Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем та програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка мобільного застосунку для рекомендацій фільмів з використанням технологій React Native

Виконав(ла): студент(ка) 4 курсу, групи СП-43
спеціальності 121
Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Ништа І.І. (прізвище та ініціали)

Керівник (підпис) Михалик Д.М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2025

АНОТАЦІЯ

Розробка мобільного застосунку для рекомендацій фільмів з використанням React Native,// Кваліфікаційна робота освітнього рівня «Бакалавр» // Ништа Ірина Іванівна // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-43 // Тернопіль, 2025 // С. – 125, рис. – 41, табл. – 6, додат. – 3, бібліогр. – 18.

Ключові слова: React Native, Firebase Firestore, TMDb API, мобільний застосунок, рекомендації фільмів, користувач, відгуки, головна сторінка, персоналізація, проектування, тестування, аналіз вимог.

Об'єктом дослідження є мобільний застосунок для рекомендацій фільмів. У рамках роботи розглянуто актуальні проблеми інформаційного перенасичення, з якими стикаються користувачі під час вибору контенту, та обґрунтовано доцільність створення інтелектуальної системи рекомендацій, яка адаптується до вподобань користувача.

Метою роботи є розробка функціонального мобільного застосунку з інтуїтивним інтерфейсом, який дозволяє користувачам отримувати персоналізовані рекомендації фільмів, зберігати улюблені фільми, залишати відгуки, ставити оцінки, переглядати історію взаємодії, а також взаємодіяти з контентом у зручний спосіб. У процесі реалізації проєкту застосовувалися методи моделювання варіантів використання (Use Case), визначення класів сутностей, побудови діаграм, а також об'єктно-орієнтоване програмування. Технічна реалізація здійснювалась за допомогою фреймворку React Native та платформи Firebase для зберігання даних і реалізації авторизації. Для отримання даних про фільми використовувався зовнішній API The Movie Database (TMDb). Проєктування інтерфейсу виконувалось у Figma, що дозволило досягти сучасного вигляду та адаптивності застосунку.

У результаті роботи розроблено застосунок FilmBuddy, що підтримує авторизацію користувачів, персональні списки фільмів, механізм рекомендацій на основі жанрових вподобань і переглядів, функцію випадкового фільму, а також систему написання та оцінювання відгуків. Забезпечено розмежування доступу між авторизованими та гостьовими користувачами. Особливістю проєкту є реалізований прототип алгоритму рекомендацій, який може бути розширений у майбутньому для покращення точності добірок.

Наукова новизна роботи полягає у побудові архітектури системи з використанням рекомендаційного підходу, заснованого на векторному поданні жанрових вподобань і історії переглядів, а також у впровадженні гнучкої структури з можливістю масштабування.

Застосунок має практичну цінність як інструмент для персоналізованого вибору фільмів, а також як основа для створення комерційних чи освітніх мобільних платформ у сфері кіноіндустрії.

Економічна ефективність проєкту полягає у використанні безкоштовних і відкритих інструментів розробки, таких як React Native, Firebase, Figma, що дозволяє уникнути витрат на ліцензування програмного забезпечення.

У підсумку, застосунок FilmBuddy реалізовано відповідно до сучасних вимог до мобільних платформ. Результати роботи можуть бути використані в подальших дослідженнях і вдосконаленнях, зокрема у напрямі інтеграції моделей машинного навчання та рекомендацій на основі штучного інтелекту.

ABSTRACT

Development of a Mobile Application for Movie Recommendations Using React Native // Bachelor's Qualification Work // Nyshta Iryna Ivanivna // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Software Engineering, Group SP-43 // Ternopil, 2025 // Pages – 125, figures – 41, tables – 6, appendices – 4, references – 18.

Keywords: React Native, Firebase Firestore, TMDb API, mobile application, movie recommendations, user, reviews, main screen, personalization, design, testing, requirements analysis.

The object of the research is a mobile application for movie recommendations. The study addresses the relevant problem of information overload, which users face when choosing content, and justifies the need to develop an intelligent recommendation system that adapts to individual user preferences.

The aim of the work is to develop a functional mobile application with an intuitive interface that allows users to receive personalized movie recommendations, save favorite movies, leave reviews, rate content, view their interaction history, and conveniently engage with content. During the project implementation, methods such as use case modeling, entity class identification, diagram creation, and object-oriented programming were applied. The technical implementation was carried out using the React Native framework and the Firebase platform for data storage and user authentication. The application utilizes The Movie Database (TMDb) API to retrieve film data. Interface design was created in Figma, enabling a modern and adaptive user experience.

As a result, the FilmBuddy application was developed, featuring user authentication, personal movie lists, a recommendation mechanism based on genre preferences and viewing history, a random movie selection function, and a review and rating system. The application provides role-based access control for authorized and

guest users. A prototype recommendation algorithm has been implemented, with potential for future extension to enhance selection accuracy.

The scientific novelty of the work lies in designing a system architecture using a recommendation approach based on vector modeling of genre preferences and viewing history, along with a flexible structure that supports scalability.

The application has practical value as a tool for personalized movie selection and can serve as a foundation for the development of commercial or educational mobile platforms in the film industry.

The economic efficiency of the project lies in the use of free and open-source development tools such as React Native, Firebase, and Figma, which eliminates the need for software licensing costs.

In conclusion, the FilmBuddy application has been implemented in accordance with modern requirements for mobile platforms. The results of this work can be used in further research and improvements, particularly in the direction of integrating machine learning models and AI-based recommendation systems.

ЗМІСТ

ВСТУП

Щоденно у світі випускається сотні нових фільмів та серіалів, що створює для глядачів не лише безмежні можливості для перегляду, а й проблему вибору. Користувачі витрачають значну частину часу на пошук чогось вартого уваги серед величезної кількості доступного контенту, не завжди розуміючи, що саме їм хочеться подивитись. У таких умовах особливої актуальності набувають системи рекомендацій, здатні запропонувати персоналізовані варіанти на основі інтересів, історії переглядів і уподобань користувача.

Метою цього дипломного проєкту стало створення зручного мобільного застосунку FilmBuddy, який не лише автоматично підбирає фільми за вподобаннями користувача, а й дозволяє зручно зберігати улюблені стрічки, писати відгуки, оцінювати інші коментарі, вести власну історію переглядів. Застосунок розроблено з урахуванням сучасних стандартів зручності та дизайну, включаючи темну тему, жестову навігацію, адаптивне відображення для різних екранів і мінімалістичне оформлення.

Особливу увагу приділено системі рекомендацій, яка формує персоналізовані добірки фільмів на основі жанрового аналізу, вподобаних стрічок, збережених у списки "улюблене" чи "переглянуте", а також взаємодії з контентом інших користувачів. На відміну від класичних підходів, де фільми пропонуються за загальною популярністю, у FilmBuddy користувач отримує добірку, сформовану індивідуально, що суттєво підвищує релевантність рекомендацій.

Технічно FilmBuddy реалізований за допомогою React Native — фреймворку, який дозволяє створювати кросплатформенні застосунки з єдиною кодовою базою. Для зберігання даних використовується Firebase Firestore — хмарна база NoSQL, що підтримує структуру колекцій та документів і дозволяє працювати в режимі реального часу. Для отримання даних про фільми застосунок інтегровано з The Movie Database API (TMDb) — відкритою базою даних з актуальною інформацією про фільми, рейтинги, акторів та опис.

У проєкті реалізовано функціонал авторизації та реєстрації, редагування профілю, можливість переглядати детальну інформацію про кожен фільм, зберігати його в один із трьох списків, залишати коментарі та оцінювати відгуки інших користувачів. Вся інформація синхронізується з хмарною базою, зберігається з урахуванням прав доступу і може бути оновлена на будь-якому пристрої користувача.

Інтерфейс проєкту побудовано з урахуванням простоти та інтуїтивності. Усі основні функції згруповано по екранах — домашній, пошук, профіль. Додаткові компоненти, такі як відображення списків фільмів, картки, модальні вікна тощо, створені з дотриманням єдиного стилю та сучасних підходів до UI/UX-дизайну.

Застосунок FilmBuddy поєднує функціональність інтелектуальної рекомендаційної системи з інтуїтивно зрозумілим і привабливим інтерфейсом. У процесі реалізації було забезпечено підтримку персоналізованих добірок фільмів, динамічної системи відгуків з можливістю відповідей і оцінок, а також збереження особистих списків користувача. Усі ключові компоненти — від головного екрана до профілю, пошуку та деталізації фільму — інтегровані в єдину систему, що працює швидко та стабільно завдяки архітектурі на базі React Native і хмарному сховищу Firebase. Застосунок орієнтований на щоденне використання та забезпечує повноцінну взаємодію з кіноконтентом у зручному мобільному форматі, відповідаючи сучасним вимогам до продуктивності, безпеки та користувацького досвіду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз вимог до системи та постановка задачі

Сфера мобільних застосунків для перегляду та пошуку фільмів останніми роками активно розвивається. Завдяки широкому доступу до смартфонів, користувачі все частіше надають перевагу саме мобільним рішенням для організації свого дозвілля. Водночас, зі зростанням обсягу доступного контенту, виникає проблема, яка давно відома під назвою «інформаційне перевантаження». Серед сотень фільмів на різних платформах вибір одного для перегляду стає складним завданням. Усе частіше користувачі витрачають більше часу на пошук, ніж на сам перегляд.

У таких умовах актуальною стає розробка інтелектуальних рекомендаційних систем, які можуть адаптуватися до інтересів конкретного користувача та запропонувати йому контент, що відповідає його очікуванням. Водночас ефективна обробка великих обсягів інформації потребує застосування сучасних методів математичного моделювання та оптимізації. У наукових дослідженнях викладачів ТНТУ, зокрема М.Р. Петрика та співавторів [1], розглядаються високопродуктивні методи моделювання та ідентифікації параметрів у складних гетерогенних середовищах. Підходи, які використовуються у подібних роботах, можуть бути адаптовані і для задач персоналізованих рекомендацій у мобільних застосунках, де також необхідно враховувати поведінкові патерни користувачів і великі обсяги даних.

Водночас багато наявних застосунків, які рекомендують фільми, мають певні недоліки: обмежена персоналізація, складний інтерфейс або відсутність механізмів зворотного зв’язку. Часто рекомендації формуються лише на основі загальних рейтингів або популярності, без урахування унікальних вподобань користувача. До того ж, більшість таких застосунків не надає можливості зберігати історію перегляду, створювати особисті списки або взаємодіяти з іншими користувачами — наприклад, через відгуки.

Враховуючи вищезазначене, було прийнято рішення розробити мобільний застосунок FilmBuddy, який дозволить користувачам не лише отримувати персоналізовані рекомендації фільмів, а й взаємодіяти з контентом — зберігати фільми до обраних списків, залишати коментарі, ставити оцінки, переглядати свою історію переглядів.

Однією з ключових особливостей FilmBuddy має стати система рекомендацій, яка враховує жанри улюблених фільмів, взаємодію користувача з контентом (перегляди, відгуки, вподобання), а також загальні вподобання подібних користувачів. Важливо, щоб ця система працювала швидко, точно та непомітно для самого користувача, пропонуючи цікаві варіанти, навіть коли той не має чіткого запиту. Схожі підходи були розглянуті в окремих дослідженнях, зокрема у сфері мобільних рекомендаційних систем, де підкреслюється важливість простоти інтерфейсу, ефективності персоналізації та зворотного зв’язку з користувачем [2].

Окрім рекомендацій, застосунок має виконувати й інші функції: авторизацію користувачів, збереження персональних списків (улюблені, переглянуті, переглянути пізніше), пошук фільмів за назвою або жанром, перегляд детальної інформації про фільми (опис, актори, дата виходу, рейтинг), залишення та перегляд відгуків. Також важливим є розмежування функціоналу для авторизованих та неавторизованих користувачів: гість може переглядати фільми, але не може зберігати їх у списки або залишати коментарі.

Ще один важливий аспект — це зручність користування. Інтерфейс має бути інтуїтивно зрозумілим, адаптивним до різних екранів та з підтримкою темної теми. Простота взаємодії, візуальна привабливість та відсутність зайвих кроків — критично важливі для будь-якого сучасного мобільного застосунку, який планується для щоденного використання. Усе це дозволяє сформулювати основну мету дипломного проєкту: розробити мобільний застосунок для рекомендацій фільмів з персоналізованим функціоналом, сучасним інтерфейсом та підтримкою збереження користувацьких даних.

Для досягнення поставленої мети необхідно виконати наступні задачі:

- Дослідити існуючі рішення в галузі рекомендаційних систем і мобільних застосунків для фільмів.
- Спроєктувати інформаційну модель застосунку: акторів, сценарії використання, структуру бази даних.
- Реалізувати систему реєстрації та авторизації користувачів.
- Інтегрувати API The Movie Database (TMDb) для отримання фільмів та їх характеристик.
- Побудувати логіку персоналізованих рекомендацій.
- Реалізувати функції збереження списків, написання і оцінювання відгуків.
- Розробити простий та адаптивний інтерфейс користувача.
- Забезпечити безпечну взаємодію з Firebase для зберігання особистих даних і відгуків.
- Протестувати застосунок на предмет відповідності функціональним вимогам.

Таким чином, предметна область проєкту охоплює сучасні напрями у сфері мобільних рекомендаційних систем, інтерфейсного дизайну, зберігання користувацьких даних та захисту інформації. Актуальність розробки підтверджується практичним запитом користувачів на подібні зручні й персоналізовані інструменти.

Після формування загальної мети та постановки задачі проєкту, наступним етапом є визначення вимог до системи — як функціональних, так і нефункціональних. Це дозволяє чітко окреслити, що саме повинна робити система, якими засобами це буде реалізовано, а також за якими критеріями оцінюватиметься її ефективність та якість.

Функціональні вимоги описують конкретні дії, які система має виконувати у відповідь на взаємодію з користувачем. Це безпосередньо стосується ключових можливостей мобільного застосунку FilmBuddy: авторизації, перегляду фільмів, додавання у списки, написання відгуків тощо. Такі вимоги визначають поведінку системи та формують основу для її архітектурної реалізації.

Нефункціональні вимоги визначають якісні характеристики системи: продуктивність, зручність користування, безпеку, надійність і масштабованість. У випадку з мобільним застосунком для рекомендацій фільмів особливу увагу приділяється швидкому завантаженню, адаптивному інтерфейсу, стабільній роботі та захисту даних користувачів.

Нижче наведено перелік функціональних можливостей, які повинні бути реалізовані в межах застосунку, з поділом на функції для авторизованих і неавторизованих користувачів, а також основні нефункціональні вимоги, які визначають загальні технічні та експлуатаційні характеристики системи.

Основними функціональними вимогами системи для авторизованих користувачів є:

1) Авторизація користувачів:

- Застосунок дозволяє входити в обліковий запис вже зареєстрованих користувачів.

2) Редагування особистої інформації:

- Зареєстрований користувач має змогу змінити фото чи ім’я свого профілю.

3) Перегляд обраних фільмів:

- Користувач може переглядати обрані фільми, доданих до списків.

4) Перегляд відгуків до фільмів :

- Користувач може переглядати написані ним відгуки до фільмів.

5) Перегляд підбірок фільмів:

- Застосунок повинен надавати можливість користувачам переглядати підбірки фільмів за жанром.

6) Пошук фільму за назвою:

- Користувач може шукати фільми за її назвою.

7) Перегляд детальної інформації:

- Користувач має змогу подивитись детальну інформацію про фільм, наприклад, опис фільму, актори, рейтинг, дата виходу.

8) Додавання фільму до списку:

- Користувач повинен мати можливість додавати фільми в списки: улюблені, переглянуті і переглянути пізніше.

9) Додавання відгуку:

- Користувач повинен мати змогу писати коментарі;
- Користувач повинен мати змогу відповідати на відгуки інших користувачів;
- Користувачі повинні мати змогу ставити вподобайку і дизлайк на відгук.

10) Перегляд рекомендованих фільмів:

- Застосунок повинен надати користувачеві підбірку рекомендованих фільмів.

11) Випадковий фільм:

- Користувач повинен мати можливість отримати фільм, обраний випадковим чином.

Неавторизовані користувачі, які ще не зареєструвались або не ввійшли в систему, потребують доступ до базових функцій застосунку. Система повинна надавати наступні функціональні можливості для них:

1) Автентифікація:

- Користувачі мають мати можливість увійти в систему, використовуючи облікові дані(email і пароль), щоб користуватись додатковими функціями застосунку

2) Реєстрація:

- Нові користувачі мають мати можливість зареєструвати свій обліковий, ввівши email і пароль.

Нефункціональні вимоги включать:

1) Швидке завантаження даних
2) Просту та адаптивну навігацію
3) Локальну кешовану підтримку даних
4) Подальшу інтеграцію Firebase для збереження відгуків та історії користувача.

У результаті аналізу було визначено чіткий перелік функціональних та нефункціональних вимог до системи. Це дозволяє сформувати уявлення про основні дії, які має виконувати застосунок, та забезпечити очікуваний рівень продуктивності, зручності й масштабованості. Урахування потреб як авторизованих, так і неавторизованих користувачів дає змогу розробити гнучкий і ефективний функціонал.

Окрім функціональних і нефункціональних характеристик, важливо також врахувати вимоги до зовнішнього вигляду системи та до її безпечної експлуатації. Саме інтерфейс є тією частиною, з якою безпосередньо взаємодіє користувач, тому він повинен бути інтуїтивно зрозумілим, привабливим та зручним у використанні. Інтерфейс мобільного застосунку має сприяти швидкому доступу до основних функцій, не перевантажувати користувача, а також підтримувати адаптацію до різних пристроїв та стилістичних налаштувань.

Не менш важливим є забезпечення належного рівня інформаційної безпеки. Оскільки система працює з персональними даними користувачів (email, список фільмів, відгуки), необхідно передбачити захищену автентифікацію, обмеження доступу до приватного контенту та збереження даних у захищених сховищах. Надійність системи має бути гарантована як з технічного, так і з правового боку — з урахуванням принципів конфіденційності, цілісності та доступності інформації.

У цьому підпункті будуть детально описані основні вимоги до інтерфейсу користувача, а також принципи та засоби, за допомогою яких буде реалізовано захист даних у системі FilmBuddy.

Вимоги до інтерфейсу:

Користувацький інтерфейс мобільного застосунку FilmBuddy повинен бути простим, інтуїтивно зрозумілим та естетично привабливим. Його головним завданням є забезпечення зручного доступу до основних функцій системи, з урахуванням як потреб користувача, так і вимог платформи.

Інтерфейс застосунку повинен підтримувати адаптивний дизайн, який дозволить однаково добре відображати компоненти на екранах смартфонів різних

розмірів. Головний екран повинен надавати швидкий доступ до пошуку фільмів, рекомендованих підбірок та особистого профілю. Кожен екран має мати чітку структуру та логічну навігацію між розділами.

Також важливою вимогою є підтримка темної та світлої теми оформлення, що покращує користувацький досвід та зменшує навантаження на зір. У налаштуваннях профілю користувач повинен мати можливість змінити тему інтерфейсу за власним бажанням. Крім того, слід передбачити чітке візуальне розмежування для авторизованих і неавторизованих користувачів.

Інтерфейс повинен підтримувати динамічне оновлення даних, наприклад, при додаванні фільму до списку або написанні відгуку. Усі дії мають супроводжуватись відповідними анімаціями, спливаючими повідомленнями про успіх чи помилку. Для введення тексту та пошуку мають використовуватись зручні компоненти введення, з підтримкою підказок та автозаповнення.

У майбутньому передбачається можливість реалізації мультимовного інтерфейсу із використанням локалізованих текстових ресурсів, що дозволить залучити ширшу аудиторію користувачів. Кожен компонент інтерфейсу має бути протестований на доступність (accessibility), зокрема для користувачів із вадами зору.

Аналіз системних вимог дозволяє сформувати чітке уявлення про необхідний функціонал застосунку. Основна увага приділяється зручності для користувача, можливості взаємодії через відгуки, а також ефективній організації інтерфейсу. Визначенні функціональні і нефункціональні вимоги слугують для подальшого проєктування застосунку.

Вимоги до захисту та доступу:

Забезпечення надійного захисту персональних даних користувачів та контроль доступу до ресурсів системи є критично важливими для безпеки мобільного застосунку FilmBuddy. Оскільки користувачі створюють облікові записи, залишають відгуки, а також зберігають персональні списки фільмів, система повинна гарантувати конфіденційність, цілісність та доступність цієї інформації.

Для цього застосовується система автентифікації через Firebase Authentication, яка дозволяє надійно ідентифікувати користувачів за допомогою електронної пошти та пароля. Тільки авторизовані користувачі мають доступ до створення контенту, редагування профілю та взаємодії з відгуками. Неавторизовані користувачі можуть лише переглядати доступні фільми та їх опис.

Операції зі створення, редагування та видалення даних користувача виконуються з урахуванням його прав доступу. Усі запити до бази даних супроводжуються перевіркою авторизації відповідно до правил безпеки Firestore, які обмежують зміну даних лише автору контенту.

Для запобігання зловживанням системою реалізовано захист від багаторазового голосування (лайків/дизлайків) з боку одного користувача. Також у системі обмежено можливість перегляду або взаємодії з певними функціями для гостьового режиму.

Усі дані передаються по захищеному каналу зв’язку (HTTPS). В майбутньому передбачено розширення системи безпеки, зокрема впровадження двофакторної автентифікації, журналу дій користувача, а також можливість блокування облікового запису за підозрілу активність.

Таким чином, система забезпечує необхідний рівень безпеки для захисту персональних даних і стабільної роботи застосунку в умовах реального використання.

Вимоги до інтерфейсу та безпеки мають критичне значення для користувацького досвіду та довіри до системи. Зручний, адаптивний і доступний інтерфейс забезпечує комфортну взаємодію з додатком, а реалізація сучасних засобів захисту даних гарантує конфіденційність та надійність збереження персональної інформації. Комплексний підхід до цих вимог формує основу для створення якісного та безпечного застосунку.

1.2 Опис методології проєктування та інструментів розробки

Розробка програмного забезпечення вимагає чіткого підходу до організації етапів створення системи. У контексті проєкту, що передбачає реалізацію мобільного застосунку для рекомендацій фільмів, особливу увагу було приділено правильному вибору метедології розробки, сучасних технологій, інструментів та фреймворків, які б дозволили створити ефективне, масштабоване і зручне рішення для кінцевого користувача.

Під час розробки застосунку було застосовано ітераційну модель програмної інженерії, передбачає повторне виконання кожного етапу проєкту з урахуванням отриманого зворотного зв’язку від користувачів або замовника [3] (див. рис. 1.1). Кожен етап включає уточнення вимог, реалізацію певного функціоналу, його перевірку та адаптацію відповідно до поставлених завдань. Такий підхід дозволив поступово створити стабільну систему з гнучкою архітектурою та можливістю подальшого розширення.

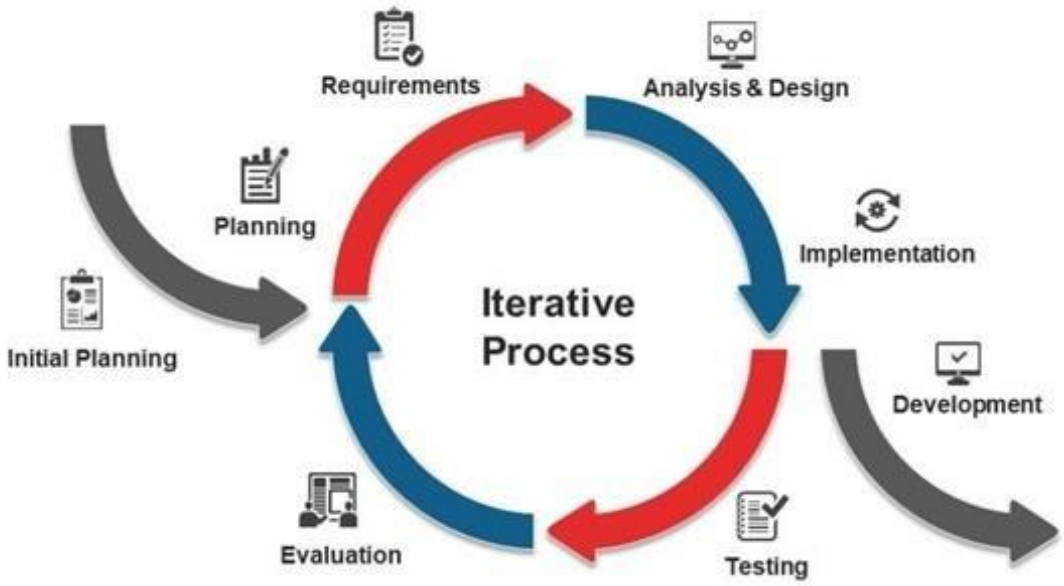

Рисунок 1.1 – Приклад метедології ітераційна модель ПІ [4]

Крім того, у проєкті застосовано принципи об’єктно-орієнтованого програмування(ООП) та модульного підходу, що забезпечили логічну структуру проєкту, повторне використання компонентів і зменшення залежностей між модулями.

Проєктування було підтримано за допомогою UML-діаграм, зокрема:

- Діаграма варіантів використання (Use Case Diagram) для відображення взаємодії користувача із системою;
- Діаграма класів, яка демонструє логіку роботи внутрішніх сутностей та їх зв’язків;
- Базові блок-схеми логіки переходів між екранами.

У процесі реалізації застосунку було використано низку сучасних інструментів, які сприяли швидкій розробці, зручному тестуванню та масштабованості рішення. Нижче описано детальніше про них.

1) React Native – це популярний фреймворк для мобільних застосунків на основі JavaScript, який дозволяє створювати мобільні застосунки для IOS та Android з нативною рендерингом. Фреймворк дозволяє створювати застосунки для різних платформ, використовуючи одну й ту саму кодову базу [5]. На рисунку 1.2 показано як працює React Native. Для застосунку для рекомендацій фільмів(FilmBuddy) React Native використовується для створення інтерфейсу, управління станом, логіки навігації та взаємодії з API. Завдяки великій спільноті розробників та великій кількості бібліотек, фреймворк дозволяє реалізовувати навіть складні функції – від swipe-жестів до push-повідомлень.

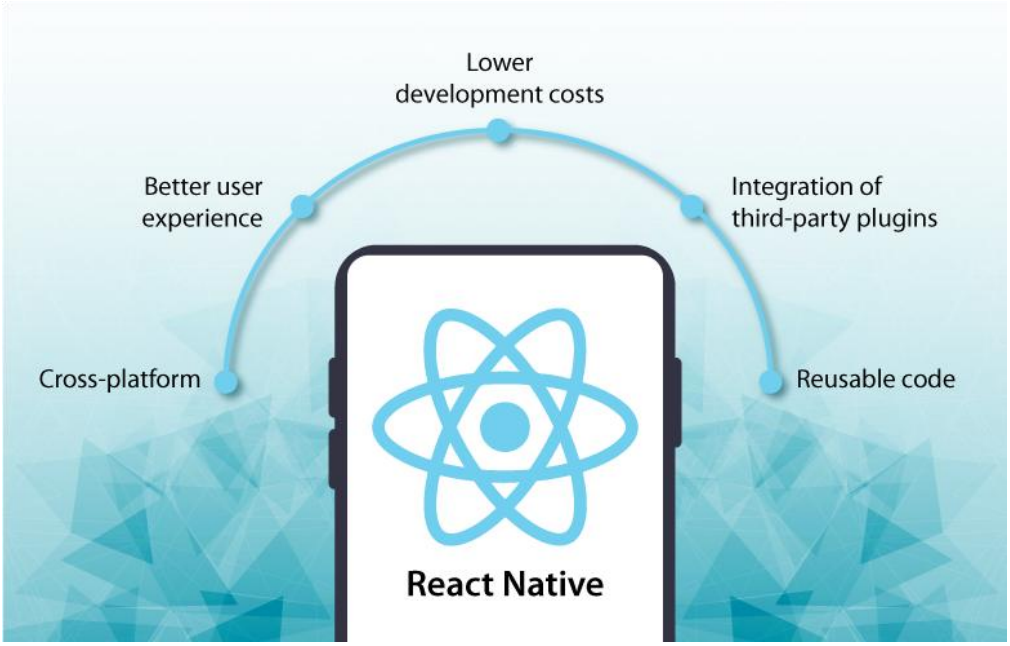

Рисунок 1.2 – React Native [5]

2) Expo – це безкоштовний фреймворк з відкритим кодом для створення React Native застосунків [6]. Expo дозволяє запускати застосунок без необхідності встановлення Android Studio або Xcode, що значно спрощує тестування.

Expo також забезпечує легкий доступ до функцій камери, мультимедіа, push-нотифікації та локального сховища, що особливо корисно в мобільних застосунках.

3) TMDb – це онлайн база даних, що стосується фільмів, телевізійних програм в інтернеті – включаючи акторський склад, виробничу групу та особисті біографії, опис сюжетів, рейтинги. Кожна частина даних була додана співтовариством починаючи з 2008 року [7].

Запити до TMDb виконуються за допомогою функцій fetch() з обробкою JSON-результатів. Система кешування зображень допомагає знизити навантаження на мережу та покращити швидкість завантаження інтерфейсу.

4) Firebase – це потужна платформа для мобільних та веб-застосунків. Firebase може забезпечити роботу серверної частини застосунку [8]. Його переваги:

- Зручна та безпечна авторизація за email/паролем;
- Хмарне зберігання відгуків, історії переглядів та списків користувача;

- Можливість створення правил безпеки;
- Масштабованість – дані автоматично синхронізуються з клієнтом у реальному часі.

5) Visual Studio Code (VS Code) – це безкоштовний застосунок, створений Microsoft на базі відкритого коду [9]. Було встановлено розширення для підтримки React Native, ESLint, Prettier, GitHub Copilot значно спрощує розробку та забезпечує єдиний стиль коду.

6) Figma – це інструмент дизайну, який швидко набирає популярності та стає все більш поширеним у компаніях по всьому світу [10]. Figma застосовують для прототипування та дизайну інтерфейсу. Це дозволяє створити естетично привабливий, зручний та логічно побудований UI.

7) Git – це система контролю версій, яка дозволяє відстежувати зміни, внесені під час створення проєкту з кодом [11]. Це дозволяє створювати гілки для кожної функціональності.

8) GitHub – це онлайн-платформа, де розробники зберігають, керують своїм кодом або проєктом [11]. Він забезпечив збереження історії змін, роботу з pull-request`ами та командну співпрацю в майбутньому.

Комбінація ітераційної моделі, об’єктно-орієнтованого підходу, UML-моделювання, а також використання сучасних технологій (React Native, Firebase, TMDb API) забезпечила надійну та ефективну реалізацію мобільного застосунку FilmBuddy. Це дозволило досягнути високого рівня інтерактивності, продуктивності та зручності для користувачів. Усі вибрані інструменти є безкоштовними, активно підтримуються спільнотою та можуть масштабуватись разом із ростом функціоналу системи.

1.3 Проєктування відношень між акторами і прецедентами

Одним із ключових етапів проєктування програмного забезпечення є побудова діаграми варіантів використання (Use Case Diagram), яка дозволяє візуалізувати взаємодію користувачів із системою та охопити всі можливі сценарії використання. Така діаграма відображає, які дії може виконувати користувач, які процеси відбуваються всередині системи та які компоненти за це відповідають.

У межах розробки мобільного застосунку FilmBuddy було проаналізовано логіку взаємодії користувача з системою та побудовано відповідну структуру відношень між актором і прецедентами. Основним актором є Користувач, який може бути або неавторизованим, або авторизованим. Тип доступних функцій безпосередньо залежить від статусу користувача в системі.

У системі визначено два основні актори:

1) Неавторизований користувач(гостьовий режим) – користувач, який не пройшов автентифікацію. Має обмежений доступ до застосунку: може переглядати список фільмів, переглядати детальну інформацію про кожен фільм, а також виконувати пошук. Для отримання розширеного функціоналу має пройти реєстрацію або вхід.

На діаграмі варіантів використання, зображеній нижче(див. рис. 1.3.), гостьовий користувач взаємодіє з базовими функціями системи, які дозволяють йому переглядати контент, але не взаємодіяти з ним.

Зокрема, гість може:

- Переглядати рекомендації та випадкові фільми;
- Здійснювати пошук фільмів за назвою;
- Переглядати список фільмів за жанрами;
- Переглядати детальну інформацію про фільми(опис, рейтинг, актори тощо);
- Перейти до форми реєстрації або входу для отримання повного доступу.

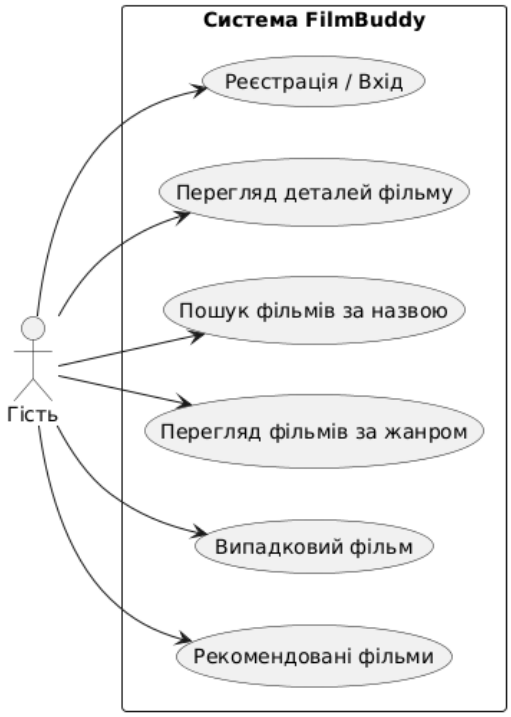

Рисунок 1.3 – Діаграма Варіантів Використання для гостя

2) Авторизований користувач – користувач, який успішно зареєстрований та виконав вхід у систему. Для нього відкритий повний функціонал застосунку. Він може:

- Керувати персональними списками фільмів (улюблені, переглянуті, переглянути пізніше);
- Переглядати рекомендації та випадкові фільми;
- Додавати фільми до списків;
- Писати відгуки до фільмів, відповідати на інші відгуки, оцінювати їх (лайки/дизлайки);
- Переглядати власні відгуки та редагувати налаштування профілю.

На діаграмі нижче (див. рис. 1.4) видно, що всі дії користувача згруповані у межах системи FilmBuddy, а також чітко позначено залежності типу <<extend>>, які підкреслюють логічну ієрархію між сценаріями.

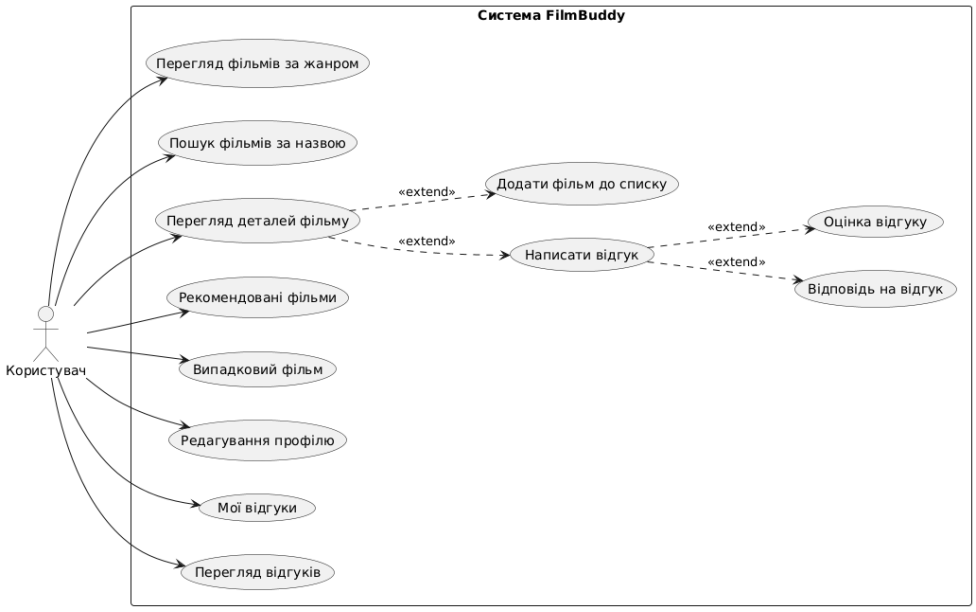

Рисунок 1.4 – Діаграма ВВ для авторизованого користувача

У результаті проєктування відношень між акторами та варіантами використання було виявлено два основні типи користувачів – гість та авторизований користувач. Для кожного х них визначено набір доступних дій у системі FilmBuddy. Побудовані діаграми варіантів використання дозволяють візуалізувати логіку взаємодії користувачів із застосунком, визначити залежності між функціональними можливостями та сформувати чітке уявлення про майбутню архітектуру системи. Це є важливим етапом підготовки до реалізації користувацьких сценаріїв та програмних модулів.

1.4 Варіанти використання за акторами

Після визначення основних акторів системи та побудови діаграм варіантів використання, було проведено деталізацію можливих сценаріїв взаємодії користувачів із функціональністю застосунку FilmBuddy. Такий підхід дозволяє описати не лише загальні дії користувачів, а й конкретну послідовність кроків, що реалізуються в межах кожного сценарію.

Для кращої структуризації варіантів використання створено таблицю, яка містить коди дій, основного актора, коротку назву дії та її покрокове формування. Це дозволяє однозначно інтерпретувати функціональні можливості системи та надалі використовувати ці описи при реалізації інтерфейсу користувача та логіки взаємодії з базою даних. Нижче наведено(див. табл. 1.1) узагальнені сценарії використання системи FilmBuddy для обох типів користувачів — неавторизованого (гостя) та авторизованого:

Таблиця 1.1 – Варіанти використання системи

Код	Основний актор	Найменування	Формування
Г1	Гість	Перегляд фільмів за жанром	1. Зайти на головний екран. 2. Обрати жанр зі списку. 3. Переглядати доступні фільми.
Г2	Гість	Пошук фільмів за назвою	1. Ввести назву у поле пошуку. 2. Отримати список відповідних результатів.
Г3	Гість	Перегляд дет. інформації про фільм	1. Обрати фільм зі списку. 2. Перейти на сторінку з детальною інформацією.
Г4	Гість	Реєстрація / Вхід	1. Перейти до розділу входу. 2. Ввести email і пароль.
			3. Підтвердити дію.
К1	Користувач (авторизований)	Додавання фільму до списку	1. Перейти на сторінку фільму. 2. Обрати тип списку. 3. Натиснути «Додати».
К2	Користувач (авторизований)	Написання відгуку	1. Відкрити сторінку фільму. 2. Ввести текст відгуку. 3. Натиснути «Надіслати».

Продовження таблиці 1.1

К3	Користувач (авторизований)	Відповідь на відгук	1. Обрати відгук іншого користувача. 2. Натиснути «Відповісти». 3. Написати коментар.
К4	Користувач (авторизований)	Оцінювання відгуку	1. Обрати відгук. 2. Натиснути «Клас» або «Дизлайк».
К5	Користувач (авторизований)	Перегляд власних відгуків	1. Зайти у профіль. 2. Перейти до розділу «Мої відгуки».
К6	Користувач (авторизований)	Отримання рекомендованих фільмів	1. Перейти на головний екран. 2. Переглянути блок «Рекомендовані фільми».
К7	Користувач (авторизований)	Отримання випадкового фільму	1. Натиснути кнопку «Випадковий фільм». 2. Перейти до сторінки фільму.
К8	Користувач (авторизований)	Редагування профілю	1. Перейти в профіль. 2. Натиснути «Редагувати». 3. Змінити ім’я або фото. 4. Зберегти.

1.5 Виявлення класів сутності

На цьому етапі розробки мобільного застосунку FilmBuddy було здійснено аналіз предметної області з метою визначення основних сутностей, які необхідні для зберігання, обробки та відображення даних у системі. Виявлення сутностей – це критичний крок у процесі моделювання, що дозволяє сформувати структуру даних та забезпечити узгодженість між логікою інтерфейсу, базою даних і внутрішньою логікою застосунку.

Сутності формуються на основі функціональних вимог, сценаріїв використання, а також взаємодії користувачів з системою. Вони лежать в основі

проєктування як бази даних, так і об’єктно-орієнтовних компонентів у коді застосунку.

Основні сутності, визначені для системи FilmBuddy:

1) User (Користувач) – відображає зареєстровану особу, яка має власний профіль у системі. Ключові атрибути:

- ID
- Ім’я користувача
- Email
- Пароль
- Фото профілю
- Списки фільмів (улюблені, переглянуті, переглянути пізніше

2) Movie (Фільм) – описує фільм, дані про який отримуються через API TMDb.

Основні поля:

- ID
- Назви
- Жанри
- Опис
- Постер
- Рік виходу
- Рейтинг
- Актори

3) Review (Відгук) -текстовий коментар, залишений користувачем до фільму.

Має такі атрибути:

- ID
- Текст відгуку
- Автор
- Кількість вподобань/дизлайків
- Відповіді

4) Genre (Жанр) – допоміжна сутність, що дозволяє класифікувати фільми. Має зв’язок багато-до-багатьох.

5) Comment (Відповідь на відгук) – реалізує вкладену комунікацію. Може бути реалізована як окремий клас або як ієрархічна структура в Review.

6) Rating (Оцінка відгуку) – фіксує дію користувача щодо вподобань або дизлайку на відгук.

Атрибути:

- ID
- ReviewID
- UserID
- Тип оцінки (like/dislike)

7) MovieListItem – допоміжна сутність для зв’язку між користувачем та фільмами, які він додав у певний список. Має тип списку: “favorite”, “watched”, “toWatch”.

У ході аналізу предметної області мобільного застосунку FilmBuddy було визначено ключові сутності, необхідні для ефективного зберігання та обробки даних у системі. Виділені класи, зокрема User, Movie, Review, Rating, MovieListItem та Genre, повністю охоплюють логіку роботи з фільмами, відгуками, оцінками та персональними діями користувачів. Описані атрибути та зв’язки між сутностями стали основою для подальшого проєктування структури бази даних та реалізації бізнес-логіки застосунку. Такий підхід забезпечує узгодженість між функціональними вимогами та технічною реалізацією системи.

1.6 Моделювання словника системи

Словник системи – це формалізований перелік термінів і понять, які використовуються під час розробки програмного забезпечення. Він є важливим артефактом проєктування, оскільки забезпечує єдине розуміння ключових

сутностей, дій і ролей серед усіх учасників розробки. Особливо це актуально для проєктів, що мають багатокомпонентну структуру, взаємодіють з базою даних, сторонніми API та підтримують різні типи користувачів.

У межах розробки мобільного застосунку FilmBuddy було створено словник основних термінів, що описують предметну область, функціональність і логіку роботи системи.

Основні терміни та їх опис:

1. Користувач – особа, яка використовує застосунок. Може бути неавторизований(гість) або авторизованим.
2. Гість – користувач, який не виконав вхід у систему. Має доступ до перегляду фільмів, але не до взаємодії з ними.
3. Фільм – основна одиниця контенту в системі. Має назву, жанр, опис, рік виходу, постер та ін.
4. Жанр – категорія, до якої належить фільм. Один фільм може мати кілька жанрів.
5. Відгук – коментар, який залишає авторизований користувач до конкретного фільму.
6. Оцінка відгуку – лайк або дизлайк, поставлений іншим користувачем до певного відгуку.
7. Відповідь на відгук – коментар до відгуку іншого користувачащо реалізує вкладену комунікацію.
8. Рекомендовані фільми – автоматично згенерований список фільмів на основі вподобань користувача.
9. Випадковий фільм – випадково вибраний фільм зі всієї доступної бази для перегляду.
10. Списки фільмів – колекції, які користувач може наповнювати вручну: «Улюблені», «Переглянуті» та «Переглянути пізніше».
11. Профіль – інформація про користувача: ім’я, аватар, його активність.
12. TMDb API – зовнішній програмний інтерфейс для отримання актуальних даних про фільми.

13. Firebase – хмарна платформа, що використовується для зберігання відгуків, даних користувачів та авторизації.

Формалізація термінів системи дозволяє чітко визначити поняття, що використовуються в межах проєкту, та забезпечити єдине тлумачення функціональних елементів застосунку. Словник системи став основою для подальшого моделювання структури даних, логіки взаємодії між сутностями та користувачами, а також проєктування інтерфейсів. Завдяки цьому етапу було знижено ризики неоднозначностей у розробці та підвищено узгодженість між технічною реалізацією та логікою предметної області.

У цьому розділі було проведено всебічний аналіз предметної області, пов’язаної з розробкою мобільного застосунку для рекомендацій фільмів. Здійснено дослідження актуального стану ринку мобільних рекомендаційних систем, виявлено основні проблеми, з якими стикаються користувачі — зокрема, інформаційне перевантаження, недостатня персоналізація та складність у навігації.

На основі аналізу сформульовано мету дипломного проєкту та поставлено конкретні задачі, що охоплюють архітектуру системи, функціональність, інтерфейсну частину, механізми рекомендацій та безпеку. Окреслено функціональні та нефункціональні вимоги до системи, враховуючи особливості взаємодії авторизованих і неавторизованих користувачів.

Особливу увагу приділено вимогам до інтерфейсу та захисту даних, що є критично важливими для користувацького досвіду та довіри до системи. Було обґрунтовано вибір інструментів розробки та методології, що дозволили забезпечити гнучкість, масштабованість та ефективність проєкту.

У рамках розділу також було побудовано діаграми варіантів використання, визначено типових акторів, описано можливі сценарії взаємодії із застосунком, виявлено основні класи системи та їх зв’язки, а також сформовано словник ключових термінів. Це створює надійну основу для подальшого етапу — проєктування структури системи та реалізації її функціональних компонентів.

2 ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Визначення класів системи

У процесі проєктування мобільного застосунку FilmBuddy одним з найважливіших етапів є побудова об'єктної моделі, яка відображає внутрішню логіку системи та дозволяє ефективно реалізовувати функціональні вимоги. Для цього застосовано принципи об'єктно-орієнтованого програмування (ООП), що базуються на таких ключових поняттях, як інкапсуляція, модульність, повторне використання коду та слабка зв’язаність між компонентами.

Класи є основою структури системи, оскільки кожен з них описує окрему сутність предметної області — користувача, фільм, відгук, оцінку тощо. Окрім структури даних (атрибутів), класи містять методи, які реалізують логіку взаємодії користувачів із системою. Завдяки правильно спроєктованим класам можна забезпечити масштабованість застосунку, підтримку багатьох сценаріїв використання та зручність супроводу.

Клас User описує зареєстрованого користувача мобільного застосунку. Кожен користувач має унікальний ідентифікатор (userId), ім’я для відображення (displayName), електронну пошту (email) та зображення профілю (photoURL). Саме цей клас є точкою входу в систему, оскільки містить усі необхідні атрибути для авторизації та взаємодії користувача з іншими сутностями. Користувач має змогу реєструватися, входити в систему, оновлювати свій профіль, а також залишати відгуки, ставити оцінки, коментувати і додавати фільми до персональних списків. На рисунку 2.1 представлено клас User.

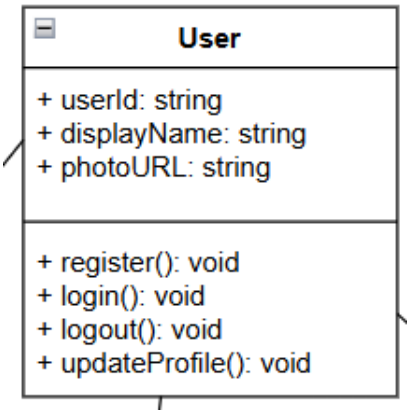

Рисунок 2.1 – клас User

У таблиці 2.1 наведено методи класу User та їх функціональне призначення.

Таблиця 2.1 – методи класу User

Метод	Роль методу
register()	Створює нового користувача, зберігаючи email, ім’я та пароль у базі даних.
login()	Перевіряє правильність введених даних і дозволяє увійти до системи.
logout()	Забезпечує вихід користувача із системи.
updateProfile()	Дозволяє змінити ім’я або фото профілю після реєстрації.

Клас Movie представляє фільм у системі. Він містить такі атрибути, як назва, опис, дата виходу, постер та рейтинг. Фільми пов’язуються із жанрами, відгуками та персональними списками користувача. На рисунку 2.2 представлено клас Movie.

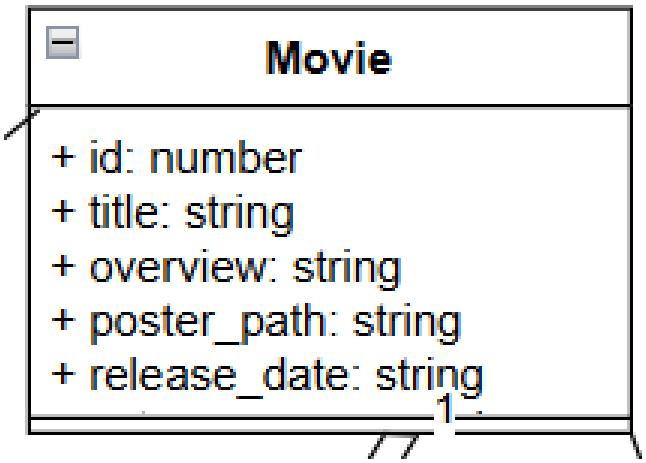

Рисунок 2.2 – клас Movie

Клас Genre класифікує фільми за жанрами. Він містить назву та ідентифікатор жанру. Фільм може належати до кількох жанрів, а жанр — містити багато фільмів. На рисунку 2.3 представлено клас Genre

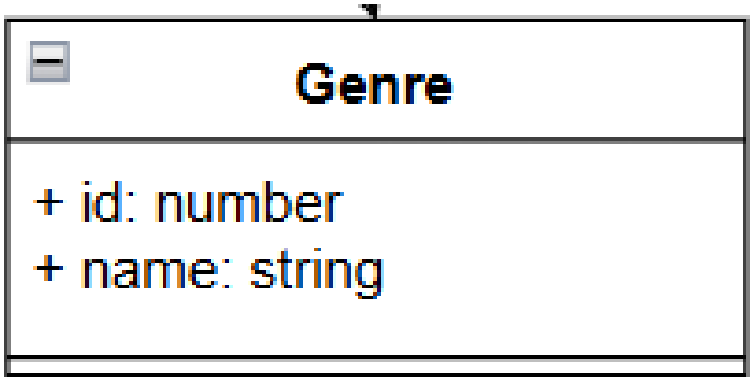

Рисунок 2.3 – клас Genre

Клас Review реалізує відгуки до фільмів. Він зберігає текст, автора, дату створення, кількість лайків і дизлайків, а також зв’язок із фільмом. Відгук може мати вкладені відповіді. На рисунку 2.4 представлено клас Review.

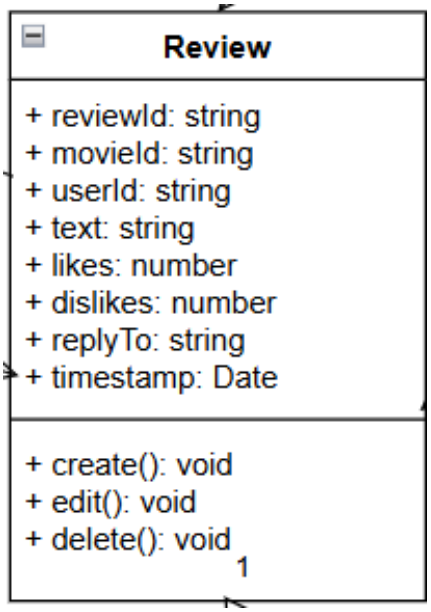

Рисунок 2.4 – клас Review

У таблиці 2.2 наведено методи класу Review та їх функціональне призначення.

Таблиця 2.2 – методи класу Review

Метод	Роль методу
create()	Створює новий відгук.
edit()	Редагує існуючий відгук.
delete()	Видаляє відгук.

Клас Comment зберігає відповіді на відгуки. Кожен коментар містить текст, дату створення та зв’язок із автором. Коментар є частиною відгуку й не існує окремо. На рисунку 2.5 представлено клас Comment.

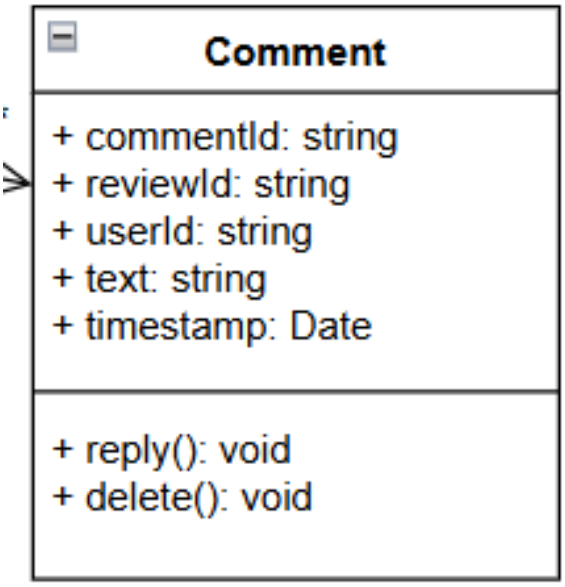

Рисунок 2.5 – клас Comment

У таблиці 2.3 наведено методи класу Comment та їх функціональне призначення.

Таблиця 2.3 – методи класу Comment

Метод	Роль методу
reply()	Додає відповідь на відгук.
delete()	Видаляє коментар.

Клас Rating зберігає оцінки користувачів до відгуків. Оцінка може бути позитивною або негативною. Користувач може оцінити кожен відгук лише один раз. На рисунку 2.6 представлено клас Rating.

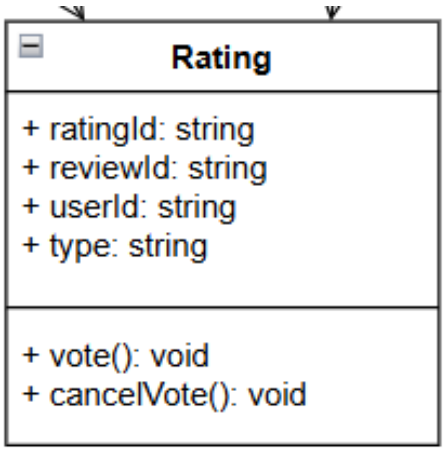

Рисунок 2.6 – Клас Rating

У таблиці 2.4 наведено методи класу Rating та їх функціональне призначення.

Таблиця 2.4 – методи класу Rating

Метод	Роль методу
vote()	Додає оцінку до відгуку.
cancelVote()	Скасовує попередню оцінку.

Клас MovieListItem є проміжною сутністю, яка зв’язує фільми з персональними списками користувача. Кожен запис містить тип списку (наприклад, улюблені чи переглянуті) і дату додавання. На рисунку 2.7 представлено клас MovieListItem

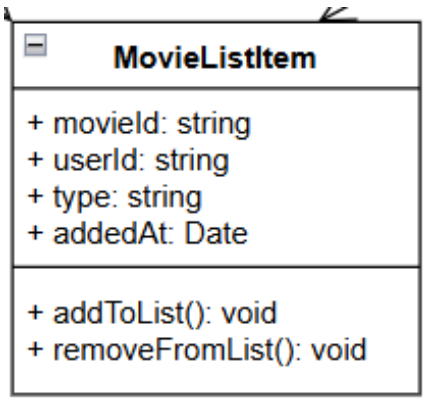

Рисунок 2.7 – Клас MovieListItem

У таблиці 2.5 наведено методи класу MovieListItem та їх функціональне призначення.

Таблиця 2.5 – методи класу MovieListItem

Метод	Роль методу
addToList()	Додає фільм до списку.
removeFromList()	Видаляє фільм зі списку.

На основі наведених вище описів було побудовано діаграму класів системи, яка відображає усі основні сутності, їх атрибути, методи та зв’язки між ними. Ця діаграма демонструє структуру програмного забезпечення FilmBuddy з точки зору об’єктно-орієнтованої моделі. На рисунку 2.8 представлено діаграму класів.

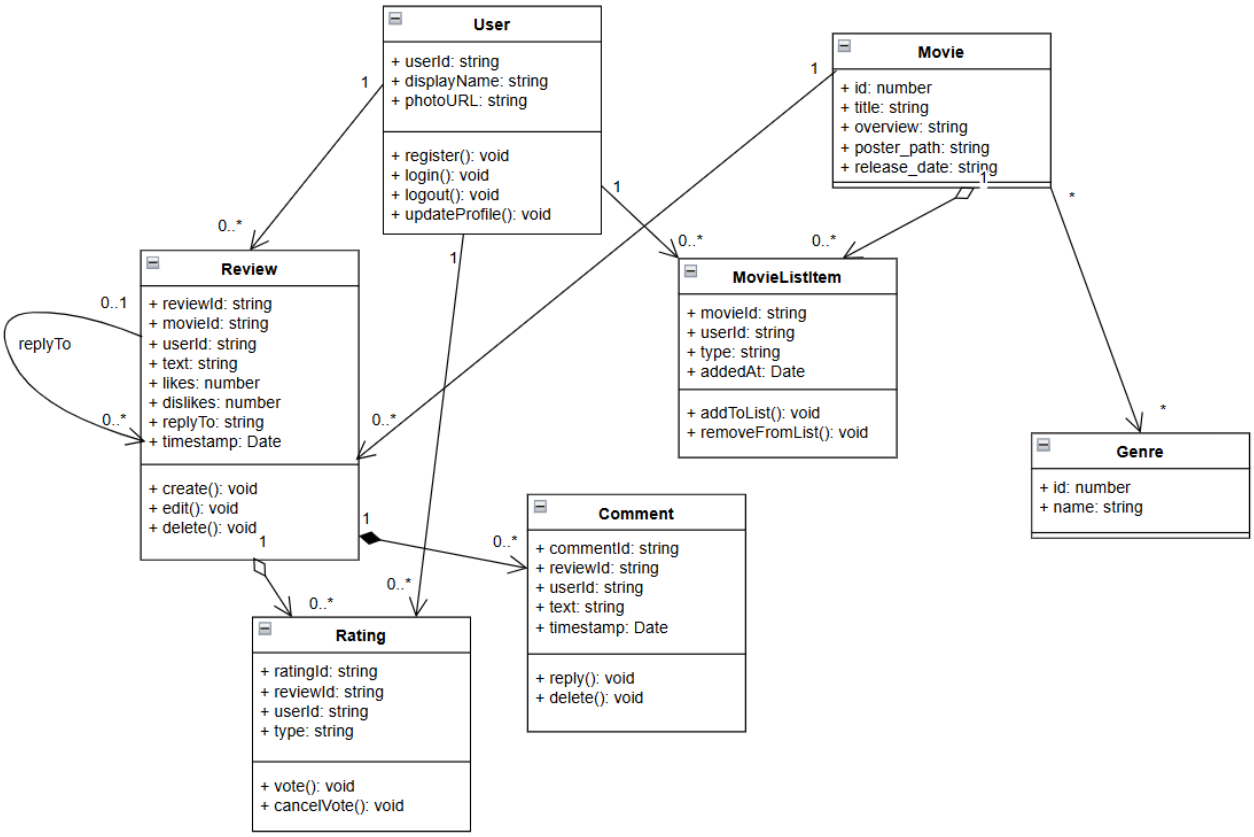

Рисунок 2.8 – Діаграма Класів

На діаграмі класів відображено наступні ключові зв’язки:

Клас User має асоціативні зв’язки з класами Review, Rating, Comment та MovieListItem. Це означає, що один користувач може створити багато відгуків, залишити багато оцінок та коментарів, а також додати декілька фільмів до власних списків.

Клас Movie пов’язаний з класами Review, MovieListItem та Genre. Один фільм може бути доданий до списків різних користувачів, отримати кілька відгуків, а також належати до одного або кількох жанрів. Зв’язок між Movie і Genre реалізований як багато-до-багатьох.

Клас Review має композиційний зв’язок із класом Comment, що означає, що коментарі не можуть існувати без основного відгуку. Аналогічно, Review пов'язаний із Rating, що забезпечує можливість оцінки відгуків. Також реалізовано рекурсивний зв’язок replyTo — відгук може бути відповіддю на інший відгук.

Клас MovieListItem є проміжною сутністю між User і Movie і реалізує агрегаційні зв’язки з обома. Це дозволяє гнучко формувати персональні списки фільмів.

Загалом, кожен клас має чітко визначені зв’язки, які не лише логічно відображають взаємодію між об’єктами, а й забезпечують правильну поведінку системи відповідно до функціональних вимог.

Отже, на основі аналізу вимог до системи було розроблено повну структуру об’єктної моделі у вигляді системи взаємопов’язаних класів. Ця модель охоплює усі основні функціональні компоненти застосунку FilmBuddy та слугує основою для подальшої реалізації бізнес-логіки. Побудована діаграма класів забезпечує зручне візуальне уявлення архітектури програмного забезпечення та стане орієнтиром у наступних етапах розробки.

2.2 Моделювання архітектури системи

Архітектура програмного забезпечення визначає загальну організацію системи, включаючи її основні компоненти, взаємозв’язки між ними, способи обміну даними, а також технології, які використовуються для реалізації кожного рівня. У процесі моделювання архітектури системи FilmBuddy було обрано клієнт-серверну архітектуру з централізованим зберіганням даних у хмарній базі Firestore, а також використанням React Native для побудови клієнтської частини мобільного застосунку.

Загальна структура системи складається з трьох логічних рівнів: рівня представлення (інтерфейсу користувача), рівня логіки (виконання бізнес-функцій) та рівня доступу до даних (збереження та отримання інформації з бази даних). Рівень представлення реалізовано за допомогою React Native, що дозволяє створювати кросплатформений мобільний застосунок. Логіка взаємодії між екраном та внутрішніми модулями реалізована через функціональні компоненти та хуки. Рівень даних реалізовано за допомогою Firebase Firestore — NoSQL хмарної бази даних, яка зберігає структуру даних у вигляді колекцій та документів.

Користувач взаємодіє із застосунком через інтерфейс (наприклад, переглядає фільми, залишає відгуки), і ці дії ініціюють виклики функцій, які надсилають або отримують дані з Firestore. Наприклад, при авторизації користувача використовуються сервіси Firebase Authentication, а відгуки зберігаються в окремій колекції бази даних. Уся взаємодія з серверною частиною є асинхронною, що забезпечує плавну роботу застосунку.

На рисунку 2.9 подано загальну архітектурну модель застосунку FilmBuddy, яка включає основні компоненти та канали обміну інформацією між ними.

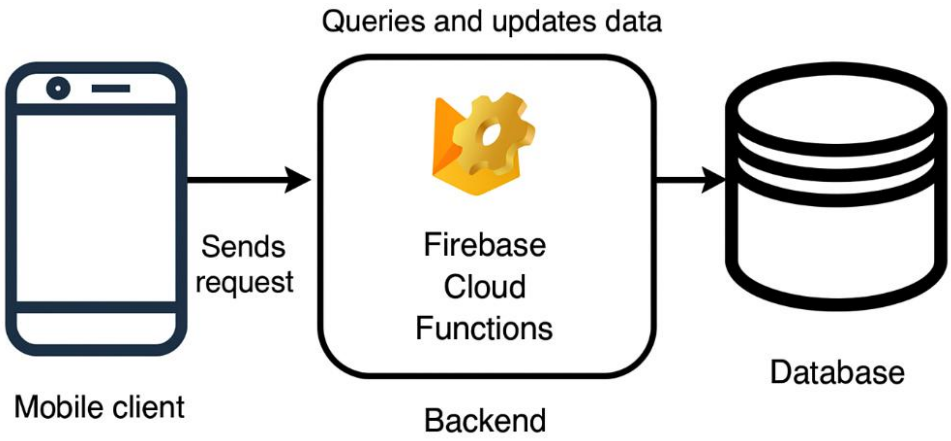

Рисунок 2.9 – Загальна архітектурна модель

2.3 Розробка мобільного застосунку

Розробка мобільного застосунку FilmBuddy охоплює всі етапи життєвого циклу програмного забезпечення – від створення архітектури та структури проєкту до реалізації функціональності, інтеграції з зовнішніми API та базою даних, а також тестування, налагодження та підготовки до релізу. Основною метою є створення сучасного, інтуїтивно зрозумілого та кросплатформенного рішення, яке дозволяє користувачеві шукати фільми, переглядати рекомендації, залишати відгуки, оцінювати контент та керувати персональними списками.

Застосунок розроблявся з використанням фреймворку React Native, який дозволяє створювати кросплатворменні мобільні застосунки для Android та IOS із використанням JavaScript або TypeScript. Для полегшення запуску й розгортання було обрано Expo - набір інструментів, що оптимізує розробку, дозволяє працювати безпосередньо на фізичному пристрої та забезпечує інтеграцію з пакунками для доступу до камери, галереї, push-сповіщень тощо.

На першому етапі проєкт було створено за допомогою CLI-команди зображено на рисунку 2.10.

```
npx create-expo-app filmbuddy
```

Рисунок 2.10 – команда для створеня проєкту

Ця команда створює базову структуру каталогу з налаштуваннями expo, App.js, assets/, screens/, а також файлом конфігурації app.json. Таким чином, рощробка застосунку не потребувала ручного налаштування середовища, SDK або конфігурацій для різних платформ – Expo бере це на себе.

Одразу після ініціалізації було встановлено ключові залежності: react-navigation(для навігації між екранами), axios(для роботи з API), firebase(для з'єднання з хмарними сервісами) та react-native-paper(для візуальних компонентів). Усі бібліотеки були вибрані з урахуванням стабільності, популярності та підтримки сучасних практик розробки.

Важливою частиною будь-якого мобільного застосунку є зручна навігація між екранами. У FilmBuddy для швидкого перемикання між головними розділами (головна сторінка, пошук та профіль) реальзовано нижню панель навігації з використанням компоненти Tabs з бібліотеки expo-router. Цей підхід дозволяє забезпечити постійну присутність вкладок у нижній частині екрана та зручну навігацію між ними без втрати контексту.

У головному файлі _layout.tsx реалізовано базову навігацію зображено на рисунку 2.11.

```
import { Ionicons } from "@expo/vector-icons";
import { Tabs } from "expo-router";

export default function TabsLayout() {
  return (
    <Tabs
      screenOptions={({ route }) => ({
        headerShown: false,
        tabBarStyle: {
            backgroundColor: "#121212",
            borderTopColor: "#7B61FF",
          },
          tabBarShowLabel: false,
        tabBarIcon: ({ focused }) => {
          let iconName: keyof typeof Ionicons.glyphMap = "ellipse-outline";

          if (route.name === "home") iconName = focused ? "home" : "home-outline";
          else if (route.name === "search") iconName = focused ? "search" : "search-outline";
          else if (route.name === "profile") iconName = focused ? "person" : "person-outline";

          return <Ionicons name={iconName} size={26} color={focused ? "#7B61FF" : "#999"} />;
        },
      })}
    />
  );
}
```

Рисунок 2.11 – Реалізована навігація застосунка

Такий підхід забезпечує зручну, сучасну та узгоджену навігацію, яка виглядає нативно як на Android, так і на IOS. Завдяки expo-router увесь роутинг організовано декларативно, а кожна вкладка відповідає окремому файлу/папці у структурі проєкту.

Однією з ключових функцій застосунку FilmBuddy є можливість перегляду актуальної інформації про фільми. Для цього було реалізовано інтеграцію з The Movie Database API (TMDb) – одним із найпопулярніших відкритих API для отримання кіно-даних.

На головному екрані (HomeScreen) користувачеві відображаються рекомендовані, популярні фільми, топові, майбутні прем’єри, а також окремо винесений випадковий фільм отримані за запитом до TMDb. Отримання цих даних реалізовано асинхронно за допомогою хуку useEffect, що викликається одразу після завантаження компонента HomeScreen. Код зображено на рисунку 3.12.

```
useEffect(() => {
const unsubscribe = auth.onAuthStateChanged(async (user) => {
  setIsGuest(!user);
  setLoading(true);

  const [top, up, now] = await Promise.all([
    fetchMovies("top_rated"),
    fetchMovies("upcoming"),
    fetchMovies("now_playing"),
  ]);

  setTopRated(top);
  setUpcoming(up);
  setNowPlaying(now);

  let rec = [];
  if (user) {
    rec = await getSmartRecommendedMovies(user.uid);
    console.log("🔒 Авторизований — рекомендацій:", rec.length);
  } else {
    rec = await getGuestTopMovies();
    console.log("👥 Гість — топ:", rec.length);
  }

  setRecommended(rec);
  setLoading(false);
});

return () => unsubscribe(); // clean up
}, []);
```

Рисунок 2.12 – Реалізована отримання фільмів

При завантаженні екрана виконується функція loadAll(), яка отримує доступ до об’єкта currentUser через Firebase Authentication. Якщо користувач не авторизований – запити не відбуваються.

Далі за допомогою Promise.all паралельного викликається функції fetchMovies() з різними параметрами (top_rated, upcoming, now_playing) – це дозволяє значно зекономити час завантаження.

Після цього додатково викликається функція getSmartRecommendedMivies(user.uid), яка формує персоналізовані рекомендації для користувача. Отримані масиви записуються у відовідні useState змінні, які відображаються у списка на головному екрані.

Після того як дані про фільми успішно отримано з TMDb API, вони мають бути виведені на головному екрані у вигляді зручних і стильних горизонтальних списків. Для цього у застосунку реалізований компонент MovieList, який

використовується для відображення будь-якої категорії фільмів – рекомендованих, топових, майбутніх або тих, що йдуть у прокаті. Показано на рисунку 2.13.

```tsx
import React from "react";
import { FlatList } from "react-native";
import MovieCard from "./MovieCard";

interface Movie {
  id: number;
  title: string;
  poster_path: string;
}

interface Props {
  movies: Movie[];
}

const MovieList: React.FC<Props> = ({ movies }) => {
  if (!movies || movies.length === 0) return null;

  return (
    <FlatList
      data={movies}
      keyExtractor={(item) => item.id.toString()}
      horizontal
      renderItem={({ item }) =>
        item ? (
          <MovieCard title={item.title} posterPath={item.poster_path} />
        ) : null
      }
      showsHorizontalScrollIndicator={false}
    />
  );
};

export default MovieList;
```

Рисунок 2.13 – Виведення даних про фільми

Після натискання на будь-яку картку фільму з головного екрана користувач переходить на детальну сторінку фільму, де відображається розширена інформація: постер, назва, жанри, опис, рейтинг, дата релізу, а також елементи взаємодії — кнопки додавання в список, блок відгуків тощо.

Цей екран реалізовано як динамічний маршрут за допомогою expo-router, де кожен фільм має унікальний id. Файл маршруту має вигляд /movie/[id].tsx.

Це дозволяє автоматично зчитувати id фільму з URL і використовувати його для запиту даних (див. рис. 2.14)

```
import { useLocalSearchParams, useRouter } from "expo-router";
const MovieDetails = () => {
  const { id } = useLocalSearchParams<{ id: string }>();
  const router = useRouter();
```

Рисунок 2.14 – Отримання параметра id з динамічного маршруту у expo-router

Для виводу повної інформації використовується окрема функція fetchMovieDetails, яка робить запит до TMDb API(рис. 2.15)

```
//Отримати детальну інформацію про фільм
export const fetchMovieDetails = async (id: number) => {
  const response = await fetch(
    `${BASE_URL}/movie/${id}?api_key=${API_KEY}&language=uk-UA`
  );
  const data = await response.json();
  return data;
};
```

Рисунок 2.15 – Вивід інформації

На сторінці деталізації фільму відображаються наступні елементи:

- Великий постер фільму
- Назва, рік, жанри, середній рейтинг
- Короткий опис (overview)
- Кнопки: "Додати в улюблене", "Додати в переглянуте"
- Компонент відгуків (<ReviewsSection />)

```
return (
  <ScrollView style={styles.container}>
    <View style={styles.headerRow}>
      <TouchableOpacity onPress={goBack} style={styles.backIcon}>
        <Ionicons name="arrow-back" size={24} color="#7B61FF" />
      </TouchableOpacity>
      <Text style={styles.title}>{movie.title}</Text>
    </View>

    <Image
      source={{ uri: `https://image.tmdb.org/t/p/w500${movie.poster_path}` }}
      style={styles.poster}
    />

    <Text style={styles.year}>
      {movie.release_date ? movie.release_date.slice(0, 4) : "–"}
    </Text>

    <Text style={styles.sectionTitle}>Актори</Text>
    <Text style={styles.castList}>
      {cast.map((actor: any) => actor.name).join(", ")}
    </Text>

    <Text style={styles.sectionTitle}>Рейтинг</Text>
    <Text style={styles.rating}>⭐ {movie.vote_average.toFixed(1)} /10</Text>

    <Text style={styles.sectionTitle}>Опис</Text>
    <Text style={styles.overview}>{movie.overview}</Text>

    <Text style={styles.sectionTitle}>Додати в список:</Text>
    <View style={styles.verticalListButtons}>
      <TouchableOpacity style={styles.listItem} onPress={() => handleAddToList("favorites")}>
        <Ionicons name={isInFavorites ? "heart" : "heart-outline"} size={24} color="#7B61FF" />
        <Text style={styles.listText}>Улюблене</Text>
      </TouchableOpacity>

      <TouchableOpacity style={styles.listItem} onPress={() => handleAddToList("watched")}>
        <Ionicons name={isInWatched ? "eye" : "eye-outline"} size={24} color="#7B61FF" />
        <Text style={styles.listText}>Переглянуте</Text>
      </TouchableOpacity>
```

Рисунок 2.16 – Деталізація фільмів

Сторінка деталей фільму є центральним елементом взаємодії користувача із застосунком. Вона дозволяє ознайомитись із контентом, додати фільм у список, оцінити його, переглянути відгуки інших користувачів або залишити власний. Динамічна побудова маршруту спрощує архітектуру та дозволяє швидко переходити між фільмами за ID.

Система відгуків у FilmBuddy дозволяє кожному авторизованому користувачеві залишати власну думку про будь-який фільм, бачити думки інших, реагувати на них (лайк/дизлайк), а також відповідати на коментарі. Всі дані про відгуки зберігаються в хмарній базі даних Firebase Firestore, що дозволяє організувати гнучку, масштабовану й реальновремінну взаємодію між користувачами.

У базі даних створено колекцію reviews, кожен документ якої містить наступні поля:

- movieId — ідентифікатор фільму
- userId — ідентифікатор автора відгуку
- text — текст відгуку
- likes, dislikes — кількість голосів
- timestamp — час створення
- replyTo — якщо це відповідь, зберігається ID батьківського відгуку

Коли користувач залишає відгук, він відправляється у Firestore через функцію(рис.2.17)

```
const handleAddReview = async () => {
  if (!text.trim() || !user) return;

  await addDoc(collection(db, "reviews"), {
    movieId,
    movieTitle,
    text,
    userId: user.uid,
    timestamp: serverTimestamp(),
    likes: 0,
    dislikes: 0,
    replyTo,
  });

  setText("");
  setReplyTo(null);
};
```

Рисунок 2.17 – Додавання нового відгуку у Firestore

На сторінці /movie/[id] рендериться компонент ReviewsSection, який:

- отримує всі відгуки по movieId

- групує відповіді під основними коментарями (як вкладені)
- сортує за датою
- відображає аватар, ім’я, час публікації, текст

Фрагмент логіки отримання зображено на рисунку 2.18

```
<ReviewsSection movieId={id as string} movieTitle={movie.title} />
```

Рисунок 2.18 – Отримання відгуків у режимі реального часу

Кожен відгук виводиться з аватаром, іменем, кнопками лайк/дизлайк і кнопкою “Відповісти”. Якщо replyTo !== null — відгук вважається вкладеним і виводиться з відступом. Лістинг коду зображено на рисунку 2.19.

```
{reviews
  .filter((r) => r.replyTo === item.id)
  .map((reply) =>
    React.createElement(
      View,
      { style: styles.reply, key: String(reply.id) },
      <>
        <View style={styles.userRow}>
          {reply.avatar && (
            <Image source={{ uri: reply.avatar }} style={styles.avatar} />
          )}
          <Text style={styles.user}>{reply.userName}</Text>
        </View>
        <Text style={styles.reviewText}>{reply.text}</Text>
      </>
    )
  )}
```

Рисунок 2.19 – Відображення одного відгуку з вкладеними відповідями

Завдяки Firestore система відгуків працює у реальному часі: доданий коментар з’являється одразу у всіх користувачів. Кожен коментар зберігає зв’язок з автором та фільмом, а також дозволяє створювати вкладені відповіді, будуючи логічну дискусію під фільмом. Компонент ReviewsSection є універсальним і автоматично відображає всі рівні вкладеності відгуків.

Застосунок FilmBuddy дозволяє кожному зареєстрованому користувачу створювати власні списки фільмів: улюблені, переглянуті та подивитися пізніше. Ця функціональність реалізована через збереження відповідної інформації в Firebase Firestore у вкладених колекціях користувача. На сторінці MovieDetails реалізовано функцію handleAddToList, яка перевіряє, чи вже є фільм у списку, і залежно від цього додає його або видаляє. Лістинг коду зображано на 2.20.

```
const handleAddToList = async (listName: "favorites" | "watched" | "watchLater") => {
console.log("⭕ Натиснуто кнопку списку:", listName);

const user = getAuth().currentUser;
console.log("👤 Поточний користувач:", user);

if (!user) {
  console.log("⚠️ Користувач не авторизований — показуємо Alert");

  Alert.alert(
    "Потрібна авторизація",
    "Щоб зберігати фільми у списках, спочатку увійдіть у свій акаунт.",
    [
      { text: "Скасувати", style: "cancel" },
      { text: "Увійти", onPress: () => router.push("/login") },
    ]
  );

  // Якщо на Web — Alert не працює. Використай заміну:
  if (Platform.OS === "web") {
    window.alert("Щоб зберігати фільми, увійдіть у свій акаунт");
  }

  return;
}

if (!id || !movie) {
  console.log("❌ Відсутній ID або дані про фільм");
  return;
}
```

```
const ref = doc(db, "users", user.uid, listName, id.toString());
const snap = await getDoc(ref);

const toggle = {
  favorites: () => setIsInFavorites(!snap.exists()),
  watched: () => setIsInWatched(!snap.exists()),
  watchLater: () => setIsInWatchLater(!snap.exists()),
};

if (snap.exists()) {
  console.log("🗑️ Фільм уже в списку — видаляємо");
  await deleteDoc(ref);
  toggle[listName]();
  Alert.alert("Видалено", "Фільм прибрано зі списку.");
} else {
  console.log("✅ Додаємо фільм до списку:", listName);
  await setDoc(ref, {
    title: movie.title,
    poster: movie.poster_path,
    addedAt: new Date(),
  });
  toggle[listName]();
  Alert.alert("Додано", `Фільм додано до списку ${listName}.`);
}
```

Рисунок 2.20 – Логіка перемикання фільму у списках користувача

Кожен список є окремою колекцією всередині документа користувача в Firestore, що дозволяє зберігати пов’язані фільми без дублювання даних. Структура users/{userId}/favorites/{movieId} забезпечує просту фільтрацію та швидкий доступ до персоналізованих даних.

У застосунку FilmBuddy реалізовано глобальне увімкнення свайп-навігації назад для всіх екранів, що входять до стек-навігатора. Це зроблено через компонент Layout, у якому використовується Stack з expo-router, де параметр gestureEnabled: true встановлено на рівні screenOptions. Лістинг коду зображено на 2.21.

```
import { Stack } from "expo-router";

export default function Layout() {
  return (
    <Stack
      screenOptions={{
        gestureEnabled: true,
        headerShown: false,
      }}
    />
  );
}
```

Рисунок 2.21 – Встановлення свайп-навігації назад для всіх екранів застосунку

Централізоване керування навігаційними опціями робить структуру застосунку чистою та гнучкою. Завдяки глобальному налаштуванню користувач може інтуїтивно використовувати свайп для повернення, що відповідає сучасним стандартам мобільного UX.

Однією з основних функцій застосунку FilmBuddy є система персоналізованих рекомендацій, яка формує добірку фільмів відповідно до індивідуальних уподобань кожного користувача. Цей функціонал спрямований на те, щоб користувач отримував фільми, які йому справді цікаві, без необхідності тривалого пошуку.

Після входу в застосунок система автоматично аналізує фільми, які користувач раніше додав до списків «улюблене» та «переглянуте». На основі цих даних обчислюються найбільш популярні жанри, актори та режисери серед збережених фільмів. Ці характеристики формують «вектор інтересів» користувача. Далі відбувається звернення до API TMDb з параметрами, які враховують найпоширеніші жанри, акторів і режисерів у вподобаннях користувача. У відповідь застосунок отримує список фільмів, які потенційно відповідають смакам цього користувача.

Рекомендовані фільми додатково фільтруються — з кінцевої вибірки автоматично виключаються ті, які вже додані користувачем до будь-якого зі списків. Завдяки цьому користувач бачить лише нові варіанти, з якими ще не взаємодіяв. Уся ця логіка реалізована у функції getSmartRecommendedMovies, яка обробляє дані з Firebase Firestore, формує запити до TMDb і повертає оптимізовану добірку (див. рис. 2.22).

Для неавторизованих користувачів також передбачено функцію рекомендацій: на головному екрані виводиться розділ з 20 найкращими фільмами за рейтингом TMDb (див.рис. 2.23). Це дозволяє навіть гостьовим користувачам одразу ознайомитися з якісним контентом і мотивує створити обліковий запис для доступу до персоналізованих функцій.

Таким чином, система рекомендацій FilmBuddy адаптується до стилю перегляду кожного користувача, пропонує нові варіанти на основі попередніх вподобань і підвищує зручність взаємодії з додатком як для зареєстрованих, так і для гостьових користувачів.

```
export const getSmartRecommendedMovies = async (userId: string): Promise<any[]> => {
  const watched = await getUserList(userId, "watched");
  const watchedIds = new Set(watched.map((m: any) => Number(m.id)));

  const detailed = await Promise.all(watched.map((m: any) => fetchMovieDetails(Number(m.id))));
  if (!detailed.length) return [];

  const userVector: number[] = detailed
    .reduce((acc: number[], movie: any) => {
      const vec = getMovieVector(movie);
      return acc.map((v: number, i: number) => v + vec[i]);
    }, Array(22).fill(0))
    .map((sum: number) => sum / detailed.length);

  const allPopular: any[] = [];
  for (let page = 1; page <= 3; page++) {
    const res = await fetchMovies("popular", page);
    allPopular.push(...res);
  }

  const scored = allPopular
    .filter((movie: any) => movie.poster_path && !watchedIds.has(movie.id))
    .map((movie: any) => {
      const vec = getMovieVector(movie);
      const score = cosineSimilarity(userVector, vec);
      return { ...movie, score };
    });

  // фільтруємо унікальні id
  const uniqueMap = new Map();
  scored.forEach((m) => {
    if (!uniqueMap.has(m.id)) uniqueMap.set(m.id, m);
  });

  return Array.from(uniqueMap.values())
    .sort((a: any, b: any) => b.score - a.score)
    .slice(0, 20);
}
```

Рисунок 2.22 – Алгоритм побудови персональних рекомендацій

```
export const getGuestTopMovies = async (): Promise<any[]> => {
  const popular = await fetchMovies("popular");
  return popular.slice(0, 20);
};
```

Рисунок 2.23 – Топ-20 фільмів для неавторизованих користувачів

2.4 Структура база даних Firestore

У застосунку FilmBuddy зберігання всіх динамічних даних реалізовано за допомогою хмарної бази Firebase Firestore. Це документо-орієнтована NoSQL база даних, яка дозволяє працювати з гнучкою схемою даних у реальному часі. Вона ідеально підходить для мобільних застосунків завдяки підтримці offline-режиму, масштабованості та автоматичній синхронізації.

У проєкті FilmBuddy реалізовано чітку логічну структуру колекцій, що відповідає ключовим функціональним блокам застосунку: збереження користувачів, відгуків, голосів, а також персональних списків фільмів.

Колекція users містить документи, ідентифіковані за UID кожного користувача. У кожному такому документі є вкладені підколекції favorites, watched та watchLater, які містять окремі фільми, додані користувачем до відповідних списків. Це дозволяє ефективно структурувати дані без дублювання й забезпечити швидкий доступ до персонального контенту кожного користувача.(Див. рис. 2.24)

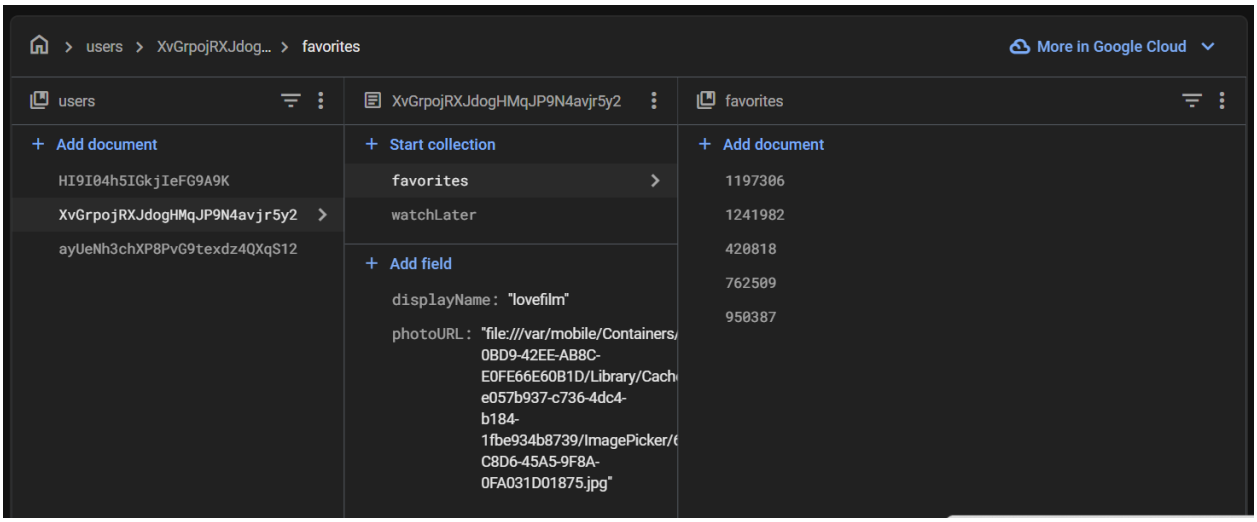

Рисунок 2.24 – Структура даних у Firebase Firestore для застосунку FilmBuddy

Колекція reviews є глобальною для всіх користувачів і фільмів. Кожен документ у ній містить текст відгуку, посилання на користувача (userId) і фільм (movieId), дату створення, а також поля для зберігання кількості реакцій (likes, dislikes). Якщо це відповідь на інший коментар — використовується поле replyTo, що зберігає ID батьківського відгуку (див. рис. 2.25).

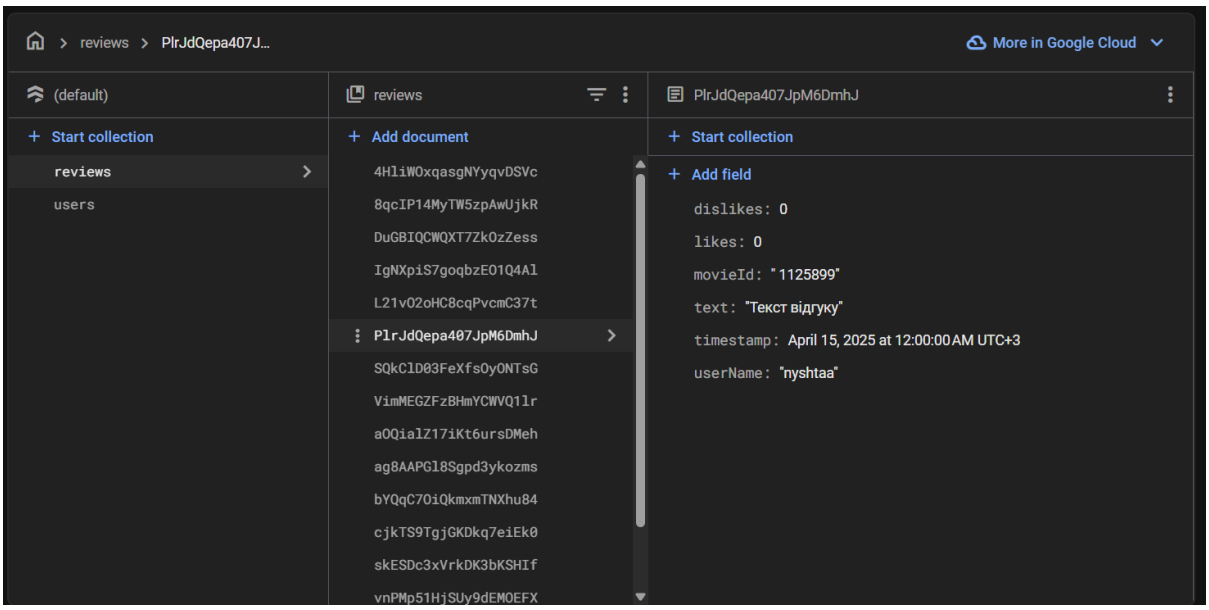

Рисунок 2.25 – Структура даних у Firebase Firestore для застосунку FilmBuddy

Завдяки такій структурі забезпечується гнучкість, масштабованість і розділення відповідальностей між користувачами, контентом і взаємодією. Крім того, вкладені підколекції дозволяють легко працювати з фільтрами, наприклад: "всі улюблені фільми певного користувача" або "всі відгуки до конкретного фільму".

У даному розділі було повністю реалізовано логіку, інтерфейс та архітектуру мобільного застосунку FilmBuddy. Здійснено аналіз функціональних вимог і визначено основні класи системи, на основі яких побудовано UML-діаграму, що описує взаємозв’язки між ключовими компонентами. Для навігації в застосунку обрано стекову та таб-навігацію з використанням react-navigation та expo-router, що забезпечує зручну побудову маршрутів і взаємодію з динамічними сторінками.

Було реалізовано повноцінний головний екран, який виводить фільми у різних категоріях (топові, новинки, персональні рекомендації), а також забезпечує перехід на сторінку конкретного фільму. Головна функція застосунку — персоналізована рекомендація — реалізована на основі зібраних користувацьких даних, аналізу жанрових переваг та інтеграції з TMDb API. Вона адаптується до інтересів кожного користувача й оновлюється в реальному часі. Сторінка фільму

містить повну інформацію про стрічку, а також компоненти для взаємодії: додавання у списки, переглянуте, улюблене та відгуки. Всі ці дії синхронізуються з базою даних Firestore, що забезпечує постійне збереження персональної інформації. Компонент ReviewsSection дає змогу залишати коментарі, відповідати на них, ставити лайки або дизлайки, а також бачити всі відгуки з вкладеною структурою. Навігація назад реалізована за допомогою свайпу, що підвищує зручність інтерфейсу. У Firebase Firestore сформовано гнучку та зрозумілу структуру з основними колекціями: users, reviews, вкладеними списками фільмів і структурованим збереженням взаємодій. Весь код компоновано у вигляді модулів і функціональних сервісів, що забезпечує повторне використання логіки, простоту масштабування і розширення функціональності в майбутньому.

Таким чином, реалізація мобільного застосунку FilmBuddy є результатом повного циклу розробки — від архітектури до взаємодії з кінцевим користувачем. Розроблена система відповідає сучасним вимогам до мобільного UI/UX та функціональності.

3 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Тестування інтерфейсу

Тестування інтерфейсу користувача (UI) є ключовим етапом забезпечення якості мобільного застосунку. Воно дозволяє виявити та усунути недоліки, що можуть вплинути на зручність використання та загальне враження користувача від продукту. Згідно з рекомендаціями компанії Wezom, ефективне UI-тестування охоплює перевірку таких аспектів, як візуальний дизайн, продуктивність, ієрархія контенту, орієнтація на користувача та безпека [12].

У процесі тестування інтерфейсу мобільного застосунку FilmBuddy особливу увагу було приділено трьом основним екранам: «Головна», «Пошук» та «Профіль».

Екран «Головна» є стартовою точкою взаємодії користувача із застосунком. Він містить динамічні добірки фільмів — рекомендовані, популярні, очікувані новинки та топ за рейтингом. У верхній частині екрана розміщена кнопка для перегляду випадкового фільму, яка відкриває модальне вікно з постером, назвою і можливістю перегляду деталей. Кожна категорія фільмів представлена горизонтальною стрічкою карток, що підтримують свайп і дають змогу швидко перейти до перегляду. Тестування цього екрана включало перевірку адаптивності інтерфейсу до різних розмірів екранів, коректності відображення елементів та зручності навігації (див. рис. 3.26).

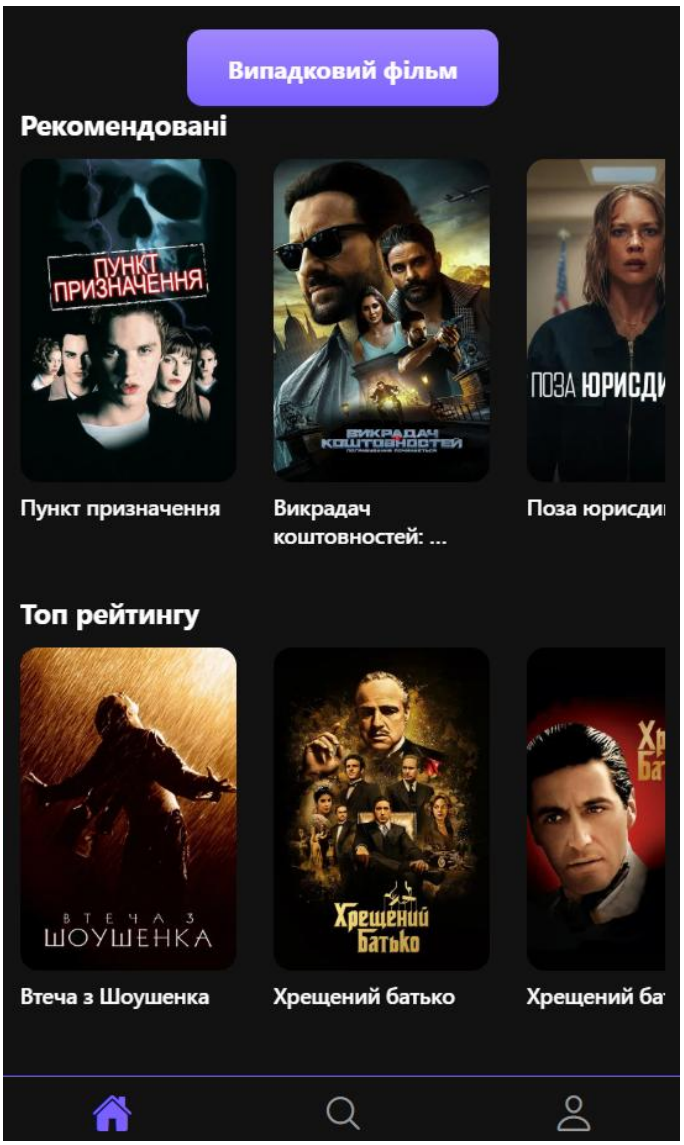

Рисунок 3.1 – Головний екран застосунку

Екран «Пошук» дозволяє користувачеві знайти потрібний фільм за назвою або переглянути добірки за жанрами. У верхній частині розташоване поле введення з темною стилізацією та м’якими межами. У процесі введення запиту застосовується затримка перед надсиланням запиту до API (debounce), що покращує зручність. Результати пошуку виводяться у вигляді списку з компактними картками. Якщо поле пошуку порожнє, користувачеві пропонуються добірки фільмів за жанрами. Тестування цього екрана охоплювало перевірку швидкості реакції на дії, стабільності верстки при виведенні великої кількості жанрів та адекватної поведінки на пристроях із різним співвідношенням сторін (див. рис. 3.27).

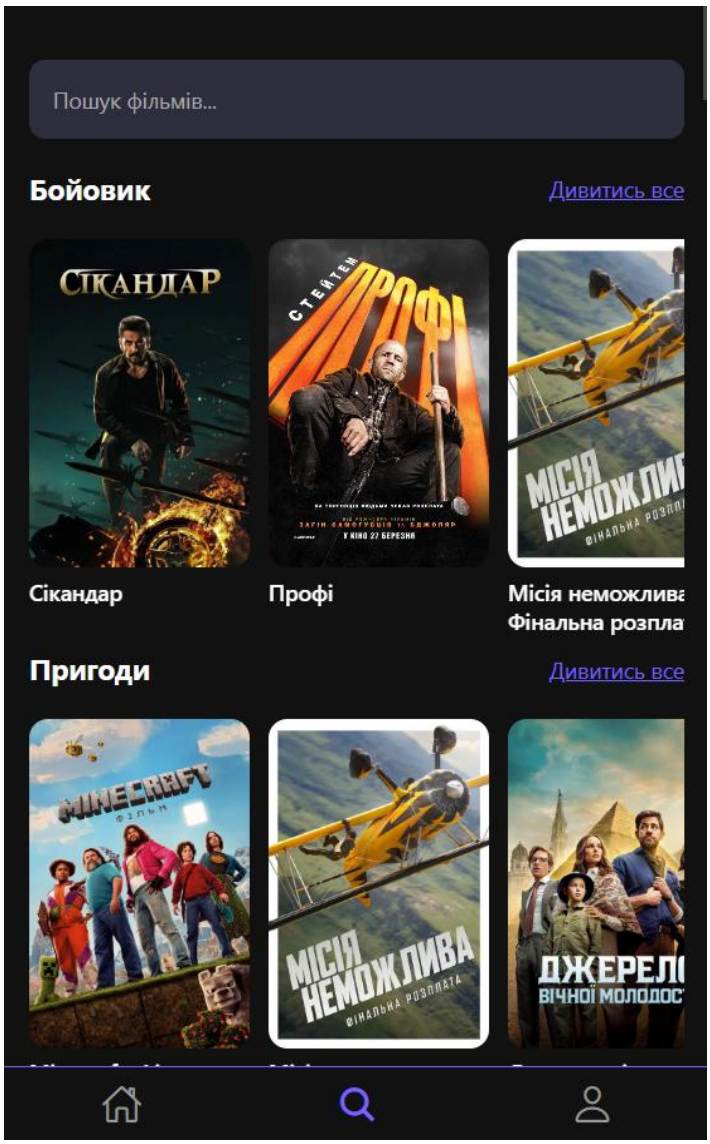

Рисунок 3.2 – Екран пошуку

Екран «Профіль» надає користувачеві доступ до особистої інформації — ім’я, аватар, електронна пошта, а також до функціональних списків: улюблені фільми, переглянуті, ті, які хоче переглянути пізніше, і написані ним відгуки. Усі розділи розташовані у вигляді кнопок або блоків, які ведуть на відповідні екрани. Інтерфейс витриманий у мінімалістичному стилі, з чіткими назвами та приємними інтервалами між елементами. Тестування цього екрана включало перевірку якості виводу аватарів, зручності розташування інформації та передбачуваності навігації (див. рис. 3.28).

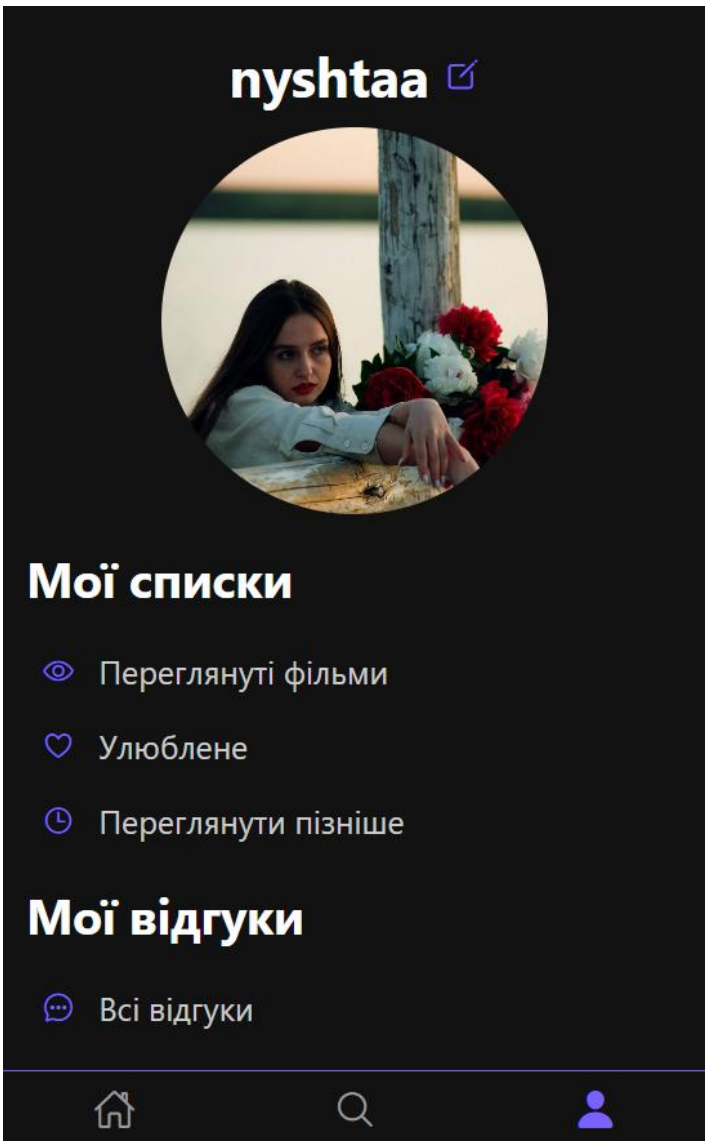

Рисунок 3.3 – Екран профілю

Загалом, результати тестування інтерфейсу FilmBuddy підтвердили, що структура застосунку побудована логічно, візуальні компоненти працюють стабільно, а інтерфейс є інтуїтивно зрозумілим і привабливим для користувача. Це створює позитивний користувацький досвід і забезпечує легкість взаємодії з усіма основними можливостями системи.

3.2 Функціональне тестування

Функціональне тестування — це процес перевірки того, як програмне забезпечення виконує свої основні функції відповідно до встановлених вимог. Метою цього виду тестування є впевнитися, що кожен елемент інтерфейсу та логіки працює так, як очікується користувачем, а система реагує коректно на дії. Згідно з класифікацією DAN.IT, функціональне тестування охоплює не лише окремі модулі, а й їхню взаємодію, перевірку на помилки, правильну обробку даних, навігацію та сценарії використання в реальних умовах [13]. У межах дипломної роботи проведено всебічну перевірку основних сценаріїв користування мобільним застосунком FilmBuddy, включно з реєстрацією, входом, переглядом фільмів, додаванням у списки, пошуком, написанням і оцінюванням відгуків, а також обробкою обмежень для неавторизованих користувачів.

1. Відображення головної сторінки для неавторизованого користувача

Передумови:

Користувач ще не створив обліковий запис і не входив у систему.

Кроки виконання:

– Встановити та відкрити застосунок FilmBuddy.

– Дочекатися автоматичного переходу на головну сторінку (див. рисунок 3.29), яка відкривається для гостей.

– Ознайомитися зі списками фільмів, що відображаються без входу в акаунт.

Очікуваний результат:

На головній сторінці буде відображено добірку з 20 найпопулярніших фільмів за даними TMDb. Кожен фільм представлений у вигляді картки з постером та назвою. Всі картки є клікабельними — натискання на фільм відкриває сторінку з його детальною інформацією. Усі інші функції (додавання у списки, відгуки, рекомендації) будуть недоступні, і при спробі використати їх система запропонує увійти в обліковий запис. При цьому інтерфейс зберігає

повну працездатність: працює кнопка випадкового фільму, навігація, перегляд постерів, прокрутка та відкриття деталей фільмів.

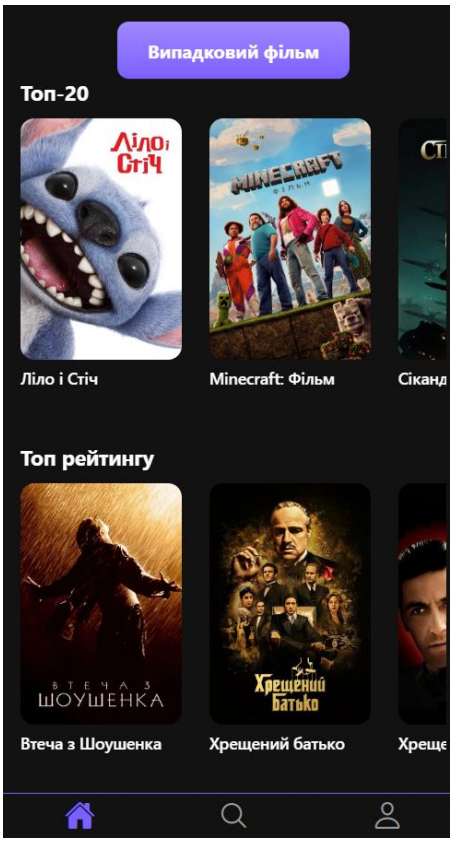

Рисунок 3.4 – Головна сторінка для неавторизованого користувача

2. Відображення головної сторінки для авторизованого користувача

Передумови:

Користувач має обліковий запис і виконав вхід у систему.

Кроки виконання:

– Увійти в застосунок FilmBuddy, використовуючи зареєстрований email і пароль (див. рисунок 3.31).

– Після успішного входу користувач автоматично потрапляє на головну сторінку (див. рисунок 3.30).

– Переглянути усі розділи, що динамічно відображаються на головному екрані.

Очікуваний результат:

На головній сторінці авторизованого користувача відображаються персоналізовані рекомендації, сформовані на основі збережених у Firestore списків «Улюблене» та «Переглянуте». Рекомендовані фільми мають власну секцію зверху. Додатково на екрані присутні категорії: «Топ рейтингу», «Очікувані», «Нові релізи», кожна з яких представлена горизонтальним списком карток. У верхній частині екрана також доступна кнопка « Випадковий фільм», яка відкриває модальне вікно з обраним фільмом і кнопками для його перегляду чи оновлення вибору. Всі елементи інтерфейсу активні: натискання на картку веде на сторінку фільму, а при додаванні фільму до списку статус кнопки змінюється без перезавантаження екрана. Відображення відбувається без помилок, з плавною анімацією та чіткою структурою контенту.

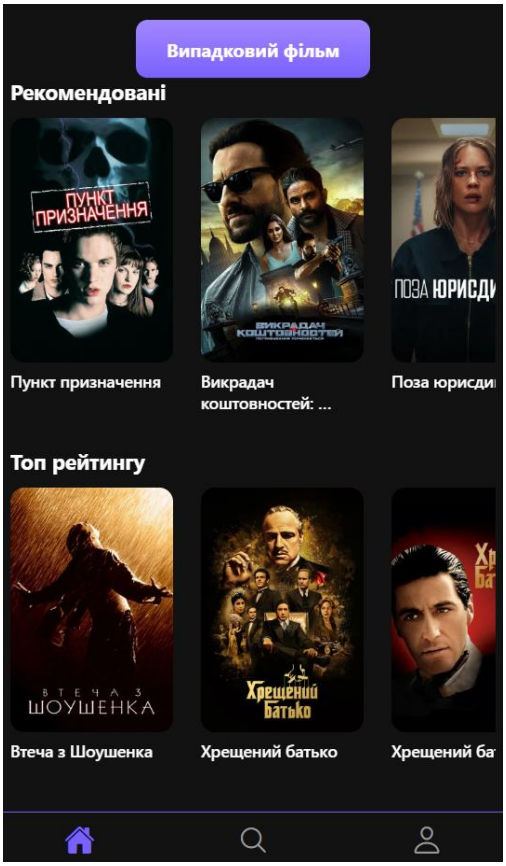

Рисунок 3.5 – Головна сторінка для авторизованого користувача

3. Вибір випадкового фільму

Передумови:

Користувач знаходиться на головній сторінці застосунку. Вхід у систему не є обов’язковим.

Кроки виконання:

– Відкрити застосунок FilmBuddy.

– На головному екрані знайти кнопку «Випадковий фільм» у верхній частині сторінки (див. рисунок 3.30).

– Натиснути на кнопку.

– Дочекатися відкриття модального вікна з випадково обраним фільмом (див. рисунок 3.32).

– Перевірити роботу кнопок «Спробувати ще раз» та «Переглянути фільм».

Очікуваний результат:

Після натискання на кнопку «Випадковий фільм» відкривається модальне вікно з інформацією про випадково обраний фільм зі списку популярних. Вікно містить назву фільму, постер, кнопку для перегляду деталей і кнопку «Спробувати ще раз». Натискання кнопки «Спробувати ще раз» змінює фільм на інший. Натискання кнопки «Переглянути фільм» веде користувача на сторінку з детальною інформацією про вибраний фільм. Усі переходи відпрацьовують коректно, модальне вікно відкривається плавно, не викликає помилок і правильно закривається. Функція працює однаково для авторизованих і неавторизованих користувачів.

Рисунок 3.6 – Модальне вікно з випадковим фільмом

4. Перегляд деталей фільму (неавторизований користувач)

Передумови:

Користувач не входив у систему, переглядає застосунок як гість.

Кроки виконання:

– Відкрити застосунок FilmBuddy.

– На головній сторінці, доступній без авторизації (див. рисунок 3.30), натиснути на будь-яку картку фільму.

– Зачекати на завантаження сторінки деталей фільму (див. рисунок 3.33).

Очікуваний результат:

Відкриється сторінка фільму з інформацією: постер, назва, жанри, рік виходу, короткий опис, рейтинг та список акторів. Усі ці дані доступні для перегляду навіть без входу в акаунт. Кнопки взаємодії («Улюблене», «Переглянуте», «Переглянути пізніше», «Залишити відгук») будуть неактивними або при спробі натискання викликатимуть повідомлення з проханням

авторизуватись. Таким чином, гість має обмежений доступ до функціональності, але може вільно ознайомлюватись зі змістом і деталями фільмів.

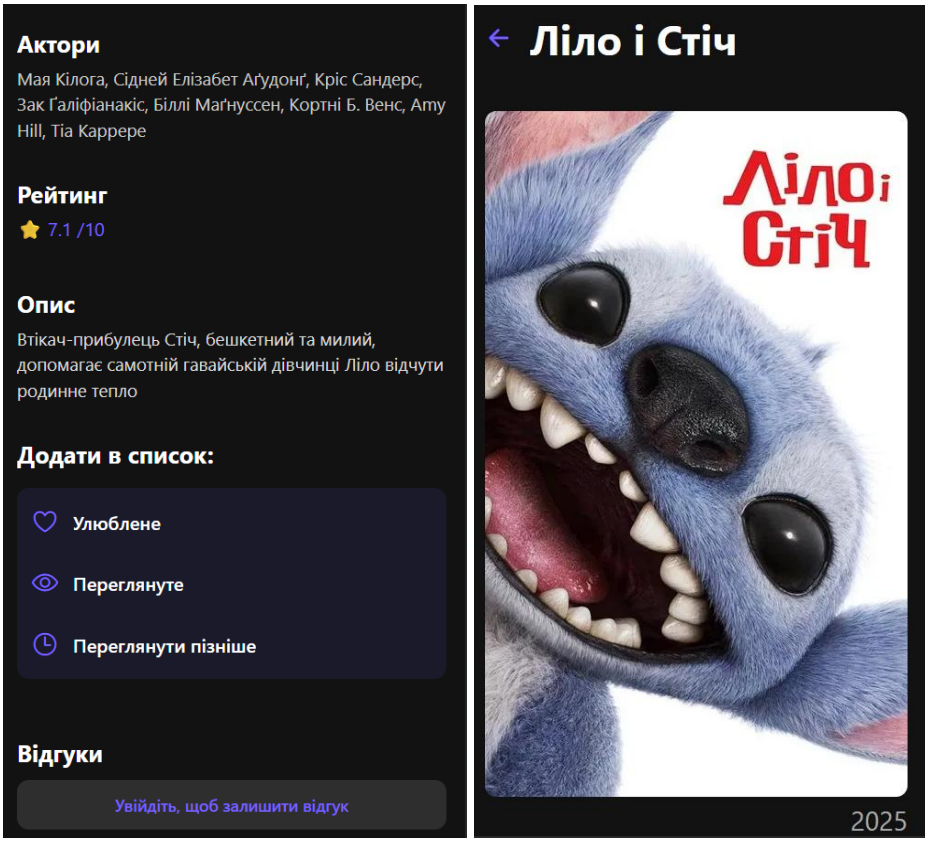

Рисунок 3.7 – Сторінка фільму для неавторизованого користувача

5. Перегляд деталей фільму (авторизований користувач)

Передумови:

Користувач зареєстрований і увійшов у свій обліковий запис.

Кроки виконання:

– Увійти в застосунок FilmBuddy під своїми даними (див. рисунок 3.35).

– На головній сторінці (див. рисунок 3.30) натиснути на картку будь-якого фільму.

– Переглянути сторінку з деталями фільму (див. рисунок 3.34).

– Взаємодіяти з кнопками: «Улюблене», «Переглянуте», «Переглянути пізніше», «Залишити відгук».

Очікуваний результат:

Користувач отримує повний доступ до функціональності сторінки фільму. Можна натискати на будь-яку з кнопок — система додасть або видалить фільм із відповідного списку. Відображення стану кнопок оновлюється миттєво, а всі дії зберігаються у Firestore. Користувач також може залишити відгук — відкривається форма, дані зберігаються й одразу виводяться під фільмом. Якщо користувач уже взаємодіяв із цим фільмом раніше, система правильно відобразить стан — наприклад, активну іконку «Улюблене». Усі елементи інтерфейсу працюють коректно, без збоїв.

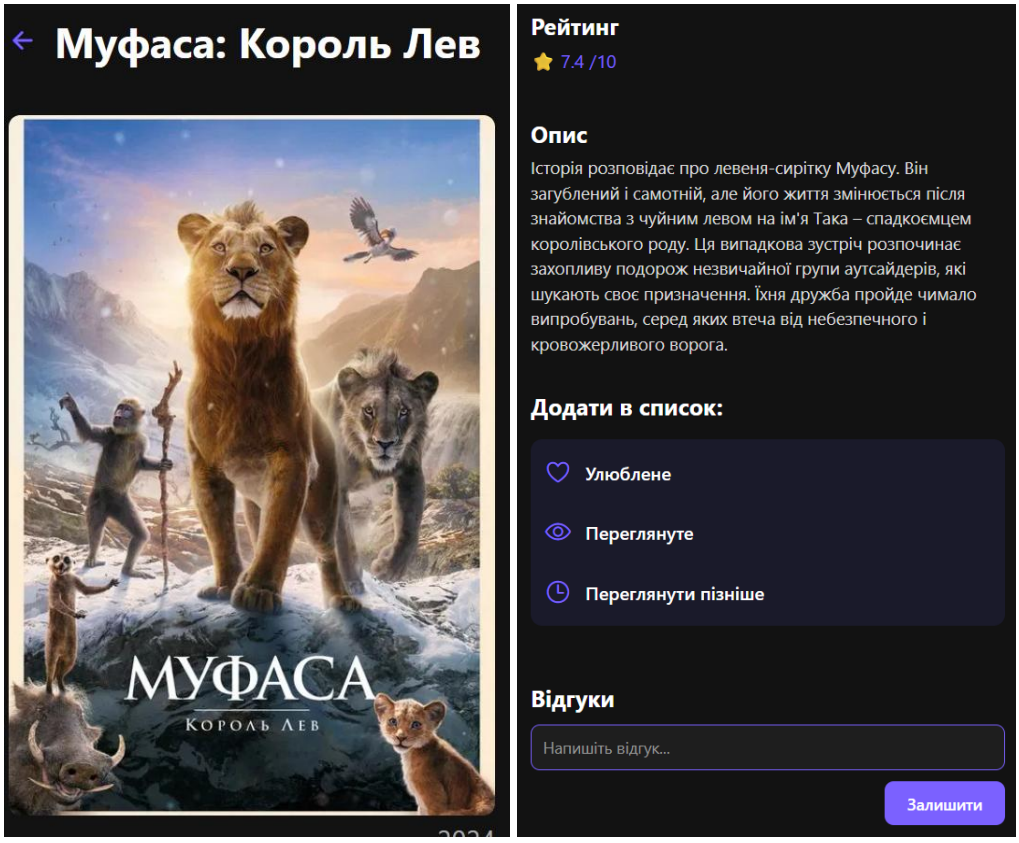

Рисунок 3.8 – Сторінка фільму для авторизованого користувача

6. Додавання фільму до списку «Улюблене»

Передумови:

Користувач авторизований у системі та знаходиться на сторінці будь-якого фільму.

Кроки виконання:

– Увійти до свого облікового запису в застосунку FilmBuddy.

– На головній сторінці або в результатах пошуку натиснути на картку фільму, щоб перейти на сторінку з деталями (див. рисунок 3.30).

– На сторінці фільму знайти кнопку з іконкою «серця» (додати до улюбленого).

– Натиснути на кнопку один раз.

– Перейти до профілю, обрати список «Улюблене» і переконатися, що фільм з’явився у переліку (див. рисунок 3.35).

– Повернутись до сторінки фільму й повторно натиснути кнопку для видалення з улюбленого.

Очікуваний результат:

Після натискання на кнопку «серце» фільм додається до списку «Улюблене». Іконка змінює стан (стає заповненою), з’являється коротке повідомлення або візуальна анімація, що підтверджує дію. При переході до списку «Улюблене» фільм відображається у вигляді картки з можливістю переходу до деталей. При повторному натисканні фільм видаляється зі списку, і його іконка повертається до неактивного вигляду. Дані зберігаються у Firestore і залишаються доступними після перезапуску застосунку.

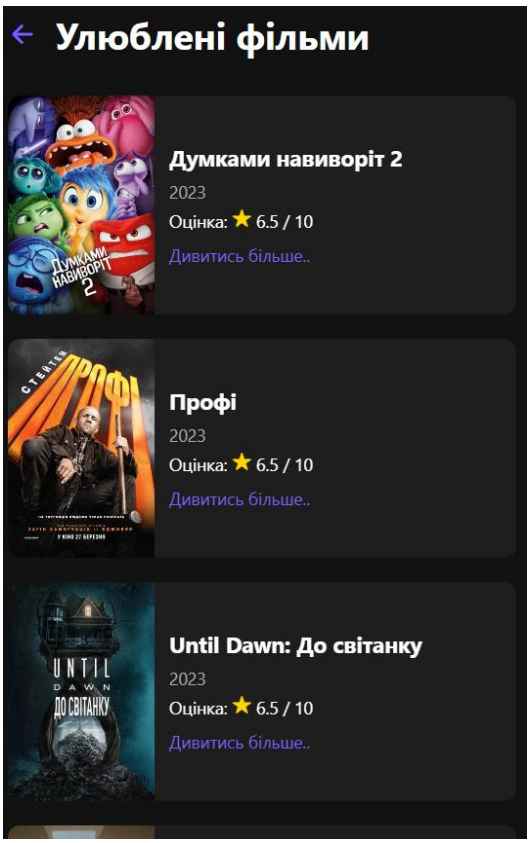

Рисунок 3.9 – Перелік улюблених фільмів у профілі користувача

7. Написання відгуку до фільму

Передумови:

Користувач авторизований та знаходиться на сторінці фільму, якому ще не залишав відгук.

Кроки виконання:

– Увійти до застосунку FilmBuddy під своїм обліковим записом.

– На головному екрані або в результатах пошуку вибрати фільм і перейти до його сторінки (див. рисунок 3.30).

– Прокрутити сторінку до секції «Відгуки».

– У полі введення написати текст відгуку.

– Натиснути кнопку «Залишити відгук».

– Дочекатися, поки новий відгук з’явиться у списку під фільмом (див. рисунок 3.36).

Очікуваний результат:

Після введення тексту та натискання на кнопку, відгук зберігається у Firestore та миттєво відображається на сторінці фільму. Разом із текстом виводяться ім’я користувача, його аватар та час створення. Якщо поле було порожнє, система блокує надсилання та повідомляє про помилку. Відгук зберігається навіть після перезавантаження застосунку. Повторне надсилання працює коректно — користувач може залишити декілька відгуків або відповісти на інші. Дані синхронізуються в реальному часі. Усі елементи — поле, кнопка, повідомлення — працюють стабільно, без збоїв і повторень.

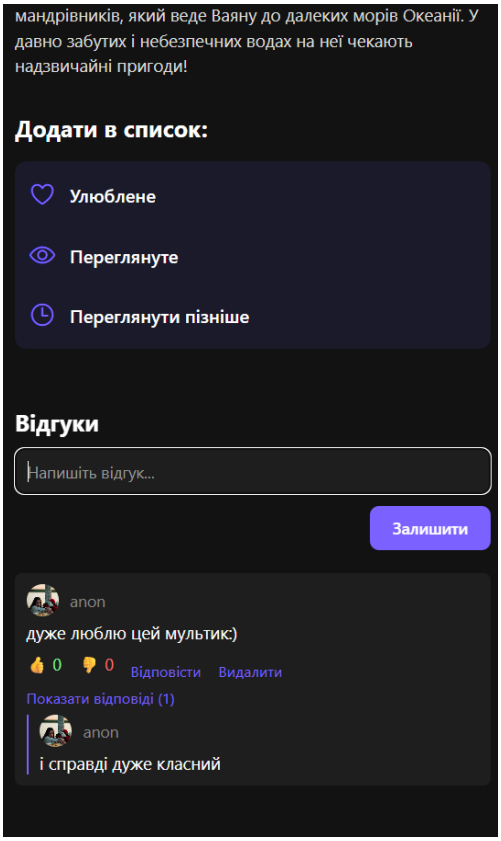

Рисунок 3.10 – Відображення нового відгуку під фільмом

8. Пошук фільмів за назвою

Передумови:

Користувач знаходиться в застосунку, авторизація необов’язкова.

Кроки виконання:

– Відкрити застосунок FilmBuddy.

– У нижньому меню натиснути на іконку «Пошук» для переходу на відповідний екран (див. рисунок 3.37).

– У полі пошуку вгорі ввести назву фільму або її частину.

– Дочекатися автоматичного відображення результатів (після невеликої затримки для debounce).

– Переконатися, що виводяться релевантні картки фільмів.

Очікуваний результат:

Після введення запиту система надсилає запит до TMDb API та виводить список фільмів, які відповідають тексту пошуку. Результати з’являються у вигляді вертикального списку карток із постером, назвою, роком випуску та рейтингом. Натискання на будь-яку картку веде до сторінки фільму. Якщо введено запит, що не відповідає жодному фільму, відображається повідомлення «Фільмів не знайдено». Після очищення поля результати зникають, і знову показуються категорії за жанрами. Усі переходи працюють коректно, пошук не викликає зависань чи збоїв, а також адаптований під різну швидкість інтернету.

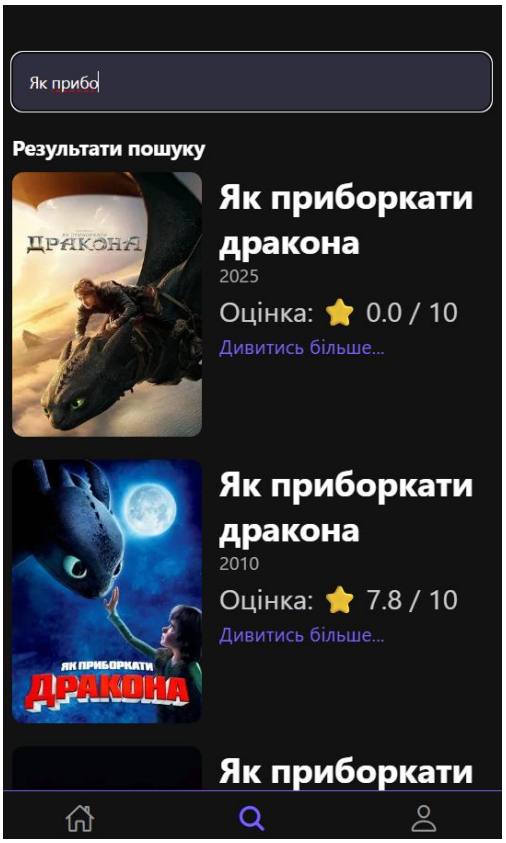

Рисунок 3.11 – Екран пошуку з активним запитом

9. Перегляд усіх фільмів у жанрі («Дивитись все»)

Передумови:

Користувач знаходиться в застосунку, авторизація необов’язкова.

Кроки виконання:

– Відкрити застосунок FilmBuddy.

– Перейти на екран пошуку через нижнє меню (див. рисунок 3.30).

– Прокрутити екран вниз до секції з жанрами.

– У жанровому блоці (наприклад, «Комедія», «Бойовик», «Фентезі») натиснути кнопку «Дивитись все» (див. рисунок 3.38).

– Перевірити, що відкривається сторінка з повним переліком фільмів у вибраному жанрі (див. рисунок 3.39).

– Натиснути на будь-який фільм зі списку, щоб перейти на його сторінку.

Очікуваний результат:

Після натискання кнопки «Дивитись все» відкривається новий екран із вертикальним списком фільмів відповідного жанру. Кожен фільм подається у вигляді картки з постером, назвою, роком та рейтингом. Назва жанру відображається у заголовку сторінки. Прокрутка працює плавно, без зависань. Натискання на картку фільму веде до сторінки з його деталями. Якщо в обраному жанрі наразі немає доступних фільмів, виводиться відповідне повідомлення. Після натискання кнопки “назад” користувач повертається на екран пошуку в те саме місце, де був. Функція працює стабільно на різних екранах і в авторизованому, і в гостьовому режимі.

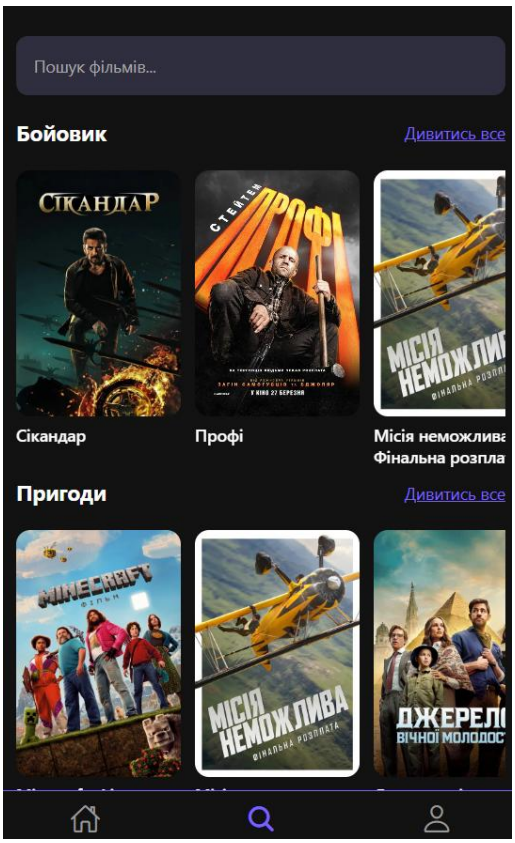

Рисунок 3.12 – Кнопка «Дивитись все» у жанровому блоці

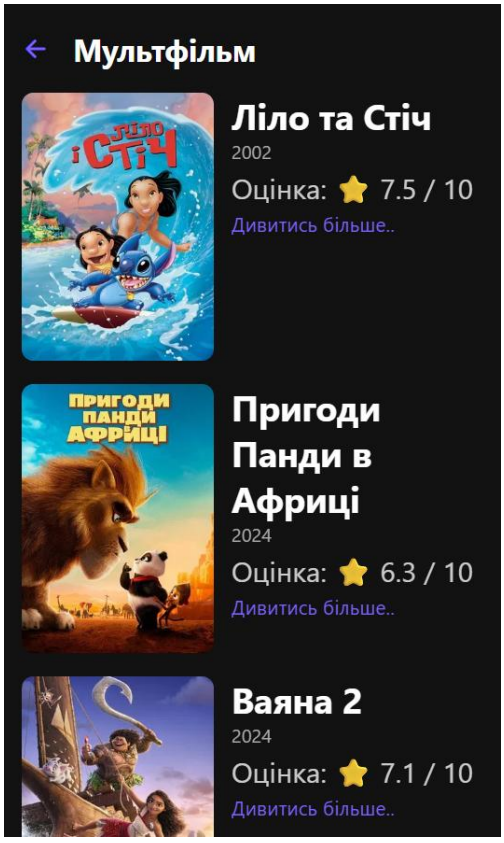

Рисунок 3.13 – Сторінка повного списку фільмів жанру

10. Зміна профілю користувача

Передумови:

Користувач увійшов у систему та знаходиться на екрані «Профіль».

Кроки виконання:

– Увійти до застосунку FilmBuddy.

– У нижньому меню натиснути на іконку «Профіль», щоб перейти до екрана з особистими даними (див. рисунок 3.40).

– Переконатися, що на екрані відображено ім’я користувача та аватар.

– Натиснути на кнопку редагування профілю.

– Змінити ім’я користувача або аватар.

– Натиснути кнопку «Зберегти зміни».

– Перезапустити застосунок і перевірити, що зміни збережено.

Очікуваний результат:

Після переходу до екрана профілю відображається поточна інформація з Firestore: ім’я, email та фото користувача. За наявності функції редагування, користувач може змінити своє ім’я або оновити зображення аватара. Після натискання кнопки «Зберегти зміни», нова інформація оновлюється у базі даних і одразу відображається в інтерфейсі. Дані залишаються збереженими при повторному вході або перезапуску застосунку. Якщо ж поля залишено порожніми або введено некоректні значення, система не дозволяє зберегти зміни та повідомляє про помилку. Всі елементи працюють стабільно та без затримок.

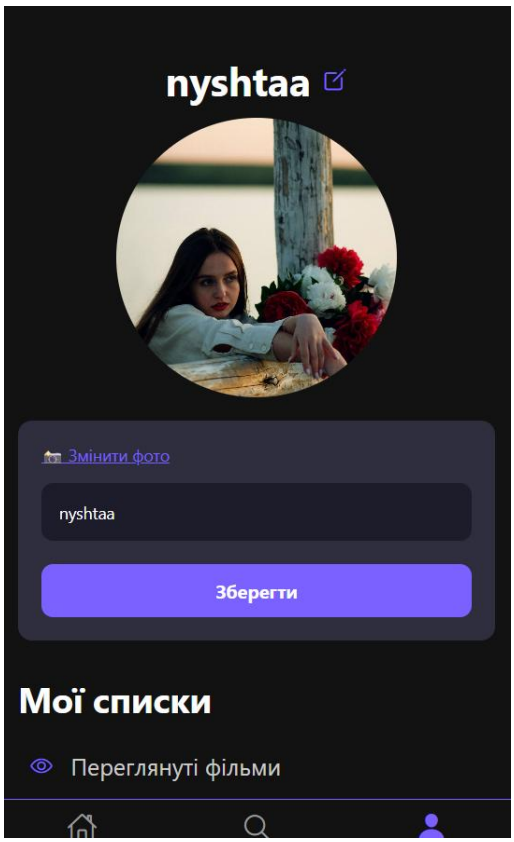

Рисунок 3.14 – Екран профілю користувача з можливістю редагування

Було здійснено поетапну перевірку всіх ключових елементів застосунку. Перевірка почалась із тестування реєстрації нового користувача, що включала валідацію вхідних даних, обробку помилок і збереження інформації в базі даних Firestore. Далі протестовано вхід до облікового запису, який підтвердив надійність авторизаційної системи та коректне завантаження профілю користувача. Окремо було розглянуто сценарії для неавторизованого користувача, який має обмежений доступ до функцій, але може переглядати головну сторінку, деталі фільмів та скористатися функцією випадкового вибору.У фокусі тестування опинилися функціональні можливості головного екрана, включно з формуванням персоналізованих рекомендацій для авторизованих користувачів, виведенням загальнодоступних добірок і взаємодією з картками фільмів. Було підтверджено стабільну роботу кнопки випадкового фільму, яка генерує результат миттєво, відкриваючи модальне вікно з можливістю переходу до повної інформації. Також важливим етапом стала перевірка функцій додавання до списків «Улюблене», «Переглянуте», «Переглянути пізніше» — як із точки зору збереження даних, так

і зміни стану інтерфейсу.Окрема частина тестування була присвячена функціональності відгуків: перевірено додавання нових коментарів, відображення імені та аватара користувача, можливість ставити лайки й дизлайки, а також залишати відповіді. Всі ці дії працюють у режимі реального часу, синхронізуються з базою та мають захист від багаторазового голосування. Ще одним важливим аспектом стала система пошуку, яка включає як пошук за назвою, так і доступ до категорій за жанрами. В обох випадках інтерфейс поводиться передбачувано, а пошукові запити обробляються ефективно.Також було протестовано можливість перегляду повного списку фільмів у певному жанрі через кнопку «Дивитись все», редагування профілю користувача, вихід з облікового запису, а також перегляд усіх написаних користувачем відгуків у розділі «Мої відгуки». Усі перевірки підтвердили правильну роботу кожної частини застосунку, збереження даних, їхнє коректне оновлення в інтерфейсі, а також стабільність навіть при великій кількості взаємодій.

Таким чином, проведене функціональне тестування підтвердило, що застосунок FilmBuddy є повністю функціональним, надійним і зручним у використанні. Всі основні сценарії були реалізовані відповідно до вимог, перевірені в реальному середовищі, та не виявили критичних помилок. Це свідчить про готовність системи до практичного використання, а також закладає основу для її подальшого вдосконалення й масштабування.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Способи проведення штучного дихання та масажу серця.

Раптова зупинка серцевої діяльності є однією з найнебезпечніших невідкладних ситуацій, що становить пряму загрозу життю людини. Відсутність ефективного кровообігу та дихання навіть протягом 4–6 хвилин призводить до незворотних змін у клітинах головного мозку та розвитку біологічної смерті. Єдиним способом зберегти життя потерпілому є негайне проведення серцево-легеневої реанімації (СЛР), яка включає штучне дихання та зовнішній масаж серця.

Надання допомоги у таких випадках регламентується Наказом МОЗ України № 398 від 16.06.2014 «Про затвердження інструкцій з надання домедичної допомоги» [15], а порядок дій у надзвичайних ситуаціях — ДСТУ ISO 22320:2022 «Захист суспільства. Надзвичайні ситуації. Вимоги до реагування» [14]. Практичні рекомендації з проведення реанімаційних заходів також подано у підручнику «Безпека життєдіяльності та охорона праці» за редакцією Грибана В.В. [16].

Ознаки клінічної смерті:

- відсутність свідомості;
- відсутність дихання;
- відсутність пульсу на сонній артерії;
- розширення зіниць;
- блідість або синюшність шкіри.

Зовнішній (непрямий) масаж серця

Послідовність дій при наданні домедичної допомоги дорослим при раптовій зупинці кровообігу за відсутності автоматичного зовнішнього дефібрилятора:

1) перед наданням допомоги переконатися у відсутності небезпеки та за її відсутності перейти до наступного кроку;

2) визначити наявність свідомості - обережно потрясти дорослого за плече та голосно звернутися до нього, наприклад, «З Вами все гаразд? Вам потрібна допомога?»;

3) якщо дорослий реагує:

а) залишити його у попередньому положенні, якщо йому нічого не загрожує;

б) з'ясувати характер події, що сталася;

в) здійснити виклик екстреної медичної допомоги та дотримуватись вказівок диспетчера прийому виклику;

г) за необхідності надати дорослому зручного положення;

г) забезпечити нагляд за дорослим до приїзду бригади екстреної (швидкої) медичної допомоги;

4) якщо дорослий не реагує:

а) звернутися до осіб, які поряд, за допомогою. Якщо випадкових свідків декілька, слід звертатися до конкретної особи;

б) якщо дорослий лежить на животі, повернути його на спину та відновити прохідність дихальних шляхів. Якщо стан дорослого пов'язаний з отриманням травми, наприклад падіння з висоти, вважати, що у нього є травма в шийному відділі хребта. В такому випадку слід максимально обмежити рухи в шийному відділі хребта;

в) відновити прохідність дихальних шляхів, визначити наявність дихання за допомогою прийому: «чути, бачити, відчувати». Наявність дихання визначати до 10 секунд. Якщо виникли сумніви чи є дихання, або воно ненормальне, вважати, що дихання відсутнє;

5) якщо дорослий дихає нормально, при відсутності свідомості слід перевести його у стабільне бокове положення, здійснити виклик екстреної медичної допомоги та перевіряти кожні 3-5 хвилин дихання до моменту приїзду бригади екстреної (швидкої) медичної допомоги. Якщо є настороженість щодо травми потрібно уникати повороту в стабільне бокове положення, а забезпечити

прохідність дихальних шляхів методом висування нижньої щелепи до приїзду бригади екстреної (швидкої) медичної допомоги;

б) якщо дихання відсутнє:

а) здійснити виклик екстреної медичної допомоги та дотримуватись вказівок диспетчера прийому виклику. Якщо є інші випадкові свідки, слід сказати їм здійснити виклик екстреної медичної допомоги та негайно перейти до наступного кроку. При наявності гучного зв'язку на телефоні слід здійснювати виклик екстреної медичної допомоги та одночасно проводити серцево-легеневу реанімацію;

б) розпочати проведення серцево-легеневої реанімації:

виконати 30 натискань на середину грудної клітки глибиною не менше 5 см (не більше 6 см), з частотою 100 натискань (не більше 120) за хвилину;

виконати 2 вдихи з використанням маски-клапану, дихальної маски тощо. При відсутності захисних засобів можна не виконувати штучне дихання, а проводити тільки натискання на грудну клітку. Виконання двох штучних вдихів повинно тривати не більше 5 секунд;

після двох вдихів продовжити натискання на грудну клітку відповідно до наведених рекомендацій у цьому підпункті;

не слід переривати натиснення на грудну клітку дорослому більше ніж на 10 секунд;

7) змінювати особу, що проводить натискання на грудну клітку, кожні 2 хвилини. У випадку якщо особа, яка проводить натискання на грудну клітку, відчуває виснаження, заміну слід виконати раніше ніж через 2 хвилини;

8) припинити проведення серцево-легеневої реанімації до прибуття бригади екстреної (швидкої) медичної допомоги за наступних умов: при появі у дорослого явних ознак життя; відновлення самостійного нормального дихання, координованої рухової активності, відкривання очей; виникненні загрози життю рятівнику та/або дорослому; неможливості проведення серцево-легеневої реанімації внаслідок значного фізичного виснаження [15].

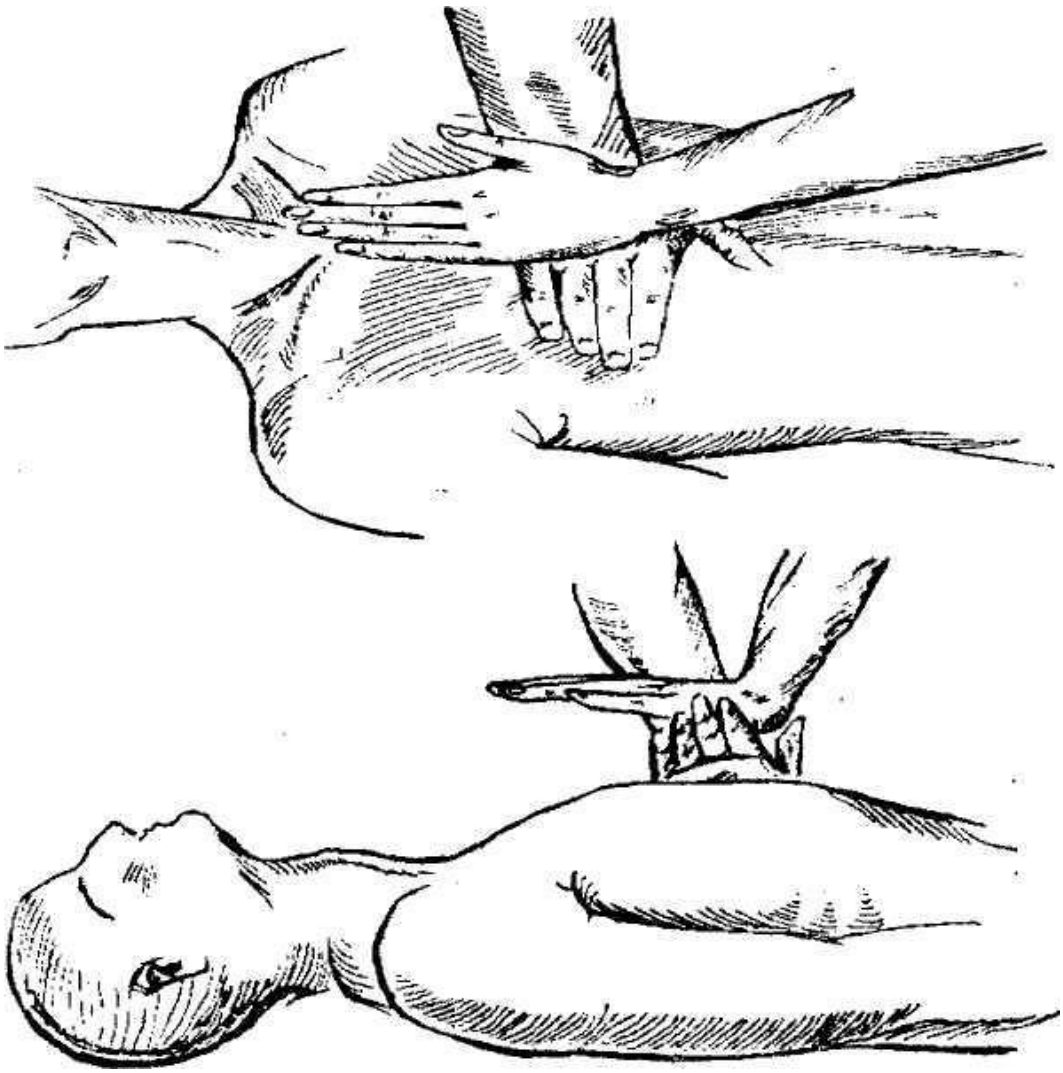

Рисунок 4.1 – Проведення непрямого масажу серця у дорослого постраждалого

У громадських місцях дедалі частіше встановлюються автоматичні зовнішні дефібрилятори, які дозволяють значно підвищити шанси на виживання постраждалого у разі фібриляції шлуночків. АЗД самостійно аналізує ритм серця та рекомендує проведення електричної дефібриляції у разі показань. Рятувальнику залишається лише правильно прикріпити електроди та дотримуватись голосових інструкцій пристрою.

Таким чином, ефективність заходів першої допомоги при раптовій зупинці серця повністю залежить від підготовленості рятувальника, чіткості виконання алгоритму СЛР та доступності автоматичних дефібриляторів. Навчання населення базовим навичкам реанімації є пріоритетним завданням у сфері охорони здоров’я та безпеки життєдіяльності.

4.2 Інструкція для обслуговуючого персоналу на випадок виникнення аварії, пожежі

Забезпечення організованих дій персоналу під час надзвичайних ситуацій — одна з ключових вимог системи охорони праці та безпеки життєдіяльності. В умовах аварії або пожежі правильні й швидкі дії можуть запобігти людським жертвам і мінімізувати матеріальні збитки.

Відповідно до Кодексу цивільного захисту України (Закон № 5403-VI від 02.10.2012) усі працівники зобов’язані знати порядок дій у разі аварії, пожежі або іншої надзвичайної ситуації, брати участь у навчаннях і дотримуватись затверджених інструкцій [17].

Раптова аварія або пожежа на підприємстві становлять серйозну загрозу для життя і здоров’я людей, збереження матеріальних цінностей та стабільності роботи об’єкта. Інструкції з пожежної безпеки мають розроблятись відповідно до Правил пожежної безпеки в Україні, затверджених наказом МВС № 1417 від 30.12.2014 [18].

Згідно з пунктами 3–4 Правил, у разі виявлення ознак пожежі (диму, відкритого вогню, запаху горіння, спрацювання пожежної сигналізації) обслуговуючий персонал повинен:

- Негайно повідомити про подію відповідальну особу, керівника чи чергового.
- Викликати пожежно-рятувальну службу за телефоном 101, чітко вказавши адресу, місце загоряння, наявність людей у приміщенні, особливості об’єкта.
- Оповістити людей на об’єкті про необхідність евакуації (в тому числі – за допомогою системи оповіщення або вручну).
- Розпочати евакуацію працівників і відвідувачів згідно з затвердженим планом евакуації. Особливу увагу слід приділити допомозі людям з інвалідністю.

- За можливості – знеструмити приміщення і розпочати гасіння пожежі на початковому етапі за допомогою первинних засобів пожежогасіння (вогнегасники, внутрішній пожежний кран тощо).
- Уникати паніки й не створювати тисняви при евакуації.

Персонал зобов’язаний знати розміщення пожежного інвентарю, основні типи вогнегасників, їхнє призначення та правила застосування. Перед початком роботи кожен співробітник повинен пройти первинний інструктаж з пожежної безпеки, а надалі – періодичні повторні інструктажі.

У разі прибуття підрозділів ДСНС працівники повинні надати їм доступ до джерел водопостачання, приміщень, в яких може перебувати вогнище займання, а також повідомити про можливу наявність небезпечних речовин чи обладнання.

Після ліквідації пожежі обслуговуючий персонал не має права самостійно відновлювати роботу об’єкта без дозволу відповідних служб, поки не буде проведено розслідування причин та перевірка безпечності приміщень.

Таким чином, дії обслуговуючого персоналу повинні бути чітко організованими, відповідно до плану евакуації, інструкцій з охорони праці та пожежної безпеки, а також базуватись на чинних нормативно-правових актах у сфері цивільного захисту.

Пожежа — це «неконтрольоване горіння, що розвивається у просторі та часі». Основними небезпечними факторами є:

- відкрите полум’я, висока температура;
- задимленість і токсичні продукти згоряння;
- зниження концентрації кисню;
- паніка серед людей;
- ризик обвалення конструкцій та утворення вогняного шторму [16].

Як зазначає Грибан В.В., дим здатен повністю обмежити видимість вже на відстані 1 м, тому знання евакуаційних шляхів є критично важливим. Токсичні речовини в диму, як-от чадний газ або синильна кислота, можуть викликати смерть упродовж кількох хвилин [16].

ВИСНОВКИ

У процесі виконання дипломної роботи було реалізовано повнофункціональний мобільний застосунок FilmBuddy для рекомендацій, пошуку та перегляду інформації про фільми. Основною метою розробки стало створення сучасного інструменту, що дозволяє користувачеві ефективно знаходити релевантний відеоконтент відповідно до його інтересів, історії взаємодії та уподобань. Реалізація такого рішення є відповіддю на проблему інформаційного перевантаження, з якою дедалі частіше стикаються користувачі під час вибору фільмів серед великої кількості доступного контенту.

Застосунок побудований із використанням фреймворку React Native та платформи Expo, що забезпечило кросплатформенність, високу швидкість розробки та сучасний зовнішній вигляд інтерфейсу. Для збереження користувацьких даних і забезпечення авторизації було використано Firebase, а для отримання інформації про фільми — API The Movie Database (TMDb). Уся структура проєкту побудована таким чином, щоб забезпечити гнучкість, масштабованість і зручність підтримки. Компонентна архітектура дозволила ефективно розділити логіку на окремі екрани та функціональні блоки.

У застосунку реалізовано систему реєстрації та входу користувачів, інтерфейс пошуку з можливістю фільтрації за жанрами, сторінки з детальною інформацією про фільми, система збереження в обрані списки («Улюблені», «Переглянуті», «Переглянути пізніше»), перегляд фільмів за жанрами, рейтингом і новизною, генерація випадкового фільму, а також інтелектуальний алгоритм рекомендацій на основі історії вподобань користувача. Одним із важливих досягнень є реалізація розширеної системи відгуків, що підтримує багаторівневі коментарі, можливість оцінки інших відгуків (лайк/дизлайк), відображення аватарів і імен користувачів, а також збереження відгуків у Firestore для подальшого аналізу.

Користувацький інтерфейс адаптований до темної теми, що відповідає сучасним стандартам дизайну та зручності. Реалізовано підтримку жесту "свайп-назад", що підвищує зручність навігації між екранами. Інтерактивні елементи розміщені інтуїтивно, а структура сторінок дозволяє швидко орієнтуватися у великому обсязі контенту.

Окрім реалізації функціонального застосунку, в рамках дипломної роботи було проведено детальний аналіз предметної області, виявлено основні проблеми користувачів при виборі фільмів, а також досліджено існуючі рішення на ринку мобільних застосунків для перегляду контенту. На основі цього аналізу сформульовано вимоги до системи та запропоновано архітектурну модель, яка забезпечує гнучке розширення та адаптацію застосунку до майбутніх потреб. Практична реалізація проекту засвідчила доцільність поєднання технологій Firebase, TMDb API та React Native у створенні сучасних сервісів персоналізованих рекомендацій. Здобутий досвід дозволяє стверджувати, що використані підходи можуть бути адаптовані і до інших галузей, де необхідна індивідуалізація контенту та висока інтерактивність користувача.

Загалом, результати реалізації свідчать про успішне досягнення поставлених цілей: застосунок FilmBuddy забезпечує повноцінний функціонал, що дозволяє користувачам швидко знаходити та зберігати цікаві фільми, отримувати персоналізовані рекомендації та ділитися власною думкою. Така система має потенціал до подальшого розвитку, включаючи впровадження алгоритмів машинного навчання, розширення соціальної взаємодії між користувачами та інтеграцію з зовнішніми сервісами. Таким чином, FilmBuddy є прикладом ефективного застосування сучасних технологій для створення зручного, адаптивного та інтелектуального мобільного рішення в галузі цифрових медіа.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. 1. Petryk M. R., Boyko I. V., Khimich O. M. High-Performance Supercomputer Technologies of Simulation and Identification of Nanoporous Systems with Feedback for n-Component Competitive Adsorption. *Cybern Syst Anal 57*. 2021. С. 316–328. URL: https://doi.org/10.1007/s10559-021-00357-7.

2. Alkhatib, N. (2011). *Mobile Application Recommender System*. Master's Thesis, Uppsala University. [Електронний ресурс] – Режим доступу до ресурсу https://uu.diva-portal.org/smash/get/diva2:428092/FULLTEXT01.pdf

3. Методології розробки програмного забезпечення | Wezom [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/metodologija-razrabotki-programmnogo-obespechenija

4. Iterative Model - Google Suche [Електронний ресурс] – Режим доступу до ресурсу: (PDF) Blockchain Enabled Smart Contract Based Applications: Deficiencies with the Software Development Life Cycle Models

5. What is React Native? Complex Guide for 2024 | Netguru [Електронний ресурс] – Режим доступу до ресурсу: https://www.netguru.com/glossary/react-native

6. Bilgili Ö. Exploring Expo in React Native: A Comprehensive Guide to Cross-Platform App Development [Електронний ресурс] – Режим доступу до ресурсу: https://omurbilgili.medium.com/exploring-expo-in-react-native-a-comprehensive-guide-to-cross-platform-app-development-45e6a3bfa111

7. All. The Movie Database (TMDB) [Електронний ресурс] – Режим доступу до ресурсу: https://developer.themoviedb.org/reference/trending-all

8. GautamManak. Introduction to Firebase [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/@gautammanak1/introduction-to-firebase-649e6b7c62bc

9. VS Code Tips. CODE Magazine [Електронний ресурс] – Режим доступу до ресурсу: https://www.codemag.com/Article/2408031/VS-Code-Tips

10. Everything Developers Need to Know About Figma – Smashing Magazine [Електронний ресурс] – Режим доступу до ресурсу: https://www.smashingmagazine.com/2020/09/figma-developers-guide/

11. Frontend Mentor | Git and GitHub Essentials: A Beginner's Guide [Електронний ресурс] – Режим доступу до ресурсу: https://www.frontendmentor.io/articles/git-and-github-essentials-a-beginners-guide-T4i1dKqfmH

12. Тестування інтерфейсу користувача (UI) | Wezom [Електронний ресурс] – Режим доступу до ресурсу: https://wezom.com.ua/ua/blog/testing-ui-user-interface

13. Що таке функціональне тестування? Типи, приклади, контрольний список і впровадження | ZAPTEST [Електронний ресурс] – Режим доступу до ресурсу: https://www.zaptest.com/uk/що-таке-функціональне-тестування-тип

14. ДСТУ ISO 22320:2022. Безпека та адаптивність. Управління в надзвичайних ситуаціях. Настанови щодо управління інцидентами [Електронний ресурс] – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page?id_doc=99547&utm_source

15. Наказ МОЗ України № 398 від 16.06.2014 «Про затвердження інструкцій з надання домедичної допомоги» [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0252-15#Text

16. Грибан В.В. Безпека життєдіяльності та охорона праці: підручник. – Кропивницький: Ексклюзив-Систем, 2022. – С. 140–143

17. Кодекс цивільного захисту України: Закон України № 5403-VI від 02.10.2012 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/5403-17

18. Правила пожежної безпеки в Україні: Наказ МВС № 1417 від 30.12.2014 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0252-15#Text

ДОДАТКИ

ДОДАТОК А – Тези конференції

УДК 004.415
Ништа І. – ст. гр. СП-43
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ РЕКОМЕНДАЦІЙ ФІЛЬМІВ З ВИКОРИСТАННЯМ REACT NATIVE

Науковий керівник: к.т.н., доцент Михалик Д. М.

Nyshta I.
Ternopil Ivan Puluj National Technical University

MOBILE APPLICATION DEVELOPMENT FOR MOVIE RECOMMENDATIONS USING REACT NATIVE

Supervisor: D. M. Mykhalyk

Ключові слова: React Native, рекомендації фільмів, мобільний застосунок
Keywords: React Native, movie recommendations, mobile application

З розвитком мобільних технологій програми стали невід'ємною частиною повсякденного життя. А з такою кількістю різноманітного контенту, включно з фільмами, зробити вибір ще важче. Тому для швидкої розробки з кросплатформними можливостями було обрано фреймворк React Native, який, згідно з опитуванням Stack Overflow 2022 року, став одним із найпопулярніших фреймворків [1].

Метою цього дослідження є розробка мобільного додатку, який би рекомендував фільми. Його основні функції — автентифікація, перегляд категорій, пошук фільмів, написання рецензій і збереження історії переглядів. Розробка базується на фреймворку React Native з використанням API The Movie Database(TMDb)[2] і служб Firebase.

Інтерфейс користувача побудовано з використанням компонентів React і бібліотеки react-navigation для керування екранами. Щоб отримати інформацію про фільм, програма отримує доступ до API TMDb і використовує Firebase для автентифікації користувача.

Перевага фреймворку React Native полягає в тому, що він пропонує високу продуктивність і скорочений час розробки з єдиною базою коду для платформ Android і iOS.

Результатом став мобільний додаток, який дозволяє користувачам миттєво отримувати персоналізовані рекомендації щодо фільмів у зручному інтерфейсі. React Native виявився гнучким інструментом, який забезпечує ефективну розробку та більшу масштабованість проекту.

Література

1. React Native Pros and Cons [2023 Update]. *Software Development Company | Netguru*. URL: https://www.netguru.com/blog/react-native-pros-and-cons .
2. Getting Started. *The Movie Database (TMDB)*. URL: https://developer.themoviedb.org/ .

ДОДАТОК Б – Лістинг коду мобільного застосунку

Файл «_layout.tsx»

```
import { Ionicons } from "@expo/vector-icons";
import { Tabs } from "expo-router";

export default function TabsLayout() {
  return (
    <Tabs
      screenOptions={({ route }) => ({
        headerShown: false,
        tabBarStyle: {
            backgroundColor: "#121212",
            borderTopColor: "#7B61FF",
          },
          tabBarShowLabel: false,
        tabBarIcon: ({ focused }) => {
            let iconName: keyof typeof Ionicons.glyphMap = "ellipse-
outline";
           if (route.name === "home") iconName = focused ? "home" :
"home-outline";
            else if (route.name === "search") iconName = focused ?
"search" : "search-outline";
            else if (route.name === "profile") iconName = focused ?
"person" : "person-outline";

          return <Ionicons name={iconName} size={26} color={focused ?
"#7B61FF" : "#999"} />;
        },
      })}
    />
  );
}
```

Файл «home.tsx»

```
import { LinearGradient } from 'expo-linear-gradient';
import { useRouter } from "expo-router";
```

```
import React, { useEffect, useState } from "react";
import { ActivityIndicator, FlatList, Image, Modal, ScrollView, StyleSheet, Text, TouchableOpacity, View } from "react-native";
import { SafeAreaView } from "react-native-safe-area-context";
import { auth } from "../../config/firebase";
import MovieCard from "../../src/components/MovieCard";
import { fetchMovies } from "../../src/services/movieService";
import { getGuestTopMovies, getSmartRecommendedMovies } from "../../src/services/recommendationService";

function HomeScreen() {
  const router = useRouter();
  const [recommended, setRecommended] = useState<any[]>([]);
  const [topRated, setTopRated] = useState<any[]>([]);
  const [upcoming, setUpcoming] = useState<any[]>([]);
  const [nowPlaying, setNowPlaying] = useState<any[]>([]);
  const [randomMovie, setRandomMovie] = useState<any | null>(null);
  const [showModal, setShowModal] = useState(false);
  const [loading, setLoading] = useState(true);
  const [isGuest, setIsGuest] = useState(false);

  useEffect(() => {
  const unsubscribe = auth.onAuthStateChanged(async (user) => {
    setIsGuest(!user);
    setLoading(true);

    const [top, up, now] = await Promise.all([
      fetchMovies("top_rated"),
      fetchMovies("upcoming"),
      fetchMovies("now_playing"),
    ]);

    setTopRated(top);
    setUpcoming(up);
    setNowPlaying(now);

    let rec = [];
    if (user) {
      rec = await getSmartRecommendedMovies(user.uid);
```

```
            console.log("🔐 Авторизований — рекомендацій:", rec.length);
          } else {
            rec = await getGuestTopMovies();
            console.log("👥 Гість — топ:", rec.length);
          }

          setRecommended(rec);
          setLoading(false);
        });

        return () => unsubscribe(); // clean up
      }, []);

        const getRandomMovie = () => {
             const all = [...recommended, ...topRated, ...upcoming,
...nowPlaying];
          if (all.length === 0) return;
          const selected = all[Math.floor(Math.random() * all.length)];
          setRandomMovie(selected);
          setShowModal(true);
        };

        const renderSection = (title: string, data: any[]) => (
          <View style={styles.section}>
            <View style={styles.sectionHeader}>
              <Text style={styles.sectionTitle}>{title}</Text>
            </View>
            <FlatList
              data={data}
              horizontal
              keyExtractor={(item) => item.id.toString()}
              renderItem={({ item }) => (
                <TouchableOpacity
                  onPress={() =>
                      router.push({ pathname: "/movie/[id]", params: { id:
item.id.toString() } })
                  }
                  style={styles.cardWrapper}
```

```
              >
                                          <MovieCard     title={item.title}
posterPath={item.poster_path} />
              </TouchableOpacity>
            )}
            showsHorizontalScrollIndicator={false}
          />
        </View>
      );

      if (loading) {
        return (
          <View style={styles.loaderContainer}>
            <ActivityIndicator size="large" color="#7B61FF" />
          </View>
        );
      }

      const recommendedTitle = isGuest ? "Топ-20" : "Рекомендовані";

      return (
        <SafeAreaView style={styles.container}>
          <ScrollView showsVerticalScrollIndicator={false}>
                          <TouchableOpacity     onPress={getRandomMovie}
activeOpacity={0.85} style={styles.buttonWrapper}>
              <LinearGradient
                colors={["#A288FF", "#7B61FF"]}
                start={{ x: 0.5, y: 0 }}
                end={{ x: 0.5, y: 1 }}
                style={styles.button}
              >
                <Text style={styles.buttonText}>Випадковий фільм</Text>
              </LinearGradient>
            </TouchableOpacity>

            {renderSection(recommendedTitle, recommended)}
            {renderSection("Топ рейтингу", topRated)}
            {renderSection("Очікувані", upcoming)}
            {renderSection("Нові релізи", nowPlaying)}
```

```
          </ScrollView>

          <Modal visible={showModal} transparent animationType="fade">
            <View style={styles.modalContainer}>
              <View style={styles.modalContent}>
                {randomMovie && (
                  <>
                    <TouchableOpacity
                      onPress={() => {
                        setShowModal(false);
                        router.push({
                          pathname: "/movie/[id]",
                          params: { id: randomMovie.id.toString() },
                        });
                      }}
                    >
                      <Image
                                                              source={{    uri:
`https://image.tmdb.org/t/p/w500${randomMovie.poster_path}` }}
                        style={styles.poster}
                      />
                    </TouchableOpacity>
                              <TouchableOpacity  onPress={getRandomMovie}
style={styles.tryAgain}>
                        <Text  style={styles.tryAgainText}>Спробувати ще
раз</Text>
                    </TouchableOpacity>
                                        <TouchableOpacity   onPress={()   =>
setShowModal(false)}>
                      <Text style={styles.closeText}>Закрити</Text>
                    </TouchableOpacity>
                  </>
                )}
              </View>
            </View>
          </Modal>
        </SafeAreaView>
      );
    }
```

```
export default HomeScreen;
```

Файл «profile.tsx»

```
import { Ionicons } from "@expo/vector-icons";
import * as ImagePicker from "expo-image-picker";
import { useRouter } from "expo-router";
import { onAuthStateChanged } from "firebase/auth";
import { doc, getDoc, setDoc } from "firebase/firestore";
import React, { useEffect, useState } from "react";
import { ActivityIndicator, Alert, Image, ScrollView, StyleSheet, Text, TextInput, TouchableOpacity, View,} from "react-native";
import { auth, db } from "../../config/firebase";
function ProfileScreen() {
  const router = useRouter();
  const [uid, setUid] = useState<string | null>(null);
  const [displayName, setDisplayName] = useState("");
  const [photoURL, setPhotoURL] = useState("https://i.imgur.com/BoN9kdC.png");
  const [isEditing, setIsEditing] = useState(false);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, async (user) => {
      if (!user) {
        console.log("❌ Користувач не авторизований");
        router.replace("/login");
        return;
      }

      setUid(user.uid);
      console.log("✅ Авторизований користувач:", user.uid);

      try {
        const docRef = doc(db, "users", user.uid);
        const docSnap = await getDoc(docRef);
```

```
            if (docSnap.exists()) {
              const data = docSnap.data();
              setDisplayName(data.displayName || "");
                                          setPhotoURL(data.photoURL      ||
"https://i.imgur.com/BoN9kdC.png");
            } else {
                  console.log("📄 Документ не існує. Можливо, новий
користувач.");
            }
          } catch (err) {
            console.log("❌ Помилка завантаження профілю:", err);
             setError("Не вдалося завантажити профіль. Перевірте інтернет
або права доступу.");
          }

          setLoading(false);
        });

        return unsubscribe;
      }, []);

      const pickImage = async () => {
        const result = await ImagePicker.launchImageLibraryAsync({
          mediaTypes: ImagePicker.MediaTypeOptions.Images,
          allowsEditing: true,
          aspect: [1, 1],
          quality: 0.7,
        });

        if (!result.canceled && result.assets.length > 0) {
          setPhotoURL(result.assets[0].uri);
        }
      };

      const saveProfile = async () => {
        if (!uid) {
          Alert.alert("Помилка", "Користувач не авторизований.");
          return;
```

```
        }

        try {
          await setDoc(doc(db, "users", uid), {
            displayName,
            photoURL,
          });

          Alert.alert("✅ Успіх", "Профіль збережено.");
          setIsEditing(false);
        } catch (err) {
          console.log("❌ Помилка збереження профілю:", err);
          Alert.alert("Помилка", "Не вдалося зберегти профіль.");
        }
      };

      if (loading) {
        return (
          <View style={[styles.container, { justifyContent: "center" }]}>
            <ActivityIndicator size="large" color="#7B61FF" />
          </View>
        );
      }

      return (
          <ScrollView contentContainerStyle={styles.container} style={{
backgroundColor: "#1c1c2a" }}>
          {error && (
            <Text style={{ color: "#ff6b6b", marginBottom: 20, fontSize:
14 }}>{error}</Text>
          )}

          <View style={styles.headerRow}>
                        <Text    style={styles.title}>{displayName    ||
"Користувач"}</Text>
            <TouchableOpacity onPress={() => setIsEditing(true)}>
              <Ionicons name="create-outline" size={22} color="#7B61FF"
/>
```

```
            </TouchableOpacity>
          </View>

          <Image source={{ uri: photoURL }} style={styles.avatar} />

          {isEditing && (
            <View style={styles.editBox}>
              <TouchableOpacity onPress={pickImage}>
                <Text style={styles.editPhoto}>📸 Змінити фото</Text>
              </TouchableOpacity>

              <TextInput
                style={styles.input}
                value={displayName}
                onChangeText={setDisplayName}
                placeholder="Нове ім’я"
                placeholderTextColor="#888"
              />

                              <TouchableOpacity    style={styles.saveButton}
onPress={saveProfile}>
                <Text style={styles.saveButtonText}>Зберегти</Text>
              </TouchableOpacity>
            </View>
          )}

          <View style={styles.section}>
            <Text style={styles.sectionTitle}>Мої списки</Text>
               <TouchableOpacity  style={styles.listItem}  onPress={()  =>
router.push("/watched")}>
              <Ionicons name="eye-outline" size={20} color="#7B61FF" />
              <Text style={styles.listText}>Переглянуті фільми</Text>
            </TouchableOpacity>
               <TouchableOpacity  style={styles.listItem}  onPress={()  =>
router.push("/favorites")}>
              <Ionicons name="heart-outline" size={20} color="#7B61FF" />
              <Text style={styles.listText}>Улюблене</Text>
            </TouchableOpacity>
```

```
                <TouchableOpacity style={styles.listItem} onPress={() =>
router.push("/watch-later")}>
                <Ionicons name="time-outline" size={20} color="#7B61FF" />
                <Text style={styles.listText}>Переглянути пізніше</Text>
              </TouchableOpacity>
            </View>

            <View style={styles.section}>
              <Text style={styles.sectionTitle}>Мої відгуки</Text>
                 <TouchableOpacity style={styles.listItem} onPress={() =>
router.push("/reviews")}>
                    <Ionicons name="chatbubble-ellipses-outline" size={20}
color="#7B61FF" />
                <Text style={styles.listText}>Всі відгуки</Text>
              </TouchableOpacity>
            </View>
          </ScrollView>
        );
      }
      export default ProfileScreen;
```

Файл «search.tsx»

```
      import { useRouter } from "expo-router";
      import React, { useEffect, useState } from "react";
      import {
        FlatList,
        Image,
        ScrollView,
        StyleSheet,
        Text,
        TextInput,
        TouchableOpacity,
        View,
      } from "react-native";
      import MovieCard from "../../src/components/MovieCard";
      import {
        fetchGenres,
        fetchMoviesByGenre,
```

```
  searchMovies,
} from "../../src/services/movieService";

// Тип для фільму
type Movie = {
  id: number;
  title: string;
  poster_path: string;
  release_date?: string;
  vote_average?: number;
};

// Тип для жанру
type Genre = {
  id: number;
  name: string;
};

function SearchScreen() {
  const router = useRouter();
  const [text, setText] = useState("");
  const [genres, setGenres] = useState<Genre[]>([]);
  const [moviesByGenre, setMoviesByGenre] = useState<{ [key: number]:
Movie[] }>({});
  const [searchResults, setSearchResults] = useState<Movie[]>([]);
  const [searching, setSearching] = useState(false);

  useEffect(() => {
    const fetchGenresAndMovies = async () => {
      const fetchedGenres = await fetchGenres();
      setGenres(fetchedGenres);

      const movies: { [key: number]: Movie[] } = {};
      for (const genre of fetchedGenres) {
        const res = await fetchMoviesByGenre(genre.id);
        movies[genre.id] = res.results;
      }
      setMoviesByGenre(movies);
    };
```

```
  fetchGenresAndMovies();
}, []);

useEffect(() => {
  const delayDebounce = setTimeout(async () => {
    if (text.trim() === "") {
      setSearchResults([]);
      setSearching(false);
      return;
    }

    setSearching(true);
    const results = await searchMovies(text);
    setSearchResults(results);
    setSearching(false);
  }, 500);

  return () => clearTimeout(delayDebounce);
}, [text]);

return (
  <ScrollView style={styles.container}>
    <TextInput
      style={styles.input}
      placeholder="Пошук фільмів..."
      placeholderTextColor="#aaa"
      value={text}
      onChangeText={setText}
    />

    {searching ? (
      <Text style={styles.sectionTitle}>Шукаємо...</Text>
    ) : searchResults.length > 0 ? (
      <>
        <Text style={styles.sectionTitle}>Результати пошуку</Text>
        <FlatList
          data={searchResults}
          keyExtractor={(item) => item.id.toString()}
```

```
                renderItem={({ item }) => (
                  <TouchableOpacity
                    onPress={() =>
                      router.push({
                        pathname: "/movie/[id]",
                        params: { id: item.id.toString() },
                      })
                    }
                  >
                    <View style={styles.movieItem}>
                      <Image
                                                          source={{    uri:
`https://image.tmdb.org/t/p/w300${item.poster_path}` }}
                        style={styles.poster}
                      />
                      <View style={styles.movieInfo}>
                                                                  <Text
style={styles.movieTitle}>{item.title}</Text>
                        <Text style={styles.movieYear}>
                          {item.release_date ? item.release_date.slice(0,
4) : "—"}
                        </Text>
                        <Text style={styles.details}>
                          Оцінка: ⭐ {item.vote_average?.toFixed(1)} / 10
                        </Text>
                                    <Text  style={styles.readMore}>Дивитись
більше...</Text>
                      </View>
                    </View>
                  </TouchableOpacity>
                )}
                scrollEnabled={false}
                showsVerticalScrollIndicator={false}
              />
            </>
          ) : (
            genres.map((genre) => (
              <View key={genre.id}>
                <View style={styles.rowBetween}>
```

```
                    <Text style={styles.sectionTitle}>{genre.name}</Text>
                    <TouchableOpacity
                      onPress={() =>
                        router.push({
                          pathname: "/genre",
                          params: {
                            genreId: genre.id.toString(),
                            genreName: genre.name,
                          },
                        })
                      }
                    >
                      <Text style={styles.viewAll}>Дивитись все</Text>
                    </TouchableOpacity>
                  </View>

                  <FlatList
                    data={moviesByGenre[genre.id]}
                    horizontal
                    keyExtractor={(item) => item.id.toString()}
                    renderItem={({ item }) => (
                      <TouchableOpacity
                        onPress={() =>
                          router.push({
                            pathname: "/movie/[id]",
                            params: { id: item.id.toString() },
                          })
                        }
                      >
                                                <MovieCard   title={item.title}
posterPath={item.poster_path} />
                      </TouchableOpacity>
                    )}
                    showsHorizontalScrollIndicator={false}
                  />
                </View>
              ))
            )}
          </ScrollView>
```

```
    );
  }
  export default SearchScreen;
```

Файл «recommendationService.ts»

```
  import { collection, getDocs } from "firebase/firestore";
  import { db } from "../../config/firebase";
  import { fetchMovieDetails, fetchMovies } from "./movieService";

  // Отримати всі фільми з певного списку користувача
  export const getUserList = async (userId: string, listName: string)
=> {
        const snap = await getDocs(collection(db,
`users/${userId}/${listName}`));
    return snap.docs.map((doc) => ({ id: doc.id, ...doc.data() }));
  };

  // Перетворити фільм у вектор жанрів + рейтинг + рік
  const getMovieVector = (movie: any): number[] => {
    const genreVec: number[] = Array(20).fill(0);
    const genreIds: number[] = movie.genre_ids || [];

    genreIds.forEach((gid: number) => {
      if (gid >= 0 && gid < 20) genreVec[gid] = 1;
    });

    const rating: number = movie.vote_average || 5;
        const year: number = movie.release_date ?
parseInt(movie.release_date.slice(0, 4)) : 2000;

    return [...genreVec, rating, year];
  };

  // cosine similarity
  const cosineSimilarity = (a: number[], b: number[]): number => {
    const dot: number = a.reduce((sum: number, v: number, i: number) =>
sum + v * b[i], 0);
```

```
        const magA: number = Math.sqrt(a.reduce((sum: number, v: number) =>
sum + v * v, 0));
        const magB: number = Math.sqrt(b.reduce((sum: number, v: number) =>
sum + v * v, 0));
        return magA && magB ? dot / (magA * magB) : 0;
      };

      // Основна функція: рекомендації для авторизованих
      export const getSmartRecommendedMovies = async (userId: string):
Promise<any[]> => {
        const watched = await getUserList(userId, "watched");
        const watchedIds = new Set(watched.map((m: any) => Number(m.id)));

         const detailed = await Promise.all(watched.map((m: any) =>
fetchMovieDetails(Number(m.id))));
        if (!detailed.length) return [];

        const userVector: number[] = detailed
          .reduce((acc: number[], movie: any) => {
            const vec = getMovieVector(movie);
            return acc.map((v: number, i: number) => v + vec[i]);
          }, Array(22).fill(0))
          .map((sum: number) => sum / detailed.length);

        const allPopular: any[] = [];
        for (let page = 1; page <= 3; page++) {
          const res = await fetchMovies("popular", page);
          allPopular.push(...res);
        }

        const scored = allPopular
                  .filter((movie:     any)     =>     movie.poster_path     &&
!watchedIds.has(movie.id))
          .map((movie: any) => {
            const vec = getMovieVector(movie);
            const score = cosineSimilarity(userVector, vec);
            return { ...movie, score };
          });
```

```
  // фільтруємо унікальні id
  const uniqueMap = new Map();
  scored.forEach((m) => {
    if (!uniqueMap.has(m.id)) uniqueMap.set(m.id, m);
  });

  return Array.from(uniqueMap.values())
    .sort((a: any, b: any) => b.score - a.score)
    .slice(0, 20);
}

// Топ-10 для гостей
export const getGuestTopMovies = async (): Promise<any[]> => {
  const popular = await fetchMovies("popular");
  return popular.slice(0, 20);
};
```

Файл «[id].tsx»

```
import { Ionicons } from "@expo/vector-icons";
import { getAuth } from "firebase/auth";
import { deleteDoc, doc, getDoc, setDoc } from "firebase/firestore";
import { useEffect, useState } from "react";
import {
  ActivityIndicator,
  Alert,
  Image,
  Platform,
  ScrollView,
  StyleSheet,
  Text,
  TouchableOpacity,
  View
} from "react-native";
import { db } from "../../config/firebase";
import ReviewsSection from "../../src/components/ReviewsSection";
import { fetchMovieCredits, fetchMovieDetails } from
"../../src/services/movieService";
```

```
import { useLocalSearchParams, useRouter } from "expo-router";
const MovieDetails = () => {
  const { id } = useLocalSearchParams<{ id: string }>();
  const router = useRouter();
  const [movie, setMovie] = useState<any>(null);
  const [loading, setLoading] = useState(true);
  const [cast, setCast] = useState<any[]>([]);
  const [isInFavorites, setIsInFavorites] = useState(false);
  const [isInWatched, setIsInWatched] = useState(false);
  const [isInWatchLater, setIsInWatchLater] = useState(false);

  const goBack = () => {
  if (router.canGoBack?.()) {
    router.back();
  } else {
    router.replace("/");
  }
};

  useEffect(() => {
    const loadMovie = async () => {
      const data = await fetchMovieDetails(Number(id));
      const credits = await fetchMovieCredits(Number(id));
      setMovie(data);
      setCast(credits.cast.slice(0, 8));
      setLoading(false);
    };

    loadMovie();
  }, [id]);

    const handleAddToList = async (listName: "favorites" | "watched" |
"watchLater") => {
    console.log("⭕ Натиснуто кнопку списку:", listName);

    const user = getAuth().currentUser;
    console.log("👤 Поточний користувач:", user);
```

```
        if (!user) {
          console.log("⚠️ Користувач не авторизований — показуємо Alert");

          Alert.alert(
            "Потрібна авторизація",
              "Щоб зберігати фільми у списках, спочатку увійдіть у свій
акаунт.",
            [
              { text: "Скасувати", style: "cancel" },
              { text: "Увійти", onPress: () => router.push("/login") },
            ]
          );

          // Якщо на Web — Alert не працює. Використай заміну:
          if (Platform.OS === "web") {
            window.alert("Щоб зберігати фільми, увійдіть у свій акаунт");
          }

          return;
        }

        if (!id || !movie) {
          console.log("❌ Відсутній ID або дані про фільм");
          return;
        }

        const ref = doc(db, "users", user.uid, listName, id.toString());
        const snap = await getDoc(ref);

        const toggle = {
          favorites: () => setIsInFavorites(!snap.exists()),
          watched: () => setIsInWatched(!snap.exists()),
          watchLater: () => setIsInWatchLater(!snap.exists()),
        };

        if (snap.exists()) {
          console.log("🗑️ Фільм уже в списку — видаляємо");
          await deleteDoc(ref);
```

```
        toggle[listName]();
        Alert.alert("Видалено", "Фільм прибрано зі списку.");
      } else {
        console.log("✅ Додаємо фільм до списку:", listName);
        await setDoc(ref, {
          title: movie.title,
          poster: movie.poster_path,
          addedAt: new Date(),
        });
        toggle[listName]();
        Alert.alert("Додано", `Фільм додано до списку ${listName}.`);
      }
    };

    if (loading || !movie) {
      return (
        <View style={styles.loading}>
          <ActivityIndicator size="large" color="#7B61FF" />
        </View>
      );
    }

    return (
      <ScrollView style={styles.container}>
        <View style={styles.headerRow}>
          <TouchableOpacity onPress={goBack} style={styles.backIcon}>
            <Ionicons name="arrow-back" size={24} color="#7B61FF" />
          </TouchableOpacity>
          <Text style={styles.title}>{movie.title}</Text>
        </View>

        <Image
                                                source={{          uri:
`https://image.tmdb.org/t/p/w500${movie.poster_path}` }}
          style={styles.poster}
        />

        <Text style={styles.year}>
          {movie.release_date ? movie.release_date.slice(0, 4) : "—"}
```

```
            </Text>

            <Text style={styles.sectionTitle}>Актори</Text>
            <Text style={styles.castList}>
              {cast.map((actor: any) => actor.name).join(", ")}
            </Text>

            <Text style={styles.sectionTitle}>Рейтинг</Text>
             <Text style={styles.rating}>⭐ {movie.vote_average.toFixed(1)}
/10</Text>

            <Text style={styles.sectionTitle}>Опис</Text>
            <Text style={styles.overview}>{movie.overview}</Text>

            <Text style={styles.sectionTitle}>Додати в список:</Text>
            <View style={styles.verticalListButtons}>
                 <TouchableOpacity style={styles.listItem} onPress={() =>
handleAddToList("favorites")}>
                <Ionicons name={isInFavorites ? "heart" : "heart-outline"}
size={24} color="#7B61FF" />
              <Text style={styles.listText}>Улюблене</Text>
            </TouchableOpacity>

                 <TouchableOpacity style={styles.listItem} onPress={() =>
handleAddToList("watched")}>
                   <Ionicons name={isInWatched ? "eye" : "eye-outline"}
size={24} color="#7B61FF" />
              <Text style={styles.listText}>Переглянуте</Text>
            </TouchableOpacity>

                 <TouchableOpacity style={styles.listItem} onPress={() =>
handleAddToList("watchLater")}>
                 <Ionicons name={isInWatchLater ? "time" : "time-outline"}
size={24} color="#7B61FF" />
              <Text style={styles.listText}>Переглянути пізніше</Text>
            </TouchableOpacity>
            </View>
```

```
            <ReviewsSection movieId={id as string} movieTitle={movie.title}
/>
          </ScrollView>
        );
      };

      export default MovieDetails;
```

Файл «register.tsx»

```
      import { useRouter } from "expo-router";
      import { createUserWithEmailAndPassword } from "firebase/auth";
      import { doc, setDoc } from "firebase/firestore";
      import React, { useState } from "react";
      import {
        ActivityIndicator,
        Alert,
        StyleSheet,
        Text,
        TextInput,
        TouchableOpacity,
        View,
      } from "react-native";
      import { auth, db } from "../config/firebase";

      function RegisterScreen() {
        const router = useRouter();
        const [email, setEmail] = useState("");
        const [password, setPassword] = useState("");
        const [confirmPassword, setConfirmPassword] = useState("");
        const [loading, setLoading] = useState(false);

        const handleRegister = async () => {
          if (!email || !password || !confirmPassword) {
            Alert.alert("Помилка", "Заповни всі поля");
            return;
          }

          if (password !== confirmPassword) {
```

```
            Alert.alert("Помилка", "Паролі не співпадають");
            return;
          }

          setLoading(true);
          try {
              const userCred = await createUserWithEmailAndPassword(auth,
email, password);

            // Створюємо профіль у Firestore
            await setDoc(doc(db, "users", userCred.user.uid), {
              displayName: "Користувач",
              photoURL: "https://i.imgur.com/BoN9kdC.png",
            });

            Alert.alert("Успіх", "Акаунт створено!");
            router.replace("/");
          } catch (err: any) {
            Alert.alert("Помилка реєстрації", err.message);
          } finally {
            setLoading(false);
          }
        };

        return (
          <View style={styles.container}>
            <Text style={styles.title}>Реєстрація</Text>

            <TextInput
              style={styles.input}
              placeholder="Email"
              placeholderTextColor="#aaa"
              value={email}
              onChangeText={setEmail}
            />
            <TextInput
              style={styles.input}
              placeholder="Пароль"
              secureTextEntry
```

```
            placeholderTextColor="#aaa"
            value={password}
            onChangeText={setPassword}
          />
          <TextInput
            style={styles.input}
            placeholder="Підтвердити пароль"
            secureTextEntry
            placeholderTextColor="#aaa"
            value={confirmPassword}
            onChangeText={setConfirmPassword}
          />

                              <TouchableOpacity       style={styles.button}
onPress={handleRegister} disabled={loading}>
            {loading ? (
              <ActivityIndicator color="#fff" />
            ) : (
              <Text style={styles.buttonText}>Зареєструватись</Text>
            )}
          </TouchableOpacity>

          <TouchableOpacity onPress={() => router.push("/login")}>
            <Text style={styles.registerLink}>
              Вже є акаунт?{" "}
              <Text style={styles.registerLinkHighlight}>Увійти</Text>
            </Text>
          </TouchableOpacity>
        </View>
      );
    }
    export default RegisterScreen;
```

Файл «login.tsx»

```
import { useRouter } from "expo-router";
import { signInWithEmailAndPassword } from "firebase/auth";
import React, { useState } from "react";
```

```
import { Alert, StyleSheet, Text, TextInput, TouchableOpacity, View } from "react-native";
import { auth } from "../config/firebase";

function LoginScreen() {
  const router = useRouter();
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");

  const handleLogin = async () => {
    if (!email || !password) return;
    try {
      await signInWithEmailAndPassword(auth, email, password);
      router.replace("/");
    } catch (err: any) {
      Alert.alert("Помилка входу", err.message);
    }
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Вхід у FilmBuddy</Text>

      <TextInput style={styles.input} placeholder="Email" placeholderTextColor="#aaa" value={email} onChangeText={setEmail} />
      <TextInput style={styles.input} placeholder="Пароль" secureTextEntry placeholderTextColor="#aaa" value={password} onChangeText={setPassword} />

      <TouchableOpacity style={styles.button} onPress={handleLogin}>
        <Text style={styles.buttonText}>Увійти</Text>
      </TouchableOpacity>

      <TouchableOpacity onPress={() => router.push("/register")}>
  <Text style={styles.registerLink}>
                    Немає акаунта? <Text style={styles.registerLinkHighlight}>Зареєструйся</Text>
  </Text>
</TouchableOpacity>
```

```
      </View>
    );
  }
  export default LoginScreen;
```

Файл «favorites.tsx»

```
  // Favorites.tsx
  import { Ionicons } from "@expo/vector-icons";
  import { useRouter } from "expo-router";
  import { collection, getDocs } from "firebase/firestore";
  import React, { useEffect, useState } from "react";
  import { FlatList, StyleSheet, Text, TouchableOpacity, View, } from
"react-native";
  import { auth, db } from "../config/firebase";
  import MovieListCard from "../src/components/MovieListCard";

  const Favorites = () => {
    const [movies, setMovies] = useState<any[]>([]);
    const router = useRouter();

    useEffect(() => {
      const fetchFavorites = async () => {
        const user = auth.currentUser;
        if (!user) return;
            const snapshot = await getDocs(collection(db, "users",
user.uid, "favorites"));
            const list = snapshot.docs.map((doc) => ({ id: doc.id,
...doc.data() }));
        setMovies(list);
      };
      fetchFavorites();
    }, []);

    return (
      <View style={styles.container}>
          <View style={styles.headerRow}>
                  <TouchableOpacity onPress={() => router.back()}
style={styles.backIcon}>
```

```
          <Ionicons name="arrow-back" size={24} color="#7B61FF" />
        </TouchableOpacity>
      <Text style={styles.title}>Улюблені фільми</Text>
      </View>
      <FlatList
        data={movies}
        keyExtractor={(item) => item.id}
        renderItem={({ item }) => (
            <MovieListCard
            id={item.id}
            title={item.title}
            poster={item.poster}
            rating={item.rating || 6.5}
            year={item.year || "2023"}
            />
        )}
        />
    </View>
  );
};

export default Favorites;
```

Файл «MovieCard.tsx»

```
import React from "react";
import { Image, StyleSheet, Text, View } from "react-native";

interface MovieCardProps {
  title: string;
  posterPath: string;
}

const MovieCard = ({ title, posterPath }: MovieCardProps) => {
  return (
    <View style={styles.card}>
      <Image
```

```
            source={{ uri: `https://image.tmdb.org/t/p/w500${posterPath}`
}}
            style={styles.image}
          />
          <Text style={styles.title} numberOfLines={2}>{title}</Text>
        </View>
      );
    };

    const styles = StyleSheet.create({
      card: {
        width: 140,
        marginRight: 12,
      },
      image: {
        width: '100%',
        height: 210,
        borderRadius: 12,
      },
      title: {
        color: '#fff',
        fontSize: 14,
        marginTop: 6,
        fontWeight: '600',
      },
    });

    export default MovieCard;
```

Файл «ReviewSection.tsx»

```
import { useRouter } from "expo-router";
import { getAuth } from "firebase/auth";
import {
    addDoc,
    collection,
    deleteDoc,
    doc,
    getDoc,
```

```
        increment,
        onSnapshot,
        query,
        serverTimestamp,
        setDoc,
        updateDoc,
        where,
    } from "firebase/firestore";
    import React, { useEffect, useState } from "react";
    import {
        FlatList,
        Image,
        StyleSheet,
        Text,
        TextInput,
        TouchableOpacity,
        View,
    } from "react-native";
    import { db } from "../../config/firebase";

    const ReviewsSection = ({ movieId, movieTitle }: { movieId: string;
movieTitle: string }) => {
      const [text, setText] = useState("");
      const [replyTo, setReplyTo] = useState<string | null>(null);
      const [reviews, setReviews] = useState<any[]>([]);
       const [expandedReplies, setExpandedReplies] = useState<{ [key:
string]: boolean }>({});
      const user = getAuth().currentUser;
      const router = useRouter();

      useEffect(() => {
        const q = query(collection(db, "reviews"), where("movieId", "==",
movieId));
        const unsubscribe = onSnapshot(q, async (snapshot) => {
          const list = await Promise.all(
            snapshot.docs.map(async (docSnap) => {
              const data = docSnap.data();
                    const userSnap = await getDoc(doc(db, "users",
data.userId));
```

```
            const userData = userSnap.exists() ? userSnap.data() : {};
            return {
              id: docSnap.id,
              ...data,
              userName: userData.userName || "anon",
              avatar: userData.photoURL || null,
            };
          })
        );
        setReviews(list);
      });
      return () => unsubscribe();
    }, [movieId]);

    const handleAddReview = async () => {
      if (!text.trim() || !user) return;

      await addDoc(collection(db, "reviews"), {
        movieId,
        movieTitle,
        text,
        userId: user.uid,
        timestamp: serverTimestamp(),
        likes: 0,
        dislikes: 0,
        replyTo,
      });

      setText("");
      setReplyTo(null);
    };

     const handleVote = async (reviewId: string, type: "like" |
"dislike") => {
      if (!user) {
        router.push("/login");
        return;
      }
```

```
  const voteRef = doc(db, "reviews", reviewId, "votes", user.uid);
  const voteSnap = await getDoc(voteRef);

  if (voteSnap.exists()) {
    const prev = voteSnap.data().type;
    if (prev === type) {
      await deleteDoc(voteRef);
      await updateDoc(doc(db, "reviews", reviewId), {
        [type === "like" ? "likes" : "dislikes"]: increment(-1),
      });
    } else {
      await setDoc(voteRef, { type });
      await updateDoc(doc(db, "reviews", reviewId), {
        likes: increment(type === "like" ? 1 : -1),
        dislikes: increment(type === "dislike" ? 1 : -1),
      });
    }
  } else {
    await setDoc(voteRef, { type });
    await updateDoc(doc(db, "reviews", reviewId), {
      [type === "like" ? "likes" : "dislikes"]: increment(1),
    });
  }
};

const handleDelete = async (id: string) => {
  if (user && confirm("Видалити відгук?")) {
    await deleteDoc(doc(db, "reviews", id));
  }
};

const toggleReplies = (id: string) => {
  setExpandedReplies((prev) => ({ ...prev, [id]: !prev[id] }));
};

const renderItem = ({ item }: { item: any }) => {
  const replies = reviews.filter((r) => r.replyTo === item.id);
  const expanded = expandedReplies[item.id];
```

```
          return (
            <View style={styles.review}>
              <View style={styles.userRow}>
                    {item.avatar && <Image source={{ uri: item.avatar }}
style={styles.avatar} />}
                <Text style={styles.user}>{item.userName}</Text>
              </View>
              <Text style={styles.reviewText}>{item.text}</Text>

              <View style={styles.actions}>
                     <TouchableOpacity onPress={() => handleVote(item.id,
"like")}>
                  <Text style={styles.like}>👍 {item.likes}</Text>
                </TouchableOpacity>
                     <TouchableOpacity onPress={() => handleVote(item.id,
"dislike")}>
                  <Text style={styles.dislike}>👎 {item.dislikes}</Text>
                </TouchableOpacity>
                {user && (
                  <>
                    <TouchableOpacity onPress={() => setReplyTo(item.id)}>
                      <Text style={styles.replyButton}>Відповісти</Text>
                    </TouchableOpacity>
                    {user.uid === item.userId && (
                                        <TouchableOpacity onPress={() =>
handleDelete(item.id)}>
                        <Text style={styles.replyButton}>Видалити</Text>
                      </TouchableOpacity>
                    )}
                  </>
                )}
              </View>

              {replies.length > 0 && (
                <TouchableOpacity onPress={() => toggleReplies(item.id)}>
                  <Text style={styles.replyButton}>
                    {expanded ? "Приховати відповіді" : `Показати відповіді
(${replies.length})`}
```

```
                </Text>
              </TouchableOpacity>
            )}

      {reviews
        .filter((r) => r.replyTo === item.id)
        .map((reply) =>
          React.createElement(
            View,
            { style: styles.reply, key: String(reply.id) },
            <>
              <View style={styles.userRow}>
                {reply.avatar && (
                              <Image   source={{   uri:   reply.avatar   }}
style={styles.avatar} />
                )}
                <Text style={styles.user}>{reply.userName}</Text>
              </View>
              <Text style={styles.reviewText}>{reply.text}</Text>
            </>
          )
        )}
            </View>
          );
        };

        return (
          <View style={styles.container}>
            <Text style={styles.title}>Відгуки</Text>

            {user ? (
              <>
                 {replyTo && <Text style={styles.replyingTo}>Відповідаєте на
коментар</Text>}
                <TextInput
                  style={styles.input}
                  value={text}
                  onChangeText={setText}
                  placeholder="Напишіть відгук..."
```

```
                placeholderTextColor="#999"
              />
                                <TouchableOpacity    style={styles.button}
onPress={handleAddReview}>
                <Text style={styles.buttonText}>Залишити</Text>
              </TouchableOpacity>
            </>
          ) : (
              <TouchableOpacity style={styles.loginPrompt} onPress={() =>
router.push("/login")}>
               <Text style={styles.loginPromptText}>Увійдіть, щоб залишити
відгук</Text>
            </TouchableOpacity>
          )}

          <FlatList
            data={reviews.filter((r) => !r.replyTo)}
            keyExtractor={(item) => item.id}
            renderItem={renderItem}
          />
        </View>
      );
    };

    export default ReviewsSection;
```

Файл «MovieService.ts»

```
    import { API_KEY, BASE_URL } from "../../config/apiConfig";

    export const fetchMovies = async (endpoint: string, page: number = 1)
=> {
      const response = await fetch(
          `${BASE_URL}/movie/${endpoint}?api_key=${API_KEY}&language=uk-
UA&page=${page}`
      );
      const data = await response.json();
      return data.results || [];
    };
```

```
    // Отримати фільми за жанром з підтримкою пагінації
    export const fetchMoviesByGenre = async (genreId: number, page:
number = 1) => {
      const response = await fetch(

`${BASE_URL}/discover/movie?with_genres=${genreId}&api_key=${API_KEY}&langu
age=uk-UA&page=${page}`
      );
      const data = await response.json();
      return data; // повертає: { page, results, total_pages }
    };

    // Отримати список усіх жанрів
    export const fetchGenres = async () => {
      const response = await fetch(
        `${BASE_URL}/genre/movie/list?api_key=${API_KEY}&language=uk-UA`
      );
      const data = await response.json();
      return data.genres; // масив жанрів: [{ id: 28, name: "Бойовики" },
...]
    };

    //Отримати детальну інформацію про фільм
    export const fetchMovieDetails = async (id: number) => {
      const response = await fetch(
        `${BASE_URL}/movie/${id}?api_key=${API_KEY}&language=uk-UA`
      );
      const data = await response.json();
      return data;
    };

    //Отримати імена акторів
    export const fetchMovieCredits = async (movieId: number) => {
      const response = await fetch(

`${BASE_URL}/movie/${movieId}/credits?api_key=${API_KEY}&language=uk-UA`
      );
      const data = await response.json();
```

```
    return data;
  };

  export const searchMovies = async (query: string) => {
    try {
      const response = await fetch(

`${BASE_URL}/search/movie?api_key=${API_KEY}&query=${encodeURIComponent(que
ry)}&language=uk-UA`
      );
      const data = await response.json();
      return data.results;
    } catch (error) {
      console.error("Помилка пошуку фільмів:", error);
      return [];
    }
  };
```

ДОДАТОК В – Диск із кваліфікаційною роботою бакалавра