Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська дипломна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ПІДБОРУ ФІЛЬМІВ

Виконала: студентка 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Лишак А.М.
(прізвище та ініціали)
Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)
« 07 » червня 2024 р.

Рецензент: к.т.н., доцент каф. АІІТ
Богач І. В.
(прізвище та ініціали)
« 07 » червня 2024 р.

Допущено до захисту
Зав. кафедри Яровий А. А.
« 07 » червня 2024 р.

Вінниця ВНТУ - 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Яровий А.А.
« 07 » 03. 2024 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТЦІ
Лишак Альоні Михайлівні

1. Тема роботи: Програмний модуль підбору фільмів.
керівник роботи: Крилик Людмила Вікторівна, к.т.н., доц.
затверджені наказом вищого навчального закладу « 11 » 03. 2024 року № 80

2. Термін подання студентом роботи 27.05.2024 р.

3. Вихідні дані до роботи:
Мова програмування – об'єктно-орієнтована;
Кількість можливих кімнат – більше 1;
Кількість підтримуваних платформ – 3;
Інтерфейс користувача має бути інтуїтивно зрозумілий.

4. Зміст текстової частини (перелік питань, які потрібно розробити):
вступ; обґрунтування доцільності розробки програмного модуля підбору фільмів; проектування програмного модуля підбору фільмів; програмна реалізація модуля підбору фільмів; висновки; список використаних джерел; додатки.

5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): схема алгоритму роботи програмного модуля підбору фільмів; загальна структурна схема функціонування системи; загальна структурна схема компонентів програмного модуля підбору фільмів; UML-діаграма класів серверної частини модуля підбору фільмів; UML-діаграма класів клієнтської частини модуля підбору фільмів; ER-модель для бази даних програмного модуля підбору фільмів; приклад роботи програми.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1–3	Крилик Л. В., к.т.н., доц. каф. КН		

7. Дата видачі завдання 07.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Вступ. Обґрунтування доцільності розробки програмного модуля підбору фільмів	06.05.2024	09.05.2024	
2	Проектування програмного модуля підбору фільмів	10.05.2024	14.05.2024	
3	Програмна реалізація модуля підбору фільмів	15.05.2024	21.05.2024	
4	Висновки та аналіз отриманих результатів	22.05.2024	24.05.2024	
5	Оформлення додатків	27.05.2024	29.05.2024	
6	Розробка інструкції користувача	30.05.2024	03.06.2024	
7	Оформлення матеріалів до захисту БДР	04.06.2024	06.06.2024	

Студентка _____ **Лишак А. М.**
(підпис) (прізвище та ініціали)

Керівник роботи _____ **Крилик Л. В.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 112 сторінок формату А4, які включають 26 рисунків і 7 таблиць. У списку використаних джерел зазначено 21 найменування.

Метою роботи є розширення функціональних можливостей програмного модуля підбору фільмів.

У загальній частині роботи на основі аналізу існуючих технічних рішень, методів та технологій доведена доцільність розробки програмного модуля підбору фільмів. Розроблено алгоритм роботи та UML діаграми класів клієнтської та серверної частин програмного модуля підбору фільмів, які спрощують розробку та розуміння структури програмного забезпечення. Програмний продукт розроблений в інтегрованому середовищі Visual Studio Code.

Розроблено програмний продукт та проведено його тестування. Результати тестування розробки вказують на те, що основні функції програмного модуля підбору фільмів реалізовано.

Ключові слова: програмний модуль, WEB-розробка, мобільні додатки, фільми.

ABSTRACT

The bachelor's thesis consists of 112 A4 pages, including 26 figures and 7 tables. The list of references includes 21 items.

The purpose of the work is to expand the functionality of the movie selection software module.

In the general part of the work, based on the analysis of existing technical solutions, methods and technologies, the expediency of developing a movie selection software module is proved. The algorithm of work and UML diagrams of classes of the client and server parts of the movie selection module are developed, which simplify the development and understanding of the software structure. The software product was developed in the integrated Visual Studio Code environment.

The software product has been developed and tested. The results of the development testing indicate that the main functions of the movie selection software module have been implemented.

Keywords: software module, WEB development, mobile applications, movies.

ЗМІСТ

ВСТУП

Актуальність досліджень.

Із зростанням популярності онлайн-стрімінгових платформ і постійною появою нових фільмів, тема «Програмний модуль підбору фільмів» є дуже актуальною в сучасному світі. Актуальність цієї теми підкреслюють такі аспекти:

- Персоналізований вибір контенту: споживачі все частіше прагнуть індивідуалізованого досвіду перегляду, з рекомендаціями фільмів на основі їхніх уподобань та історії переглядів. Дедалі важливішою стає розробка програмних модулів, які аналізують уподобання користувачів і надають рекомендації.

- Збільшення обсягів контенту: з кожним роком кіновиробництво стає різноманітнішим і доступнішим. Програмний модуль підбору фільмів може допомогти користувачам орієнтуватися у великій кількості фільмів і знаходити ті, які їм потрібні.

- Зростання конкуренції в індустрії: онлайн-стрімінгові платформи змагаються за увагу глядачів. Розробка ефективного програмного модуля підбору фільмів може стати конкурентною перевагою для таких платформ.

Отже, тема «Програмний модуль підбору фільмів» є важливою та актуальною в контексті сучасних тенденцій розвитку ринку розваг та медіа, і може бути цікавою для дослідження та розробки.

Метою дослідження є розширення функціональних можливостей програмного модуля підбору фільмів.

Об’єктом дослідження є процес підбору фільмів.

Предметом дослідження є програмні засоби для реалізації програмного модуля підбору фільмів.

Задачі дослідження:

1. Обґрунтувати доцільність розробки програмного модуля підбору фільмів;
2. Спроектувати програмний модуль підбору фільмів;

3. Програмно реалізувати модуль підбору фільмів;

4. Виконати тестування програми та провести аналіз отриманих результатів.

5. Розробити інструкцію користувача.

Апробація роботи.

Результати досліджень було апробовано на LIII Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (НТКП ВНТУ) м. Вінниці у 2024 р. [1].

Публікації: за основними результатами досліджень опубліковано тези доповіді на науково-технічній конференції [1]; подано заявку на реєстрацію авторського права на твір у формі комп'ютерної програми – «Програмний модуль підбору фільмів», номер заявки С202404448 від 07.06.2024 р.

1 ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ ПРОГРАМНОГО МОДУЛЯ ПІДБОРУ ФІЛЬМІВ

У сучасному цифровому світі, де ринок розваг постійно зростає, а конкуренція між онлайн-платформами для перегляду фільмів стає дедалі гострішою, розробка програмних модулів для підбору фільмів стає не лише актуальною, але й дуже доречною. Цей модуль є ключовим елементом стратегії залучення та утримання аудиторії та оптимізації вмісту платформи. Нижче наведено переваги, які це принесе користувачам і платформам перегляду:

1. Персоналізація контенту: сучасна аудиторія має різні смаки та вподобання. Розробка програмного модуля для аналізу вподобань користувачів і рекомендацій фільмів на основі цих уподобань, забезпечує персоналізований досвід перегляду, що є ключовим фактором для задоволення користувачів і збереження їхньої вірності платформі.

2. Підвищення залученості користувачів: забезпечення зручного та ефективного способу вибору фільмів привертає більше користувачів на платформу. Коли користувачі отримують рекомендації, які відповідають їхнім інтересам, вони, як правило, проводять більше часу на платформі та використовують її частіше.

3. Ефективне управління вмістом: розробка даного програмного модуля дозволяє платформі використовувати свій вміст більш ефективно. Аналізуючи взаємодію користувачів із фільмами, ви можете зрозуміти їхні вподобання та придбати або створити контент, який найбільше цікавить вашу цільову аудиторію.

4. Конкурентна перевага: у світі зростаючої конкуренції серед платформ перегляду фільмів розробка програмного модуля підбору фільмів може стати ключовою конкурентною перевагою. Платформи, які надають ефективніші та персоналізовані рекомендації, здатні залучити й утримати більше користувачів.

Таким чином, розробка програмного модуля підбору фільмів забезпечує значні переваги для користувачів і платформ перегляду фільмів, забезпечуючи ефективне керування вмістом і покращений досвід користувача.

1.1 Огляд середовищ для реалізації програмного продукту

Вибір ідеального середовища для реалізації програмного продукту є важливим моментом у процесі розробки. Враховуючи різноманітність платформ та їхніх можливостей, важливо робити свій вибір на основі потреб користувачів, особливостей та функціональних вимог.

В таблиці 1.1 подано порівняльна характеристика платформ для розробки програмного модуля підбору фільмів.

Таблиця 1.1 – Порівняльна характеристика платформ для розробки програмного модуля підбору фільмів

	ПК	Мобільні пристрої	WEB-застосунки
Доступність	Зручний доступ на робочому місці, вдома або в офісі	Мобільний доступ з будь-якого місця	Зручний доступ з будь-якого пристрою з Інтернетом
Операційні системи	Windows, macOS, Linux	Android, iOS	Сумісність з будь-яким сучасним WEB-браузером
Охоплення користувачів	Менше охоплення порівняно з телефонами	Велике охоплення через популярність мобільних пристроїв	Велике охоплення через універсальність доступу
Встановлення	Потребує установки окремого додатку на ПК	Потребує установки мобільного додатку на пристрій	Не потребує установки, доступ через WEB-браузер

Продовження таблиці 1.1

	ПК	Мобільні пристрої	WEB-застосунки
Зручність використання	Обмежена мобільність, залежність від ОС	Компактність, портативність, залежність від ОС	Універсальність доступу
Екосистема	Різноманітна, залежить від платформи з фреймворками та інструментами, такими як .NET Core, Electron, Qt	Розділена між Android і iOS, обмеженіша	Велика кількість бібліотек React, Angular, Vue.js, Django, Flask, Laravel
Мови програмування	Не обмежений (Java, C++, C#, Python, JavaScript тощо)	Обмежений − Java, Kotlin (Android), Swift, Objective-C (iOS)	Обмежений для FrontEnd (JavaScript/Typescript, WASM). Необмежений для BackEnd (JavaScript/Typescript, Python, PHP, Ruby, .NET)

WEB-застосунки є одним із найпопулярніших середовищ для розробки платформ для підбору і перегляду фільмів. Вони забезпечують зручний доступ з будь-якого пристрою з Інтернетом.

Важливою перевагою WEB-застосунків є їх універсальність. Вони працюють на різних операційних системах та можуть бути використані як на комп'ютерах, так і на планшетах або смартфонах.

WEB-застосунки також забезпечують достатню функціональність для більшості потреб користувачів і не потребують встановлення спеціальних додатків, достатньо мати WEB-браузер, а він є на більшості пристроїв.

На основі аналізу даних з таблиці 1.1, стає очевидним, що WEB-застосунки є зручним та універсальним середовищем для розробки програмного продукту і є найкращим варіантом в нашому випадку. Однак, для ще більшого охоплення

користувачів та зручності в користуванні, цей програмний модуль буде реалізовано як для WEB-браузерів, так і для мобільних пристроїв

1.2 Огляд систем-аналогів

Існує велика кількість програмних продуктів з подібними можливостями, і кожен з них містить свої переваги та недоліки. Тому варто провести аналіз систем-аналогів для того, щоб визначити перспективи для розвитку та покращення, розроблюваного модуля.

Для порівняння, розглянемо такі сервіси, як «Popcorn», «Letterboxd», «IMDb» [2 − 4].

Popcorn – це мобільний додаток, який пропонує фільми для перегляду на основі особистих вподобань (рис. 1.1).

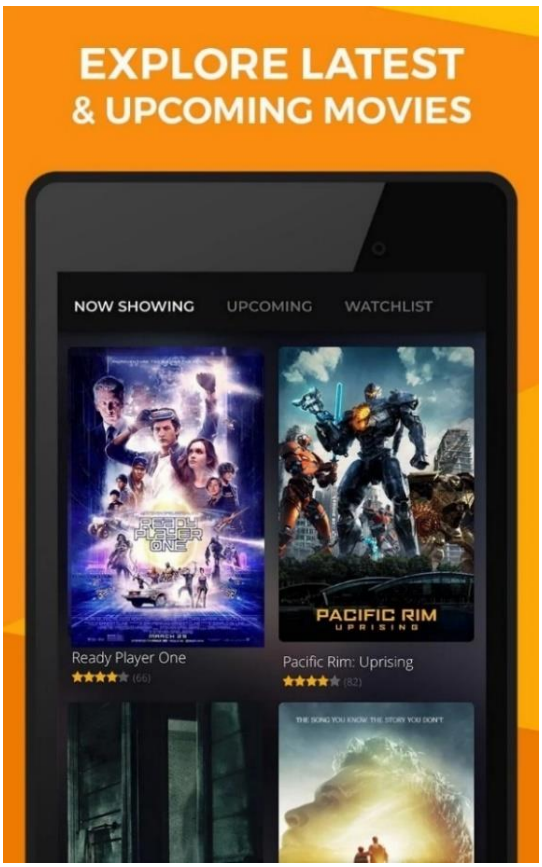

Рисунок 1.1 − Загальний вигляд інтерфейсного вікна мобільного додатку Popcorn

Особливості мобільного додатку Popcorn:

- персоналізований підбір фільмів;
- перегляд детальної інформації про фільм;
- створення персоналізованого або групового списку вподобаних фільмів; можливість ділитися списком обраних фільмів з друзями [2].

Недоліки:

- відсутність української локалізації;
- немає в списку підтримуваних країн України;
- в додатку є реклама, яка займає значну частину екрана.

Letterboxd – платформа для пошуку фільмів, відстеження переглянутих фільмів та обміну списком рекомендованих фільмів з друзями (рис. 1.2).

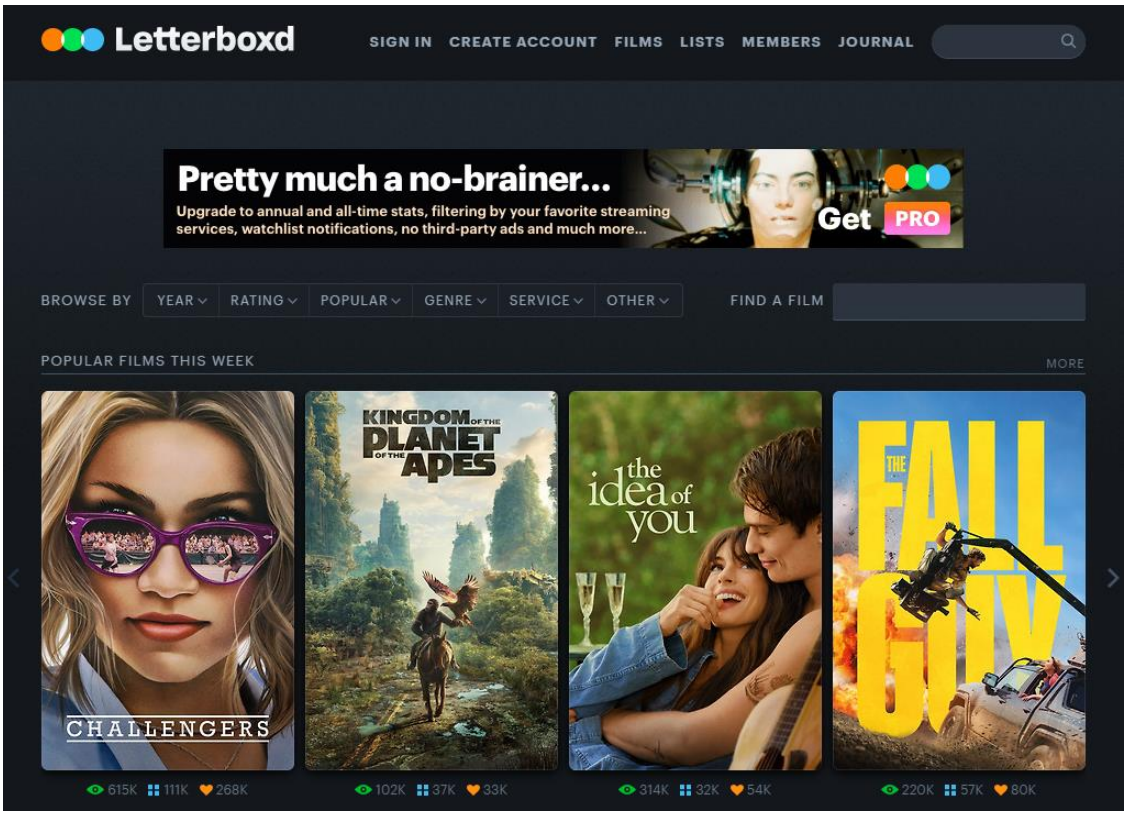

Рисунок 1.2 – Загальний вигляд інтерфейсного вікна платформи Letterboxd

Особливості платформи Letterboxd:

- створення, перегляд та редагування історії переглянутих фільмів;
- створення списку вподобаних фільмів;

- оцінювання фільмів за п'ятизірковою шкалою включаючи коментування фільмів;
- персоналізовані рекомендації [3].

Недоліки платформи Letterboxd:

- обмеженість безкоштовної версії;
- містить нав'язливу рекламу;
- відсутність локалізації (інтерфейс доступний лише англійською мовою).

IMDb – платформа, яка надає інформацію про рейтинги, коментарі та рецензії фільмів. Крім того, платформа є великою онлайн базою фільмів, серіалів, акторів та кінематографістів (рис. 1.3).

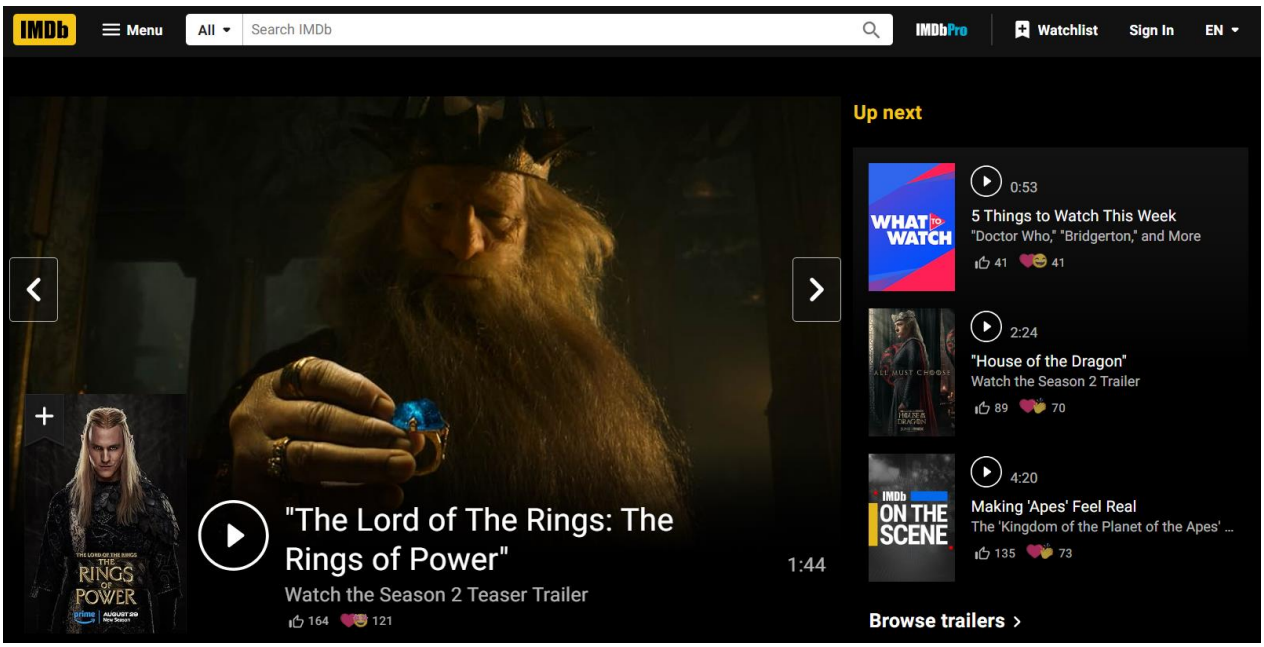

Рисунок 1.3 – Загальний вигляд інтерфейсного вікна платформи IMDb

Особливості платформи IMDb:

- можливість залишати оцінки на фільми та серіали;
- велика база даних фільмів та серіалів;
- перегляд трейлерів;
- персоналізовані рекомендації;
- новини та інтерв'ю;

- платформа доступна як WEB та мобільний додаток [4].

Недоліки платформи IMDb:

- незручність пошуку;
- нестабільність рейтингів.

1.3 Формулювання вимог та постановка задачі

Щороку кількість фільмів та серіалів зростає, що призводить до ускладнення вибору перегляду фільму чи серіалу. Ще складніше стає підібрати фільм чи серіал для перегляду декількома людьми, адже в кожного свої смаки та інтереси, одним можуть подобатися певні жанри, іншим – ні. Вибір фільму чи серіалу для перегляду декількома людьми часто призводить до довгих обговорень та суперечок. Тож програмне забезпечення для підбору фільму за вподобаннями користувача чи декількох користувачів вирішило би такі проблеми.

IMDb та Letterboxd є найбільшими платформами з великими базами даних фільмів та серіалів. Ці платформи надають можливість залишати оцінки та коментарі до фільмів, а також підбирають фільми за персоналізованими вподобаннями. IMDb додатково доступна як додаток на мобільних пристроях. Однак ці платформи мають недоліки. Обидві системи не надають можливість пошуку фільмів за вподобаннями декількох людей, що не вирішує проблему спільного підбору фільмів чи серіалів. Крім того, в Letterboxd є сильно нав’язлива реклама, а платформа IMDb містить дуже не зручну систему пошуку, а рейтинги до фільмів є нестабільними, коли з’являються перші оцінки.

Альтернативою наведених платформ є мобільний додаток Popcorn, який надає можливість також підбирати фільми за вподобаннями, а також ділитися списком відібраних фільмів з друзями. Проте як і IMDb та Letterboxd платформи, мобільний додаток Popcorn немає української локалізації. Крім того, мобільний додаток Popcorn не працює для України (в списку підтримуваних країн немає України).

Програмний модуль підбору фільмів матиме такі функції:

- Авторизація та аутентифікація користувача.
- Створення кімнати для пошуку фільму для спільного перегляду.
- Редагування існуючої кімнати.
- Пропозиція фільму відносно вподобань користувачів кімнати.
- Сторінка із списком популярних на сервісі фільмів.

У роботі потрібно створити WEB-ресурс та мобільний додаток, який дасть можливість користувачам здійснювати пошук фільмів для спільного та індивідуального перегляду.

WEB-ресурс та мобільний додаток мають бути інтуїтивно зрозумілими для користувача, тому будуть розроблятись за допомогою користувацького досвіду (UX). Вхідні дані мають легко задаватися користувачем за допомогою інтуїтивного та зрозумілого інтерфейсу. Вихідні дані мають бути зрозуміло представлені, щоб користувач міг без зайвих зусиль знайти потрібну йому інформацію.

У розроблюваному модулі підбору фільмів, зручність є дуже важливим аспектом. Такий програмний модуль має забезпечувати користувачам можливість легко та швидко знаходити цікавий фільм або серіал як для індивідуального перегляду, так і для перегляду групою осіб, враховуючи всі їх вподобання.

Для написання коректного інтуїтивного модуля для підбору фільмів, потрібно вирішити такі завдання:

- Реалізувати модуль аутентифікації та авторизації;
- Реалізувати модуль, який рекомендує фільми за вподобаннями користувача;
- Реалізувати модуль створення кімнати для спільного перегляду;
- Реалізувати модуль, який рекомендує фільми за вподобаннями користувачів кімнати;
- Реалізувати модуль для редагування існуючої кімнати;
- Здійснити тестування програмного модуля підбору фільмів.

1.4 Висновок

В цьому розділі проаналізовано існуючі платформи для реалізації програмного продукту. Було вирішено реалізувати програмний модуль для WEB-браузерів та мобільних пристроїв, враховуючи їх універсальність, функціональність та велике охоплення користувачів. Проведено аналіз систем-аналогів, подібні додатки не повністю задовольняють потреби користувачів, адже більшість з них концентрується лише на індивідуальному підборі фільму.

Програмний модуль для підбору фільмів є актуальним, оскільки велика кількість людей для дозвілля обирає саме перегляд фільмів, а враховуючи їх велику кількість та різні вподобання людей, обрати фільм є не легким завданням. Тож цей програмний продукт допоможе з легкістю підібрати фільм для одного користувача або декількох користувачів, враховуючи всі вподобання. Такий ресурс допоможе економити велику кількість часу.

В цьому розділі також було поставлено задачі для подальшого дослідження та описано призначення розроблюваного модуля для підбору фільмів.

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО МОДУЛЯ ПІДБОРУ ФІЛЬМІВ

2.1 Розробка структури програмного модуля підбору фільмів

Програмний модуль підбору фільмів є WEB та мобільним застосунком, який складається з двох частин – клієнтської та серверної. Кожна із цих частин має певну структуру. Для спрощеної розробки кожна із частин програмного модуля підбору фільмів поділена на модулі. Для серверної частини ці модулі називаються мікросервісами, а архітектура проекту при їх використанні – мікросервісною.

Простими словами, мікросервісна архітектура – це архітектура, яка передбачає поділ великих застосунків чи служб на невеликі частини, які можна окремо один від одного розробляти та підтримувати, а збій одного чи декількох сервісів не призведе до глобального збою всього застосунку. Основне правило мікросервісної архітектури – слабка пов’язаність між сервісами. В мікросервісній архітектурі кожен сервіс є самодостатнім та виконує якусь певну функцію. Та сама функція не має виконуватися в більше чим одному сервісі. Для комунікації між службами в програмному модулі підбору фільмів використовується загальний інтерфейс, а саме API [5].

Оскільки мікросервісна архітектура передбачає слабку пов’язаність між сервісами, то сервіси можуть бути розміщені в ізольованому просторі. Такий ізольований простір називається контейнером. Контейнери дозволяють розробнику упакувати програму з усіма потрібними частинами, наприклад, бібліотеками та іншими залежностями, і відправити та запустити все як один пакет. Контейнери можуть бути запущені в декількох екземплярах для балансування навантаження.

Кожен модуль має унікальну назву та може мати відміну структуру файлів та папок. Модульна структура дозволяє налагодити організовану розробку програмного забезпечення, зробити її простішою.

Для програмного модуля підбору фільмів створені такі модулі:

- Модуль авторизації, реєстрації та кабінет користувача.
- Модуль пошуку та фільтрації фільмів, отримання опису про фільм.
- Модуль для роботи з кімнатами для колективного підбору фільмів.

Кожен модуль складається із сторінок. Модуль підбору фільмів складається з таких сторінок: авторизації, реєстрації нового користувача, перегляду та пошуку фільмів, створення кімнати та колективного підбору фільмів у вибраній чи створеній кімнаті. Загальна структурна схема функціонування програмного модуля підбору фільмів зображена на рисунку 2.1.

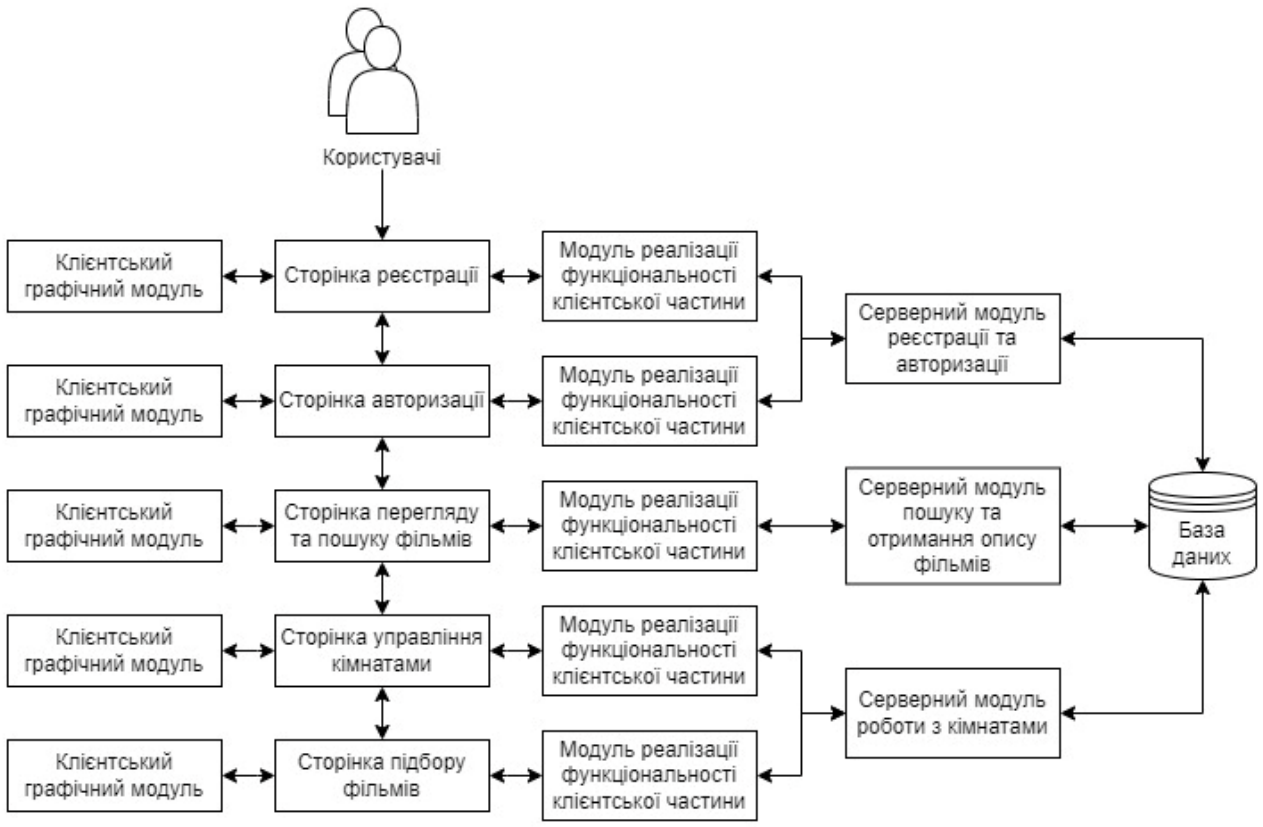

Рисунок 2.1 − Загальна структурна схема функціонування системи

На сторінці реєстрації користувач може створити новий обліковий запис, використовуючи свою електрону пошту. Користувач водить свої дані, після чого вони передаються на сервер за допомогою модуля для реалізації клієнтської

частини. Потім запит обробляється мікросервісом, який відповідає за реєстрацію користувачів.

Після реєстрації користувач може авторизуватися на відповідній сторінці. Користувач водить свою пошту та пароль у відповідні поля, потім за допомогою модуля для реалізації клієнтської частини, на сервер відправляється відповідний запит. Якщо пароль відповідає введеній електронній пошті – користувач завершує авторизацію і сервер переадресовує користувача на головну сторінку із фільмами.

Користувач може перейти на сторінку з кімнати для вибору або створення кімнати для колективного підбору фільму. Для створення кімнати користувач водить назву кімнати і вибирає користувачів. Запити до сервера для роботи із кімнатами опрацьовуються окремим модулем, який відповідає лише за роботу із кімнатами.

Сервер зберігає інформацію в реляційній базі даних. База даних зберігає інформацію про користувача, фільми та кімнати для спільного підбору фільмів. База даних забезпечує швидкий доступ до інформації та швидке оновлення і добавлення інформації.

Всі API запити від клієнтської частини до мікросервісів відбуваються через балансувальник навантаження. Задача балансувальника навантаження полягає в розподілені запитів від клієнтської частини між екземплярами відповідних мікросервісів, а також в направлені запитів на потрібний мікросервіс. Наприклад, запити на реєстрацію будуть перенаправленні балансувальником навантаження на мікросервіс для реєстрації користувачів.

Загальна структурна схема компонентів програмного модуля підбору фільмів зображена на рисунку 2.2.

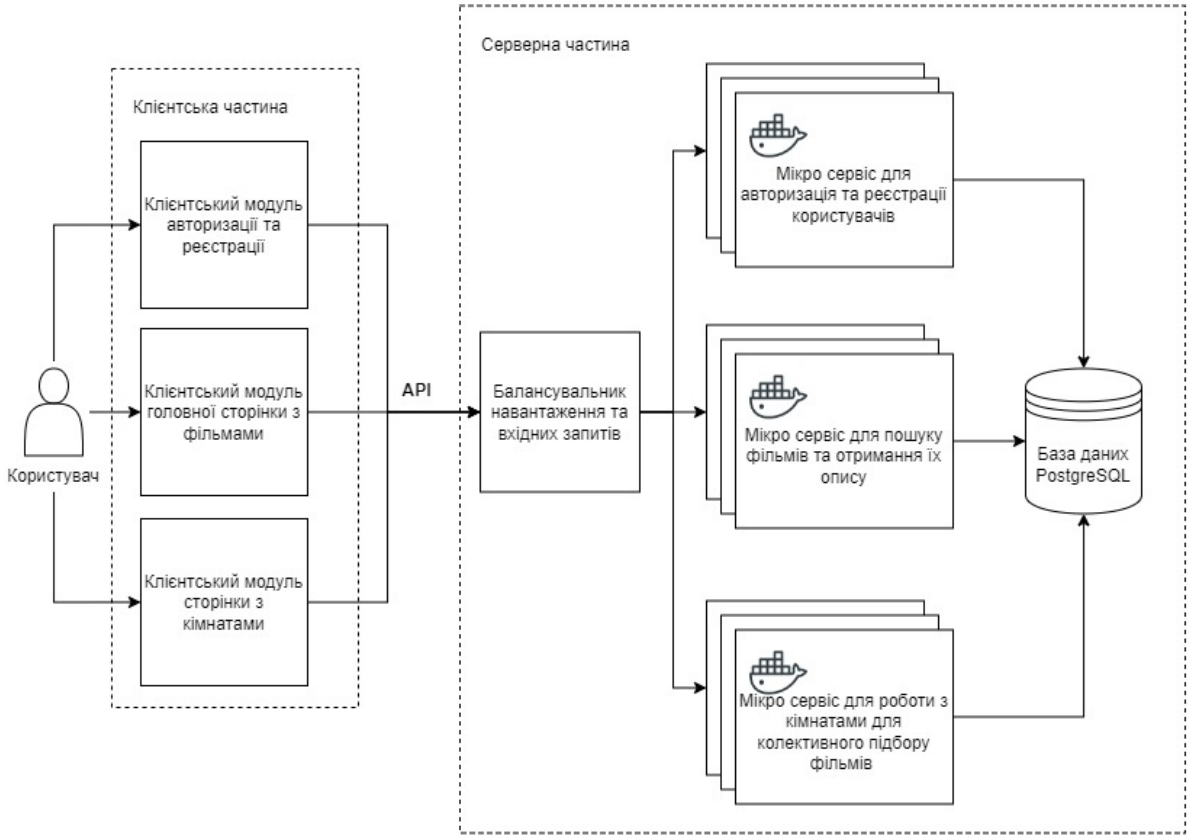

Рисунок 2.2 –Загальна структурна схема компонентів програмного модуля підбору фільмів

2.2 Розробка UML-діаграми класів

Розробка програм у мові програмування TypeScript базується на об’єктно-орієнтованому підході.

Об’єктно-орієнтоване програмування (ООП) — це підхід, який використовує поняття об’єктів і класів для визначення структури та функціональності програми. Цей підхід став основним у сучасному програмуванні та забезпечує високу ефективність у великих і складних проектах. ООП базується на декількох основних принципах, які роблять програми не тільки легшими для розробки, але й для розуміння та підтримки [6].

Основними принципами об’єктно-орієнтованого програмування є:

- Інкапсуляція: вимагає приховування деталей реалізації та надання лише інтерфейсу для взаємодії з об’єктом. Це дозволяє ізолювати зміни в одній частині програми від інших частин, роблячи код більш стабільним і стійким до змін.

- Наслідування: дає можливість створювати нові класи на основі існуючих класів. Це полегшує повторне використання коду та сприяє створенню ієрархій класів. Наслідування дозволяє наслідникам використовувати властивості та методи своїх предків і змінювати або розширювати їх за потреби [7].

- Поліморфізм: відноситься до здатності об'єктів різних класів мати спільний інтерфейс. Це дозволяє використовувати загальні методи та функції для роботи з різними типами об’єктів. Також завдяки поліморфізму код стає більш гнучким та розширюваним [8].

- Абстракція: процес виділення загальних властивостей об’єктів і встановлення абстрактних категорій або інтерфейсів для їх вираження. Абстракція спрощує модельну систему, полегшуючи її розуміння та керування [9].

У серверній частині програмного модуля для підбору фільмів є такі ключові класи: AuthController, MoviesController та RoomsController.

Клас AuthController містить такі методи: signIn(signInRequest), signUp(signUpRequest), getProfile(), refrestTokens(refrestTokenRequest). Метод signIn обробляє API метод, що надходить від клієнтської сторони для авторизації користувача. Метод signUp обробляє API метод, що надходить від клієнтської сторони для реєстрації користувача. Метод getProfile повертає інформацію про авторизованого користaвача. Метод refrestTokens оновлює токени доступу авторизованих користувачів.

Клас MoviesController містить такі методи: getTrendingDay(), getTrendingWeek(), findMovies(searchText). Метод getTrendingDay повертає список популярних фільмів на теперішній день. Метод getTrendingWeek повертає список популярних фільмів на теперішній тиждень. Метод findMovies виконує пошук фільму за його назвою.

Клас RoomsController відповідає за роботу із кімнати для спільного підбору фільмів та містить такі методи: createRoom(createRoomRequest), deleteRoom(roomId), addUserToRoom(userId, roomId), deleteUserFromRoom(userId, roomId), getUserRooms(roomId), getRoomById(roomId). Метод createRoom створює кімнату з вказаною назвою. Метод deleteRoom видаляє кімнату з вказаним ідентифікатором. Метод addUserToRoom добавляє користувача з вказаним ідентифікатором в кімнату з вказаним ідентифікатором. Метод deleteUserFromRoom видаляє користувача з вказаним ідентифікатором з кімнати з вказаним ідентифікатором. Метод getUserRooms повертає список користувачів для кімнати з вказаним ідентифікатором. Метод getRoomById повертає інформацію про кімнату з вказаним ідентифікатором, а також список запропонованих для перегляду фільмів.

На рисунку 2.3 зображена UML-діаграма класів серверної частини модуля підбору фільмів.

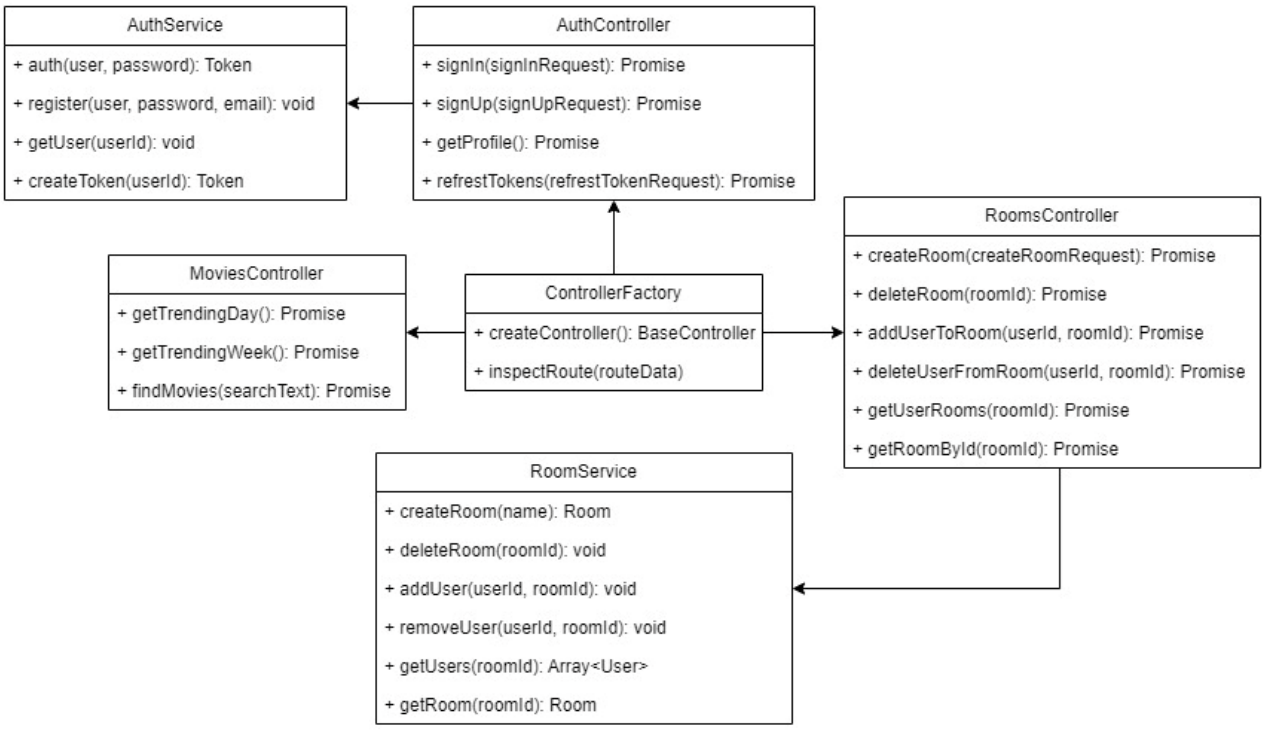

Рисунок 2.3 – UML-діаграма класів серверної частини модуля підбору фільмів

У клієнтській частині програмного модуля для підбору фільмів є такі ключові класи: LoginComponent, HeaderComponent, HomeComponent, RoomPageComponent, CreateRoomComponent, RoomSettingsComponent та RoomComponent. Всі перелічені класи наслідуються від базового класу Component.

Клас LoginComponent містить такі методи: login() та checkForError(controlName: string). Метод login відправляє API метод для аутентифікації користувача на сервері. checkForError метод перевіряє форму на помилки.

Клас HeaderComponent має тільки поля: user$ та title$. Поле user$ містить інформацію про авторизованого користувача, а поле title$ – назву сторінки, яку відкрив користувач.

Клас HomeComponent також має тільки поля: worldPopularDay та worldPopularWeek. Поле worldPopularDay містить список фільмів з описом, які є популярними на теперішній день, worldPopularWeek – список фільмів з описом, які є популярними на теперішній тиждень. Значення цих полів повинні бути отримані із сервера за допомогою API метода.

Клас RoomPageComponent мітить методи: getFilms() та nextFilm(). getFilms метод отримує список фільмів з описом, які відображаються користувачеві в кімнаті. Метод nextFilm переходить до перегляду опису наступного фільму.

Клас CreateRoomComponent відповідає за створення кімнати та має наступні поля: checkForError(controlName: string) та createRoom. Метод checkForError перевіряє валідність введених даних, а метод createRoom відправляє API метод з введеними даними для створення кімнати.

Клас RoomSettingsComponent містить наступні методи: findUsers(query: string), removeUser(user: IUser) та selectUser(event: MatAutocompleteSelectedEvent). Метод findUsers виконує пошук користувача по його імені для добавлення в кімнату. Метод removeUser видаляє користувача з кімнати, а метод selectUser навпаки, добавляє користувача в кімнату.

Клас RoomsComponent містить метод getRooms, який повертає список кімнат користувача.

На рисунку 2.4 зображена UML-діаграма класів клієнтської частини модуля підбору фільмів.

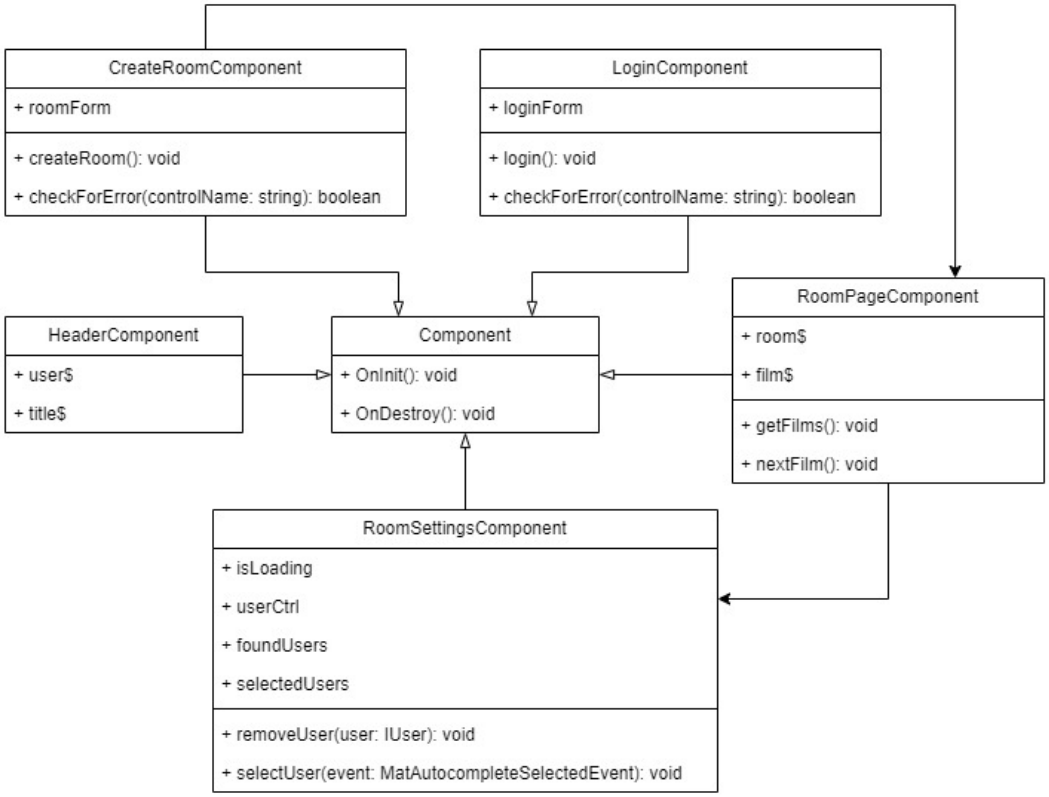

Рисунок 2.4 – UML-діаграма класів клієнтської частини модуля підбору фільмів

2.3 Розробка схеми алгоритму роботи програмного модуля підбору фільмів

Алгоритм – це впорядкована послідовність вказівок, виконання якої приводить від початкових даних до бажаного результату. Іншими словами алгоритм це план розв'язання задачі, записаний у вигляді послідовності вказівок чи груп вказівок.

Схема алгоритму − геометричне подання алгоритму, в якому оператори зображують послідовно у вигляді геометричних фігур та з’єднують стрілками.

При побудові схеми алгоритму використовуються загальноприйняті геометричні фігури, які використовуються для різних видів операторів [10].

Для опису алгоритму роботи програмного модуля підбору фільмів було вибрано представлення алгоритму у вигляді блоків схеми. Схему алгоритму зображено на рисунку 2.5.

І випадок

Алгоритм реєстрації користувача складається з таких кроків:

Крок 1. Користувач відкриває WEB-сторінку та якщо не авторизований, переходить до сторінки авторизації та реєстрації.

Крок 2. Користувач натискає на кнопку «Реєстрація».

Крок 3. Користувач вводить своє ім'я у відповідне поле форми реєстрації.

Крок 4. Користувач вводить свою електрону пошту у відповідне поле форми реєстрації.

Крок 5. Користувач вводить бажаний пароль у відповідне поле форми реєстрації.

Крок 6. Користувач натискає кнопку «Зареєструватися». Сервер вносить запис з даними користувача у базу даних.

ІІ випадок

Алгоритм аутентифікації користувача складається з таких кроків:

Крок 1. Користувач відкриває WEB-сторінку та якщо не авторизований, переходить до сторінки авторизації та реєстрації.

Крок 7. Користувач натискає на кнопку «Вхід».

Крок 8. Користувач вводить свою електрону пошту у відповідне поле форми аутентифікації.

Крок 9. Користувач вводить свій пароль у відповідне поле форми аутентифікації.

Крок 10. Користувач натискає кнопку «Увійти». Сервер перевіряє відповідність електронної пошти та пароля в базі даних.

Крок 11. Сервер перенаправляє користувача на сторінку «Головна».

III випадок

Алгоритм програмного модуля підбору фільмів складається з таких кроків:

Крок 1. Користувач відкриває WEB-сторінку та якщо авторизований, переходить до головної сторінки.

Крок 13. Користувач переходить на сторінку «Створити кімнату».

Крок 14. Користувач вказує ім’я кімнати та натискає кнопку «Далі».

Крок 15. На наступній сторінці користувач добавляє користувачів із списку. Після чого як у поточного, користувача так і у вибраних користувачів з’явиться кімната на сторінці «Кімнати».

Крок 16. Користувач добавляє додаткових користувачів у кімнату.

Крок 17. Сервер створює кімнату у базі даних.

Крок 18. Користувачам відображаються кімнати.

Крок 19. Користувачі вибирають відповідну кімнату на сторінці «Кімнати».

Крок 20. Сервер відображає довільний фільм з описом.

Крок 21. Користувачі вибирають чи сподобався їм фільм, чи ні.

Крок 22. Якщо користувачеві сподобався фільм, користувач натискає кнопку «Сподобалося».

Крок 23. Якщо користувачеві сподобався фільм, користувач натискає кнопку «Не сподобалося».

Крок 24. Сервер виконує пошук спільного вибору.

Крок 25. Якщо сервер не знайшов спільного вибору між користувачами кімнати, повернутися до кроку 20.

Крок 26. Сервер повертає всім користувачам вибраної кімнати опис з підібраним фільмом.

Алгоритм всіх трьох випадків зображено на рисунку 2.5.

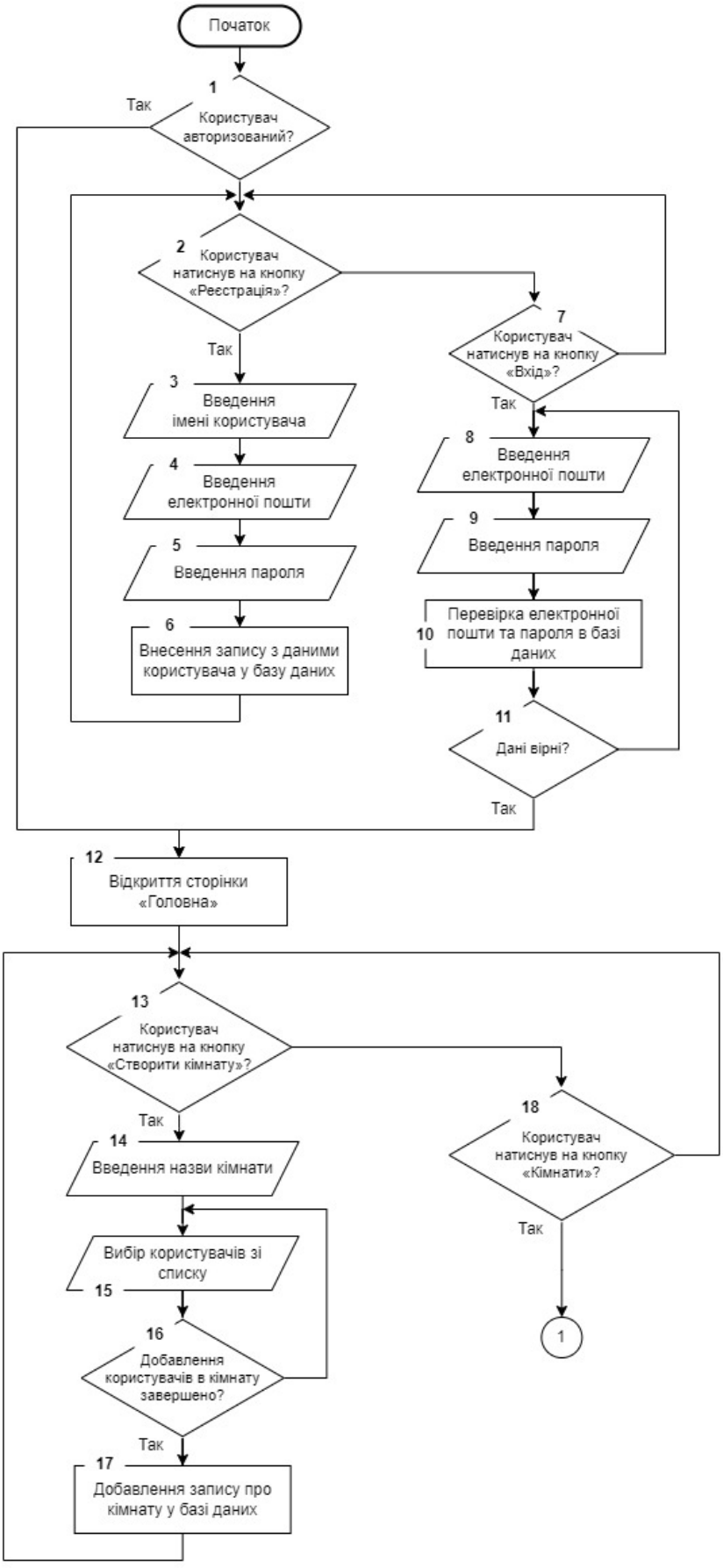

Рисунок 2.5 – Схема алгоритму роботи програмного модуля підбору фільмів

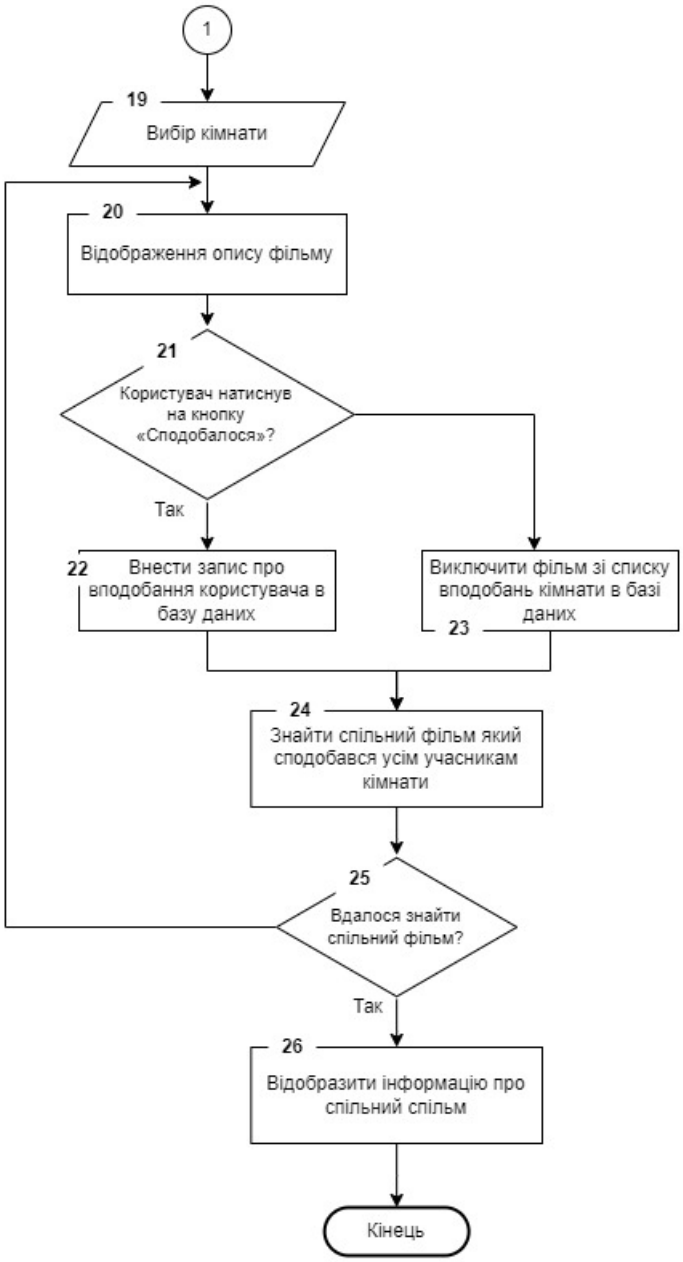

Рисунок 2.5, аркуш 2

2.4 Висновок

В цьому розділі було розроблено структуру програмного модуля підбору фільмів, вона дозволяє налагодити організовану розробку програмного забезпечення та зробити її простішою, завдяки чіткому уявленню, які компоненти буде містити програмний продукт, та як вони взаємодіятимуть між собою. Було визначено, що WEB-ресурс та мобільний додаток міститимуть такі сторінки: авторизації, реєстрації нового користувача, перегляду та пошуку

фільмів, створення кімнати та колективного підбору фільмів у вибраній чи створеній кімнаті.

Також було створено UML-діаграми класів для серверної та клієнтської частини модуля підбору фільмів та описано їх методи. Це дозволяє візуалізувати структуру програми, зобразити які є класи, які між ними взаємозв’язки та які методи містить кожен клас. UML-діаграми служать як документація для проекту, яку можуть використовувати для розуміння системи.

Крім того, було розроблено схему алгоритму функціонування програмного модуля підбору фільмів. Схема алгоритму допомагає зрозуміти покрокову роботу програми, цим самим значно полегшує розробку продукту та зменшує вірогідність неправильного функціонування. Крім того, добре спроектований алгоритм допомагає оптимізувати продуктивність програми, забезпечуючи швидке та ефективне виконання завдань.

Всі ці кроки є необхідними для ефективної розробки продукту, вони забезпечують чітке розуміння системи, допомагають організувати та спланувати роботу та дозволяють виявити проблеми на ранніх етапах. Все це сприяє успішній реалізації продукту, полегшує його підтримку та розвиток в майбутньому.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МОДУЛЯ ПІДБОРУ ФІЛЬМІВ

3.1 Обґрунтування вибору мови та бібліотек програмування

Програмна реалізація модуля підбору фільмів складається з клієнтського та серверного додатку. Тому в цьому розділі буде розглянуто порівняння мов програмування та обґрунтування їх вибору окремо для клієнтського та серверного додатку.

Для клієнтського додатку порівняємо мови програмування TypeScript та CoffeeScript.

TypeScript – це типізований набір JavaScript, спрямований на те, щоб зробити мову більш надійнішою та масштабованою. TypeScript має відкритий вихідний код і підтримується корпорацією Майкрософт з моменту його створення в 2012 році. Проте TypeScript отримав свій перший прорив, як основна мова програмування в таких фреймворках як Angular. Пізніше TypeScript почав розвиватися для таких фреймворків React та Vue [11].

CoffeeScript – це спрощена мова, яка компілюється в мову програмування JavaScript. Оскільки це мова програмування, яка компілюється в іншу мову програмування, інколи можна класифікувати CoffeeScript як транскомпілятор. CoffeeScript було створено через розчарування синтаксисом JavaScript. Синтаксис JavaScript має тенденцію ставати громіздким, багатослівним і, можливо, синтаксично безладним. З іншого боку, синтаксис CoffeeScript покращує читабельність і можливість запису еквівалентного коду [12].

У таблиці 3.1 наведено основні відмінності між мовами програмування TypeScript та CoffeeScript.

Таблиця 3.1 – Порівняння мов програмування TypeScript та CoffeeScript

	TypeScript	CoffeeScript
Статична типізація	+	–
Підтримка об’єктно орієнтованого програмування (ООП)	+	+
Популярність	+	–
Виявлення помилок на етапі компіляції	+	–
Ручне управління пам’яттю	–	–
Інтеграція з сучасними IDE	+	–
Спрощений синтаксис	–	+

Як видно з результатів порівняння у таблиці 3.1, TypeScript має перевагу над мовою програмування CoffeeScript. Основні переваги TypeScript: статична типізація, велика популярність, виявлення помилок на етапі компіляції та інтеграція з сучасними IDE. Тому для розробки клієнтської частини було обрано мову програмування TypeScript.

Для серверного додатку порівняємо мови програмування TypeScript, PHP та C++.

PHP (Hypertext Processor) – це мова програмування загального призначення та інтерпретатор, який має відкритий вихідний код та широко використовується для WEB-розробки. Мова використовується переважно для сценаріїв на стороні сервера, хоча її також можна використовувати для сценаріїв командного рядка та, в обмеженій мірі, для настільних програм.

У таблиці 3.2 наведено основні відмінності між мовами програмування TypeScript, PHP та C++.

Таблиця 3.2 – Порівняння мов програмування TypeScript, PHP та C++

	TypeScript	PHP	C++
Статична типізація	+	–	+
Підтримка об'єктно-орієнтованого програмування (ООП)	+	+	+
Популярність	+	+	+
Підтримує компіляцію	+	–	+
Висока продуктивність	–	–	+
Виявлення помилок на етапі компіляції	+	–	+
Інтеграція з сучасними IDE	+	–	+
Зворотна сумісність	+	+	+/–
Простий синтаксис	+	–	–
Інтеграція з базами даних	+	+	+

Як видно з результатів порівняння у таблиці 3.2, TypeScript має значну перевагу над мовою програмування PHP та більшу перевагу в порівнянні з C++ через повну зворотну сумісність з попередніми версіями. Також TypeScript більш простий у вивчені та використані ніж C++, це дозволяє швидше та простіше розробити додаток. Тож для розробки серверного додатку була вибрана мова програмування TypeScript, як і мова програмування для розробки клієнтського додатку.

Для розробки програмного забезпечення було використано бібліотеку Angular Material UI. Angular Material UI – це бібліотека, розроблена компанією Google, яка реалізує її методології щодо дизайну та взаємодії з користувачем у WEB-додатках. Бібліотека надає багато функцій для створення послідовного та сучасного дизайну, який дотримується суворих принципів [13].

Також було використано такі фреймворки:

- Angular – це фреймворк, який розроблений Google із відкритим вихідним кодом на основі TypeScript для односторінкових програм.

- Nest.js – це фреймворк для створення серверних додатків за допомогою JavaScript або TypeScript. Nest.js створений на основі Express.js.

3.2 Обґрунтування вибору мови програмування для управління організованою базою даних

Розглянемо мови програмування для управління організованою базою даних OQL (Object Query Language) та SQL (Structured Query Language).

Структурована мова запитів (SQL) – мова програмування для управління організованими базами даних, яка використовується для зберігання та обробки інформації в реляційних базах даних. Реляційна база даних зберігає інформацію в табличній формі, у якій стовпці є атрибутами сутності. SQL запити можна використовувати для отримання, оновлення, зберігання та видалення інформації з бази даних. Мова програмування SQL добре інтегрується з різними мовами програмування. Також мова програмування SQL досить проста у вивчені, оскільки використовує у своїх запитах загально прийняті англійські слова.

Традиційні бази даних розроблялися за реляційною моделлю, тобто дані розділені на умовні таблиці, які мають спільні ідентифікатори для створення відношень між записами у цих таблицях.

Мова запитів до об’єктів (OQL) – удосконалена версія мови структурованих запитів (SQL). Основною метою мови OQL є приховання схеми моделі даних, забезпечення безпечного та ефективного доступу до даних, використовуючи простий синтаксис. Синтаксис OQL подібний до синтаксису SQL.

Основна відмінністю між даними мовами програмування для управління організованими базами даних є те, що OQL повністю складається з виразів, включаючи вираз SELECT, який є основним виразом для запитів. SELECT не є інструкцією, а є виразом, тому його можна використовувати навіть у складних виразах із вкладенням. Це робить мову OQL дуже зручною для компонування,

де вирази можуть бути вкладені в декілька рівнів. У порівняні від SQL, OQL дозволяє оцінювати довільні складні ланцюжки запитів.

Проте під час використання методів, реалізованими мовою програмування OQL потрібно бути обережним, щоб не допустити мутацію даних та побічні ефекти. OQL є декларативним, тому порядок виконання виразів не завжди передбачуваний.

Основними перевагами OQL є:

- можливість запитувати будь-який довільний об’єкт з бази даних;
- можливість переміщатися по колекції об’єктів без завантаження всього списку об’єктів;
- підтримується відображення даних;
- немає обмеження схемою бази даних;
- можливість маніпулювати даними за допомогою методів.

У таблиці 3.3 наведено порівняння основних характеристик OQL (Object Query Language) та SQL (Structured Query Language).

Таблиця 3.3 – Порівняння основних характеристик OQL та SQL мов програмування для управління організованою базою даних.

	SQL (Structured Query Language)	OQL (Object Query Language)
Модель даних	Реляційна модель даних	Об’єктна модель даних
Синтаксис	Простий синтаксис для роботи із даними у формі таблиць	Схожий до SQL із підтримкою управління даними як об’єктами
Запити	Підтримка операція отримання, добавлення, оновлення та видалення записів у таблиці	Підтримка операцій роботи з об’єктами
Призначення	Реляційні бази даних	Об’єктно-орієнтовані бази даних

Продовження таблиці 3.3

	SQL (Structured Query Language)	OQL (Object Query Language)
Нормалізація даних	Необхідність нормалізації даних для уникнення або зменшення дублювання даних	Об’єкти зберігаються у формі посилання, тому не мають дублікатів
Відношення між записами та даними	Використання зовнішніх ключів	Використання посилань між об’єктами

Вибір між SQL та OQL залежить від конкретних потреб проекту. Якщо потрібно працювати з реляційними базами даних і забезпечити простоту та ефективність, SQL є кращим вибором. Отже, було вирішено обрати мову SQL.

3.3 Обґрунтування вибору середовища розробки

Інтегроване середовище розробки (IDE − Integrated Development Environment) – це програмне забезпечення, яке надає розробникам інструменти та функції, які допомагають у написанні, налагодженні та тестуванні програмного коду. Зазвичай IDE включає в себе текстовий редактор, налагоджувачі, інтегрований контроль версій, інструменти аналізу коду, інструменти автоматизації збірки та безліч інших корисних функцій, які значно полегшують та пришвидшують роботу розробника [14].

Під час вибору середовища розробки було розглянуто такі додатки: Visual Studio Code (VS Code), WebStorm, Atom.

Visual Studio Code − це легкий, але потужний редактор коду від компанії Microsoft, який доступний для Windows, macOS і Linux. Він має вбудовану підтримку JavaScript, TypeScript і Node.js та велику кількість розширень для інших мов і середовищ виконання (таких як C, C#, Java, Python, PHP, Go, .NET) [15].

WebStorm − інтегроване середовище розробки для JavaScript, TypeScript та суміжних технологій, є одним з найпопулярніших інструментів розробки

програмного забезпечення від JetBrains. Інтегровані інструменти розробника, які дозволяють налагоджувати та тестувати клієнтські та Node.js додатки, а також працювати з контролем версій, інструментами збірки, терміналом та HTTP-клієнтом [16].

Atom - це текстовий редактор і редактор коду, створений з інтеграцією HTML, JavaScript, CSS та Node.js. Він працює на Electron, фреймворку для створення крос-платформних додатків з використанням WEB-технологій. Є зручним середовищем розробки завдяки великій кількості налаштувань [17].

У таблиці 3.4 наведено порівняння основних характеристик Visual Studio Code (VS Code), WebStorm та Atom.

Таблиця 3.4 – Порівняння середовищ розробки Visual Studio Code (VS Code), WebStorm та Atom

	Visual Studio Code (VS Code)	WebStorm	Atom
Вартість	Безкоштовний	Платний	Безкоштовний
Продуктивність	Легкий і швидкий	Висока продуктивність, але важчий за VS Code	Може ставати повільним з великою кількістю пакетів
Інтерфейс користувача	Простий і зручний, легко налаштовується під індивідуальні потреби	Інтуїтивний інтерфейс, але більш насичений, що може бути важким для новачків	Простий інтерфейс, легко налаштовується, але менш потужний в деяких аспектах
Підтримка TypeScript	Відмінна підтримка TypeScript	Відмінна підтримка TypeScript	Підтримка через пакети, але менш інтегрована, ніж у VS Code та WebStorm

Продовження таблиці 3.4

	Visual Studio Code (VS Code)	WebStorm	Atom
Можливості	Інтелектуальне автозавершення, відлагоджувач, вбудований термінал, підтримка Git, багато розширень	Інтелектуальне автозавершення, аналіз коду в реальному часі, відлагоджувач, інструменти для тестування	Інтелектуальне автозавершення, підтримка Git, але менш потужні можливості відлагодження та рефакторингу

Згідно табл. 3.4, середовище розробки Visual Studio Code має перевагу над іншими середовищами розробки завдяки своїй безкоштовності, легкості, потужним інструментам для відлагодження та відмінній підтримці TypeScript. Тому було прийнято рішення обрати середовище Visual Studio для розробки програмного додатку.

3.4 Розробка ER-моделі бази даних

В універсальне відношення потрібно включити атрибути, що описують такі сутності: користувачі, рейтинги фільмів, фільми, кімнати та користувачі в кімнатах. Універсальне відношення для реалізації бази даних модуля підбору фільмів буде мати такий вигляд: R (код користувача, ім’я користувача, аватар користувача, електрона пошта користувача, пароль користувача, код фільму, рейтинг фільму, назва фільму, опис фільму, фото фільму, код кімнати, назва кімнати, код адміністратора кімнати, код користувача кімнати). Ступінь універсального відношення – 14.

Між двома сутностями А та В у базі даних програмного модуля підбору фільмів виділено такі типи зв’язку: зв’язок ОДИН-ДО-БАГАТЬОХ (1:Б) використовується між сутностями «Користувачі»-«Рейтинги фільмів», «Користувачі»-«Користувачі в кімнатах», «Фільми»-«Рейтинги фільмів», «Кімнати»-«Користувачі в кімнатах». зв’язок БАГАТО-ДО-БАГАТЬОХ (Б:Б) використовується між сутностями «Кімнати» - «Фільми». Характеристики

відношень між сутностями програмного модуля підбору фільмів представлено у таблиці 3.5.

Таблиця 3.5 – Характеристики відношень між сутностями програмного модуля підбору фільмів

Ім’я сутності 1	Ім’я сутності 2	Тип зв’язку	Клас належності
Користувачі	Рейтинги фільмів	1:Б	Необов’язковий
Користувачі	Користувачі в кімнатах	1:Б	Необов’язковий
Фільми	Рейтинги фільмів	1:Б	Обов’язковий
Кімнати	Користувачі в кімнатах	1:Б	Обов’язковий
Кімнати	Фільми	Б:Б	Необов’язковий

Представлення ER-моделі для бази даних програмного модуля підбору фільмів зображено на рисунку 3.1.

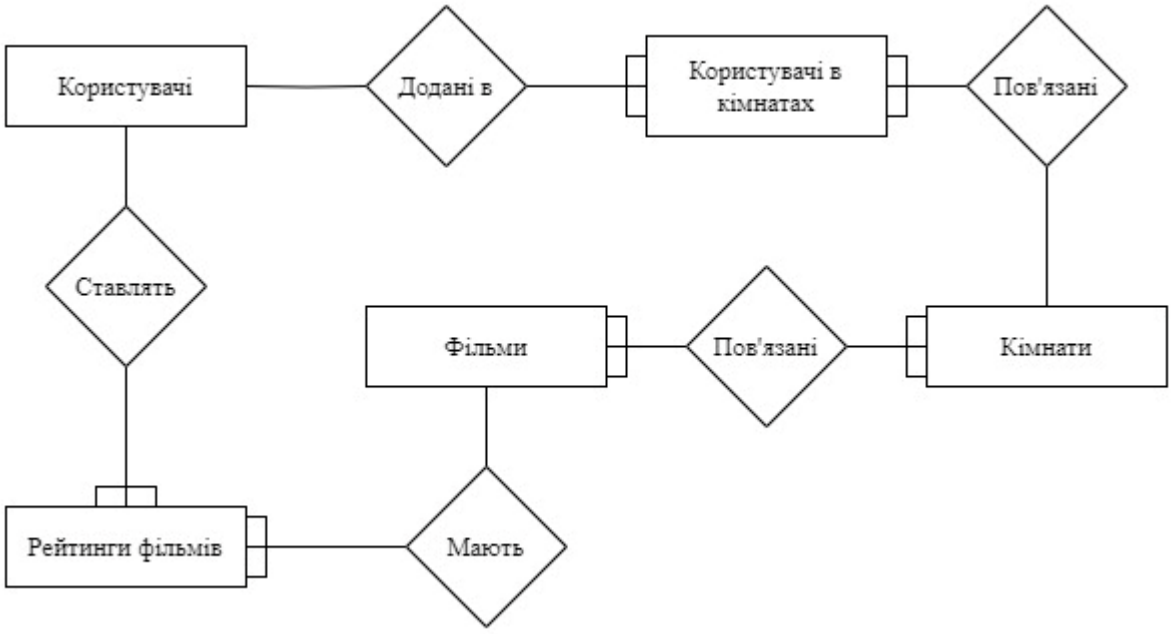

Рисунок 3.1 –ER-модель для бази даних програмного модуля підбору фільмів

3.5 Розробка клієнтської та серверної частин

Для розробки програмного модуля використано фреймворки nest.js та Angular 17.

Nest.js − це прогресивний фреймворк для створення серверних застосунків Node.js, який використовує TypeScript і надихався архітектурними патернами Angular. Nest.js дозволяє легко розділяти застосунок на модулі, що сприяє кращій структурованості та масштабованості коду. Він використовує декоратори для оголошення маршрутів, служб, інжекцій залежностей тощо. Nest.js побудований на TypeScript, що забезпечує статичну типізацію та поліпшену розробку. В ньому можна легко інтегрувати WebSockets, gRPC, GraphQL і багато іншого. Nest.js має вбудовану підтримку інжекції залежностей, що спрощує тестування та повторне використання коду. Він легко інтегрується з різними базами даних використовуючи ORM такі, як TypeORM, Sequelize. Nest.js підходить для створення масштабованих та продуктивних серверних застосунків будь-якої складності, від невеликих мікросервісів до великих монолітних застосунків [18].

Для взаємодією з базою даних використовується TypeORM, але для того щоб коректно взаємодіяти з нею потрібно спочатку створити класи об’єктів, які TypeORM конвертує в таблиці бази даних [19]. У розроблювальному ресурсі буде три класи:

- User – цей клас відповідає для створення, редагування та отримання даних користувача;
- Room – цей клас відповідає за створення, редагування та отримання даних кімнати;
- UserMovieRating – цей клас відповідає за збереження та отримання вподобань користувачів.

Також потрібно створити класи для валідації об’єктів, які будуть використовуватись для створення нових записів в базі даних. Для валідації створення та редагування користувача було реалізовано клас CreateUserDto, який описує обмеження для таких полів як name, email, password. Клас CreateRoomDto

використовується для валідації створення та редагування кімнат. CreateRoomDto описує поля name, users, admin.

Для зручності серверна частина застосунку була розділена на логічні модулі, кожен з яких відповідає за свою частину застосунку. Кожен поділений на кілька шарів абстракції: контролери, сервіси, сутності. Розглянемо детальніше основні модулі.

Першим є модуль авторизації AuthModule. Вхідною точкою в цей модуль є контролер AuthController. В ньому описані 4 запити, які цей модуль може опрацювати. За замовчуванням усі запити є приватними і не будуть опрацьовуватись якщо, користувач, який їх надіслав не буде авторизований, але запити для авторизації, реєстрації та оновлення токенів є публічними. Можливість створювати публічні та приватні запити була реалізована за допомогою класу AuthGuard, який перевіряє токени користувачів та декоратору Public, який дає змогу пропустити цю перевірку для обраних запитах.

Реалізація класу AuthGuard та декоратору Public наведена нижче.

```
export const IS_PUBLIC_KEY = 'isPublic';
export const Public = () => SetMetadata(IS_PUBLIC_KEY, true);


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

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

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

    private extractTokenFromHeader(request: Request): string | undefined {
        const [type, token] = request.headers.authorization?.split(' ') ?? [];
        return type === 'Bearer' ? token : undefined;
    }
}
```

Наступним є модуль користувачів UsersModule. Вхідною точкою в цей модуль є контролер UsersController. В ньому описані 2 запити, які цей модуль може опрацювати:

- отримання даних користувача за його ідентифікатором;
- пошук користувачів за іменем.

Усі запити цього модуля є приватними.

Модуль RoomsModule є модулем, який відповідає за взаємодію з кімнатами. Він використовує UsersModule для додавання користувачів до

кімнати та встановлення адміністратора кімнати. Вхідною точкою в RoomsModule є контролер RoomsController. В ньому описані 6 запитів, які цей модуль може опрацювати:

- створення кімнати;
- видалення кімнати;
- додавання до кімнати користувача;
- видалення користувача з кімнати;
- отримання списку кімнат користувача;
- отримання інформації про кімнату за її ідентифікатором.

Усі запити цього модуля є приватними.

Наступним є модуль фільмів MoviesModule. MoviesModule використовує відкрите API “The Movie DB”, оскільки актуалізація даних про фільми є ресурсоємним процесом. The Movie DB (TMDb) API — це сервіс, який надає доступ до великої бази даних інформації про фільми, серіали та акторів. Він дозволяє отримувати інформацію про популярні, нові, рейтингові фільми та серіали, а також здійснювати пошук за різними параметрами. Щоб використовувати TMDb API, потрібно зареєструватися та отримати API ключ. Вхідною точкою в MoviesModule є контролер MoviesController. В ньому описані 4 запити, які цей модуль може опрацювати:

- отримання списку найпопулярніших фільмів за сьогодні;
- отримання списку найпопулярніших фільмів за тиждень;
- отримання деталей фільму за його ідентифікатором;
- отримання списку жанрів фільмів.

Усі запити цього модуля є приватними.

MatchModule відповідає за пошук фільмів для спільного перегляду користувачів з однієї кімнати. MatchModule для спілкування з клієнтами використовує з’єднання socket замість HTTP запитів, як це було в попередніх модулях. Вхідною точкою в MatchModule є клас MatchGateway. MatchGateway в свою чергу має лише 1 метод, який опрацьовує з’єднання з клієнтом. Після з’єднання клієнти можуть відправляти події likeMovie та dislikeMovie для оцінки

фільмів, які MatchGateway відправляє їм використовуючи подію nextMovie. Якщо усі користувачі кімнати вподобають запропонований фільм, то MatchGateway відправить подію matchedMovie з цим фільмом.

Angular 17 − це одна з останніх версій популярного фреймворку для розробки WEB-застосунків від Google. Angular забезпечує потужний набір інструментів для створення динамічних і продуктивних односторінкових застосунків (SPA). Основні особливості Angular 17 включають [20]:

- оптимізацію рендерингу та збірки, що дозволяє застосункам працювати швидше та споживати менше ресурсів;
- підтримку нових версій TypeScript та інших інструментів, це забезпечує покращену продуктивність розробки та кращу підтримку сучасних можливостей мови;
- покращену підтримка модульності;
- включення підтримки таких технологій, як Web Components і Progressive Web Apps (PWA), що дозволяє створювати більш інтерактивні та мобільні WEB-застосунки;
- більш гнучку та потужну система маршрутизації, що дозволяє легше управляти складними маршрутами та станами застосунку;
- вбудовані інструменти та практики для забезпечення безпеки застосунків, включаючи покращену підтримку захисту від атак XSS і CSRF [21].

Для оптимізації швидкості завантаження клієнтського застосунку, він також був розділений на модулі, які «ліниво» завантажуються при потребі. Вхідною точкою в клієнтську частину застосунку є AppModule. AppModule містить в собі основу клієнтської частини застосунку та опис маршрутів, за якими будуть завантажуватись необхідні модулі. Як і серверна частина застосунку, клієнтська використовує функції, які обмежують доступ до маршрутів тих, чи інших користувачів. Функція guestGuard застосовується для маршрутів авторизації та реєстрації. Вона перенаправляє авторизованих користувачів, які намагаються відкрити ці сторінки на головну сторінку. Реалізація функції guestGuard наведена нижче.

```
export const guestGuard = (route: ActivatedRouteSnapshot): boolean | UrlTree
=> {

    const authService = inject(AuthService);
    const router = inject(Router);
    const accessToken = authService.accessToken;
    if (!accessToken) {
      return true;
    }
    const currentLang: string = route.params['lang'];
    const homePath: string = `${currentLang}/home`;
    return router.createUrlTree([homePath]);
  }
```

В свою чергу функція authGuard перенаправляє не авторизованих користувачів, які намагаються відкрити сторінки, що вимагають від користувача бути авторизованим, на сторінку авторизації. Реалізація функції authGuard наведена нижче.

```
export const authGuard = (route: ActivatedRouteSnapshot): boolean | UrlTree
=> {
    const authService = inject(AuthService);
    const router = inject(Router);
    const accessToken = authService.accessToken;
    if (accessToken) {
      return true;
    }
    const currentLang: string = route.params['lang'];
    const loginPath: string = `${currentLang}/login`;
    return router.createUrlTree([loginPath]);
  }
```

Функція authGuard застосовується до маршрутів кімнат та головної сторінки.

Користувач вважається авторизованим, якщо в сервісі авторизації зберігається access токен, який клієнтська частина застосунку отримує як відповідь на успішний запит авторизації. В подальшому цей токен також використовується для авторизованих запитів на сервер. Коли термін дії access токена закінчується, клієнт використовує refresh токен, який він також отримав, як відповідь на успішний запит авторизації, для отримання нового access токена без необхідності повторного введення облікових даних користувача. Це підвищує зручність для користувачів і забезпечує безперервний доступ до ресурсів. Refresh токени мають триваліший термін дії, ніж access токени. Це дозволяє мінімізувати ризик компрометації облікових даних, оскільки access токени мають коротший термін дії і частіше оновлюються. Для автоматичного оновлення токенів користувача було реалізовано перехоплювач запитів, який відловлює помилки «401 Unauthorized», оновлює токени та після оновлення знову відправляє запит з новим access токеном. Реалізація перехоплювача наведена нижче.

```
@Injectable()
export class TokenInterceptor implements HttpInterceptor {
  constructor(
    private authService: AuthService,
  ) { }
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<any>
{
    return next.handle(request).pipe(
      catchError((error) => {
        if (error instanceof HttpErrorResponse && error.status === 401 &&
!request.url.includes('/auth/login')) {
          return this.handle401Error(request, next);
        } else {
```

```
                return throwError(() => error);
            }
        })
    );
}

private addToken(request: HttpRequest<any>) {
    const token = this.authService.accessToken;
    return request.clone({
        setHeaders: {
            Authorization: `Bearer ${token}`,
        },
    });
}

private handle401Error(request: HttpRequest<any>, next: HttpHandler) {
    return this.authService.updateTokens().pipe(
        switchMap(() => {
            return next.handle(this.addToken(request));
        })
    );

}
}
```

Для взаємодії з сервером клієнтська частина використовує SocketService та ApiService.

SocketService використовує бібліотеку socket.io. Сервіс реалізовує два методи:

- connectToRoom – створення socket з’єднання з кімнатою на сервері;

- emitEvent – відправка події використовуючи раніше створене socket з’єднання з кімнатою. Реалізація SocketService наведена нижче.

```
@Injectable()
export class SocketService {
  private socket: Socket;
  private _events = new Subject<ISocketEvent<any>>();
  public events = this._events.asObservable();
  constructor(
    private localStorageService: LocalStorageService,
  ) { }

  public connectToRoom(roomId: string) {
    this.socket = io(environment.apiUrl, {
      transports: ['websocket'],
      query: { token: this.localStorageService.getAccessToken(), roomId }
    });
    this.socket.on(ESocketEvents.connect, () => {
      console.log('connected');
    });
    this.socket.on(ESocketEvents.matchedMovie, (filmId: number) => {
      this._events.next({
        event: ESocketEvents.matchedMovie,
        data: filmId,
      })
    });
    this.socket.on(ESocketEvents.nextMovie, (filmId: number) => {
      this._events.next({
        event: ESocketEvents.nextMovie,
        data: filmId,
      })
```

```
        });
    }

    public emitEvent(event: ESocketEvents, data: any) {
        this.socket.emit(event, data);
    }
}
```

ApiService відправляє HTTP запити на сервер. Ці запити формуються методом apiCall, який використовуючи список об’єктів IApiMapItem, де описані усі необхідні деталі запиту: адреса, метод, заголовки, тип відповіді. Реалізація інтерфейсу об’єкту IApiMapItem наведена нижче.

```
export interface IApiMapItem {
    url: string;
    method: 'GET' | 'POST' | 'DELETE' | 'PUT';
    mockup?: any;
    headers?: {
        [name: string]: boolean;
    };
    responseType?: 'arraybuffer' | 'blob' | 'json' | 'text';
}
```

3.6 Тестування роботи програми та аналіз результатів

Тестування програмного продукту є невід’ємною частиною процесу розробки для забезпечення якості, надійності, безпеки та продуктивності програмного забезпечення. Це допомагає зменшити витрати на виправлення помилок, підвищує задоволеність користувачів і гарантує відповідність вимогам і специфікаціям.

Було проведено детальне тестування розробленого WEB-ресурсу та мобільного додатку (понад 100 тестових запусків). Після запуску з’являється

сторінка для авторизації користувача. На рисунку 3.2 зображена сторінка для авторизації користувача на WEB-ресурсі, а на рисунку 3.3 вигляд на мобільному пристрої.

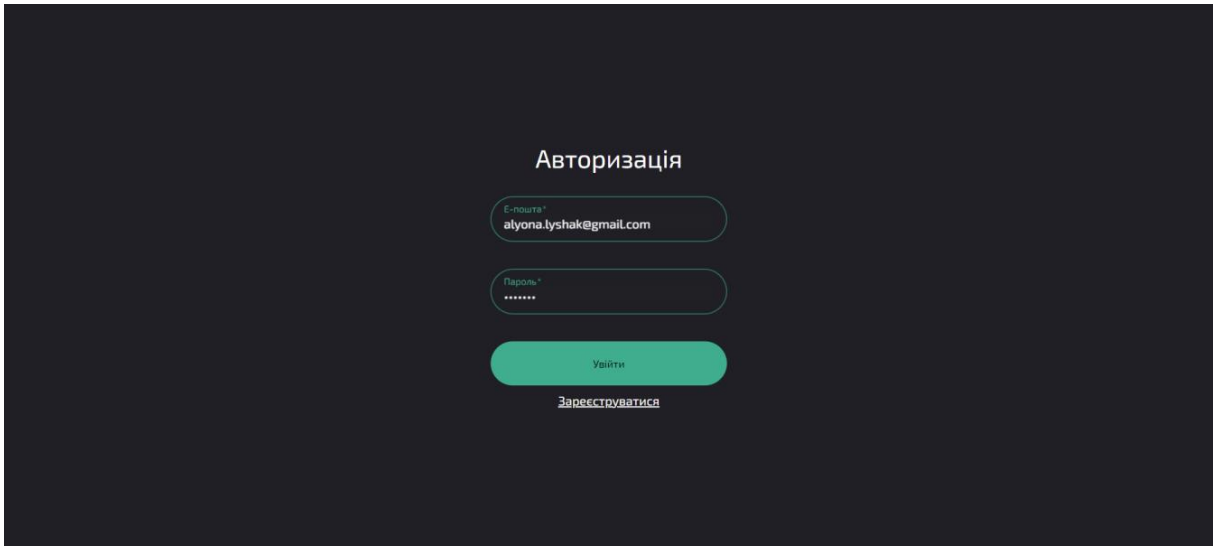

Рисунок 3.2 – Сторінка авторизації користувача на WEB-ресурсі

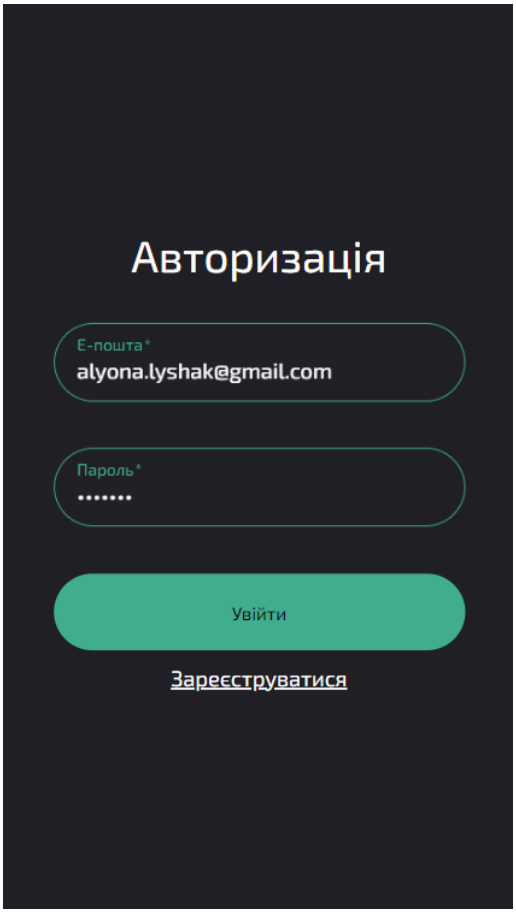

Рисунок 3.3 – Сторінка авторизації користувача на мобільному пристрої

Якщо натиснути кнопку «Зареєструватися», то користувача буде перенаправлено на сторінку реєстрації. Вигляд на WEB-ресурсі та мобільному пристрої зображено на рисунку 3.4 та 3.5 відповідно.

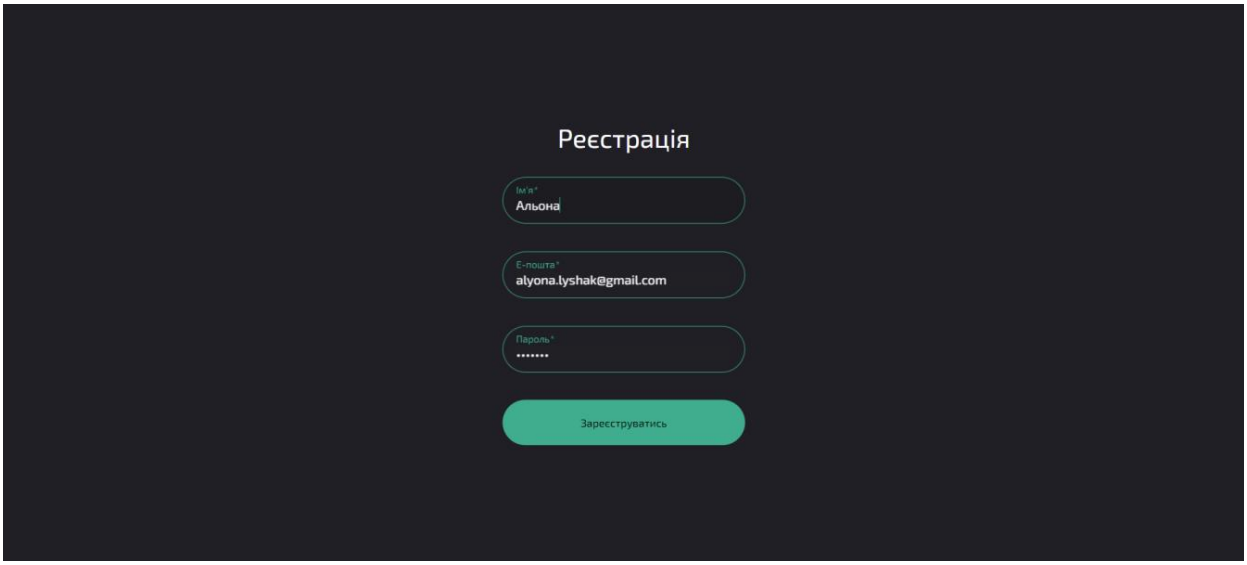

Рисунок 3.4 – Сторінка реєстрації користувача на WEB-ресурсі

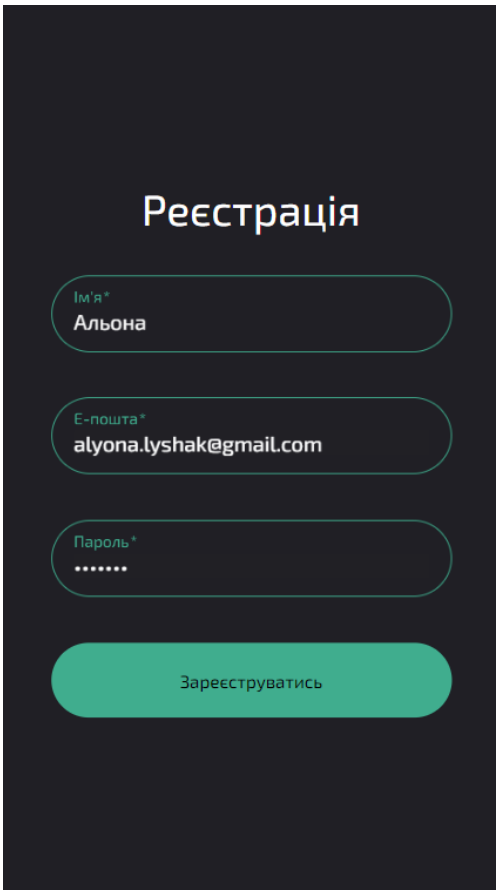

Рисунок 3.5 − Сторінка реєстрації користувача на мобільному пристрої

Після успішного входу або реєстрації, користувача буде перенаправлено на головну сторінку, де міститься інформація про сьогоднішні світові тренди по фільмах та світові тренди за весь тиждень (рис. 3.6, 3.7).

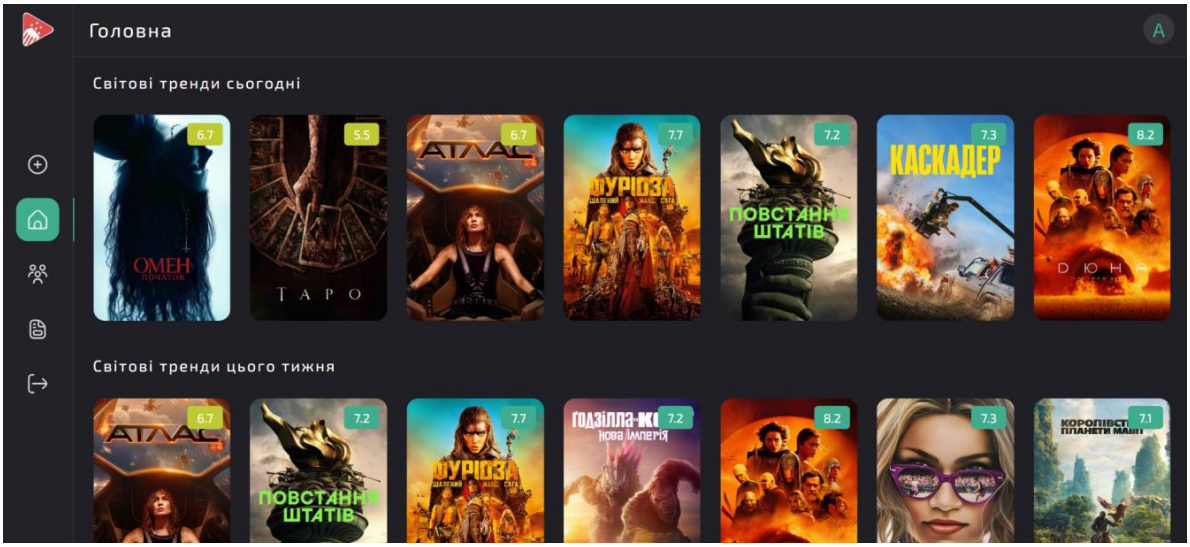

Рисунок 3.6 – Головна сторінка на WEB-ресурсі

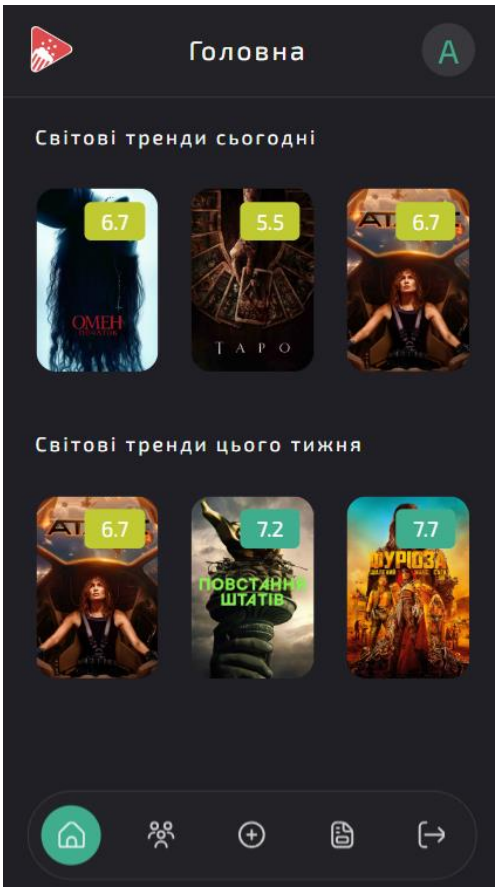

Рисунок 3.7 – Головна сторінка на мобільному пристрої

Користувач може створити нову кімнату для підбору фільмів. Для цього йому потрібно ввести ім’я кімнати та додати користувачів. Вигляд цих сторінок на WEB-ресурсі зображено на рисунку 3.8 та 3.9.

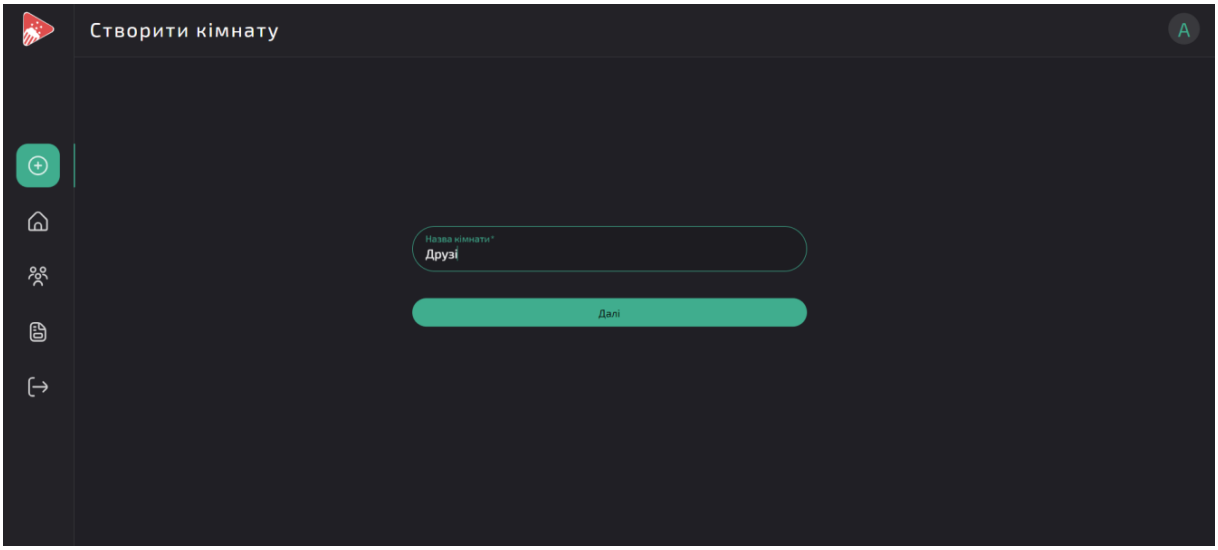

Рисунок 3.8 – Сторінка для введення назви кімнати на WEB-ресурсі

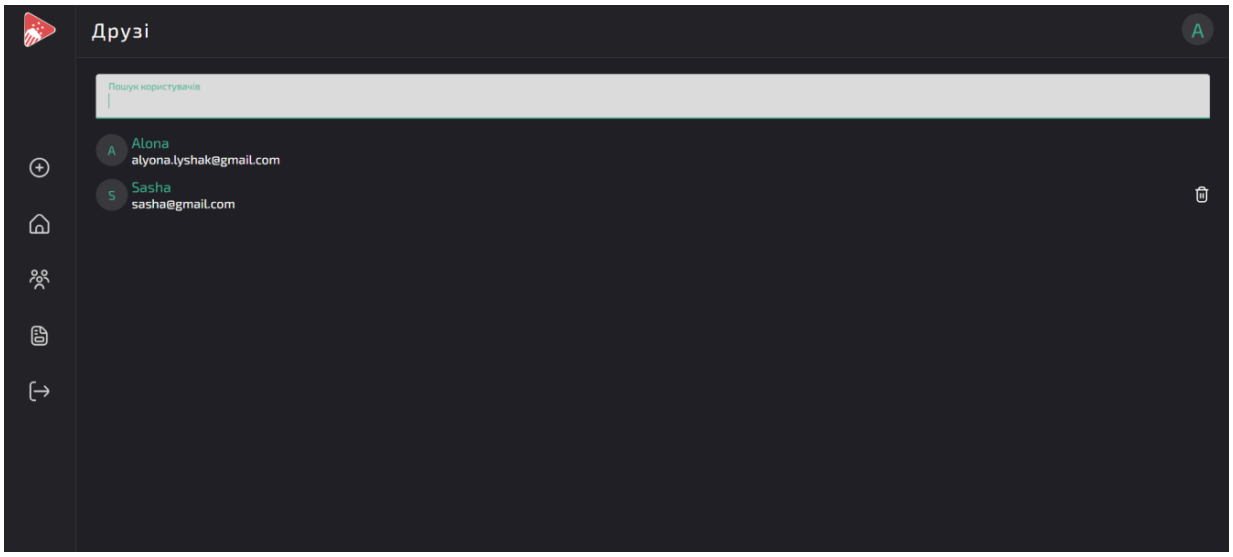

Рисунок 3.9 – Сторінка для додавання користувачів кімнати на WEB-ресурсі

Також розглянемо вигляд таких сторінок на мобільному пристрої (рис. 3.10, 3.11).

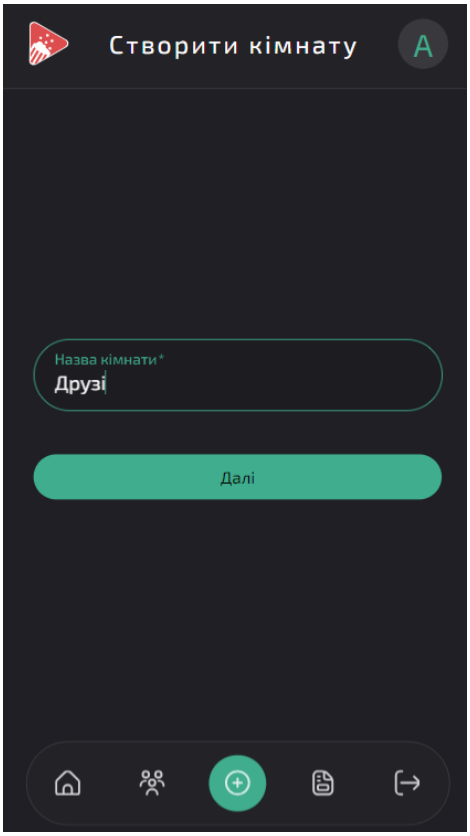

Рисунок 3.10 – Сторінка для введення назви кімнати на мобільному пристрої

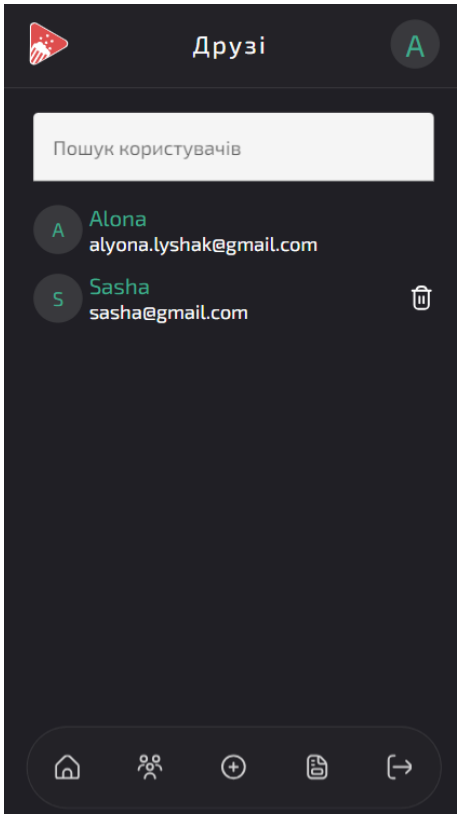

Рисунок 3.11 – Сторінка для додавання користувачів кімнати на мобільному пристрої

Користувач може перейти на сторінку з усіма кімнатами в яких він перебуває та відредагувати, видалити кімнату або відкрити її щоб розпочати підбір фільму. Сторінка з кімнатами зображена на рисунку 3.12 та 3.13.

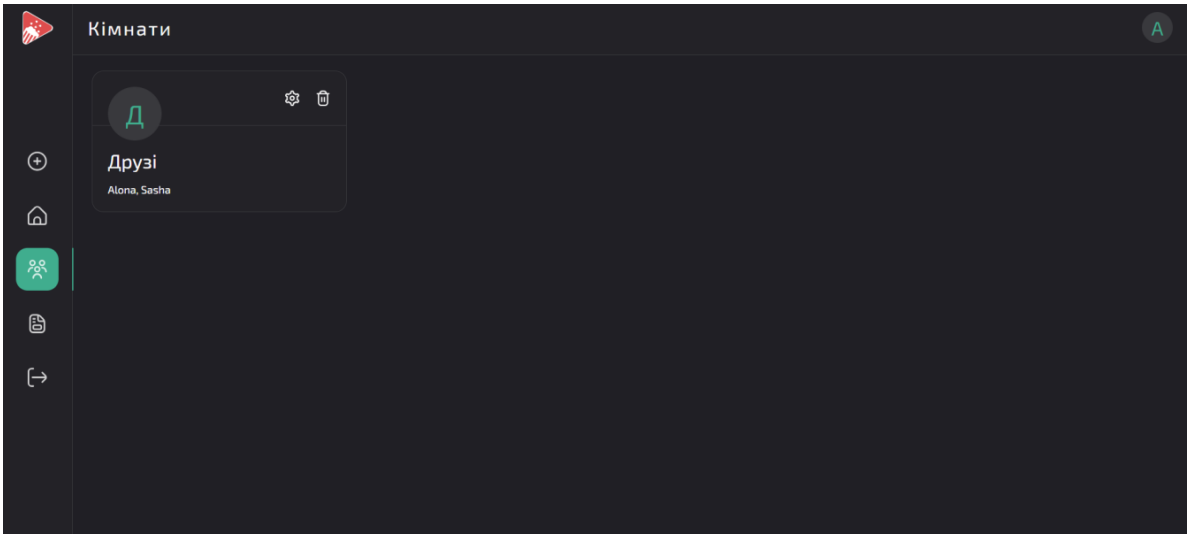

Рисунок 3.12 – Сторінка з кімнатами користувача на WEB-ресурсі

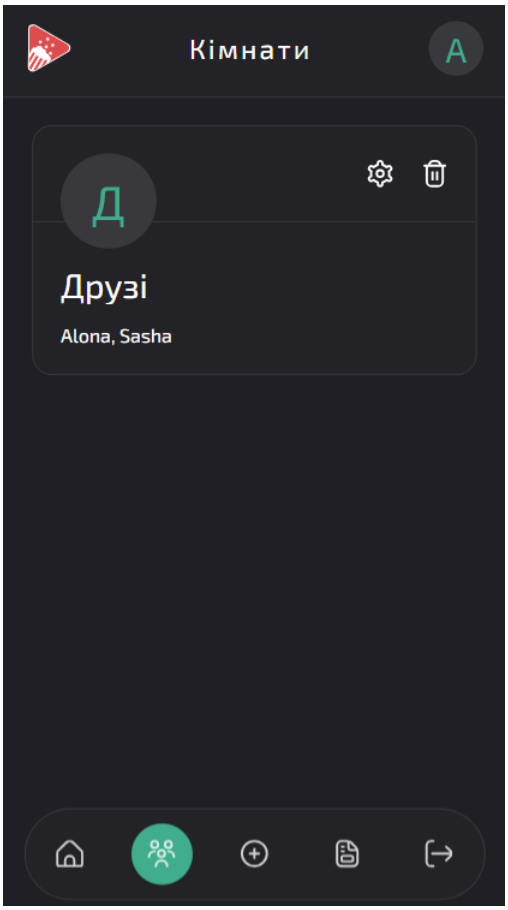

Рисунок 3.13 – Сторінка з кімнатами користувача на мобільному пристрої

Після відкриття кімнати користувачу починають пропонуватись фільми, а він в свою чергу обирає чи подобається йому фільм, чи ні. Теж саме роблять інші користувачі обраної кімнати. На рисунку 3.14 зображено сторінку для підбору фільмів на WEB-ресурсі, а на рисунку 3.15 вигляд на мобільному пристрої.

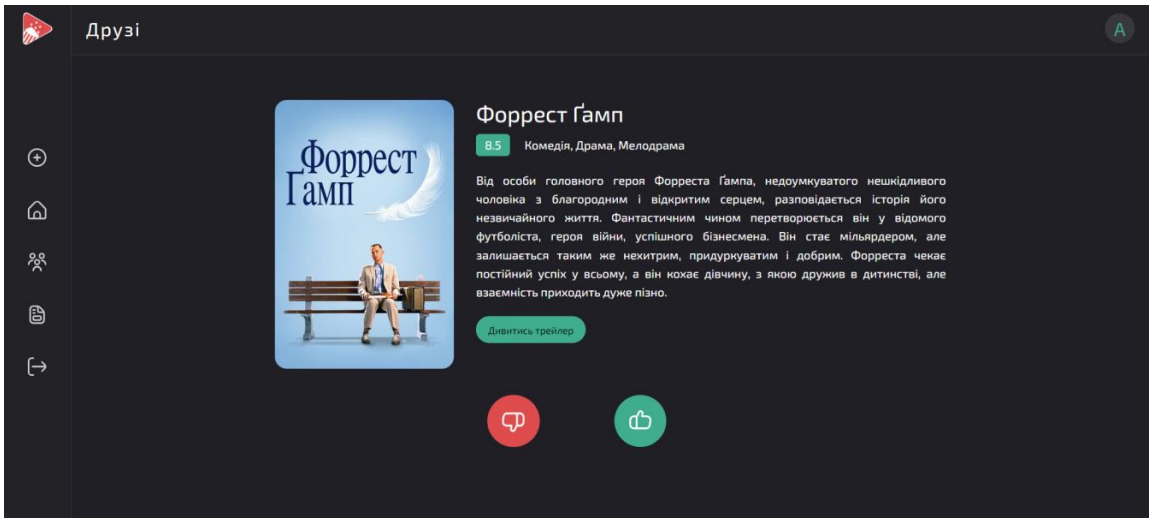

Рисунок 3.14 – Сторінка для підбору фільмів на WEB-ресурсі

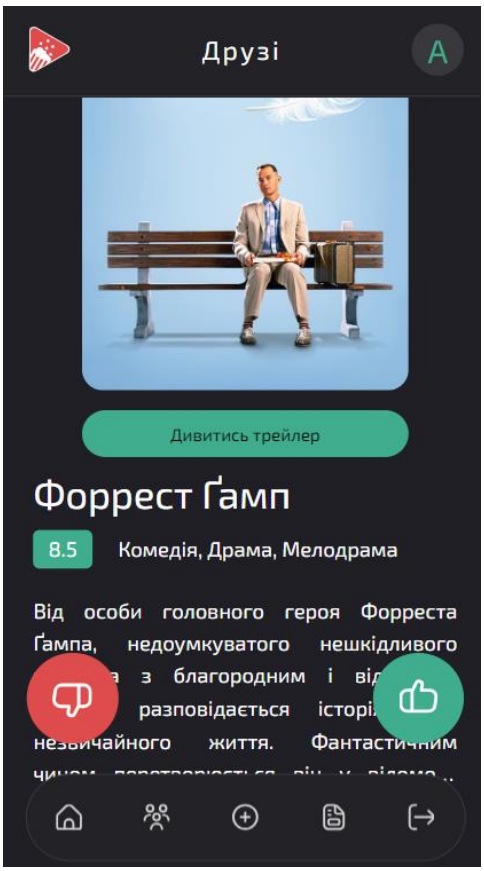

Рисунок 3.15 – Сторінка для підбору фільмів на мобільному пристрої

Якщо ж всі користувачі оберуть однаковий фільм, то з’явиться вікно з повідомленням про те, що фільм для перегляду обрано (рис. 3.16, 3.17).

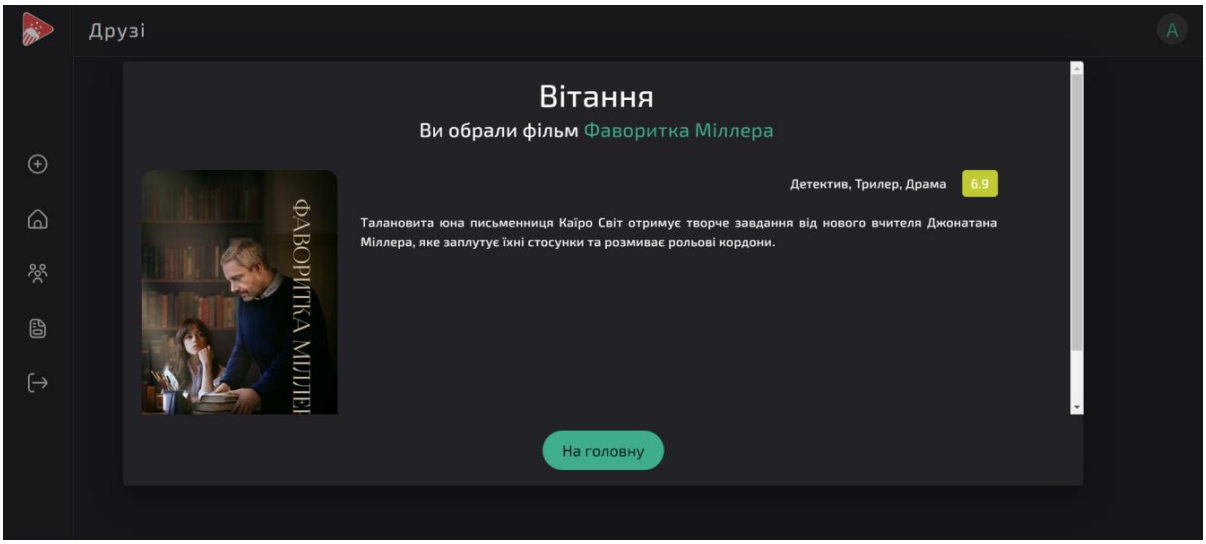

Рисунок 3.16 – Сторінка з повідомленням про обраний для перегляду фільм на WEB-ресурсі

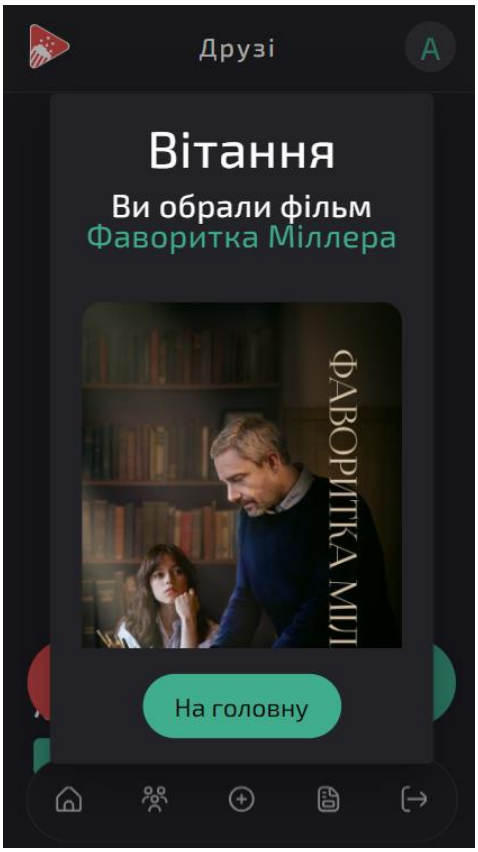

Рисунок 3.17 – Сторінка з повідомленням про обраний для перегляду фільм на мобільному пристрої

Розроблений програмний модуль підбору фільмів успішно пройшов тестування та відповідає всім заданим вимогам.

В таблиці 3.6 подано результати тестування розробленого програмного продукту та аналогів.

В результаті розробки було реалізовано функціонал, який був відсутній в системах-аналогах, а саме:

- українська локалізація;
- зручний пошук;
- колективний підбір фільмів.

Таблиця 3.6 – Результати тестування розробленого програмного продукту та аналогів

	Розроблений модуль	Popcorn	Letterboxd	IMDb
Українська локалізація	+	–	–	–
Зручний пошук	+	+	–	–
Персоналізований підбір фільмів	+	+	+	+
Перегляд детальної інформації про фільми	+	+	+	+
Колективний підбір фільмів	+	–	–	–

Отже, проаналізувавши та протестувавши розроблений програмний продукт, можна дійти висновку, що поставлених цілей було досягнуто.

З таблиці 3.6 можна побачити, що розроблений програмний модуль підбору фільмів має покращений функціонал у порівнянні з системами-аналогами щонайменше на 3 можливості. Розширений функціонал забезпечить вище задоволення користувачів від використання цього програмного продукту.

3.7 Висновок

В цьому розділі було проаналізовано мови програмування для розробки клієнтської частини – TypeScript, CoffeeScript, та серверної частини – TypeScript, PHP, C++. Розглянуто їх переваги та недоліки, визначено особливості та доцільність використання. В результаті такого аналізу було обрано мову програмування TypeScript, тому що TypeScript має статичну типізацію, підтримує об’єктно орієнтоване програмування, популярний, виявляє помилки на етапі компіляції та має інтеграцію із сучасними IDE та базами даних.

Крім того, було проаналізовано мови програмування для управління організованою базою даних – SQL та OQL. В результаті аналізу обрано SQL, оскільки саме SQL призначений для роботи з реляційною базою даних.

Як середовище розробки обрано Visual Studio Code завдяки значним перевагам над іншими середовищами, а саме Visual Studio Code має відкритий вихідний код, простий, швидкий та зручний, має інтелектуальне автозавершення, відлагоджувач вбудований в термінал, підтримка Git та багато розширень.

Розроблено ER-модель бази даних, яка спростить проектування бази даних та розуміння взаємозв’язків між сутностями бази даних та реалізовано програмний продукт.

Було проведено тестування програмного модуля підбору фільмів та здійснено його порівняння з аналогами. Як результат, зроблено висновок, що основні функції, які потрібно було реалізувати – реалізовано. Розроблений програмний продукт має покращений функціонал порівняно з аналогами щонайменше на 3 можливості, а саме: додано українську локалізацію, покращено пошук фільмів, додано функціонал для колективного підбору фільмів.

ВИСНОВКИ

Всі задачі, поставлені для реалізації програмного модуля підбору фільмів були виконані в повному обсязі, а саме: обґрунтовано доцільність розробки програмного модуля підбору фільмів; спроектовано програмний модуль підбору фільмів; програмно реалізовано модуль підбору фільмів; виконано тестування програми та проведено аналіз отриманих результатів; розроблено інструкцію користувача.

Для розробки була використана мова програмування TypeScript, для управління базою даних – SQL, а платформами для реалізації обрано WEB-браузери та мобільні пристрої.

Для реалізації була розроблена структура програмного модуля підбору фільмів, створено UML-діаграму класів для клієнтської та серверної частин, створено ER-модель бази даних, а також було розроблено схему алгоритму роботи.

Розроблений програмний продукт успішно пройшов тестування та в порівнянні з системами-аналогами надає користувачам щонайменше 3 додаткові можливості, що суттєво підвищує задоволення користувачів від його використання.

Мета роботи була досягнута шляхом розширення функціональних можливостей програмного модуля підбору фільмів. Порівняно з аналогами, розроблений продукт містить українську локалізацію, зручний пошук та колективний підбір фільму.

Отже, розроблений програмний модуль підбору фільмів відповідає заданим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Лишак А. М., Крилик Л. В. Перспективи розробки програмного модуля підбору фільмів. *LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації*. URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20577 (дата звернення: 06.05.2024).

2. Popcorn. URL: https://play.google.com/store/apps/details?id=com.pocketdeals.popcorn&hl=uk&gl=US (дата звернення: 07.05.2024).

3. Letterboxd. URL: https://letterboxd.com/ (дата звернення: 07.05.2024).

4. IMDb. URL: https://www.imdb.com/ (дата звернення: 07.05.2024).

5. What Are Microservices? URL: https://medium.com/@uzairhussain98/what-are-microservices-38fa6aaa3305 (дата звернення: 07.05.2024).

6. Що таке об’єктно-орієнтоване програмування: принципи, переваги та недоліки. URL: https://dan-it.com.ua/uk/blog/chto-takoe-obektno-orientirovannoe-programmirovanie-principy-preimushhestva-i-nedostatki/ (дата звернення: 10.05.2024).

7. Наслідування. URL: http://megalib.com.ua/content/3241_Naslidyvannya.html (дата звернення: 10.05.2024).

8. Поліморфізм в ООП. URL: http://ruslan.rv.ua/python-essential/oop/polymorphism/polymorphism_basis.html (дата звернення: 10.05.2024).

9. Вивчення основних принципів ООП у програмуванні. URL: https://itproger.com/ua/news/izuchenie-osnovnih-printsipov-oop-v-programmirovanii (дата звернення: 10.05.2024).

10. Савчук Т. О., Ваховська Л. М. Основи інформатики та обчислювальної техніки. *Навчальні посібники Вінницького національного технічного*

університету. Мережні електронні видання. URL: https://web.posibnyky.vntu.edu.ua/fitki/1savchuk_osnovy_inform_obchisl_tehn/Topic5.html (дата звернення: 07.05.2024).

11. Learn TypeScript in 5 minutes. URL: https://medium.com/free-code-camp/learn-typescript-in-5-minutes-13eda868daeb (дата звернення: 07.05.2024).

12. Brief Introduction to CoffeeScript. URL: http://tutorials.jumpstartlab.com/topics/javascript/coffeescript.html (дата звернення: 07.05.2024).

13. What is Angular Material? URL: https://distantjob.com/blog/what-is-angular-material/ (дата звернення: 07.05.2024).

14. Інтегроване середовище розробки (IDE). URL: https://w3schoolsua.github.io/hyperskill/ide.html#gsc.tab=0 (дата звернення: 27.05.2024).

15. Visual Studio Code. URL: https://code.visualstudio.com/docs (дата звернення: 27.05.2024).

16. WebStorm. URL: https://www.jetbrains.com/help/webstorm/meet-webstorm.html (дата звернення: 27.05.2024).

17. Atom. URL: https://atom-editor.cc/ (дата звернення: 27.05.2024).

18. Nest.js: A Progressive Node.js Framework / Джей Белл, Грег Маголан, Девід Гіхарро, Адріан де Перетті. Bleeding Edge Press, 2018. 408 с.

19. TypeORM − Amazing ORM for TypeScript and JavaScript. URL: https://typeorm.io/ (дата звернення: 27.05.2024).

20. What is Angular? URL: https://angular.dev/overview (дата звернення: 27.05.2024).

21. Angular XSS Guide: Examples and Prevention. URL: https://www.stackhawk.com/blog/angular-xss-guide-examples-and-prevention/ (дата звернення: 27.05.2024).

ДОДАТКИ

ДОДАТОК А (обов'язковий)

Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень

Назва роботи: Програмний модуль підбору фільмів

Тип роботи: бакалаврська дипломна робота
(БДР, МКР)

Підрозділ кафедра комп'ютерних наук, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 95,5% Схожість 4,5%

Аналіз звіту подібності (відмітити потрібне):

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

Особа, відповідальна за перевірку Озеранський В. С.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи Лишак А. М.

Керівник роботи Крилик Л. В.

ДОДАТОК Б (обов'язковий)
Лістинг програми

```
export class AuthController {
  constructor(private authService: AuthService) {}

  @Public()
  @HttpCode(HttpStatus.OK)
  @Post('login')
  signIn(@Body() signInDto: Record<string, any>) {
    return this.authService.signIn(signInDto.email, signInDto.password);
  }

  @Public()
  @HttpCode(HttpStatus.OK)
  @Post('register')
  signUp(@Body() createUserDto: CreateUserDto) {
    return this.authService.signUp(createUserDto);
  }

  @Get('profile')
  getProfile(@Request() req) {
    return this.authService.getProfile(req.user.id);
  }

  @Public()
  @Post('refresh')
  refrestTokens(@Body() req: Record<string, string>) {
    return this.authService.refresh(req.token);
  }
```

```
}

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

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

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

```
  private extractTokenFromHeader(request: Request): string | undefined {
    const [type, token] = request.headers.authorization?.split(' ') ?? [];
    return type === 'Bearer' ? token : undefined;
  }
}

export class AuthService {
  constructor(
    private usersService: UsersService,
    private jwtService: JwtService,
    private configService: ConfigService,
  ) {}

  async signIn(email: string, pass: string): Promise<any> {
    const user = await this.usersService.findOneByEmail(email, true);

    if (!pass || !email || !user?.password) {
      throw new UnauthorizedException();
    }
    const isMatch = await bcrypt.compare(pass, user?.password);
    if (!isMatch) {
      throw new UnauthorizedException();
    }
    const payload = { id: user.id, email: user.email };

    return {
      access_token: await this.jwtService.signAsync(payload),
      refresh_token: await this.jwtService.signAsync(payload, {
        expiresIn: this.configService.get<string>(
```

```
            'JWT_REFRESH_EXPIRATION',
          ),
        }),
    };
}

async signUp(createUserDto: CreateUserDto): Promise<any> {
    const user = await this.usersService.createUser({
      ...createUserDto,
      password: await this.hashPassword(createUserDto.password),
    });
    const payload = { id: user.id, email: user.email };

    return {
      access_token: await this.jwtService.signAsync(payload),
      refresh_token: await this.jwtService.signAsync(payload, {
        expiresIn: this.configService.get<string>(
          'JWT_REFRESH_EXPIRATION',
        ),
      }),
    };
}

async refresh(refreshToken: string): Promise<any> {
    const data = this.jwtService.decode(refreshToken);
    const { iat, exp, ...payload } = data;

    return {
      access_token: await this.jwtService.signAsync(payload),
      refresh_token: await this.jwtService.signAsync(payload, {
```

```
            expiresIn: this.configService.get<string>(
              'JWT_REFRESH_EXPIRATION',
            ),
          }),
        };
    }

    async getProfile(id: string): Promise<User> {
        return this.usersService.getById(id);
    }

    async hashPassword(password: string): Promise<string> {
        const saltOrRounds = 10;
        return await bcrypt.hash(password, saltOrRounds);
    }
}

export class ActiveRoom {
    public readonly roomInfo: Room;

    private _users: Set<string> = new Set();

    public get users(): Array<string> {
        return [...this._users];
    }

    private _movies: Record<string, Record<string, number>> = {};

    private currentPage = 0;
```

```
constructor(
    private room: Room,
    private socket: Socket,
    private loadMoviesFn: (page: number) => Observable<number[]>,
) {
    this.roomInfo = this.room;
}

public addUser(id: string): void {
    this._users.add(id);
}

public deleteUser(id: string): void {
    this._users.delete(id);
}

public likeMovie(movieId: string, userId: string): void {
    if (!this._users.has(userId)) {
        return;
    }
    if (!this._movies[movieId]) {
        this._movies[movieId] = { userId: 1 };
    } else {
        this._movies[movieId][userId] = 1;
    }

    const likesCount = Object.values(this._movies[movieId]).reduce(
        (a, b) => a + b,
        0,
    );
```

```
    if (likesCount === this._users.size && this._users.size !== 1) {
      this.socket
        .to(this.room.id)
        .emit(ESocketEvents.matchedMovie, movieId);
      this.socket.emit(ESocketEvents.matchedMovie, movieId);
    }
  }

  public dislikeMovie(movieId: string, userId: string): void {
    if (!this._users.has(userId)) {
      return;
    }
    if (!this._movies[movieId]) {
      this._movies[movieId] = { userId: 0 };
    } else {
      this._movies[movieId][userId] = 0;
    }
  }

  public async getNextMovie(userId: string): Promise<string> {
    const nextMovie = Object.keys(this._movies).find(
      (movieId: string) =>
        !Number.isInteger(this._movies[movieId][userId]),
    );

    if (!nextMovie) {
      this.currentPage++;
      await this.loadMovies();
```

```
            return this.getNextMovie(userId);
        }
        return nextMovie;
    }

    private addMovies(movies: Array<string>): void {
        movies.forEach((movie: string) => {
            if (!this._movies[movie]) {
                this._movies[movie] = {};
            }
        });
    }

    private async loadMovies(): Promise<string[]> {
        this.currentPage++;
        const newMovies = (
            await firstValueFrom(this.loadMoviesFn(this.currentPage))
        ).map((m: number) => m.toString());

        this.addMovies(newMovies);

        return newMovies;
    }
}

export class MatchService {
    private activeRooms: Record<string, ActiveRoom> = {};

    constructor(
        private jwtService: JwtService,
```

```
    private roomsService: RoomsService,
    private userService: UsersService,
    private moviesService: MoviesService,
) {}

public async handleConnection(socket: Socket) {
    try {
        const token = socket.handshake?.query?.token as string;
        const roomId = socket.handshake?.query?.roomId as string;

        if (!token && !roomId) {
            socket.disconnect();
            return;
        }

        const user: User = this.jwtService.verify(token);
        const room: Room = await this.roomsService.getRoomById(
            user?.id,
            roomId,
        );

        if (!room) {
            socket.disconnect();
            return;
        }

        socket.join(roomId);

        let activeRoom = this.activeRooms[roomId];
```

```
if (!activeRoom) {
    activeRoom = await this.createActiveRoom(room, socket);
    this.activeRooms[roomId] = activeRoom;
}

activeRoom.addUser(user.id);

activeRoom.getNextMovie(user.id).then((nextMovie: string) => {
    socket.emit(ESocketEvents.nextMovie, +nextMovie);
});

socket.on(ESocketEvents.disconnect, () => {
    activeRoom.deleteUser(user.id);
    if (activeRoom.users.length === 0) {
        delete this.activeRooms[roomId];
    }
});

socket.on(ESocketEvents.likeMovie, (movieId: number) => {
    this.userService.addRating(user.id, movieId, 10);
    activeRoom.likeMovie(movieId.toString(), user.id);

    activeRoom.getNextMovie(user.id).then((nextMovie: string) => {
        socket.emit(ESocketEvents.nextMovie, +nextMovie);
    });
});

socket.on(ESocketEvents.dislikeMovie, (movieId: number) => {
    this.userService.addRating(user.id, movieId, 0);
    activeRoom.dislikeMovie(movieId.toString(), user.id);
```

```
            activeRoom.getNextMovie(user.id).then((nextMovie: string) => {
                socket.emit(ESocketEvents.nextMovie, +nextMovie);
            });
        });
      } catch {
        socket.disconnect();
      }
    }

    private async createActiveRoom(
      room: Room,
      socket: Socket,
    ): Promise<ActiveRoom> {
      const ratedMovies = await this.userService.getUserRatedMoviesIds(
        room.users.map((u: User) => u.id),
      );

      return new ActiveRoom(room, socket, (page: number) => {
        return this.moviesService.getRecommendationsForGroup(
          ratedMovies,
          page,
        );
      });
    }
}

export class MoviesController {
    constructor(private moviesService: MoviesService) {}
```

```
  @Get('/trending/day')
  getTrendingDay(@Query('language') language?: string) {
    return this.moviesService.getTrendingDay(language);
  }

  @Get('/trending/week')
  getTrendingWeek(@Query('language') language?: string) {
    return this.moviesService.getTrendingWeek(language);
  }

  @Get('/find')
  findMovies(@Query('language') language?: string) {
    return this.moviesService.findMovies(language);
  }

  @Get('/details/:filmId')
  getMovieById(@Param() params, @Query('language') language?: string) {
    return this.moviesService.getMovieById(params.filmId, language);
  }

  @Get('/genres')
  getGenresList(@Query('language') language?: string) {
    return this.moviesService.getGenresList(language);
  }
}

export class MoviesService {
  constructor(private themoviedbService: TheMovieDBService) {}

  public getTrendingDay(lang: string = DEFAULT_LANG) {
```

```
    return this.themoviedbService.getTheTrendingMovies(
        ETimeWindow.day,
        lang,
    );
}

public getTrendingWeek(lang: string = DEFAULT_LANG) {
    return this.themoviedbService.getTheTrendingMovies(
        ETimeWindow.week,
        lang,
    );
}

public findMovies(lang: string = DEFAULT_LANG) {
    return this.themoviedbService.findMovies(lang);
}

public getMovieById(
    id: number | string,
    lang: string = DEFAULT_LANG,
): Observable<IMovie> {
    return this.themoviedbService.getMovieById(id, lang);
}

public getRecommendationsForGroup(
    movieIds: string[],
    page: number,
): Observable<number[]> {
    let popularGenres$ = of([]);
    if (movieIds.length) {
```

```
        popularGenres$ = zip(movieIds.map((id: string) =>
this.getMovieById(id))).pipe(
            map((movies: IMovie[]) => {
                const genresCount: { [key: number]: number } = {};
                movies.forEach((movie) => {
                    movie.genres?.forEach((genre) => {
                        if (!genresCount[genre]) {
                            genresCount[genre] = 0;
                        }
                        genresCount[genre]++;
                    });
                });

                const popularGenres = Object.keys(genresCount).sort(
                    (a, b) => genresCount[+b] - genresCount[+a],
                );
                    return popularGenres;
            })
        );
    }

    return popularGenres$.pipe(
        switchMap((popularGenres: Array<string>) => {
            return this.themoviedbService.getRecommendations(
                popularGenres,
                [],
                page,
            );
        }),
        map((movies: IRespList<IMovie[]>) => {
```

```
            return movies.results.map((m) => m.id);
          }),
        );
    }

    public getGenresList(
        lang: string = DEFAULT_LANG,
    ): Observable<IGenreTMDB[]> {
        return this.themoviedbService.getGenresList(lang);
    }
}

export class TheMovieDBService {
    get currentDate(): string {
        return new Date().toISOString().split('T')[0];
    }

    constructor(private readonly httpService: HttpService) {}

    public getTheTrendingMovies(
        timeWindow: TimeWindow,
        language: string,
    ): Observable<IRespList<IMovie>> {
        return this.httpService
            .get(`${TMDB_URL}/trending/movie/${timeWindow}`, {
                headers: {
                    accept: 'application/json',
                    Authorization: TOKEN,
                },
                params: { language, 'release_date.gte': this.currentDate },
```

```
        })
        .pipe(
            map((resp) => resp.data),
            map((data) => {
                return {
                    ...data,
                    results: data.results
                        .filter((film: IMovieTMDB) => {
                            return new Date(film.release_date) < new Date();
                        })
                        .map((m) => this.mapMovies(m)),
                };
            }),
        );
}

public findMovies(language: string): Observable<IRespList<IMovie>> {
    return this.httpService
        .get(`${TMDB_URL}/discover/movie`, {
            headers: {
                accept: 'application/json',
                Authorization: TOKEN,
            },
            params: {
                language,
                'release_date.gte': this.currentDate,
                sort_by: 'popularity.desc',
                'vote_average.gte': 6,
            },
        })
```

```
        .pipe(
            map((resp) => resp.data),
            map((data) => {
                return {
                    ...data,
                    results: data.results
                        .filter((film: IMovieTMDB) => {
                            return new Date(film.release_date) < new Date();
                        })
                        .map((m) => this.mapMovies(m)),
                };
            }),
        );
}

public getMovieById(id: number | string, language: string) {
    return this.httpService
        .get(`${TMDB_URL}/movie/${id}`, {
            headers: {
                accept: 'application/json',
                Authorization: TOKEN,
            },
            params: {
                language,
            },
        })
        .pipe(
            map((resp) => resp.data),
            map((data) => {
                return this.mapMovies(data);
```

```
        }),
      );
}

public getRecommendations(
  genres: Array<string>,
  actors: Array<string>,
  page: number = 1,
): Observable<IRespList<IMovie[]>> {
  const genreString = genres.map((id: string) => +id).join('|');
  const actorString = actors.map((id: string) => +id).join('|');

  return this.httpService
    .get(`${TMDB_URL}/discover/movie`, {
      headers: {
        accept: 'application/json',
        Authorization: TOKEN,
      },
      params: {
        'release_date.gte': this.currentDate,
        sort_by: 'popularity.desc',
        'vote_average.gte': 6,
        with_genres: genreString,
        page,
        with_cast: actorString,
      },
    })
    .pipe(
      map((resp) => resp.data),
      map((data) => {
```

```
            return {
                ...data,
                results: data.results
                    .filter((film: IMovieTMDB) => {
                        return new Date(film.release_date) < new Date();
                    })
                    .map((m) => this.mapMovies(m)),
            };
        }),
    );
}

public getGenresList(language: string): Observable<IGenreTMDB[]> {
    return this.httpService
        .get(`${TMDB_URL}/genre/movie/list`, {
            headers: {
                accept: 'application/json',
                Authorization: TOKEN,
            },
            params: { language },
        })
        .pipe(
            map((resp) => resp.data),
            map((data) => data.genres),
        );
}

private mapMovies(sorce: IMovieTMDB): IMovie {
    return {
        id: sorce.id,
```

```
        title: sorce.title,
        posterPath:
`https://image.tmdb.org/t/p/w600_and_h900_bestv2${sorce.poster_path}`,
        voteAverage: sorce.vote_average,
        genres: sorce.genres?.map((g) => g.id),
        overview: sorce.overview,
      };
    }
}

@Controller('rooms')
export class RoomsController {
  constructor(private roomsService: RoomsService) {}

  @Post('create')
  createRoom(@Body() body, @Request() req) {
    return this.roomsService.createRoom(body.name, req.user.id);
  }

  @Post('delete')
  async deleteRoom(@Body() body, @Request() req): Promise<boolean> {
    const result = await this.roomsService.deleteRoom(
      body.roomId,
      req.user.id,
    );
    if (!result) {
      throw new BadRequestException();
    }
    return true;
  }
```

```
@Post('add-user')
async addUserToRoom(@Body() body, @Request() req): Promise<Room> {
   const result = await this.roomsService.addUserToRoom(body, req.user.id);

   if (!result) {
      throw new BadRequestException();
   }
   return result;
}

@Post('delete-user')
async deleteUserFromRoom(@Body() body, @Request() req): Promise<Room> {
   const result = await this.roomsService.deleteUserFromRoom(
      body,
      req.user.id,
   );
   if (!result) {
      throw new BadRequestException();
   }
   return result;
}

@Get('list')
getUserRooms(@Request() req) {
   return this.roomsService.getUserRooms(req.user.id);
}

@Get(':roomId')
async getRoomById(@Request() req, @Param() params): Promise<Room> {
```

```
        const room = await this.roomsService.getRoomById(
            req.user.id,
            params.roomId,
        );
        if (!room) {
            throw new BadRequestException();
        }
        return room;
    }
}

export class RoomsService {
    constructor(
        @InjectRepository(Room)
        private readonly roomRepository: Repository<Room>,

        @InjectRepository(User)
        private readonly userRepository: Repository<User>,
    ) {}

    async createRoom(name: string, adminId: string): Promise<Room> {
        const admin = await this.userRepository.findOne({
            where: { id: adminId },
        });

        const room: Room = new Room();
        room.name = name;
        room.users = [admin];
        room.admin = admin;
        return this.roomRepository.save(room);
```

```
}

async addUserToRoom(
    body: IRoomUserManagementReq,
    adminId: string,
): Promise<Room> {
    const room = await this.roomRepository
        .findOne({
            relations: { users: true, admin: true },
            where: { id: body.roomId },
            select: { admin: { id: true } },
        })
        .then((room) => {
            if (room.admin.id === adminId) {
                return room;
            }
            return;
        });

    const user = await this.userRepository.findOne({
        where: { id: body.userId },
    });

    if (user && room) {
        room.users = [...room.users, user];
    }
    return this.roomRepository.save(room);
}

async deleteUserFromRoom(
```

```
    body: IRoomUserManagementReq,
    adminId: string,
): Promise<Room> {
    const room = await this.roomRepository
        .findOne({
            relations: { users: true, admin: true },
            where: { id: body.roomId },
            select: { admin: { id: true } },
        })
        .then((room) => {
            if (room.admin.id === adminId) {
                return room;
            }
            return;
        });

    if (room) {
        if (body.userId === adminId) {
            return;
        }
        room.users = room.users.filter((user) => user.id !== body.userId);
    }
    return this.roomRepository.save(room);
}
async deleteRoom(
    roomId: string,
    userId: string,
): Promise<Room | DeleteResult> {
    const room: Room = await this.roomRepository.findOne({
        relations: { users: true, admin: true },
```

```
        select: { admin: { id: true } },
        where: { id: roomId },
    });
    if (!room) {
        return;
    }
    if (room.admin.id === userId && room.users.length === 1) {
        return this.roomRepository.delete(roomId);
    } else {
        room.users = room.users.filter((user) => user.id !== userId);
        if (room.admin.id === userId) {
            room.admin = room.users[0];
        }
        return this.roomRepository.save(room);
    }
}

async getUserRooms(userId: string): Promise<Room[]> {
    return this.roomRepository
        .find({
            relations: { users: true, admin: true },
            select: { admin: { id: true } },
        })
        .then((rooms) => {
            return rooms.filter((room) =>
                room.users.find((user) => user.id === userId),
            );
        });
}
```

```
    async getRoomById(userId: string, roomId: string): Promise<Room> {
        return this.roomRepository
            .findOne({
                relations: { users: true, admin: true },
                where: { id: roomId },
                select: { admin: { id: true } },
            })
            .then((room) => {
                if (room.users.find((user) => user.id === userId)) {
                    return room;
                }
                return;
            });
    }
}

export class UsersController {
    constructor(private usersService: UsersService) {}

    @Get(':id')
    getUserById(@Param() params: any) {
        return this.usersService.getById(params.id);
    }

    @Get()
    findUser(@Query('search') search?: string) {
        return this.usersService.findAllByUserName(search);
    }
}
```

```
export class UsersService {
  constructor(
    @InjectRepository(User)
    private readonly userRepository: Repository<User>,

    @InjectRepository(UserMovieRating)
    private readonly userMovieRatingRepository: Repository<UserMovieRating>,
  ) {}

  async createUser(createUserDto: CreateUserDto): Promise<User> {
    const user: User = new User();
    user.name = createUserDto.name;
    user.email = createUserDto.email;
    user.password = createUserDto.password;
    return this.userRepository.save(user);
  }

  async findOneByEmail(
    email: string,
    showPass?: boolean,
  ): Promise<User | undefined> {
    return this.userRepository.findOne({
      where: { email: email },
      select: {
        id: true,
        name: true,
        email: true,
        logo: true,
        password: showPass,
      },
```

```
    });
}

async getById(id: string): Promise<User | undefined> {
    const userId = id || null;
    return this.userRepository.findOne({
        where: { id: userId },
        select: { id: true, name: true, email: true, logo: true },
    });
}

async removeUser(id: number): Promise<{ affected?: number }> {
    return this.userRepository.delete(id);
}

async findAllByUserName(query: string): Promise<User[] | undefined> {
    return this.userRepository.find({
        where: [
            { email: ILike(`%${query}%`) },
            { name: ILike(`%${query}%`) },
        ],
        take: 10,
    });
}

async addRating(
    userId: string,
    movieId: number,
    mark: number,
): Promise<UserMovieRating> {
```

```
        const user = await this.getById(userId);
        const rating: UserMovieRating = new UserMovieRating();
        rating.user = user;
        rating.movie = movieId?.toString();
        rating.rating = mark;

        return this.userMovieRatingRepository.save(rating);
    }

    async getUserRatedMoviesIds(usersId: Array<string>): Promise<string[]> {
        return Promise.all(
            usersId.map(async (id: string) => {
                return this.userMovieRatingRepository.find({
                    where: { user: { id }, rating: MoreThanOrEqual(7) },
                });
            }),
        ).then((data) => {
            const moviesIds = []
                .concat(...data)
                .map((r: UserMovieRating) => r.movie);
            const uniqueIds = new Set(moviesIds);
            return Array.from(uniqueIds);
        });
    }
}
```

ДОДАТОК В (обов'язковий)

ГРАФІЧНА ЧАСТИНА

«ПРОГРАМНИЙ МОДУЛЬ ПІДБОРУ ФІЛЬМІВ»

Виконала: студентка 4 курсу, групи 2КН-20б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Лишак А.М.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Крилик Л. В.
(прізвище та ініціали)
«07» 06 2024 р.

Вінниця ВНТУ - 2024 рік

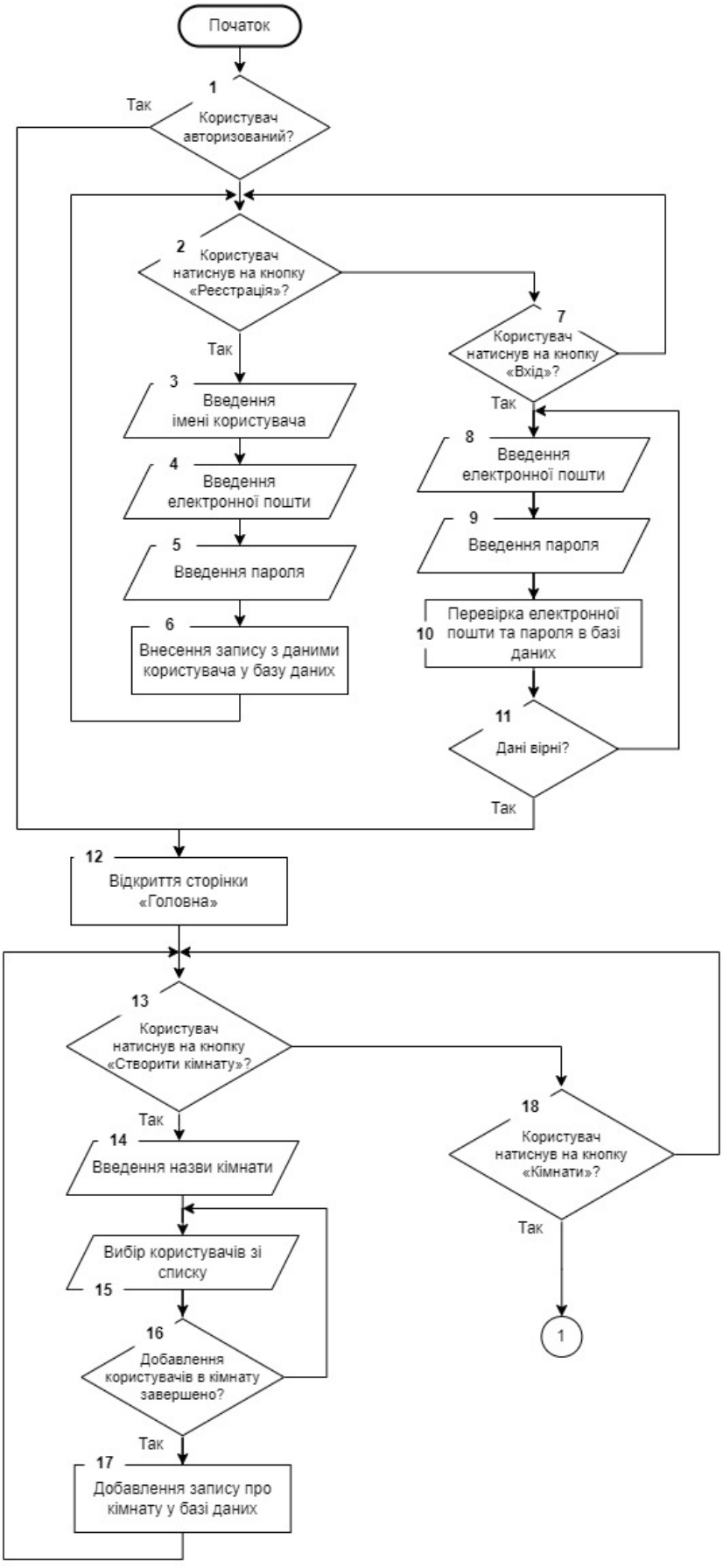

Рисунок В.1 – Схема алгоритму роботи програмного модуля підбору фільмів

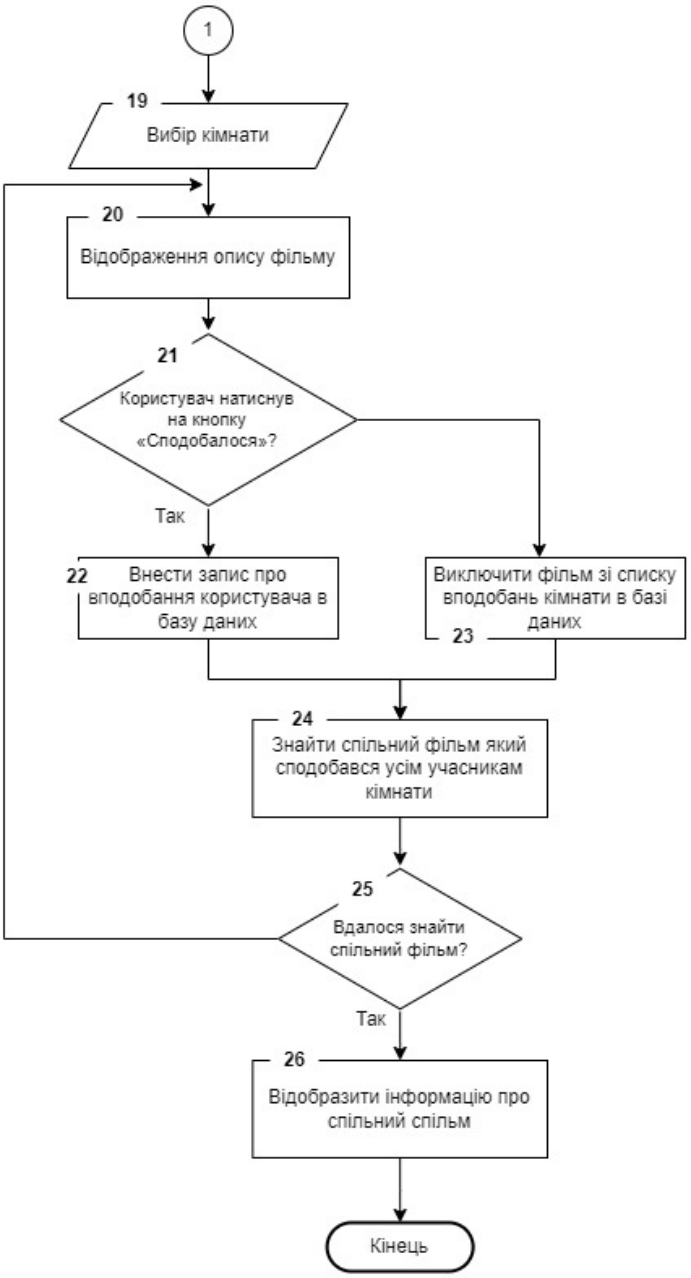

Рисунок В.1, аркуш 2

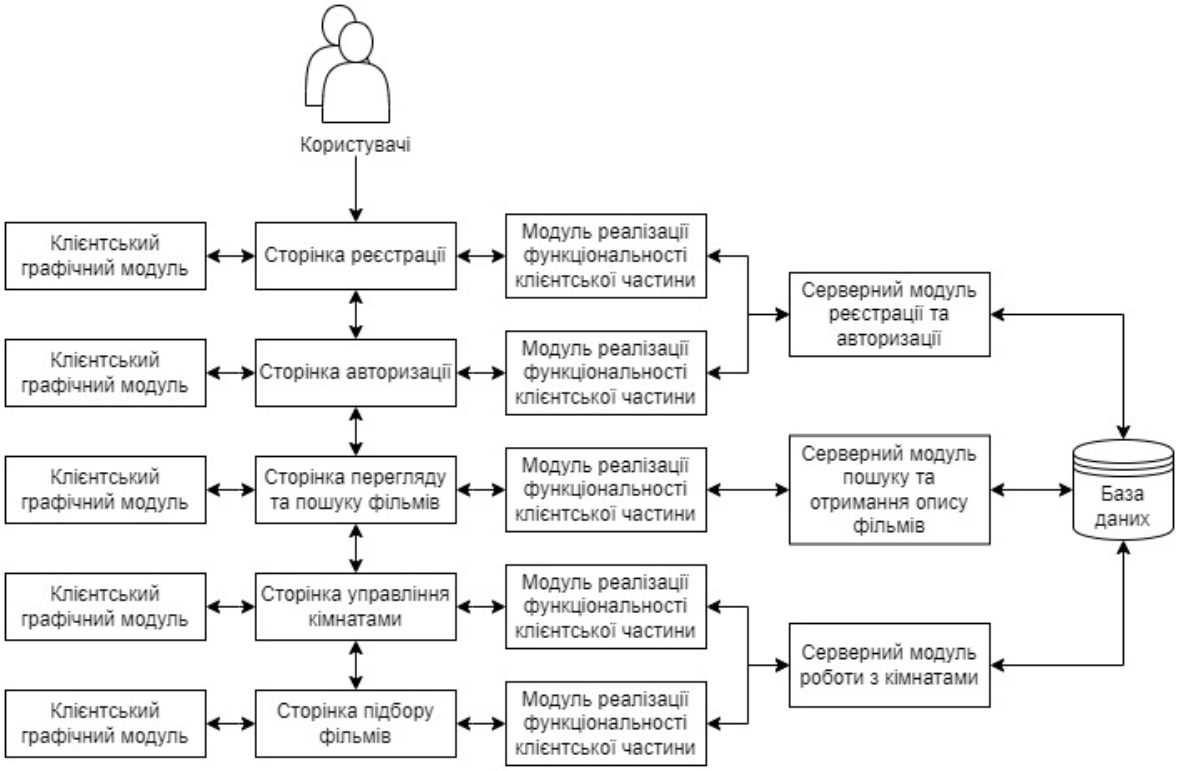

Рисунок В.2 − Загальна структурна схема функціонування системи

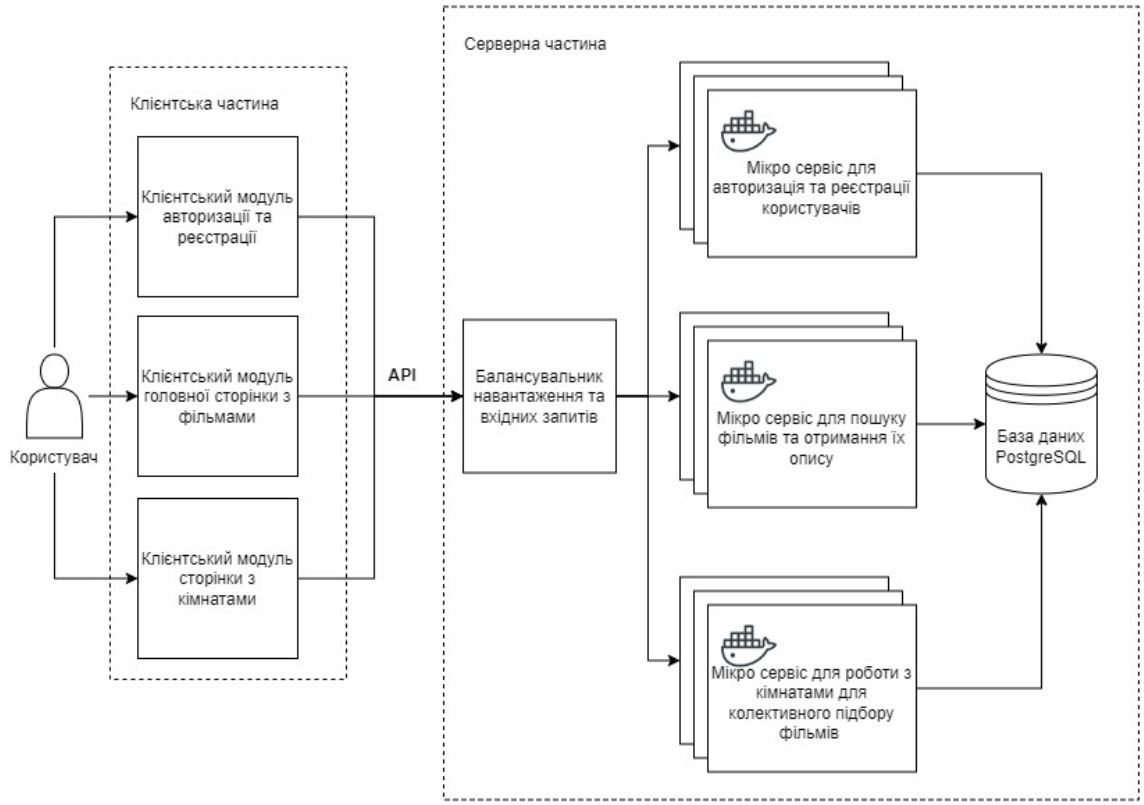

Рисунок В.3 – Загальна структурна схема компонентів програмного модуля підбору фільмів

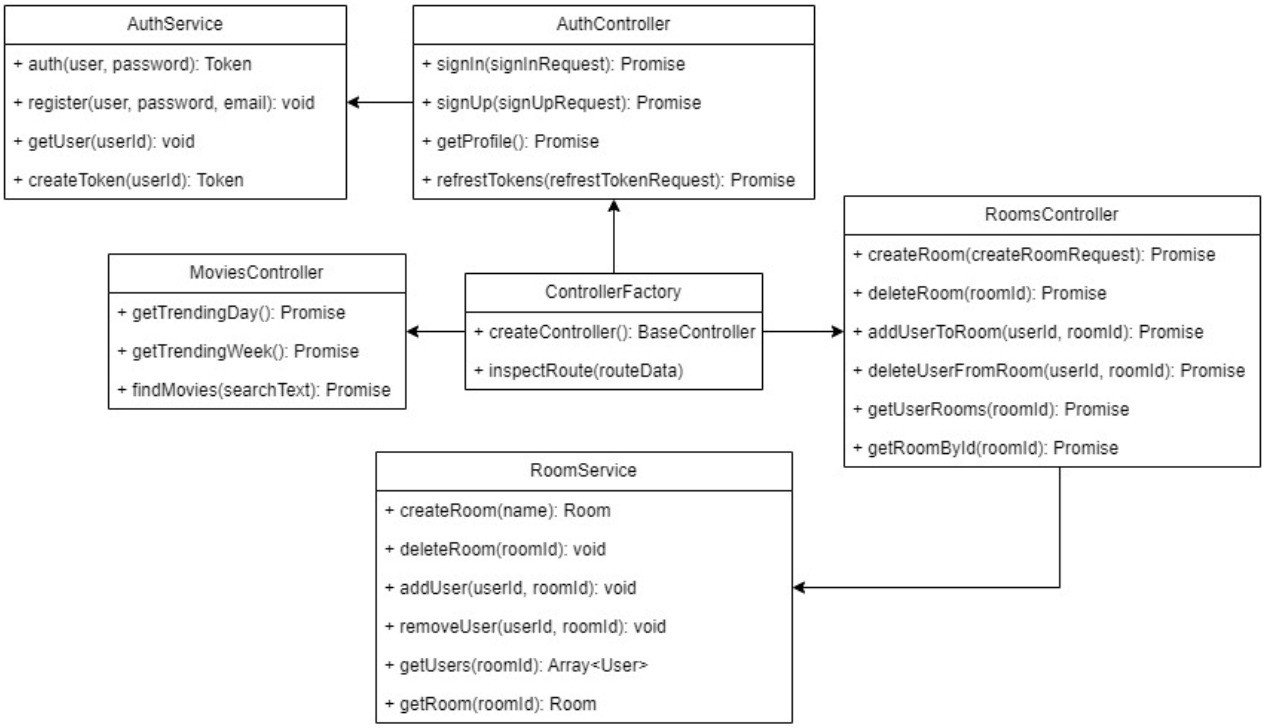

Рисунок В.4 – UML-діаграма класів серверної частини модуля підбору фільмів

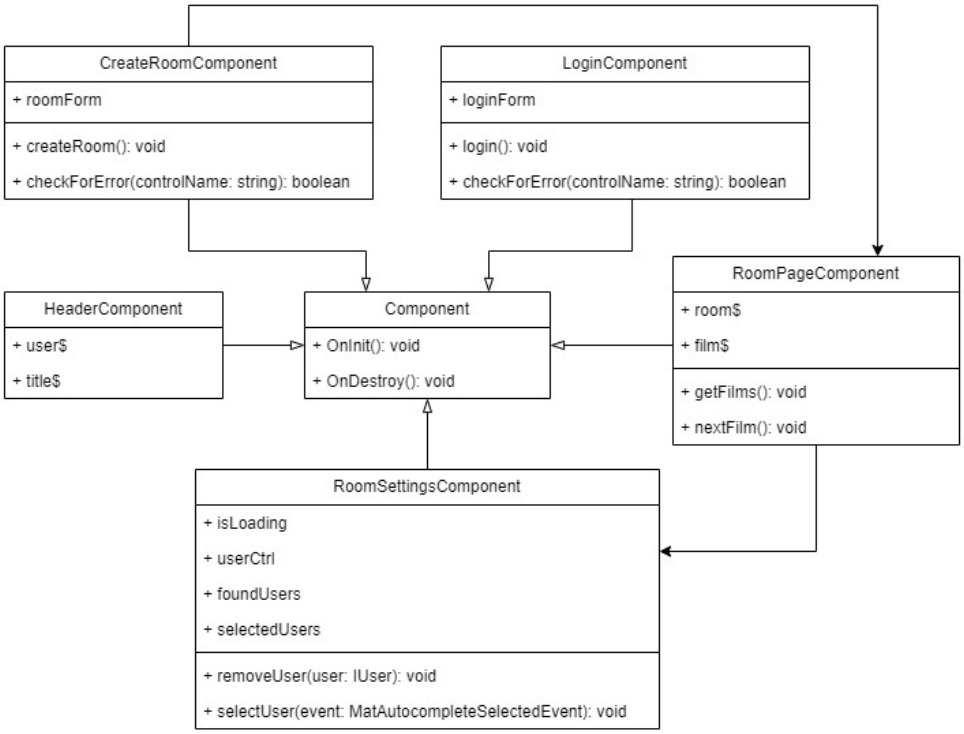

Рисунок В.5 – UML-діаграма класів клієнтської частини модуля підбору фільмів

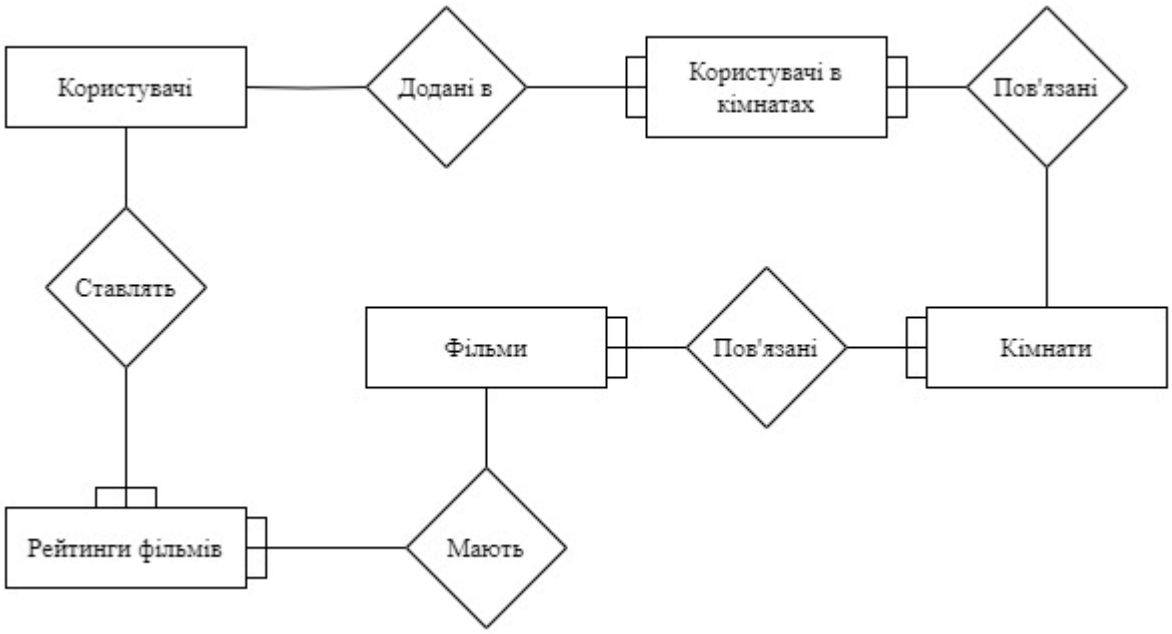

Рисунок В.6 –ER-модель для бази даних програмного модуля підбору фільмів

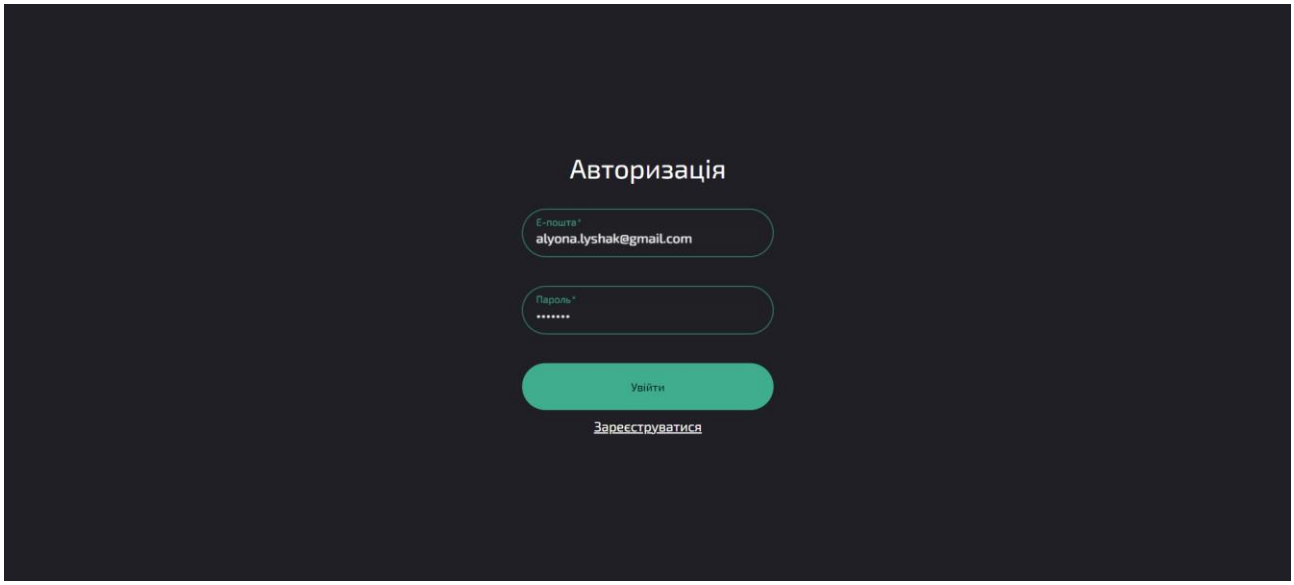

Рисунок В.7 – Сторінка авторизації користувача на WEB-ресурсі

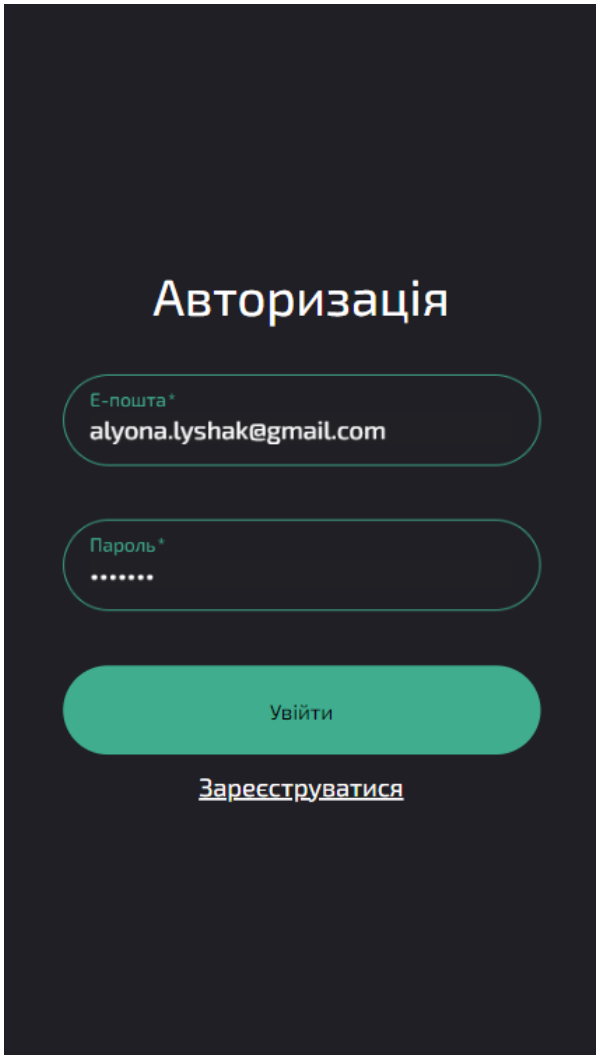

Рисунок В.8 − Сторінка авторизації користувача на мобільному пристрої

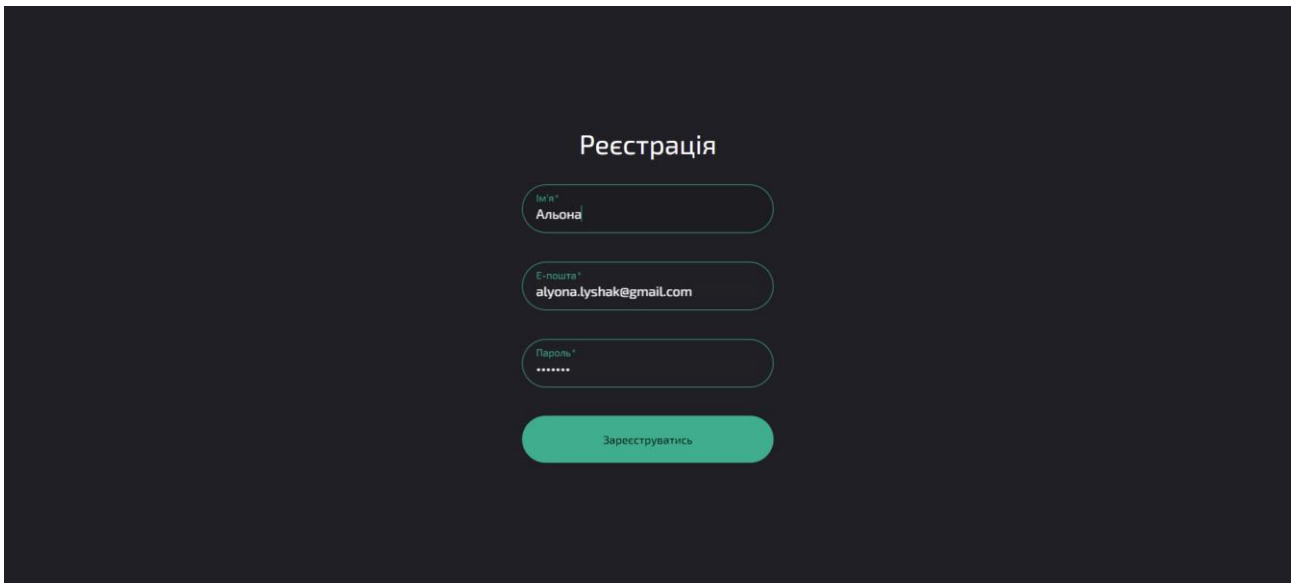

Рисунок В.9 – Сторінка реєстрації користувача на WEB-ресурсі

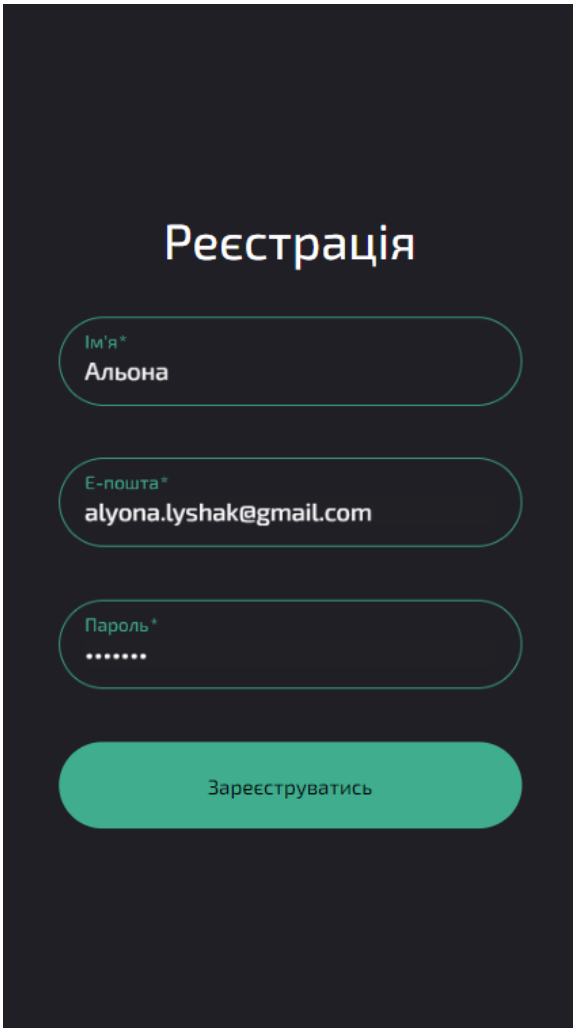

Рисунок В.10 –Сторінка реєстрації користувача на мобільному пристрої

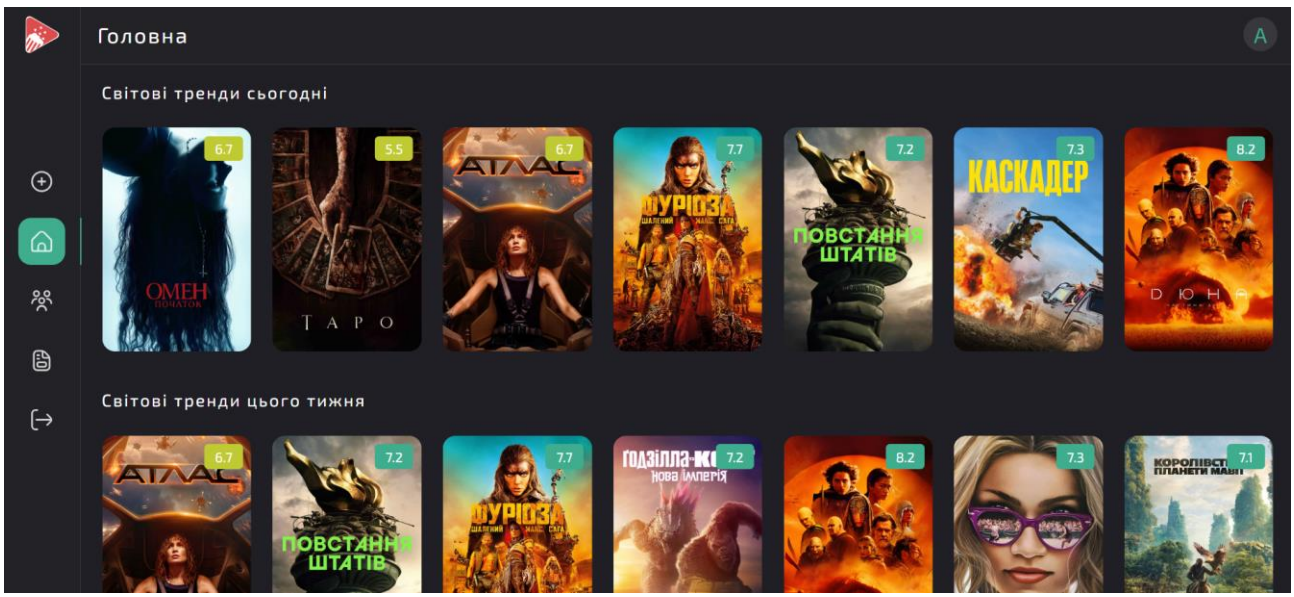

Рисунок В.11 – Головна сторінка на WEB-ресурсі

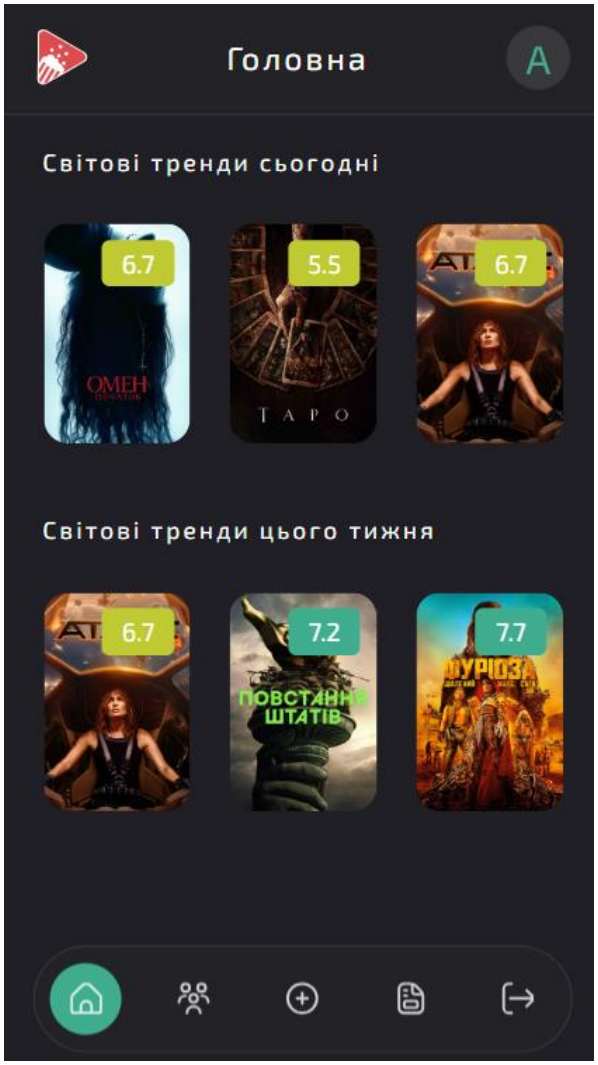

Рисунок В.12 – Головна сторінка на мобільному пристрої

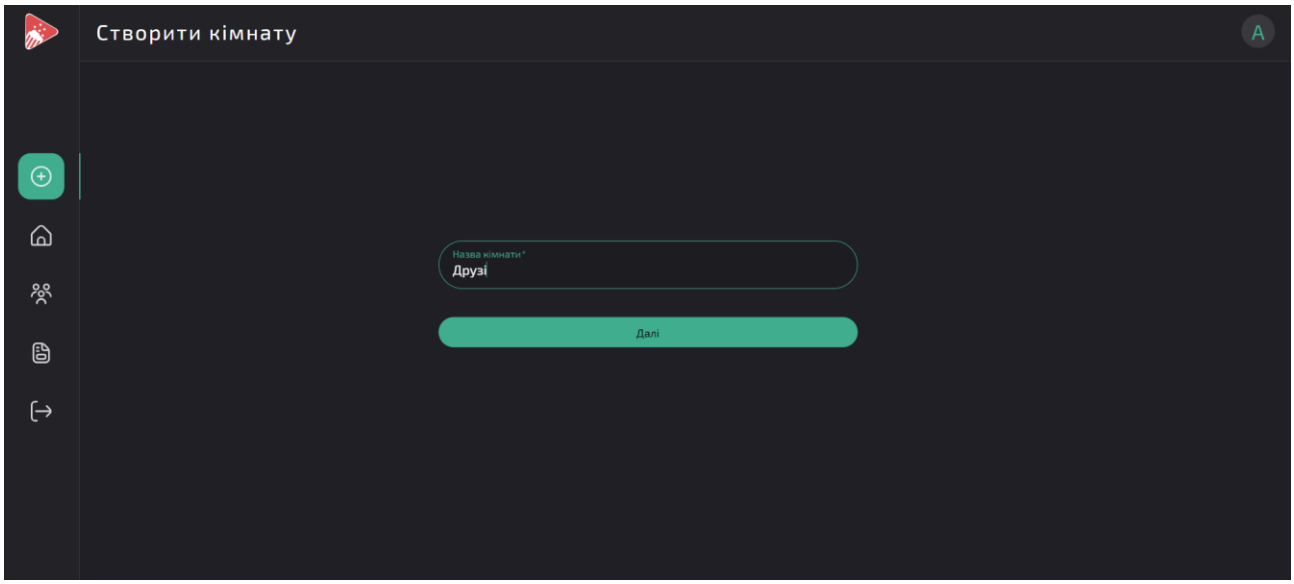

Рисунок В.13 – Сторінка для введення назви кімнати на WEB-ресурсі

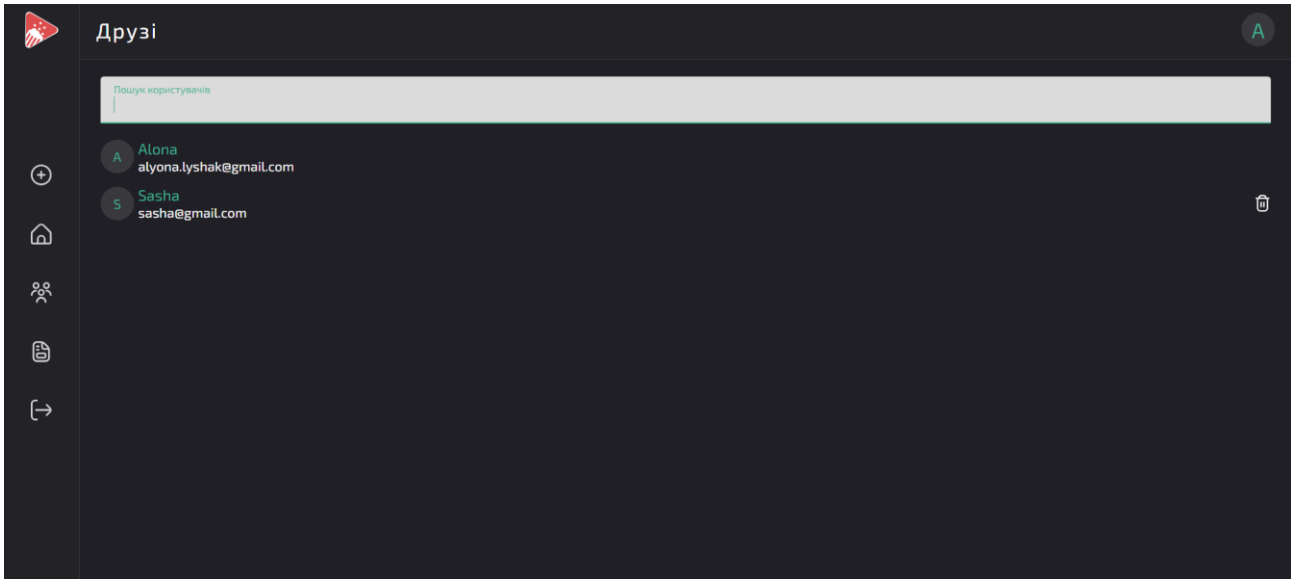

Рисунок В.14 – Сторінка для додавання користувачів кімнати на WEB-ресурсі

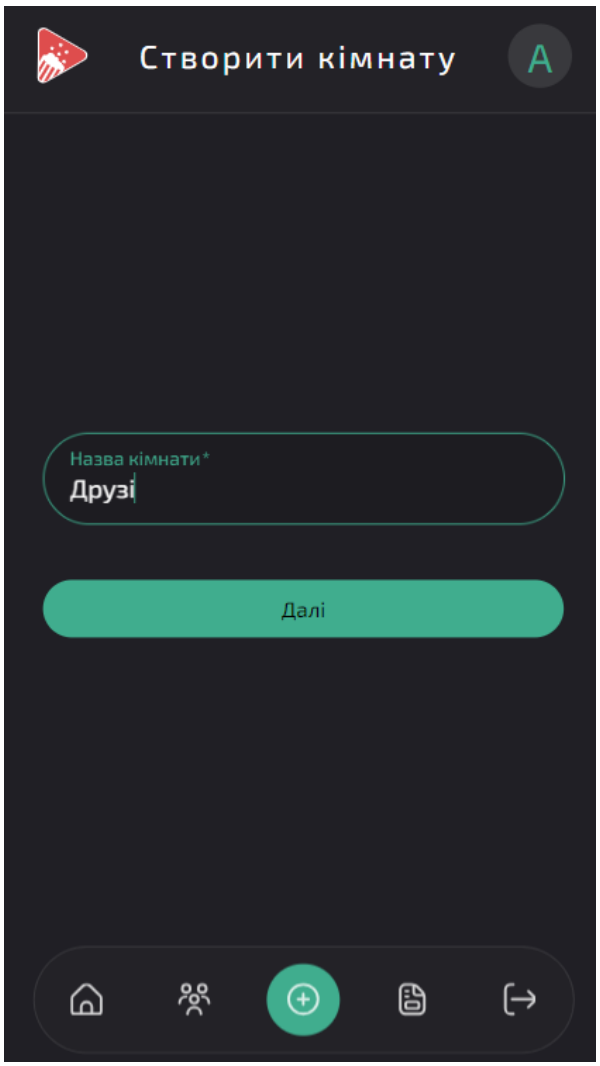

Рисунок В.15 – Сторінка для введення назви кімнати на мобільному пристрої

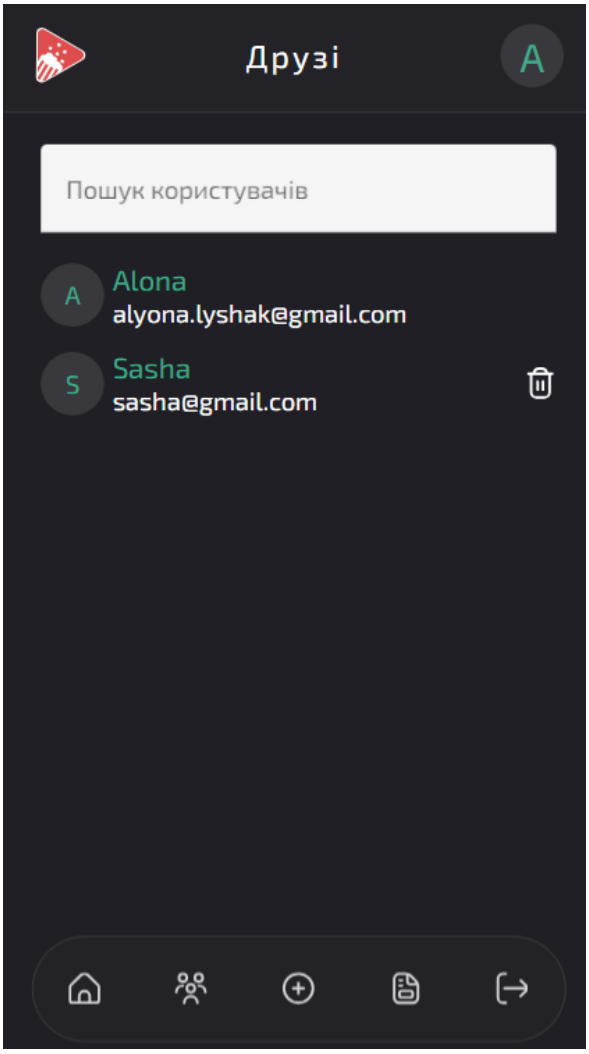

Рисунок В.16 – Сторінка для додавання користувачів кімнати на мобільному пристрої

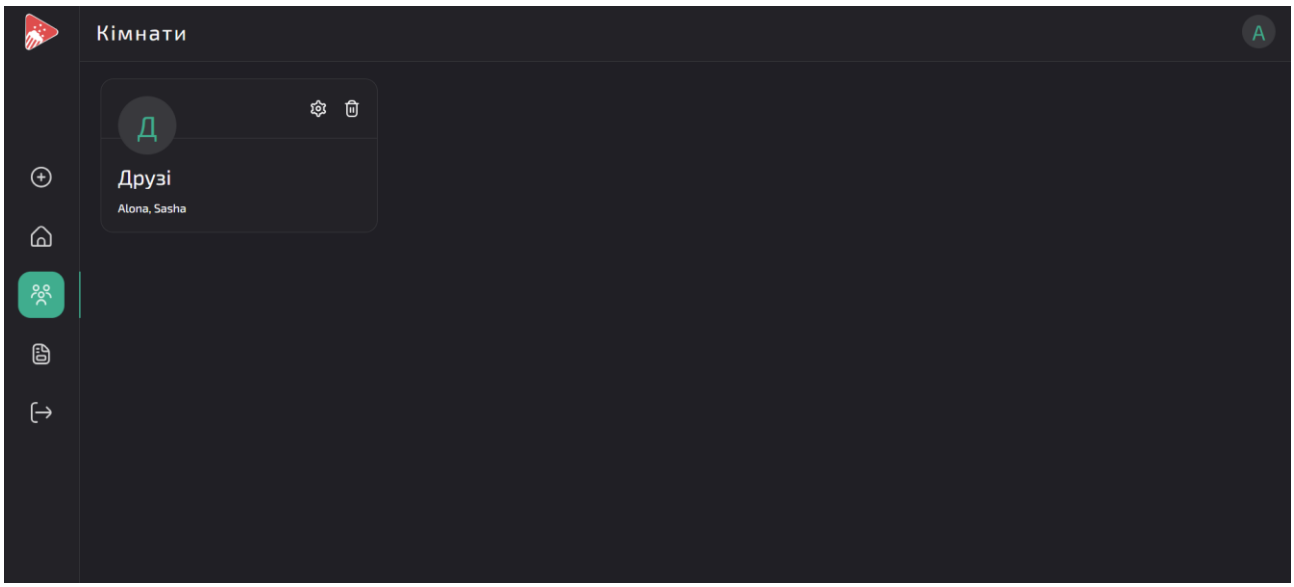

Рисунок В.17 – Сторінка з кімнатами користувача на WEB-ресурсі

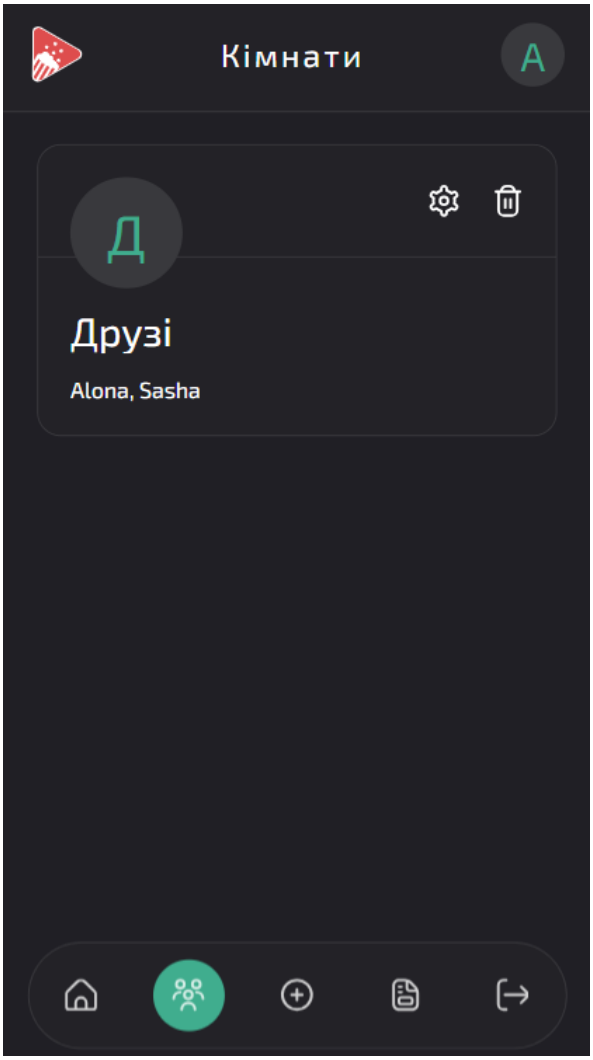

Рисунок В.18 – Сторінка з кімнатами користувача на мобільному пристрої

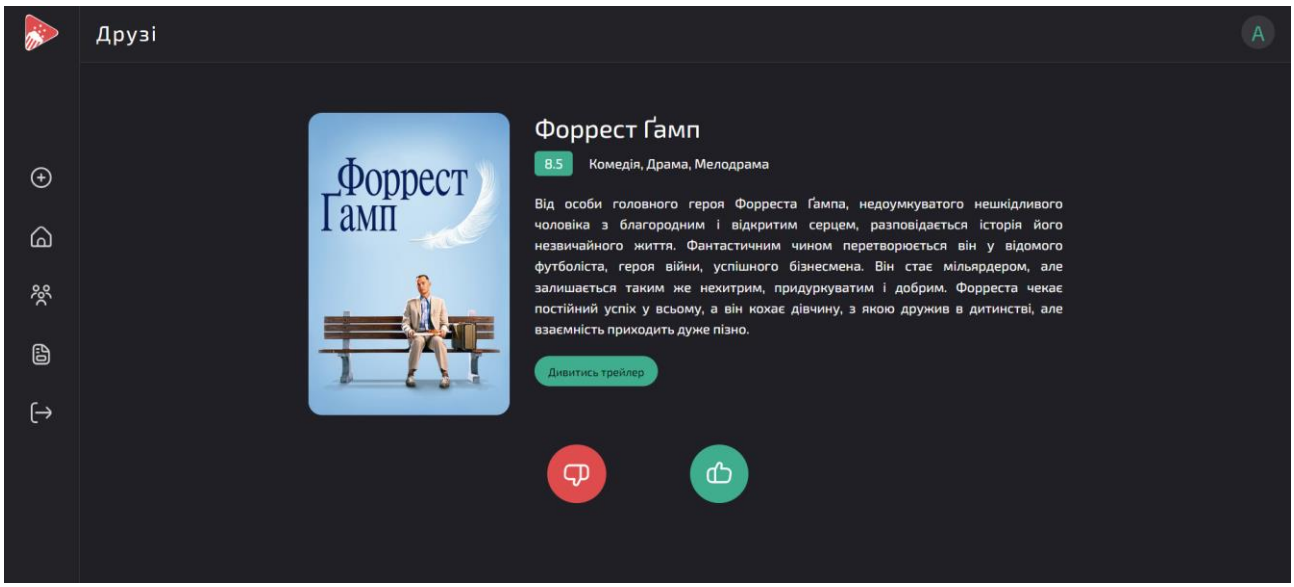

Рисунок В.19 – Сторінка для підбору фільмів на WEB-ресурсі

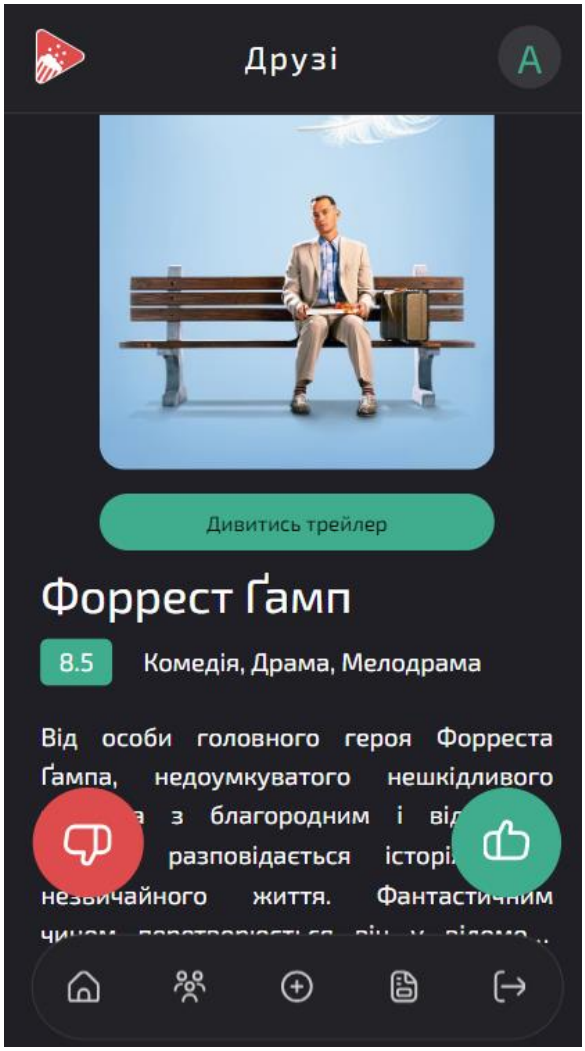

Рисунок В.20 – Сторінка для підбору фільмів на мобільному пристрої

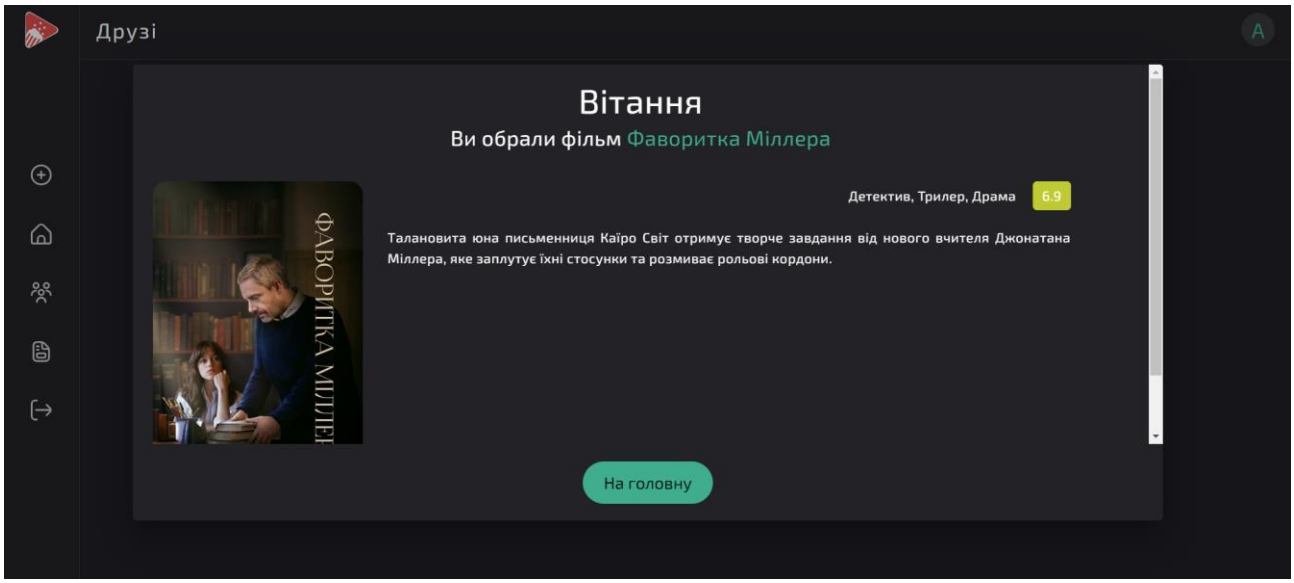

Рисунок В.21 – Сторінка з повідомленням про обраний для перегляду фільм на WEB-ресурсі

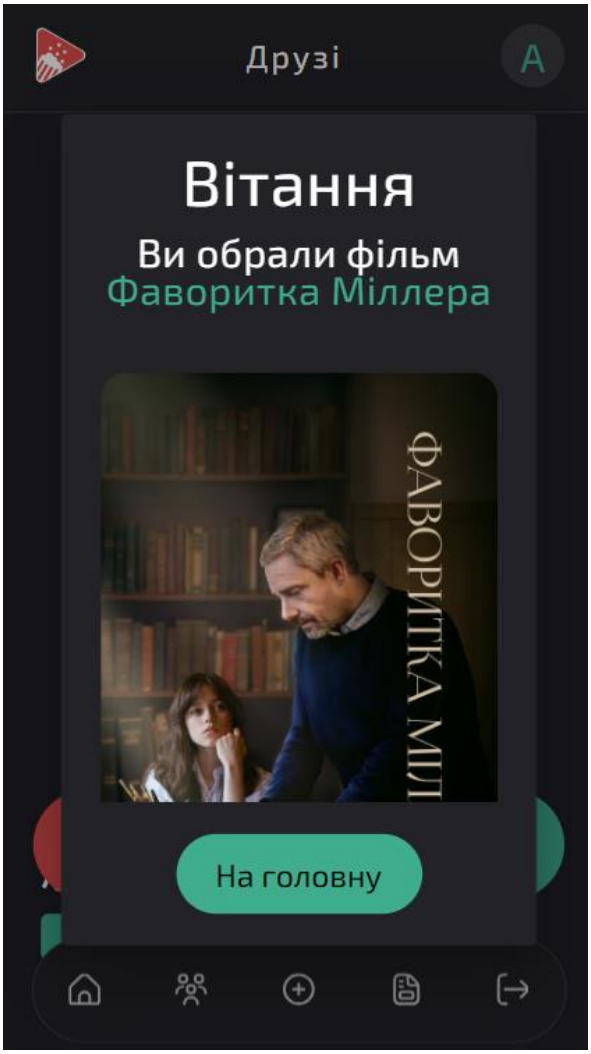

Рисунок В.22 – Сторінка з повідомленням про обраний для перегляду фільм на мобільному пристрої

ДОДАТОК Г (довідниковий)
Інструкція користувача

Для початку роботи з ресурсом потрібно зареєструватись або увійти в раніше створений акаунт (рис. Г.1, Г.2).

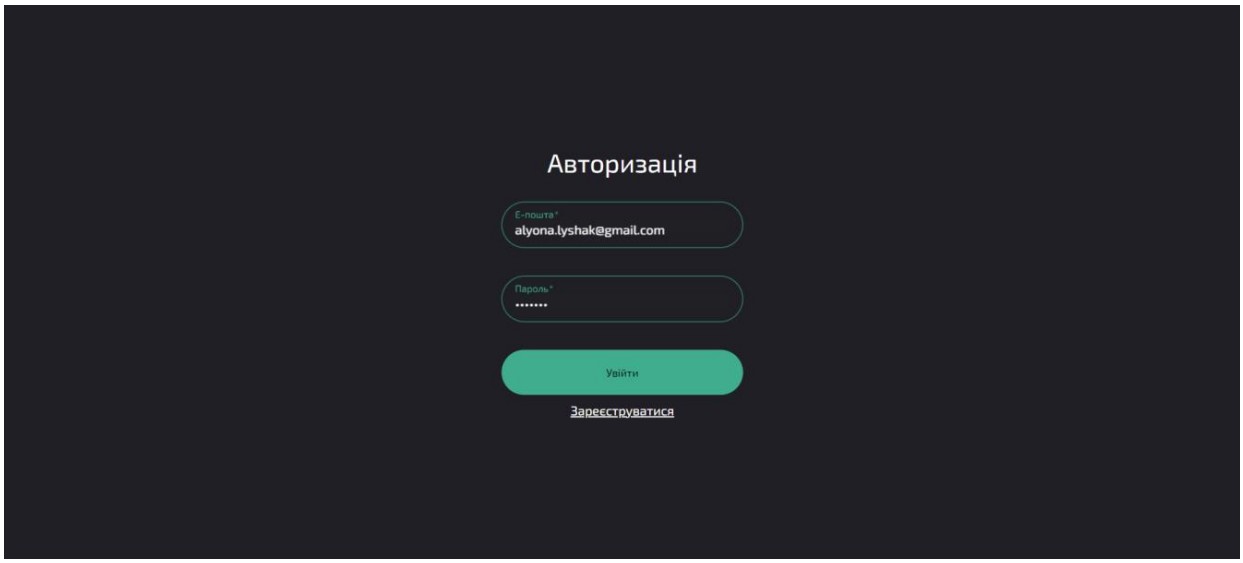

Рисунок Г.1 – Сторінка авторизації користувача

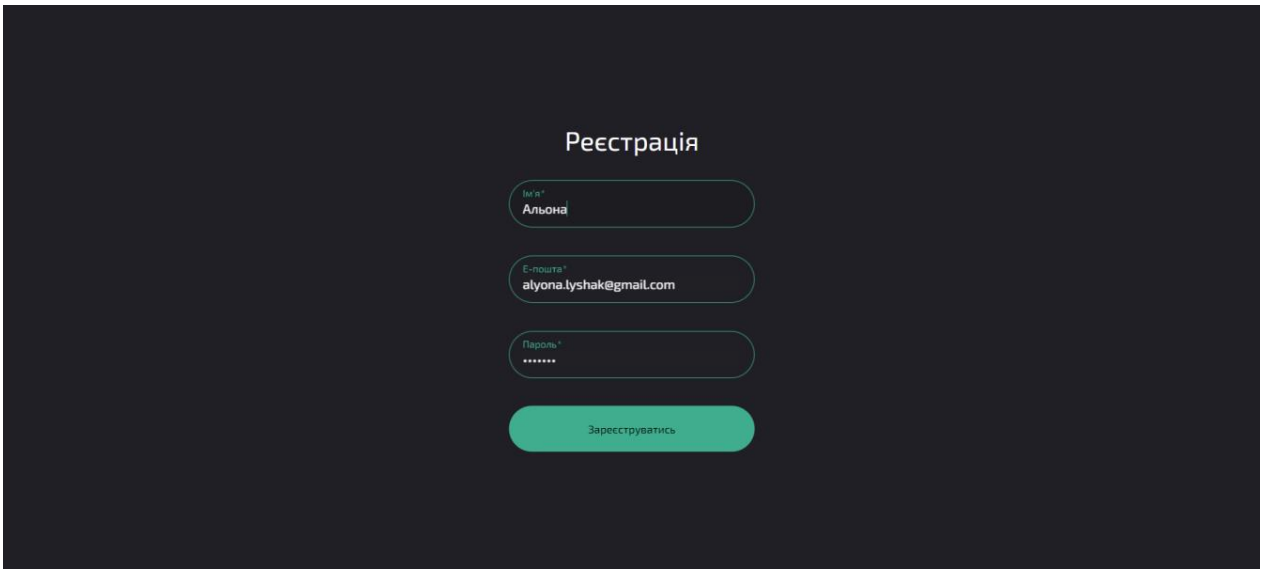

Рисунок Г.2 – Сторінка реєстрації користувача

Після реєстрації або авторизації користувача перекидає на головну сторінку (рис. Г.3). Тут користувач має можливість обрати в меню на боковій

панелі (або на нижній панелі якщо це мобільний пристрій) створення нової кімнати, перегляд існуючих кімнат або вихід з акаунту.

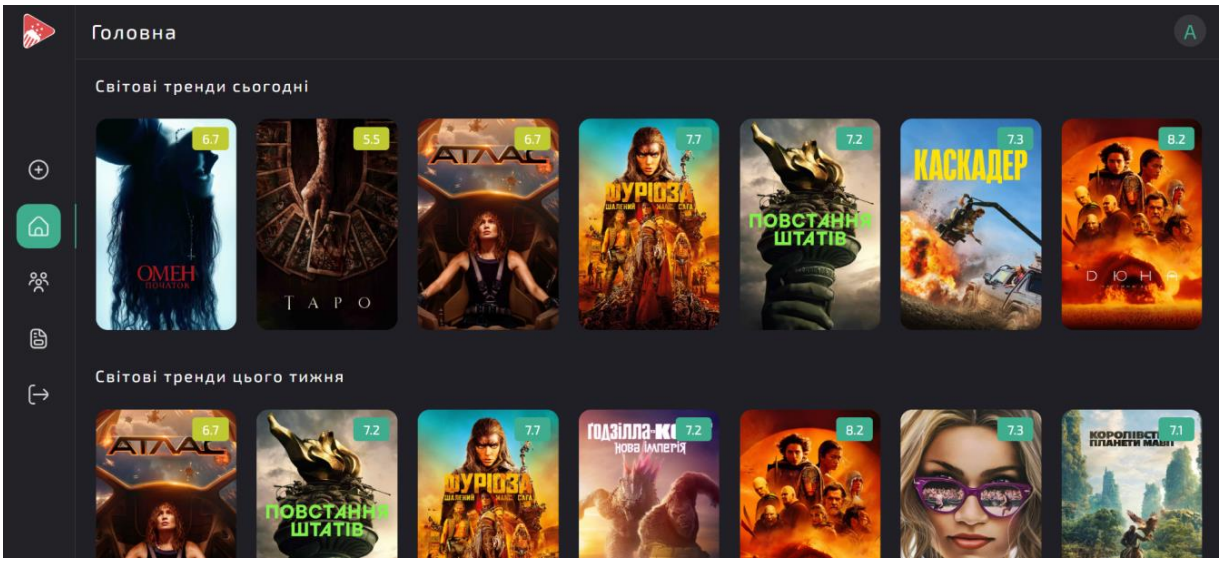

Рисунок Г.3 – Головна сторінка

Для того, щоб створити нову кімнату потрібно ввести назву кімнати та натиснути кнопку «Далі» (рис. Г.4). Після цього потрібно додати користувачів, для цього в полі пошуку потрібно ввести пошту користувача або ім’я та обрати його з випадаючого списку (рис. Г.5).

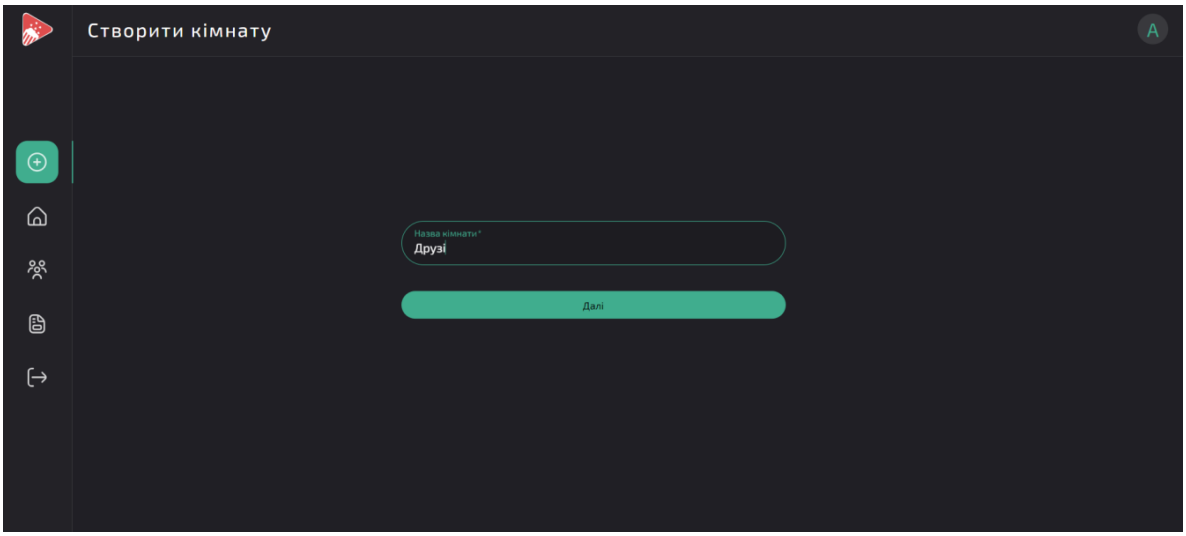

Рисунок Г.4 – Сторінка для введення назви кімнати на WEB-ресурсі

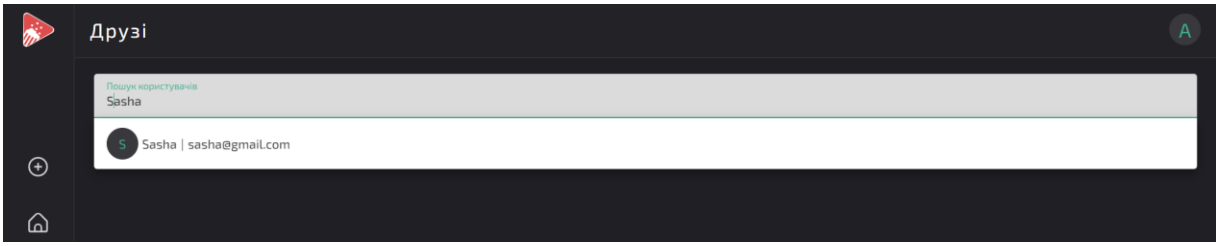

Рисунок Г.5 – Пошук та додавання користувача на WEB-ресурсі

На сторінці зі списком кімнат можна відредагувати або видалити будь-яку кімнату (рис. Г.6). Для того щоб розпочати підбір фільму потрібно вибрати одну з кімнат.

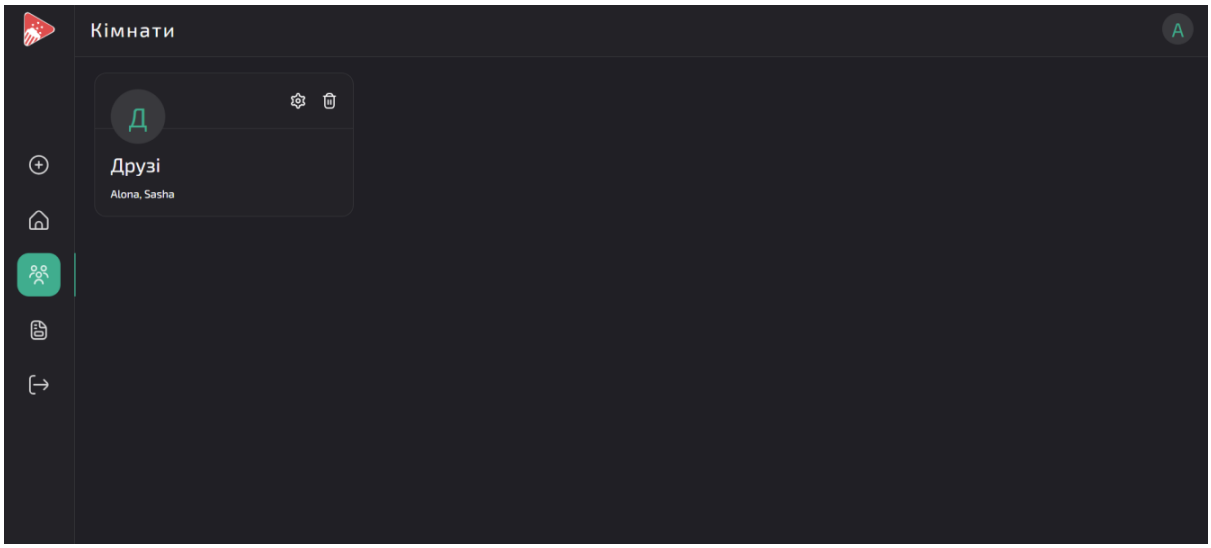

Рисунок Г.6 – Сторінка з кімнатами користувача на WEB-ресурсі

Після того як користувач обере кімнату алгоритм почне пропонувати фільми. Користувачу потрібно натиснути кнопку «Лайк», якщо фільм подобається або кнопку «Дизлайк», якщо фільм йому не сподобався (рис. Г.7).

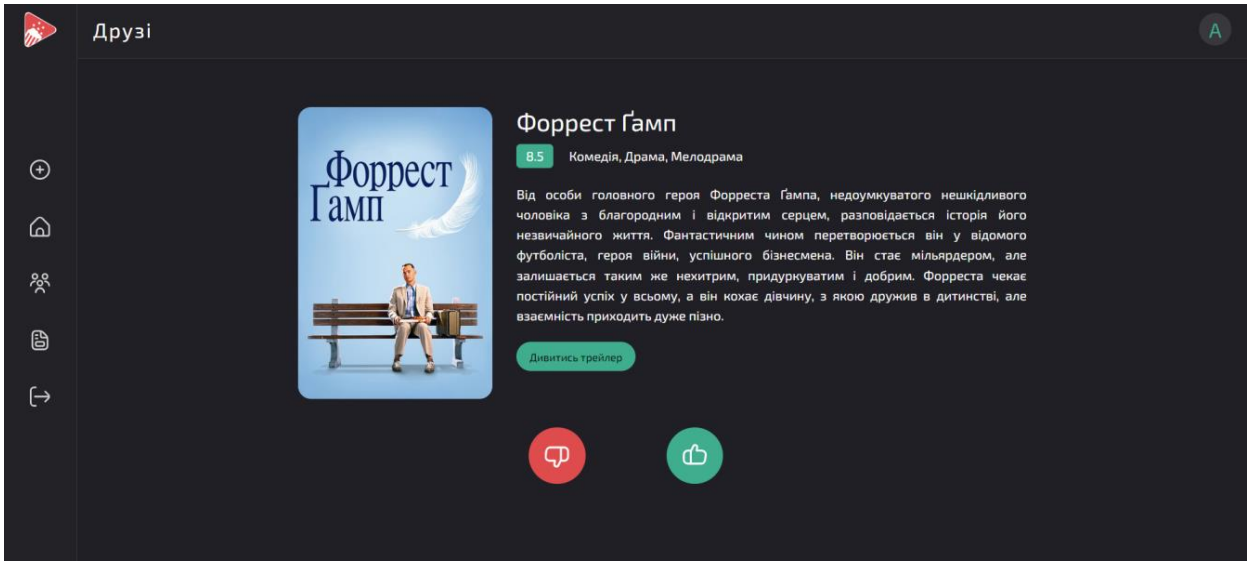

Рисунок Г.7 – Сторінка для підбору фільмів на WEB-ресурсі

Після успішного підбору фільму в користувача з’явиться вікно з повідомленням про підібраний фільм (рис. Г.8).

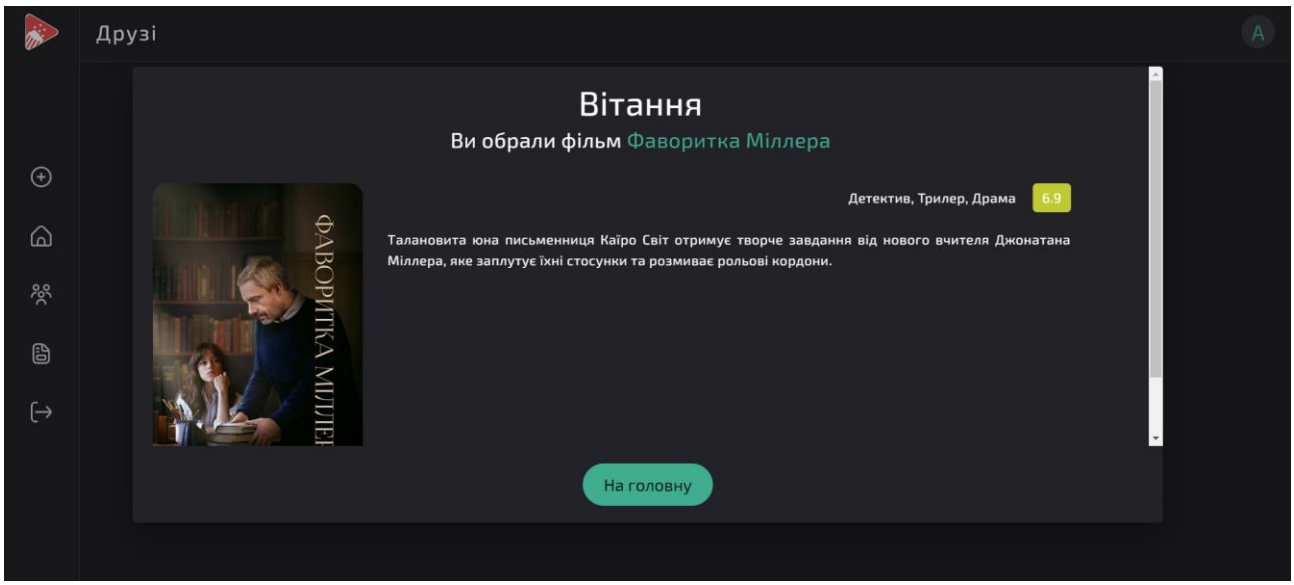

Рисунок Г.8 – Сторінка з повідомленням про обраний для перегляду фільм на WEB-ресурсі

На мобільному пристрої послідовність кроків аналогічна, змінюється лише розташування головного меню.