить пагінацію результатів, розділяючи результати на сторінки. Користувач отримує лише дані сторінки, на якій він знаходиться.

Дуже важливою частиною реалізації цієї функції є її асинхронна структура, що дає системі можливість дуже ефективно працювати з великою кількістю файлів.

Реалізація автоматичної генерації субтитрів за допомогою штучного інтелекту у коді.

```
    private async createJob(
        expressFile: IExpressFile,
        generatedSubtitles: IGeneratedSubtitles
      ) {
        const jobID = generatedSubtitles.id;

        try {
          const db = this.database.getDB();
          this.logger.log(
            `Job(${jobID}): Created generated subtitles
${generatedSubtitles.id}`
          );
          const uploadedFile = await stepUploadFile(
            db,
            generatedSubtitles,
            expressFile
          );
          this.logger.log(`Job(${jobID}): Uploaded file
${uploadedFile.path}`);
          const audioFile = await stepGetAudioFromVideo(
            db,
            generatedSubtitles,
            uploadedFile
          );
          this.logger.log(`Job(${jobID}): Generated audio file
${audioFile.path}`);
          const subtitlesFile = await
stepCreateSubtitlesFromAudio(
            db,
            generatedSubtitles,
```

```
            audioFile
          );
          this.logger.log(
            `Job(${jobID}): Generated subtitles file
${subtitlesFile.path}`
          );

          const videoWithSubtitlesFile = await
stepBurnSubtitlesIntoVideo(
            db,
            generatedSubtitles,
            uploadedFile,
            subtitlesFile
          );
          this.logger.log(
            `Job(${jobID}): Generated video with subtitles file
${videoWithSubtitlesFile.path}`
          );
        } catch (error) {
          this.logger.error(`createJob(${jobID}): ${error}`);

          const db = this.database.getDB();
          await db
            .updateTable('generated_subtitles')
            .set({ error: error?.toString() })
            .where('generated_subtitles.id', '=',
generatedSubtitles.id)
            .execute();

          throw error;
        }
      }
```

Реалізація функції автоматичної генерації субтитрів є асинхронною функцією, яка приймає декілька параметрів: відео файл та об’єкт згенерованих субтитрів. Файл містить в собі відео для якого користувач хоче згенерувати субтитри. Об’єкт генерованих субтитрів - це об’єкт, який був підготовлений для роботи з таблицею генерованих субтитрів у базі даних.

Функція генерація субтитрів розділена на декілька важливих кроків. Спочатку система зберігає файл для подальшої обробки за допомогою асинхронної stepUploadFile функції.

```
export const stepUploadFile = async (
  db: IDatabase,
  generatedSubtitles: IGeneratedSubtitles,
  expressFile: IExpressFile
```

```
    ): Promise<IDbFile> => {
      const nowSQL = getNowSQL();

      const uploadedFile: IDbFile = {
        id: randomUUID(),
        name: expressFile.originalname,
        path: path.join(
          'generated',
          generatedSubtitles.id,
          expressFile.originalname
        ),
        created_at: nowSQL,
        updated_at: nowSQL,
      };

      await fs.mkdir(
        path.join(process.cwd(), 'storage', 'generated',
generatedSubtitles.id),
        {
          recursive: true,
        }
      );
      await fs.writeFile(
        path.join(process.cwd(), 'storage', uploadedFile.path),
        expressFile.buffer
      );

      await
db.insertInto('files').values(uploadedFile).executeTakeFirstOrTh
row();

      await db
        .updateTable('generated_subtitles')
        .set({ video_file_id: uploadedFile.id, updated_at:
getNowSQL() })
        .where('generated_subtitles.id', '=',
generatedSubtitles.id)
        .executeTakeFirstOrThrow();

      return uploadedFile;
    };
```

Функція створює об’єкти файлів та субтитрів для подальшої роботи з базою даних та зберігає файл до файлової системи. Після цього система генерує аудіо файл на основі відео за допомогою асинхронної stepGetAudioFromVideo функції.

```
export const stepGetAudioFromVideo = async (
  db: IDatabase,
  generatedSubtitles: IGeneratedSubtitles,
  videoFile: IDbFile
): Promise<IDbFile> => {
  const audioFileName = `${path.basename(
    videoFile.name,
    path.extname(videoFile.name)
  )}.mp3`;

  await new Promise((resolve, reject) => {
    FFMPEG(path.join(process.cwd(), 'storage',
videoFile.path))
      .outputOptions('-vn')
      .audioCodec('libmp3lame')
      .save(
        path.join(
          process.cwd(),
          'storage',
          'generated',
          generatedSubtitles.id,
          audioFileName
        )
      )
      .on('end', resolve)
      .on('error', reject);
  });

  const audioFile: IDbFile = {
    id: randomUUID(),
    name: audioFileName,
    path: path.join('generated', generatedSubtitles.id,
audioFileName),
    created_at: getNowSQL(),
    updated_at: getNowSQL(),
  };

  await
db.insertInto('files').values(audioFile).executeTakeFirstOrThrow
();

  await db
    .updateTable('generated_subtitles')
    .set({ audio_file_id: audioFile.id, updated_at:
getNowSQL() })
    .where('generated_subtitles.id', '=',
generatedSubtitles.id)
    .executeTakeFirstOrThrow();

  return audioFile;
};
```

Система генерує файл аудіо на основі відео файлу за допомогою FFmpeg. FFmpeg є стандартом для роботи з відео та аудіо файлами та підтримує велику кількість різних форматів цим файлів, будь то mp4, mp3, waw, тощо. Також ця функція заносить дані про новий аудіо файл до бази даних. Потім система генерує субтитри на основі аудіо файлу за допомогою асинхронної stepCreateSubtitlesFromAudio функції.

```
export const stepCreateSubtitlesFromAudio = async (
  db: IDatabase,
  generatedSubtitles: IGeneratedSubtitles,
  audioFile: IDbFile
): Promise<IDbFile> => {
  const selectedGeneratationModel = await db
    .selectFrom('generated_subtitles_models')
    .select('generated_subtitles_models.name')
    .where('generated_subtitles_models.id', '=',
generatedSubtitles.model_id)
    .executeTakeFirst();
  const model_name = selectedGeneratationModel.name;

  const subtitleName = `${path.basename(
    generatedSubtitles.name,
    path.extname(generatedSubtitles.name)
  )}.srt`;

  const fullDirPath = path.join(
    process.cwd(),
    'storage',
    'generated',
    generatedSubtitles.id
  );

  const translatePart =
    generatedSubtitles.should_translate === SQLITE_BOOL.true
      ? `--task translate`
      : ``;
  const whispererCommand = `cd ${fullDirPath} && whisper
"${audioFile.name}" --model ${model_name} ${translatePart} --
output_format srt`;

  await new Promise((resolve, reject) => {
    exec(whispererCommand, (error, stdout, stderr) => {
      if (error) {
        reject(error);
      }

      resolve(stdout);
    });
```

```
      });

      const subtitlesFile: IDbFile = {
        id: randomUUID(),
        name: subtitleName,
        path: path.join('generated', generatedSubtitles.id,
subtitleName),
        created_at: getNowSQL(),
        updated_at: getNowSQL(),
      };

      await
db.insertInto('files').values(subtitlesFile).execute();

      const subtitles: ISubtitles = {
        id: randomUUID(),
        name: subtitleName,
        file_id: subtitlesFile.id,
        created_at: getNowSQL(),
        updated_at: getNowSQL(),
      };

      await
db.insertInto('subtitles').values(subtitles).execute();

      await db
        .updateTable('generated_subtitles')
        .set({ subtitles_id: subtitles.id, updated_at:
getNowSQL() })
        .where('generated_subtitles.id', '=',
generatedSubtitles.id)
        .execute();

      return subtitlesFile;
    };
```

Система генерує команду для штучного інтелекту на основі даних, що вибрав користувач. Після генерації субтитрів система записує дані до файлової системи та бази даних для подальшої роботи. Після цього система генерує відео з вбудованими субтитрами за допомогою асинхронної stepBurnSubtitlesIntoVideo функції.

```
    export const stepBurnSubtitlesIntoVideo = async (
      db: IDatabase,
      generatedSubtitles: IGeneratedSubtitles,
      videoFile: IDbFile,
      subtitlesFile: IDbFile
    ): Promise<IDbFile> => {
      const outputVideoName = `${path.basename(
```

```
        videoFile.name,
        path.extname(videoFile.name)
      )}-with-subtitles.mp4`;

      const outputVideoFile: IDbFile = {
        id: randomUUID(),
        name: outputVideoName,
        path: path.join('generated', generatedSubtitles.id,
outputVideoName),
        created_at: getNowSQL(),
        updated_at: getNowSQL(),
      };

      const fullVideoPath = path.join(process.cwd(), 'storage',
videoFile.path);
      const fullSubtitlesPath = path.join(
        process.cwd(),
        'storage',
        subtitlesFile.path
      );
      const fullOutputPath = path.join(
        process.cwd(),
        'storage',
        outputVideoFile.path
      );

      const command = `ffmpeg -i "${fullVideoPath}" -vf
subtitles="${fullSubtitlesPath}" "${fullOutputPath}"`;

      await new Promise((resolve, reject) => {
        exec(command, (error, stdout, stderr) => {
          if (error) {
            reject(error);
            return;
          }

          resolve(stdout || '');
        });
      });

      await
db.insertInto('files').values(outputVideoFile).execute();

      await db
        .updateTable('generated_subtitles')
        .set({
          video_with_subtitles_file_id: outputVideoFile.id,
          updated_at: getNowSQL(),
        })
        .where('id', '=', generatedSubtitles.id)
        .execute();

      return outputVideoFile;
```

```
};
```

Система знову використовує FFmpeg, але цього разу для додавання субтитрів до відео файлу. В результаті створюється новий відео файл з вбудованими субтитрами. Також функція оновлює базу даних новою інформацію про нове відео.

Функція автоматичної генерації за допомогою штучного інтелекту і її кроки є асинхронними функціями, щоб збільшити продуктивність системи та використати переваги асинхронних функцій для роботи з файлами NodeJS.

3.6 Методика роботи користувача

Головна сторінка має велику кількість важливих посилань для користувача.

Неавторизовані користувачі одразу бачать повідомлення про те, що вони можуть авторизуватися в системі і отримати додатковий функціонал.

Зверху сторінки є блок пошуку тексту в субтитрах за допомогою якого можна перейти на сторінку пошуку.

Нижче є важливий блок з якого можна перейти на сторінку автоматичної генерації субтитрів.

Після цього на сторінці є блоки серіалів і фільмів з можливістю перейти на цікаву користувачу сторінку і дізнатися більше.

Внизу сторінки в футері також є корисні посилання: на мапу сайту, головну сторінку, сторінку серіалів, фільмів та інші.

3.6.1 Гість

Гість – користувач, який не є авторизованим в системі. Одразу на головній сторінці гість бачить повідомлення про те, що він може отримати доступ до додаткового функціоналу, якщо він авторизується в системі. Головна сторінка для неавторизованого користувача виглядає наступним чином.

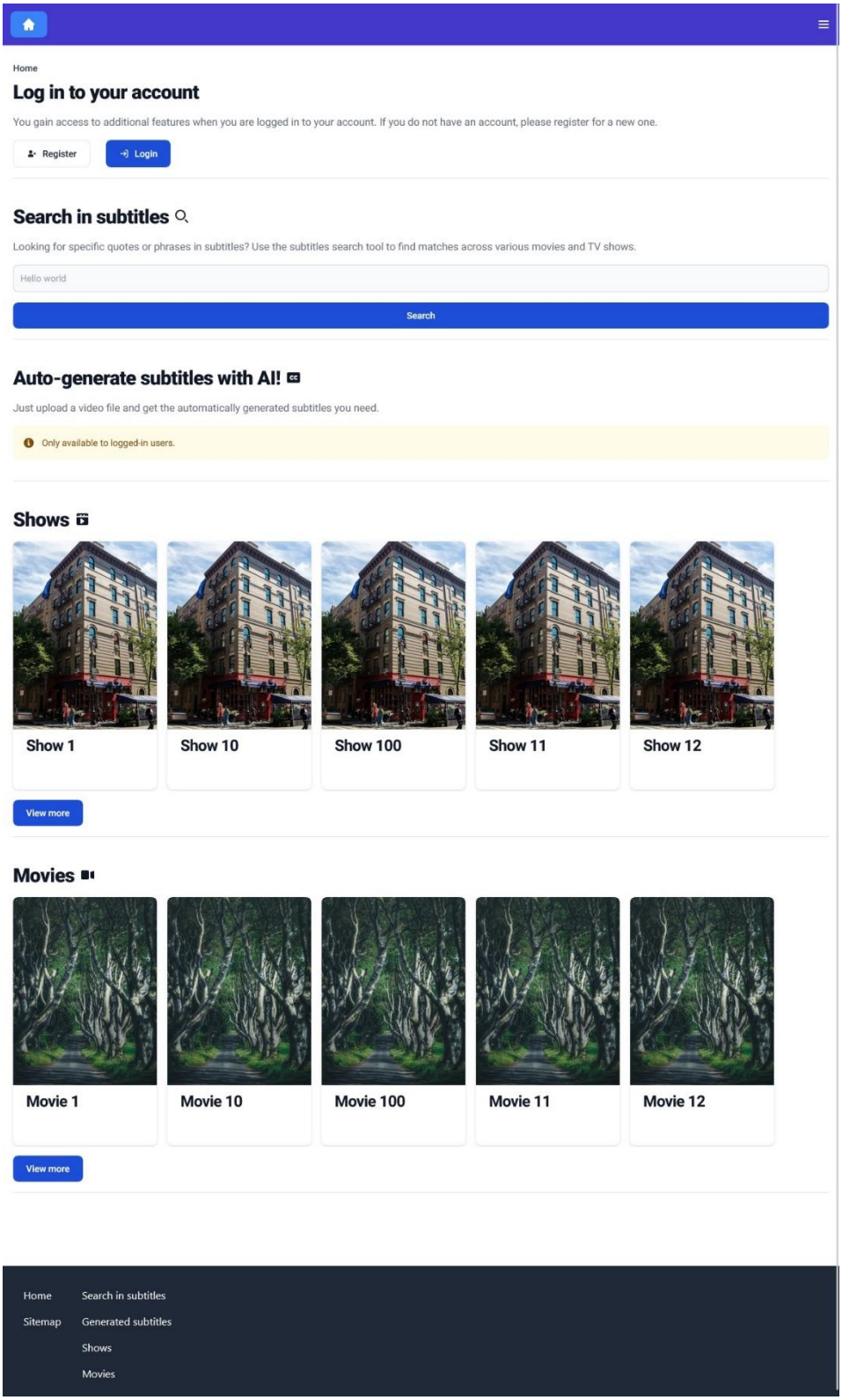

Рисунок 3.1 — Головна сторінка для неавторизованого користувача

Перше, що бачить гість – повідомлення про те, що йому потрібно авторизуватися в системі. Це повідомлення каже, що у авторизованих користувачів є доступ до додаткового функціоналу. Також є корисні

посилання на реєстрацію та авторизацію в системі. Якщо користувач вперше зайшов до системи, то йому спочатку потрібно перейти на сторінку реєстрації в системі і зареєструватися, а вже потім авторизуватися в системі.

Розглянемо хід дій на сторінці реєстрації в системі.

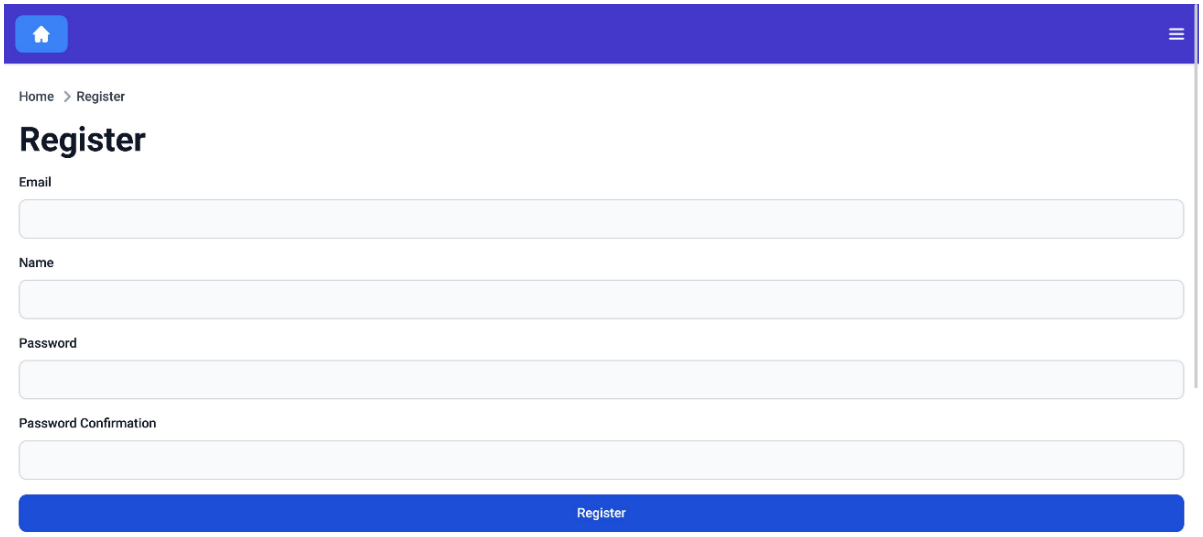

Рисунок 3.2 — Сторінка реєстрації користувача

На цій сторінці користувачу потрібно заповнити форму, щоб створити власний обліковий запис в системі. Для цього потрібно заповнити всі поля форми.

Якщо користувач зробить помилку під час заповнення форми, система повідомить його про це. Наприклад, якщо користувач не заповнить одне з обов’язкових полів форми – він побачить повідомлення, що воно є обов’язковим для заповнення. Якщо в системі вже існує користувач з введеною користувачем електронною поштою, то користувач побачить повідомлення, що в системі вже є зареєстрованим користувач з такою електронною поштою.

У разі успішної реєстрації користувач зможе перейти на сторінку авторизації.

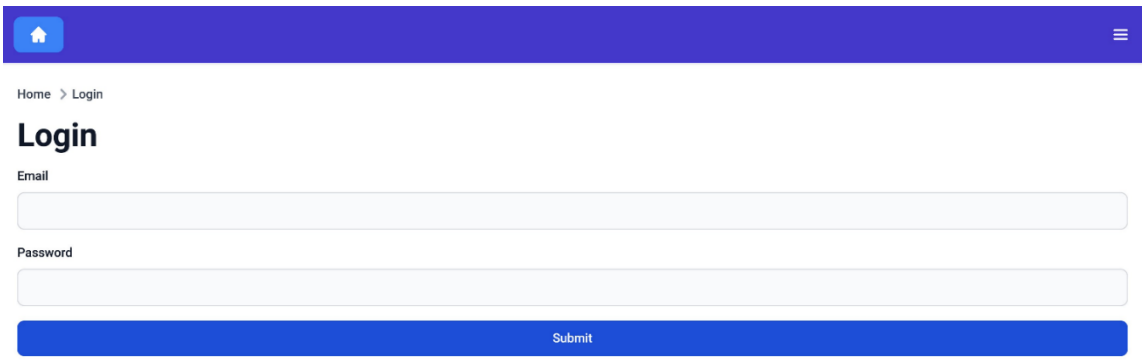

Рисунок 3.3 — Сторінка авторизації

На цій сторінці користувачу потрібно заповнити форму на основні раніше введених даних на сторінці реєстрації.

Якщо користувач зробить помилку під час заповнення форми, то система повідомить його про це.

У разі успішної авторизації користувач потрапить на головну сторінку.

Користувач може перейти на сторінку пошуку тексту у субтитрах з головної сторінки.

Розглянемо хід дій на сторінці пошуку тексту в субтитрах.

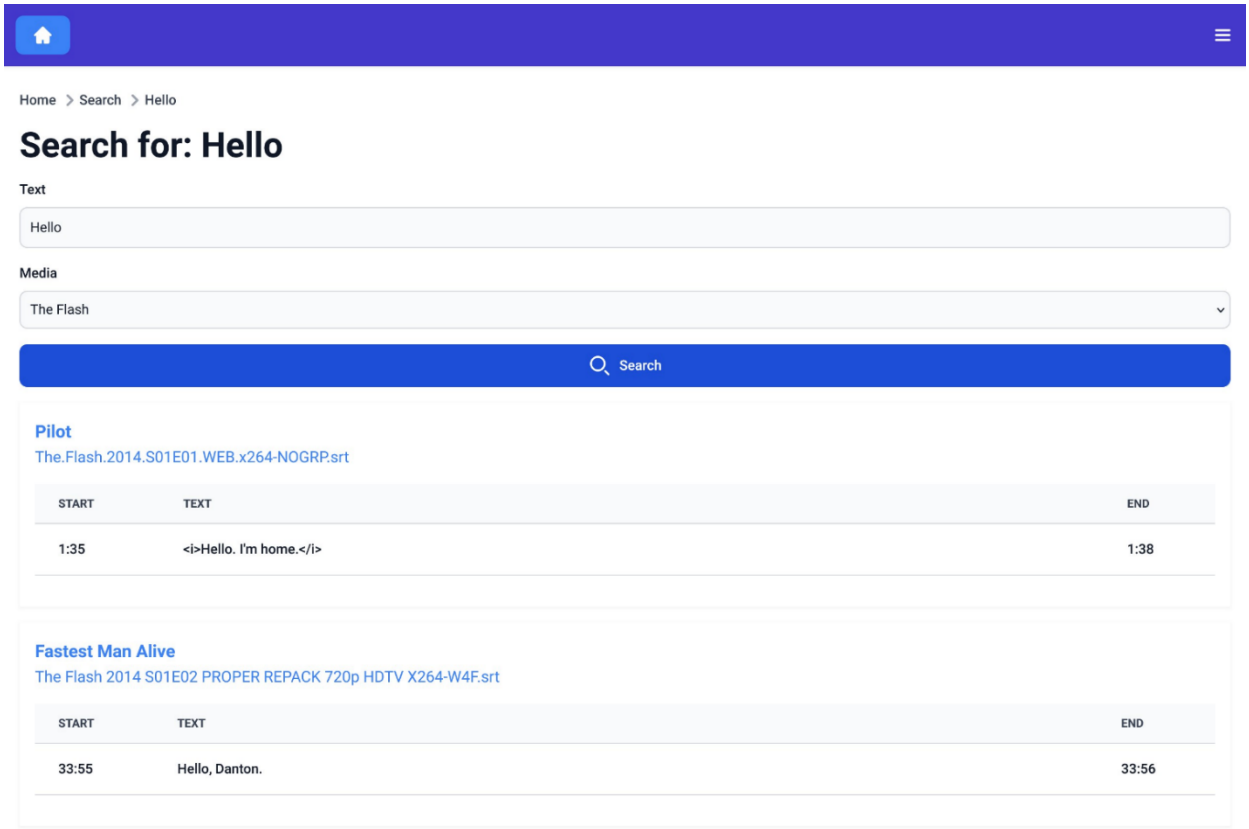

Рисунок 3.4 — Сторінка пошуку тексту в субтитрах

На цій сторінці користувач може зробити пошук тексту в субтитрах, це може бути одне слово, фраза, або навіть ціле речення.

Для пошуку потрібно заповнити форму. Головним полем форми є текст для пошуку. Після заповнення тексту користувач може натиснути кнопку пошуку та система почне шукати цей текст у всій файлах субтитрів, які є у системі. Під час того, як система робить пошук, користувач буде бачити повідомлення про те, що система зараз знаходиться в процесі пошуку і йому потрібно почекати.

Після того, як система закінчить свої розрахунки, користувач побачить результат пошуку.

В результаті можна побачити всі файли субтитрів, котрі підходять під критерії пошуку. Користувач може побачити зверху назву фільму або серіалу в якому було знайдено результат, назва також є посиланням на сторінку цього контенту. Під назвою знаходиться посилання на конкретний файл субтитрів в якому було знайдено співпадіння. У одного фільми або серіалу

може бути декілька різних файлів субтитрів, тому система чітко показує який файл субтитрів було використано. Під назвою файлу субтитрів користувач може побачити витяг з файлу з субтитрами, де було знайдено співпадіння. Цей витяг містять рядки субтитрів з текстом та часом.

Результат пошуку може мати декілька сторінок, тому знизу сторінки користувач може перейти на наступну сторінку результату, або повернутися на попередню сторінку результату.

Цей функціонал пошуку тексту в субтитрах є дуже корисним для користувачів. Система переглядає всі файли субтитрів для того, щоб знайти відповідь для користувача. Перегляд великої кількості файлів може зайняти деякий час, тому система має спосіб оптимізації пошуку субтитрів.

Оптимізація пошуку полягає у тому, щоб обмежити кількість файлів, котрі потрібно обробити системі. У системі може бути велика кількість різних серіалів, але, наприклад, користувач хоче знайти в якій серії серіалу було вживано конкретне слово і користувач впевнений в якому серіалі це було. У такому випадку користувач може вибрати лише цей серіал у формі пошуку та система буде шукати співпадіння лише у файлах субтитрів цього серіалу. Це може зробити пошук значно швидшим, адже замість того, щоб переглядати, наприклад, 1000 різних файлів з різних серіалів, система переглядає лише 100 файлів з одного серіалу. У результаті користувач отримує правильну відповідь значно швидше. Звісно, не кожен користувач буде знайти в якому контенті він хоче зробити пошук, тому це поле в формі не є обов’язковим.

Сторінка перегляду субтитрів виглядає наступним чином.

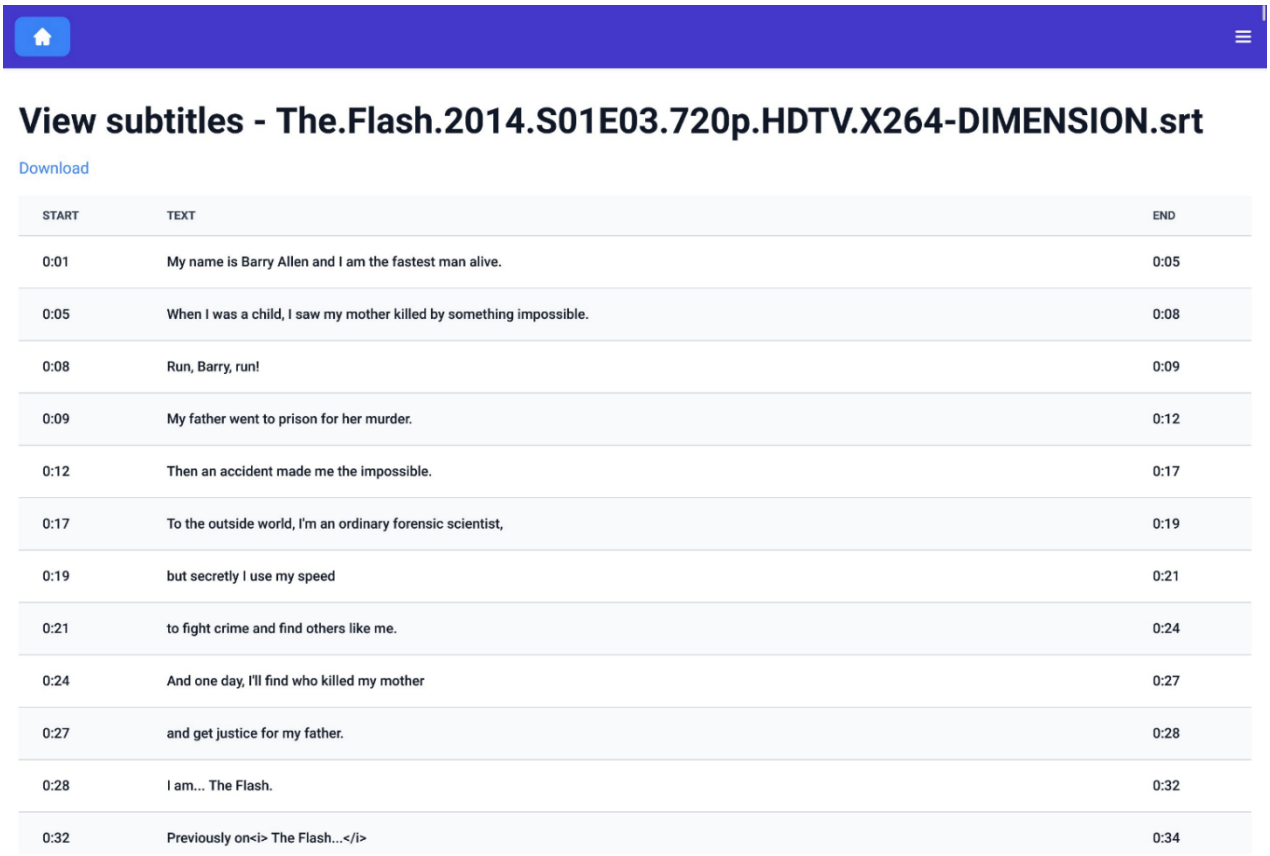

Рисунок 3.5 — Сторінка перегляду субтитрів

На цій сторінці користувач може переглянути інформацію про субтитри. Спочатку можна побачити назву файлів субтитрів. Нижче знаходиться посилання на завантаження файлу, якщо користувач захоче завантажити цей файл субтитрів на свій комп’ютер. Потім знаходиться зміст файлу субтитрів, рядки тексту з позначенням часу для кожного рядку.

У неавторизованого користувача є доступ до перегляду інформації про фільми. Сторінка фільмів виглядає наступним чином.

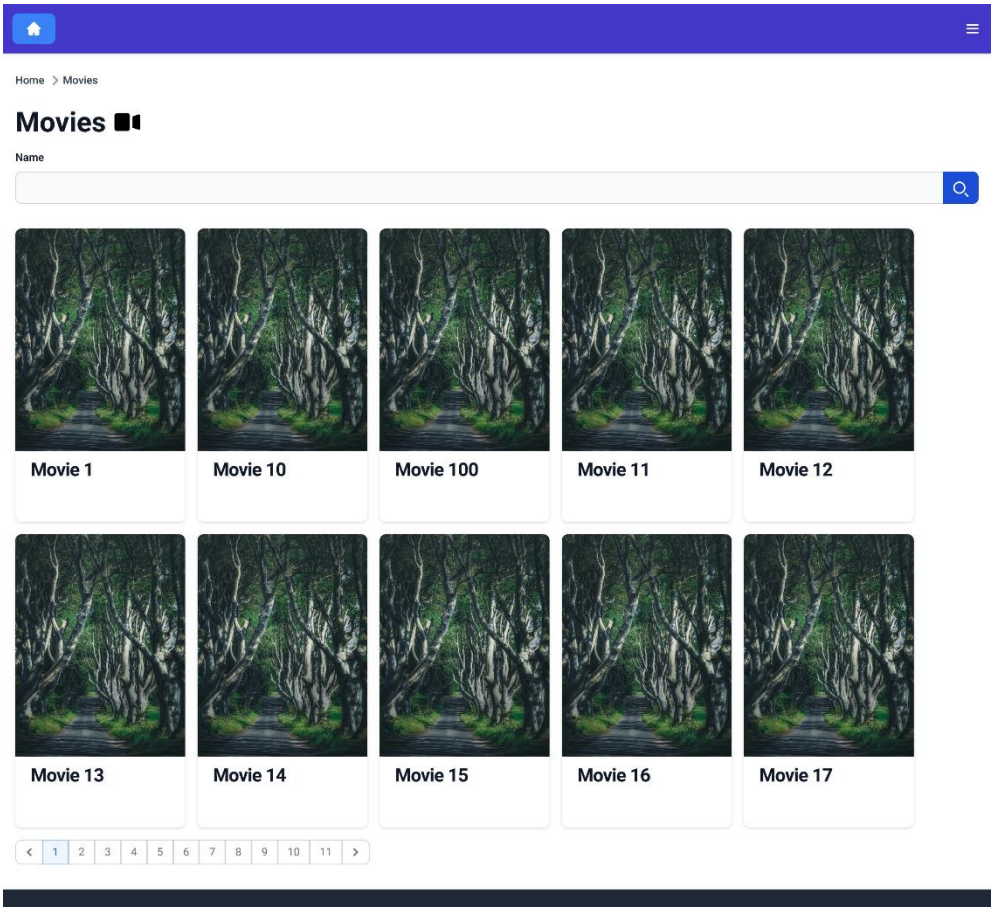

Рисунок 3.6 — Сторінка фільмів

На цій сторінці користувач може знайти всі фільмі, які находяться в системі. Кожен фільм на цій сторінці має назву та фото, щоб перейти на сторінку фільму потрібно просто натиснути на його назву.

Є функціонал пошуку фільму за назвою, для цього потрібно ввести назву фільмі в поле та натиснути кнопку пошуку, після цього система зробить пошук та покаже користувачу результат.

В системі може бути велика кількість різних фільмів, тому знизу сторінки є посилання на різні сторінки, першу, другу та інші.

Сторінка одного фільму виглядає наступним чином.

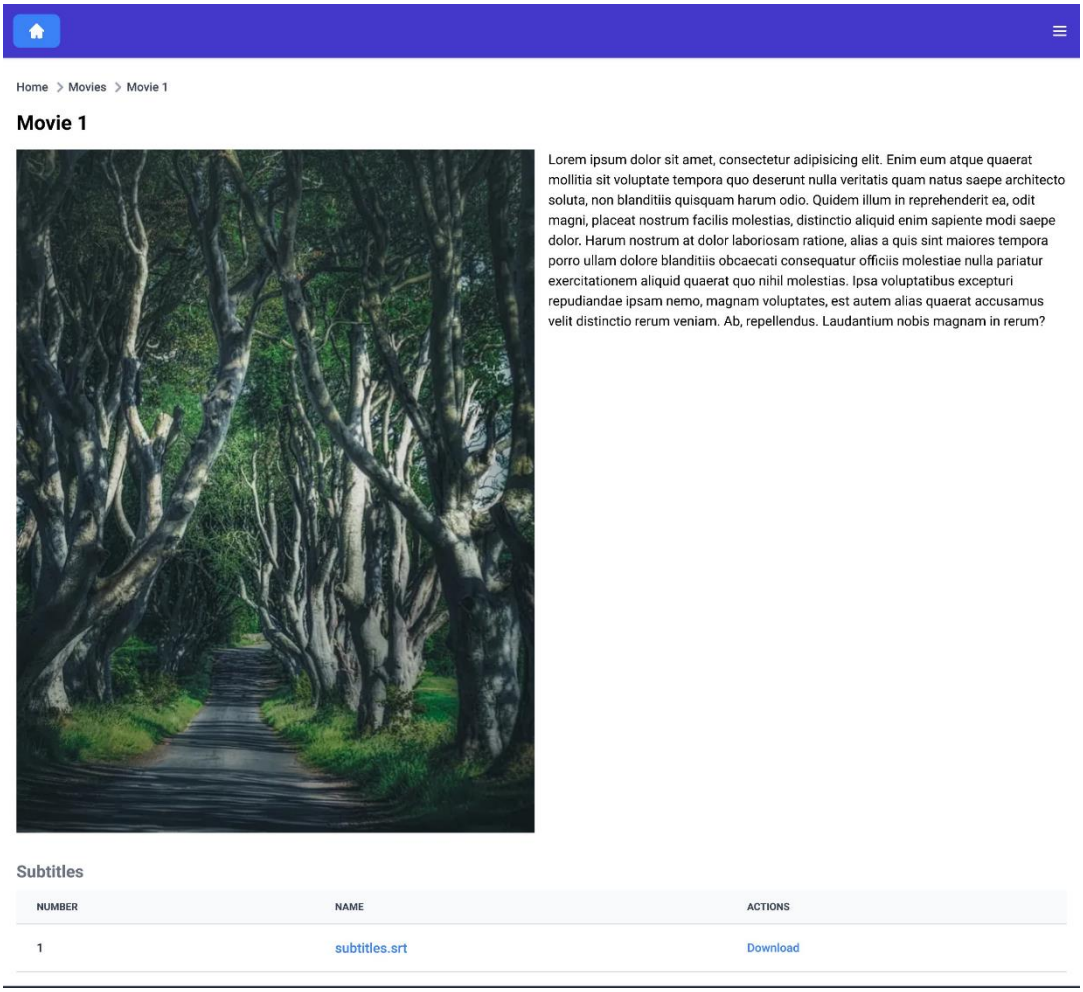

Рисунок 3.7 — Сторінка фільму

На цій сторінці користувач може побачити інформацію про фільм. Зверху можна побачити назву фільму. Нижче знаходиться блок с головним фото фільму та його описом. Знизу знаходиться список всіх файлів субтитрів, які цей фільм має. Якщо користувач хоче, то він може перейти на окрему сторінку та переглянути зміст цього файлу субтитрів або завантажити його на свій комп’ютер.

Розглянемо сторінку серіалів.

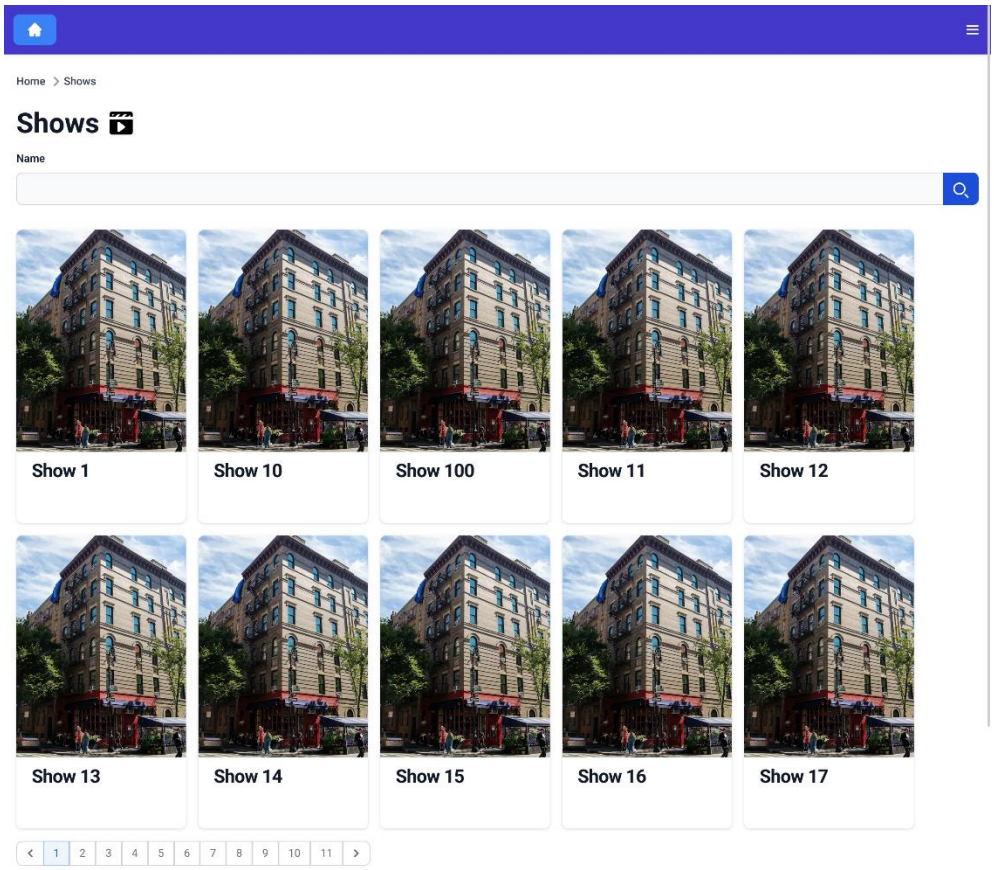

Рисунок 3.8 — Сторінка серіалів

Користувач може знайти всі серіали, які є в системі, на цій сторінці. Зверху сторінки є форма за допомогою якої можна зробити пошук серіалу за назвою. Користувачу просто потрібно написати назву та натиснути на кнопку пошуку. У кожного серіалу є назва та головне фото. Знизу сторінки є посилання на різні сторінки пошуку, якщо користувач не знайшов потрібний йому серіал на поточній сторінці.

Розглянемо сторінку серіалу.

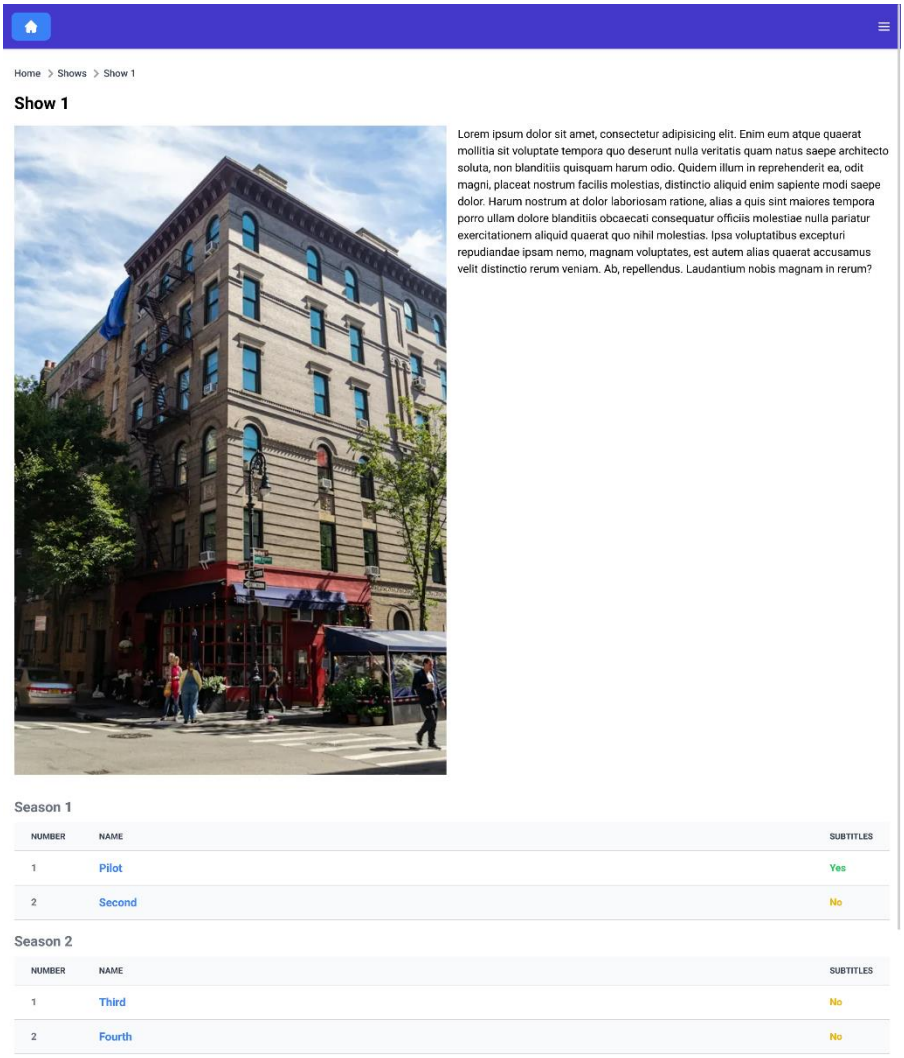

Рисунок 3.9 — Сторінка серіалу

На сторінці серіалу користувач може побачити інформацію про серіал, яка зберігається в системі.

Спочатку на сторінці є назва серіалу. Нижче знаходиться головне фото та опис. Після цього йде список сезонів та серій. У кожного сезону може бути від однієї до багатьох серій, у кожної серії є свій номер, назва та можуть бути різні субтитри. Щоб перейти на окрему сторінку серії серіалу потрібно перейти за посиланням в назві серії.

Розглянемо сторінку серіалу.

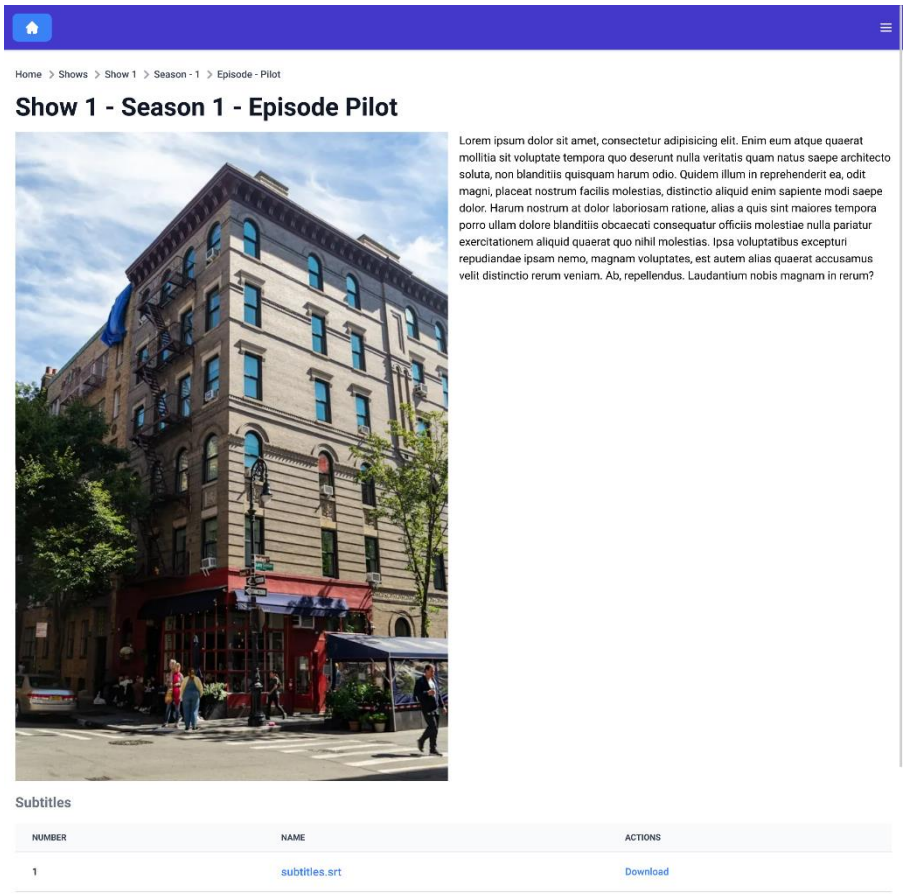

Рисунок 3.10 — Сторінка серії серіалу

На цій сторінці можна побачити назву серії, фото та опис серії. Нижче знаходиться таблиця зі списком субтитрів, які є прив’язані до цієї серії. У кожного рядку в таблиці є номер, посилання на сторінку перегляду субтитрів та посилання на завантаження субтитрів.

3.6.2 Авторизований користувач

Авторизований користувач – користувач, який є авторизованим в системі. Це означає, що у нього є власний аккаунт в системі. Користувач може управляти своїм акаунтом на сторінці профілю.

Сторінка управління профілем виглядає наступним чином.

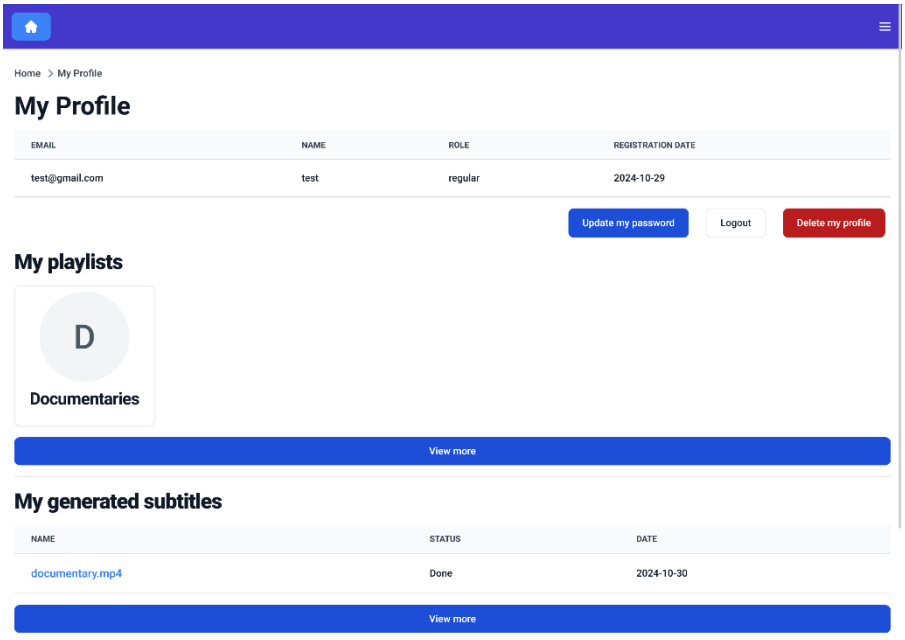

Рисунок 3.11 — Сторінка профілю користувача

На цій сторінці користувач може побачити свою інформацію, а також змінити деякі дані. В таблиці можна побачити електронну пошту користувача, його ім’я, роль та дату реєстрації.

Є функціонал виходу з акаунту, для цього просто потрібно натиснути на кнопку та підтвердити свою дію. Після цього користувач отримає статус неавторизованого та потрапить на головну сторінку.

Користувач має можливість видалити свій акаунт з системи. Для цього користувачі потрібно натиснути на кнопку видалення акаунта, після чого система ще раз спитає користувача чи впевнений він. Якщо користувач вибере, що він не впевнений, то дія буде відмінена. У разі, якщо користувач впевнений в тому, що він хоче видалити свій акаунт, система видалить акаунт користувача та перенесе користувача на головну сторінку.

Користувач може оновити свій пароль на сторінці оновлення паролю.

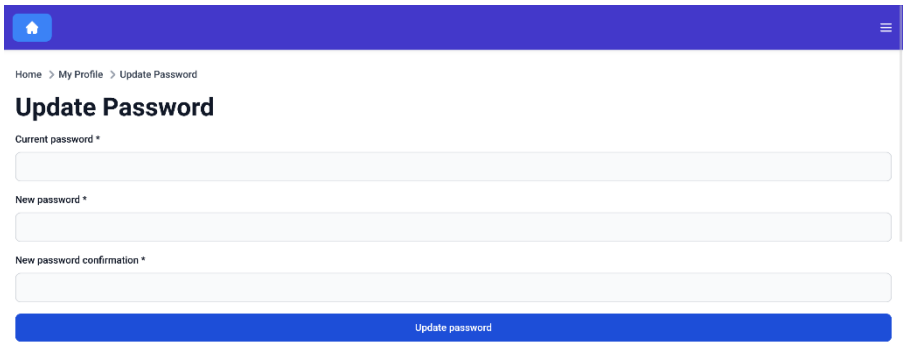

Рисунок 3.12 — Сторінка оновлення паролю

Для оновлення паролю користувач повинен написати свій пароль, новий пароль та підтвердження нового паролю. Підтвердження нового паролю потрібно для того, щоб вберегти користувача від випадкового натискання неправильної літери на клавіатурі, у результаті чого користувач може змінити пароль на той пароль, якого він не знає. Суть перевірки підтвердження нового паролю полягає в тому, щоб користувач два рази написав новий пароль, у такому випадку шанси того, що у паролі є помилки дуже сильно зменшуються. Після написання нових паролів потрібно натиснути на кнопку оновлення пароля та, якщо користувач ввів всі дані вірно, його буде повідомлено про успішне оновлення паролю. Якщо користувач зробив помилку у вхідних даних, то йому буде повідомлено про це.

Однією з найбільших переваг авторизованого користувача є можливість автоматично генерувати субтитри. Для цього потрібно перейти на сторінку автоматичної генерації субтитрів.

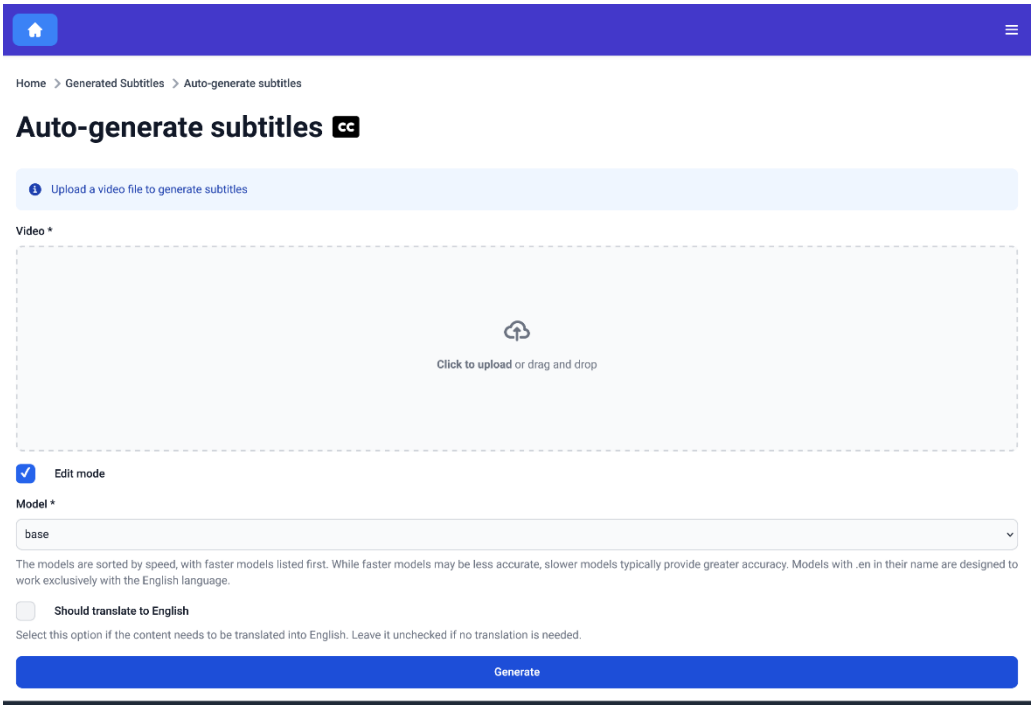

Рисунок 3.13 — Сторінка генерації субтитрів

На цій сторінці можна побачити головні поля форми, котрі відповідають за налаштування генерації. Головним полем, звісно, є поле завантаження відеофайлу. Користувач повинен завантажити відео зі своєї файлової системи у цю форму. Після успішного завантаження користувач може побачити і переглянути своє відео на цій сторінці.

Також в формі є додаткові налаштування для користувачів, які хочуть більше контролю над процесом автоматичної генерації субтитрів.

Важливим полем є вибір моделі генерації. На вибір є широкий вибір моделей: tiny, tiny.en, base, base.en, medium, medium.en, тощо. Модель генерації впливає на точність субтитрів, швидкість їх генерації, ресурси потрібні для генерації. Моделі відсортовані в порядку від найшвидшої до найповільнішої. Моделі з .en в назві працюють лише з англійською мовою. Моделі є на спектрі від швидкості до точності. Швидкі моделі віддають перевагу швидкості роботи, а інші моделі віддають перевагу точності, але через це вони потребують більших ресурсів. Це не означає, що швидкі моделі погані, здебільшого вони мають досить непогані результати, але вони все ж

таки можуть частіше робити помилки в порівнянні з моделями, котрі налаштовані на більшу точність.

Система може генерувати субтитри на багатьох різних мовах. Це може бути дуже корисно для вивчення різних мов. Користувач може одразу завантажите потрібне йому відео в системи і отримати автоматично згенеровані субтитри на цій мові. Також існує можливість автоматично перевести субтитри з іншою мови на англійську. Ця опція може бути дуже корисною для користувачів, які тільки почали вивчати мову, тому ще не можуть дивитися відео тільки з субтитрами на цій мові. В такому випадку вони можуть генерувати субтитри з автоматичним перекладом на англійську мову. Це дозволить користувачу поступову занурюватися у нову мову, спочатку слухаючи мову з англійськими субтитрами, а вже потім використовувати субтитри вже без автоматичного перекладу на англійську мову.

Після заповнення всієї потрібної інформації у формі користувачу потрібно натиснути на кнопку початку генерації. Після цього він потрапить на нову сторінку.

Рисунок 3.14 — Сторінка генерації субтитрів в роботі

На цій сторінці користувач може слідкувати за різними етапами генерації субтитрів. Автоматична генерація субтитрів за допомогою штучного інтелекту займає деякий час, тому ця сторінка була створена, щоб повідомити користувачу на якому етапі зараз знаходиться генерація субтитрів. Існує загальні чотири етапи для користувача: завантаження відео,

відокремлення аудіо файлу, генерація файлу субтитрів за допомогою штучного інтелекту, створення відео з вбудованими субтитрами. Користувачу не потрібно постійно оновлювати сторінку, щоб слідувати за процесом роботи, сторінка автоматично оновлюється та слідкує за етапами роботи. Після того, як система завершить фінальний етап, користувач побачить фінальний результат.

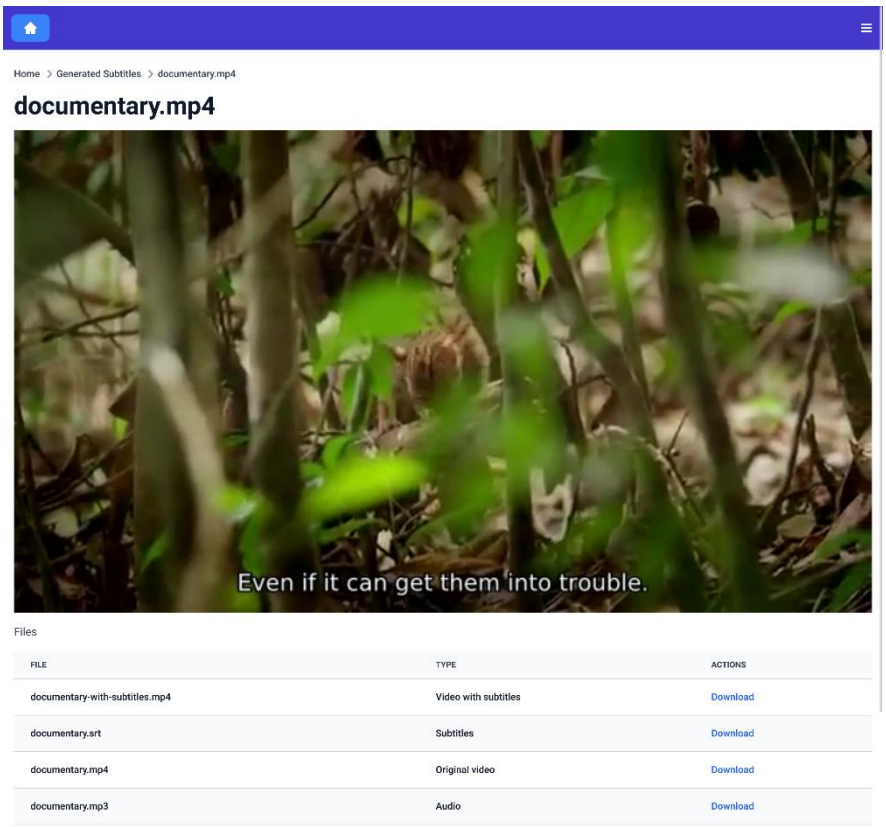

Рисунок 3.15 — Сторінка згенерованих субтитрів

Найголовніше на цій сторінці це - відео з вбудованими автоматичними субтитрами. Користувач може прямо в браузері почати перегляд відео з субтитрами.

Також на сторінці буде таблиця з додатковими файлами для завантаження. Користувач зможе завантажити на свій комп’ютер відео з вбудованими субтитрами, оригінальне відео, або окремий файл з субтитрами. Опція окремого файлу з субтитрами є дуже корисною, тому ще з нею можна

завантажити цей файл на власний комп’ютер і додати його до майже будь-якого сучасного медіа плеєра.

Автоматичні генерації субтитрів зберігаються в історії генерацій клієнта і користувач може в будь-яку мить переглянути ці дані. Це означає, що якщо користувач випадково виключить комп’ютер, то він не втратить даних автоматичної генерації субтитрів. Він зможе відкрити історію і знову переглянути всі потрібні йому дані.

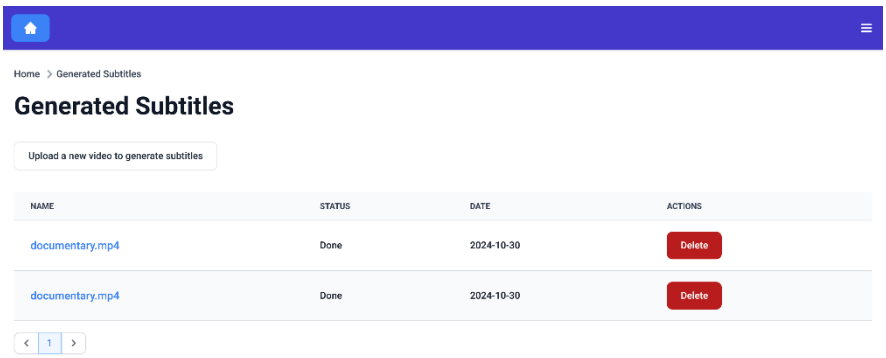

Рисунок 3.16 — Сторінка історії згенерованих субтитрів

В історії користувач може побачити коли були згенеровані субтитри, їх назву та статус. Також в нього є можливість видалити автоматично згенеровані субтитри з історії, якщо вони йому більше не потрібні.

Можливість створювати власні вибірки та колекції контенту також є частиною функціоналу, яку мають авторизовані користувачі. Користувачі можуть відкрити сторінку власних колекцій. Вибравши потрібну колекції користувач переходить на сторінку окремої колекції, де він може побачити повний список контенту який належить до цієї колекції. Колекція може містити в собі як фільмі, так і різні серіали, тобто різні види контенту. Якщо користувач передумає, то він має можливість видалити конкретний контент з колекції або видалити колекцію повністю.

Щоб додати контент до колекції потрібно зайти на сторінку контенту, це може бути фільм або серіал, та натиснути на кнопку додавання до колекції. Після цього потрібно вибрати до якої колекції потрібно додати цей

контент і натиснути кнопку підтвердження. Після цього контент буде успішно додано до колекції.

Контент в систему додають авторизовані користувачі та адміністратори. Система підтримує два основні види контенту: фільми та серіали.

На головній сторінці фільмів є кнопка додання нового фільму до систему. Після неї користувач потрапляє на сторінку додання фільму до системи.

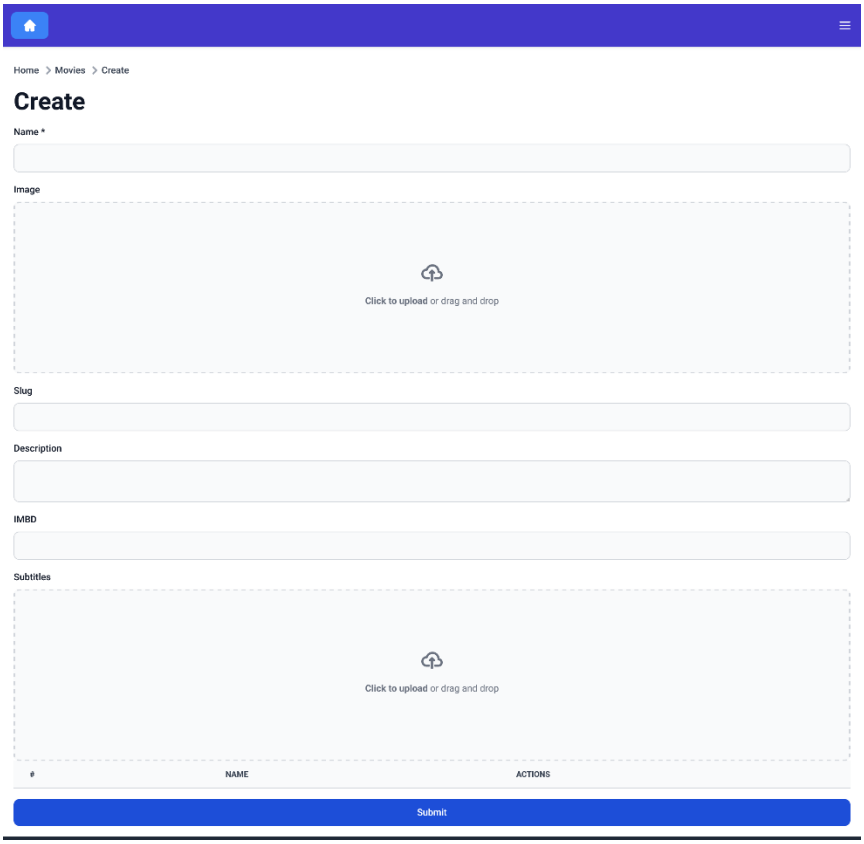

Рисунок 3.17 — Сторінка створення фільму

На цій сторінці користувачу потрібно заповнити форму, щоб додати фільм. Є обов’язкові поля котрі потрібно заповнити, наприклад назва та фото, але є й необов’язкові поля, наприклад, IMDB ID.

Також при створенні фільму можна завантажити файли субтитрів для нього. Це не є обов’язковим полем, фільм можна додати без жодного файлу субтитрів, а вже потім в формі оновленню фільму завантажити субтитри.

Після заповнення всіх обов’язкових даних користувачу потрібно натиснути на кнопку відправлення форми, після цього відбудеться перевірка даних. Система перевіряє чи правильно була заповнена інформація та інформує користувача, якщо виникнуть якісь помилки. У разі успішної перевірки система додає новий фільм до своєї бази даних.

У користувача є можливість оновити дані фільму, якщо вони змінилися після створення або у випадку, якщо під час створення була якась помилка, яку потрібно виправити. Для цього на сторінці фільму потрібно натиснути на кнопку оновлення, після цього користувач потрапить на сторінку оновлення фільму. Вона є досить схожою зі сторінкою створення фільму, найбільша різниця полягає в тому, що в полях форми вже є інформація на основі тих даних, що були введені при створенні фільму.

На сторінці перегляду серіалів є кнопка додання нового серіалу до системи. Після неї користувач потрапляє на сторінку додання нового серіалу до системи.

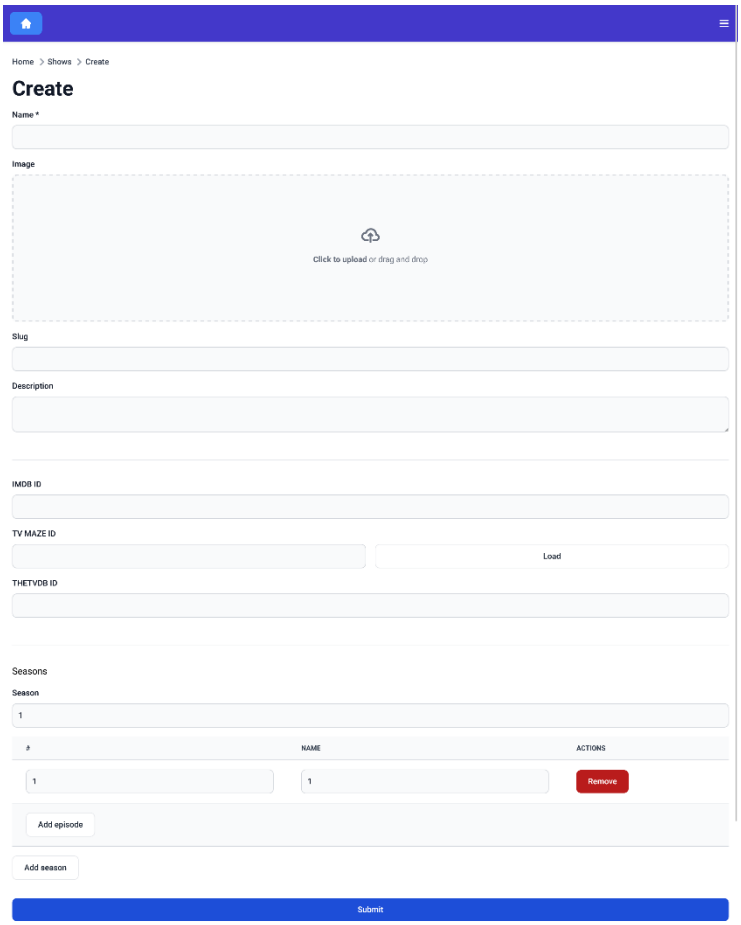

Рисунок 3.18 — Сторінка створення серіалу

Для доданні серіалу потрібно заповнити форму, яка має деякі обов'язкові та деякі необов’язкові поля. На відміну від сторінки фільму, сторінка серіалу має поля сезонів та епізодів. Фільми не мають різних сезонів та епізодів. Якщо фільм отримує продовження то зазвичай це просто інший фільм з цифрою «2» в назві, тому вони не потребують такої інформації. Один серіал може йти багато років, мати велику кількість різних сезонів та епізодів, тому потрібно записати цю інформацію в систему, тому, що вона буде дуже корисною та інформативною для користувача. Після заповнення потрібної інформації користувачу потрібно натиснути на кнопку відправки форми, після чого відбувається перевірка даних. Якщо під час перевірки буде знайдено помилки, то користувача буде повідомлено про ці помилки. У разі успішної перевірки користувач потрапить на сторінку створеного ним серіалу.

Дані серіалу можна оновити натиснувши на кнопку оновлення на сторінці серіалу. Після цього користувач потрапляє на сторінку оновлення серіалу, де він може оновити потрібні йому дані. Можливо, в серіалі вийшов новий сезон і потрібно додати його до системи, в такому випадку цей функціонал оновлення є дуже корисним.

Також користувач може оновити дані окремого епізоду серіалу натиснувши на кнопку оновлення на сторінці потрібного йому епізоду.

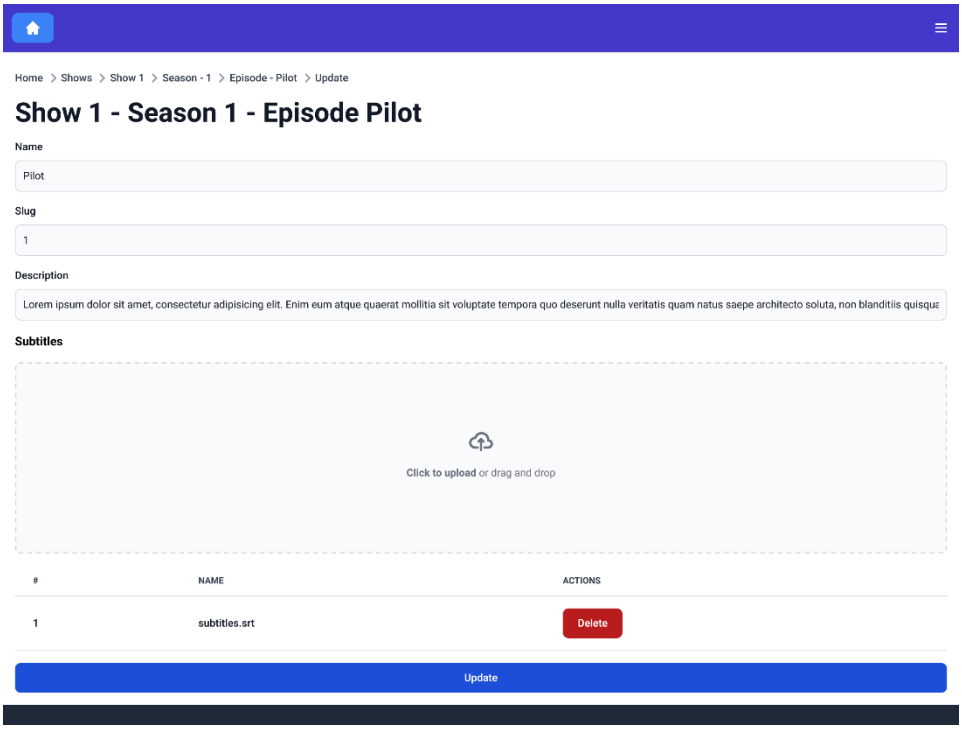

Рисунок 3.19 — Сторінка оновлення серії

На цій сторінці можна оновити назву серії та її опис. Також є можливість завантажити, переглянути або видалити субтитри для цього конкретного епізоду.

3.6.3 Адміністратор

Адміністратор – користувач, який слідкує за тим, щоб система працювала так, як потрібно. Він має доступ до всього того ж функціоналу,

що й авторизований користувач, але ще в додачу він має доступ до панелі управління аккаунтами користувачів.

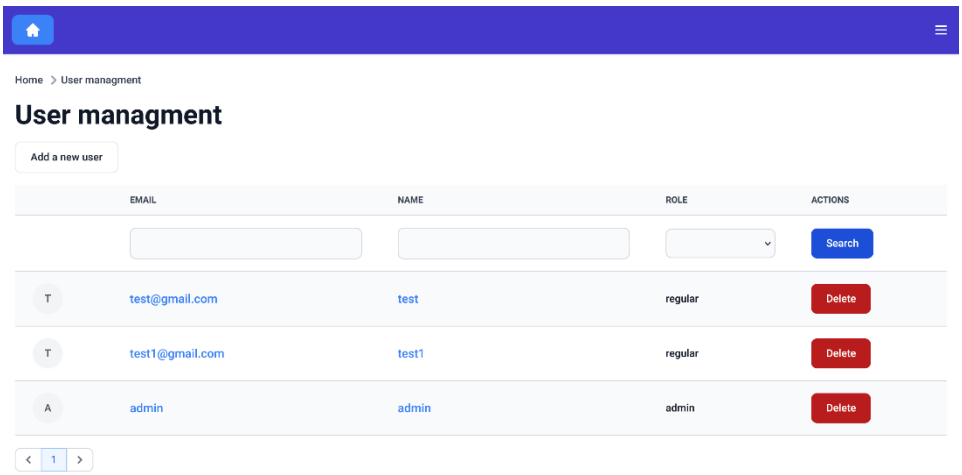

Рисунок 3.20 — Сторінка управління аккаунта користувачів

Панель управління користувача має функціонал пошуку користувача. Адміністратор може написати електронну пошту користувача і система знайде цього користувача, якщо він існує. Також адміністратор може видалити аккаунт користувача з системи. Адміністратор може перейти за посиланням на сторінку перегляду аккаунту користувача.

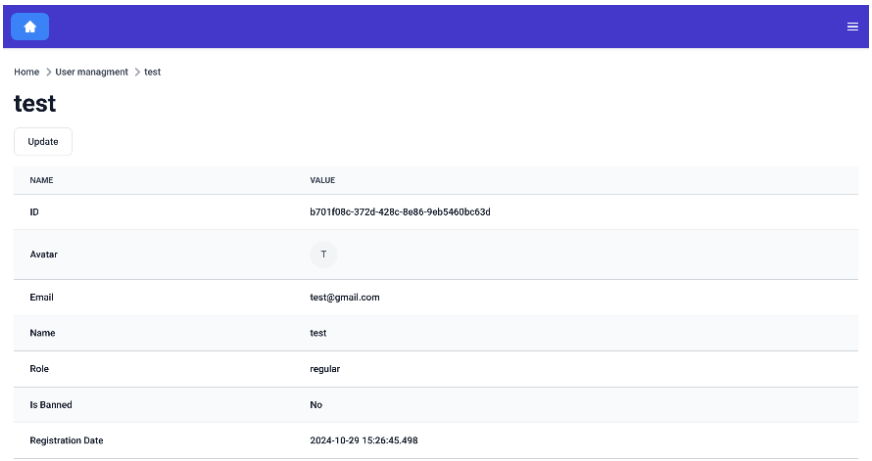

Рисунок 3.21 — Сторінка перегляду аккаунту користувача

На цій сторінці можна детально переглянути дані користувачі, його унікальний ідентифікатор в базі даних, пошту, ім’я, роль, дату реєстрації,

статус блокування та коли і яким адміністратором його було заблоковано. Вгорі сторінки знаходиться посилання на сторінку оновлення аккаунту користувача.

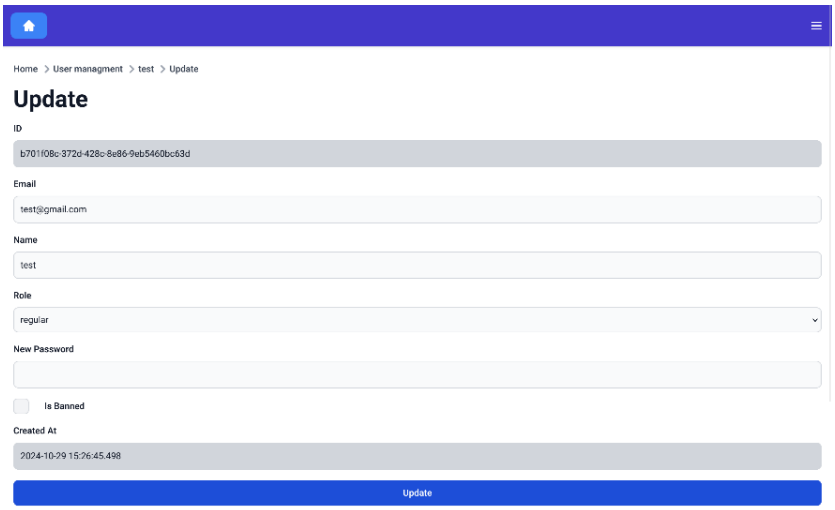

Рисунок 3.22 — Сторінка управління аккаунтом користувача

На цій сторінці адміністратор може оновити дані користувача, його пошту або ім’я, пароль, видалити його з системи, або заблокувати.

Якщо користувач буде заблокований, то він не зможе авторизуватися в системі. Під час авторизації користувач побачить повідомлення про те, що його було заблоковано. Це означає, що користувач втрачає доступ до функціоналу авторизованого користувача. В нього більше не буде можливості, наприклад, автоматично генерувати субтитри, але в нього залишиться доступ до функціоналу неавторизованого користувача. В нього все ще буде можливість зробити пошук тексту у субтитрах, переглядати субтитри.

ВИСНОВКИ

Робота систем управління субтитрами з використанням штучного інтелекту була досліджена у ході виконання цього проекту.

Була визначена актуальність автоматичної генерації субтитрів та їх організації за допомогою аналізу різних джерел та аналогів.

Користувачі мають можливість створювати власні облікові записи в системі, що надає їх доступ до додаткового функціоналу в порівнянні з неавторизованими користувачами. Вони отримують можливість створювати власні колекції різного контенту, управління своїм обліковим записом, автоматичну генерацію субтитрів за допомогою штучного інтелекту, тощо.

Система має функціонал пошуку тексту субтитрах, що дозволяє користувачам швидко знайти потрібну інформацію в файлах субтитрів. В додачу до того, що користувачі мають велику свободу у виборі контенту, це дає користувачу великі можливості.

Автоматична генерація субтитрів за допомогою штучного інтелекту є важливою частиною системи. Авторизований користувач може завантажити майже будь-який відео файл і отримати автоматично згенеровані субтитри для нього. На вибір є великий список іноземних мов та можливість автоматичного перекладу субтитрів на англійську мову. Історія такої генерації зберігається в обліковому записі користувача та дає йому можливість переглядати ці дані за власним бажання.

Система має базу даних з різними видами контенту, будь то фільмі або серіали, які авторизовані користувачі можуть самостійно додавати до системи. До цього контенту користувачі можуть додавати власні файли субтитрів для подальшого використання.

В результаті роботи була розроблена система управління субтитрами з використанням штучного інтелекту, яка дає користувачам свободу та ефективність в роботі з субтитрами.

ПЕРЕЛІК ПОСИЛАНЬ

1. Film industry in Europe - statistics & facts [Електронний ресурс]. Режим доступу: https://www.statista.com/topics/8042/film-industry-in-europe/#topicOverview

2. Devices used to watch online video content weekly among viewers in the United States as of March 2024, by age group [Електронний ресурс]. Режим доступу: https://www.statista.com/statistics/605628/frequency-video-services-used-by-smarpthone-users-united-states/

3. Languages without borders: Why Americans prefer subtitles to dubbing [Електронний ресурс]. Режим доступу: https://preply.com/en/blog/subbing-vs-dubbing-by-country/

4. Using subtitles for language learning can improve comprehension by 17% – here's how to do it [Електронний ресурс]. Режим доступу: https://preply.com/en/blog/using-subtitles-for-language-learning/

5. Netflix for the World's Most Downstream Internet Traffic in 2022-2023 [Електронний ресурс]. Режим доступу: https://www.marketresearchfuture.com/news/netflix-for-the-world-s-most-downstream-internet-traffic-in-2022-2023

6. Why do TV shows and movies leave Netflix? [Електронний ресурс]. Режим доступу: https://help.netflix.com/en/node/60541

7. HTML: HyperText Markup Language [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTML

8. CSS: Cascading Style Sheets [Електронний ресурс]. Режим доступу: https://developer.mozilla.org/en-US/docs/Web/CSS

9. TypeScript for JavaScript Programmers [Електронний ресурс]. Режим доступу: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html

10. Thinking in React [Електронний ресурс]. Режим доступу: https://react.dev/learn/thinking-in-react

11. Introduction to Node.js [Електронний ресурс]. Режим доступу: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs

12. Introducing Whisper [Електронний ресурс]. Режим доступу: https://openai.com/index/whisper/

13. Setting up Visual Studio Code [Електронний ресурс]. Режим доступу: https://code.visualstudio.com/docs/setup/setup-overview

14. About SQLite [Електронний ресурс]. Режим доступу: https://www.sqlite.org/about.html

Додаток А – Код програми

Код – Пошук тексту в субтитрах

```
    public async search(
        page: number,
        searchRequest: ISubtitlesSearchRequestDto
      ): Promise<ISubtitlesSearchResponseDto> {
        try {
          const db = this.database.getDB();

          const subtitles = await db
            .selectFrom('subtitles')
            .innerJoin('files', 'files.id', 'subtitles.file_id')
            .innerJoin(
              'episode_subtitles',
              'episode_subtitles.subtitles_id',
              'subtitles.id'
            )
            .innerJoin('episodes', 'episodes.id',
'episode_subtitles.episode_id')
            .innerJoin('seasons', 'seasons.id',
'episodes.season_id')
            .innerJoin('shows', 'shows.id', 'seasons.show_id')
            .select([
              'subtitles.id as subtitles_id',
              'subtitles.name as subtitles_name',
              'files.path',

              'shows.name as show_name',
              'shows.slug as show_slug',
              'seasons.name as season_name',
              'seasons.slug as season_slug',
              'episodes.name as episode_name',
              'episodes.slug as episode_slug',
            ])
            //do not find anything if there is not text
            .$if(searchRequest.text.length === 0, (qb) =>
              qb.where('subtitles.id', 'is', null)
            )
            .$if(searchRequest.show_id.length > 0, (qb) =>
              qb.where('shows.id', '=', searchRequest.show_id)
            )
            .orderBy('shows.name asc')
            .orderBy('seasons.order asc')
            .orderBy('episodes.order asc')
            .orderBy('subtitles.name asc')
            .execute();

          const allItems: ISubtitlesSearchResponseDto['items'] =
[];
          for (const subtitle of subtitles) {
            const fileResult = await searchForTextInFile(
```

```
              this.logger,
              searchRequest.text,
              subtitle.path
            );
            if (fileResult.length > 0) {
              allItems.push({
                subtitles_id: subtitle.subtitles_id,
                subtitles_name: subtitle.subtitles_name,
                nodes: [...fileResult],

                name: subtitle.episode_name,
                href:
`/shows/${subtitle.show_slug}/seasons/${subtitle.season_slug}/ep
isodes/${subtitle.episode_slug}`,
              });
            }
          }

          const itemsPaginated = allItems.slice(
            getPageOffset(page),
            getPageOffset(page + 1)
          );

          return {
            items: itemsPaginated,
            pagination: {
              currentPage: page,
              totalPages: Math.ceil(allItems.length /
PAGE_SIZE),
            },
          };
        } catch (error) {
          this.logger.error(`search: ${error}`);
          throw error;
        }
      }
```

Код – Оновлення паролю

```
    public async updatePassword(
        user_id: string,
        dto: IUsersMyProfileUpdatePasswordDto
      ): Promise<ICommonResponseDto> {
        try {
          const db = this.database.getDB();

          const password_hash = await
hashPassword(dto.new_password);

          const userObject = await db
            .selectFrom('users')
            .select('users.password_hash')
            .where('users.id', '=', user_id)
```

```
            .executeTakeFirst();

          const isPasswordHashMatch = await bcrypt.compare(
            dto.current_password,
            userObject.password_hash
          );
          if (!isPasswordHashMatch) {
            return getCommonResponseError('Wrong password.');
          }

          await db
            .updateTable('users')
            .set({
              password_hash: password_hash,
              updated_at: getNowSQL(),
            })
            .where('users.id', '=', user_id)
            .execute();
          return getCommonResponseSuccess();
        } catch (error) {
          this.logger.error(`updateProfile: ${error}`);
          throw error;
        }
      }
```

Код – Автоматична генерація субтитрів

```
    private async createJob(
        expressFile: IExpressFile,
        generatedSubtitles: IGeneratedSubtitles
      ) {
        const jobID = generatedSubtitles.id;

        try {
          const db = this.database.getDB();
          this.logger.log(
            `Job(${jobID}): Created generated subtitles
${generatedSubtitles.id}`
          );
          const uploadedFile = await stepUploadFile(
            db,
            generatedSubtitles,
            expressFile
          );
          this.logger.log(`Job(${jobID}): Uploaded file
${uploadedFile.path}`);
          const audioFile = await stepGetAudioFromVideo(
            db,
            generatedSubtitles,
            uploadedFile
          );
          this.logger.log(`Job(${jobID}): Generated audio file
${audioFile.path}`);
```

```
          const subtitlesFile = await
stepCreateSubtitlesFromAudio(
            db,
            generatedSubtitles,
            audioFile
          );
          this.logger.log(
            `Job(${jobID}): Generated subtitles file
${subtitlesFile.path}`
          );

          const videoWithSubtitlesFile = await
stepBurnSubtitlesIntoVideo(
            db,
            generatedSubtitles,
            uploadedFile,
            subtitlesFile
          );
          this.logger.log(
            `Job(${jobID}): Generated video with subtitles file
${videoWithSubtitlesFile.path}`
          );
        } catch (error) {
          this.logger.error(`createJob(${jobID}): ${error}`);

          const db = this.database.getDB();
          await db
            .updateTable('generated_subtitles')
            .set({ error: error?.toString() })
            .where('generated_subtitles.id', '=',
generatedSubtitles.id)
            .execute();

          throw error;
        }
      }
```

Код – Перевірка авторизації користувача на сервері.

```
    @Injectable()
    export class AuthGuard implements CanActivate {
      constructor(
        private readonly jwtService: JwtService,
        private readonly reflector: Reflector
      ) {}

      public async canActivate(context: ExecutionContext):
Promise<boolean> {
        const isPublic =
this.reflector.getAllAndOverride<boolean>(IS_PUBLIC_KEY, [
          context.getHandler(),
          context.getClass(),
        ]);
```

```
        const request = context.switchToHttp().getRequest();
        const token = this.extractTokenFromRequest(request);

        if (!token) {
          if (!isPublic) {
            throw new UnauthorizedException();
          }
        }
        try {
          const payload: IJwtDTo = await
this.jwtService.verifyAsync(token);
          request['user'] = payload;
        } catch {
          if (!isPublic) {
            throw new UnauthorizedException();
          }
        }
        return true;
      }

      private extractTokenFromRequest(request: Request): string
| undefined {
        const token =
          request.cookies?.[JWT_COOKIES.accessToken]?.toString()
?? undefined;
        return token;
      }
    }
```

Код – Перевірка ролі користувача на сервері.

```
    @Injectable()
    export class RolesGuard implements CanActivate {
      constructor(private reflector: Reflector) {}

      canActivate(context: ExecutionContext): boolean {
        const requiredRoles =
this.reflector.getAllAndOverride<TUserRoleID[]>(
          ROLES_KEY,
          [context.getHandler(), context.getClass()]
        );

        if (!requiredRoles || requiredRoles.length === 0) {
          return true;
        }
        const { user } = context
          .switchToHttp()
          .getRequest<{ user: IJwtDTo | null }>();
        return requiredRoles.some((role) => role ===
user?.role_id);
      }
    }
```

Код – Контролер авторизації на сервері.

```
@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  private readonly logger = new Logger(AuthController.name);

  @Public()
  @Post('register')
  public async register(
    @Body() body: IRegisterDto
  ): Promise<ICommonResponseDto> {
    return this.authService.register(body);
  }

  @Public()
  @Post('login')
  public async login(
    @Res({ passthrough: true }) res: Response,
    @Body() body: ILoginDto
  ): Promise<ICommonResponseDto> {
    const { jwt, refresh } = await
this.authService.login(body);
    res.cookie(JWT_COOKIES.accessToken, jwt, {
      httpOnly: true,
      maxAge: 10 * 60 * 1_000, //10 minutes
    });
    res.cookie(JWT_COOKIES.refreshToken, refresh, {
      httpOnly: true,
      maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days
    });
    return getCommonResponseSuccess();
  }

  @Public()
  @Post('logout')
  public async logout(
    @Res({ passthrough: true }) res: Response
  ): Promise<ICommonResponseDto> {
    res.clearCookie(JWT_COOKIES.accessToken);
    res.clearCookie(JWT_COOKIES.refreshToken);
    return getCommonResponseSuccess();
  }

  @Public()
  @Post('refresh')
  public async refresh(
    @Req() request: Request,
    @Res({ passthrough: true }) res: Response
  ): Promise<ICommonResponseDto> {
```

```
        const refreshCookie =
request.cookies[JWT_COOKIES.refreshToken];
        const { jwt, refresh, success } = await
this.authService.refesh(
          refreshCookie
        );
        if (success) {
          res.cookie(JWT_COOKIES.accessToken, jwt, {
            httpOnly: true,
            maxAge: 10 * 60 * 1_000, //10 minutes
          });
          res.cookie(JWT_COOKIES.refreshToken, refresh, {
            httpOnly: true,
            maxAge: 7 * 24 * 60 * 60 * 1000, // 7 days
          });
        } else {
          res.clearCookie(JWT_COOKIES.accessToken);
          res.clearCookie(JWT_COOKIES.refreshToken);
          return getCommonResponseError();
        }

        return getCommonResponseSuccess();
      }

      @HttpCode(HttpStatus.OK)
      @Post('profile')
      public async getProfile(
        @JwtDto() jwtDto: IJwtDTo | null
      ): Promise<IJwtDTo | null> {
        return jwtDto;
      }
    }
```

Код – Сервіс авторизації на сервері.

```
    @Injectable()
    export class AuthService {
      constructor(
        private readonly database: DatabaseService,
        private readonly jwtService: JwtService
      ) {}

      private readonly logger = new Logger(AuthService.name);

      public async register(dto: IRegisterDto):
Promise<ICommonResponseDto> {
        try {
          const db = this.database.getDB();

          const password_hash = await
hashPassword(dto.password);

          const nextUser: IUser = {
```

```
            id: randomUUID(),
            email: dto.email,
            name: dto.name,
            password_hash: password_hash,
            role_id: USER_ROLES.regular,
            created_at: getNowSQL(),
            updated_at: getNowSQL(),
          };

          const isEmailDuplicate = await db
            .selectFrom('users')
            .select('users.email')
            .where('users.email', '=', dto.email)
            .executeTakeFirst();

          if (isEmailDuplicate) {
            throw new BadRequestException(`This email is already
taken.`);
          }

          const isNameDuplicate = await db
            .selectFrom('users')
            .select('users.id')
            .where('users.name', '=', dto.name)
            .executeTakeFirst();

          if (isNameDuplicate) {
            throw new BadRequestException(`This name is already
taken.`);
          }
          await db.transaction().execute(async (trx) => {
            await
trx.insertInto('users').values(nextUser).execute();
          });

          return getCommonResponseSuccess();
        } catch (error) {
          this.logger.error(`register: ${error}`);
          throw error;
        }
      }

      public async login(dto: ILoginDto) {
        try {
          const db = this.database.getDB();

          const userMatch = await db
            .selectFrom('users')
            .select([
              'users.id',
              'users.email',
              'users.name',
              'users.password_hash',
```

```
            'users.role_id',
          ])
          .where('users.email', '=', dto.email)
          .executeTakeFirst();

        if (!userMatch) {
          throw new BadRequestException('Wrong email and
password match.');
        }

        const isPasswordMatch = await bcrypt.compare(
          dto.password,
          userMatch.password_hash
        );
        if (!isPasswordMatch) {
          throw new BadRequestException('Wrong email and
password match.');
        }

        const isUserBanned = await db
          .selectFrom('bans')
          .select('bans.id')
          .where('bans.user_id', '=', userMatch.id)
          .executeTakeFirst();

        if (isUserBanned) {
          throw new BadRequestException('Your account is
banned.');
        }

        const payload: IJwtDTo = {
          sub: userMatch.id,
          name: userMatch.name,
          role_id: userMatch.role_id,
        };

        const jwt = await this.jwtService.signAsync(payload);

        const refreshTokenDto: IRefreshTokenDto = {
          user_id: userMatch.id,
        };
        const refresh = await
this.jwtService.signAsync(refreshTokenDto, {
          expiresIn: '7d',
        });

        const refreshToken: IRefreshToken = {
          id: randomUUID(),
          user_id: userMatch.id,
          token: refresh,
          created_at: getNowSQL(),
        };
        await db
```

```
            .deleteFrom('refresh_tokens')
            .where('refresh_tokens.user_id', '=', userMatch.id)
            .execute();
          await
db.insertInto('refresh_tokens').values(refreshToken).execute();

          return { jwt, refresh };
        } catch (error) {
          this.logger.error(`login: ${error}`);
          throw error;
        }
      }

      public async refesh(refreshCookie: string) {
        try {
          if (!refreshCookie) {
            throw new BadRequestException('Refresh Token not
present.');
          }
          const db = this.database.getDB();

          const isValidToken = await db
            .selectFrom('refresh_tokens')
            .select('refresh_tokens.id')
            .where('refresh_tokens.token', '=', refreshCookie)
            .executeTakeFirst();
          if (!isValidToken?.id) {
            throw new BadRequestException(`Invalid refresh
token.`);
          }

          const refreshTokenDtoOld: IRefreshTokenDto =
            await this.jwtService.verifyAsync(refreshCookie);

          const userMatch = await db
            .selectFrom('users')
            .select([
              'users.id',
              'users.name',
              'users.password_hash',
              'users.role_id',
            ])
            .where('users.id', '=', refreshTokenDtoOld.user_id)
            .executeTakeFirst();

          const isUserBanned = await db
            .selectFrom('bans')
            .select('bans.id')
            .where('bans.user_id', '=', userMatch.id)
            .executeTakeFirst();

          if (isUserBanned) {
```

```
            throw new BadRequestException('Your account is
banned.');
          }

          const payload: IJwtDTo = {
            sub: userMatch.id,
            name: userMatch.name,
            role_id: userMatch.role_id,
          };

          const jwt = await this.jwtService.signAsync(payload);

          const refreshTokenDto: IRefreshTokenDto = {
            user_id: userMatch.id,
          };
          const refresh = await
this.jwtService.signAsync(refreshTokenDto, {
            expiresIn: '7d',
          });

          const refreshToken: IRefreshToken = {
            id: randomUUID(),
            user_id: userMatch.id,
            token: refresh,
            created_at: getNowSQL(),
          };
          await db
            .deleteFrom('refresh_tokens')
            .where('refresh_tokens.user_id', '=', userMatch.id)
            .execute();
          await
db.insertInto('refresh_tokens').values(refreshToken).execute();

          return { jwt, refresh, success: true };
        } catch (error) {
          this.logger.error(`refresh: ${error}`);

          if (error instanceof BadRequestException) {
            return { jwt: '', refresh: '', success: false };
          }
          throw error;
        }
      }
    }
```

Код – Сервіс головної сторінки на сервері.

```
    @Injectable()
    export class MainPageService {
      constructor(private readonly database: DatabaseService) {}

      private readonly logger = new
Logger(MainPageService.name);
```

```
    private readonly PAGE_SIZE = 5;

    public async view(): Promise<IMainPageViewDto> {
      try {
        const db = this.database.getDB();

        const shows = await db
          .selectFrom('shows')
          .innerJoin('files', 'shows.image_file_id',
'files.id')
          .select([
            'shows.id',
            'shows.slug',
            'shows.name',

            (qb) =>
              qb
                .fn<string>('concat',
[qb.val(FILE_URL_PREFIX), 'files.path'])
                .as('image_file_url'),
          ])
          .where('shows.is_movie', '=', SQLITE_BOOL.false)
          .orderBy('shows.name', 'asc')
          .limit(this.PAGE_SIZE)
          .execute();

        const movies = await db
          .selectFrom('shows')
          .innerJoin('files', 'shows.image_file_id',
'files.id')
          .select([
            'shows.id',
            'shows.slug',
            'shows.name',

            (qb) =>
              qb
                .fn<string>('concat',
[qb.val(FILE_URL_PREFIX), 'files.path'])
                .as('image_file_url'),
          ])
          .where('shows.is_movie', '=', SQLITE_BOOL.true)
          .orderBy('shows.name', 'asc')
          .limit(this.PAGE_SIZE)
          .execute();

        return { shows, movies };
      } catch (error) {
        this.logger.error(`view: ${error}`);
        throw error;
      }
    }
  }
```

Код – Контролер управління користувачами для адміністратора на сервері.

```
     @Roles(ADMIN_PANEL_PERMISSIONS)
     @Controller('user-managment')
     export class UserManagmentController {
       constructor(private readonly userManagmentService:
UserManagmentService) {}

       @Get('view-all')
       public async viewAll(
         @Page() page: number,
         @Query('email') email: string,
         @Query('name') name: string,
         @Query('role_id') role_id: string
       ): Promise<IUserManagmentViewAllDto> {
         return this.userManagmentService.viewAll(page, { email,
name, role_id });
       }

       @Get('view/:id')
       public async view(@Param('id') id: string):
Promise<IUserManagmentViewDto> {
         return this.userManagmentService.view(id);
       }

       @Post('create')
       public async create(
         @Body() body: IUserManagmentCreateDto,
         @JwtDto() jwtDto: IJwtDTo
       ): Promise<IUserManagmentCreateDtoResponse> {
         return this.userManagmentService.create(body, jwtDto);
       }

       @Get('read/:id')
       public async read(@Param('id') id: string):
Promise<IUserManagmentReadDto> {
         return this.userManagmentService.read(id);
       }

       @Put('update')
       public async update(
         @Body() body: IUserManagmentUpdateDto,
         @JwtDto() jwtDto: IJwtDTo
       ): Promise<IUserManagmentUpdateDtoResponse> {
         return this.userManagmentService.update(body, jwtDto);
       }

       @Delete(':id')
       public async delete(@Param('id') id: string):
Promise<ICommonResponseDto> {
         return this.userManagmentService.delete(id);
       }
```

```
    //additional
    @Get('get-user-roles-options')
    public async getUserRolesOptions(): Promise<IOption[]> {
      return this.userManagmentService.getUserRolesOptions();
    }
  }
```

Код – Сервіс управління користувачами для адміністратора на сервері.

```
  @Injectable()
  export class UserManagmentService {
    constructor(private readonly database: DatabaseService) {}

    private readonly logger = new
Logger(UserManagmentService.name);

    public async viewAll(
      page: number,
      searchRequest: IUserManagmentViewAllSearchRequest
    ): Promise<IUserManagmentViewAllDto> {
      try {
        const db = this.database.getDB();

        const items = await db
          .selectFrom('users')
          .innerJoin('user_roles', 'user_roles.id',
'users.role_id')
          .select([
            'users.id as user_id',
            'users.email as user_email',
            'users.name as user_name',
            'user_roles.name as user_role',
          ])
          .orderBy('users.created_at')
          .orderBy('users.id')
          .$if(Boolean(searchRequest.email), (qb) =>
            qb.where('users.email', 'like',
`%${searchRequest.email}%`)
          )
          .$if(Boolean(searchRequest.name), (qb) =>
            qb.where('users.name', 'like',
`%${searchRequest.name}%`)
          )
          .$if(Boolean(searchRequest.role_id), (qb) =>
            qb.where('users.role_id', '=',
searchRequest.role_id)
          )
          .limit(PAGE_SIZE)
          .offset(getPageOffset(page, PAGE_SIZE))
          .execute();

        const pagination = await db
```

```
            .selectFrom('users')
            .select([
              (qb) => qb.val(page).as('currentPage'),
              sql<number>`CEIL(MAX(COUNT(*), 1) /
${PAGE_SIZE})`.as('totalPages'),
            ])
            .$if(Boolean(searchRequest.email), (qb) =>
              qb.where('users.email', 'like',
`%${searchRequest.email}%`)
            )
            .$if(Boolean(searchRequest.name), (qb) =>
              qb.where('users.name', 'like',
`%${searchRequest.name}%`)
            )
            .$if(Boolean(searchRequest.role_id), (qb) =>
              qb.where('users.role_id', '=',
searchRequest.role_id)
            )
            .executeTakeFirst();

          return { items, pagination };
        } catch (error) {
          this.logger.error(`viewAll: ${error}`);
          throw error;
        }
      }

      public async view(id: string):
Promise<IUserManagmentViewDto> {
        try {
          const db = this.database.getDB();

          const item = await db
            .selectFrom('users')
            .innerJoin('user_roles', 'user_roles.id',
'users.role_id')
            .select([
              'users.id',
              'users.email',
              'users.name',
              'user_roles.name as role_name',
              'users.created_at',
            ])
            .where('users.id', '=', id)
            .executeTakeFirst();

          const ban = await db
            .selectFrom('bans')
            .innerJoin('users', 'users.id', 'bans.admin_id')
            .select([
              'bans.id',
              'bans.admin_id',
              'bans.user_id',
```

```
              'bans.created_at',
              'users.email as admin_email',
            ])
            .where('bans.user_id', '=', id)
            .executeTakeFirst();

          if (!item) {
            throw new NotFoundException();
          }

          return { ...item, is_banned: Boolean(ban), ban: ban ??
null };
        } catch (error) {
          this.logger.error(`view: ${error}`);
          throw error;
        }
      }

      public async read(id: string):
Promise<IUserManagmentReadDto> {
        try {
          const db = this.database.getDB();

          const item = await db
            .selectFrom('users')
            .select([
              'users.id',
              'users.email',
              'users.name',
              'users.role_id',
              'users.created_at',
            ])
            .where('users.id', '=', id)
            .executeTakeFirst();

          const ban = await db
            .selectFrom('bans')
            .selectAll()
            .where('bans.user_id', '=', id)
            .executeTakeFirst();

          if (!item) {
            throw new NotFoundException();
          }

          return { ...item, ban: ban ?? null };
        } catch (error) {
          this.logger.error(`view: ${error}`);
          throw error;
        }
      }

      public async create(
```

```
        dto: IUserManagmentCreateDto,
        jwtDto: IJwtDTo
      ): Promise<IUserManagmentCreateDtoResponse> {
        try {
          const db = this.database.getDB();

          const isEmailDuplicate = await db
            .selectFrom('users')
            .select('users.email')
            .where('users.email', '=', dto.email)
            .executeTakeFirst();

          if (isEmailDuplicate) {
            throw new BadRequestException(`This email is already
taken.`);
          }

          const isNameDuplicate = await db
            .selectFrom('users')
            .select('users.id')
            .where('users.name', '=', dto.name)
            .executeTakeFirst();

          if (isNameDuplicate) {
            throw new BadRequestException(`This name is already
taken.`);
          }

          const password_hash = await
hashPassword(dto.password);
          const nextUser: IUser = {
            id: randomUUID(),
            email: dto.email,
            name: dto.name,
            password_hash: password_hash,
            role_id: dto.role_id,
            created_at: getNowSQL(),
            updated_at: getNowSQL(),
          };
          const nextBan: IBan | null = dto.ban
            ? {
                id: randomUUID(),
                admin_id: jwtDto.sub,
                user_id: nextUser.id,
                created_at: getNowSQL(),
              }
            : null;

          await db.transaction().execute(async (trx) => {
            await
trx.insertInto('users').values(nextUser).execute();
            if (nextBan) {
```

```
              await
trx.insertInto('bans').values(nextBan).execute();
            }
          });

          return getCommonResponseSuccess(null, { user_id:
nextUser.id });
        } catch (error) {
          this.logger.error(`create: ${error}`);
          throw error;
        }
      }

      public async update(
        dto: IUserManagmentUpdateDto,
        jwtDto: IJwtDTo
      ): Promise<IUserManagmentUpdateDtoResponse> {
        try {
          const db = this.database.getDB();

          const oldBan = await db
            .selectFrom('bans')
            .selectAll()
            .where('bans.user_id', '=', dto.id)
            .executeTakeFirst();

          const isEmailDuplicate = await db
            .selectFrom('users')
            .select('users.email')
            .where('users.email', '=', dto.email)
            .where('users.id', '!=', dto.id)
            .executeTakeFirst();

          if (isEmailDuplicate) {
            throw new BadRequestException(`This email is already
taken.`);
          }

          const isNameDuplicate = await db
            .selectFrom('users')
            .select('users.id')
            .where('users.name', '=', dto.name)
            .where('users.id', '!=', dto.id)
            .executeTakeFirst();

          if (isNameDuplicate) {
            throw new BadRequestException(`This name is already
taken.`);
          }

          const oldUserObject = await db
            .selectFrom('users')
            .select(['users.password_hash', 'users.created_at'])
```

```
            .where('users.id', '=', dto.id)
            .executeTakeFirst();

          const oldPasswordHash = oldUserObject.password_hash;
          const password_hash =
            dto.password.length > 0
              ? await hashPassword(dto.password)
              : oldPasswordHash;

          const nextUser: IUser = {
            id: dto.id,
            email: dto.email,
            name: dto.name,
            password_hash: password_hash,
            role_id: dto.role_id,
            created_at: oldUserObject.created_at,
            updated_at: getNowSQL(),
          };

          const nextBan: IBan | null = dto.ban
            ? {
                id: oldBan?.id ?? randomUUID(),
                admin_id: oldBan?.admin_id ?? jwtDto.sub,
                user_id: nextUser.id,
                created_at: oldBan?.created_at ?? getNowSQL(),
              }
            : null;

          await db.transaction().execute(async (trx) => {
            await trx
              .updateTable('users')
              .set(nextUser)
              .where('users.id', '=', dto.id)
              .execute();

            if (oldBan) {
              await trx
                .deleteFrom('bans')
                .where('bans.user_id', '=', nextUser.id)
                .execute();
            }
            if (nextBan) {
              await
trx.insertInto('bans').values(nextBan).execute();
            }
          });

          return getCommonResponseSuccess(null, { user_id:
nextUser.id });
        } catch (error) {
          this.logger.error(`update: ${error}`);
          throw error;
        }
```

```
    }

    public async delete(id: string):
Promise<ICommonResponseDto> {
      try {
        const db = this.database.getDB();

        const generated_subtitles_of_user = await db
          .selectFrom('generated_subtitles')
          .select('generated_subtitles.id')
          .where('generated_subtitles.user_id', '=', id)
          .execute();

        for (const generated_subtitle_of_user of
generated_subtitles_of_user) {
          await generatedSubtitlesRespositoryDeleteOne(
            db,
            generated_subtitle_of_user.id
          );
        }

        await db.transaction().execute(async (trx) => {
          await trx.deleteFrom('users').where('users.id', '=',
id).execute();
        });

        return getCommonResponseSuccess();
      } catch (error) {
        this.logger.error(`delete: ${error}`);
        throw error;
      }
    }

    public async getUserRolesOptions(): Promise<IOption[]> {
      try {
        const db = this.database.getDB();

        const items = await db
          .selectFrom('user_roles')
          .select(['user_roles.id as value', 'user_roles.name
as label'])
          .orderBy('user_roles.id')
          .execute();

        return items;
      } catch (error) {
        this.logger.error(`getUserRolesOptions: ${error}`);
        throw error;
      }
    }
  }
```

Код – Головна сторінка (ReactJS).

```
    interface IProps extends IAuthContextProps {
      dto: IMainPageViewDto;
    }

    export const getServerSideProps: GetServerSideProps<IProps>
= async (
      context
    ) => {
      const axiosServer = getAxiosServer(context);
      const jwtDto = await getJwtDtoHttp(axiosServer);

      const dto = await getMainPageViewDtoSendHttp(axiosServer);
      return { props: { jwtDto, dto } };
    };

    export default function IndexPage({ dto }: IProps) {
      return (
        <Layout>
          <Head>

<title>{getTitleFromBreadcrumbs(MAIN_PAGE_BREADCRUMBS)}</title>
          </Head>

          <Breadcrumbs breadcrumbs={MAIN_PAGE_BREADCRUMBS} />

          <div className="flex flex-col gap-10 mb-24">
            <MainPageLoginBlock />

            <MainPageSubtitlesSearchBlock />

            <MainPageGeneratedSubtitlesBlock />

            <MainPageShowsBlock
              dto={dto.shows.map((item) => ({
                name: item.name,
                slug: item.slug,
                image: item.image_file_url,
              }))}
            />

            <MainPageMoviesBlock
              dto={dto.movies.map((item) => ({
                name: item.name,
                slug: item.slug,
                image: item.image_file_url,
              }))}
            />
          </div>
        </Layout>
      );
    }
```

Код – Сторінка автоматичних субтитрів (ReactJS).

```
     interface IProps extends IAuthContextProps {
       generatedSubtitlesModelsOptions: IOption[];
     }

     export const getServerSideProps: GetServerSideProps<IProps>
= async (
       context
     ) => {
       const axiosServer = getAxiosServer(context);
       const jwtDto = await getJwtDtoHttp(axiosServer);
       if (!checkUserRoleIncludes(jwtDto,
GENERATED_SUBTITLES_PERMISSIONS)) {
         return { notFound: true };
       }

       const generatedSubtitlesModelsOptions =
         await getGeneratedSubtitlesModelsOptions(axiosServer);
       return { props: { jwtDto, generatedSubtitlesModelsOptions
} };
     };

     export default function GeneratedSubtitlesCreate({
       generatedSubtitlesModelsOptions,
     }: IProps) {
       const formContext =
useForm<IGeneratedSubtitlesCreateForm>({
         defaultValues:
GENERATED_SUBTITLES_CREATE_FORM_DEFAULT_VALUES,
         resolver:
zodResolver(GENERATED_SUBTITLES_CREATE_FORM_SCHEMA),
       });

       const router = useRouter();

       const onResponseSuccess = useCallback(
         (data: IGeneratedSubtitlesCreateResponseDto) => {
           router.push(`/generated-
subtitles/${data.data.generated_subtitles_id}`);
         },
         [router]
       );

       const mutation = useMutation({
         mutationFn: generateSubtitlesCreateFormSendHttp,
         onSuccess: onResponseSuccess,
       });

       const onSubmit:
SubmitHandler<IGeneratedSubtitlesCreateForm> = (data) => {
         mutation.mutate(data);
       };

       const title = getTitleFromBreadcrumbs(
```

```
        GENERATE_SUBTITLES_CREATE_FORM_BREADCRUMBS
      );

      // form fields may be watched to display additional
information

      const watchFile = formContext.watch('file');
      const videoUrlFromFile = useMemo(() => {
        if (!watchFile) {
          return null;
        }

        const isVideoFile = watchFile.type.includes('video');
        if (!isVideoFile) {
          return null;
        }

        return URL.createObjectURL(watchFile);
      }, [watchFile]);

      const is_edit_mode = formContext.watch('is_edit_mode');

      return (
        <Layout>
          <Head>
            <title>{title}</title>
          </Head>

          <Breadcrumbs
breadcrumbs={GENERATE_SUBTITLES_CREATE_FORM_BREADCRUMBS} />

          <div className="flex gap-2 items-center justify-
start">
            <H1>{title}</H1>
            <CaptionIcon size={ICON_SIZE.h1} />
          </div>
          <MutationDisplay mutation={mutation} />

          <AlertInfo>Upload a video file to generate
subtitles</AlertInfo>

          <form onSubmit={formContext.handleSubmit(onSubmit,
onValidationError)}>
            <Fieldset>
              <Label htmlFor="video">Video *</Label>
              <FileInputArea
                id="video"
                fileName={formContext.watch('file')?.name}
                onChangeFile={(file) =>
formContext.setValue('file', file)}
                accept="video/*"
              />
              <HelperTextError>
```

```
                {formContext.formState.errors?.file?.message}
              </HelperTextError>
            </Fieldset>

            {videoUrlFromFile ? (
              <div className="my-4">
                <video src={videoUrlFromFile} controls={true}
className="w-full" />
              </div>
            ) : null}

            <Fieldset>
              <Checkbox
                label="Edit mode"
                {...formContext.register('is_edit_mode')}
              />
              <HelperTextError>

{formContext.formState?.errors?.is_edit_mode?.message}
              </HelperTextError>
            </Fieldset>

            {is_edit_mode ? (
              <React.Fragment>
                <Fieldset>
                  <Label htmlFor="model_id">Model *</Label>
                  <Select
                    id="model_id"
                    {...formContext.register('model_id')}
                    options={generatedSubtitlesModelsOptions}
                  />
                  <HelperTextError>

{formContext.formState?.errors?.model_id?.message}
                  </HelperTextError>
                  <HelperText>
                    The models are sorted by speed, with faster
models listed first.
                    While faster models may be less accurate,
slower models
                    typically provide greater accuracy. Models
with .en in their
                    name are designed to work exclusively with
the English language.
                  </HelperText>
                </Fieldset>

                <Fieldset>
                  <Checkbox
                    label="Should translate to English"

{...formContext.register('should_translate')}
                  />
```

```
                  <HelperTextError>

{formContext.formState?.errors?.should_translate?.message}
                  </HelperTextError>
                  <HelperText>
                    Select this option if the content needs to
be translated into
                    English. Leave it unchecked if no
translation is needed.
                  </HelperText>
                </Fieldset>
              </React.Fragment>
            ) : null}

            <Button type="submit" isFullWidth={true}
disabled={mutation.isLoading}>
              Generate
            </Button>
          </form>
        </Layout>
      );
    }
```

Код – Сторінка перегляду автоматичних субтитрів (ReactJS).

```
    interface IProps extends IAuthContextProps {
      dto: IGeneratedSubtitlesViewDto;
    }

    export const getServerSideProps: GetServerSideProps<IProps>
= async (
      context
    ) => {
      const axiosServer = getAxiosServer(context);
      const jwtDto = await getJwtDtoHttp(axiosServer);
      if (!checkUserRoleIncludes(jwtDto,
GENERATED_SUBTITLES_PERMISSIONS)) {
        return { notFound: true };
      }
      const id = context.params?.id?.toString();

      if (!id) {
        return { notFound: true };
      }

      const response = await
axiosServer.get<IGeneratedSubtitlesViewDto>(
        `/generated-subtitles/view/${id}`
      );
      const dto = response.data;
      return { props: { jwtDto, dto } };
    };
```

```
     export default function GeneratedSubtitlesId({ dto }:
IProps) {
       const title = getTitleFromBreadcrumbs(
         generatedSubtitlesViewGetBreadcrumbs(dto)
       );

       const router = useRouter();

       const isDone = dto.video_with_subtitles_file_id ? true :
false;
       const isError = Boolean(dto.error);

       useEffect(() => {
         const ONE_SECOND = 1000;

         const timeout = setTimeout(() => {
           if (isDone || isError) {
             return;
           }
           router.replace(router.asPath);
         }, ONE_SECOND);
         return () => {
           clearTimeout(timeout);
         };
       }, [isDone, isError, router]);

       if (
         dto.video_file_id &&
         dto.video_file_url &&
         dto.video_file_name &&
         dto.audio_file_id &&
         dto.audio_file_url &&
         dto.audio_file_name &&
         dto.subtitles_file_id &&
         dto.subtitles_file_url &&
         dto.subtitles_file_name &&
         dto.video_with_subtitles_file_id &&
         dto.video_with_subtitles_file_url &&
         dto.video_with_subtitles_file_name
       ) {
         return (
           <Layout>
             <Head>
               <title>{title}</title>
             </Head>

             <Breadcrumbs
breadcrumbs={generatedSubtitlesViewGetBreadcrumbs(dto)} />

             <H1>{title}</H1>

             {dto.error ? <AlertDanger>{dto.error}</AlertDanger>
: null}
```

```
              <div className="block my-4">
                <video
                  className="w-full aspect-video"
                  src={dto.video_with_subtitles_file_url}
                  controls
                />
                <p className="font-normal text-gray-700 my-
4">Files</p>
                <Table>
                  <Thead>
                    <TRow isDisableColors={true}>
                      <Th>File</Th>
                      <Th>Type</Th>
                      <Th>Actions</Th>
                    </TRow>
                  </Thead>
                  <TBody>
                    <TRow>

<Td>{dto.video_with_subtitles_file_name}</Td>
                      <Td>Video with subtitles</Td>
                      <Td>
                        <a
                          href={dto.video_with_subtitles_file_url}

download={dto.video_with_subtitles_file_name}
                          className="font-medium text-blue-600
hover:underline"
                        >
                          Download
                        </a>
                      </Td>
                    </TRow>

                    <TRow>
                      <Td>{dto.subtitles_file_name}</Td>
                      <Td>Subtitles</Td>
                      <Td>
                        <a
                          href={dto.subtitles_file_url}
                          download={dto.subtitles_file_name}
                          className="font-medium text-blue-600
hover:underline"
                        >
                          Download
                        </a>
                      </Td>
                    </TRow>

                    <TRow>
                      <Td>{dto.video_file_name}</Td>
                      <Td>Original video</Td>
```

```
                    <Td>
                      <a
                        href={dto.video_file_url}
                        download={dto.video_file_name}
                        className="font-medium text-blue-600
hover:underline"
                      >
                        Download
                      </a>
                    </Td>
                  </TRow>

                  <TRow>
                    <Td>{dto.audio_file_name}</Td>
                    <Td>Audio</Td>
                    <Td>
                      <a
                        href={dto.audio_file_url}
                        download={dto.audio_file_name}
                        className="font-medium text-blue-600
hover:underline"
                      >
                        Download
                      </a>
                    </Td>
                  </TRow>
                </TBody>
              </Table>
            </div>
          </Layout>
        );
      }

      return (
        <Layout>
          <Head>
            <title>{title}</title>
          </Head>

          <Breadcrumbs
breadcrumbs={generatedSubtitlesViewGetBreadcrumbs(dto)} />

          <H1>{title}</H1>

          {dto.error ? <AlertDanger>{dto.error}</AlertDanger> :
null}

          <h2 className="mb-2 text-lg font-semibold text-gray-
900">
            Auto-generating subtitles
          </h2>
          <ul className="max-w-md space-y-2 text-gray-500 list-
inside">
```

```
            <li className="flex items-center">
              {dto.video_file_url ? <DoneIcon /> : <SpinnerIcon
/>}
              Upload of your video file
            </li>
            <li className="flex items-center">
              {dto.audio_file_url ? <DoneIcon /> : <SpinnerIcon
/>}
              Getting the audio from your video
            </li>
            <li className="flex items-center">
              {dto.subtitles_file_id ? <DoneIcon /> :
<SpinnerIcon />}
              Generating subtitles
            </li>
            <li className="flex items-center">
              {dto.video_with_subtitles_file_id ? <DoneIcon /> :
<SpinnerIcon />}
              Generating a new video with subtitles
            </li>
          </ul>
        </Layout>
      );
    }
```

Код – Сторінка перегляду історії автоматичних субтитрів (ReactJS).

```
    interface IProps extends IAuthContextProps {
      dto: IGeneratedSubtitlesViewAllDto;
    }

    export const getServerSideProps: GetServerSideProps<IProps>
= async (
      context
    ) => {
      const axiosServer = getAxiosServer(context);
      const jwtDto = await getJwtDtoHttp(axiosServer);
      if (!checkUserRoleIncludes(jwtDto,
GENERATED_SUBTITLES_PERMISSIONS)) {
        return { notFound: true };
      }

      const page = context.query.page?.toString() ?? '';
      const response = await
axiosServer.get<IGeneratedSubtitlesViewAllDto>(
        '/generated-subtitles/view-all',
        { params: { page } }
      );
      return { props: { jwtDto, dto: response.data } };
    };

    export default function GeneratedSubtitlesViewAllPage({ dto
}: IProps) {
```

```
        const title = getTitleFromBreadcrumbs(
          GENERATED_SUBTITLES_VIEW_ALL_BREADCRUMBS
        );

        const router = useRouter();

        const onSuccessfulDelete = useCallback(() => {
          router.replace(router.asPath);
        }, [router]);

        const mutation = useMutation({
          mutationFn: generatedSubtitlesDeleteSendHttp,
          onSuccess: onSuccessfulDelete,
        });

        return (
          <Layout>
            <Head>
              <title>{title}</title>
            </Head>

            <Breadcrumbs
breadcrumbs={GENERATED_SUBTITLES_VIEW_ALL_BREADCRUMBS} />

            <H1>{title}</H1>

            <AuthGuard includes={CREATE_PERMISSIONS}>
              <div className="my-8">
                <LinkButton
                  href="/generated-subtitles/create"
                  buttonType={ButtonTypes.ALTERNATIVE}
                  testId="create_link"
                >
                  Upload a new video to generate subtitles
                </LinkButton>
              </div>
            </AuthGuard>

            <MutationDisplay mutation={mutation} />

            <GeneratedSubtitlesViewAllTable
              items={dto.items}
              handleDelete={(id) => {
                mutation.mutate(id);
              }}
            />

            <Pagination
              currentPage={dto.pagination.currentPage}
              totalPages={dto.pagination.totalPages}
            />
          </Layout>
        );
```

```
    }
```

Код – Сторінка пошуку тексту в субтитрах (ReactJS).

```
    interface IProps extends IAuthContextProps {
      defaultValues: ISubtitlesSearchForm;
      dto: ISubtitlesSearchResponseDto;

      showsOptions: IOption[];
    }

    export const getServerSideProps: GetServerSideProps<IProps>
= async (
      context
    ) => {
      const axiosServer = getAxiosServer(context);
      const jwtDto = await getJwtDtoHttp(axiosServer);

      const page = context.query?.page?.toString() ?? '1';
      const show_id = context?.query?.show_id?.toString() ?? '';
      const text = context.query?.text?.toString() ?? '';

      const dto = await getSubtitlesSearchResultDto(axiosServer,
Number(page), {
        text,
        show_id,
      });

      const showsOptions = await
getShowsOptionsHttp(axiosServer);
      const moviesOptions = await
getMoviesOptionsHttp(axiosServer);

      const defaultValues: ISubtitlesSearchForm = {
        text,
        show_id,
      };

      return {
        props: {
          jwtDto,
          defaultValues,
          dto,
          showsOptions: [...showsOptions, ...moviesOptions],
        },
      };
    };

    export default function SubtitlesSearchPage({
      defaultValues,
      dto,
      showsOptions,
    }: IProps) {
```

```
        const form = useForm<ISubtitlesSearchForm>({
          defaultValues: defaultValues,
          resolver: zodResolver(SUBTITLES_SEARCH_FORM_SCHEMA),
        });

        const router = useRouter();

        const isRouterLoading = useIsRouterLoading();

        const onSubmit = (data: ISubtitlesSearchForm) => {
          router.push({
            pathname: router.pathname,
            query: { text: data.text, show_id: data.show_id, page:
'1' },
          });
        };
        const title = `Search for: ${defaultValues.text}`;

        return (
          <Layout>
            <Head>
              <title>{title}</title>
            </Head>

            <Breadcrumbs

breadcrumbs={getSubtitlesSearchBreadcrumbs(defaultValues.text)}
            />

            <H1>{title}</H1>

            <form onSubmit={form.handleSubmit(onSubmit)}
className="w-full">
              <Fieldset>
                <Label htmlFor="text">Text</Label>
                <Input id="text" {...form.register('text')} />
                <HelperTextError>
                  {form.formState.errors?.text?.message}
                </HelperTextError>
              </Fieldset>
              <Fieldset>
                <Label htmlFor="show_id">Media</Label>
                <Select
                  id="show_id"
                  options={showsOptions}
                  {...form.register('show_id')}
                />
                <HelperTextError>
                  {form.formState.errors?.show_id?.message}
                </HelperTextError>
              </Fieldset>
              <Button type="submit" isFullWidth={true}>
                <SearchIcon />
```

```
              Search
            </Button>
          </form>

          {isRouterLoading ? <AlertInfo>Loading...</AlertInfo> :
null}

          {defaultValues.text.length > 0 &&
dto.pagination.totalPages === 0 ? (
            <AlertWarning>No matches found.</AlertWarning>
          ) : null}

          <div id="results" className="flex flex-col gap-4 py-
4">
            {dto.items.map((item) => {
              return (
                <div
                  key={item.subtitles_id}
                  className="p-4 shadow-sm overflow-auto max-h-
[900px]"
                >
                  <Link
                    href={item.href}
                    className="font-bold text-lg pb-4 text-blue-
500"
                  >
                    {item.name}
                  </Link>

                  <Link

href={`/subtitles/view/${item.subtitles_id}`}
                    className="block text-blue-500 mb-4"
                  >
                    {item.subtitles_name}
                  </Link>

                  {item.nodes.map((nodeList, nodeListIndex) => {
                    return (
                      <div key={nodeListIndex} className="mb-4">
                        <SubtitlesViewTable nodes={nodeList} />
                      </div>
                    );
                  })}
                </div>
              );
            })}
          </div>

          {dto.pagination.totalPages > 0 ? (
            <Pagination
              currentPage={dto.pagination.currentPage}
              totalPages={dto.pagination.totalPages}
```

```
          />
        ) : null}
      </Layout>
    );
  }
```