ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка Android-застосунку онлайн-кінотеатру з автоматизованим збором контенту»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Денис ЦУРКАН
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Денис ЦУРКАН

Керівник: Тимур ДОВЖЕНКО
канд. техн. наук

Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Цуркану Денису Олександровичу

1. Тема кваліфікаційної роботи: «Розробка Android-застосунку онлайн-кінотеатру з автоматизованим збором контенту»
керівник кваліфікаційної роботи канд. техн. наук Тимур ДОВЖЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: Вихідні дані до кваліфікаційної роботи: теоретичні відомості про Android-застосунки онлайн-кінотеатрів, клієнт-серверну архітектуру, REST API, бази даних PostgreSQL, автоматизований збір контенту, а також матеріали та програмний код системи KinoFox.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної галузі онлайн-кінотеатрів, потреб користувачів, існуючих програмних рішень і вимог до Android-застосунку.
2. Обґрунтування засобів реалізації Android-застосунку онлайн-кінотеатру, серверної частини, бази даних і модуля автоматизованого збору контенту.
3. Проєктування та реалізація системи KinoFox, зокрема клієнтської частини, REST API, бази даних і модуля автоматизованого збору контенту.
4. Тестування Android-застосунку KinoFox та аналіз отриманих результатів.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Загальна архітектура системи.
5. Діаграма варіантів використання.
6. Діаграма послідовності автоматизованого збору контенту.
7. Екранні форми Android-застосунку.
8. Результати тестування.

6. Дата видачі завдання «23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	23.02-28.02.2026	
2	Аналіз та дослідження існуючих аналогів	01.03-07.03.2026	
3	Огляд існуючих алгоритмів обробки тексту та методів і технологій побудови тематичного словника	08.03-20.03.2026	
4	Проектування застосунку для автоматизованої побудови тематичного словника навчальної дисципліни	21.03-31.03.2026	
5	Програмна реалізація застосунку	01.04-20.04.2026	
6	Тестування застосунку	21.04-30.04.2026	
7	Оформлення роботи: вступ, висновки, реферат	01.05-07.05.2026	
8	Розробка демонстраційних матеріалів	08.05-11.05.2026	
9	Попередній захист роботи	12.05-22.05.2026	
10	Подання квалікафінцої роботи	23.05-30.05.2026	

Здобувач вищої освіти _____ Денис ЦУРКАН
(підпис)

Керівник
кваліфікаційної роботи _____ Тимур ДОВЖЕНКО
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 88 стор., 17 табл., 14 рис., 23 джерела.

Мета роботи – підвищення зручності, оперативності та актуальності доступу користувачів до медіаконтенту шляхом створення Android-застосунку онлайн-кінотеатру з механізмами автоматизованого збору та оновлення контенту.

Об'єкт дослідження – процес надання користувачам доступу до каталогу відеоконтенту в Android-застосунку онлайн-кінотеатру з автоматизованим оновленням метаданих.

Предмет дослідження – програмне забезпечення для реалізації Android-застосунку онлайн-кінотеатру з клієнтською, серверною та сервісною підсистемами автоматизованого збору контенту.

Короткий зміст роботи: У роботі проаналізовано особливості Android-застосунків онлайн-кінотеатрів, потреби користувачів, проблему формування відеокаталогу та існуючі програмні рішення. Обґрунтовано вибір клієнт-серверної архітектури, Flutter, Dart, Bun, Elysia, TypeScript, PostgreSQL, pgvector і Python. Спроєктовано та реалізовано систему KinoFox, що містить мобільний клієнт, REST API, базу даних і модуль автоматизованого збору контенту. Проведено тестування основних функціональних можливостей застосунку.

Сфера використання – цифрові медіасервіси, онлайн-кінотеатри, комерційні платформи для розповсюдження відеоконтенту.

КЛЮЧОВІ СЛОВА: ANDROID-ЗАСТОСУНОК, ОНЛАЙН-КІНОТЕАТР, KINOFOX, FLUTTER, DART, REST API, POSTGRESQL, PGVECTOR, АВТОМАТИЗОВАНИЙ ЗБІР КОНТЕНТУ

ЗМІСТ

ВСТУП

Сучасний розвиток мобільних технологій суттєво змінив підхід користувачів до споживання мультимедійного контенту. Якщо раніше перегляд фільмів, серіалів і мультфільмів переважно був пов’язаний із телевізійним ефіром або персональним комп’ютером, то сьогодні значна частина користувачів надає перевагу мобільним пристроям. Смартфон став зручним інструментом для швидкого доступу до відеоконтенту, пошуку потрібного фільму, перегляду опису, рейтингу, жанру, країни виробництва та запуску відтворення без необхідності використовувати додаткові пристрої.

Особливої актуальності набувають Android-застосунки онлайн-кінотеатрів, оскільки операційна система Android широко використовується на мобільних пристроях різних цінових категорій і забезпечує доступ до цифрових сервісів великій кількості користувачів. Для онлайн-кінотеатру важливо не лише надати можливість перегляду контенту, а й забезпечити зручну навігацію, пошук, персоналізацію, роботу з обліковим записом, збереження історії переглядів і формування списку обраного. Такі функції впливають на зручність користування застосунком і дозволяють створити цілісну систему взаємодії користувача з відеокаталогом.

Актуальність теми також зумовлена тим, що одним із найбільш трудомістких процесів у роботі онлайн-кінотеатру є наповнення каталогу. Для кожного фільму або серіалу необхідно зібрати назву, постер, рейтинг, рік випуску, країну, жанри, тривалість, опис, зображення та посилання на відтворення. Якщо виконувати цей процес вручну, він потребує значних часових витрат і ускладнює регулярне оновлення каталогу. Тому доцільним є використання автоматизованого збору контенту, який дозволяє прискорити наповнення бази даних, зменшити кількість однотипних дій адміністратора та підтримувати актуальність інформації.

У межах цієї роботи розглядається розробка Android-застосунку онлайн-кінотеатру KinoFox, який поєднує мобільний клієнт, серверну частину, базу даних і модуль автоматизованого збору контенту. Клієнтська частина застосунку реалізується з використанням Flutter, що дає змогу створити зручний інтерфейс для перегляду каталогу, сторінок фільмів, авторизації користувача та взаємодії з основними функціями системи. Серверна частина відповідає за обробку запитів, роботу з користувачами, фільмами, історією переглядів, обраним і пошуком. Для збереження даних використовується PostgreSQL, а для автоматизованого збору інформації про контент — окремий Python-модуль.

Аналіз останніх досліджень і публікацій показує, що питання розробки мобільних застосунків, клієнт-серверних систем, REST API, баз даних, автоматизованого збору даних і пошукових механізмів є актуальними в сучасній інженерії програмного забезпечення. У наукових і технічних джерелах значна увага приділяється побудові масштабованих мобільних систем, організації взаємодії між клієнтською та серверною частинами, захисту користувацьких сесій, роботі з реляційними базами даних і застосуванню алгоритмів пошуку [1-4]. Окремий напрям становлять дослідження та практичні матеріали, присвячені автоматизованому збору даних із відкритих джерел, їх структуризації, очищенню та подальшому використанню в інформаційних системах [5-7].

Водночас більшість існуючих рішень у сфері онлайн-кінотеатрів орієнтована на готові комерційні платформи, де внутрішні механізми збору, обробки та збереження контенту залишаються закритими для користувача. Це створює потребу в розробці навчального програмного рішення, у якому можна простежити повний цикл роботи системи: від автоматизованого отримання даних до їх збереження в базі, обробки сервером і відображення в Android-застосунку. Саме такий підхід дозволяє не лише реалізувати приклад онлайн-кінотеатру, а й продемонструвати практичне застосування сучасних технологій мобільної та серверної розробки.

Метою кваліфікаційної роботи є підвищення зручності, оперативності та актуальності доступу користувачів до медіаконтенту шляхом створення Android-

застосунку онлайн-кінотеатру з механізмами автоматизованого збору та оновлення контенту.

Для досягнення поставленої мети необхідно виконати такі завдання:

1. проаналізувати особливості функціонування Android-застосунків онлайн-кінотеатрів;
2. обґрунтувати вибір клієнт-серверної архітектури та технологічного стеку;
3. спроєктувати загальну архітектуру системи, структуру бази даних, серверну частину та REST API KinoFox;
4. реалізувати клієнтську частину Android-застосунку;
5. реалізувати серверну частину для обробки запитів, авторизації та роботи з даними;
6. реалізувати модуль автоматизованого збору контенту;
7. провести тестування основних функціональних можливостей застосунку.

Об’єктом дослідження є процес надання користувачам доступу до каталогу відеоконтенту в Android-застосунку онлайн-кінотеатру з автоматизованим оновленням метаданих.

Предметом дослідження є програмне забезпечення для реалізації Android-застосунку онлайн-кінотеатру з клієнтською, серверною та сервісною підсистемами автоматизованого збору контенту.

Методи дослідження та розроблення включають аналіз предметної галузі, порівняльний аналіз існуючих програмних рішень, методи проєктування клієнт-серверних систем, моделювання структури бази даних, об’єктно-орієнтований підхід до програмування, методи розробки REST API, методи автоматизованого збору та обробки даних, а також функціональне тестування програмного забезпечення. Для реалізації клієнтської частини використано Flutter і Dart, для серверної частини — Bun, Elysia та TypeScript, для збереження даних — PostgreSQL, для роботи з векторними представленнями — pgvector, а для автоматизованого збору контенту — Python.

Практичне значення роботи полягає у створенні програмної системи, яка може бути використана як основа для мобільного онлайн-кінотеатру з автоматизованим наповненням каталогу. Розроблений застосунок дозволяє користувачу переглядати список фільмів, відкривати детальну інформацію про вибраний контент, працювати з обліковим записом, зберігати сесію, а також використовувати функції історії та обраного. Автоматизований збір контенту зменшує потребу в ручному введенні даних і спрощує підтримку актуального стану каталогу.

Науково-практична новизна роботи полягає у поєднанні мобільного Android-застосунку, серверної частини, реляційної бази даних, векторного пошуку та окремого модуля автоматизованого збору контенту в межах єдиної системи онлайн-кінотеатру. Особливістю розробки є те, що система охоплює повний цикл роботи з контентом: від отримання та структуризації даних до їх збереження, пошуку й відображення користувачу через мобільний інтерфейс.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку посилань і додатків.

У першому розділі розглянуто предметну галузь онлайн-кінотеатрів, особливості Android-застосунків для перегляду відеоконтенту, потреби користувачів, проблему наповнення відеокаталогу, існуючі програмні рішення та вимоги до розроблюваного застосунку.

У другому розділі обґрунтовано вибір засобів реалізації Android-застосунку онлайн-кінотеатру, зокрема клієнт-серверної архітектури, Flutter, Dart, Bun, Elysia, TypeScript, PostgreSQL, pgvector і Python-засобів для автоматизованого збору контенту.

У третьому розділі описано проєктування та реалізацію системи KinoFox, зокрема загальну архітектуру, клієнтську частину, серверну частину, базу даних, REST API та модуль автоматизованого збору контенту.

У четвертому розділі наведено організацію тестування застосунку, перевірку основних функціональних можливостей, аналіз результатів тестування та методику користування Android-застосунком KinoFox.

У висновках узагальнено результати виконання кваліфікаційної роботи та визначено відповідність отриманих результатів поставленій меті й завданням.У третьому розділі описано проєктування та реалізацію системи KinoFox, зокрема загальну архітектуру, клієнтську частину, серверну частину, базу даних, REST API та модуль автоматизованого збору контенту.

У четвертому розділі наведено організацію тестування застосунку, перевірку основних функціональних можливостей, аналіз результатів тестування та методику користування Android-застосунком KinoFox.

У висновках узагальнено результати виконання кваліфікаційної роботи та визначено відповідність отриманих результатів поставленій меті й завданням.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ ДЛЯ ОНЛАЙН-КІНОТЕАТРІВ

1.1 Особливості функціонування Android-застосунків онлайн-кінотеатрів

Онлайн-кінотеатри стали звичним способом перегляду фільмів, серіалів, мультфільмів та іншого відеоконтенту. Користувач уже не залежить від телевізійної програми, часу трансляції або наявності завантажених файлів на пристрої. Достатньо відкрити застосунок, знайти потрібний фільм у каталозі, ознайомитися з його описом і запустити перегляд. Саме тому мобільні застосунки для онлайн-кінотеатрів сьогодні виконують не лише роль програвача, а й роль повноцінного сервісу для пошуку, вибору, збереження та організації відеоконтенту.

Особливо важливим напрямом є розробка застосунків для Android, оскільки ця операційна система використовується на великій кількості смартфонів і планшетів. Android-пристрої можуть відрізнятися за розміром екрана, продуктивністю, версією системи та якістю інтернет-з’єднання. Через це мобільний онлайн-кінотеатр має бути достатньо гнучким: коректно відображати інтерфейс на різних екранах, швидко реагувати на дії користувача, не перевантажувати пристрій і стабільно працювати з мережевими запитами. Для такого типу програмного забезпечення важливо не тільки реалізувати сам факт доступу до відео, а й зробити взаємодію з каталогом зрозумілою та зручною.

У більшості випадків Android-застосунок онлайн-кінотеатру є клієнтською частиною ширшої програмної системи. Він не зберігає весь каталог локально, а отримує потрібні дані із серверу. Наприклад, коли користувач відкриває головний екран, застосунок надсилає запит до API, отримує список фільмів або серіалів і показує їх у вигляді карток. Якщо користувач переходить на сторінку конкретного фільму, клієнтська частина знову звертається до серверу, щоб отримати детальну

інформацію: назву, постер, рейтинг, рік випуску, країну, жанри, тривалість, опис і доступні посилання для перегляду. Такий підхід дозволяє розділити інтерфейс і логіку обробки даних, що спрощує підтримку та подальший розвиток системи.

Важливим елементом онлайн-кінотеатру є каталог відеоконтенту. Для користувача саме каталог є основною частиною застосунку, бо через нього відбувається пошук і вибір фільму. Якщо інформація подана хаотично або сторінки довго завантажуються, користувачеві складно орієнтуватися в сервісі. Тому каталог має бути структурованим і візуально зрозумілим. Зазвичай контент подається у вигляді сітки або окремих тематичних блоків, де кожна позиція має постер, назву, рейтинг і короткі характеристики. Такий спосіб подання добре підходить для мобільного екрана, оскільки дає можливість швидко переглядати багато об’єктів без надмірного текстового навантаження.

Для Android-застосунків онлайн-кінотеатрів велике значення має візуальна складова. Користувач часто обирає фільм не лише за назвою, а й за постером, жанром, рейтингом або коротким описом. Тому інтерфейс повинен допомагати швидко оцінити контент. Надмірна кількість дрібного тексту або невиразні елементи керування ускладнюють роботу із застосунком. Натомість картки фільмів, великі постери, чіткі кнопки та зрозуміла навігація роблять сервіс простішим у використанні. Для онлайн-кінотеатрів також характерне використання темної теми інтерфейсу, оскільки вона краще поєднується з медіаконтентом і є зручнішою під час перегляду у вечірній час.

Окрему роль відіграє сторінка детального перегляду фільму. Вона повинна дати користувачеві достатньо інформації для прийняття рішення щодо перегляду. На такій сторінці доцільно розміщувати постер або фонове зображення, назву, рейтинг, рік випуску, тривалість, жанри, країну виробництва та сюжетний опис. Також саме тут має бути доступна дія, пов’язана із запуском перегляду або вибором джерела відтворення. Фактично сторінка фільму є проміжною ланкою між каталогом і переглядом, тому її структура повинна бути логічною та не перевантаженою.

Ще однією важливою складовою є авторизація користувача. Без облікового запису онлайн-кінотеатр може працювати лише як відкритий каталог, але персоналізовані функції в такому разі обмежені. Реєстрація та вхід до системи дають змогу зберігати історію переглядів, формувати список обраного, показувати дані профілю та підтримувати користувацьку сесію між запусками застосунку. Для мобільного сервісу це особливо важливо, оскільки користувач очікує, що після повторного відкриття програми йому не доведеться щоразу вводити логін і пароль. Тому застосунок має зберігати токен сесії локально та використовувати його для подальшої взаємодії із сервером.

Історія переглядів і список обраного підвищують зручність роботи з відеокаталогом. Історія дозволяє повернутися до фільмів, які користувач уже відкривав або переглядав, а обране - зберегти контент для подальшого перегляду. Такі функції здаються додатковими, але на практиці вони формують відчуття персонального сервісу. Для їх реалізації необхідно пов’язати дії користувача з його обліковим записом, передати відповідні дані на сервер і зберегти їх у базі даних. Отже, навіть прості на вигляд функції потребують правильної організації взаємодії між мобільним клієнтом, API та серверною частиною.

Пошук є ще одним важливим механізмом для онлайн-кінотеатру. Якщо каталог містить багато фільмів і серіалів, користувач не завжди готовий переглядати всі позиції вручну. Тому застосунок повинен підтримувати швидке знаходження контенту за назвою або іншими характеристиками. У більш розвинених системах можна використовувати не тільки звичайний текстовий пошук, а й семантичний, який враховує зміст запиту. Це корисно тоді, коли користувач не пам’ятає точну назву фільму або формулює запит описово. У такому випадку система може повертати не лише точні збіги, а й близькі за змістом результати.

Технічно Android-застосунок онлайн-кінотеатру має стабільно працювати з мережею. Під час отримання даних можуть виникати різні ситуації: повільне з’єднання, помилка серверу, відсутність відповіді, порожній список або некоректні дані. Якщо ці стани не обробити, користувач може побачити порожній екран або

застосунок працюватиме нестабільно. Тому під час розробки важливо передбачити індикатори завантаження, повідомлення про помилки, перевірку отриманих даних і коректне оновлення інтерфейсу після відповіді серверу.

Не менш важливою є продуктивність. Онлайн-кінотеатр працює з великою кількістю зображень: постерами, кадрами, фоновими ілюстраціями. Якщо завантажувати їх без оптимізації, це може сповільнити роботу застосунку та збільшити використання мобільного інтернету. Тому інтерфейс повинен бути побудований так, щоб завантаження контенту не створювало помітних затримок. Для цього використовуються списки з поступовим відображенням елементів, кешування зображень, обмеження кількості даних в одному запиті та оптимізована робота з API.

У системі KinoFox Android-застосунок виконує роль мобільного клієнта онлайн-кінотеатру. Він забезпечує доступ до головної сторінки, перегляд добірок фільмів, серіалів і мультфільмів, відкриття детальної інформації про контент, авторизацію, реєстрацію та роботу з профілем користувача. Застосунок взаємодіє із серверною частиною через API, а дані про фільми зберігаються в базі даних. Така структура дозволяє не прив’язувати мобільний клієнт до статичного набору даних, а оновлювати каталог через серверну частину та модуль автоматизованого збору контенту.

Таким чином, Android-застосунок онлайн-кінотеатру є не просто інтерфейсом для перегляду відео. Це частина комплексної системи, у якій поєднуються каталог контенту, серверна логіка, база даних, авторизація, пошук, персональні функції та відтворення медіа. Для якісної роботи такого застосунку необхідно враховувати особливості мобільних пристроїв, зручність інтерфейсу, швидкість завантаження даних, стабільність API та актуальність відеокаталогу. Саме ці чинники визначають вимоги до подальшого проєктування Android-застосунку онлайн-кінотеатру з автоматизованим збором контенту.

1.2 Аналіз потреб користувачів та проблеми наповнення відеокаталогу

Під час створення Android-застосунку онлайн-кінотеатру важливо враховувати не тільки технічну реалізацію, а й те, як користувач буде сприймати та використовувати сервіс у реальних умовах. Для більшості людей такий застосунок не є складною програмною системою. Вони очікують, що після відкриття програми зможуть швидко знайти фільм або серіал, переглянути коротку інформацію про нього і без зайвих дій перейти до перегляду. Тому з погляду користувача головними характеристиками онлайн-кінотеатру є простота, швидкість, зрозуміла навігація та наявність достатньої кількості актуального контенту.

Однією з основних потреб користувача є швидкий доступ до каталогу. Після запуску застосунку користувач не повинен витрачати час на пошук основних розділів або розбиратися зі складною структурою меню. Доцільно, щоб головний екран одразу показував добірки контенту: фільми, серіали, мультфільми або інші категорії, які можуть зацікавити користувача. Такий підхід скорочує шлях від запуску застосунку до вибору конкретного матеріалу. Крім того, головна сторінка фактично стає центральною частиною застосунку, з якої користувач переходить до каталогу, сторінки фільму, профілю, історії або списку обраного.

Для мобільного онлайн-кінотеатру особливо важливим є спосіб подання інформації. На відміну від великого екрана комп’ютера, екран смартфона має обмежений простір, тому інтерфейс не можна перевантажувати зайвими деталями. Картка фільму має містити лише найнеобхідніше: постер, назву, рейтинг, жанр або тривалість. Цих даних достатньо, щоб користувач міг швидко зорієнтуватися в каталозі. Повний опис, країну виробництва, рік випуску, сюжет та інші характеристики краще розміщувати вже на окремій сторінці детального перегляду.

Візуальне оформлення також безпосередньо впливає на зручність користування. У випадку онлайн-кінотеатру користувач часто спочатку звертає увагу саме на постер, а вже потім читає назву або опис. Тому якість і правильне розміщення зображень мають важливе значення. Якщо постери відображаються

некоректно, мають різний розмір або завантажуються занадто повільно, це погіршує загальне враження від застосунку. Навпаки, зрозуміла сітка карток, помітні назви та акуратне розміщення основної інформації роблять каталог зручним навіть за великої кількості контенту.

Не менш важливою є сторінка детального перегляду фільму. Саме на ній користувач отримує більше інформації та вирішує, чи варто відкривати відео. На такій сторінці мають бути зібрані всі основні відомості: назва, постер, рейтинг, жанри, рік випуску, країна, тривалість і короткий опис сюжету. Якщо ці дані подані повно й зрозуміло, користувачу легше зробити вибір. Якщо ж опис відсутній або інформація виглядає неповною, застосунок сприймається менш якісним, навіть якщо технічно він працює правильно.

Ще одна потреба користувача пов’язана з навігацією. У мобільному застосунку основні розділи мають бути доступні в один або два натискання. Для онлайн-кінотеатру такими розділами є головна сторінка, каталог, обране, історія та акаунт. Зручним рішенням є нижня панель навігації, оскільки вона постійно доступна на екрані та відповідає звичній логіці мобільних застосунків. Завдяки цьому користувач швидко розуміє структуру програми й не губиться під час переходів між сторінками.

Окрему групу потреб становлять персоналізовані функції. Якщо застосунок не зберігає дії користувача, він працює лише як загальний каталог. Наявність акаунта дозволяє зробити взаємодію більш індивідуальною: користувач може авторизуватися, переглядати інформацію про свій профіль, повертатися до історії переглядів і зберігати фільми в обране. Такі можливості підвищують практичну цінність сервісу, оскільки користувач не втрачає знайдений контент і може повернутися до нього пізніше.

Для реалізації таких функцій застосунок повинен підтримувати роботу з користувацькою сесією. Після входу в акаунт користувач очікує, що застосунок запам’ятає його стан і не буде вимагати повторної авторизації під час кожного запуску. Тому токен сесії доцільно зберігати локально на пристрої та використовувати його для подальших запитів до серверної частини. Це робить

роботу із застосунком зручнішою, а також дає змогу пов’язувати дії користувача з його обліковим записом.

Пошук є ще однією важливою функцією онлайн-кінотеатру. Якщо каталог містить багато фільмів, серіалів і мультфільмів, користувач не завжди хоче переглядати всі доступні позиції вручну. Він може шукати конкретну назву або підбирати контент за жанром, роком, рейтингом чи іншими ознаками. Тому система пошуку має бути швидкою та зрозумілою. У більш складних випадках доцільно використовувати не тільки пошук за точним збігом, а й семантичний підхід, коли система враховує зміст запиту та може знаходити близькі за значенням результати.

Однак зручність онлайн-кінотеатру залежить не лише від інтерфейсу. Велике значення має якість самого відеокаталогу. Якщо каталог містить мало позицій, застарілі дані, неповні описи або дублікати, користувач швидко втрачає інтерес до сервісу. Для кожного фільму або серіалу потрібно зберігати значну кількість інформації: назву, ідентифікатор, тип контенту, постер, рейтинг, рік, країну, жанри, тривалість, опис, вікове обмеження та посилання для відтворення. Усе це має бути подано в єдиній структурі, щоб серверна частина могла коректно обробляти дані, а мобільний застосунок - відображати їх користувачеві.

Саме на цьому етапі виникає проблема наповнення відеокаталогу. Якщо додавати контент вручну, адміністратор повинен для кожного фільму окремо знаходити інформацію, копіювати її, перевіряти правильність, завантажувати постери та вносити дані до бази. Такий процес займає багато часу і погано масштабується. Для невеликої кількості фільмів ручне введення ще може бути прийнятним, але для повноцінного онлайн-кінотеатру воно стає незручним і неефективним.

Ручне наповнення також збільшує ризик помилок. Наприклад, можна неправильно вказати рік випуску, пропустити один із жанрів, додати неповний опис або створити повторний запис для того самого фільму. Такі помилки не завжди одразу помітні, але вони впливають на якість каталогу. Крім того, відеокаталог потрібно не тільки створити, а й підтримувати в актуальному стані. З

часом додаються нові фільми, змінюються рейтинги, з’являються нові постери або уточнені описи. Якщо всі ці дії виконуються вручну, оновлення системи стає занадто повільним.

Автоматизований збір контенту дозволяє зменшити цю проблему. Його суть полягає в тому, що окремий програмний модуль отримує дані з визначених джерел, знаходить потрібні елементи на сторінках, приводить інформацію до єдиного формату та зберігає її в базі даних. Такий підхід не повністю скасовує потребу в контролі з боку розробника або адміністратора, але значно зменшує кількість однотипної ручної роботи. У результаті каталог можна поповнювати швидше, а структура даних залишається більш послідовною.

Разом із цим автоматизований збір контенту має власні складності. Джерела даних можуть мати різну структуру сторінок, а окремі поля можуть бути відсутні або записані в різному форматі. Наприклад, тривалість фільму може бути подана як число, як текстовий рядок або разом із додатковими словами. Жанри можуть бути записані списком, а рейтинг може бути відсутній для частини контенту. Тому після збору даних їх потрібно очищати, перевіряти й перетворювати до формату, який підходить для бази даних.

Ще однією важливою вимогою є уникнення дублікатів. Якщо модуль автоматизованого збору повторно обробить сторінку того самого фільму, система не повинна створювати ще один ідентичний запис. Для цього кожен об’єкт контенту має мати унікальний ідентифікатор. За ним можна визначити, чи існує такий фільм у базі даних. Якщо запис уже наявний, його потрібно оновити, а не створювати повторно. Такий підхід підтримує порядок у базі даних і спрощує подальшу роботу з каталогом.

У системі KinoFox ця проблема вирішується через окремий модуль автоматизованого збору контенту. Він отримує посилання на сторінки з фільмами, обробляє їх, виділяє основні поля та передає структуровані дані до PostgreSQL. Після цього серверна частина може використовувати збережену інформацію для формування відповідей API, а Android-застосунок - відображати її у вигляді карток,

списків і сторінок детального перегляду. Завдяки цьому мобільний клієнт працює не з випадковими неструктурованими даними, а з уже підготовленою інформацією.

Важливо зазначити, що модуль збору контенту не можна розглядати як другорядний елемент системи. Він прямо впливає на якість роботи онлайн-кінотеатру. Якщо дані зібрані неповно або некоректно, це буде видно в інтерфейсі застосунку. Якщо в базі з'являться дублікати, каталог стане менш зручним. Якщо структура даних не відповідатиме потребам API, серверна частина не зможе передати всю необхідну інформацію клієнту. Отже, наповнення каталогу є не лише адміністративним завданням, а важливою частиною загальної архітектури системи.

Таким чином, потреби користувачів Android-застосунку онлайн-кінотеатру пов'язані з простим доступом до контенту, зручною навігацією, повною інформацією про фільми, швидким пошуком, персоналізацією та стабільною роботою застосунку. Водночас для забезпечення цих потреб необхідно мати якісний і регулярно оновлюваний відеокаталог. Саме тому для системи KinoFox доцільно використовувати автоматизований збір контенту, який скорочує обсяг ручної роботи, допомагає підтримувати структуру даних і створює основу для роботи мобільного застосунку, серверної частини та бази даних.

1.3 Аналіз існуючих програмних рішень у сфері онлайн-кінотеатрів

Для порівняння було обрано Netflix, MEGOGO та SWEET.TV, оскільки ці сервіси представляють різні підходи до організації онлайн-перегляду: міжнародну стримінгову платформу, український медіасервіс із широким набором контенту та сервіс, який поєднує онлайн-кінотеатр із телебаченням.

Netflix можна вважати одним із найбільш відомих прикладів мобільного онлайн-кінотеатру. У його застосунку основна увага приділяється швидкому доступу до фільмів, серіалів та іншого медіаконтенту. Користувач бачить не просто загальний список матеріалів, а добірки, рекомендації, нові релізи, список збереженого контенту та персональні розділи. Офіційний опис застосунку також вказує на наявність персоналізованих рекомендацій, вкладки My Netflix, списку My

List і сповіщень про нові випуски [8]. Для мобільного формату це важливо, оскільки користувач часто відкриває застосунок не з конкретною назвою фільму, а з бажанням швидко обрати щось для перегляду.

Інтерфейс Netflix побудований навколо візуального сприйняття. Значну роль відіграють постери, великі обкладинки, горизонтальні блоки та тематичні добірки. Це дозволяє швидко переглядати багато позицій і не читати довгі описи на першому етапі вибору. Детальніша інформація відкривається вже після переходу на сторінку конкретного фільму або серіалу. Такий підхід є зручним для мобільного екрана, де зайва кількість тексту може заважати сприйняттю.

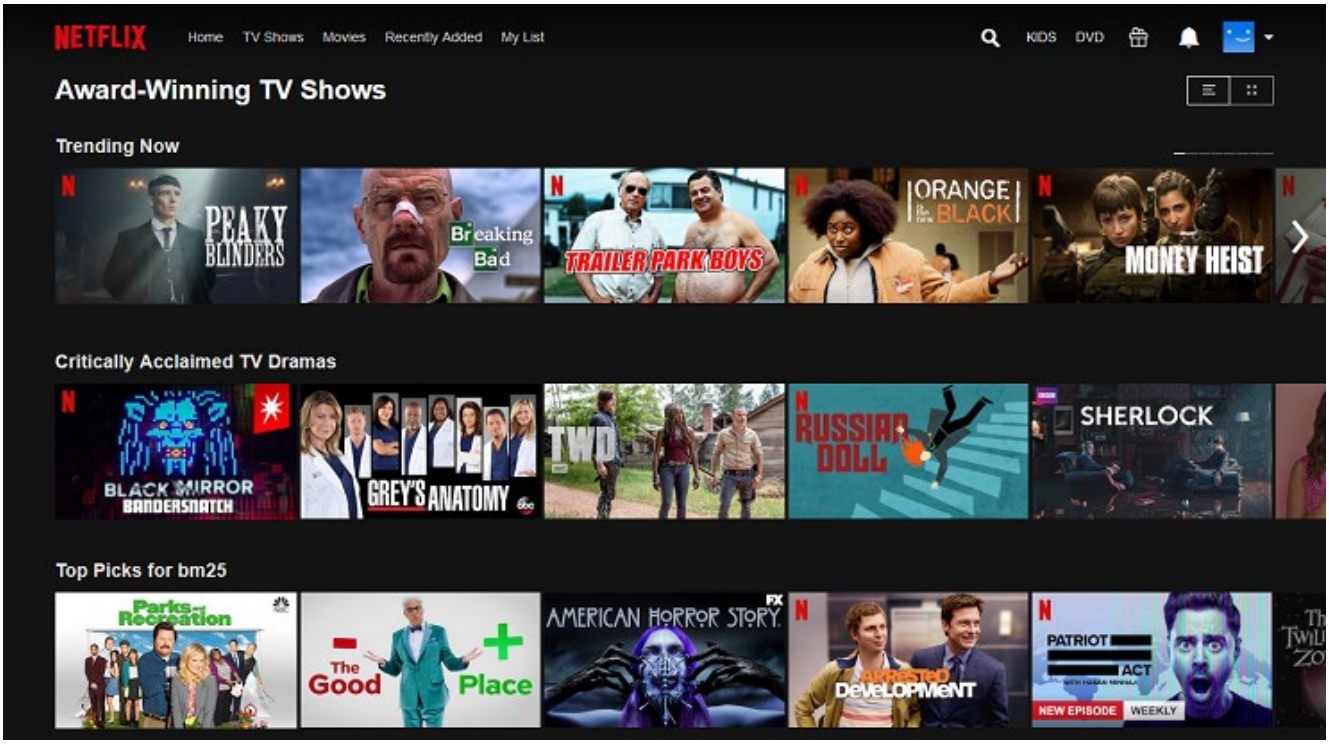


Рис. 1.1 Головний екран Netflix

Сильною стороною Netflix є персоналізація. Сервіс враховує взаємодію користувача з контентом і на цій основі формує рекомендації. Також у застосунку передбачено можливість завантаження окремих матеріалів для перегляду без постійного доступу до інтернету [9]. Це корисно саме для мобільних пристроїв, оскільки користувач не завжди має стабільне з’єднання. Водночас Netflix є закритою комерційною платформою, тому її внутрішні механізми наповнення

каталогу, робота серверної частини, структура бази даних і алгоритми рекомендацій недоступні для аналізу. Через це Netflix доцільно розглядати передусім як приклад якісної організації інтерфейсу та користувацької взаємодії.

MEGOGO є більш близьким прикладом для українського ринку. Цей сервіс поєднує фільми, серіали, телебачення, спорт, освітній контент, подкасти та інші медіаматеріали. В описі застосунку зазначено, що користувач може дивитися телеканали, фільми, мультфільми та новини на смартфоні, планшеті або Smart TV, а також використовувати один акаунт із різними профілями для родини [10]. Такий підхід показує, що сучасний онлайн-кінотеатр часто не обмежується лише фільмами, а поступово перетворюється на універсальний медіасервіс.

Для MEGOGO характерна широка структура каталогу. З одного боку, це є перевагою, бо користувач отримує багато різних форматів контенту в одному застосунку. З іншого боку, така кількість розділів може ускладнювати навігацію. Якщо в одному сервісі одночасно присутні фільми, телеканали, спорт, подкасти, аудіокнижки та освітні матеріали, інтерфейс повинен бути дуже добре продуманим. Інакше користувачеві буде важко швидко знайти потрібний тип контенту.

Для розробки KinoFox досвід MEGOGO є корисним насамперед через його орієнтацію на українську аудиторію. Важливими є україномовна подача інформації, підтримка різних категорій контенту, робота з акаунтом і можливість використання сервісу на кількох пристроях. Водночас у межах дипломного проєкту недоцільно одразу охоплювати таку широку кількість напрямів. Більш раціональним є зосередження на базовому функціоналі онлайн-кінотеатру: головній сторінці, каталозі, сторінці фільму, авторизації, профілі користувача, історії, обраному та пошуку.

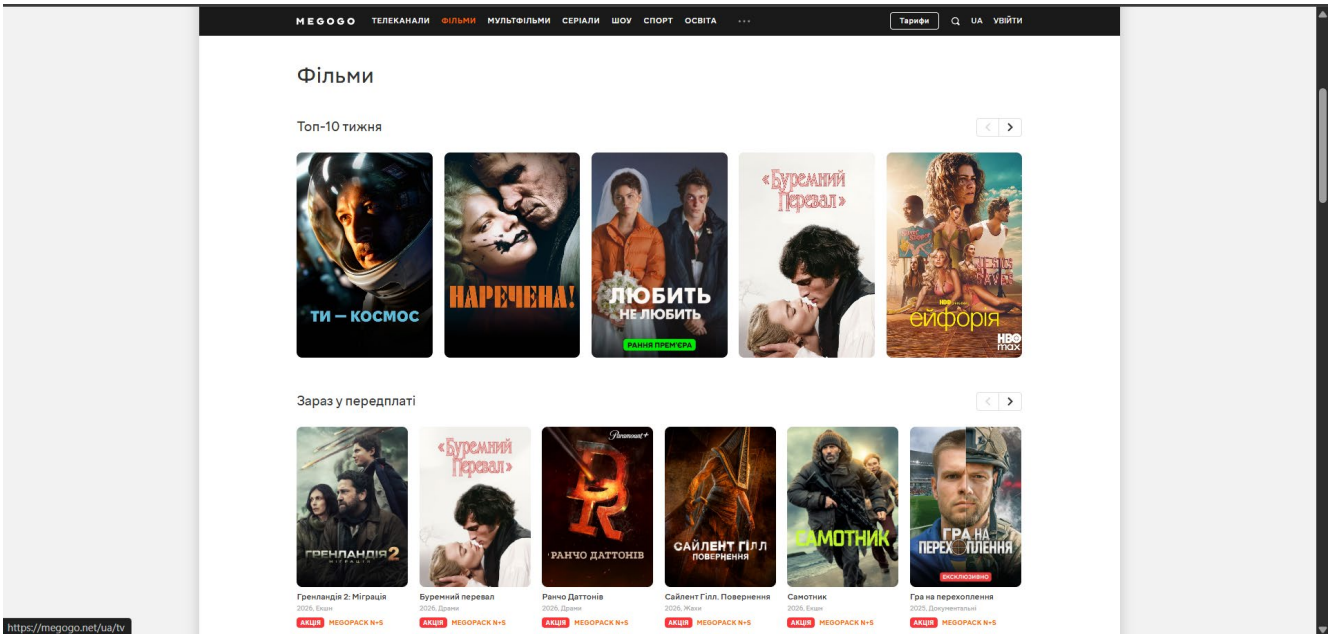


Рис. 1.2 Головний екран MEGOGO

SWEET.TV також є українським сервісом, який поєднує онлайн-кінотеатр і телебачення. На офіційних сторінках сервісу акцентується можливість дивитися телеканали, фільми, мультфільми та серіали на різних пристроях. Окремо виділяється український дубляж, перегляд на Smart TV, смартфоні чи планшеті, а також сімейний формат використання [11]. Це важливо для локального ринку, де користувачі очікують не тільки наявності контенту, а й зручної мови інтерфейсу та озвучення.

SWEET.TV цікавий тим, що показує інший варіант організації медіасервісу. Якщо Netflix більше асоціюється зі стримінгом фільмів і серіалів за запитом, то SWEET.TV значну увагу приділяє телебаченню та перегляду ефірного контенту. Для користувача це зручно, коли він хоче мати в одному застосунку і фільми, і телеканали. Водночас для розроблюваної системи KinoFox важливішим є не повторення телевізійної частини, а принцип поділу матеріалів за типами. У застосунку KinoFox такий підхід може бути використаний через окремі добірки фільмів, серіалів і мультфільмів.

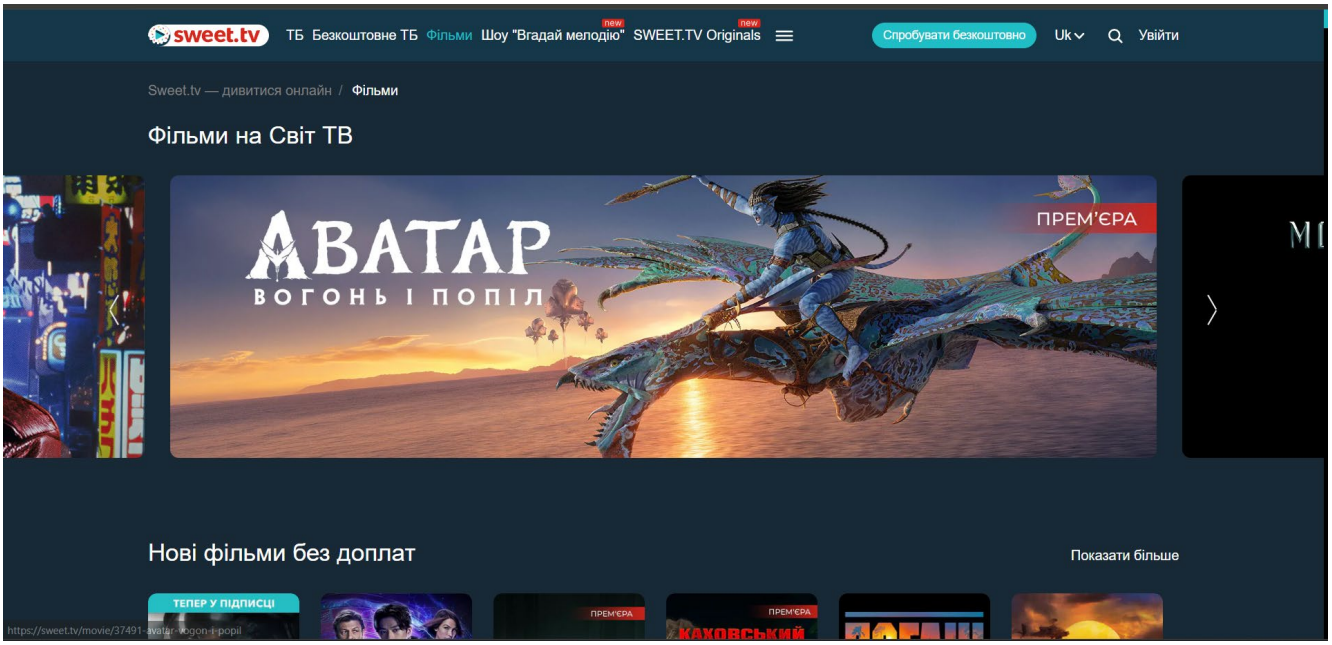


Рис. 1.3 Головний екран SWEET.TV

Після розгляду трьох сервісів можна виділити кілька спільних ознак. По-перше, усі вони використовують візуальний каталог, у якому постер і коротка інформація про контент мають велике значення. По-друге, у таких системах важливу роль відіграє персоналізація: профілі, історія, списки збереженого або рекомендованого контенту. По-третє, сучасні онлайн-кінотеатри намагаються зробити навігацію максимально простою, оскільки користувач часто взаємодіє із сервісом зі смартфона, де немає місця для складної структури меню.

Разом із цим розглянуті рішення мають і певні обмеження з погляду теми цієї роботи. Вони демонструють готовий результат для користувача, але не показують, як саме працює внутрішня частина системи. Для користувача прихованими залишаються процеси наповнення каталогу, обробки даних, формування API-відповідей, збереження фільмів у базі даних і оновлення інформації. У дипломному проєкті, навпаки, важливо показати не лише інтерфейс, а й повний цикл роботи: від отримання даних до їх відображення в Android-застосунку.

Ще одним важливим спостереженням є те, що великі комерційні сервіси мають дуже широкий функціонал. Для них це виправдано, оскільки вони працюють із великою аудиторією та мають розвинену інфраструктуру. Однак у межах

навчального проєкту краще обрати менший, але завершений набір можливостей. Саме тому для KinoFox доцільно реалізувати ті функції, які формують основу онлайн-кінотеатру: каталог, сторінку детального перегляду, авторизацію, профіль користувача, історію, обране, пошук і автоматизоване наповнення бази контентом.

Окремою відмінністю KinoFox від розглянутих аналогів є наявність модуля автоматизованого збору контенту як частини програмної системи. У Netflix, MEGOGO та SWEET.TV користувач бачить уже готовий каталог і не взаємодіє з процесом його формування. У KinoFox цей процес винесено в окремий модуль, який отримує дані, структурує їх і передає до бази даних. Завдяки цьому система може бути описана як повний програмний комплекс, а не лише як мобільний інтерфейс.

Отже, аналіз Netflix, MEGOGO та SWEET.TV показує, що для Android-застосунку онлайн-кінотеатру важливими є зручний каталог, швидка навігація, якісна сторінка фільму, персоналізовані функції, пошук і стабільна робота з користувацькими даними. При цьому система KinoFox має власну особливість - автоматизований збір контенту, який дозволяє розглядати її як клієнт-серверний застосунок із окремим модулем підготовки відеокаталогу. Саме це визначає подальші вимоги до проєктування клієнтської частини, API, бази даних і механізму наповнення контентом.

1.4 Визначення вимог до Android-застосунку онлайн-кінотеатру

Для системи KinoFox такі вимоги мають враховувати дві сторони роботи програмного продукту: з одного боку, зручність користувача під час перегляду каталогу й вибору фільму, а з іншого — технічну організацію отримання, збереження та передавання даних між клієнтською частиною, сервером, базою даних і модулем автоматизованого збору контенту.

Застосунок що розробляється не повинен обмежуватися лише відображенням списку фільмів. Він має працювати як частина клієнт-серверної системи, у якій Android-клієнт відповідає за інтерфейс і взаємодію з користувачем, серверна

частина — за обробку запитів і бізнес-логіку, база даних — за збереження інформації, а парсер — за автоматизоване наповнення каталогу. Саме тому вимоги до системи доцільно поділити на функціональні та нефункціональні.

Функціональні вимоги описують, які можливості повинна надавати система. Для онлайн-кінотеатру до таких можливостей належать перегляд каталогу, відкриття сторінки фільму, авторизація, робота з користувацьким профілем, пошук, обране, історія переглядів і автоматизоване оновлення контенту. Ці функції формують основний сценарій використання застосунку: користувач відкриває головну сторінку, переглядає доступний контент, переходить до фільму, ознайомлюється з описом і за потреби зберігає його або повертається до нього пізніше.

У таблиці 1.1 наведено основні функціональні вимоги до Android-застосунку онлайн-кінотеатру KinoFox.

Таблиця 1.1

Функціональні вимоги до Android-застосунку онлайн-кінотеатру KinoFox

Вимога	**Опис**
Перегляд головної сторінки	Застосунок має відображати головну сторінку з добірками фільмів, серіалів і мультфільмів
Перегляд каталогу контенту	Користувач має мати можливість переглядати доступний відеоконтент у вигляді карток
Відображення картки фільму	Картка має містити постер, назву, рейтинг, жанри або інші короткі характеристики
Перегляд детальної інформації	Система має відкривати сторінку з повним описом фільму, роком, країною, жанрами, тривалістю та рейтингом
Відтворення відеоконтенту	Застосунок має надавати можливість перейти до перегляду доступного відео через вбудований механізм відтворення
Реєстрація користувача	Система має дозволяти створення нового облікового запису
Авторизація користувача	Користувач має мати можливість увійти до системи за email і паролем

Вимога	Опис
Збереження сесії	Після входу застосунок має зберігати токен сесії для подальшої роботи без повторного введення даних

Продовження таблиці 1.1

Функціональні вимоги до Android-застосунку онлайн-кінотеатру KinoFox

Вимога	Опис
Перегляд профілю	Авторизований користувач має мати доступ до інформації про свій акаунт і стан сесії
Робота з обраним	Система має передбачати можливість додавання фільмів до списку обраного та видалення їх із нього
Історія переглядів	Застосунок має зберігати інформацію про контент, з яким взаємодіяв користувач
Пошук контенту	Користувач має мати можливість шукати фільми за назвою або змістовим запитом
Фільтрація та сортування	Серверна частина має підтримувати відбір контенту за типом, рейтингом, роком, країною та жанрами
Автоматизований збір контенту	Система має містити окремий модуль для отримання, обробки та збереження інформації про фільми
Оновлення наявних записів	Під час повторного збору даних система має оновлювати наявний запис, а не створювати дублікати

Наведені функціональні вимоги відображають основну логіку роботи системи. Частина з них стосується безпосередньо мобільного застосунку, а частина — серверної частини й модуля збору даних. Наприклад, перегляд карток і сторінки фільму реалізується на клієнтському рівні, але самі дані надходять із серверу. Авторизація також не може бути повністю реалізована тільки в мобільному застосунку, оскільки перевірка користувача, створення сесії та збереження токена відбуваються на серверному рівні. Тому функціональні вимоги потрібно розглядати не ізольовано, а як вимоги до всієї системи KinoFox.

Окремої уваги потребує вимога до автоматизованого збору контенту. Для звичайного мобільного застосунку достатньо було б реалізувати відображення вже

готового каталогу. Проте тема кваліфікаційної роботи передбачає саме автоматизоване наповнення, тому парсер є важливою частиною системи. Він має отримувати посилання на сторінки з контентом, виділяти з них назву, постер, рейтинг, рік, країну, жанри, тривалість, опис та інші дані, після чого передавати їх до бази даних. Завдяки цьому каталог може формуватися швидше, ніж у разі ручного введення.

Крім функціональних вимог, необхідно визначити нефункціональні вимоги. Вони не описують окремі функції, але визначають якість роботи системи: швидкодію, зручність, безпеку, стабільність, сумісність і можливість подальшого розвитку. Для Android-застосунку онлайн-кінотеатру ці вимоги є не менш важливими, оскільки навіть правильно реалізований функціонал може бути незручним, якщо застосунок повільно завантажується, погано відображається на різних екранах або нестабільно працює з мережею.

У таблиці 1.2 наведено основні нефункціональні вимоги до системи KinoFox.

Таблиця 1.2

Нефункціональні вимоги до системи KinoFox

Вимога	**Опис**
Швидкодія інтерфейсу	Швидкодія інтерфейсу Основні екрани застосунку мають відкриватися не довше ніж за 1 секунду після натискання на відповідний пункт навігації.
Час отримання даних	Час отримання даних Запити до API для стандартних операцій мають виконуватися не довше ніж за 2 секунди за стабільного інтернет-з’єднання зі швидкістю від 10 Мбіт/с.
Адаптивність інтерфейсу	Адаптивність інтерфейсу Інтерфейс має коректно відображатися на смартфонах з діагоналлю екрана від 5 до 7 дюймів та роздільною здатністю від 720×1280 px.
Зручність навігації	Зручність навігації Перехід до основних розділів застосунку має виконуватися не більше ніж за 2 натискання з будь-якого основного екрана.

Вимога	Опис
Безпека авторизації	Безпека авторизації Паролі користувачів мають зберігатися лише у хешованому вигляді; строк дії користувацької сесії має становити не більше 30 днів.

Для системи KinoFox важливою є вимога щодо розділення відповідальності між компонентами. Мобільний застосунок не повинен містити логіку прямого збирання контенту або безпосередньо працювати з базою даних. Його завдання — надсилати запити, отримувати відповіді та відображати інформацію користувачеві. Серверна частина повинна бути проміжним рівнем між клієнтом і базою даних. Вона приймає запити, перевіряє користувацькі сесії, отримує дані з бази, формує відповіді у відповідному форматі та передає їх клієнту. Парсер, у свою чергу, виконує окреме завдання — збір і підготовку інформації для каталогу.

Така побудова системи дає змогу уникнути надмірної залежності між компонентами. Якщо в майбутньому потрібно змінити інтерфейс мобільного застосунку, це не повинно вимагати зміни структури парсера. Якщо потрібно додати нові поля до фільму, достатньо оновити модель даних, API та відповідні елементи інтерфейсу. Якщо потрібно розширити автоматизований збір контенту, це можна зробити в межах окремого модуля, не переписуючи весь застосунок. Саме тому клієнт-серверний підхід є доцільним для розроблюваної системи.

До вимог також належить коректна робота з користувацькими даними. Оскільки система передбачає авторизацію, вона повинна забезпечувати захист паролів і сесій. Паролі не мають зберігатися у відкритому вигляді, а токен сесії повинен використовуватися для перевірки користувача під час звернення до захищених функцій. Для мобільного застосунку також важливо зберігати сесію так, щоб користувач міг повторно відкривати застосунок без постійного введення логіна й пароля, але водночас не втрачалась можливість перевірити актуальність цієї сесії на сервері.

Вимоги до бази даних пов’язані з тим, що вона повинна зберігати не тільки інформацію про фільми, а й дані користувачів, сесії, обране та історію. Для фільмів

необхідно передбачити поля для назви, унікального ідентифікатора, типу контенту, постера, рейтингу, року випуску, країни, жанрів, тривалості, опису та списку посилань для відтворення. Для користувачів потрібно зберігати email, ім’я користувача, пароль у захищеному вигляді та пов’язані з ним записи. Також необхідно передбачити зв’язки між користувачем і фільмами, які він додав до обраного або переглядав раніше.

Для пошуку контенту важливо, щоб серверна частина могла не тільки повертати весь список фільмів, а й обробляти параметри запиту. До таких параметрів можуть належати тип контенту, мінімальний і максимальний рейтинг, рік випуску, країна, жанр, порядок сортування та текстовий пошуковий запит. Оскільки в системі передбачено використання векторних представлень, пошук може бути розширений до семантичного, тобто такого, що враховує зміст запиту, а не лише буквальний збіг.

Вимоги до автоматизованого збору контенту передбачають, що модуль парсингу має працювати послідовно та контрольовано. Він не повинен просто копіювати сторінку як текст, а має виділяти конкретні поля, які потім можна використовувати в базі даних і застосунку. Також модуль має перевіряти, чи існує фільм у базі даних, і залежно від цього створювати новий запис або оновлювати наявний. Це дозволяє уникнути повторів і підтримувати каталог у більш впорядкованому стані.

Загалом сформовані вимоги дають змогу перейти від загального опису предметної галузі до проєктування конкретної системи. Вони визначають, які функції має виконувати KinoFox, якою має бути структура взаємодії між компонентами, які дані потрібно зберігати та які якісні характеристики повинна мати система. Надалі ці вимоги будуть використані під час вибору засобів реалізації, проєктування архітектури, створення клієнтської та серверної частин, а також під час тестування готового застосунку.

2 ЗАСОБИ РЕАЛІЗАЦІЇ ANDROID-ЗАСТОСУНКУ ОНЛАЙН-КІНОТЕАТРУ

2.1 Обґрунтування вибору клієнт-серверної архітектури

Під час розробки Android-застосунку онлайн-кінотеатру важливо правильно визначити загальну архітектуру системи, оскільки від цього залежить подальша організація програмного коду, спосіб збереження даних, робота з користувачами, пошук контенту та можливість масштабування. Для системи KinoFox було обрано клієнт-серверну архітектуру, оскільки вона найкраще відповідає особливостям онлайн-кінотеатру, де мобільний застосунок має отримувати дані з віддаленого серверу, а каталог контенту повинен зберігатися централізовано.

Клієнт-серверна архітектура передбачає поділ програмної системи на окремі частини, кожна з яких виконує свою роль. Клієнтська частина відповідає за інтерфейс користувача та надсилання запитів, серверна частина — за обробку цих запитів, перевірку даних, виконання бізнес-логіки та формування відповіді. База даних використовується для збереження основної інформації про користувачів, фільми, сесії, історію переглядів і список обраного. У випадку KinoFox до цієї структури також додається окремий модуль автоматизованого збору контенту, який наповнює базу даних інформацією про відеоматеріали.

Для Android-застосунку онлайн-кінотеатру такий підхід є більш доцільним, ніж зберігання всіх даних безпосередньо в мобільному застосунку. Відеокаталог може містити велику кількість записів, постерів, описів, жанрів, рейтингів і посилань. Якщо зберігати всі ці дані локально, застосунок буде важче оновлювати, а користувач не завжди отримуватиме актуальну інформацію. У клієнт-серверній моделі мобільний застосунок отримує лише ті дані, які потрібні в конкретний момент: список фільмів для головної сторінки, інформацію про вибраний фільм або дані профілю користувача. Це робить систему гнучкішою та зручнішою для підтримки.

Однією з головних переваг клієнт-серверної архітектури є централізоване збереження даних. Усі основні записи зберігаються в базі даних PostgreSQL, а мобільний застосунок отримує їх через серверну частину. Завдяки цьому інформацію про фільми можна оновлювати на сервері без необхідності змінювати сам Android-застосунок. Наприклад, якщо до каталогу додається новий фільм або оновлюється опис уже наявного контенту, користувач бачить ці зміни після чергового запиту до API. Це особливо важливо для онлайн-кінотеатру, де актуальність каталогу є однією з ключових умов нормальної роботи сервісу.

Клієнт-серверний підхід також полегшує реалізацію авторизації та роботи з користувацькими сесіями. У системі KinoFox користувач може зареєструватися, увійти в акаунт, отримати токен сесії та надалі використовувати його для доступу до персональних функцій. Перевірка користувача, створення сесії, збереження її строку дії та оновлення даних відбуваються на серверному рівні. Мобільний застосунок у цьому випадку не виконує перевірку пароля самостійно, а лише передає введені дані серверу й отримує результат. Це є більш безпечним і логічним рішенням, оскільки критична логіка не повинна зберігатися на клієнтському боці.

Ще однією причиною вибору клієнт-серверної архітектури є потреба в реалізації персоналізованих функцій. Історія переглядів і список обраного мають бути пов’язані з конкретним користувачем. Якщо ці дані зберігати тільки локально на пристрої, вони можуть бути втрачені після перевстановлення застосунку або недоступні на іншому пристрої. Коли ж вони зберігаються в базі даних на сервері, система може пов’язувати дії користувача з його акаунтом. Це створює основу для подальшого розширення функціоналу, наприклад для рекомендацій, продовження перегляду або синхронізації між різними пристроями.

Для онлайн-кінотеатру важливою є також можливість фільтрації та пошуку контенту. Якщо обробляти всі пошукові операції на мобільному пристрої, застосунок повинен був би отримувати великий обсяг даних і виконувати зайву роботу на клієнтському боці. Натомість у клієнт-серверній архітектурі пошук і фільтрація можуть виконуватися на сервері або рівні бази даних. Клієнт надсилає параметри запиту, а сервер повертає вже відібраний список результатів. Це

дозволяє зменшити навантаження на мобільний застосунок і зробити взаємодію з каталогом швидшою та стабільнішою.

У системі KinoFox серверна частина виконує роль проміжного рівня між Android-застосунком і базою даних. Вона приймає HTTP-запити, обробляє їх, звертається до PostgreSQL, формує відповідь і повертає її клієнту у структурованому форматі. Така організація відповідає принципам REST API, коли кожна операція має окремий маршрут: отримання списку фільмів, отримання інформації про конкретний фільм, реєстрація, авторизація, робота з обраним або історією. Для мобільного застосунку це зручно, оскільки він не взаємодіє з базою даних напряму, а працює лише з визначеним набором API-запитів.

Окремою перевагою клієнт-серверної архітектури є можливість незалежної розробки різних частин системи. Клієнтська частина може створюватися й удосконалюватися окремо від серверної, якщо між ними визначено зрозумілий формат обміну даними. Наприклад, у KinoFox мобільний застосунок на Flutter відповідає за відображення головної сторінки, карток контенту, сторінки фільму, форм авторизації та профілю. Серверна частина на Bun та Elysia відповідає за API, роботу з користувачами й отримання даних із бази. Python-парсер виконує окреме завдання — збирає та готує інформацію для каталогу. Завдяки такому поділу кожен компонент має зрозумілу зону відповідальності.

Клієнт-серверна модель також є зручною з погляду масштабування. Якщо в майбутньому потрібно буде додати вебверсію, адміністративну панель або інший мобільний клієнт, вони зможуть використовувати ту саму серверну частину й базу даних. Тобто API стає спільною точкою доступу до системи. Це важливо для програмного продукту, який може розвиватися після завершення базової реалізації. У такому випадку не потрібно створювати всю логіку заново — достатньо підключити новий клієнт до вже наявних серверних маршрутів.

Водночас клієнт-серверна архітектура має і певні вимоги до якості реалізації. Оскільки мобільний застосунок залежить від серверної частини, необхідно передбачити обробку помилок мережі, відсутність відповіді серверу, некоректні дані або завершення строку дії сесії. Користувач не повинен бачити порожній екран

без пояснення. Застосунок має показувати стан завантаження, повідомлення про помилку або можливість повторити запит. Це особливо важливо для Android-застосунку, який може працювати в умовах нестабільного мобільного інтернету.

Для безпеки така архітектура також є більш придатною, ніж збереження всієї логіки на клієнті. Сервер може контролювати доступ до захищених функцій, перевіряти токени сесій, не передавати клієнту зайві дані та приховувати внутрішню структуру бази. Мобільний застосунок отримує лише ту інформацію, яка потрібна для відображення інтерфейсу. Це зменшує ризик некоректної роботи з даними й дозволяє краще контролювати доступ користувачів до функцій системи.

Ще одним важливим чинником є автоматизований збір контенту. Для KinoFox парсер не повинен бути частиною Android-застосунку, оскільки мобільний клієнт не призначений для обробки зовнішніх сторінок і запису даних до бази. Доцільніше винести цей процес в окремий серверний або допоміжний модуль, який працює незалежно від користувацького інтерфейсу. Після збору та обробки дані потрапляють до бази, а вже звідти через API використовуються мобільним застосунком. Така схема є більш логічною, бо вона відокремлює процес підготовки контенту від процесу його перегляду.

Таким чином, вибір клієнт-серверної архітектури для Android-застосунку онлайн-кінотеатру KinoFox є обґрунтованим з погляду збереження даних, безпеки, масштабованості, підтримки персоналізованих функцій і автоматизованого наповнення каталогу. Такий підхід дозволяє розділити систему на зрозумілі компоненти: мобільний клієнт, серверну частину, базу даних і модуль збору контенту. Завдяки цьому KinoFox можна розглядати не лише як мобільний застосунок для перегляду фільмів, а як повноцінну програмну систему, у якій кожен компонент виконує окреме завдання та взаємодіє з іншими через визначені механізми.

2.2 Засоби реалізації клієнтської частини застосунку

Клієнтська частина системи KinoFox є тією частиною програмного продукту, з якою безпосередньо взаємодіє користувач. Саме через мобільний інтерфейс користувач переглядає каталог, відкриває сторінки фільмів, авторизується, переходить до профілю, історії переглядів та обраного. Тому під час вибору засобів реалізації клієнтської частини потрібно було врахувати не лише можливість створення Android-застосунку, а й зручність побудови інтерфейсу, підтримку мережевих запитів, роботу з локальним збереженням даних і можливість інтеграції з WebView для відтворення відеоконтенту.

Для реалізації клієнтської частини було обрано Flutter і мову програмування Dart. Flutter є фреймворком для створення застосунків із єдиною кодовою базою, який дозволяє розробляти інтерфейси для мобільних платформ, зокрема Android. У межах системи KinoFox Flutter використовується для побудови екранів застосунку, нижньої навігації, карток фільмів, сторінки детального перегляду, форм авторизації та профілю користувача. Такий вибір є доцільним, оскільки Flutter має гнучку систему віджетів і дозволяє швидко створювати адаптивний інтерфейс, що коректно відображається на різних екранах мобільних пристроїв [12].

Мова Dart використовується як основна мова програмування клієнтської частини. Вона добре поєднується з Flutter, має зрозумілий синтаксис і підтримує об’єктно-орієнтований підхід. Для застосунку KinoFox це важливо, оскільки клієнтська частина містить окремі моделі даних, сервіси для роботи з API, сторінки інтерфейсу та допоміжні класи. Dart дозволяє структурувати код таким чином, щоб логіка відображення, обробка даних і мережеві запити не змішувалися в одному місці.

У проєкті KinoFox головною точкою запуску клієнтського застосунку є файл main.dart. Саме з нього починається виконання Flutter-застосунку. Далі основна структура інтерфейсу формується у файлі app.dart, де реалізовано загальну логіку навігації між основними розділами. У застосунку використовується нижня панель навігації, яка містить п’ять основних сторінок: головну, каталог, улюблене, історію

та акаунт. Такий підхід відповідає звичній логіці мобільних застосунків, оскільки користувач може швидко перейти до потрібного розділу без складного меню.

Клієнтська частина KinoFox має темне оформлення з помаранчевим акцентним кольором. Така кольорова схема є доречною для онлайн-кінотеатру, оскільки темний фон не відволікає від постерів і відеоконтенту, а яскравий акцент допомагає виділяти активні елементи інтерфейсу. Крім того, у застосунку використовуються шрифти e-Ukraine та e-Ukraine Head, що робить інтерфейс більш цілісним і наближеним до україномовного користувача. Для дипломного проєкту це важливо, оскільки система орієнтована на зрозумілу подачу інформації українською мовою.

Основним екраном застосунку є головна сторінка. Вона містить блоки з фільмами, серіалами та мультфільмами. Кожен блок отримує дані з серверної частини через відповідний сервіс і передає їх до віджета, який формує сітку карток. У такій структурі головна сторінка не зберігає всі дані вручну, а працює з результатами API-запитів. Це відповідає клієнт-серверній архітектурі, описаній у попередньому підрозділі, де мобільний застосунок відповідає за відображення, а сервер — за отримання й підготовку даних.

Для відображення контенту використовується модель Film. Вона описує основні поля, які надходять із серверної частини: ідентифікатор, тип контенту, назву, постер, рейтинг, рік, країну, жанри, тривалість, опис і список посилань для перегляду. Наявність окремої моделі є важливою, оскільки вона дозволяє привести отримані JSON-дані до зрозумілої структури в коді застосунку. Завдяки цьому інтерфейс працює не з неструктурованою відповіддю серверу, а з об’єктами, які мають конкретні поля й можуть використовуватися на різних сторінках.

Відображення списку фільмів реалізовано через окремий компонент, який формує картки контенту. Картка фільму містить постер, назву, рейтинг, тривалість і жанри. Такий формат подання є зручним для мобільного екрана, оскільки користувач бачить основну інформацію ще до відкриття повної сторінки. При натисканні на картку здійснюється перехід до сторінки детального перегляду, куди

передається ідентифікатор фільму. Далі застосунок отримує повну інформацію про цей фільм із серверу.

Сторінка детального перегляду є однією з ключових частин клієнтського застосунку. Вона показує розширену інформацію про вибраний контент: постер, назву, рейтинг, рік випуску, тривалість, жанри, країну, сюжетний опис і доступні посилання для перегляду. У системі KinoFox ця сторінка також відповідає за відображення відеоплеєра через вбудований WebView-компонент. Це дає змогу відкривати джерело відтворення без переходу до зовнішнього браузера, що робить роботу із застосунком більш цілісною.

Для роботи з відеовідтворенням у клієнтській частині використовується flutter_inappwebview. Цей пакет дозволяє вбудовувати вебвміст безпосередньо в Flutter-застосунок. Для онлайн-кінотеатру це важливо, оскільки частина відеоконтенту може відкриватися через iframe або вебплеєр. Використання WebView дає змогу інтегрувати такий механізм у мобільний застосунок без необхідності створювати окремий повноцінний відеоплеєр з нуля. Водночас цей підхід потребує обережної обробки посилань і перевірки коректності завантаження сторінок відтворення.

Для взаємодії із серверною частиною в застосунку використовуються API-сервіси. Вони відповідають за надсилання HTTP-запитів, отримання відповідей і перетворення даних у відповідні моделі. Наприклад, окремі сервіси використовуються для отримання списку фільмів, серіалів і мультфільмів, а також для отримання детальної інформації за ідентифікатором фільму. Такий поділ є зручним, оскільки сторінки інтерфейсу не повинні містити всю логіку мережевої взаємодії. Їх завдання — викликати потрібний сервіс і відобразити отриманий результат.

У таблиці 2.1 наведено основні засоби, які використовуються для реалізації клієнтської частини системи KinoFox.

Таблиця 2.1

Засоби реалізації клієнтської частини Android-застосунку KinoFox

Засіб	Призначення у системі KinoFox
Flutter	Створення інтерфейсу Android-застосунку та побудова екранів
Dart	Реалізація логіки клієнтської частини, моделей даних і сервісів
Material-компоненти	Побудова кнопок, панелей навігації, карток і загальної структури інтерфейсу
http	Надсилання запитів до серверної частини та отримання відповідей API
flutter_dotenv	Робота з конфігураційними змінними, зокрема базовою адресою API
shared_preferences	Локальне збереження токена сесії користувача
flutter_inappwebview	Вбудоване відображення вебплеєра або iframe для перегляду відео
iconify_flutter	Використання іконок в інтерфейсі застосунку
e-Ukraine fonts	Формування єдиного стилю україномовного інтерфейсу

Окрему роль у клієнтській частині відіграє робота з авторизацією. У застосунку передбачено сторінки входу та реєстрації. Під час авторизації користувач вводить email і пароль, після чого застосунок надсилає ці дані на сервер. Якщо авторизація успішна, сервер повертає токен сесії. Цей токен зберігається локально за допомогою shared_preferences і надалі може використовуватися для запитів, які потребують підтвердження користувача. Завдяки цьому користувач не повинен повторно вводити дані під час кожного відкриття застосунку.

Сторінка профілю користувача використовує збережений токен для отримання інформації про акаунт і стан сесії. Якщо токен відсутній або недійсний, користувач має пройти авторизацію повторно. Такий підхід є типовим для клієнт-

серверних мобільних застосунків, оскільки дозволяє зберігати зручність роботи й водночас залишати перевірку сесії на серверному боці. У KinoFox це важливо для подальшої роботи з історією переглядів і списком обраного.

Сторінки історії та обраного у клієнтській частині передбачені як окремі розділи навігації. Вони логічно пов’язані з персоналізацією користувача. Навіть якщо частина їх функціональності може потребувати подальшого розширення, сама наявність цих сторінок у структурі застосунку показує, що архітектура клієнтської частини вже враховує розвиток системи. Надалі ці сторінки можуть отримувати дані з відповідних API-маршрутів і відображати персональні списки користувача.

Для роботи з конфігурацією в застосунку використовується файл середовища, у якому задається базова адреса серверної частини. Це зручно, оскільки дозволяє змінювати адресу API без переписування основного коду. Наприклад, під час розробки застосунок може працювати з локальним сервером, а після розгортання — з віддаленим. Такий підхід спрощує тестування й підтримку клієнтської частини.

Важливою особливістю реалізації є розділення коду за папками та файлами. Окремо розміщуються сторінки, моделі, сервіси, віджети та допоміжні класи. Така структура робить проєкт зрозумілішим і зручнішим для подальшої підтримки. Наприклад, зміни в моделі фільму не потребують повної переробки інтерфейсу, а оновлення API-сервісу не повинно впливати на всю структуру застосунку. Для дипломного проєкту це є важливим показником того, що клієнтська частина створена не як набір випадкових екранів, а як структурований програмний модуль.

Під час реалізації Android-застосунку також потрібно враховувати можливі помилки під час отримання даних. Якщо сервер не відповідає, повертає порожній список або виникає проблема з мережею, інтерфейс не повинен працювати некоректно. Для цього в клієнтській частині доцільно використовувати стани завантаження, перевірку отриманих даних і повідомлення про помилки. Це особливо важливо для онлайн-кінотеатру, де більшість інформації надходить із серверу, а якість інтернет-з’єднання користувача може бути нестабільною.

Отже, для реалізації клієнтської частини KinoFox було обрано Flutter і Dart, оскільки вони дозволяють створити зручний Android-застосунок із сучасним інтерфейсом, підтримкою API-запитів, локальним збереженням сесії та вбудованим відображенням відеоконтенту. Клієнтська частина виконує роль мобільного інтерфейсу системи, але не працює ізольовано: вона постійно взаємодіє із серверною частиною, отримує дані з бази через API та відображає їх користувачеві. Такий підхід відповідає обраній клієнт-серверній архітектурі та створює основу для подальшого розвитку функцій онлайн-кінотеатру.

2.3 Засоби реалізації серверної частини та API

Серверна частина є важливою складовою системи KinoFox, оскільки саме вона забезпечує обробку запитів від Android-застосунку, взаємодію з базою даних, авторизацію користувачів, отримання інформації про фільми, роботу з історією переглядів та списком обраного. Якщо клієнтська частина відповідає за зручне відображення інформації, то серверна частина відповідає за те, щоб ця інформація була правильно отримана, перевірена, оброблена та передана у відповідному форматі.

Для реалізації серверної частини системи KinoFox було використано Bun, Elysia, TypeScript, Drizzle ORM та набір допоміжних бібліотек для роботи з базою даних, авторизацією, хешуванням паролів і формуванням API. Такий стек було обрано через його швидкодію, відносно просту структуру коду, підтримку сучасного JavaScript/TypeScript-підходу та зручність побудови REST API.

Bun у проєкті використовується як середовище виконання серверної частини. Його можна розглядати як альтернативу традиційному Node.js-середовищу, але з акцентом на швидкий запуск, вбудовану роботу з пакетами та зручну розробку серверних застосунків. Для дипломного проєкту важливо, що Bun дозволяє запускати TypeScript-код без складного додаткового налаштування, що спрощує розробку та тестування API [13]. У KinoFox через Bun запускається backend-

застосунок, виконуються скрипти розробки, міграції бази даних і старт серверної частини.

Основним фреймворком для побудови API було обрано Elysia [14]. Це вебфреймворк, орієнтований на використання разом із Bun. У системі KinoFox Elysia застосовується для створення серверного застосунку, опису маршрутів, групування API-ендпоінтів і підключення окремих модулів. Головний серверний файл формує застосунок із префіксом /api, після чого до нього підключаються маршрути, пов’язані з фільмами, користувачами, авторизацією, улюбленим і історією переглядів.

Використання Elysia є зручним для такого проєкту, оскільки API онлайн-кінотеатру складається з кількох логічних груп. Наприклад, окремо можна виділити маршрути для роботи з фільмами, маршрути авторизації, маршрути обраного та маршрути історії. Такий поділ робить серверну частину зрозумілішою: кожна група маршрутів відповідає за свою частину функціональності, а головний файл сервера не перевантажується всією логікою одразу.

У серверній частині KinoFox використовується TypeScript [15]. Його застосування є доцільним, оскільки сервер працює з різними структурами даних: користувачами, сесіями, фільмами, списками жанрів, країнами, рейтингами, історією переглядів і результатами пошуку. TypeScript дозволяє описувати типи даних і зменшує кількість помилок, які можуть виникнути через неправильну структуру об’єкта або некоректну роботу з полями. Для серверного API це важливо, оскільки помилки в обробці даних можуть вплинути на роботу клієнтської частини.

У KinoFox серверна частина не містить усю логіку в одному файлі. Код поділено на маршрути, сервіси, схеми бази даних, утиліти авторизації та допоміжні модулі. Такий підхід полегшує підтримку проєкту. Наприклад, логіка отримання списку фільмів відокремлена від логіки авторизації користувача, а робота з базою даних винесена в окремі модулі. Завдяки цьому зміна одного функціонального блоку не повинна порушувати роботу всієї серверної частини.

Для роботи з базою даних використовується Drizzle ORM [16]. ORM дає змогу описувати таблиці бази даних у вигляді програмних схем і виконувати запити

до бази більш структуровано. У проєкті KinoFox через Drizzle описано таблиці фільмів, користувачів, користувацьких сесій, обраного та історії. Це дозволяє зберігати зв'язок між структурою бази даних і кодом серверної частини. Якщо таблиця має конкретні поля, ці поля можна використовувати в запитах без постійного написання великих SQL-конструкцій вручну.

Важливою перевагою використання Drizzle ORM є підтримка міграцій. Міграції дозволяють створювати та оновлювати структуру бази даних контрольовано. Для системи KinoFox це важливо, оскільки база даних містить кілька пов'язаних таблиць і спеціальне розширення для роботи з векторними представленнями. Завдяки міграціям можна відтворити структуру бази даних на іншому середовищі, наприклад під час локальної розробки, тестування або розгортання.

Серверна частина KinoFox працює з PostgreSQL через підключення до бази даних. У коді перевіряється наявність змінної середовища з адресою підключення до бази, після чого створюється пул підключень. Такий підхід є типовим для серверних застосунків, оскільки база даних не повинна відкривати нове з'єднання для кожного окремого запиту без контролю. Пул підключень дозволяє ефективніше працювати з кількома запитами та зменшує ризик перевантаження бази.

Одним із ключових завдань серверної частини є надання REST API для клієнтського застосунку. REST API дозволяє Android-застосунку звертатися до серверу через HTTP-запити й отримувати відповіді у структурованому форматі. У KinoFox через API реалізовано отримання списку фільмів, отримання детальної інформації про фільм, реєстрацію, авторизацію, перевірку сесії, роботу з обраним та історією переглядів. Для мобільного застосунку це означає, що він не працює з базою даних напряму, а взаємодіє тільки з визначеними серверними маршрутами.

Основні API-маршрути системи KinoFox можна подати так:

Таблиця 2.2

Основні API-маршрути серверної частини KinoFox

Група API	Маршрут	Призначення
Movie API	/api/movie/list	Отримання списку фільмів із можливістю фільтрації та сортування
Movie API	/api/movie/info/:id	Отримання детальної інформації про фільм за його ідентифікатором
Movie API	/api/movie/seach	Виконання семантичного пошуку контенту
Auth API	/api/auth/registration	Реєстрація нового користувача
Auth API	/api/auth/login	Авторизація користувача та створення сесії
Auth API	/api/auth/info	Отримання інформації про користувача та його сесію
Favourite API	/api/favourite	Додавання або видалення фільму зі списку обраного
History API	/api/history	Додавання або видалення запису історії переглядів
History API	/api/history/:id	Отримання історії переглядів конкретного користувача

Наявність таких маршрутів дозволяє чітко відокремити різні сценарії роботи системи. Наприклад, мобільний застосунок звертається до /api/movie/list, коли потрібно показати добірку контенту на головній сторінці, а маршрут /api/movie/info/:id використовується для сторінки детального перегляду. Маршрути авторизації застосовуються під час входу користувача та отримання даних профілю. Маршрути історії й обраного потрібні для персоналізованих функцій.

Окремо потрібно зазначити, що в поточній реалізації маршруту семантичного пошуку в коді використовується назва /api/movie/seach. З технічного погляду це робочий маршрут, але в подальшому його доцільно перейменувати на /api/movie/search, щоб назва відповідала англійському слову search і була зрозумілішою для підтримки. У дипломній роботі цей момент можна розглядати як невелику технічну неточність, яку варто врахувати під час удосконалення системи.

Авторизація користувачів у серверній частині реалізується через створення сесії. Під час реєстрації система перевіряє, чи не використовується вже вказаний

email, після чого створює нового користувача та формує сесію. Під час входу система перевіряє пароль і, якщо дані правильні, повертає клієнтській частині токен. Пароль не повинен зберігатися у відкритому вигляді, тому для нього використовується хешування [17]. Такий підхід зменшує ризики, пов’язані зі зберіганням облікових даних.

Для роботи із сесіями в KinoFox використовується токен, який передається клієнту після успішної авторизації. На сервері зберігається не сам токен у відкритому вигляді, а його хешоване представлення. Це є важливим елементом безпеки, оскільки навіть у випадку доступу до записів сесій зловмисник не отримує готовий токен для входу. Крім того, сесії мають строк дії, що дозволяє обмежити час використання одного токена.

Серверна частина також відповідає за роботу з каталогом контенту. Сервіс отримання списку фільмів підтримує параметри фільтрації: тип контенту, рейтинг, рік випуску, країну, жанри, а також сортування. Це важливо для онлайн-кінотеатру, оскільки каталог може містити багато різних позицій, і клієнтській частині не завжди доцільно отримувати всі дані одразу. Сервер може сформувати відповідь відповідно до параметрів запиту й передати клієнту тільки потрібний набір результатів.

Для сторінки детального перегляду використовується окремий сервіс отримання інформації про фільм за його ідентифікатором. Це зручно, тому що картка на головній сторінці не повинна містити всі дані про фільм. На етапі перегляду каталогу достатньо короткої інформації, а повний опис, перелік зображень або посилань для відтворення можна отримати тільки після переходу на детальну сторінку. Такий підхід зменшує обсяг даних, які передаються під час початкового завантаження списку.

У серверній частині також передбачено семантичний пошук. Його суть полягає в тому, що пошуковий запит перетворюється на векторне представлення, після чого система порівнює його з векторами, які збережені для фільмів у базі даних. Такий підхід дозволяє знаходити результати не тільки за точним збігом слів, а й за змістовою близькістю. У KinoFox для цього використовується OpenAI API

для створення embedding і можливості PostgreSQL/pgvector для порівняння векторів. Детальніше робота з базою даних і pgvector розглядається в наступному підрозділі.

Для документування API у серверній частині використовується Swagger. Наявність Swagger-документації є корисною під час розробки, оскільки вона дозволяє переглядати доступні маршрути, перевіряти параметри запитів і тестувати відповіді серверу. Для дипломного проєкту це також є перевагою, бо показує, що серверна частина має не випадковий набір маршрутів, а структурований API, який можна аналізувати та тестувати.

У таблиці 2.3 наведено основні засоби, які застосовуються для реалізації серверної частини системи KinoFox.

Таблиця 2.3

Засоби реалізації серверної частини системи KinoFox

Засіб	**Призначення у системі**
Bun	Запуск серверної частини, виконання TypeScript-коду, робота зі скриптами проєкту
Elysia	Побудова REST API, опис маршрутів і групування серверної логіки
TypeScript	Реалізація серверної логіки з підтримкою типізації
Drizzle ORM	Опис схем бази даних, виконання запитів і підтримка міграцій
PostgreSQL driver	Підключення серверної частини до PostgreSQL
bcrypt	Хешування та перевірка паролів користувачів
Swagger	Документування та перевірка API-маршрутів
OpenAI API	Формування векторних представлень для семантичного пошуку
dotenv	Робота зі змінними середовища без зберігання конфігурації в коді

Під час роботи з конфігураційними файлами важливо враховувати безпеку. У серверному проєкті використовуються змінні середовища для підключення до бази даних та зовнішніх сервісів. Такі значення не потрібно вказувати в тексті дипломної роботи або відкрито публікувати разом із кодом. У межах опису достатньо зазначити, що серверна частина використовує конфігураційні змінні для

підключення до бази даних і зовнішніх API. Це відповідає кращій практиці, коли секретні ключі та паролі не зберігаються безпосередньо в програмному коді.

Загалом обрані засоби реалізації серверної частини відповідають потребам системи KinoFox. Bun забезпечує запуск і виконання backend-застосунку, Elysia дозволяє організувати API, TypeScript робить код більш контрольованим, Drizzle ORM спрощує роботу з PostgreSQL, а допоміжні бібліотеки забезпечують авторизацію, документування, пошук і конфігурацію. У поєднанні ці засоби дають змогу побудувати серверну частину, яка обробляє запити мобільного застосунку, працює з базою даних і підтримує основні функції онлайн-кінотеатру.

2.4 Засоби збереження, обробки та пошуку даних

Для онлайн-кінотеатру база даних є однією з основних частин системи, оскільки саме в ній зберігається інформація про фільми, користувачів, сесії, історію переглядів і список обраного. Без правильно організованого збереження даних мобільний застосунок міг би тільки відображати статичну інформацію, але не мав би змоги працювати як повноцінний сервіс. Саме тому під час розробки KinoFox було важливо обрати таку систему управління базами даних, яка дозволяє працювати зі структурованою інформацією, підтримує зв’язки між таблицями та може бути розширена для реалізації пошуку.

У системі KinoFox для збереження даних використовується PostgreSQL. Це реляційна система управління базами даних, яка добре підходить для клієнт-серверних застосунків, де потрібно зберігати різні типи пов’язаних даних. Для онлайн-кінотеатру це особливо важливо, оскільки інформація про фільми не існує окремо від інших частин системи. Наприклад, користувач може додати певний фільм до обраного, переглянути його, а сервер повинен зберегти відповідний зв’язок між користувачем і контентом. У реляційній базі даних такі зв’язки можна організувати через окремі таблиці та зовнішні ключі.

PostgreSQL було обрано також через його підтримку складних типів даних. У випадку відеокаталогу частина полів може зберігати не одне значення, а набір

значень. Наприклад, фільм може мати кілька жанрів, кілька країн виробництва, декілька зображень або кілька посилань для відтворення. У KinoFox це враховано в структурі таблиці фільмів, де окремі поля можуть містити масиви даних. Такий підхід дозволяє зберігати опис контенту більш гнучко й не створювати надмірну кількість допоміжних таблиць для кожного спискового поля.

Основною таблицею в базі даних є таблиця movies. Вона містить інформацію про відеоконтент, який відображається в Android-застосунку. До цієї таблиці належать такі поля, як внутрішній ідентифікатор, ідентифікатор фільму, тип контенту, назва, постер, кадри, рейтинг, рік випуску, країна, жанри, тривалість, опис, вікове обмеження, список посилань для перегляду та векторне представлення. Така структура дозволяє зберігати як коротку інформацію для картки фільму, так і повні дані для сторінки детального перегляду.

Окреме значення має поле movie_id. Воно використовується як унікальний ідентифікатор контенту, який дозволяє знаходити конкретний фільм і не створювати дублікати під час автоматизованого збору даних. Якщо парсер повторно обробляє сторінку вже наявного фільму, система може визначити це за ідентифікатором і оновити запис замість створення нового. Для відеокаталогу це важливо, оскільки дублікати погіршують якість пошуку й ускладнюють роботу користувача з каталогом.

Крім таблиці фільмів, у базі даних передбачено таблицю користувачів. Вона зберігає дані облікового запису: ідентифікатор користувача, email, ім’я користувача та пароль у захищеному вигляді. Наявність окремої таблиці користувачів дозволяє реалізувати реєстрацію, авторизацію та подальшу роботу з персональними функціями. Саме через цю таблицю система може пов’язувати історію переглядів і список обраного з конкретним користувачем.

Для роботи з авторизацією використовується таблиця користувацьких сесій. Після успішного входу система створює сесію, яка має власний ідентифікатор, посилання на користувача та строк дії. Такий підхід дозволяє не перевіряти пароль під час кожної дії користувача. Замість цього клієнтська частина використовує

токен сесії, а сервер перевіряє його дійсність. Це робить роботу із застосунком зручнішою та водночас дозволяє контролювати доступ до персональних функцій.

Для збереження обраного та історії переглядів використовуються окремі таблиці зв’язків. Це логічно, оскільки один користувач може додати багато фільмів до обраного, а один і той самий фільм може бути доданий різними користувачами. Така сама ситуація виникає з історією переглядів. Тому ці дані не варто зберігати безпосередньо в таблиці користувача або фільму як простий текстовий список. Окремі таблиці дозволяють коректно описати зв’язок між користувачами та контентом.

У таблиці 2.4 наведено основні таблиці бази даних системи KinoFox та їх призначення.

Таблиця 2.4

Основні таблиці бази даних системи KinoFox

Таблиця	**Призначення**
movies	Збереження інформації про фільми, серіали, мультфільми та інший відеоконтент
users	Збереження даних користувачів, необхідних для реєстрації та авторизації
user_session	Збереження користувацьких сесій і строку їх дії
user_favorites	Збереження зв’язку між користувачами та фільмами, доданими до обраного
history	Збереження інформації про контент, з яким взаємодіяв користувач

Така структура бази даних відповідає основним потребам онлайн-кінотеатру. Вона не є надмірно складною, але дозволяє реалізувати базові сценарії роботи: перегляд каталогу, відкриття детальної сторінки, авторизацію, збереження сесій, роботу з історією та обраним. Крім того, її можна розширювати. Наприклад, у майбутньому можна додати таблиці для коментарів, оцінок користувачів, підписок або рекомендацій.

Для опису структури бази даних у серверній частині використовується Drizzle ORM. Схеми таблиць описуються в коді, що дозволяє підтримувати відповідність між програмною логікою та реальною структурою бази даних. Такий

підхід є зручним, оскільки розробник бачить, які поля має кожна таблиця, які типи даних використовуються та які зв’язки існують між сутностями. Крім того, Drizzle дозволяє створювати міграції, тобто контрольовані зміни структури бази даних.

Міграції відіграють важливу роль під час розробки. Якщо структура таблиць змінюється, ці зміни потрібно застосувати до бази даних без ручного створення всіх таблиць заново. У KinoFox міграції використовуються для створення таблиць, індексів і необхідних розширень. Завдяки цьому базу даних можна розгорнути в іншому середовищі більш передбачувано. Це важливо як для локальної розробки, так і для можливого розгортання серверної частини на віддаленому сервері.

Окремою особливістю системи KinoFox є використання pgvector [18]. Це розширення для PostgreSQL, яке дозволяє зберігати векторні представлення та виконувати пошук за схожістю. У звичайному текстовому пошуку система шукає збіг між словами в запиті та словами в назві або описі фільму. Такий підхід працює, але має обмеження: якщо користувач формулює запит не точно або використовує інші слова, потрібний фільм може не потрапити до результатів. Семантичний пошук вирішує цю проблему частково, тому що порівнює не тільки текст, а й змістове представлення.

У KinoFox для кожного фільму може зберігатися embedding — векторне представлення текстових даних [19]. Воно формується за допомогою зовнішнього API та записується в поле embedding таблиці movies. Коли користувач виконує пошук, його запит також може бути перетворений на вектор. Після цього система порівнює вектор запиту з векторами фільмів і визначає, які записи є найбільш близькими за змістом. Це дозволяє зробити пошук гнучкішим, ніж звичайне порівняння рядків.

Для прискорення такого пошуку в базі даних використовується індекс. У випадку векторного пошуку індексація має велике значення, оскільки без неї порівняння великої кількості векторів може бути повільним. У KinoFox для поля embedding створюється HNSW-індекс із використанням відповідного типу операцій для косинусної схожості. Це дозволяє ефективніше знаходити близькі за змістом записи в таблиці фільмів.

Семантичний пошук у системі не замінює повністю звичайні фільтри. Навпаки, ці механізми можуть доповнювати один одного. Наприклад, користувач може шукати фільм за змістовим запитом, а сервер додатково може враховувати тип контенту, рік, рейтинг або жанр. Для онлайн-кінотеатру це зручно, оскільки користувачі часто шукають контент не тільки за точною назвою, а й за загальним описом: жанром, настроєм, тематикою або схожістю на інші фільми.

Крім семантичного пошуку, серверна частина підтримує звичайну фільтрацію даних. Список фільмів може відбиратися за типом контенту, рейтингом, роком випуску, країною та жанрами. Також може використовуватися сортування за рейтингом або роком. Ці можливості важливі для каталогу, оскільки дозволяють не передавати клієнтській частині весь набір даних, а формувати відповідь відповідно до запиту користувача або логіки конкретного екрана.

Для обробки масивів, які використовуються в полях країни, жанрів, сцен або списку посилань для перегляду, PostgreSQL є зручним рішенням. Наприклад, один фільм може належати одночасно до кількох жанрів. Якщо потрібно знайти всі фільми певного жанру, сервер може сформувати запит, який перевірить наявність цього жанру в масиві. Так само можна працювати з країнами виробництва або іншими списковими характеристиками. Це спрощує модель даних і дозволяє швидко отримувати потрібні результати.

Збереження даних у KinoFox тісно пов’язане з модулем автоматизованого збору контенту. Парсер отримує інформацію про фільми, приводить її до потрібної структури й записує до бази даних. Якщо база вже містить запис із таким movie_id, дані можуть бути оновлені. Якщо запису немає, створюється новий. Завдяки цьому база даних не є статичним сховищем, а постійно поповнюється підготовленою інформацією, яку потім використовує API та мобільний застосунок.

Під час роботи з базою даних важливо враховувати цілісність даних. Наприклад, запис в історії переглядів не повинен існувати без користувача або без відповідного фільму. Так само запис в обраному має бути пов’язаний із конкретним користувачем і конкретним контентом. Для цього в базі даних використовуються

зв’язки між таблицями. Вони допомагають уникнути ситуацій, коли в системі залишаються «сирі» або непов’язані записи.

Ще одним важливим аспектом є безпека даних. У базі не повинні зберігатися паролі у відкритому вигляді. Серверна частина повинна зберігати хеш пароля та перевіряти його під час авторизації. Також не потрібно розкривати в дипломній роботі реальні значення змінних середовища, адрес підключення до бази або ключів зовнішніх сервісів. У тексті достатньо описати, що для підключення використовуються конфігураційні змінні, а секретні значення мають зберігатися окремо від основного коду.

У контексті розробки Android-застосунку база даних залишається прихованою від клієнта. Мобільний застосунок не виконує SQL-запити та не підключається до PostgreSQL напряму. Уся взаємодія відбувається через серверну частину. Це є правильним підходом, оскільки прямий доступ мобільного застосунку до бази даних був би небезпечним і незручним для підтримки. Сервер виступає проміжним рівнем, який контролює запити, перевіряє дані та повертає клієнту лише потрібну інформацію.

Таким чином, для збереження, обробки та пошуку даних у системі KinoFox використовується PostgreSQL із підтримкою Drizzle ORM і розширенням pgvector. PostgreSQL забезпечує надійне збереження структурованих даних, Drizzle ORM спрощує роботу серверної частини зі схемами й запитами, а pgvector дозволяє реалізувати семантичний пошук контенту. У поєднанні ці засоби створюють основу для роботи відеокаталогу, авторизації, історії переглядів, обраного та пошукових можливостей Android-застосунку.

2.5 Засоби автоматизованого збору контенту

Однією з особливостей системи KinoFox є наявність окремого модуля для автоматизованого збору контенту. Для онлайн-кінотеатру це має важливе значення, оскільки наповнення каталогу вручну потребує багато часу та постійного контролю. Кожен фільм або серіал має містити не тільки назву, а й постер, рейтинг,

рік випуску, країну, жанри, тривалість, опис, вікове обмеження, додаткові зображення та посилання для відтворення. Якщо додавати такі дані вручну, процес швидко стає однотипним і незручним, особливо коли каталог потрібно регулярно оновлювати.

Автоматизований збір контенту дає змогу частково усунути цю проблему. У системі KinoFox для цього використовується окремий Python-модуль, який працює незалежно від Android-застосунку та серверної частини. Його завдання полягає в тому, щоб отримати сторінки з даними про відеоконтент, виділити з них потрібну інформацію, привести її до єдиної структури та записати в базу даних. Завдяки цьому мобільний застосунок працює вже з підготовленими даними, які надходять із серверу через API.

Для реалізації модуля автоматизованого збору контенту було використано мову програмування Python. Вона добре підходить для таких задач, оскільки має велику кількість бібліотек для роботи з HTTP-запитами, HTML-сторінками, асинхронною взаємодією з базою даних і обробкою структурованих даних. У межах KinoFox Python використовується не для клієнтської частини, а саме для допоміжного модуля, який відповідає за формування та оновлення відеокаталогу.

Основна логіка парсера поділена на кілька частин. Окремий модуль відповідає за отримання посилань на сторінки з контентом, інший — за обробку конкретної сторінки фільму або серіалу, ще один — за запис отриманих даних у базу даних. Такий поділ є зручним, оскільки кожна частина має своє призначення. Наприклад, якщо змінюється спосіб пошуку посилань на сторінках каталогу, це не обов’язково потребує зміни логіки запису в базу даних. Якщо потрібно змінити структуру обробки сторінки фільму, можна працювати з відповідним модулем, не переписуючи весь парсер.

Першим етапом автоматизованого збору є отримання списку URL-адрес сторінок із контентом. Для цього використовується модуль, який звертається до сторінок каталогу, аналізує їхню HTML-структуру та знаходить посилання на окремі фільми або серіали. Також враховується наявність кількох сторінок у

каталозі, оскільки контент може бути розподілений за сторінками. Після цього отримані посилання передаються на наступний етап обробки.

Другим етапом є парсинг сторінки конкретного фільму. На цьому етапі модуль відкриває сторінку, аналізує її HTML-код і виділяє потрібні поля. До таких даних належать ідентифікатор фільму, тип контенту, назва, постер, зображення, рейтинг, рік випуску, країни, жанри, тривалість, опис, вікове обмеження та посилання для відтворення. Після отримання цих даних вони приводяться до структури, яку може прийняти база даних. Це важливо, оскільки інформація на вебсторінці не завжди подана в тому вигляді, у якому її зручно зберігати та використовувати в застосунку.

Під час автоматизованого збору даних потрібно враховувати, що не всі сторінки мають однакову структуру. Частина полів може бути відсутня, деякі значення можуть містити зайві символи або службовий текст, а окремі характеристики можуть бути записані в різному форматі. Наприклад, тривалість може містити не тільки число, а й текстове пояснення, жанри можуть бути подані як набір посилань, а опис може містити додаткові фрагменти, які не потрібні для відображення в застосунку. Тому парсер повинен не лише отримувати HTML-код, а й очищати та нормалізувати зібрані дані.

Для роботи з HTML-сторінками використовується BeautifulSoup. Ця бібліотека дозволяє аналізувати структуру HTML-документа та знаходити потрібні елементи за тегами, класами або іншими ознаками. У системі KinoFox вона застосовується для виділення назв, постерів, зображень, описів, жанрів та інших характеристик сторінки. Використання BeautifulSoup є доцільним, оскільки бібліотека має зрозумілий синтаксис і добре підходить для обробки сторінок, де дані розміщені в HTML-елементах.

Для виконання HTTP-запитів використовується бібліотека requests. Вона дозволяє отримувати вміст сторінок, який потім передається на обробку. У контексті KinoFox requests використовується на початкових етапах збору даних: для завантаження сторінок каталогу та сторінок окремих фільмів. Після отримання відповіді парсер перевіряє її та переходить до виділення потрібних даних.

Для опису структури зібраного контенту доцільно використовувати модель даних. У парсері KinoFox така модель містить поля, які відповідають майбутньому запису в базі даних: ідентифікатор, тип, назву, постер, список зображень, рейтинг, рік, країни, жанри, тривалість, опис, вікове обмеження, посилання для відтворення та векторне представлення. Наявність моделі допомагає підтримувати однакову структуру даних незалежно від того, з якої сторінки вони були отримані.

Окрему роль у модулі збору контенту відіграє взаємодія з базою даних. Після обробки сторінки отримані дані потрібно не просто зберегти, а правильно додати або оновити. Якщо фільм уже є в базі даних, система не повинна створювати дублікат. Для цього використовується унікальний ідентифікатор контенту. Якщо запис із таким ідентифікатором уже існує, дані оновлюються. Якщо запису немає, створюється новий. Такий підхід дозволяє підтримувати каталог у впорядкованому стані.

Для асинхронної роботи з PostgreSQL у парсері використовується asyncpg [20]. Це дає змогу виконувати операції з базою даних у зручному форматі та ефективно обробляти записи. У системі KinoFox база даних може ініціалізуватися на основі SQL-скрипта, після чого парсер додає або оновлює записи в таблиці фільмів. Завдяки цьому модуль збору контенту пов’язаний із основною структурою бази даних, але не залежить від Android-застосунку.

Важливою особливістю KinoFox є також використання векторних представлень для пошуку. Під час підготовки даних для контенту може формуватися embedding — числове векторне представлення текстової інформації. Надалі воно зберігається в базі даних і використовується для семантичного пошуку. Це означає, що модуль автоматизованого збору не тільки отримує основні поля фільму, а й готує дані для більш гнучкого пошукового механізму. Для цього використовується зовнішній API формування embeddings, а сам результат записується в поле, яке потім обробляється через pgvector.

У таблиці 2.5 наведено основні засоби, які використовуються для автоматизованого збору контенту в системі KinoFox.

Таблиця 2.5

Засоби автоматизованого збору контенту в системі KinoFox

Засіб	**Призначення у системі KinoFox**
Python	Реалізація окремого модуля збору, обробки та запису контенту
requests	Отримання HTML-сторінок із даними про відеоконтент
BeautifulSoup	Аналіз HTML-структури та виділення потрібних елементів сторінки
Pydantic-модель	Опис структури зібраних даних перед записом у базу
asyncpg	Асинхронна взаємодія парсера з PostgreSQL
PostgreSQL	Збереження підготовленої інформації про фільми та серіали
OpenAI API	Формування векторних представлень для подальшого семантичного пошуку
SQL-скрипт ініціалізації	Створення необхідної структури бази даних для роботи парсера

Процес автоматизованого збору контенту можна подати як послідовність взаємопов’язаних дій. Спочатку парсер отримує сторінки каталогу та формує список посилань на окремі одиниці контенту. Після цього кожне посилання обробляється окремо: програма відкриває сторінку, виділяє необхідні дані, очищає їх і формує об’єкт із єдиною структурою. Далі виконується перевірка наявності такого запису в базі даних. Якщо запис існує, він оновлюється, якщо ні — додається як новий. На завершальному етапі дані стають доступними для серверної частини, яка передає їх Android-застосунку через API.

Такий підхід має кілька переваг. По-перше, він зменшує обсяг ручної роботи під час наповнення каталогу. По-друге, дані зберігаються в єдиній структурі, що спрощує роботу серверної частини. По-третє, оновлення каталогу може виконуватися повторно без створення великої кількості дублікатів. По-четверте,

підготовка embedding на етапі збору контенту дозволяє одразу формувати основу для семантичного пошуку.

Разом із цим автоматизований збір контенту потребує обережного підходу. Необхідно враховувати, що джерела даних можуть змінювати структуру сторінок, тому парсер може потребувати оновлення. Також важливо перевіряти коректність зібраних даних, оскільки автоматичне отримання інформації не гарантує повної відсутності помилок. У реальному використанні доцільно поєднувати автоматизований збір із перевіркою або редагуванням окремих записів адміністратором. У межах дипломного проєкту основний акцент зроблено саме на демонстрації механізму автоматизованого формування каталогу.

Варто також враховувати правовий і етичний аспект роботи з контентом. Автоматизований збір даних має виконуватися з дозволених або відкритих джерел, а використання отриманих матеріалів повинно відповідати умовам цих джерел. У дипломній роботі модуль збору контенту розглядається як технічний засіб автоматизації наповнення каталогу метаданими, а не як спосіб порушення прав власників контенту. Для практичного використання такої системи необхідно додатково перевіряти права на використання відеоматеріалів, постерів, описів і посилань.

З погляду архітектури важливо, що модуль автоматизованого збору контенту не змішується з клієнтською частиною. Android-застосунок не повинен виконувати парсинг сторінок або записувати дані в базу. Він отримує вже готову інформацію через API. Це робить систему більш правильно організованою: парсер відповідає за підготовку даних, база даних — за їх збереження, серверна частина — за обробку запитів, а мобільний застосунок — за відображення інформації користувачеві.

Отже, засоби автоматизованого збору контенту є важливою частиною системи KinoFox. Використання Python, requests, BeautifulSoup, asyncpg, PostgreSQL і засобів формування векторних представлень дозволяє створити модуль, який отримує дані про відеоконтент, структурує їх і передає до бази даних. Завдяки цьому онлайн-кінотеатр не обмежується ручним наповненням каталогу, а має окремий механізм підготовки контенту для подальшого використання в Android-застосунку.

3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Проєктування загальної архітектури системи KinoFox

Для застосунку KinoFox архітектура має враховувати не лише роботу мобільного інтерфейсу, а й серверну частину, базу даних, модуль автоматизованого збору контенту та механізм пошуку. Це пов’язано з тим, що онлайн-кінотеатр не може функціонувати як ізольований мобільний застосунок. Він потребує централізованого збереження даних, обробки запитів, авторизації користувачів і регулярного наповнення відеокаталогу.

Загальна архітектура системи KinoFox побудована за клієнт-серверним принципом. Клієнтським рівнем є Android-застосунок, реалізований за допомогою Flutter. Він відповідає за відображення інтерфейсу, навігацію між сторінками, обробку дій користувача та надсилання запитів до серверної частини. Серверний рівень реалізовано на основі Bun, Elysia та TypeScript. Він приймає HTTP-запити, виконує перевірку даних, працює з авторизацією, звертається до бази даних і повертає клієнту відповідь у структурованому форматі.

Рівень збереження даних представлений базою PostgreSQL. У ній зберігається інформація про фільми, серіали, мультфільми, користувачів, сесії, історію переглядів та обране. Окремо використовується розширення pgvector, яке дає змогу зберігати векторні представлення контенту й застосовувати їх для семантичного пошуку. Такий підхід дозволяє не обмежуватися лише звичайним пошуком за назвою, а створити основу для пошуку за змістовою близькістю.

Додатковим компонентом системи є модуль автоматизованого збору контенту, реалізований мовою Python. Його завдання полягає в отриманні інформації про відеоконтент із визначених джерел, виділенні основних полів, очищенні даних і записі результатів до бази даних. Цей модуль не взаємодіє безпосередньо з користувачем, але він має важливе значення для роботи всієї

системи, оскільки саме завдяки йому формується каталог, який надалі відображається в Android-застосунку.

Загальну архітектуру системи можна умовно поділити на чотири основні частини: мобільний клієнт, серверне API, базу даних і модуль збору контенту. Кожна з цих частин має окрему відповідальність. Мобільний клієнт не виконує обробку бази даних напряму, сервер не займається відображенням інтерфейсу, база даних не відповідає за логіку користувацької взаємодії, а парсер не бере участі в роботі екранів застосунку. Завдяки такому поділу система є більш зрозумілою, підтримуваною та придатною для подальшого розширення.

На рисунку 3.1 подано загальну архітектуру системи KinoFox, де Android-застосунок взаємодіє із серверною частиною через REST API, серверна частина працює з PostgreSQL, а Python-парсер наповнює базу даних інформацією про відеоконтент.

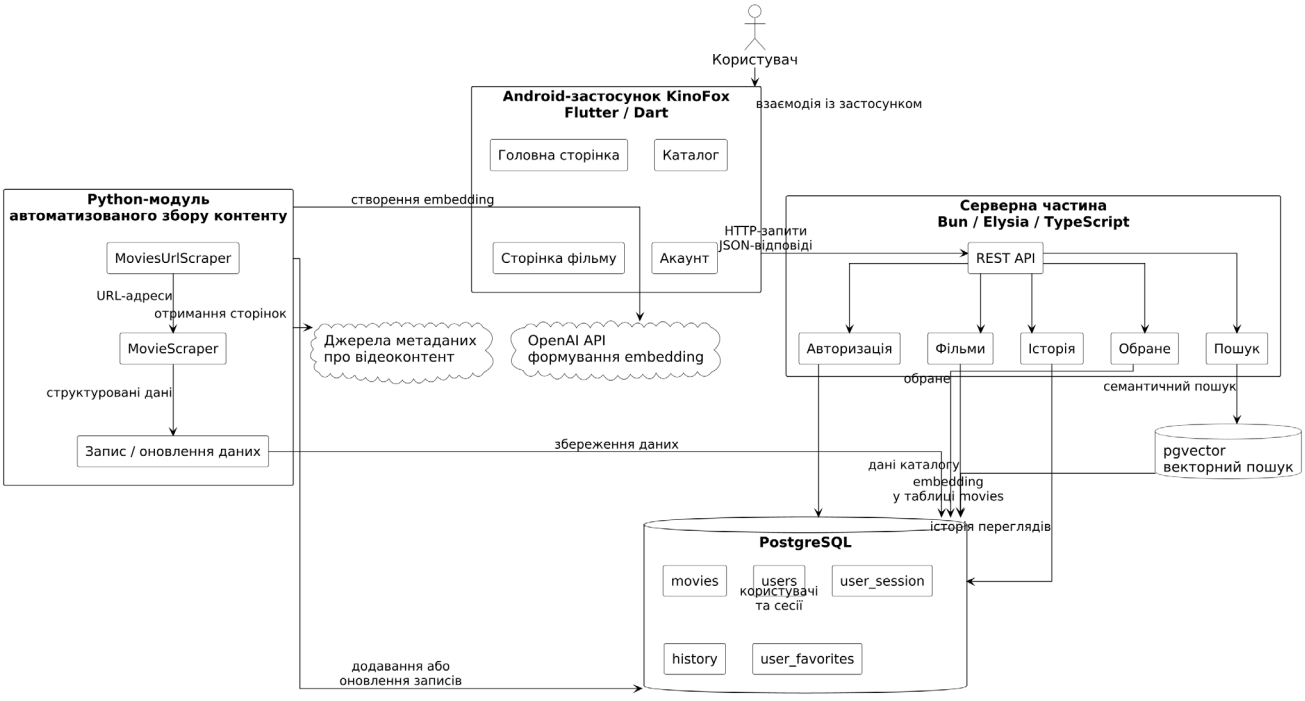


Рис. 3.1 Загальна архітектура системи KinoFox

Основний сценарій роботи системи починається з дії користувача в Android-застосунку. Наприклад, користувач відкриває головну сторінку, після чого

клієнтська частина надсилає запит до серверу для отримання списку фільмів, серіалів або мультфільмів. Сервер обробляє запит, звертається до бази даних, отримує потрібні записи та повертає відповідь у форматі JSON. Після цього застосунок перетворює отримані дані на об’єкти моделі та відображає їх у вигляді карток контенту.

Подібним чином працює сторінка детального перегляду. Коли користувач натискає на картку фільму, Android-застосунок передає серверу ідентифікатор вибраного контенту. Сервер знаходить відповідний запис у таблиці movies і повертає повну інформацію: назву, постер, рейтинг, рік випуску, країну, жанри, тривалість, опис і список посилань для перегляду. Клієнтська частина відображає ці дані на окремій сторінці, де користувач може ознайомитися з матеріалом і перейти до відтворення.

Авторизація користувача також відбувається через серверну частину. Коли користувач вводить email і пароль, мобільний застосунок надсилає ці дані до відповідного API-маршруту. Сервер перевіряє наявність користувача, порівнює пароль із хешованим значенням і, якщо дані правильні, створює сесію. Після цього клієнт отримує токен, який зберігається локально та використовується для подальших запитів. Завдяки цьому користувач може залишатися авторизованим після повторного відкриття застосунку.

Окремий сценарій пов’язаний із модулем автоматизованого збору контенту. На відміну від користувацьких дій, цей процес не запускається з мобільного інтерфейсу. Python-парсер отримує сторінки з відеоконтентом, виділяє потрібні поля, формує структурований об’єкт і записує його до PostgreSQL. Якщо запис із таким ідентифікатором уже існує, він оновлюється. Якщо запису немає, створюється новий. Після цього дані стають доступними для серверного API та можуть бути показані в Android-застосунку.

Зв’язок між компонентами системи можна описати через послідовність обміну даними. Android-застосунок надсилає HTTP-запити до API. API звертається до бази даних і повертає результат клієнту. Парсер, у свою чергу, наповнює базу

даних, але не взаємодіє з мобільним інтерфейсом напряму. Така схема дозволяє уникнути змішування відповідальностей і зробити систему більш контрольованою.

У таблиці 3.1 наведено основні компоненти системи KinoFox та їх призначення.

Таблиця 3.1

Основні компоненти системи KinoFox

Компонент	**Призначення**
Android-застосунок	Відображення інтерфейсу, навігація, робота з користувачем, надсилання запитів до API
REST API	Обробка запитів клієнта, авторизація, робота з фільмами, історією та обраним
PostgreSQL	Збереження інформації про користувачів, сесії, фільми, історію переглядів і обране
pgvector	Збереження векторних представлень і підтримка семантичного пошуку
Python-парсер	Автоматизований збір, очищення та запис інформації про відеоконтент
OpenAI API	Формування embedding для подальшого використання в пошуку

Однією з переваг такої архітектури є можливість незалежного розвитку окремих частин системи. Наприклад, інтерфейс Android-застосунку можна змінювати без повної переробки бази даних. Серверну частину можна доповнити новими API-маршрутами без зміни логіки парсера. Модуль збору контенту можна вдосконалювати окремо, якщо змінюється структура джерел даних або потрібно додати нові поля. Це робить архітектуру гнучкою для майбутнього розвитку.

Для дипломного проєкту важливо, що архітектура KinoFox демонструє повний цикл роботи з контентом. Дані не вводяться вручну безпосередньо в мобільний застосунок, а проходять кілька етапів: автоматизований збір, обробку, збереження в базі даних, отримання через API та відображення в інтерфейсі. Саме цей ланцюг відрізняє систему від простого статичного каталогу й дозволяє розглядати її як повноцінний Android-застосунок онлайн-кінотеатру з автоматизованим збором контенту.

Також така архітектура створює основу для подальшого масштабування. У майбутньому до системи можна додати адміністративну панель, рекомендації, розширену фільтрацію, коментарі, оцінки користувачів або підтримку інших платформ. Оскільки основна логіка зосереджена на сервері, а клієнтська частина працює через API, нові компоненти зможуть використовувати вже наявну базу даних і серверну інфраструктуру.

Отже, загальна архітектура системи KinoFox побудована так, щоб розділити відповідальність між клієнтською частиною, сервером, базою даних і модулем автоматизованого збору контенту. Android-застосунок забезпечує зручну взаємодію користувача з відеокаталогом, серверна частина обробляє запити й керує доступом до даних, PostgreSQL зберігає основну інформацію, а Python-парсер відповідає за наповнення каталогу. Такий підхід відповідає вимогам до онлайн-кінотеатру й забезпечує основу для подальшої реалізації окремих функціональних модулів.

3.2 Проєктування та реалізація клієнтської частини Android-застосунку

Клієнтська частина системи KinoFox є основним засобом взаємодії користувача з онлайн-кінотеатром. Саме через Android-застосунок користувач відкриває головну сторінку, переглядає добірки фільмів, переходить до детальної інформації про контент, авторизується, переглядає дані профілю та користується навігацією між основними розділами. Тому під час проєктування клієнтської частини головну увагу було приділено зрозумілій структурі екранів, зручності переходів, візуальному поданню контенту та коректній взаємодії із серверним API.

Клієнтська частина реалізована за допомогою Flutter, що дозволяє створити мобільний інтерфейс на основі системи віджетів. У межах проєкту KinoFox Flutter використовується для побудови всіх основних екранів застосунку, карток контенту, нижньої панелі навігації, форм авторизації та сторінки детального перегляду фільму. Логіка застосунку написана мовою Dart. Такий підхід є зручним, оскільки

Dart дозволяє описувати моделі даних, сервіси для API-запитів і компоненти інтерфейсу в межах єдиної клієнтської частини.

Структура Flutter-застосунку побудована так, щоб розділити основні частини інтерфейсу за призначенням. Головним файлом запуску є main.dart, у якому ініціалізується застосунок. Основна структура навігації та підключення головних сторінок зосереджені у файлі app.dart. Саме тут формується каркас мобільного застосунку, який містить нижню панель навігації та забезпечує перемикання між основними розділами. Така побудова дозволяє зберегти зрозумілу логіку роботи застосунку: головний файл відповідає за запуск, а окремий файл застосунку - за його основну структуру.

У клієнтській частині передбачено п’ять основних розділів: головна сторінка, каталог, улюблене, історія та акаунт. Така структура відповідає типовому сценарію роботи користувача онлайн-кінотеатру. Головна сторінка потрібна для швидкого доступу до добірок контенту. Каталог має забезпечувати загальний перегляд доступних матеріалів. Розділ улюбленого призначений для збереження цікавих фільмів, історія - для повернення до раніше відкритого контенту, а акаунт - для авторизації та перегляду інформації про користувача.

Нижня панель навігації є зручним рішенням для Android-застосунку, оскільки основні розділи завжди залишаються доступними для користувача. У KinoFox вона містить україномовні назви розділів: «Головна», «Каталог», «Улюблені», «Історія», «Акаунт». Такий підхід робить застосунок зрозумілим для користувача без додаткового навчання. Користувач може швидко перейти до потрібного екрана, не відкриваючи складних меню та не повертаючись багато разів назад.

На рисунку 3.2 подано схему структури клієнтської частини Android-застосунку KinoFox, у якій показано зв’язок між основним файлом запуску, головним модулем застосунку, сторінками, моделями даних, API-сервісами та допоміжним сховищем сесії.

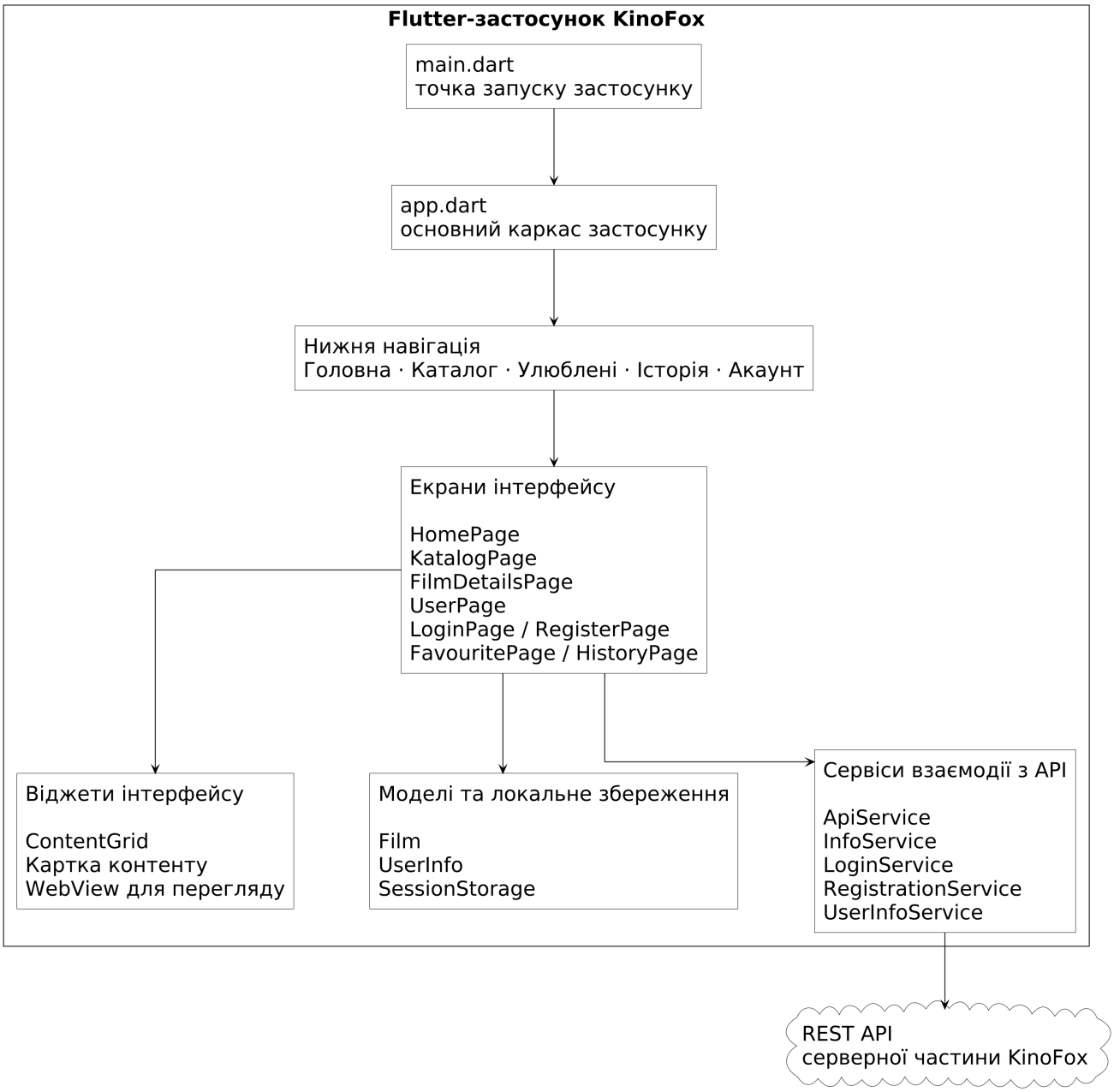


Рис. 3.2 Структура клієнтської частини Android-застосунку KinoFox

Візуальне оформлення застосунку виконано в темній кольоровій гамі з помаранчевим акцентом. Такий вибір є доречним для онлайн-кінотеатру, оскільки темний фон не відволікає від постерів і відеоконтенту, а акцентний колір використовується для виділення активних елементів, кнопок і важливих частин інтерфейсу. У застосунку також використовуються шрифти e-Ukraine та e-Ukraine Head, що підтримує єдиний стиль україномовного інтерфейсу.

Головна сторінка застосунку є стартовим екраном для користувача. Вона містить кілька блоків із відеоконтентом, зокрема фільми, серіали та мультфільми. Такий поділ спрощує сприйняття каталогу, оскільки користувач одразу бачить різні типи матеріалів. Кожен блок отримує дані через відповідний API-сервіс і передає їх до компонента відображення карток. У результаті головна сторінка не є статичним екраном, а формується на основі даних, які надходять із серверної частини.

Для подання списку контенту використовується окремий компонент сітки. Він відповідає за відображення фільмів у вигляді карток. Картка містить постер, назву, рейтинг, тривалість і жанри. Такий формат є зручним для мобільного екрана, оскільки користувач бачить достатньо інформації для первинного вибору, але екран не перевантажується зайвим текстом. Якщо користувач хоче дізнатися більше, він натискає на картку та переходить до сторінки детального перегляду.

Для роботи з даними про відеоконтент у клієнтській частині створено модель Film. Вона описує структуру об’єкта, який надходить із серверної частини. До моделі входять поля для ідентифікатора, типу контенту, назви, постера, рейтингу, року випуску, країни, жанрів, тривалості, опису та посилань для відтворення. Наявність окремої моделі дозволяє зручно перетворювати JSON-відповідь серверу на об’єкт Dart і використовувати його на різних сторінках застосунку.

Окрему роль відіграють сервіси для роботи з API. Вони відповідають за надсилання HTTP-запитів до серверної частини та отримання відповідей. Наприклад, для головної сторінки використовуються запити для отримання списків фільмів, серіалів і мультфільмів. Для сторінки детального перегляду використовується сервіс, який отримує повну інформацію про фільм за його movieId. Такий поділ є правильним, оскільки сторінки інтерфейсу не повинні самостійно містити всю логіку мережевої взаємодії.

Сторінка детального перегляду фільму є однією з найважливіших у клієнтській частині. Вона відкривається після натискання на картку контенту та отримує ідентифікатор вибраного фільму. Після цього застосунок звертається до серверної частини й отримує детальні дані. На сторінці відображаються постер,

назва, рейтинг, рік випуску, тривалість, жанри, країна, опис сюжету та доступні посилання для перегляду. Завдяки цьому користувач має достатньо інформації, щоб прийняти рішення щодо перегляду.

Для відображення відеоконтенту в застосунку використовується вбудований WebView-компонент. Це дозволяє відкривати iframe або вебплеєр без переходу до зовнішнього браузера. Такий підхід робить взаємодію із застосунком більш цілісною, оскільки користувач залишається в межах KinoFox. Водночас робота з WebView потребує коректної обробки посилань і перевірки того, що джерело відтворення доступне для завантаження.

Авторизація в клієнтській частині реалізована через сторінку входу. Користувач вводить email і пароль, після чого застосунок надсилає ці дані до серверного API. Якщо авторизація проходить успішно, сервер повертає токен сесії. Цей токен зберігається локально за допомогою механізму shared_preferences. Завдяки цьому користувач може повторно відкривати застосунок без необхідності щоразу вводити дані для входу.

Крім входу, у клієнтській частині передбачено сторінку реєстрації. Вона дозволяє створити новий обліковий запис і передати введені дані до серверної частини. Реєстрація є важливою для реалізації персоналізованих функцій, оскільки без акаунта неможливо пов’язати історію переглядів або список обраного з конкретним користувачем. Після реєстрації користувач може перейти до входу та працювати із застосунком уже як авторизований користувач.

Сторінка акаунта відповідає за відображення інформації про користувача та його сесію. Вона перевіряє наявність збереженого токена й виконує запит до серверу для отримання даних профілю. Якщо токен відсутній або недійсний, користувачеві потрібно авторизуватися повторно. Такий підхід дозволяє поєднати зручність локального збереження сесії з контролем її дійсності на серверному боці.

Сторінки «Улюблені» та «Історія» передбачені як окремі частини навігаційної структури. Вони логічно відповідають персоналізованим функціям онлайн-кінотеатру. У поточній структурі застосунку ці розділи вже виділені в інтерфейсі, що дозволяє надалі пов’язати їх із відповідними API-маршрутами

серверної частини. Такий підхід показує, що клієнтська частина спроєктована з урахуванням подальшого розвитку.

У таблиці 3.2 наведено основні екрани клієнтської частини KinoFox та їх призначення.

Таблиця 3.2

Основні екрани клієнтської частини KinoFox

Екран	**Призначення**
Головна сторінка	Відображення добірок фільмів, серіалів і мультфільмів
Каталог	Перегляд загального списку відеоконтенту
Сторінка фільму	Відображення детальної інформації про вибраний контент і запуск перегляду
Улюблені	Розділ для збереженого користувачем контенту
Історія	Розділ для контенту, з яким користувач взаємодіяв раніше
Вхід	Авторизація користувача за email і паролем
Реєстрація	Створення нового облікового запису
Акаунт	Відображення інформації про користувача, сесію та вихід із профілю

Під час проєктування клієнтської частини важливо було не змішувати інтерфейс, моделі та мережеву логіку. Саме тому сторінки відповідають за відображення даних, моделі - за опис структури об’єктів, а сервіси - за отримання інформації із серверу. Такий підхід робить код зрозумілішим і полегшує подальшу підтримку. Наприклад, якщо зміниться формат відповіді API, потрібно буде оновити модель або сервіс, але не обов’язково переписувати всі екрани.

Клієнтська частина також враховує потребу в конфігурації. Адреса серверної частини не повинна бути жорстко прописана в усіх файлах застосунку. Для цього використовується змінна середовища, яка містить базову адресу API. Такий підхід полегшує перехід між локальним середовищем розробки та віддаленим сервером. У тексті роботи не потрібно вказувати реальні значення таких змінних, достатньо описати сам принцип їх використання.

Важливим елементом є обробка станів завантаження. Оскільки більшість даних надходить із серверу, застосунок повинен враховувати, що відповідь може

надходити із затримкою або не надійти взагалі. Для користувача це має бути зрозуміло: інтерфейс повинен показувати стан завантаження, а у випадку помилки - відповідне повідомлення або можливість повторити дію. Це особливо актуально для мобільного застосунку, де якість інтернет-з’єднання може змінюватися.

Отже, клієнтська частина Android-застосунку KinoFox спроєктована як окремий модуль клієнт-серверної системи, що відповідає за інтерфейс, навігацію, відображення каталогу, роботу зі сторінкою фільму, авторизацію та локальне збереження сесії. Її реалізація на Flutter і Dart дозволяє створити зручний мобільний інтерфейс, який взаємодіє із серверною частиною через API та відображає дані, підготовлені в базі даних. Така структура робить застосунок зрозумілим для користувача і придатним для подальшого розширення функціональності.

3.3 Проєктування та реалізація серверної частини, бази даних і REST API

Серверна частина системи KinoFox виконує роль проміжного рівня між Android-застосунком, базою даних і допоміжними модулями обробки контенту. Її основне призначення полягає в тому, щоб приймати запити від клієнтської частини, виконувати необхідну логіку, звертатися до бази даних і повертати результат у структурованому форматі. Такий підхід дозволяє не надавати мобільному застосунку прямого доступу до бази даних і водночас забезпечити централізовану обробку даних, авторизації, пошуку, історії переглядів та списку обраного.

Під час проєктування серверної частини було враховано, що онлайн-кінотеатр працює з кількома групами даних. До них належать дані про фільми, користувачів, сесії, обране, історію переглядів та пошукові запити. Через це серверна логіка не повинна бути зосереджена в одному файлі. У системі KinoFox вона розділена на окремі маршрути, сервіси, схеми бази даних і допоміжні утиліти. Така структура робить backend більш зрозумілим і дозволяє окремо розвивати різні функціональні частини.

Точкою входу серверної частини є файл src/index.ts. У ньому створюється серверний застосунок на основі Elysia з префіксом /api. Після цього до основного застосунку підключаються окремі групи маршрутів. Така організація зручна тим, що всі запити до серверу мають спільний початок адреси, а далі розділяються за призначенням. Наприклад, маршрути для роботи з фільмами розміщені в групі movie, маршрути для авторизації - в групі auth, а окремі маршрути відповідають за обране та історію.

Серверна частина побудована за принципом REST API. Це означає, що клієнтський застосунок надсилає HTTP-запити до певних маршрутів, а сервер повертає відповідь у форматі JSON. Для Android-застосунку такий підхід є зручним, оскільки Flutter-клієнт може отримувати дані у вигляді об'єктів, перетворювати їх на моделі Dart і відображати на екранах. У свою чергу, серверна частина не залежить від конкретного інтерфейсу, а лише надає дані через визначені маршрути.

Основними групами API в системі є робота з фільмами, авторизація користувача, обране та історія переглядів. Маршрути фільмів відповідають за отримання списку контенту, детальної інформації про окремий фільм і виконання пошуку. Маршрути авторизації забезпечують реєстрацію, вхід до системи та отримання інформації про поточну сесію. Маршрути обраного й історії призначені для персоналізованих функцій, які пов'язують конкретного користувача з певним відеоконтентом.

У таблиці 3.3 наведено основні групи серверних модулів системи KinoFox.

Робота з фільмами є однією з головних частин серверної логіки. Для цього в API передбачено маршрут отримання списку контенту. Він використовується Android-застосунком для формування головної сторінки, блоків фільмів, серіалів і мультфільмів. Серверна частина може приймати параметри запиту, які визначають тип контенту, рейтинг, рік, країну, жанри, порядок сортування та параметри пагінації. Завдяки цьому мобільний застосунок не повинен отримувати весь каталог одразу, а може запитувати саме той набір даних, який потрібен для конкретного екрана.

Таблиця 3.3

Основні серверні модулі системи KinoFox

Серверний модуль	Призначення
movie	Отримання списку фільмів, детальної інформації та пошук контенту
auth	Реєстрація, авторизація користувача та перевірка сесії
favourite	Додавання та видалення фільмів зі списку обраного
history	Збереження й отримання історії взаємодії користувача з контентом
db	Підключення до PostgreSQL і робота зі схемами бази даних
utils	Допоміжні функції для сесій, токенів, хешування та зовнішніх сервісів

Окремо реалізовано маршрут отримання детальної інформації про фільм за його ідентифікатором. Цей маршрут використовується тоді, коли користувач натискає на картку контенту в мобільному застосунку. На цьому етапі клієнтська частина передає серверу movieId, а сервер знаходить відповідний запис у базі даних і повертає повну інформацію. До такої інформації належать назва, постер, рейтинг, рік випуску, країна, жанри, тривалість, опис, вікове обмеження, зображення та список доступних посилань для перегляду.

Пошуковий маршрут у серверній частині реалізує пошук контенту за змістовим запитом. У поточній структурі коду він має адресу /api/movie/seach. Технічно такий маршрут працює як окремий endpoint, однак у подальшому його доцільно перейменувати на /api/movie/search, щоб назва відповідала стандартному англійському написанню слова search. Це не змінює загального принципу роботи системи, але покращує зрозумілість API та підтримку коду.

Для реалізації авторизації використовується окрема група маршрутів. Під час реєстрації користувач передає email, ім'я користувача та пароль. Сервер перевіряє, чи не існує вже користувача з таким email, після чого створює новий запис у таблиці користувачів. Пароль не зберігається у відкритому вигляді, а проходить хешування.

Це важливо, оскільки зберігання паролів як звичайного тексту створювало б значні ризики для безпеки.

Під час входу до системи користувач передає email і пароль. Сервер знаходить відповідний запис користувача та перевіряє пароль за допомогою механізму порівняння з хешованим значенням. Якщо дані правильні, створюється сесія користувача. Клієнтська частина отримує токен, який надалі зберігається локально в Android-застосунку. При наступних зверненнях до серверу цей токен може використовуватися для підтвердження особи користувача.

Окремої уваги потребує механізм сесій. У базі даних зберігається не сам токен у відкритому вигляді, а його хешоване значення. Це зменшує ризики у випадку несанкціонованого доступу до таблиці сесій. Кожна сесія також має строк дії. Якщо сесія застаріла, користувач повинен авторизуватися повторно. Такий підхід забезпечує баланс між зручністю користування та безпекою.

Серверна частина також відповідає за персоналізовані функції. Список обраного дозволяє зберігати фільми, які користувач хоче переглянути пізніше. Історія переглядів дозволяє фіксувати контент, з яким користувач уже взаємодіяв. Для реалізації цих функцій використовуються окремі таблиці зв'язків між користувачами та фільмами. Це правильніше, ніж зберігати всі ідентифікатори фільмів безпосередньо в таблиці користувача, оскільки такий підхід краще відповідає реляційній моделі даних.

На рисунку 3.3 подано схему взаємодії серверної частини з Android-застосунком і базою даних. На схемі потрібно показати, що мобільний застосунок надсилає запити до REST API, серверна частина звертається до PostgreSQL, а результат повертається назад клієнту у форматі JSON.

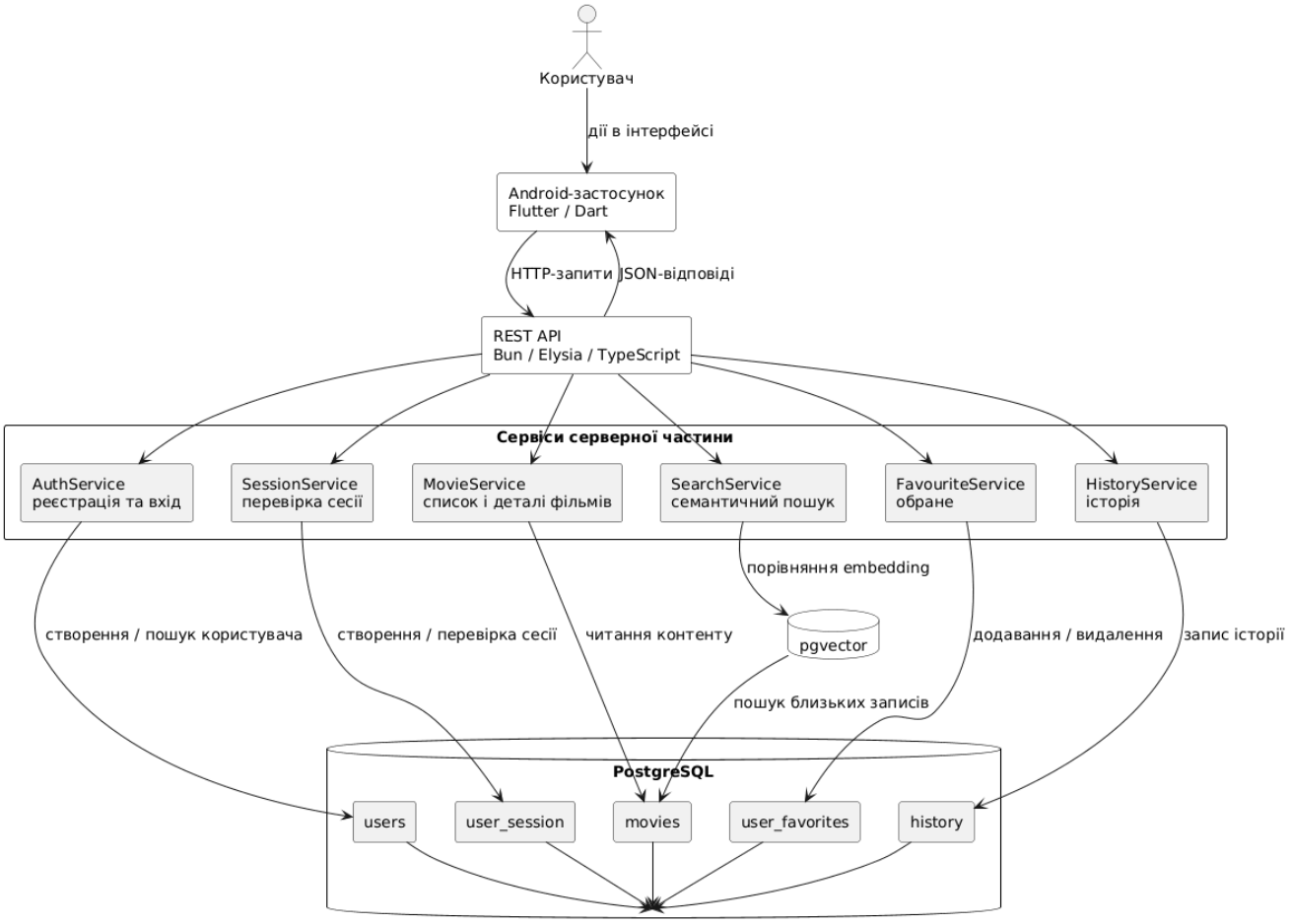


Рис. 3.3 Схема взаємодії серверної частини, бази даних і Android-застосунку KinoFox

База даних системи проєктується навколо кількох основних сутностей: фільм, користувач, сесія, обране та історія. Центральною таблицею для відеокаталогу є movies. Вона містить основні дані про контент, який відображається в застосунку. Таблиця users зберігає облікові записи користувачів. Таблиця user_session використовується для збереження активних сесій. Таблиці user_favorites і history описують зв'язки між користувачами та фільмами.

Структура таблиці movies враховує потреби онлайн-кінотеатру. Для короткого відображення в каталозі потрібні назва, постер, рейтинг, жанри та тривалість. Для сторінки детального перегляду потрібні рік випуску, країна, опис, вікове обмеження, зображення та playlist-посилання. Окреме поле embedding використовується для збереження векторного представлення, яке потрібне для

семантичного пошуку. Завдяки цьому одна таблиця забезпечує дані і для простого перегляду каталогу, і для розширених пошукових можливостей.

У таблиці 3.4 наведено основні сутності бази даних і зв'язки між ними

Таблиця 3.4

Сутності бази даних системи KinoFox

Сутність	**Основне призначення**	**Зв’язки**
movies	Збереження відеоконтенту та його характеристик	Пов’язується з історією та обраним
users	Збереження облікових записів користувачів	Пов’язується із сесіями, історією та обраним
user_session	Збереження активних користувацьких сесій	Належить конкретному користувачу
user_favorites	Збереження обраних фільмів користувача	Пов’язує користувача з фільмом
history	Збереження історії взаємодії з контентом	Пов’язує користувача з фільмом

Для опису структури бази даних на серверному рівні використовується Drizzle ORM. Схеми таблиць описані в окремих файлах, що дозволяє підтримувати порядок у коді. Наприклад, структура таблиці фільмів розміщується окремо від структури таблиці користувачів або сесій. Це полегшує підтримку й дозволяє швидше знаходити потрібні частини коду під час внесення змін.

Взаємодія серверної частини з базою даних відбувається через підключення до PostgreSQL. У серверному коді створюється пул підключень, який дозволяє виконувати запити до бази більш контрольовано. Це важливо для застосунку, де одночасно можуть виконуватися різні типи запитів: отримання каталогу, авторизація, перегляд профілю, додавання до обраного або пошук. Пул підключень допомагає уникнути нераціонального створення великої кількості окремих з'єднань.

Для створення й оновлення структури бази даних використовуються міграції. Міграції описують, які таблиці, поля, індекси та розширення мають бути створені.

У KinoFox через міграції створюються основні таблиці та підключається розширення для роботи з векторними представленнями. Такий підхід зручний тим, що структуру бази можна відтворити в іншому середовищі без ручного створення всіх таблиць.

Серверна частина також містить логіку пошуку. Звичайний пошук може працювати за конкретними параметрами, наприклад типом контенту, роком, рейтингом, країною або жанром. Семантичний пошук використовує інший підхід: текстовий запит перетворюється на embedding, після чого порівнюється з векторними представленнями фільмів у базі даних. Це дозволяє знаходити контент не тільки за прямим збігом слів, а й за змістовою близькістю.

З погляду реалізації API важливо, щоб відповіді серверу були передбачуваними для клієнтської частини. Android-застосунок очікує отримати дані в певній структурі. Якщо сервер повертає список фільмів, кожен елемент повинен мати поля, які може обробити модель Film. Якщо сервер повертає інформацію про користувача, вона повинна відповідати моделі профілю. Якщо виникає помилка, сервер має повертати зрозуміле повідомлення, а не неочікувану структуру даних. Це спрощує обробку відповідей на клієнтському рівні.

Для розробки та перевірки API використовується Swagger. Він дозволяє переглядати доступні маршрути й тестувати запити до серверу. Це корисно під час створення мобільного застосунку, оскільки розробник може перевірити, яку відповідь повертає API, ще до повного підключення відповідного екрана в Flutter. Наявність документації також полегшує подальшу підтримку серверної частини.

Безпека серверної частини пов'язана не тільки з хешуванням паролів, а й із правильним використанням конфігураційних даних. Підключення до бази даних і ключі зовнішніх сервісів не повинні бути відкрито розміщені в тексті роботи або в публічному доступі. Для цього використовуються змінні середовища. У межах дипломного опису достатньо зазначити, що серверна частина отримує такі значення з конфігурації, а конкретні секретні дані не наводяться.

Реалізована серверна частина відповідає основним потребам системи KinoFox. Вона забезпечує отримання та передавання даних про фільми, підтримує

авторизацію користувачів, працює із сесіями, має маршрути для обраного та історії, а також містить основу для пошуку. Завдяки розділенню коду на маршрути, сервіси, схеми й утиліти backend можна підтримувати та розширювати без повної переробки всієї системи.

Отже, серверна частина, база даних і REST API у системі KinoFox спроєктовані як взаємопов'язані компоненти клієнт-серверної архітектури. Сервер приймає запити від Android-застосунку, виконує необхідну логіку, звертається до PostgreSQL і повертає результат у форматі JSON. База даних зберігає основні сутності системи, а API забезпечує зручний спосіб доступу до цих даних. Така реалізація дозволяє мобільному застосунку працювати з актуальним каталогом, підтримувати користувацькі функції та використовувати пошукові можливості без прямої взаємодії з базою даних.

3.4 Проєктування та реалізація модуля автоматизованого збору контенту

Модуль автоматизованого збору контенту є однією з ключових частин системи KinoFox, оскільки саме він відповідає за формування відеокаталогу. Без такого модуля наповнення онлайн-кінотеатру довелося б виконувати вручну: окремо додавати назву фільму, постер, рейтинг, рік випуску, жанри, країну, опис, тривалість, вікове обмеження та посилання для перегляду. Для невеликої кількості записів такий підхід ще може бути прийнятним, але для повноцінного каталогу він стає незручним і потребує багато часу. Саме тому в системі KinoFox передбачено окремий Python-модуль, який автоматизує отримання та збереження інформації про відеоконтент.

Проєктування модуля автоматизованого збору контенту виконувалося з урахуванням загальної архітектури системи. Цей модуль не є частиною Android-застосунку і не взаємодіє безпосередньо з користувачем. Його завдання полягає у підготовці даних для бази PostgreSQL. Після цього серверна частина отримує ці дані з бази й передає їх мобільному застосунку через REST API. Такий поділ є

логічним, оскільки мобільний клієнт повинен відповідати за інтерфейс, сервер - за обробку запитів, база даних - за збереження інформації, а парсер - за наповнення каталогу.

Загальний сценарій роботи системи з погляду користувача можна подати через діаграму варіантів використання. На ній доцільно відобразити основних акторів: користувача, серверну частину та модуль автоматизованого збору контенту. Користувач взаємодіє із застосунком: переглядає каталог, відкриває сторінку фільму, авторизується, переглядає профіль, додає контент до обраного або звертається до історії. Модуль збору контенту, на відміну від користувача, виконує службові дії: отримує дані, обробляє їх і записує до бази даних.

З технічного погляду модуль збору контенту складається з кількох логічних частин. Перша частина відповідає за отримання URL-адрес сторінок із контентом. Друга частина відкриває кожну сторінку та виділяє з неї потрібні дані. Третя частина формує структурований об'єкт фільму або серіалу. Четверта частина виконує запис у базу даних: створює новий запис або оновлює вже наявний. Такий поділ дозволяє не змішувати різні завдання в одному фрагменті коду.

У структурі парсера KinoFox для отримання посилань використовується модуль MoviesUrlScraper. Він звертається до сторінок каталогу, визначає кількість сторінок і знаходить посилання на окремі одиниці контенту. Це необхідно для того, щоб не обробляти вручну кожну сторінку фільму. Після формування списку URL-адрес ці посилання передаються до наступного етапу, де вже виконується детальний аналіз кожної сторінки.

Обробка окремої сторінки виконується модулем MovieScraper. Його завдання полягає в тому, щоб отримати HTML-код сторінки, знайти в ньому потрібні елементи та сформувати об'єкт із даними про фільм. До таких даних належать ідентифікатор, тип контенту, назва, постер, зображення, рейтинг IMDb, рік випуску, країни, жанри, тривалість, опис, вікове обмеження та playlist-посилання. Саме ці поля в подальшому використовуються в серверній частині й Android-застосунку.

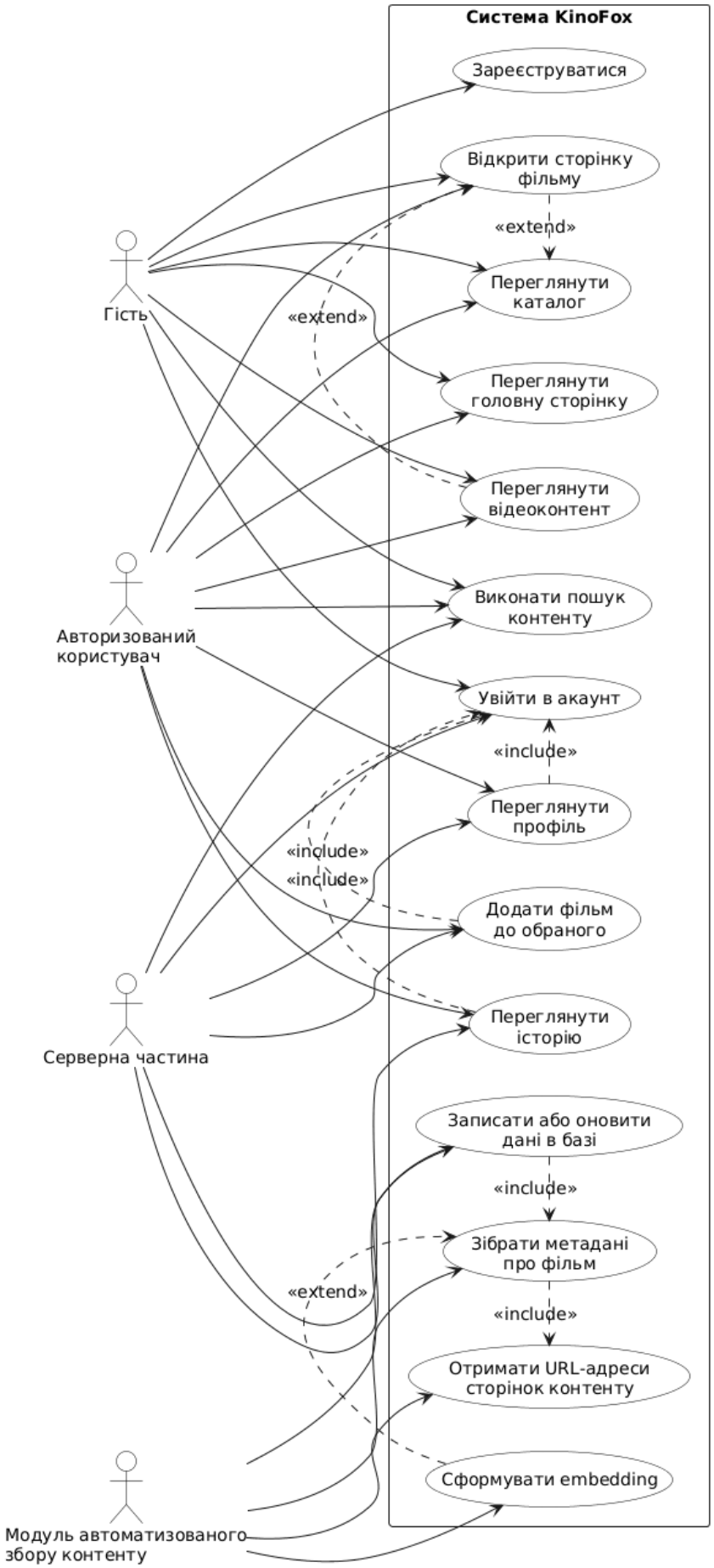


Рис. 3.4 Діаграма варіантів використання системи KinoFox

Під час парсингу важливо не просто отримати текст зі сторінки, а привести його до придатного для збереження вигляду. Наприклад, назва може містити зайві слова, опис може включати службові фрагменти, а списки жанрів або країн потрібно перетворити у формат, який можна зберігати в базі даних. Через це модуль збору контенту виконує не лише отримання даних, а й їх первинне очищення та структурування.

Послідовність взаємодії між компонентами під час отримання контенту доцільно подати у вигляді діаграми послідовності. На такій діаграмі потрібно показати, як Python-парсер звертається до джерела даних, отримує HTML-сторінку, передає її на обробку, формує об'єкт контенту, перевіряє наявність запису в PostgreSQL і виконує додавання або оновлення даних. Така діаграма дозволяє краще пояснити, що модуль збору працює як окремий технічний процес і не залежить від дій користувача в мобільному застосунку.

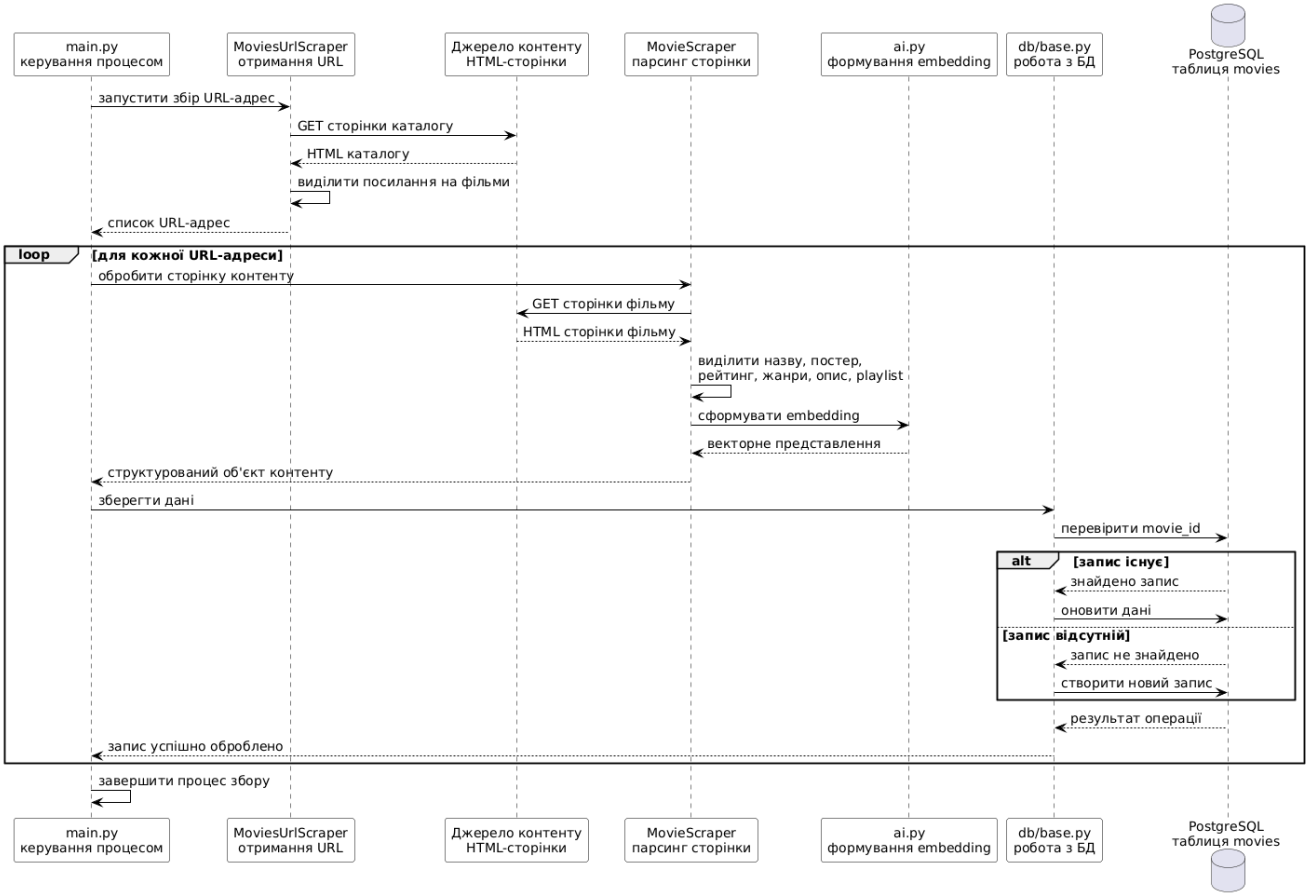


Рис. 3.5 Діаграма послідовності автоматизованого збору контенту

Для опису зібраних даних використовується модель, яка задає єдину структуру майбутнього запису. Це важливо, тому що дані з різних сторінок можуть мати неоднаковий вигляд. Якщо не привести їх до спільної структури, серверній частині буде складно обробляти такі записи. У KinoFox модель контенту містить поля, які відповідають таблиці movies у базі даних. Завдяки цьому дані, отримані парсером, можна одразу передавати до шару роботи з PostgreSQL.

Окремим елементом реалізації є формування векторного представлення для пошуку. Для цього використовується допоміжний модуль, який звертається до зовнішнього API та отримує embedding для текстової інформації. У подальшому це значення зберігається в полі embedding таблиці movies. Такий підхід дозволяє підготувати дані не тільки для звичайного відображення в каталозі, а й для семантичного пошуку, який враховує зміст запиту.

Запис даних до PostgreSQL виконується через окремий модуль роботи з базою. Він відповідає за підключення до бази, ініціалізацію структури та створення або оновлення записів. У системі KinoFox використовується принцип, за яким фільм не додається повторно, якщо запис із таким movie_id уже існує. У такому випадку дані оновлюються. Якщо ж такого запису немає, створюється новий. Це дозволяє уникнути дублювання контенту в каталозі.

Алгоритм роботи модуля автоматизованого збору контенту включає запуск Python-модуля, отримання списку сторінок каталогу, виділення URL-адрес окремих фільмів або серіалів, послідовну обробку кожної сторінки, отримання основних характеристик контенту, очищення та нормалізацію даних, формування об'єкта з єдиною структурою, створення embedding, перевірку наявності запису в базі даних і створення нового запису або оновлення наявного.

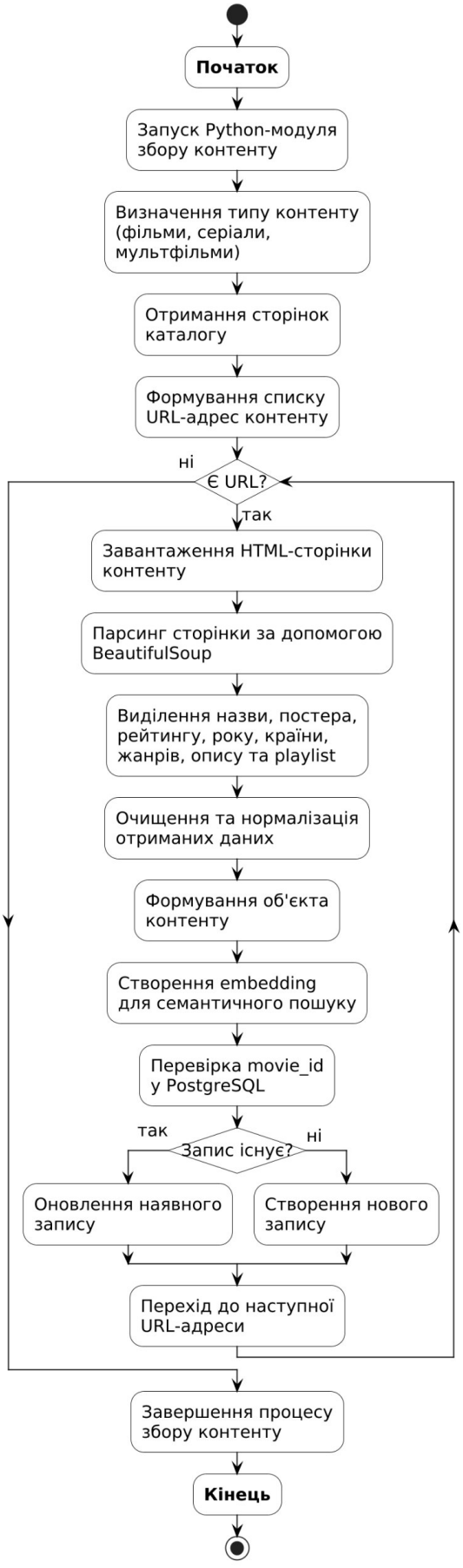


Рис. 3.6 Блок-схема роботи модуля автоматизованого збору контенту

У таблиці 3.5 наведено основні файли та модулі, які використовуються для реалізації автоматизованого збору контенту.

Таблиця 3.5

Основні модулі автоматизованого збору контенту KinoFox

Модуль	**Призначення**
main.py	Запуск процесу збору контенту та керування основним циклом обробки
MoviesUrlScraper.py	Отримання URL-адрес сторінок із відеоконтентом
MovieScraper.py	Парсинг окремої сторінки фільму або серіалу
db/base.py	Підключення до бази даних, ініціалізація та запис контенту
postgres/postgres.py	Допоміжні операції для роботи з PostgreSQL
ai.py	Формування embedding для семантичного пошуку
init_db.sql	SQL-скрипт для створення початкової структури бази даних

Основний файл main.py запускає процес збору контенту. У ньому визначається тип контенту, який потрібно обробити, після чого виконується отримання посилань і послідовна обробка сторінок. У проєкті передбачено паузи між частиною запитів, що дозволяє зменшити надмірне навантаження на зовнішнє джерело даних. Такий підхід є правильним, оскільки автоматизований збір не повинен виконуватися агресивно або створювати зайві запити за короткий проміжок часу.

Модуль MoviesUrlScraper.py відповідає за пошук посилань на окремі сторінки контенту. Він аналізує сторінки каталогу та знаходить елементи, які містять URL-адреси фільмів або серіалів. Після цього список посилань використовується для подальшого парсингу. Завдяки такому підходу система може обробляти не одну сторінку вручну, а цілу групу сторінок, які належать до певного типу контенту.

Модуль MovieScraper.py реалізує основну логіку виділення даних із конкретної сторінки. Він аналізує HTML-структуру, знаходить назву, постер, сцени, рейтинг, рік випуску, країни, жанри, тривалість, опис, вікову сертифікацію

та playlist. Саме результат роботи цього модуля є основою для майбутнього запису в таблицю movies.

Після отримання даних вони передаються до модуля роботи з базою. У db/base.py реалізовано логіку підключення до PostgreSQL, ініціалізації бази та створення або оновлення запису. Якщо запис із відповідним movie_id уже існує, дані оновлюються. Якщо його немає, створюється новий запис. Це забезпечує контроль за дублюванням і підтримує каталог у більш впорядкованому стані.

Окремий модуль ai.py використовується для отримання embedding. Векторне представлення формується на основі текстових даних і зберігається разом з іншою інформацією про фільм. Надалі це поле використовується серверною частиною для пошуку за змістовою близькістю. Таким чином, модуль автоматизованого збору контенту готує не лише основні дані для відображення, а й інформацію для розширеного пошукового механізму.

Процес автоматизованого збору має бути пов'язаний із перевіркою якості даних. Оскільки джерела можуть мати різну структуру, частина полів може бути відсутня або мати неправильний формат. Тому парсер повинен коректно обробляти ситуації, коли певне значення не знайдене. Наприклад, якщо відсутній рейтинг або країна виробництва, система не повинна повністю зупиняти обробку. У таких випадках можна використовувати порожнє значення або стандартне значення, яке надалі не заважатиме роботі застосунку.

Також важливо, щоб автоматизований збір не порушував логіку роботи інших частин системи. Android-застосунок не повинен залежати від того, у який саме момент запускається парсер. Якщо каталог уже містить записи, користувач може працювати з ними через застосунок. Якщо парсер додає новий контент, він стає доступним після запису до бази та наступного запиту до API. Такий підхід робить процес оновлення каталогу прозорим для користувача.

З погляду безпеки та відповідального використання важливо зазначити, що автоматизований збір контенту має виконуватися з урахуванням правових обмежень і умов використання джерел. У межах дипломної роботи цей модуль розглядається як технічний засіб автоматизації збору метаданих для каталогу. Для

реального використання онлайн-кінотеатру потрібно окремо перевіряти права на використання відеоматеріалів, постерів, описів та інших пов'язаних даних.

Реалізація модуля автоматизованого збору контенту має практичне значення для всієї системи KinoFox. Вона дозволяє зменшити обсяг ручної роботи, підтримувати єдину структуру відеокаталогу, уникати дублювання записів і готувати дані для пошуку. Крім того, модуль демонструє, що система складається не лише з мобільного інтерфейсу, а має повний цикл роботи з контентом: від отримання даних до їх збереження й подальшого відображення користувачу.

Отже, модуль автоматизованого збору контенту в системі KinoFox спроєктований як окремий Python-компонент, який взаємодіє з джерелами даних і базою PostgreSQL. Він отримує посилання на сторінки контенту, виділяє необхідну інформацію, очищає її, формує структурований запис, створює векторне представлення та зберігає результат у базі даних. Завдяки цьому Android-застосунок отримує через API вже підготовлений каталог, а сама система відповідає темі кваліфікаційної роботи - розробці Android-застосунку онлайн-кінотеатру з автоматизованим збором контенту.

4 ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Організація тестування Android-застосунку KinoFox

Після проєктування та реалізації основних компонентів системи KinoFox необхідно перевірити, чи правильно працює застосунок і чи відповідає він визначеним вимогам. Тестування є важливим етапом розробки програмного забезпечення, оскільки дозволяє виявити помилки в роботі інтерфейсу, серверної частини, бази даних, авторизації, пошуку та модуля автоматизованого збору контенту. Для Android-застосунку онлайн-кінотеатру тестування має особливе значення, оскільки система складається з кількох взаємопов'язаних частин, і помилка в одному компоненті може вплинути на роботу всього застосунку.

Тестування KinoFox доцільно розглядати не лише як перевірку окремих кнопок або сторінок, а як перевірку повного сценарію роботи користувача. Користувач відкриває застосунок, переходить між розділами, переглядає картки фільмів, відкриває сторінку детального перегляду, авторизується, працює з профілем, історією та обраним. При цьому кожна дія пов'язана з роботою клієнтської частини, серверного API та бази даних. Тому організація тестування повинна охоплювати як інтерфейс Android-застосунку, так і взаємодію між компонентами системи.

Основною метою тестування є перевірка працездатності Android-застосунку KinoFox та підтвердження того, що реалізовані функції відповідають вимогам, сформульованим у першому розділі. Під час тестування необхідно переконатися, що користувач може переглядати каталог, відкривати сторінки фільмів, отримувати дані із серверу, проходити авторизацію, зберігати сесію та працювати з основними розділами застосунку. Також потрібно перевірити, чи коректно серверна частина обробляє запити, повертає очікувані відповіді та взаємодіє з базою даних.

Об'єктом тестування є програмна система KinoFox, яка складається з Android-застосунку, REST API, PostgreSQL-бази даних і Python-модуля

автоматизованого збору контенту. Предметом тестування є коректність роботи функцій, які забезпечують взаємодію користувача з онлайн-кінотеатром, а також правильність обміну даними між клієнтською та серверною частинами.

Під час організації тестування було визначено кілька основних напрямів перевірки. Перший напрям стосується клієнтської частини, тобто Android-застосунку, його екранів, навігації, відображення даних і реакції на дії користувача. Другий напрям охоплює серверну частину та API: перевірку маршрутів, відповідей серверу, авторизації, отримання списку фільмів і детальної інформації. Третій напрям пов'язаний із базою даних: перевіряється наявність потрібних таблиць, коректність збереження записів і зв'язків між ними. Четвертий напрям стосується автоматизованого збору контенту, тобто перевірки отримання, структурування та запису даних до бази.

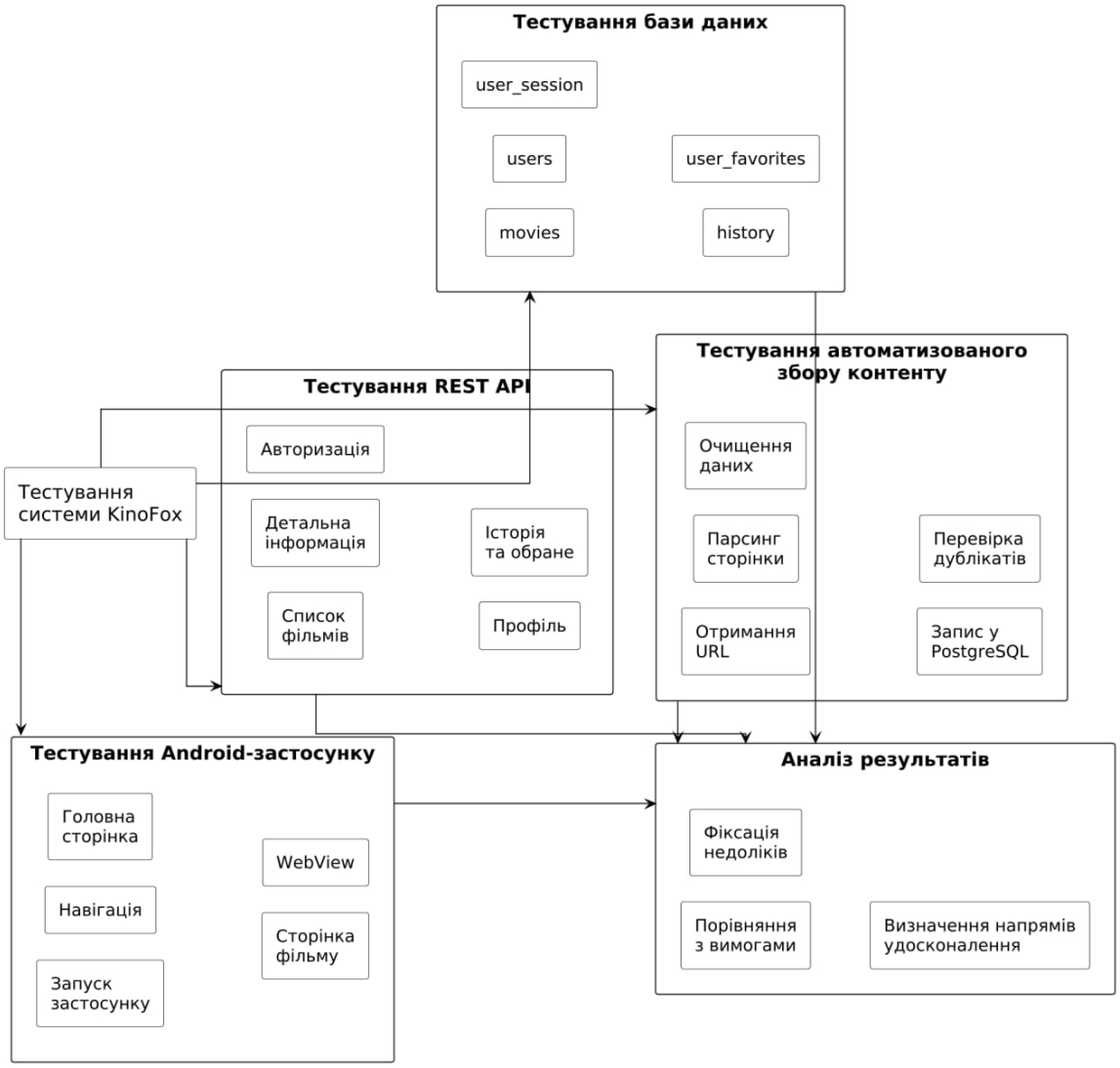


Рис. 4.1 Схема організації тестування системи KinoFox

Для тестування Android-застосунку важливо перевірити роботу основних екранів. До них належать головна сторінка, каталог, сторінка детального перегляду фільму, сторінки входу та реєстрації, акаунт користувача, історія та обране. Кожен екран має відкриватися без помилок, коректно відображати дані й не порушувати навігацію застосунку. Особливу увагу потрібно приділити головній сторінці та сторінці фільму, оскільки вони є основними для онлайн-кінотеатру.

Під час перевірки головної сторінки необхідно переконатися, що застосунок отримує дані із серверу та відображає їх у вигляді карток. Картка контенту має містити постер, назву, рейтинг, жанри або інші короткі характеристики. Якщо дані не завантажилися, застосунок не повинен завершувати роботу з помилкою. У такому випадку має відображатися порожній стан, повідомлення про помилку або інший зрозумілий для користувача результат. Це важливо, оскільки мобільний застосунок залежить від якості інтернет-з'єднання та доступності серверної частини.

Сторінка детального перегляду перевіряється окремо, оскільки вона використовує інший сценарій отримання даних. Після натискання на картку застосунок передає серверу ідентифікатор фільму, отримує повну інформацію та відображає її на екрані. Під час тестування потрібно перевірити, чи правильно передається movieId, чи повертає сервер відповідний запис, чи коректно відображаються назва, постер, рейтинг, рік, тривалість, жанри, країна, опис і playlist-посилання. Також потрібно звернути увагу на роботу вбудованого WebView, якщо сторінка містить джерело відтворення.

Окремий блок тестування пов'язаний з авторизацією. Для онлайн-кінотеатру акаунт користувача потрібен для персоналізованих функцій, тому необхідно перевірити сценарії реєстрації, входу, збереження сесії, отримання інформації профілю та виходу із системи. Під час тестування потрібно врахувати як правильні, так і помилкові дії користувача. Наприклад, потрібно перевірити, що система не дозволяє увійти з неправильним паролем, коректно реагує на відсутність токена та не показує дані профілю без авторизації.

Для серверної частини тестування передбачає перевірку API-маршрутів. Основними є маршрути отримання списку фільмів, отримання інформації про конкретний фільм, реєстрації, авторизації, отримання даних користувача, роботи з історією та обраним. Для кожного маршруту потрібно перевірити, чи приймає він очікувані параметри, чи повертає коректну відповідь і чи правильно обробляє помилкові ситуації. Наприклад, якщо фільм з указаним ідентифікатором не знайдено, сервер не повинен повертати некоректну або порожню відповідь без пояснення.

Тестування бази даних має підтвердити, що таблиці створені правильно, а зв'язки між ними працюють очікувано. Особливо важливо перевірити таблиці movies, users, user_session, user_favorites і history. У таблиці movies мають зберігатися дані про контент, у таблиці users - облікові записи, у user_session - активні сесії, а таблиці user_favorites і history повинні пов'язувати користувачів із фільмами. Якщо ці зв'язки працюють некоректно, персональні функції застосунку не зможуть стабільно працювати.

Автоматизований збір контенту також потребує перевірки. У цьому випадку тестування полягає в тому, щоб переконатися, що Python-модуль отримує сторінки з контентом, знаходить потрібні поля, формує структурований об'єкт і записує його до PostgreSQL. Додатково потрібно перевірити, чи не створюються дублікати під час повторної обробки того самого фільму. Якщо запис уже існує, система має оновити його, а не додавати ще один такий самий запис.

Під час тестування слід використовувати як позитивні, так і негативні сценарії. Позитивні сценарії перевіряють правильну роботу системи за нормальних умов. Наприклад, користувач вводить правильний email і пароль, відкриває наявний фільм або переглядає каталог за стабільного з'єднання. Негативні сценарії перевіряють поведінку системи у випадку помилок: неправильні дані для входу, відсутність фільму, порожній каталог, недійсний токен або помилка серверу. Такі перевірки потрібні для того, щоб застосунок був стійким до неочікуваних ситуацій.

Важливо також враховувати особливості мобільного середовища. Android-застосунок може запускатися на різних пристроях, які відрізняються розміром

екрана, продуктивністю та версією операційної системи. Тому під час тестування потрібно звертати увагу не лише на правильність даних, а й на відображення інтерфейсу. Елементи не повинні виходити за межі екрана, текст має залишатися читабельним, а кнопки та навігаційні елементи - доступними для натискання.

Окремо перевіряється робота із зовнішніми залежностями. Система KinoFox використовує API, базу даних, WebView і модуль збору контенту. Якщо один із цих компонентів тимчасово недоступний, застосунок має працювати передбачувано. Наприклад, за відсутності відповіді серверу користувач не повинен бачити некоректний екран. Якщо WebView не може завантажити сторінку відтворення, застосунок має обробити цю ситуацію без аварійного завершення.

У межах тестування також потрібно перевірити відповідність реалізованого функціоналу вимогам, сформульованим у підрозділі 1.4. Це дозволяє показати, що тестування не є випадковим, а прямо пов'язане з попередньо визначеними функціональними та нефункціональними вимогами. Наприклад, якщо в вимогах зазначено авторизацію, то в тестуванні мають бути сценарії входу та перевірки сесії. Якщо зазначено автоматизований збір контенту, то має бути перевірено додавання або оновлення записів у базі даних.

Отже, організація тестування системи KinoFox передбачає перевірку всіх основних компонентів: Android-застосунку, серверної частини, REST API, бази даних і модуля автоматизованого збору контенту. Такий підхід дозволяє оцінити не лише роботу окремих екранів, а й повний процес взаємодії між компонентами. У наступних підрозділах доцільно детальніше розглянути тестування основних функціональних можливостей системи та проаналізувати отримані результати.

4.2 Тестування основних функціональних можливостей системи

Тестування основних функціональних можливостей системи KinoFox виконувалося з урахуванням структури застосунку та вимог, визначених у попередніх розділах. Оскільки система складається з Android-застосунку, серверної частини, бази даних і модуля автоматизованого збору контенту, перевірка

проводилася не тільки на рівні окремих екранів, а й на рівні взаємодії між компонентами. Основна увага приділялася тим функціям, які безпосередньо впливають на роботу користувача з онлайн-кінотеатром: відкриттю застосунку, навігації, відображенню каталогу, перегляду сторінки фільму, авторизації, роботі з профілем, API-запитам і наповненню бази даних.

Першим етапом було перевірено запуск Android-застосунку та відкриття головного екрана. Після запуску застосунок повинен ініціалізувати основну структуру інтерфейсу, завантажити тему, підключити потрібні сторінки та відобразити нижню панель навігації. Під час перевірки зверталася увага на те, чи коректно відкривається головний екран, чи не виникають помилки під час завантаження інтерфейсу, чи правильно відображаються основні розділи: «Головна», «Каталог», «Улюблені», «Історія» та «Акаунт». Наявність цих розділів підтверджує, що базова навігаційна структура клієнтської частини працює відповідно до проєктного рішення.

Окремо перевірялася навігація між сторінками. Для цього послідовно виконувалися переходи між усіма пунктами нижнього меню. Очікуваним результатом було відкриття відповідної сторінки без аварійного завершення застосунку або втрати стану інтерфейсу. У межах цього тесту також перевірялося, чи активний пункт меню змінюється відповідно до вибраного розділу. Така перевірка є важливою, оскільки навігація є основою взаємодії користувача із застосунком.

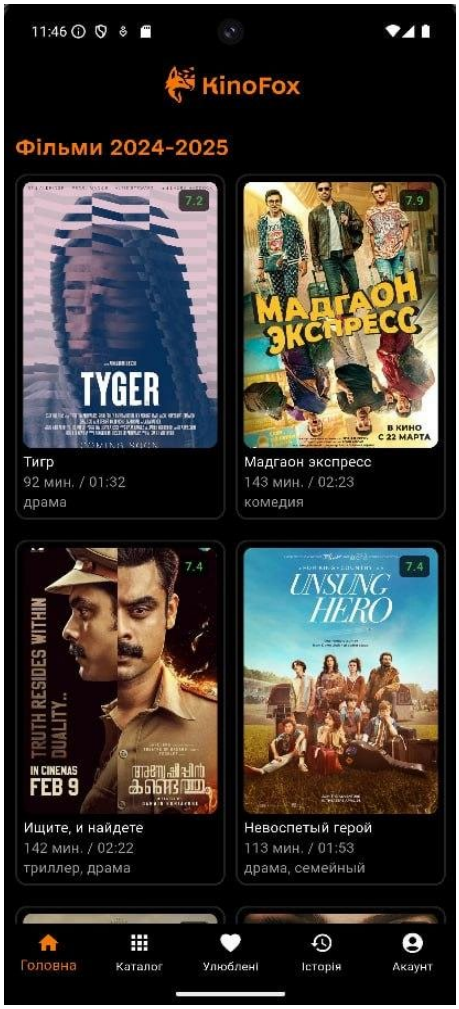


Рис. 4.2 Приклад перевірки головного екрана та навігації застосунку KinoFox

Наступним етапом було перевірено відображення контенту на головній сторінці. Застосунок KinoFox отримує дані про фільми, серіали та мультфільми через API-запити до серверної частини. Під час тестування перевірялося, чи виконується запит до серверу, чи надходить відповідь у правильному форматі та чи відображаються отримані дані у вигляді карток. Для кожної картки перевірялися основні елементи: постер, назва, рейтинг, тривалість і жанри. Якщо хоча б одне з цих полів не відображається, користувач отримує неповну інформацію, тому перевірка карток є важливою частиною тестування.

Під час перевірки головної сторінки також враховувалися ситуації, коли дані можуть завантажуватися із затримкою. У таких випадках інтерфейс не повинен зависати або аварійно завершувати роботу. Користувач має бачити коректний стан

завантаження або порожній стан, якщо сервер не повернув дані. Це особливо важливо для мобільного застосунку, оскільки якість інтернет-з'єднання на Android-пристроях може змінюватися.

Далі було протестовано відкриття сторінки детального перегляду фільму. Для цього на головній сторінці обиралася картка контенту, після чого застосунок мав передати серверу ідентифікатор вибраного фільму. Очікуваним результатом було відкриття сторінки з розширеною інформацією: назвою, постером, рейтингом, роком випуску, тривалістю, жанрами, країною, описом і доступними посиланнями для перегляду. Під час тестування перевірялося, чи правильно передається movieId, чи повертає сервер потрібний запис і чи коректно клієнтська частина відображає отриману інформацію.

Окремо перевірялася робота WebView-компонента на сторінці фільму. Оскільки в застосунку використовується вбудоване відображення вебплеєра або iframe, потрібно було переконатися, що сторінка відтворення відкривається в межах застосунку, а не порушує його роботу. У випадку недоступного посилання застосунок не повинен завершуватися з помилкою. Такий сценарій є важливим, оскільки джерела відтворення можуть бути тимчасово недоступними або змінювати свою структуру.

Важливим блоком тестування була перевірка авторизації. У системі KinoFox користувач має можливість увійти до акаунта за email і паролем. Під час тестування перевірялися два основні сценарії: успішний вхід із правильними даними та невдалий вхід із помилковими даними. У першому випадку сервер повинен повернути токен сесії, який зберігається в локальному сховищі застосунку. У другому випадку система повинна повідомити про помилку та не створювати сесію.

Після успішної авторизації перевірялося локальне збереження токена. Для цього після входу застосунок повторно відкривався, а сторінка акаунта мала отримати інформацію про поточного користувача через серверний маршрут перевірки сесії. Очікуваним результатом було відображення даних користувача без повторного введення email і пароля. Якщо токен відсутній або недійсний,

користувач повинен бути перенаправлений до авторизації або отримати повідомлення про необхідність входу.

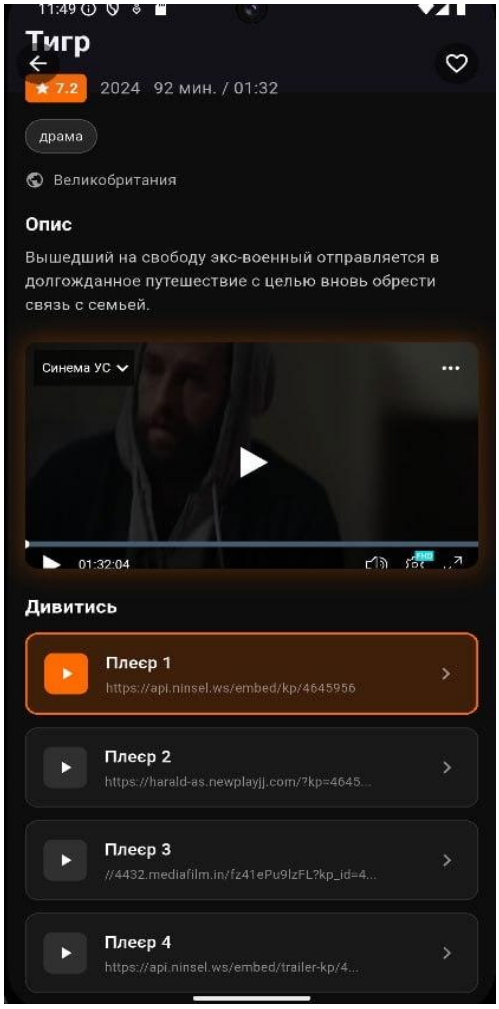


Рис. 4.3 Приклад перевірки сторінки детального перегляду фільму

Сторінка реєстрації перевірялася через введення даних нового користувача. Під час тестування було важливо переконатися, що сервер не дозволяє створити два акаунти з однаковим email. Також перевірялося, чи пароль не зберігається у відкритому вигляді, а обробляється на серверному рівні з використанням хешування. Така перевірка підтверджує базову безпеку роботи з обліковими даними користувача.

Окрему увагу було приділено тестуванню API-маршрутів. Перевірялися маршрути отримання списку фільмів, отримання детальної інформації про фільм, реєстрації, авторизації, отримання інформації про користувача, роботи з обраним та історією. Для кожного маршруту перевірявся тип запиту, наявність потрібних

параметрів і структура відповіді. Якщо маршрут повертає список фільмів, відповідь має містити масив об’єктів із потрібними полями. Якщо маршрут відповідає за авторизацію, він має повертати токен або повідомлення про помилку.

Результати перевірки основних функціональних можливостей наведено в таблиці 4.2.

Таблиця 4.2

Результати тестування основних функцій системи KinoFox

№	Функція	Дія для перевірки	Очікуваний результат	Результат
1	Запуск застосунку	Відкрити Android-застосунок	Відображається головний екран і нижня навігація	Виконано
2	Навігація	Перейти між основними розділами	Відкриваються відповідні сторінки	Виконано
3	Головна сторінка	Завантажити добірки контенту	Відображаються картки фільмів, серіалів і мультфільмів	Виконано
4	Картка контенту	Перевірити постер, назву, рейтинг і жанри	Дані відображаються на картці	Виконано
5	Детальна сторінка	Натиснути на картку фільму	Відкривається сторінка з повною інформацією	Виконано
6	WebView	Відкрити доступне посилання для перегляду	Відеоджерело відкривається в межах застосунку	Виконано
7	Реєстрація	Створити нового користувача	Користувач створюється в базі даних	Виконано
8	Авторизація	Увести правильний email і пароль	Сервер повертає токен сесії	Виконано

Продовження таблиці 4.2

Результати тестування основних функцій системи KinoFox

№	Функція	Дія для перевірки	Очікуваний результат	Результат
9	Помилкова авторизація	Увести неправильний пароль	Система не створює сесію	Виконано
10	Профіль	Відкрити сторінку акаунта після входу	Відображаються дані користувача та сесії	Виконано
11	Список фільмів API	Надіслати запит до /api/movie/list	Повертається список контенту	Виконано
12	Інформація про фільм API	Надіслати запит до /api/movie/info/:id	Повертається детальна інформація про фільм	Виконано
13	Обране API	Додати або видалити фільм з обраного	Сервер обробляє запит без помилки	Виконано
14	Історія API	Додати запис до історії	Дані зберігаються у відповідній таблиці	Виконано
15	Автоматизований збір	Запустити Python-модуль збору контенту	Дані додаються або оновлюються в PostgreSQL	Виконано

Під час тестування API було важливо перевірити не тільки позитивні сценарії, а й помилкові ситуації. Наприклад, якщо клієнтська частина передає неіснуючий ідентифікатор фільму, сервер повинен повернути зрозумілу відповідь, а не викликати аварійне завершення. Якщо користувач намагається отримати дані профілю без токена, сервер має відхилити запит або повідомити про відсутність авторизації. Такий підхід дозволяє перевірити стійкість системи до некоректних дій користувача або помилок у запитах.

Для перевірки бази даних аналізувалося збереження записів у таблицях movies, users, user_session, user_favorites і history. Після виконання відповідних дій у застосунку або через API перевірялося, чи з’являються потрібні записи в таблицях і чи зберігаються зв’язки між користувачем і контентом. Наприклад, після авторизації має створюватися запис у таблиці сесій, а після додавання фільму до обраного - запис у таблиці зв’язку між користувачем і фільмом.

Тестування автоматизованого збору контенту проводилося через запуск Python-модуля. Під час перевірки контролювалося, чи отримуються URL-адреси сторінок, чи обробляється кожна сторінка, чи виділяються потрібні дані та чи записуються вони до PostgreSQL. Особливо важливо було перевірити поведінку під час повторної обробки вже наявного фільму. Очікуваним результатом було оновлення запису без створення дубліката. Це підтверджує правильність використання унікального ідентифікатора контенту.

Окремо перевірялася відповідність отриманих даних структурі, яка використовується клієнтською частиною. Якщо парсер записує до бази даних поле з постером, сервер повинен повернути його в API-відповіді, а Flutter-застосунок - коректно відобразити на картці. Якщо в записі є список жанрів, він має бути переданий у форматі, який може обробити модель Film. Таким чином, тестування автоматизованого збору не обмежується лише перевіркою парсера, а охоплює весь шлях даних до мобільного інтерфейсу.

Слід зазначити, що сторінки «Улюблені» та «Історія» у клієнтській частині передбачені як окремі розділи навігації, а відповідні API-маршрути на серверному рівні реалізовані. Під час тестування серверної частини перевіряється можливість додавання та видалення записів, однак клієнтська інтеграція цих функцій може потребувати подальшого розширення. Це не порушує загальну архітектуру системи, але визначає напрям подальшого вдосконалення застосунку.

Крім функціонального тестування, було перевірено коректність відображення інтерфейсу. Зверталася увага на те, чи не виходять елементи за межі екрана, чи зберігається читабельність тексту, чи достатньо зручно розташовані кнопки й навігація. Для онлайн-кінотеатру це важливо, оскільки користувач багато

взаємодіє саме з візуальним каталогом. Якщо картки, постери або текстові блоки відображаються некоректно, навіть правильно працююча серверна частина не забезпечить комфортного користування застосунком.

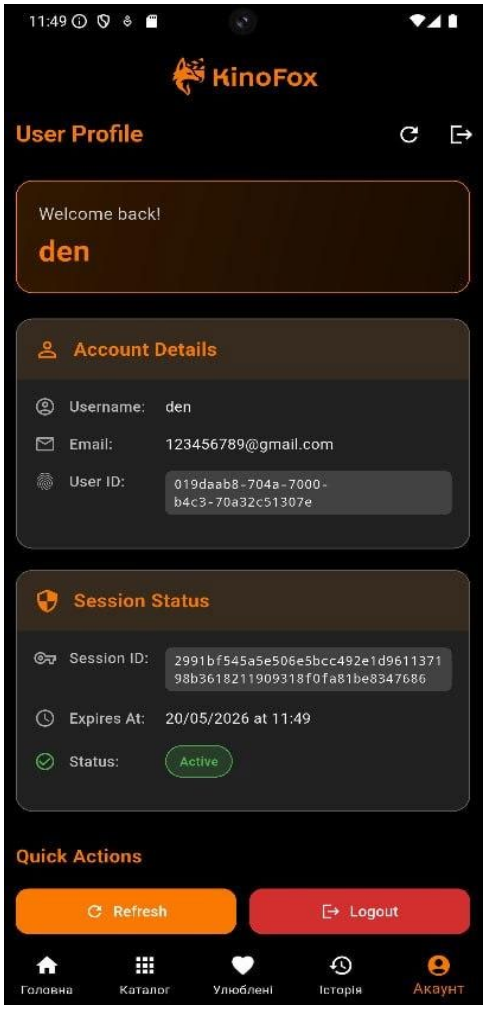


Рис. 4.4 Приклад перевірки авторизації та сторінки акаунта користувача

Для узагальнення перевірки API можна подати окрему таблицю тестових запитів. Вона допомагає показати, які саме маршрути використовувалися та який результат очікувався.

Під час перевірки API також було враховано, що маршрут семантичного пошуку в поточній реалізації має назву /api/movie/seach. Він може використовуватися для перевірки пошукової логіки, однак у майбутньому доцільно перейменувати його на /api/movie/search. Це покращить зрозумілість структури API та відповідність назви маршруту його призначенню.

Таблиця 4.3

Перевірка основних API-маршрутів системи KinoFox

№	API-маршрут	Метод	Мета перевірки	Очікуваний результат
1	/api/movie/list	GET	Отримання списку контенту	Повертається масив фільмів
2	/api/movie/info/:id	GET	Отримання даних конкретного фільму	Повертається об’єкт із детальною інформацією
3	/api/auth/registration	POST	Реєстрація користувача	Створюється новий користувач
4	/api/auth/login	POST	Авторизація користувача	Повертається токен сесії
5	/api/auth/info	GET	Отримання інформації про користувача	Повертаються дані акаунта та сесії
6	/api/favourite	POST/DELETE	Робота зі списком обраного	Запис додається або видаляється
7	/api/history	POST/DELETE	Робота з історією переглядів	Запис додається або видаляється
8	/api/history/:id	GET	Отримання історії користувача	Повертається список записів історії
9	/api/movie/seach	GET/POST	Перевірка семантичного пошуку	Повертаються релевантні результати

Загалом тестування основних функціональних можливостей показало, що структура системи KinoFox дозволяє виконувати основні сценарії роботи онлайн-кінотеатру. Android-застосунок забезпечує навігацію, відображення контенту та роботу з акаунтом. Серверна частина обробляє запити, повертає дані у форматі JSON, підтримує авторизацію, обране, історію та пошук. База даних зберігає основні сутності системи, а модуль автоматизованого збору контенту забезпечує наповнення каталогу. Виявлені напрями для подальшого вдосконалення пов’язані

переважно з розширенням клієнтської інтеграції сторінок історії та обраного, а також уточненням назви маршруту пошуку.

4.3 Аналіз результатів тестування та методика користування застосунком

Після проведення тестування основних функціональних можливостей системи KinoFox було виконано узагальнення отриманих результатів. Метою цього етапу є не лише фіксація того, які функції працюють, а й визначення відповідності реалізованої системи вимогам, сформульованим у першому розділі. Для Android-застосунку онлайн-кінотеатру важливо, щоб окремі компоненти не просто існували в коді, а працювали як єдина система: мобільний клієнт має отримувати дані із серверу, сервер має коректно взаємодіяти з базою даних, а модуль автоматизованого збору контенту має забезпечувати наповнення каталогу.

За результатами перевірки було встановлено, що система KinoFox реалізує основні функції, необхідні для роботи онлайн-кінотеатру. Android-застосунок запускається, відображає головну сторінку, містить нижню навігацію між основними розділами, отримує дані з API та показує контент у вигляді карток. Користувач може перейти до сторінки детального перегляду фільму, де відображається розширена інформація про вибраний контент. Також реалізовано сторінки авторизації, реєстрації та акаунта користувача, що дає змогу працювати із сесіями.

Окремо було підтверджено працездатність серверної частини. API забезпечує отримання списку контенту, отримання детальної інформації про фільм, реєстрацію, авторизацію, перевірку сесії, а також роботу з історією та обраним на серверному рівні. Це свідчить про те, що серверна частина виконує роль проміжного рівня між клієнтом і базою даних, не надаючи мобільному застосунку прямого доступу до PostgreSQL. Такий підхід відповідає обраній клієнт-серверній архітектурі.

База даних системи містить основні сутності, необхідні для роботи застосунку: фільми, користувачів, сесії, обране та історію. Перевірка показала, що дані про відеоконтент можуть зберігатися у структурованому вигляді та передаватися через серверну частину до Android-застосунку. Наявність окремих таблиць для сесій, історії та обраного дозволяє реалізовувати персоналізовані можливості, які є важливими для онлайн-кінотеатру.

У таблиці 4.4 наведено узагальнену оцінку відповідності реалізованої системи основним вимогам.

Таблиця 4.4

Відповідність реалізованої системи KinoFox основним вимогам

Вимога	**Реалізація в системі KinoFox**	**Стан виконання**
Перегляд головної сторінки	Реалізовано головний екран із блоками контенту	Виконано
Перегляд каталогу	Передбачено розділ каталогу та відображення карток контенту	Виконано частково
Детальна сторінка фільму	Реалізовано отримання та показ повної інформації за movieId	Виконано
Авторизація користувача	Реалізовано вхід за email і паролем через API	Виконано
Реєстрація користувача	Передбачено створення нового акаунта	Виконано
Збереження сесії	Токен сесії зберігається локально через shared_preferences	Виконано
Профіль користувача	Реалізовано отримання інформації про акаунт і сесію	Виконано
Обране	Серверні маршрути передбачені, клієнтська інтеграція потребує розширення	Виконано частково
Історія переглядів	Серверні маршрути передбачені, клієнтська інтеграція потребує розширення	Виконано частково
Автоматизований збір контенту	Реалізовано Python-модуль для збору та запису даних	Виконано
Пошук	Передбачено серверну логіку семантичного пошуку	Виконано частково
Збереження даних	Використовується PostgreSQL із відповідними таблицями	Виконано

Важливим результатом є також перевірка модуля автоматизованого збору контенту. Його використання дозволяє наповнювати базу даних інформацією про фільми та серіали без повністю ручного введення кожного запису. Парсер отримує дані, виділяє основні поля, приводить їх до потрібної структури та записує до PostgreSQL. Таким чином, у системі реалізовано не тільки мобільний інтерфейс, а й механізм підготовки відеокаталогу.

Аналіз результатів показує, що більшість базових функцій системи реалізовано. Найбільш повно працюють ті частини, які пов’язані з відображенням відеокаталогу, отриманням даних із серверу, сторінкою детального перегляду та авторизацією користувача. Саме ці можливості формують основний сценарій використання онлайн-кінотеатру: від відкриття застосунку до вибору конкретного фільму.

Разом із цим під час аналізу було визначено кілька напрямів, які доцільно вдосконалити в подальшій роботі. По-перше, клієнтську частину для сторінок «Улюблені» та «Історія» можна розширити так, щоб вона повністю використовувала відповідні серверні маршрути. По-друге, маршрут семантичного пошуку в серверному коді має назву /api/movie/seach, тому в майбутньому його доцільно перейменувати на /api/movie/search. По-третє, для стабільної роботи Android-застосунку з мережею необхідно перевірити наявність потрібних дозволів у конфігурації Android-проєкту, зокрема дозволу на доступ до інтернету.

Такі зауваження не змінюють загальної працездатності системи, але показують, що проєкт має потенціал для подальшого розвитку. Для дипломної роботи це важливо, оскільки програмний продукт не обмежується демонстрацією окремих екранів, а має зрозумілу архітектуру, яку можна поступово вдосконалювати. Надалі до системи можна додати повноцінну адміністративну панель, розширені рекомендації, коментарі, оцінки користувачів, фільтри каталогу та кращу обробку помилок під час відтворення відео.

Крім аналізу результатів тестування, необхідно описати методику користування застосунком. Вона потрібна для того, щоб показати, як саме користувач може працювати з розробленим програмним продуктом. Методика

користування також підтверджує, що застосунок має логічну структуру й не потребує складного попереднього навчання.

Перед початком роботи користувач відкриває Android-застосунок KinoFox на мобільному пристрої. Після запуску відображається головна сторінка з основними добірками контенту. На нижній панелі навігації доступні головні розділи застосунку: «Головна», «Каталог», «Улюблені», «Історія» та «Акаунт». Такий спосіб організації інтерфейсу дає змогу швидко перейти до потрібної частини застосунку.

Для перегляду контенту користувач може залишитися на головній сторінці або перейти до каталогу. На екрані відображаються картки фільмів, серіалів або мультфільмів. Кожна картка містить коротку інформацію, яка допомагає зробити попередній вибір: постер, назву, рейтинг, тривалість і жанри. Якщо користувач хоче отримати більше інформації, він натискає на картку.

Після натискання на картку відкривається сторінка детального перегляду. На цій сторінці користувач бачить повну інформацію про вибраний контент: назву, постер, рейтинг, рік випуску, країну, жанри, тривалість і опис сюжету. Якщо для фільму доступне посилання на перегляд, воно відкривається через вбудований WebView-компонент. Це дозволяє залишатися в межах застосунку й не переходити до зовнішнього браузера.

Для використання персоналізованих функцій користувач переходить до розділу «Акаунт». Якщо він ще не авторизований, застосунок пропонує виконати вхід або реєстрацію. Під час входу потрібно ввести email і пароль. Після успішної авторизації сервер повертає токен сесії, який зберігається локально на пристрої. Надалі користувач може відкривати застосунок без повторного введення даних, поки сесія залишається дійсною.

Якщо користувач не має облікового запису, він може перейти до сторінки реєстрації. Після створення акаунта він отримує можливість авторизуватися та користуватися персональними функціями. Наявність акаунта потрібна для роботи зі збереженими даними, такими як обране або історія переглядів. У майбутньому

ці функції можуть бути використані також для рекомендацій і продовження перегляду.

Розділ «Улюблені» призначений для зберігання контенту, який користувач хоче переглянути пізніше. Розділ «Історія» використовується для повернення до фільмів або серіалів, з якими користувач уже взаємодіяв. У поточній структурі застосунку ці розділи передбачені як частина навігації, а їх повне функціональне наповнення може бути розширене через інтеграцію з відповідними API-маршрутами.

Загальний порядок користування застосунком KinoFox можна подати у вигляді послідовності дій:

1. запустити Android-застосунок KinoFox;
2. переглянути головну сторінку або перейти до каталогу;
3. обрати фільм, серіал або мультфільм із доступних карток;
4. відкрити сторінку детального перегляду;
5. ознайомитися з описом, рейтингом, жанрами та іншими характеристиками;
6. за потреби перейти до перегляду через вбудований WebView;
7. перейти до розділу «Акаунт»;
8. виконати реєстрацію або авторизацію;
9. користуватися персональними розділами «Улюблені» та «Історія»;
10. завершити роботу із застосунком або вийти з акаунта.

На рисунку 4.6 доцільно подати діаграму діяльності або схему користування Android-застосунком KinoFox, яка відображає основний шлях користувача від запуску застосунку до перегляду контенту та авторизації.

Методика користування показує, що застосунок має зрозумілий сценарій роботи. Користувач не повинен виконувати складні налаштування для перегляду каталогу. Основні можливості доступні через нижню панель навігації, а перехід до сторінки фільму виконується через натискання на картку. Такий підхід відповідає типовій логіці мобільних онлайн-кінотеатрів і робить застосунок зручним для користувача.

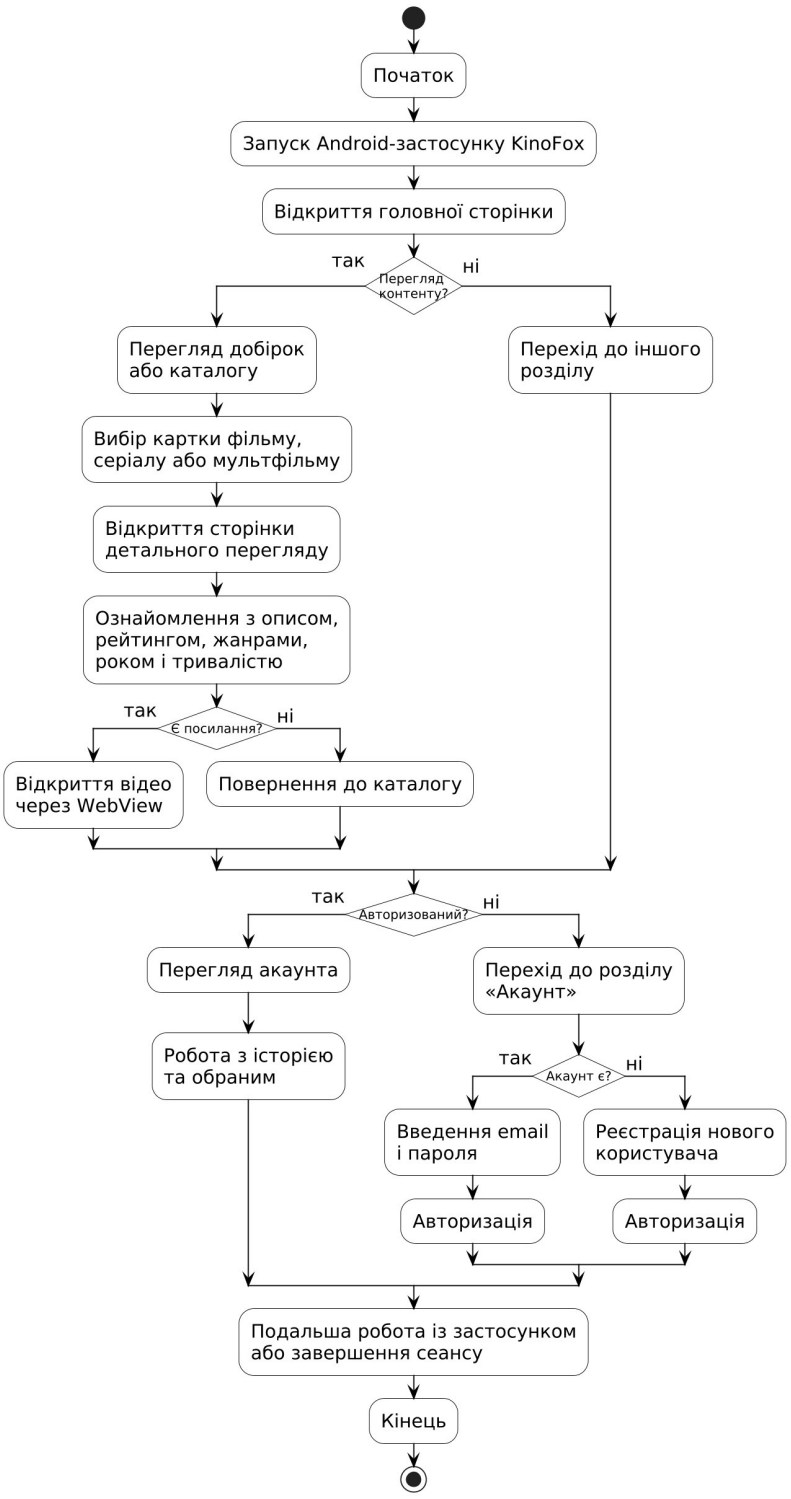


Рис. 4.5 Блок-схема користування Android-застосунком KinoFox

У таблиці 4.5 наведено короткий опис основних дій користувача в застосунку.

Таблиця 4.5

Методика користування Android-застосунком KinoFox

Дія користувача	Елемент застосунку	Результат
Запуск застосунку	Головний екран	Відкривається стартова сторінка з добірками контенту
Перехід між розділами	Нижня панель навігації	Відкривається вибраний розділ застосунку
Вибір фільму	Картка контенту	Відкривається сторінка детального перегляду
Перегляд інформації	Сторінка фільму	Користувач бачить опис, рейтинг, жанри та інші дані
Запуск перегляду	WebView-компонент	Відкривається доступне джерело відтворення
Авторизація	Сторінка входу	Користувач отримує активну сесію
Реєстрація	Сторінка реєстрації	Створюється новий обліковий запис
Перегляд профілю	Розділ «Акаунт»	Відображаються дані користувача та сесії
Робота з обраним	Розділ «Улюблені»	Користувач може працювати зі збереженим контентом після розширення інтеграції
Перегляд історії	Розділ «Історія»	Користувач може повернутися до раніше відкритого контенту після розширення інтеграції

Підсумовуючи результати тестування, можна зазначити, що розроблена система KinoFox відповідає основній меті кваліфікаційної роботи. У межах проєкту реалізовано Android-застосунок онлайн-кінотеатру, серверну частину з REST API, базу даних PostgreSQL і модуль автоматизованого збору контенту. Система дозволяє відображати відеокаталог, відкривати сторінки фільмів, працювати з авторизацією та готувати контент для подальшого використання в застосунку.

Результати тестування також показали, що система має логічну структуру для подальшого розвитку. Виявлені недоліки не є критичними для демонстрації

основної роботи проєкту, але можуть бути використані як напрями вдосконалення. До таких напрямів належать повна інтеграція клієнтських сторінок історії та обраного з API, виправлення назви маршруту пошуку, розширення обробки помилок, додавання адміністративної панелі та покращення механізмів фільтрації й рекомендацій.

Отже, тестування підтвердило працездатність основних компонентів системи KinoFox і показало, що розроблений Android-застосунок може використовуватися як програмна основа онлайн-кінотеатру з автоматизованим збором контенту. Методика користування демонструє, що застосунок має зрозумілий сценарій роботи для користувача: від відкриття каталогу до перегляду детальної інформації, авторизації та використання персональних розділів.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досягнуто поставленої мети — розроблено Android-застосунок онлайн-кінотеатру KinoFox з автоматизованим збором контенту. Система поєднує мобільний клієнт, серверну частину, базу даних PostgreSQL та Python-модуль для наповнення відеокаталогу.

Проаналізовано особливості Android-застосунків онлайн-кінотеатрів, потреби користувачів мобільних сервісів, проблему формування й оновлення відеокаталогу та наявні аналоги. На основі цього визначено функціональні й нефункціональні вимоги до застосунку KinoFox.

Обґрунтовано вибір клієнт-серверної архітектури та технологічного стеку. Для клієнтської частини використано Flutter і Dart, для серверної частини — Bun, Elysia та TypeScript, для збереження даних — PostgreSQL і pgvector, а для автоматизованого збору контенту — Python.

Спроєктовано загальну архітектуру системи, структуру бази даних, серверну частину та REST API. Реалізовано Android-застосунок із головною сторінкою, картками контенту, сторінкою фільму, авторизацією, реєстрацією й акаунтом користувача, а також серверні маршрути для роботи з каталогом, профілем, пошуком, історією та обраним.

Реалізовано модуль автоматизованого збору контенту, який отримує дані про відеоматеріали, структурує їх і записує до PostgreSQL. Проведене тестування підтвердило працездатність основних функцій системи та коректну взаємодію Android-застосунку, REST API, бази даних і модуля збору контенту. Розробка може бути використана як основа для цифрових медіасервісів, онлайн-кінотеатрів і платформ з автоматизованим формуванням мультимедійних каталогів.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Цуркан Д.О. Шахматов І.О. Визначення вимог до системи безпеки Android-застосунку онлайн-кінотеатру з захистом даних користувачів та контенту
Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.253 - 255.

2. Цуркан Д.О. Шахматов І.О. Стратегії оптимізації продуктивності та ресурсомісткості Android-застосунку онлайн-кінотеатру
Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2026, С. 414-416.

ПЕРЕЛІК ПОСИЛАНЬ

1. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York: McGraw-Hill Education, 2020. 705 p.

2. Sommerville I. Engineering Software Products: An Introduction to Modern Software Engineering. Global ed. Harlow: Pearson, 2020. 352 p.

3. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston: Addison-Wesley Professional, 2021. 464 p.

4. Geewax J. J. API Design Patterns. Shelter Island: Manning Publications, 2021. 480 p.

5. Mitchell R. Web Scraping with Python: Data Extraction from the Modern Web. 3rd ed. Sebastopol: O’Reilly Media, 2024. 300 p.

6. McKinney W. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter. 3rd ed. Sebastopol: O’Reilly Media, 2022. 579 p.

7. Beaulieu A. Learning SQL: Generate, Manipulate, and Retrieve Data. 3rd ed. Sebastopol: O’Reilly Media, 2020. 380 p.

8. Netflix - Apps on Google Play. Google Play. URL: https://play.google.com/store/apps/details?id=com.netflix.mediaclient (дата звернення: 16.05.2026).

9. How to download titles to watch offline. Netflix Help Center. URL: https://help.netflix.com/en/node/54816 (дата звернення: 16.05.2026).

10. MEGOGO: ТБ та Кіно - додаток Android. Google Play. URL: https://play.google.com/store/apps/details?id=com.megogo.application&hl=uk (дата звернення: 16.05.2026).

11. SWEET.TV: телебачення і онлайн-кінотеатр в HD якості. SWEET.TV. URL: https://sweet.tv/uk (дата звернення: 16.05.2026).

12. Windmill E. Flutter in Action. Shelter Island: Manning Publications, 2020. 368 p.

13. Welcome to Bun. Bun Documentation. URL: https://bun.com/docs (дата звернення: 16.05.2026).

14. ElysiaJS: Elysia - Ergonomic Framework for Humans. ElysiaJS. URL: https://elysiajs.com/ (дата звернення: 16.05.2026).

15. Fain Y., Moiseev A. TypeScript Quickly. Shelter Island: Manning Publications, 2020. 500 p.

16. PostgreSQL 17 Documentation. PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/17/ (дата звернення: 16.05.2026).

17. Hoffman A. Web Application Security: Exploitation and Countermeasures for Modern Web Applications. Sebastopol: O'Reilly Media, 2020. 330 p.

18. pgvector: Open-source vector similarity search for Postgres. GitHub. URL: https://github.com/pgvector/pgvector (дата звернення: 16.05.2026).

19. Vector embeddings. OpenAI API. URL: https://developers.openai.com/api/docs/guides/embeddings (дата звернення: 16.05.2026).

20. Hattingh C. Using Asyncio in Python: Understanding Python's Asynchronous Programming Features. Sebastopol: O'Reilly Media, 2020. 166 p.

21. Kleppmann M. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. Sebastopol: O'Reilly Media, 2017. 616 p.

22. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol: O'Reilly Media, 2021. 616 p.

23. Richardson C. Microservices Patterns: With Examples in Java. Shelter Island: Manning Publications, 2018. 520 p.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка Android-застосунку
онлайн-кінотеатру з автоматизованим збором контенту

Виконав студент 4 курсу
групи ПД-42
Цуркан Денис Олександрович
Керівник роботи
канд.техн.наук, доц. Довженко Тимур Павлович

Київ – 2026

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну область Android-застосунків онлайн-кінотеатрів.
2. Проаналізувати особливості функціонування Android-застосунків онлайн-кінотеатрів.
3. Визначення функціональниї і не функціональних вимог.
4. Спроєктувати загальну архітектуру системи, структуру бази даних, серверну частину та REST API KinoFox.
5. Реалізувати клієнтську частину Android-застосунку.
6. Реалізувати серверну частину для обробки запитів, авторизації та роботи з даними.
7. Реалізувати модуль автоматизованого збору контенту.
8. Провести тестування основних функціональних можливостей застосунку.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - спрощення доступу користувачу до медіаконтенту шляхом створення Android-застосунку онлайн-кінотеатру з механізмами автоматизованого збору та оновлення контенту, що забезпечує зручний перегляд фільмів і серіалів, персоналізовані рекомендації та актуальну медіатеку без необхідності ручного втручання адміністратора.

Об'єкт дослідження - процес автоматизованого збору та оновлення контенту в системах онлайн-кінотеатрі.

Предмет дослідження - програмне забезпечення для реалізації Android-застосунку онлайн-кінотеатру з клієнтською, серверною та сервісною підсистемами автоматизованого збору контенту.

3

АНАЛІЗ АНАЛОГІВ

	Netflix	Megogo	SWEET.TV	HDRezka	KinoFox
Android-застосунок	+	+	+	-	+
Безкоштовність	-	-	-	+	+
Пошук та фільтри	+	+	+	+	+
Наявність API	+	+	+	+	+
Автоматизований збір контенту	-	-	-	-	+
Семантичний пошук	-	-	-	-	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

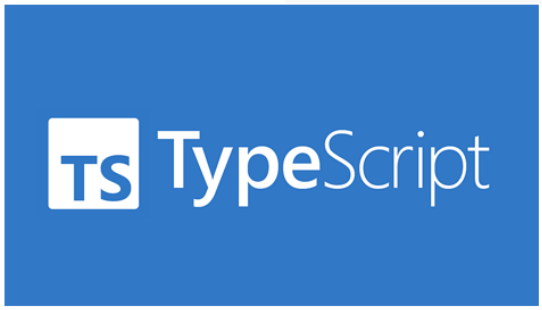
1. Реєстрація, вхід і перевірка активної сесії.
2. Відображення каталогу: фільми, серіали, мультфільми.
3. Перегляд картки контенту та запуск відтворення.
4. Робота з обраним та історією перегляду.
5. Автоматизоване оновлення метаданих контенту.

Нефункціональні вимоги:

1. Застосунок повинен працювати на пристроях з Android 8.0 і вище.
2. Інтерфейс повинен коректно відображатися на екранах з роздільною здатністю від 720×1280 px.
3. Обмін даними між клієнтом і сервером повинен здійснюватися через REST API по HTTPS.
4. База даних повинна бути реалізована на PostgreSQL 14+.
5. Система повинна підтримувати одночасну роботу не менше 100 користувачів.
6. Система повинна забезпечувати можливість перемикання мови інтерфейсу між українською, англійською та російською.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

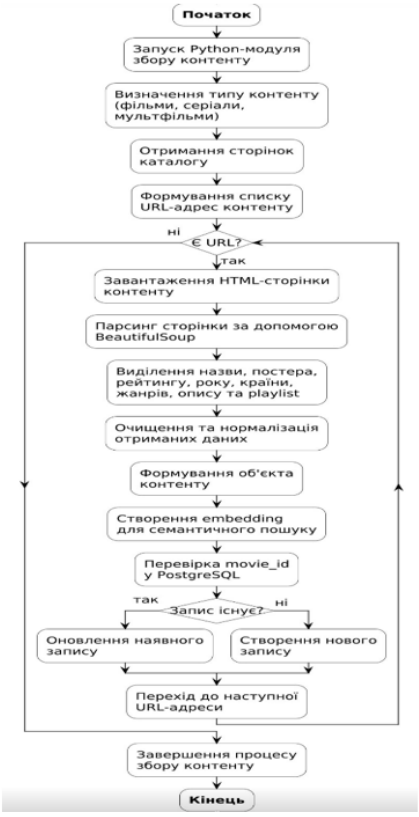


6

АРХІТЕКТУРУ ЗАСТОСУНКУ

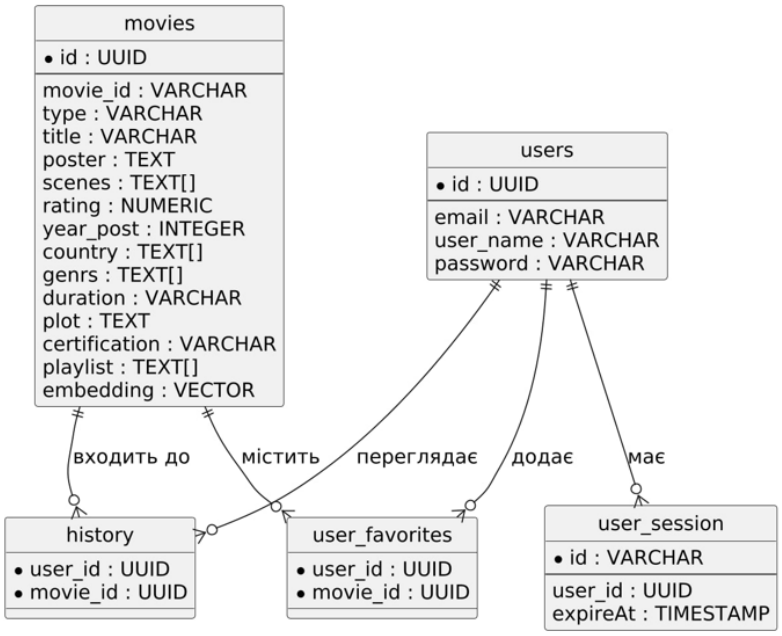
7

БЛОК-СХЕМА РОБОТИ МОДУЛЯ АВТОМАТИЗОВАНОГО ЗБОРУ КОНТЕНТУ



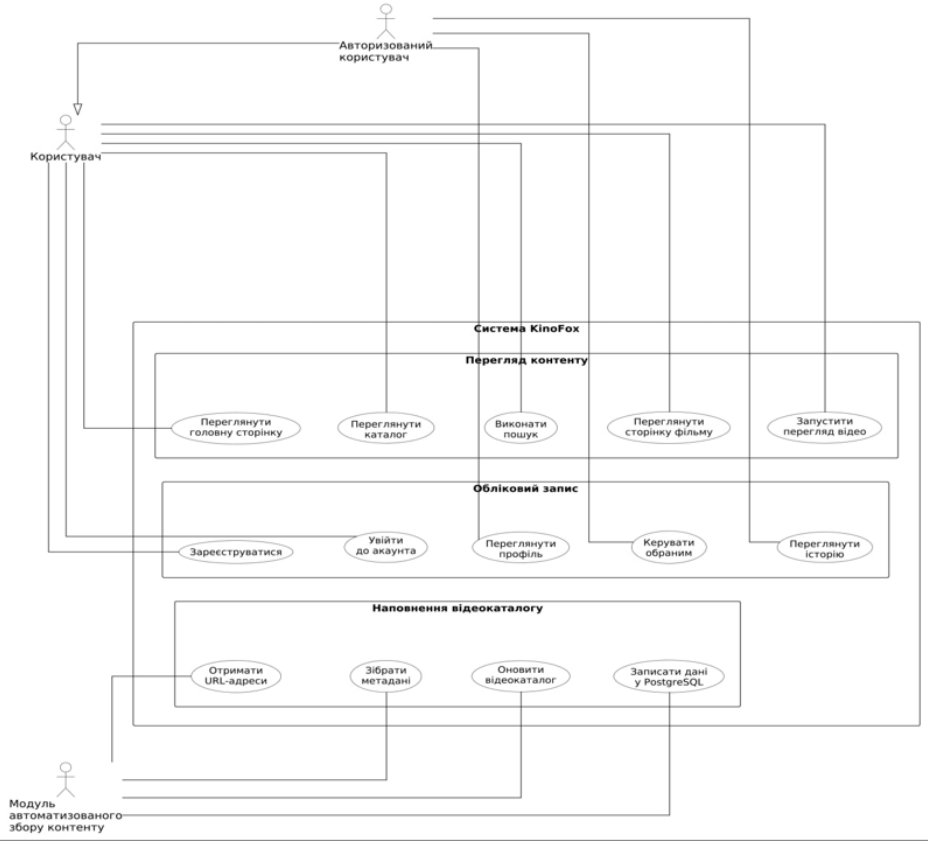
8

СХЕМА БАЗИ ДАНИХ



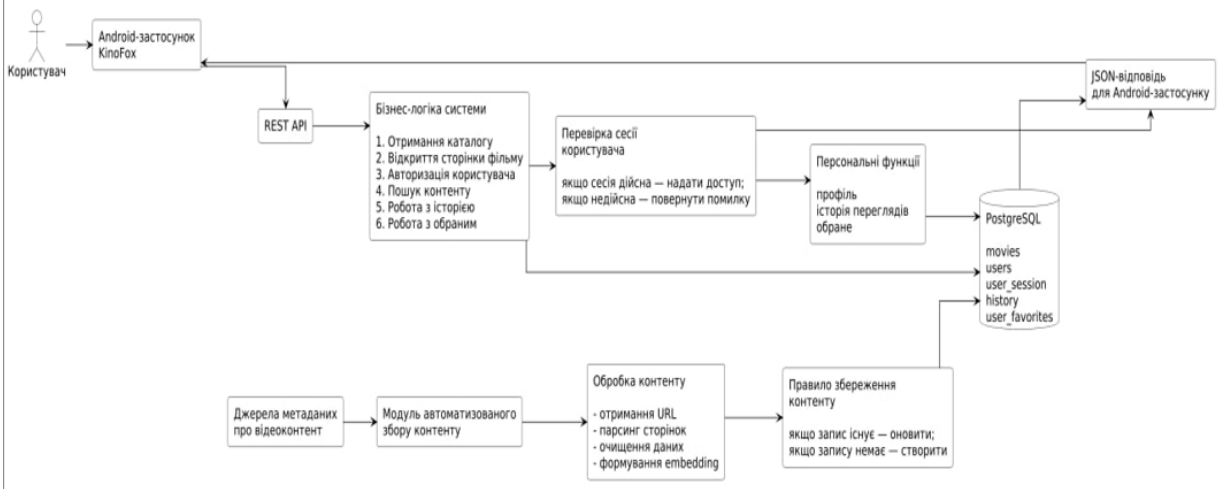
9

СХЕМА ПРИЦИНДЕНТІВ



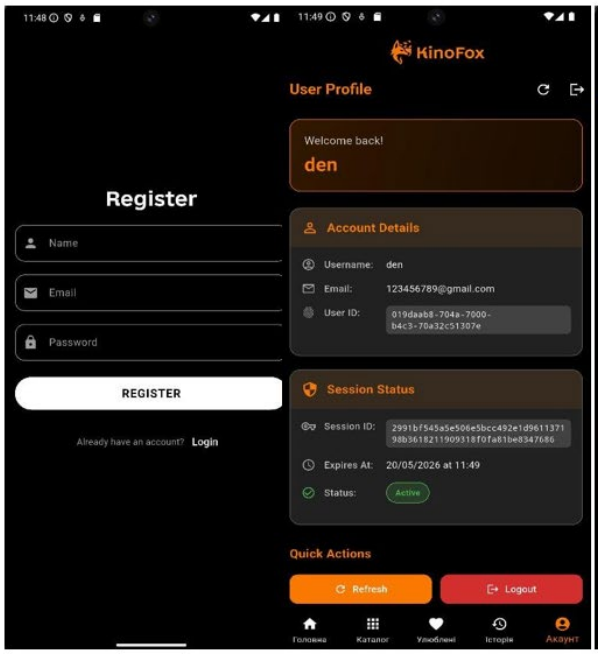
10

БІЗНЕС ЛОГІКА



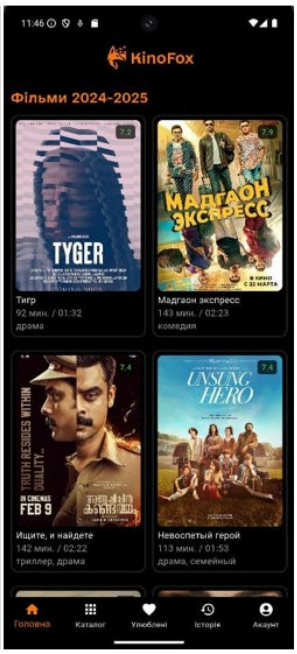
11

ЕКРАННІ ФОРМИ

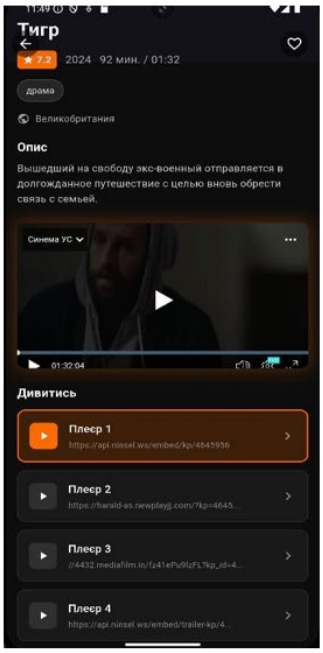


Вікно реєстрації KinoFox

сторінка акаунта



Головна сторінка KinoFox



Сторінка фільму з плеєром

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Цуркан Д.О. Шахматов І.О. Визначення вимог до системи безпеки Android-застосунку онлайн-кінотеатру з захистом даних користувачів та контенту. Матеріали ІІ Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.253 -255.

2. Цуркан Д.О. Шахматов І.О. Стратегії оптимізації продуктивності та ресурсомісткості Android-застосунку онлайн-кінотеатру. Матеріали VII Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 23 квітня 2026 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2026, С. 414-416.

ВИСНОВКИ

1. Проаналізовано предметну область Android-застосунків онлайн-кінотеатрів, що дозволило визначити основні підходи до організації подібних систем та виявити їх ключові характеристики.
2. Проаналізовано особливості функціонування Android-застосунків онлайн-кінотеатрів, що дозволило виявити типові архітектурні рішення, принципи роботи з медіаконтентом та взаємодії з користувачем.
3. Визначено функціональні та нефункціональні вимоги до застосунку, що дозволило чітко окреслити межі системи та сформувати основу для її проєктування.
4. Спроєктовано загальну архітектуру системи, структуру бази даних, серверну частину та REST API KinoFox, що забезпечило чітку організацію компонентів та взаємодію між ними.
5. Реалізовано клієнтську частину Android-застосунку, що забезпечило користувачу доступ до головної сторінки, каталогу, сторінки фільму, авторизації та профілю.
6. Реалізовано серверну частину для обробки запитів, авторизації та роботи з даними, що забезпечило повноцінну взаємодію клієнта з базою даних.
7. Реалізовано модуль автоматизованого збору контенту, який отримує, структурує та записує дані до PostgreSQL, що дозволило автоматизувати наповнення медіатеки без ручного втручання.
8. Проведено тестування основних функціональних можливостей застосунку, що дозволило підтвердити відповідність системи встановленим вимогам та оцінити її працездатність.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ ПРОГРАМНИХ МОДУЛІВ

Лістинг А.1 – Ініціалізація Flutter-застосунку та нижня навігація

```
// lib/main.dart
import 'package:flutter/material.dart';
import
'package:flutter_dotenv/flutter_dotenv.dar
t';
import 'package:kino_fox_ua/app.dart';

void main() async {
  await dotenv.load(fileName: ".env");
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'KinoFox',
      theme: ThemeData(fontFamily: 'e-
Ukraine'),
      debugShowCheckedModeBanner: false,
      home: App(),
    );
  }
}

// lib/app.dart
class App extends StatefulWidget {
  const App({super.key});

  @override
  State<App> createState() => _AppState();
}

class _AppState extends State<App> {
  int _selectedIndex = 0;

  static const List<Widget> _widgetOptions
= <Widget>[
    HomePage(),
    KatalogPage(),
    FavouritePage(),
    HistoryPage(),
    UserPage(),
  ];

  void _onItemTapped(int index) {
    setState(() {
      _selectedIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
        home: Scaffold(
          appBar: AppBar(
            title: Row(
              mainAxisSize:
MainAxisSize.min,
              children: [
                Image.asset('logo/kinofox-
nobg.png', width: 40, height: 40),
                const Text('KinoFox', style:
TextStyle(
                  color: Color(0xFFfa7902),
                  fontFamily: 'e-Ukraine
Head',
                  fontWeight:
FontWeight.w700,
                )),
              ],
            ),
            centerTitle: true,
            backgroundColor: Colors.black,
          ),
          body: Center(child:
_widgetOptions.elementAt(_selectedIndex)),
          bottomNavigationBar: BottomNavigationBar(
            items: const
<BottomNavigationBarItem>[
              BottomNavigationBarItem(icon:
Icon(Icons.home), label: 'Головна'),
              BottomNavigationBarItem(icon:
Icon(Icons.apps), label: 'Каталог'),
              BottomNavigationBarItem(icon:
Icon(Icons.favorite), label: 'Улюблені'),
              BottomNavigationBarItem(icon:
Icon(Icons.history), label: 'Історія'),
              BottomNavigationBarItem(icon:
Icon(Icons.account_circle), label:
'Акаунт'),
            ],
            currentIndex: _selectedIndex,
            selectedItemColor: const
Color(0xFFfa7902),
            unselectedItemColor:
Colors.white,
            backgroundColor: Colors.black,
            type:
BottomNavigationBarType.fixed,
            onTap: _onItemTapped,
          ),
        ),
      );
  }
}
```

Лістинг А.2 – Модель відеоконтенту та API-сервіс отримання добірок

```
// lib/models/film_model.dart
class Film {
  final String id;
  final String movieId;
  final String type;
  final String title;
  final String poster;
  final List<dynamic> scenes;
  final double rating;
  final int year_post;
  final List<dynamic> country;
  final List<dynamic> genrs;
  final String duration;
  final String plot;
  final String certification;
```

```
  final List<dynamic> playlist;

  Film({
    required this.id,
    required this.movieId,
    required this.type,
    required this.title,
    required this.poster,
    required this.scenes,
    required this.rating,
    required this.year_post,
    required this.country,
    required this.genrs,
    required this.duration,
    required this.plot,
    required this.certification,
    required this.playlist,
  });

  factory Film.fromJson(Map<String,
dynamic> json) {
    return Film(
          id: json['id'] ?? '',
          movieId: json['movie_id'] ?? '',
          type: json['type'] ?? '',
          title: json['title'] ?? '',
          poster: json['poster'] ?? '',
          scenes: json['scenes'] ?? [],
          rating: (json['rating'] ?? 0).toDouble(),
          year_post: json['year_post'] ?? json['year'] ?? 0,
          country: json['country'] ?? [],
          genrs: json['genrs'] ?? [],
          duration: json['duration'] ?? '',
          plot: json['plot'] ?? '',
          certification: json['certification'] ?? '',
          playlist: json['playlist'] ?? [],
    );
  }
}
```

```
       // lib/services/api/movie/rating_movie.dart
class ApiService {
  static String get baseUrl {
    final raw = dotenv.env['BASE_URL'] ??
'';
    return raw.endsWith('/') ?
raw.substring(0, raw.length - 1) : raw;
  }

  static Future<List<dynamic>>
fetchByType(String type) async {
    final uri =
Uri.parse('$baseUrl/movie/list');
    final body = jsonEncode({
      'page': 1,
      'size': 20,
      'filters': {
        'type': type,
        'yearRange': {'min': 2024, 'max':
2025},
        'ratingRange': {'min': 7, 'max':
10},
      }
    });

    final response = await http.post(
      uri,
      headers: {'Content-Type':
'application/json'},
      body: body,
    );

    if (response.statusCode == 200) {
      final Map<String, dynamic> data =
jsonDecode(response.body);
      return data['results'] ?? [];
    } else {
      throw Exception('Failed to load
$type: ${response.statusCode}');
    }
  }

  static Future<List<dynamic>>
fetchFilms() => fetchByType('film');
  static Future<List<dynamic>>
fetchMultfilms() =>
fetchByType('multfilm');
  static Future<List<dynamic>>
fetchSerials() => fetchByType('serials');
}
```

Лістинг А.3 – Точка входу серверної частини та групування API-маршрутів

```
       // API/src/index.ts
import { Elysia } from 'elysia';
import { swagger } from
'@elysiajs/swagger';
import movieRoutes from
'./routes/movie.routes';
import userRoutes from
'./routes/user.routes';
import favouriteRoutes from
'./routes/favourite.routes';
import historyRoutes from
'./routes/history.routes';

const app = new Elysia({ prefix: '/api' })
  .use(swagger())
  .use(movieRoutes)
  .use(userRoutes)
  .use(favouriteRoutes)
  .use(historyRoutes)
  .listen(process.env.PORT ?? 3000);

console.log('Elysia is running at',
`${app.server?.hostname}:${app.server?.por
t}`);
```

```
       // API/src/routes/movie.routes.ts
import { Elysia } from 'elysia';
import list from './movie/list';
import info from './movie/info';
import seach from './movie/search';

const movieRoutes = new Elysia({ prefix:
'/movie' })
  .use(list)
  .use(info)
  .use(seach);

export default movieRoutes;
```

```
       // API/src/routes/user.routes.ts
import { Elysia } from 'elysia';
import login from './user/login';
import registration from
'./user/registration';
import info from './user/info';

const userRoutes = new Elysia({ prefix:
'/auth' })
```

```
  .use(registration)
  .use(login)
  .use(info);

export default userRoutes;
```

Лістинг А.4 – Отримання списку фільмів і семантичний пошук на сервері

// API/src/services/movie/list.ts

```
export default async function (page:
number, size: number,
  filters?: MovieFilters, order?: Order) {
  const conditions: SQL[] = [];
  let orderCondition: SQL =
desc(movies.year_post);

  if (filters) {
    if (filters.type)
conditions.push(eq(movies.type,
filters.type));

    if (filters.ratingRange) {
      if (filters.ratingRange.max &&
filters.ratingRange.min) {

conditions.push(between(movies.rating,
          filters.ratingRange.min,
filters.ratingRange.max));
      } else if (filters.ratingRange.max)
{
        conditions.push(lte(movies.rating,
filters.ratingRange.max));
      } else if (filters.ratingRange.min)
{
        conditions.push(gte(movies.rating,
filters.ratingRange.min));
      }
    }

    if (filters.yearRange) {
      if (filters.yearRange.max &&
filters.yearRange.min) {

conditions.push(between(movies.year_post,
          filters.yearRange.min,
filters.yearRange.max));
      } else if (filters.yearRange.max) {

conditions.push(lte(movies.year_post,
filters.yearRange.max));
      } else if (filters.yearRange.min) {

conditions.push(gte(movies.year_post,
filters.yearRange.min));
      }
    }

    if (filters.country)
conditions.push(arrayOverlaps(movies.count
ry, filters.country));
    if (filters.genres)
conditions.push(arrayOverlaps(movies.genrs
, filters.genres));
  }

  const movie_count = await db.select({
count: count() })
    .from(movies)
        .where(conditions.length > 0 ? and(...conditions) :
undefined)
    .limit(1);

  const results = await db.select({
      id: movies.id,
      movie_id: movies.movie_id,
      title: movies.title,
      poster: movies.poster,
      type: movies.type,
      rating: movies.rating,
      year: movies.year_post,
      country: movies.country,
      genrs: movies.genrs,
      duration: movies.duration,
    })
    .from(movies)
        .where(conditions.length > 0 ? and(...conditions) :
undefined)
    .orderBy(orderCondition)
    .limit(size)
    .offset(page == 1 ? 0 : page * size);

  return { results, metadata: { page,
size, max_pages:
Math.ceil(movie_count[0].count / size) }
};
}
```

// API/src/services/movie/search.ts

```
export default async function (body:
Seach) {
  if (!body.search) throw new
Error('Search term is required');

  const embedding = await
generateEmbedding(body.search);
  const similarity = sql<number>`1 -
(${cosineDistance(movies.embedding,
embedding)})`;

  return await db.select({
      id: movies.id,
      movie_id: movies.movie_id,
      title: movies.title,
      poster: movies.poster,
      rating: movies.rating,
      year: movies.year_post,
      country: movies.country,
      genrs: movies.genrs,
      duration: movies.duration,
          similarity
    })
    .from(movies)
    .where(gt(similarity, 0.5))
    .orderBy((t) => desc(t.similarity));
}
```

Лістинг А.5 – Схеми бази даних і механізм користувацьких сесій

// API/src/db/schema/movies.ts

```
export const movies = pgTable('movies', {
  id: text('id').primaryKey(),
  movie_id: text('movie_id'),
  type: text('type'),
  title: text('title'),
  poster: text('poster'),
  scenes: text('scenes').array(),
  rating: real('rating'),
  year_post: integer('year_post'),
  country: text('country').array(),
  genrs: text('genrs').array(),
  duration: text('duration'),
  plot: text('plot'),
```

```
  certification: text('certification'),
  playlist: text('playlist').array(),
  embedding: vector('embedding', {
dimensions: 1536 }),
},
(table) => [
  index('embeddingIndex').using('hnsw',
table.embedding.op('vector_cosine_ops')),
]);
```

```
// API/src/db/schema/user.ts
export const users = pgTable('users', {
  id: text('id').primaryKey(),
  email: text('email').notNull().unique(),
  user_name: text('user_name').notNull(),
  password: text('password').notNull(),
  favourite: text('favourite').array(),
  lates_see: text('lates_see').array(),
});
```

```
// API/src/utils/Lucia/index.ts
export function generateSessionToken():
string {
  const bytes = new Uint8Array(20);
  crypto.getRandomValues(bytes);
  return
encodeBase32LowerCaseNoPadding(bytes);
}

export async function createSession(token:
string, userId: string): Promise<Session>
{
  const sessionId =
encodeHexLowerCase(sha256(new
TextEncoder().encode(token)));
  const session: Session = {
    id: sessionId,
    user_id: userId,
      expireAt: new Date(Date.now() + 1000 * 60 * 60 *
24 * 30)
  };
  await
db.insert(user_session).values(session);
  return session;
}

export async function
validateSessionToken(token: string):
Promise<SessionValidationResult> {
  const sessionId =
encodeHexLowerCase(sha256(new
TextEncoder().encode(token)));
  const result = await db.select({ user:
users, session: user_session })
    .from(user_session)
    .innerJoin(users,
eq(user_session.user_id, users.id))
    .where(eq(user_session.id,
sessionId));

  if (result.length < 1) return { session:
null, user: null };
  const { user, session } = result[0];

  if (Date.now() >=
session.expireAt.getTime()) {
    await
db.delete(user_session).where(eq(user_sess
ion.id, session.id));
    return { session: null, user: null };
  }
  return { session, user };
}
```

Лістинг А.6 – Основний цикл автоматизованого збору контенту та отримання URL-адрес

```
# Parser/ScripingEx-Fs/main.py
import asyncio
import time
from db.base import init_db, create_movie
from ExFsParser import MoviesUrlScraper,
MovieScraper

async def main():
    await init_db()
    start_time = time.time()

    movies_url_scraper =
MoviesUrlScraper('serials')
    max_pages =
movies_url_scraper.max_pages()

    for page in range(1, max_pages + 1):
        if page % 3 == 0:
            await asyncio.sleep(60)
        for movie_url in
movies_url_scraper.get_movie_urls(page):
            try:
                movie =
MovieScraper(movie_url)
                await create_movie(movie)
            except Exception as e:
              print(e)

    print(f"--- {(time.time() -
start_time):.2f} seconds ---")

if __name__ == "__main__":
        asyncio.run(main())
```

```
      #                              Parser/ScripingEx-
Fs/ExFsParser/MoviesUrlScraper.py
class MoviesUrlScraper:
    _site_url: str = 'https://ex-fs.net'
        _movie_type: str

    def __init__(self, movie_type: str):
        self._movie_type = movie_type

    def get_movie_urls(self, page: int):
        response =
requests.get(f'{self._site_url}/{self._mov
ie_type}/page/{page}/')
        soup =
BeautifulSoup(response.text,
features='html.parser')
        return
self._extract_movie_urls(soup)

    def max_pages(self):
        response =
requests.get(f'{self._site_url}/{self._mov
ie_type}')
        soup =
BeautifulSoup(response.text,
features='html.parser')
        nav = soup.find('div', {'class':
'navigations'})
        pages_array_str = []
        for page in nav.find_all('a'):
            pages_array_str.append(page.text)
```

```
            pages_array_str.pop()
        pages_array = [int(num_string) for
num_string in pages_array_str]
            pages_array.sort()
        return pages_array[-1]

    def _extract_movie_urls(self, soup) ->
List[str]:
        url_elements = soup.find_all('a',
{'class': 'MiniPostPoster'})
        return [element['href'] for
element in url_elements]
```

Лістинг А.7 – Парсинг сторінки фільму та створення або оновлення запису в PostgreSQL

```
        # Parser/ScripingEx-Fs/ExFsParser/MovieScraper.py
class Movie(BaseModel):
    id: Optional[str] = None
    type: Optional[str] = None
    title: Optional[str] = None
    poster: Optional[str] = None
    scenes: Optional[List[str]] = None
    rating: Optional[float] = None
    year: Optional[int] = None
    country: Optional[List[str]] = None
    genrs: Optional[List[str]] = None
    duration: Optional[str] = None
    plot: Optional[str] = None
    certification: Optional[str] = None
    playlist: Optional[List[str]] = None
        embedding: Optional[Any]

class MovieScraper:
    def __init__(self, movie_url: str):
        self._movie =
self._parse_site(movie_url)

    def get_id(self): return
self._movie.id if self._movie else 'No
movie data'
    def get_type(self): return
self._movie.type if self._movie else 'No
movie data'
    def get_title(self): return
self._movie.title if self._movie else 'No
movie data'
    def get_poster(self): return
self._movie.poster if self._movie else 'No
movie data'
    def get_rating(self): return
self._movie.rating if self._movie else 'No
movie data'
    def get_year(self): return
self._movie.year if self._movie else 'No
movie data'
    def get_playlist(self): return
self._movie.playlist if self._movie else
'No movie data'
    def get_embedding(self): return
self._movie.embedding if self._movie else
'No movie data'

    def _parse_site(self, movie_url: str)
-> Movie:
        response = requests.get(movie_url)
        response.encoding = 'utf-8'
        soup =
BeautifulSoup(response.text,
features='html.parser')

        id_film = movie_url.split('/')[-
1].split('.')[0]
        type_movie =
movie_url.split('/')[-2]
        name_film =
soup.select_one('h1.view-caption')
        name_film =
name_film.text.strip().replace('смотреть
онлайн', '').strip() \
            if name_film else 'Unknown
Title'

        poster_film = soup.find('div',
{'class': 'FullstoryFormLeft'})
        poster_url =
poster_film.find('img')['src'] \
            if poster_film and
poster_film.find('img') else None

        scenes = [element['href'] for
element in soup.find_all('a', {'class':
'lightbox'})]
        rating_film = soup.find('div',
{'class': 'in_name_imdb'}).text
        year_film = int(soup.find('p',
{'class': 'FullstoryInfoin'}).text)
        countries = [e.get_text() for e in
soup.select('p.FullstoryInfoin
a[href*="/country/"]')]
        genres = [e.get_text() for e in
soup.select('p.FullstoryInfoin
a[href*="/genre/"]')]
        duration_film =
soup.select_one('p.FullstoryInfoin:-soup-
contains("мин.")').text.strip()
        plot_film = soup.find('div',
{'class': 'FullstorySubFormText'}).text

        playlist_film = soup.find('div',
{'class': 'tab-content'})
        films_array = []
        for pane in
playlist_film.find_all('div', {'class':
'tab-pane'}):
            iframe = pane.find('iframe')
            if iframe:
              films_array.append(iframe['src'])

        json_embedding =
json.dumps(create_embedding(name_film))
        return Movie(id=id_film,
type=type_movie, title=name_film,
            poster=poster_url,
scenes=scenes, rating=float(rating_film),
            year=year_film,
country=countries, genrs=genres,
            duration=duration_film,
plot=plot_film,
            playlist=films_array,
embedding=json_embedding)

        # Parser/ScripingEx-Fs/db/base.py
async def create_movie(movie:
MovieScraper):
    cond = f"movie_id =
'{movie.get_id()}'"
    selected = await
db.select_data('movies', 'movie_id', cond)

    data = {
        'type': movie.get_type(),
```

```
        'title': movie.get_title(),
        'poster': movie.get_poster(),
        'scenes': movie.get_scenes(),
        'rating': movie.get_rating(),
        'year_post': movie.get_year(),
        'country': movie.get_country(),
        'genrs': movie.get_genrs(),
        'duration': movie.get_duration(),
        'plot': movie.get_plot(),
        'certification':
movie.get_certification(),
        'playlist': movie.get_playlist(),
        'embedding':
movie.get_embedding(),
    }

    if len(selected) > 0:
        await db.update_data('movies',
data, cond)
    else:
        data['id'] =
uuid.uuid4().__str__()
        data['movie_id'] = movie.get_id()
        await db.insert_data('movies',
data)
```

Лістинг А.8 – Реєстрація та авторизація користувача на серверному рівні

// API/src/services/users/login.ts

```
export default async function (body:
Login) {
  const dataUser = await db.select()
    .from(users)
    .where(eq(users.email, body.email))
    .limit(1);

  if (dataUser.length == 0) {
    throw new Error('Email or password
uncorect');
  }

  const isMatch = await
Bun.password.verify(body.password,
dataUser[0].password);

  if (isMatch == false) {
    throw new Error('Email or password
uncorect');
  }

  const token = generateSessionToken();
  await createSession(token,
dataUser[0].id);
  return token;
}
```

// API/src/services/users/registration.ts

```
export default async function (body:
Registration) {
  const existUser = await db.select({
count: count() })
    .from(users)
    .where(eq(users.email, body.email))
    .limit(1);

  if (existUser[0].count > 0) {
    throw new Error('User already exist');
  }

  const hashedPassword = await
Bun.password.hash(body.password);
  const user_id = await
Bun.randomUUIDv7();

  await db.insert(users).values({
    id: user_id,
    user_name: body.user_name,
    email: body.email,
    password: hashedPassword,
  });

  const token = generateSessionToken();
  await createSession(token, user_id);
  return token;
}
```