Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему:

Інформаційна мобільна система допомоги пошуку фільмів

Виконав: студент 6 курсу, групи КН-61
напряму підготовки
6.050101 – "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Крохмалюк Богдан Володимирович
(прізвище та ініціали)

Керівник Соколовський Ярослав Іванович
(прізвище та ініціали)

Рецензент *д.т.н.,професор Кособуський П. С.*
(прізвище та ініціали)

Львів – 2021 року

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"
(шифр і назва)

ЗАТВЕРДЖУЮ
завідувач кафедри

_____ Крошний І. М.
"____" _____20___ року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Крохмалюк Богдан Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна мобільна система допомоги пошуку фільмів

керівник роботи доктор технічних наук, професор, Соколовський Ярослав Іванович ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "31" грудня 2020 року № С-593
2. Термін подання студентом роботи
3. Вихідні дані до роботи смартфон на базі операційної систами IOS або Android із доступом до інтернету.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1.Вибір технологій; 2. Системний аналіз; 3. Проектування системи; 4. Розробка проекту; 5. Тестування; 6. Економічна вигода.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Дерево цілей, дерево проблем, схеми архітектур, діаграми (прецедентів, активностей, потоків даних, IDEF0, класів), скріншоти готового продукту, таблиці економічної характеристики

6. Дата видачі завдання 18 грудня 2020

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел та інтернет ресурсів, пошук загальної інформації за вказаною темою	04.02.2021 20.02.2021	
2	Вибір технологій, підходів і фреймворків для поставленої задачі	09.03.2021 21.03.2021	
3	Проектування програмного забезпечення, налаштування середовища	03.04.2021 20.04.2021	
4	Створення користувацького інтерфейсу	01.06.2021 20.06.2021	
5	Імплементація функціональності соціальної мережі, завершення створення дизайну	*24.06.2021* *30.07.2021*	
6	Тестування програми та аналіз результатів	02.08.2021 26.08.2021	
7	Оформлення записки до дипломної роботи	28.10.2021 30.11.2021	
8	Здача пояснювальної записки на рецензування	01.12.2021 10.12.2021	
9	Створення презентації та підготовка до захисту	11.12.2021 15.12.2021	

Студент _____ Крохмалюк Б. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Соколовський Я. І.
(підпис) (прізвище та ініціали)

ТЕХНІЧНЕ ЗАВДАННЯ

Необхідно розробити програмне забезпечення мобільної системи пошуку фільмів, акторів, знімальної групи та їх фільмографії:

1. Реалізувати сторінку із різноманітними категоріями фільмами;
2. Пошук фільмів, акторів та людей із знімальної групи;
3. Можливість збереження фільмів в різні користувацькі списки;
4. Функціонал менеджменту користувацьких списків;
5. Перегляд трейлеру до фільму;

Можливість користування даною мережею на будь-якому мобільному пристрої піл керування Android та IOS операційних систем, зручний, простий і інтуїтивно зрозумілий інтерфейс

РЕФЕРАТ

Магістерська дипломна робота на тему “Інформаційна мобільна система допомоги пошуку фільмів”.

Магістерська робота складається з 82 сторінок друкованого тексту, з 7 розділів, 32 рисунків, 10 таблиць і налічує 21 джерело використаної літератури.

У першому розділі висвітлені базові вимоги системи та вимоги до окремих компонентів. У другому розділі розглянуто літературу пов’язану із проектом та вибором технології розробки. У третьому аналізуються підходи до розробки мобільних системи та визначається оптимальний. Четвертий розділ складається із діаграм системи. П’ятий описує розроблені та використані бібліотеки, відображає користувацький інтерфейс розробленої програми. У шостому розділі описані процеси встановлення додатку, та відстеження помилок. Сьомий розділ визначає економічний потенціал роботи.

В роботі розглянуто основні технології для побудови мобільних додатків. Докладно висвітлено актуальність проблеми браку якісних ресурсів для пошуку фільмів та досліджено головні принципи побудови мобільних додатків на різні платформи. Проаналізовано економічну складову розробки програмного забезпечення для подальшого управління даним проектом. Для виконання даної роботи використано здобуті нові знання з об'єктно-орієнтованого програмування, системного аналізу та тестування. Значне місце у роботі приділене опису програмного продукту та відстежуванню збоїв програми на різних пристроях.

Ключові слова: підбір фільмів, пошук, кросплатформність, гнучкий дизайн, архітектурне рішення, BLoC, Dart, Flutter, Crashlytics, SQFlite, Shared Preferences, TMDB

ABSTRACT

Master thesis topic "Mobile application for movie selection support"

Master thesis consists of 82 pages of printed text, 7 chapters, 32 figures, 10 tables and has 21 sources of literature.

The first section highlights the basic requirements for system and its components. The second section contains the information related to the project and explains the choice of development technologies. In the third section, there are different approaches to the development of mobile systems and analysis between them. The fourth section contains system diagrams used for development. The fifth section reveals information about developed and used libraries used for creation of the project. Also, you will find here screenshots with application interface. The sixth section describes the additional details about system installation and error tracking. The seventh section explains the economic potential of the work.

The document considers the main technologies for building mobile applications. The urgency of the problem of lack of quality resources for searching movies is highlighted in detail and the main principles of building mobile applications on different platforms are studied. The economic component of software development for further management of this project is analyzed. This work uses the acquired knowledge of object-oriented programming, systems analysis, and testing. A significant place in the work is given to the description of the software product and tracking program failures on various devices.

Keywords: movie selection, search, crosplatform, flexible design, architectural solution, BLoC, Dart, Flutter, Crashlytics, SQFlite, Shared Preferences, TMDB

Зміст

ПЕРЕЛІК СКОРОЧЕНЬ ТА ТЕРМІНІВ

SDK - набір із засобів розробки, утиліт і документації, який дозволяє програмістам створювати програми.

IDE – комплексне рішення для розробки програмного забезпечення.

ПЗ – програмне забезпечення.

ОС – операційна система.

IOS – мобільна операційна система від компанії Apple.

Android – мобільна операційна система від компанії Google.

UI – користувацький інтерфейс.

Widget – у Flutter SDK – це одиниця визначення та конструювання UI.

State – у Flutter SDK – це набір інформації який використовується для відображення UI.

JSON – це текстовий формат обміну даними між комп'ютерами.

URL – стандартизована адреса певного ресурсу в інтернеті чи іншій мережі.

SSH – протокол, що дозволяє проводити віддалене управління комп'ютером.

Dependency Injection – шаблон проектування ПЗ, що передбачає використання зовнішньої залежності замість головного класу, для отримання залежностей.

Крашлітика – це система моніторингу проблем різної пріоритетності, що виникають у розробленому ПЗ.

TMDB API – це набір визначених запитів до ресурсу TMDB, які при наявності авторизаційного ключа, надають інформацію про кінострічки та людей із кіноіндустрії.

ВСТУП

Актуальність роботи

Важливою характеристикою людської діяльності в теперішній час є вміння поєднувати багато завдань протягом дня та відпочинок з сім'єю чи друзями. Відповідно для виконання поставленої мети необхідно багато часу та планування кожного наступного кроку. На допомогу сучасній людині приходять сучасні вирішення – мобільні додатки та інтернет-ресурси, які зменшують в рази затрати часу на виконання дрібних завдань.

Сьогодні мало хто телефонує в службу таксі, щоб кудись доїхати, чи очікує в довгих чергах банку, щоб просто дізнатись свій баланс. Тепер це можна зробити за лічені хвилини в своєму смартфоні. Мобільні додатки є доступними, захищеними та зручними в користуванні, тому все більше сервісів стають доступними онлайн.

Базуючись на своєму досвіді та опитуванні інших людей, отримано висновок, що проблема вибору фільму не є вирішеною, адже існуючі інтернет-ресурси не охоплюють багато важливих аспектів, таких, як вподобання людини чи попередні переглянуті фільми. Отже, доцільність даної роботи полягає в вирішенні даної проблеми та створенні зручного мобільного додатку, що буде доступним на будь-якому смартфоні.

Сьогодні практично кожна людина любить дивитися фільми, адже це гарне проведення часу, яке допоможе розслабитися, відпочити після робочого дня. Але, на превеликий жаль, не всі фільми можна назвати хорошими, є і такі, які тільки зіпсують проведення часу і змусять турбуватися.

Найціннішим ресурсом для людини є час. І всі мобільні додатки спрямовані на те, щоб зекономити людині хоча б декілька хвилин. Великий вибір кіно, пропонований мережею інтернет, може бути втратою власного часу та ще тим головним болем для людини яка не знає, що хоче подивитись.

Ось кілька факторів, що вдало розкривають актуальність даного мобільного застосунку:

1. Великий попит на перегляд фільмів в умовах карантину;
2. Через малу кількість якісних додатків для вибору кіно, більшість людей використовують сайти, які зазвичай не є зручними для користування на смартфоні;
3. Часто люди збираються переглянути стрічку, проте через ті чи інші причини забувають про неї та не можуть потім згадати;
4. Не завжди люди одразу дізнаються про вихід нового фільму, який міг би бути для них цікавим;
5. Часто люди можуть запитувати своїх друзів пораду щодо вибору фільму, та вони не завжди можуть відповісти, адже не пам’ятають, що переглянули і що варто порадити.

Метою магістерської роботи — є створення мобільного додатку, який дозволяє здійснювати швидкий вибір фільмів і надавати користувачам можливість бачити найпопулярніші та найновіші кінокартини відразу при відкритті програми.

Для досягнення поставленої мети необхідно виконати наступні задачі:

- Провести аналіз ринку, користувачів, які використовують даний тип мобільних додатків.
- Проаналізувати конкурентів та аналогів;
- Обґрунтувати інструменти та технології для розробки мобільного додатку;
- Спроектувати архітектуру мобільного додатку;
- Програмно реалізувати розробку мобільного додатку;

Об’єктом дослідження є інформаційна система для пошуку фільмів, додаткових даних про акторів та знімальну групу, колекції та рейтинг фільмів.

Предметом дослідження є кросплатформні мобільні додатки.

Новизна роботи полягає у тому що даний мобільний додаток надає можливість здійснювати не тільки пошук фільмів, а й пошук акторів та членів знімальної групи, і зокрема надає фільмографію кожної людини з індустрії кіно. Також це єдиний мобільний додаток для пошуку фільмів який готовий для використання на платформах Android та IOS а також може бути перенесеним на Windows/Linux/MacOS та Web з невеликими змінами в коді.

Практична цінність полягає швидкому процесі вибору фільмів для особистих інтересів, ввести каталог переглянутих фільмів і відзначати які сподобались. Дана інформаційна система спрощує процес вибору фільму враховуючи при цьому фактори пошуку та попередню історію переглядів.

РОЗДІЛ 1
ХАРАКТЕРИСТИКА ОБ'ЄКТУ ПРОЕКТУВАННЯ

Завданням цього мобільного додатку, що містить актуальну інформацію про фільми та дає змогу знайти в найкоротші терміни бажану стрічку, є реалізувати, можливо, поширити мобільні додатки для знаходження кінострічок. Мобільний застосунок повинен виконувати функцію пошуку та збереження переглянутих та улюблених фільмів, для того, щоб користувач із середньостатистичним телефоном мав можливість завантажити даний застосунок та зручно користуватися ним, не зважаючи на апаратне забезпечення смартфону та операційну систему.

Вимоги до окремих компонентів системи:

1. Модуль цікавої інформації про фільми:
   - Можливість побачити фільми, які зараз ідуть в кінотеатрах.
   - Можливість побачити фільми, що наразі користуються популярністю.
   - Можливість побачити фільми, які з'являться в найближчому майбутньому.
   - Можливість побачити фільми, що займають топ рейтингів усіх часів.
2. Модуль пошуку фільмів та акторів:
   - Можливість пошуку інформації про фільми за введеним користувачем запитом.
   - Можливість пошуку акторів та інших людей в кіноіндустрії за іменем.
3. Модуль відображення списків фільмів користувача:
   - Список улюблених фільмів.
   - Список фільмів, які користувач хоче подивитись.
4. Модуль відображення інформації про фільм:
   - Обкладинка.

- Загальний опис.
- Трейлер.
- Актори.
- Колекція фільмів в яку дана картина входить.
- Оцінка та інша цікава інформація.

5. Модуль відображення інформації про актора/кіно-персоналу:
    - Фото та ім’я.
    - Список фільмів в яких знімався актор та назва його персонажу.
    - Список фільмів в яких брав участь персонал та його відділ і посада.
6. Модуль із різними збереженими списками фільмів.
7. Модуль із деталями про список фільмів.

Система мусить оперувати фільмами, акторами та іншими даними які несуть певну інформацію для користувача. Сутності, які будуть задіяні в програмі:

- Фільм;
- Деталізований фільм;
- Акторський склад;
- Склад людей що працювали над фільмом;
- Актор;
- Працівник;
- Інформація про країну;
- Колекція фільмів;

Функціонал, що необхідний мобільному застосунку:

- Можливість перегляду основних категорій фільмів;
- Можливість перегляду інформації про фільм;
- Відображення інформації про акторський склад;
- Інформація про склад людей які працювали над фільмом;
- Перегляд фільмографії актора;

- Перегляд фільмографії людини що працює над фільмами;
- Перегляд трейлеру до фільму;
- Збереження фільму до списку улюблених;
- Збереження фільму до списку переглянутих;
- Можливість змінити локалізацію додатку;
- Пошук фільму;
- Пошук акторів.

РОЗДІЛ 2
ОГЛЯД ЛІТЕРАТУРИ

Мобільні додатки замінили веб-програми та спосіб придбання товарів, пошуку інформації, спільного використання чи продажу продукції. Компанії повинні пропонувати зручні програми, які працюють безперебійно на різних мобільних платформах, щоб залишатись конкурентно-спроможними в порівнянні з іншими компаніями.

2.1 Android і його характеристика

За рівнем масштабів та різноманітністю з Android нічого не конкурує. Можна обрати дорожчі варіанти - наприклад, найновіша лінійка Z Samsung коштує майже 1400 доларів, але також існує величезний вибір хороших, недорогих телефонів від найрізноманітніших виробників. Платформа була свідомо оптимізована працювати на апаратному забезпеченні низького класу. Те, що Android лідирує в галузі у безкоштовних додатків, робить його природним вибором для бюджетних проектів.

За даними Google Trends за пошуком ‘Android’ найбільший рівень зацікавленістю темою є в Індії, Індонезії та Марокко.

Правда полягає в тому, що розробка Android коштує приблизно так само, як і розробка iOS. Єдине, що різноманітність пристроїв Android та розмірів екранів ускладнює роботу цього середовища, а отже, для розробки програм Android може знадобитися трохи більше часу.

2.2 Опис платформи IOS

Apple завжди була на високому рівні з точки зору ціноутворення, але iPhone X збільшив планку на абсолютно новий рівень зі стартовою ціною 1000 доларів, а ціни на iPhone 11 Pro Max починаються від 1100 доларів. Якщо потрібно щось більш доступне, iPhone 11 починається від 700 доларів, а останній iPhone SE - від 400 доларів.

За даними Google Trends за пошуком ‘iPhone’ найбільший рівень зацікавленістю темою є в США, Росії та Саудівській Аравії.

За статистикою Clutch.co, середня вартість розробки додатків Swift для iOS компаніями, що базуються в США та Великобританії, становить 50 000 доларів. Однак фактична ціна програми може залежати від кількох факторів, таких як стек технологій, архітектура та інше.

2.3 Кросплатформність і її властивості

У сьогоднішньому світі, що швидко розвивається, ринок мобільних додатків розширюється надзвичайно швидко. Отже, мобільний маркетинг стає все більш конкурентоспроможним. Щоб забезпечити видимість мобільного додатка за таким складним сценарієм, необхідно бути дуже конкретним щодо підходу, який слід дотримуватися для розробки мобільних додатків. Щоб створити успішний мобільний додаток, потрібно дотримуватися системного підходу до розробки додатків.

Хороший мобільний додаток може багато в чому допомогти бізнесу. Наприклад, збільшити дохід за рахунок поліпшення продажів або введення нового потоку доходів, чи збільшити залучення та співтовариство, надаючи ресурс для своєї аудиторії. Можна поліпшити спілкування працівників, будучи основним внутрішнім додатком для вашого бізнесу та підвищити рівень поінформованості про бренд та вдосконалити стратегію мобільного маркетингу.

Як ми знаємо, кросплатформна розробка додатків - це спосіб створення мобільних додатків таким чином, щоб їх можна було запускати на різних платформах. Розробникам подобається такий підхід, оскільки в цьому програмування виконується лише один раз, і додаток підтримується Android, iOS або Windows.

З розвитком архітектурних підходів та сторонніх бібліотек, які мають спільну логіку, структуру та шаблони поведінки, при розробці мобільних

застосунків все менше бізнес логіки залежить від платформної реалізації. Цю незалежну логіку можна легко виносити у спільні модулі та поширювати між реалізаціями програм, створених під певні системи розробки додатків[15].

Розробка мультиплатформних мобільних додатків стає все більше популярною на сучасному ринку розробників. Це спричинено збільшенням використання мобільними платформами користувачами та загальною його ефективністю. Галузь йде в ногу з тенденціями, орієнтується на оптимізацію та економію виробничих витрат при одночасному підвищенні або принаймні збереженні існуючої продуктивності.

Цей та багато інших чинників сприяють поширенню тенденцій мультиплатформової діяльності.

2.4 Flutter SDK і її характеристики

Flutter - молода, але дуже перспективна платформа, що вже привернула до себе увагу великих компаній, які запустили свої додатки. Цікава ця платформа своєю простотою порівнянно з розробкою веб-додатків, і швидкістю роботи на рівні з нативними додатками[1]. Висока продуктивність програми і швидкість розробки досягається за рахунок декількох технік:

1. На відміну від багатьох відомих на сьогоднішній день мобільних платформ, Flutter не використовує JavaScript ні в якому вигляді. В якості мови програмування для Flutter вибрали Dart, який компілюється в бінарний код, за рахунок чого досягається швидкість виконання операцій порівнянна з Objective-C, Swift, Java або Kotlin.
2. Flutter не використовує нативні компоненти для промальовування інтерфейсу, так що не доводиться писати ніяких прошарків для комунікації з ними. SDK відмальовує увесь інтерфейс самостійно. Кнопки, текст, медіа-елементи, фон - все це промальовується всередині

графічного двигуна в самому Flutter. Після вищесказаного варто відзначити, що "Hello World" додаток на Flutter займає зовсім небагато місця: iOS ≈ 2.5Mb і Android ≈ 4Mb.

3. Для побудови UI під Flutter використовується декларативний підхід, який був написаний, на основі ReactJS та віджетів. Для ще більшого приросту в швидкості роботи інтерфейсу віджети перемальовуються виключно при необхідності, тобто зміні даних які відображаються на UI.
4. У фреймворк вбудована Hot-reload функція, що дозволяє вносити зміни в код і одразу бачити їх на екрані.

Flutter активно просувається Google, поступово набирає популярність і, швидше за все, в подальшому буде витіснювати інші технології, які використовуються зараз в сегменті кросплатформної розробки (React Native, Xamarin, та інші), особливо за умови широкого поширення Fuchsia. З урахуванням того, що Google позиціонує дану операційну систему як заміну Android, рано чи пізно Flutter витіснить нативну розробку під Android. Тому перспективність і активний розвиток - основні плюси Flutter.

Як працює Flutter

На мові програмування Dart створюється мобільний додаток з описом графічного інтерфейсу і всією логікою проекту. Проект додається в нативний додаток, як і картинки, шрифти, тощо (зрозуміло, цей процес автоматизований).

Одночасно в нативній частині програми створюється один-єдиний екран, де підгружається віртуальна машина Dart, яка і виконує Flutter код. Один з незначних мінусів є те, що кінцевий інсталяційний пакет важить більше, так як в нього додається віртуальна машина Dart[14].

У складі віртуальної машини є власний графічний двигун, він малює інтерфейс програми з усіма переходами між екранами, діалогами, фрагментами і т.д. У цьому розробка під Flutter значно відрізняється від розробки з Xamarin і React Native, які використовують реальні Android і iOS компоненти. У випадку з ними неможливо використовувати специфічні для платформи компоненти, якщо така необхідність є, доводиться створювати два переходи між нативним UI та інтерфейсом Flutter.

Специфіка фреймворку

Перше, що кидається в очі - це спосіб створення екранів, який значно відрізняється від використовуваних на Android і iOS. В Android розділена логіка і інтерфейс: логіка задається кодом, а інтерфейс - верстkою в xml. На Flutter все це задається за допомогою коду. Хоча тут для інтерфейсу використовується особливий стиль – елементи інтерфейсу створюються вкладеними один в одний. Це трохи схоже на верстку, дуже схожий спосіб діє в React Native. При цьому відсутня можливість прямого доступу до елементів. Щоб щось змінити на екрані поза межами одного віджету, потрібно використовувати архітектурні рішення[10].

Оскільки інтерфейс створюється за допомогою коду, через що границя між логікою і дизайном набагато тонша, це можна вважати мінусом. З іншого боку, даний підхід дозволяє простіше розбивати екрани на окремі

компоненти. По суті, будь-який блок вкладених елементів інтерфейсу можна винести в окремий віджет всього за пару кроків, і це набагато простіше, аніж створення різноманітних View і фрагментів.

У Flutter віджети поділяються на три типи:

- StatelessWidget
- StatefulWidget
- InheritedWidget

InheritedWidget – віджет, що не має UI частини і ніяк не промальовується. Єдина функція цього класу – можливість передавати дані в низ по дереву UI віджетів.

StatelessWidget – описується одним класом з фіксованим станом, і його можна змінити тільки шляхом зміни стану батьківського елемента.

StatefulWidget – компонента, для роботи якої потрібно два класи: клас самого віджета і клас його стану, в якому і буде відбуватися основна робота. Клас віджету слугує як прошарок прийому даних від батьківського віджету. Клас стану може перебудовувати себе в залежності від зміни даних стану, або ж при заданих розробником правилах. Для цього в момент, коли потрібно виконати перебудування – достатньо викликати функцію setState(). Ця функція повідомляє фреймворк, що внутрішній стан змінився і потрібно перемалювати об’єкти.

Розберемо життєвий цикл віджета який є найбільш використовуваним і єдиним який може міняти свій стан не тільки від зміни батьківських класів [17].

CreateState() – коли ми створюємо новий StatefulWidget, він викликає createState () одразу, і цей метод обов’язково повинен існувати.

InitState() – перший метод викликається після створення віджету. Це еквівалент OnCreate () та viewDidLoad () в Android та IOS платформах.

Mounted - усі віджети мають цей параметр який отримує значення true, коли buildContext призначений і в даний час знаходиться на дереві віджетів. Ці значення зберігатимуться доки не викличуть метод dispose().

DidChangeDependencies() – цей метод задіяний відразу після initState(). Також, якщо StatefulWidgets залежить від InheritedWidget, він знову викликатиметься, якщо буде потрібна перебудова віджету.

Build() – можна точно сказати, що цей метод є найбільш “важливим”. Тут він ретранслює ваше ціле дерево віджетів для візуалізації та викликається відразу після didChangeDependencies (). Тут відображається весь графічний інтерфейс, і він буде викликатися кожен раз, коли користувальницький інтерфейс потребує візуалізації.

DidUpdateWidget() – даний метод буде викликаний, як тільки головний віджет змінив свої параметри і потрібно переробити інтерфейс користувача. Метод надає параметр oldWidget і його можна порівняти із теперішнім, щоб на основі цього зробити додаткову логіку.

Dispose() – цей дуже важливий метод призначений для того щоб очистити ресурси віджета як тільки даний об’єкт буде видалений із дерева.

2.5 Мова програмування Dart

Dart – високорівнева, інтерпретована мова програмування. Розробляється корпорацією Google і є альтернативою JavaScript - принаймні так позиціонує цю мову програмування сама компанія. Перша версія інтерпретатора стала доступна в середині осені 2011 року[9].

При створенні мови програмісти прагнули зробити його:

- Схожим на інші мови, щоб Dart було відносно просто та швидко освоїти;
- Структурованим і гнучким для швидкої і зручної веб-розробки;

- Швидким – щоб написані на даній мові додатки працювали блискавично як у браузерах, так і в інших середовищах, на смартфонах.

Dart програмування сильно нагадує написання коду на C і JavaScript. Людина без спеціальної підготовки, побачивши частину інструкцій, може легко подумати, що вони написані саме на C. Це універсальна мова. З її допомогою можна створювати утиліти командного рядка, серверні додатки, займатися Web-розробкою і робити додатки для мобільних платформ.

Увесь інструменти мови написані на самій ній же. Тобто pub, analyzer, dartdoc, dartfmt, dart2js - все це створено з використанням самого Dart. Утиліти командного рядку досить просто писати з урахуванням того, що є прості інструкції на офіційних ресурсах мови.

Однією дуже крутою особливістю віртуальної машини Dart є запуск відладчика і профілювальника, який входить в саму машину і називається Observatory. Як стверджує Google, відладчик практично не впливає на продуктивність виконуваного коду, при цьому через цей інструмент не можна підмінити виконувані вихідні дані, але можна змінювати стан.

Мова підтримує розробку веб додатків, можна писати на чистому Dart без фреймворків або використовувати пакет AngularDart. Також мова має фреймворк Flutter для розробки мобільних додатків[13].

2.6 Чому саме Flutter

Уникнення специфічних платформних проблем

Оскільки даний SDK – це мультиплатформне, тому не потрібно інвестувати індивідуально у кожну платформу. У звичайному розвитку подій де розробляються Android та IOS додатки, компанія витрачає вдвічі більше грошей для створення того чи іншого проекту, і часто створення одного і того ж самого функціоналу займатиме різний час для обох платформ, що заважає своєчасно оновлювати функціонал для користувачів

різних платфом. Швидкість розробки певного функціоналу залежить від багатьох факторів, таких як: розробники, архітектури розробки та специфіки платформи. На противагу, із платформою Flutter як правило працює одна команда і не існує великої кількості розбіжностей для розробки під якусь конкретну платформу [18].

Перевикористання коду

Мова Dart дуже гнучка і дозволяє розробникам з легкістю створити код один єдиний раз і перевикористовувати у безлічі місць. За допомогою Dependency Injection один і той самий клас логіки може мати багато різних використань і видозмінень, залежно від випадку в якому використовується. Завдяки структурі побудови застосунків під SDK Flutter всі UI компоненти можуть бути використані знову, що допомагає будувати простий, логічний UI, притримуватись стилю та розробляти із рекордною швидкістю.

Плагіни та бібліотеки

Завдяки великій спільноті у вільному доступі є широкий спектр плагінів та бібліотек, які роблять процес розробки швидшим та простішим. Доступ до великої кількості різноманітних плагінів є відкритим і безкоштовним, тому це вирішує велику частину головного болю команди розробників.

Швидке тестування

Через всі вищезгадані причини як правило код пишеться один раз, і розробники повторно використовують його переносячи у різні плагінів та бібліотеки, а отже єдиного тестування якості достатньо для тестування всіх функцій окремого плагіну. Тобто протестувавши єдиний раз той чи інший функціонал, перетестовувати його потрібно буде не швидко.

Дизайн

Інтерактивний дизайн мобільного користувальницького інтерфейсу відіграє вирішальну роль у зацікавленності користувачів та дає більший шанс утримання клієнтів (тобто що клієнт не видалить саме ваш додаток із часом). Flutter надзвичайно добре працює з фірмовими наборами віджетів.

Насправді, оскільки Flutter має власний набір віджетів, це дуже корисно, коли ви не хочете повністю створювати дизайн інтерфейсу, а лише створити частину, а решту підігнати під колірну гаму.

Проста інтеграція анімацій

Платформа використовує величезну кількість наперед заготовлених віджетів для анімацій яка покриває 99% затребувань розробників. Тим не менше, створити власну анімацію теж дуже просто і логічно. Якщо навчитись це робити один раз – потім з легкістю можна анімувати все що завгодно.

Проста документація

Flutter легко вивчити та почати писати код через зрозумілу документацію, яку вона надає своїм користувачам. Ви можете знайти відповіді на запитання запити розгорнуто та із детальним поясненням в документації Flutter.

Виходячи із вище вказаної інформації можна зробити висновок, що Android системи на сьогодні найпоширеніші власники смартфонів з даною операційною системою складають близько 80%, на противагу власники смартфонів на платформі IOS часто люди із кращим достатком і імовірність того що вони придбають вашу програму є вищою. Очевидно, що найвигідніший варіант розробки – це розробка під обидві платформи одразу.

На сьогодні є велика кількість технологій для мультиплатформної розробки, проте більшість з них значно програє у стабільності та швидкості роботи звичайній нативній розробці. Тим не менше, найвигіднішим варіантом в даній задачі буде розробити програму із фреймворком Flutter – новітній технології компанії Google. Вона є відносно новою у світі розробки проте вже набирає неабияку популярність і володіє безліччю бібліотек для швидкої та комфортної роботи.

РОЗДІЛ 3
СИСТЕМНИЙ АНАЛІЗ

3.1 Дерево проблем та цілей

Для формулювання та розуміння основних задач програми і подальшого вибору архітектури побудови системи потрібно побудувати дерево проблем та дерево цілей.

Дерево проблем

Знизу дерева відображаються причини, через які створюється центральна основна задача, яку вирішується системою, зверху відображені наслідки до яких ця проблема призводить. Створене дерево проблем відображено на рисунку 3.1.

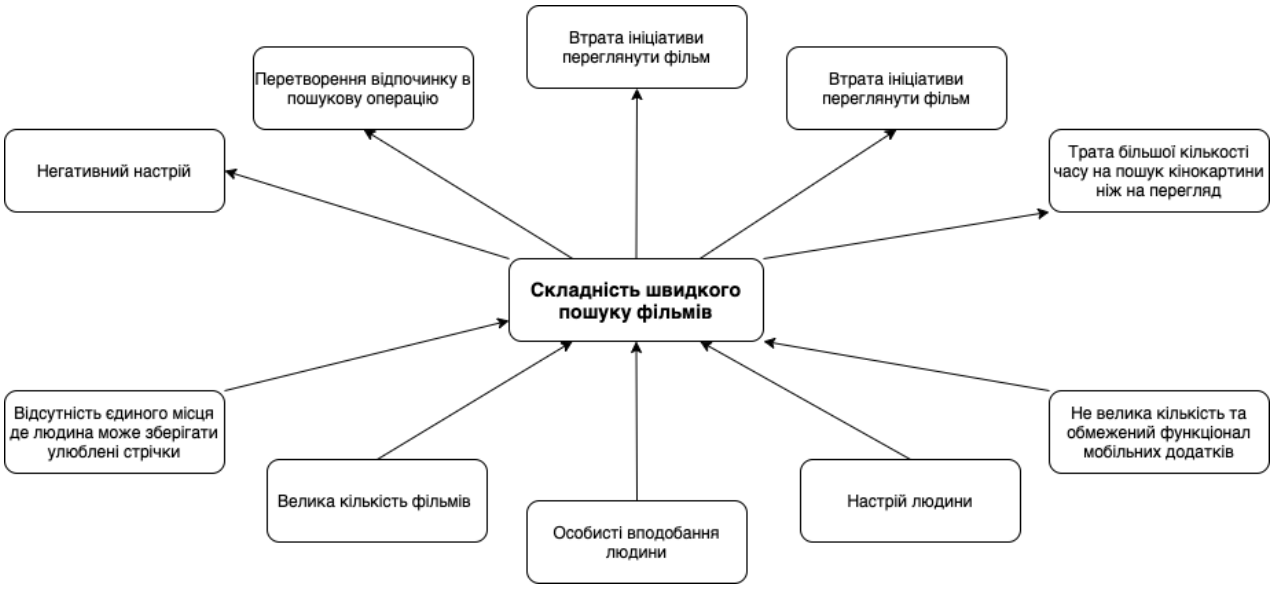

Рис 3.1 Дерево проблем

Оскільки додаток, що розробляється буде позиціонуватись як інформаційно-розважальний, очевидно що система не вирішуватиме глобальних проблем, проте вона покликана для вирішення проблем із зручністю та швидкістю пошуку та перегляду інформації, що для двадцять першого століття є досить вагомим фактором для використання саме моєї програми.

Дерево цілей

Успішна програма повинна базуватись на вирішенні певних проблем, які у нашому випадку представленні у дереві проблем. На рисунку 3.2 побудоване дерево цілей, на якому можна побачити, які цілі потрібно досягнути щоб користувачі змогли легко користуватись програмним продуктом.

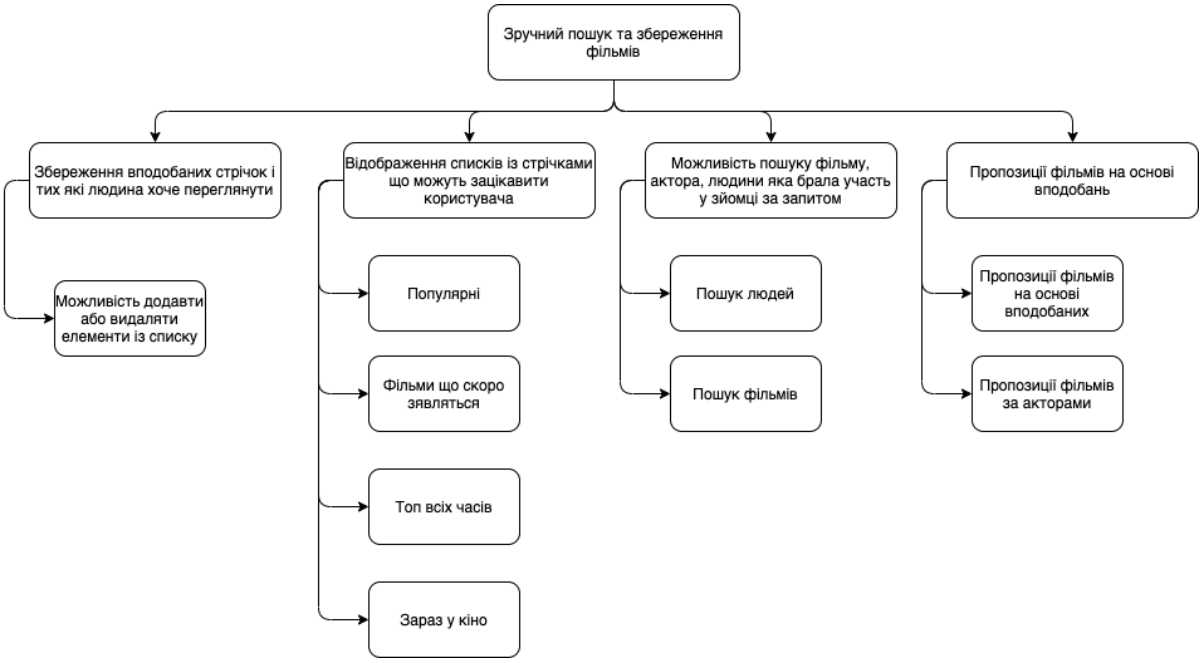

Рис 3.2 Дерево цілей

Зверху дерева цілей відображається головна ціль, яка є комплексною та складається з менш складних, які в свою чергу теж можуть розбиватись на підцілі. Вирішення проблем призводить до отримання певних цілей, які мають бути досягненні після виконання програми для успішного і повного старту роботи користувачів.

3.2 Аналіз архітектур для проблемної області

Архітектура програного забезпечення намагається визначити як найкраще розбити систему на частини, як ці частини визначають та взаємодіють одна з одною, як між ними передається інформація, як ці частини розвиваються поодинці [3].

Першим кроком і найважливішим для розробки ПЗ є вибір архітектури коду додатку. В залежності від цього рішення, буде залежати як швидкість розробки так і можливість додавати нові функції до програми в майбутньому. Розглянемо дві найбільш використовуваних архітектур, що використовуються при розробці із Flutter:

- BLoC
- Redux

3.2.1 BLoC

BLoC (Business Logic Component) - шаблон, створений Google для управління складним станом додатку, ґрунтуючись на реактивній парадигмі [19].

Основна ідея полягає в тому, що наш додаток розбитий на модулі, що реалізують бізнес-логіку. Кожен модуль має одну або кілька Sink (труб), які є деяким вхідними потоками для агрегування подій ззовні. Для вихідних даних використовується Stream (потік), який визначає асинхронний формат даних для віджетів. Щоб скористатися модулем на рівні керування,

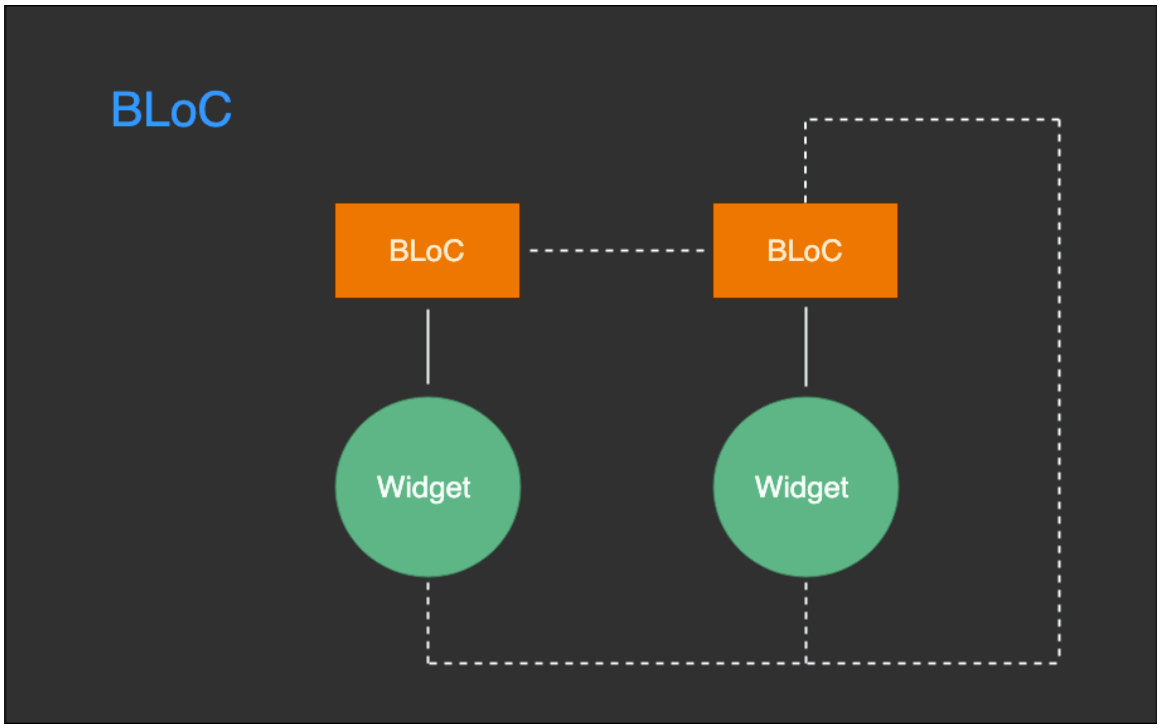

Рис 3.5 Схема архітектури BLoC

застосовують спеціальний віджет StreamBuilder, який керує потоком даних і автоматично вирішує проблеми підписки і перемальовування дочірнього дерева віджетів[12].

Незважаючи на це, використовувати BLoC в чистому вигляді - досить складна робота, оскільки треба застосовувати бібліотеку RxDart для маніпуляції з потоками, вручну відписуватися від потоків, інакше можна отримати витік пам'яті.

Переваги і недоліки

Сам підхід цікавий і має велику кількість переваг:

- угруповання логіки в одному місці;
- легкість в тестуванні стану з сайд-ефектами за рахунок вбудованого в Dart API тестування потоків;
- мінімальна кількість провалювань завдяки використанню StreamBuilder (і звісно при розумному використанні блоків).

З недоліків можна виділити тільки завищену складність для початківців, оскільки не всі розробники можуть швидко вникнути в суть роботи потоків.

3.2.2 Redux

Майже всі, хто прийшов у Flutter зі світу фронтенду (зокрема, з React), знають про Flux-архітектуру і найпопулярнішу її реалізацію - Redux. Причини популярності прості:

- Централізованість - стан всього програми знаходиться в одному місці, що дозволяє зберігати його в будь-якому зручному для вас сховищі.
- Передбачуваність - не потрібно імперативно міняти залежні один від одного моделі, ми просто реагуємо на дії, які посилає нам система.
- Простота налагодження - завжди є можливість побачити повне дерево стану, а також можливість time-travel налагодження, коли ви можете послідовно пройтися по всіх змінах в стані і своєчасно знайти і виправити помилки.

- Гнучкість - існує велика кількість розширень на всі потреби програміста в управлінні станом.

Деякі Redux реалізації переносяться на інші платформи, і Flutter не став винятком. У dartpub є пакет redux, який може бути використовуватись як в інтернеті, так і на мобільних платформах з впровадженням додаткового пакета flutter_redux [2]. В Redux немає поділу на локальне і глобальне. Стан завжди глобальний - доступний кожному віджету і доступний до змін через екшени в будь-якому місці системи.

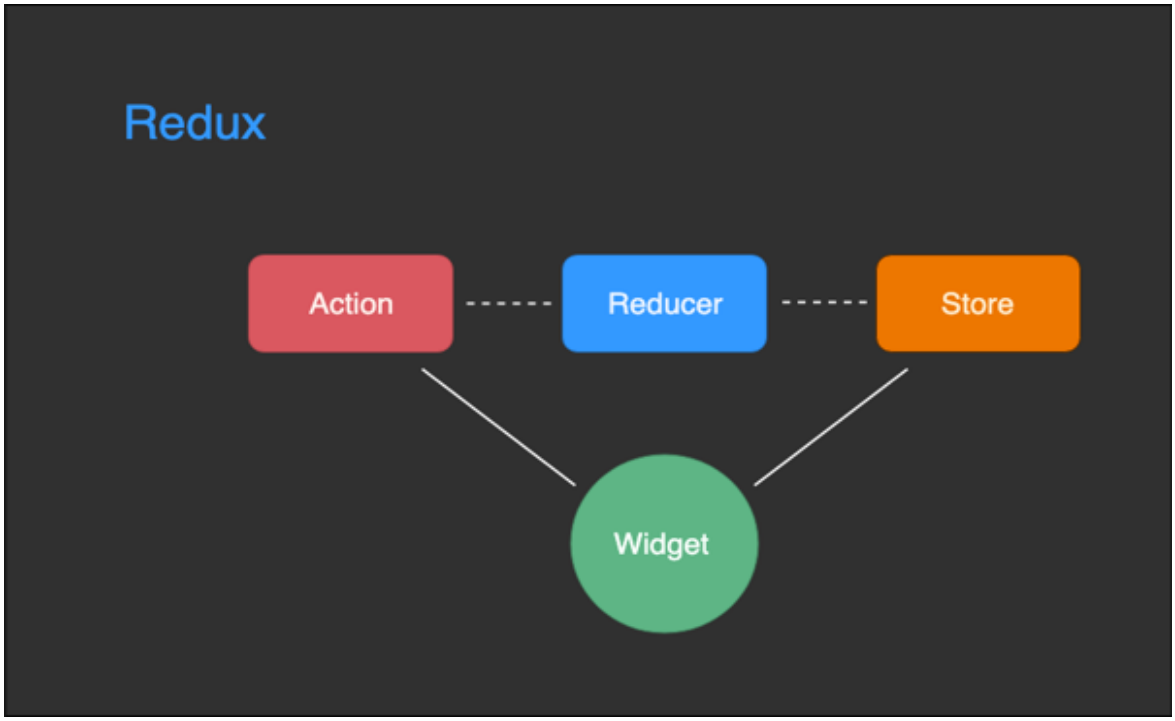

Рис 3.6 Схема архітектури Redux

Формально існують такі поняття:

- State - модель стану.
- Action - клас-ідентифікатор події, який зберігає в собі payload для передачі корисних властивостей.
- Reducer - обробник екшенів, має доступ до поточного стану і екшену, який чекає обробки.
- Dispatch - метод виклику екшену, який обробляється одним з редюсерів.

- Store - дерево стану програми, комбінує в собі всі редюсери, які ми визначили в додатку.

Переваги і недоліки

Працювати з Redux досить зручно завдяки супроводжуючим бібліотекам, що розвиваються:

- flutter_redux_dev_tools - налагодження і time-travel debug;
- redux_thunk - робота з сайд-ефектами за допомогою thunk;
- redux_epics - робота з сайд-ефектами за допомогою епіків, які базуються на потоках;
- redux_logging – логування станів та/або екшенів;
- redux_persist_flutter - збереження стану в постійному сховищі.

Однак є кілька недоліків:

- Кожен віджет може отримати доступ до всього стану додатку, що здатне легко порушити принцип єдиної відповідальності;
- Локальний стан віджетів зберігається в глобальному дереві стану, що істотно збільшує його розміри;

3.2.3 Висновок та вибір архітектури

Очевидний вибір для даної програми – BloC архітектура що дозволить відокремити логіку від UI, а також налагодити процес співпраці з TMDB API – який і є основним місцем з якого користувачі отримуватимуть інформацію про фільми. Оскільки архітектура BLoC є модульною – всі процеси, що пов’язані із роботою з віддаленим сервером будуть винесені в окрему бібліотеку. Приклади коду основних елементів даної архітектури продемонстровані у додатках А- Ґ.

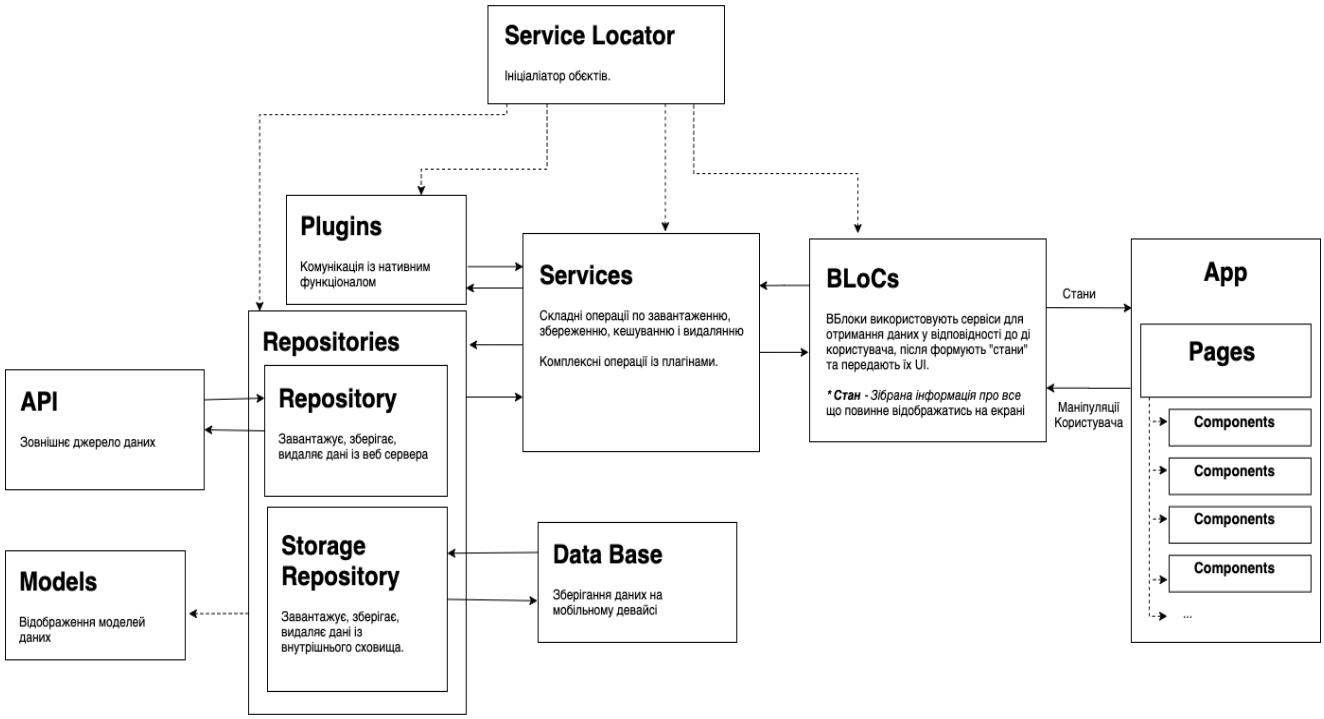

Рис 3.7 Детальна BLoC архітектура

РОЗДІЛ 4
ПРОЕКТУВАННЯ КОМПОНЕНТІВ ТА СИНТЕЗ СИСТЕМИ

4.1 Моделювання системи

Будь-яка програма може бути успішною на ринку, тільки якщо вона є високоякісною, вчасно розробленою і відповідає вимогам користувачів. Для цього необхідно постійно контактувати з користувачем, з'ясовуючи реальні вимоги до створюваної системи.

Чим більша і складніша система, тим більшого значення набуває моделювання при її розробленні, без моделі складну систему неможливо сприйняти як одне ціле. Людське сприйняття складних сутностей є обмеженим. Моделювання системи чи об'єкта дозволяє звузити проблему, зосередивши увагу в кожен момент тільки на певних її аспектах, що відповідає принципу "розділяй і володій". Складне завдання завжди легше вирішити, якщо розбити його на менші. Моделювання підсилює можливості людського інтелекту. Правильно обрана модель дозволяє створювати проекти на вищих рівнях абстракції.

Вибір моделі впливає на підхід до розв'язання проблеми і на те, як буде виглядати рішення. Правильно вибрана модель висвітить найпідступніші проблеми розроблення й дозволить проникнути в саму суть завдання, що при іншому підході було б просто неможливо. Неправильна модель може завести розробника у безвихідь, оскільки основна увага буде приділена несуттєвим питанням.

Для додатку пошуку фільмів було створено UML діаграми прецедентів та активностей, діаграма потоків даних, діаграми IDEF0, класів та сховища даних.

4.2 UML діаграми

Діаграма прецедентів

Дана діаграма прецедентів(Рис 4.1) дозволяє уявити можливості самої програми, екрани та послідовності виконання дій для переходу до потрібного елементу системи. Цієї діаграми достатньо для загального розуміння можливостей програми.

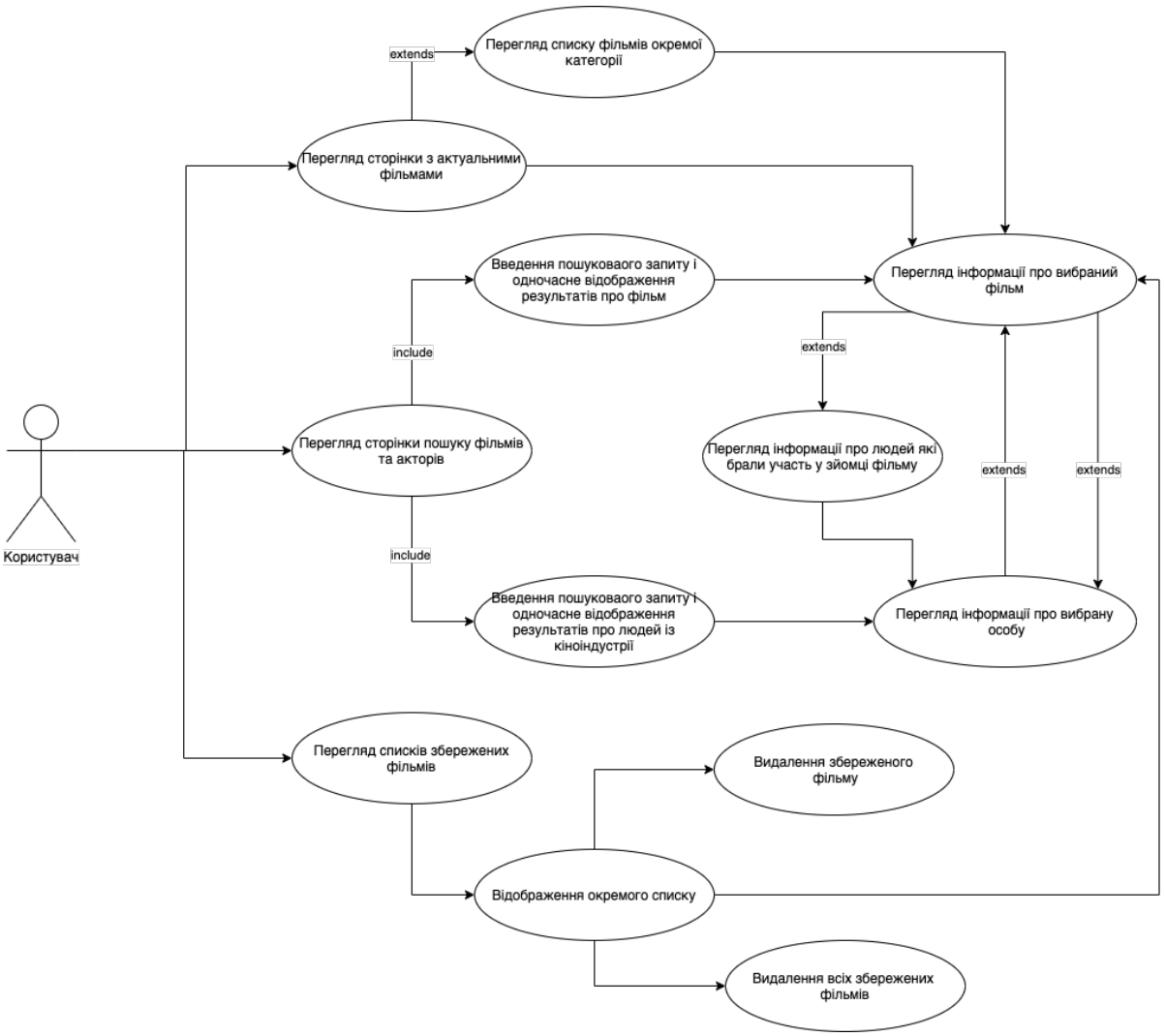

Рис 4.1 Діаграма прецедентів

Зокрема завдяки даному відображенню подій та послідовності переходів розробник спокійно може визначити як і з яких сторінок користувачу прийдеться переходити, дизайнер зможе відобразити всі скріни програми з інтерактивними переходами у призначених для цього програмах.

Діаграма активностей

Суть діаграми активностей в тому, щоб розкрити детальні дії користувача для досягнення певного результату в системі. На рисунку 4.2 можна побачити детальні інструкції щоб здійснити пошук фільму або людину із кінематографу.

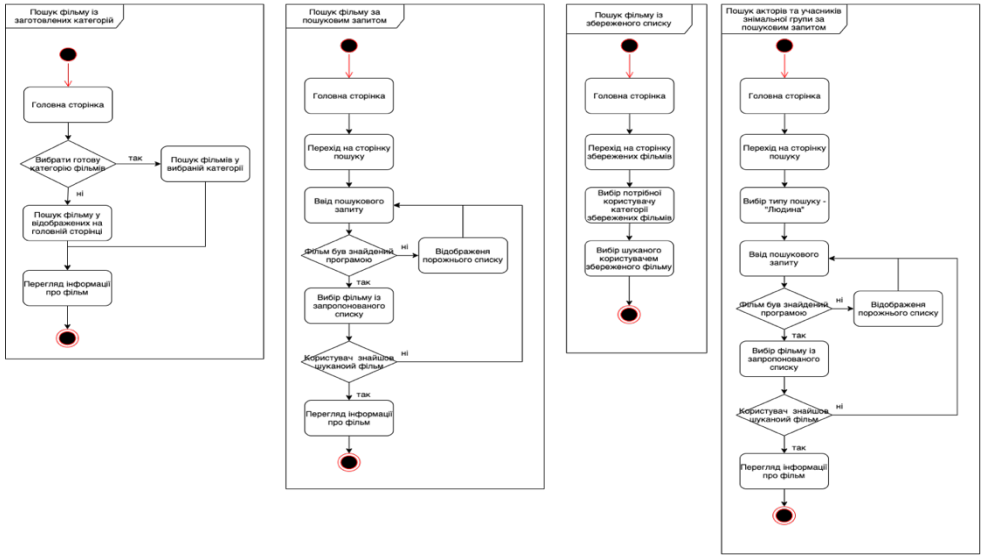

Рис 4.2 Діаграма активностей (пошуку кінострічок та кіно-персоналу)

На рисунку 4.3 зображені користувацькі дії для дій із списками фільмами, зокрема додавання кінострічки, видалення стрічки і видалення всіх стрічок до списку.

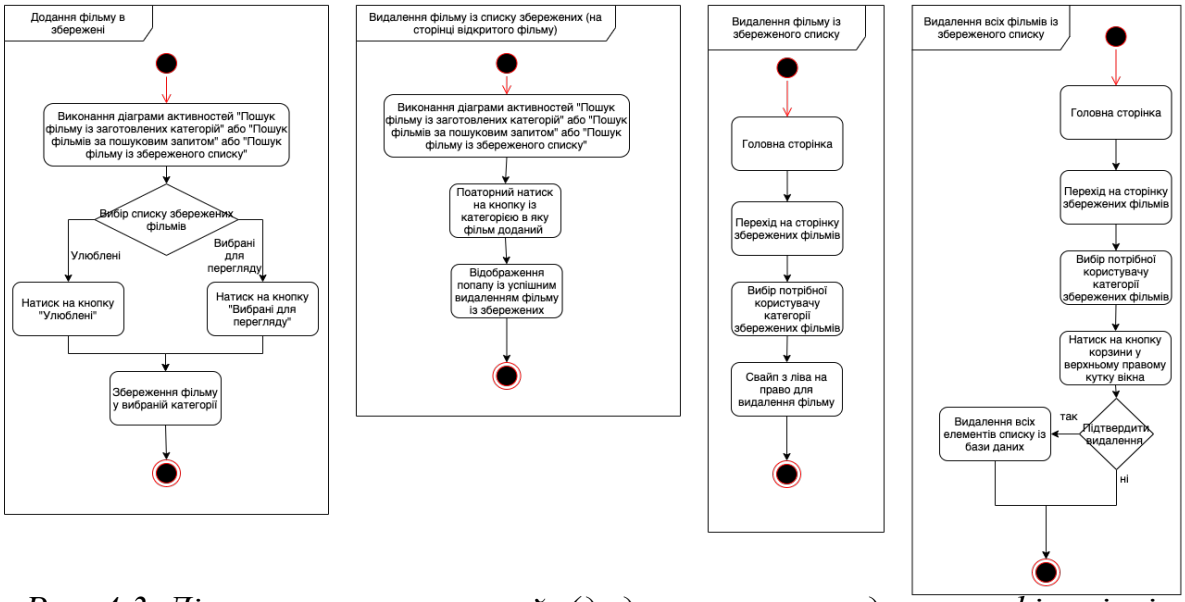

Рис 4.3 Діаграма активностей (додавання та видалення фільмів із

Для того щоб переглянути трейлер фільму або ж знайти інформацію про кіноперсонал вибраного фільму користувачу потрібно виконати дії на рисунку 4.4.

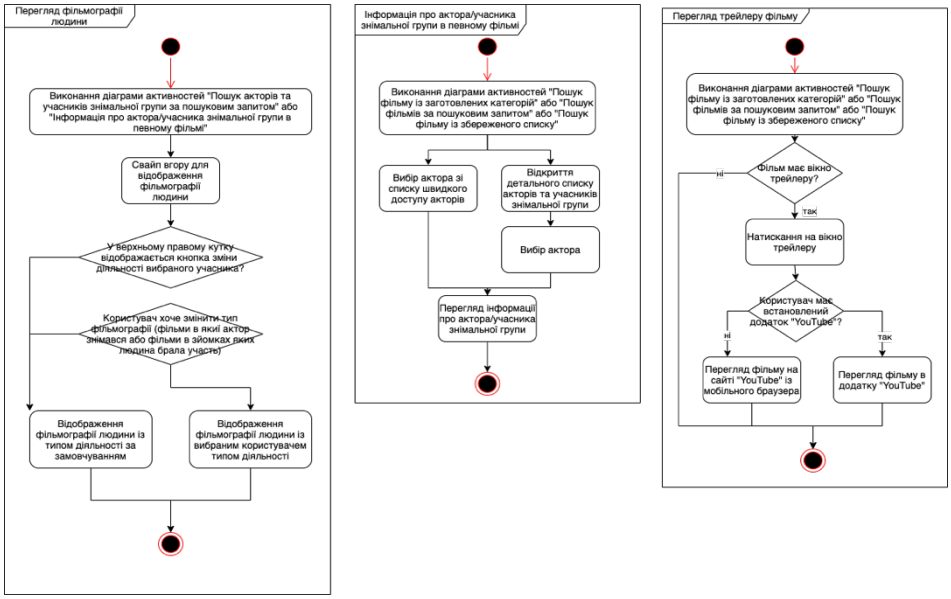

Рис 4.4 Діаграма активностей (перегляд трейлеру та інформації про кіноперсонал)

Діаграма потоків даних

Діаграма потоків даних(Рис 4.5) дає зрозуміти як дані з'являються та існують в програмі.

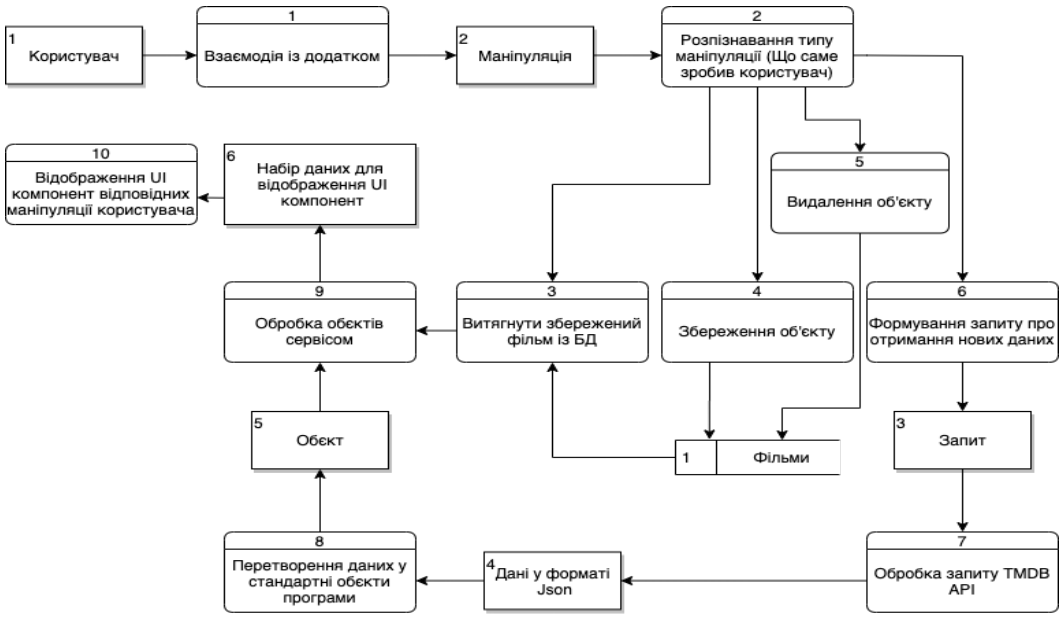

Рис 4.5 Діаграма потоків даних

Дана діаграма відображає взаємодію системи із зовнішніми модулями.

4.3 IDEF0 діаграми

IDEF0 діаграми дозволяють розробнику побачити систему від найзагальнішого до найдетальнішого елементу системи. Побудова такої діаграми відображає всі зв’язки і правила що впливають на систему, всі залежності, які вона має, всі вхідні та вихідні результати. IDEF0 дає змогу побачити загальний рівень який, як правило, складається із єдиного елементу й відображає основні впливи та дані, і заглибитись в кожен з рівнів з усе більшим розбиттям на простіші операції, таким чином розробнику ПЗ дуже зручно побудувати зв’язки, абстракції та правила взаємодії в коді. Загалом діаграма може мати дуже велику кількість рівнів декомпозиції, залежно від складності програми і затребуваної детальності опису системи.

На рисунку 4.6 зображений найвищий рівень IDEF0 діаграми і втілює в собі всю систему, її зв’язки, залежності та правила яким вона підпорядковується.

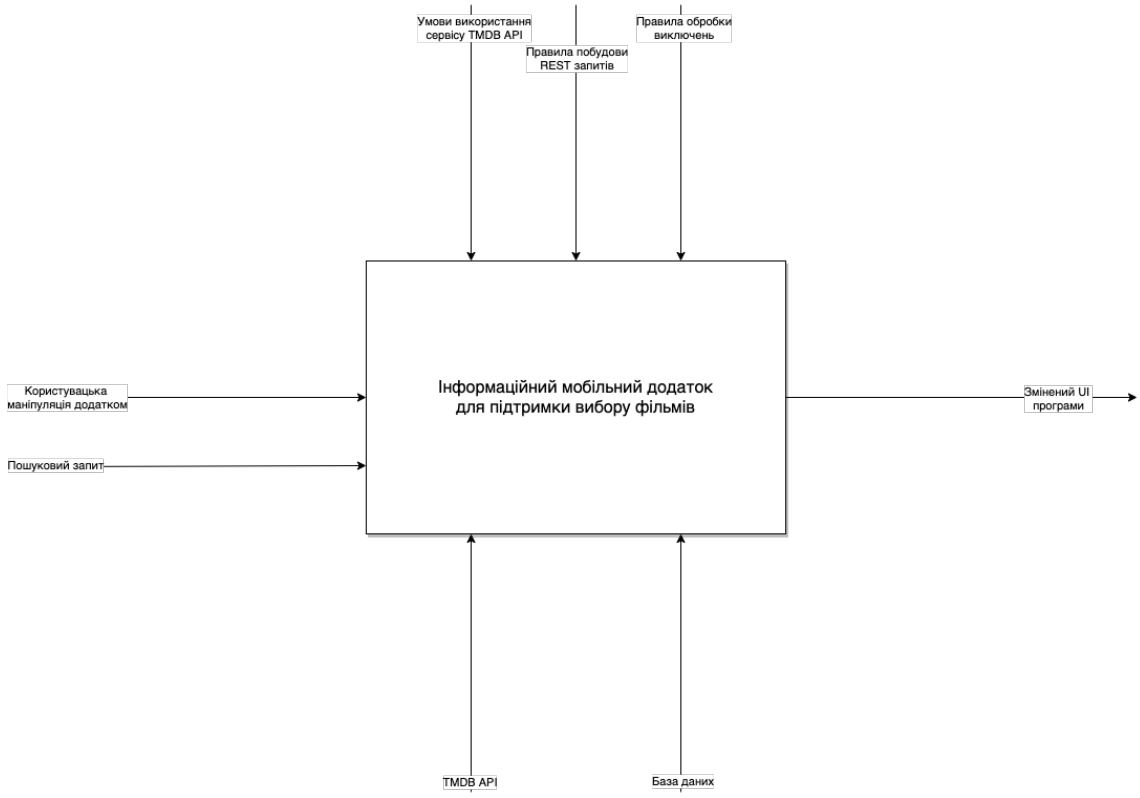

Рис 4.6 Діаграма IDEF0 – 1ий рівень декомпозиції

Рисунок 4.7 ілюструє всю систему розбиту на найосновніші пункти і взаємозв’язки між елементами правилами та залежностями.

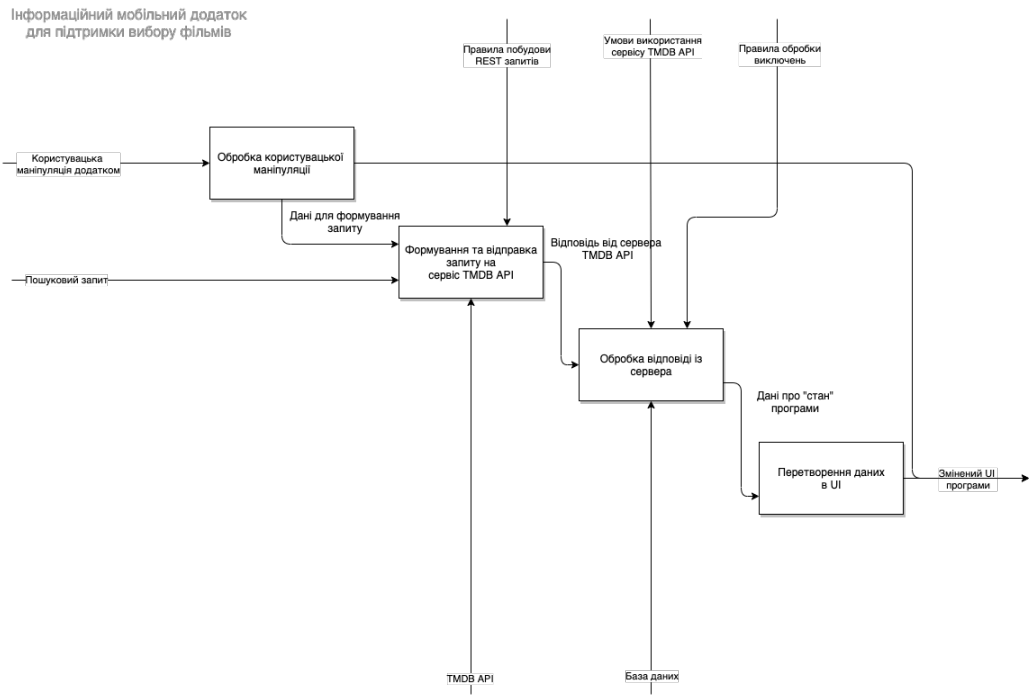

Рис 4.7 Діаграма IDEF0 – 2ий рівень декомпозиції

Те як формується запит на зовнішній сервер і процес його обробки можна побачити на рисунку 4.8.

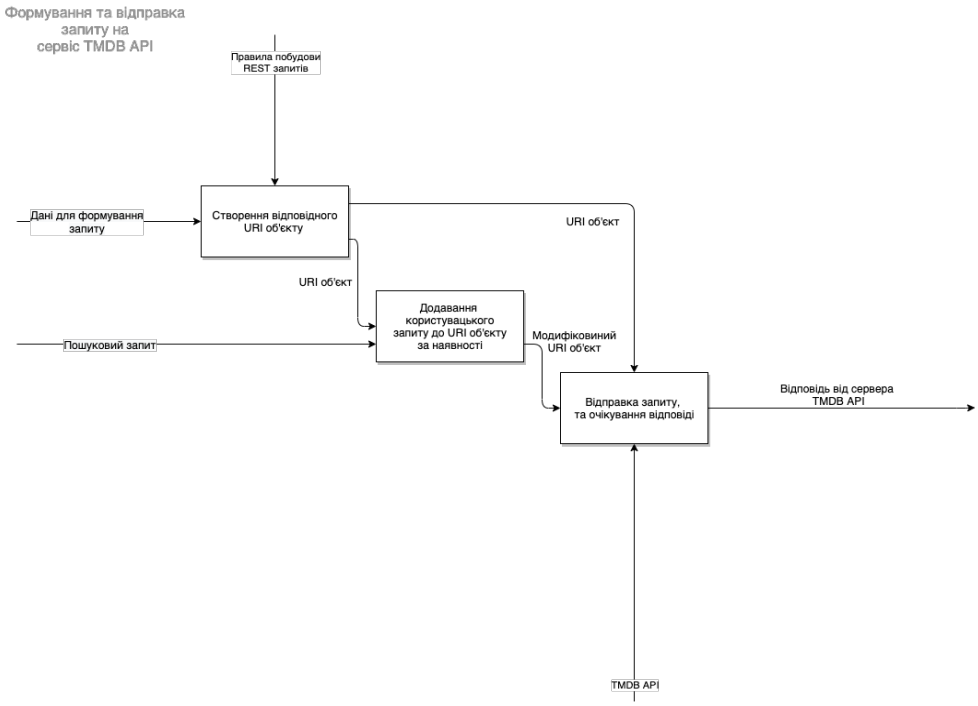

Рис 4.8 Діаграма IDEF0 – 3ій рівень декомпозиції
Активності “Формування та відправка запиту на сервіс

Алгоритм обробки даних отриманих із віддаленого сервера та перетворення їх в об’єкти, що використовуються у системі зображено на рисунку 4.9.

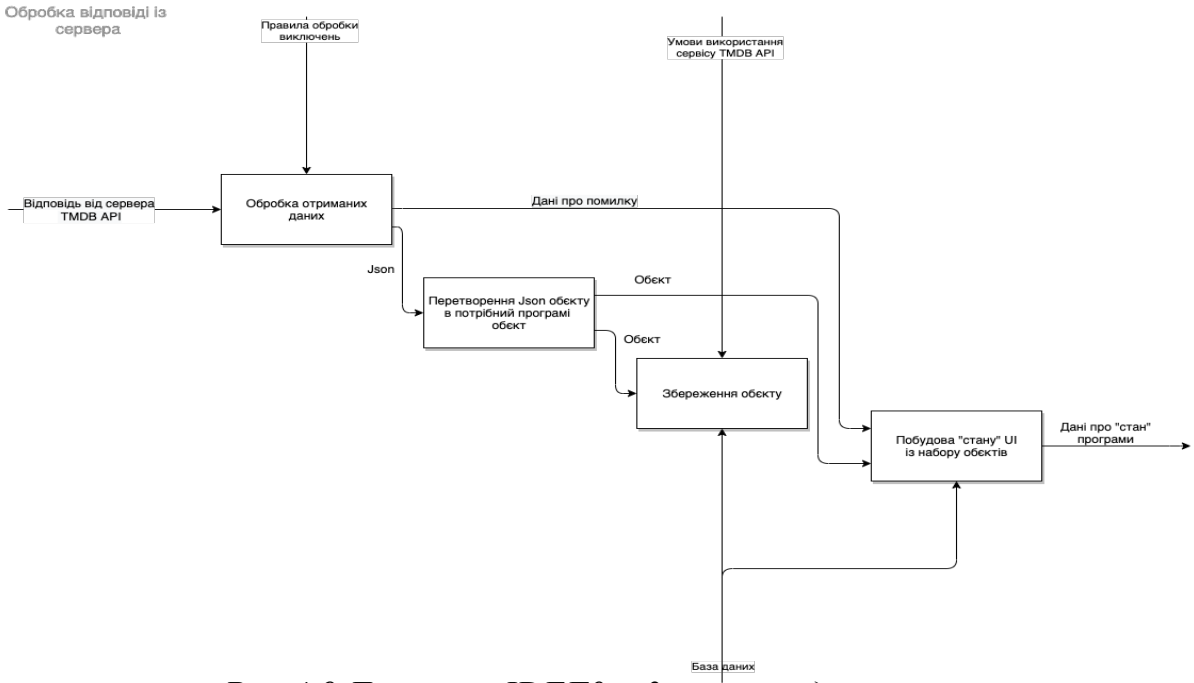

Рис 4.9 Діаграма IDEF0 – 3ій рівень декомпозиції Активності “Обробка відповіді із сервера”

Зокрема на діаграмі 4го рівня декомпозиції(Рис 4.10) відображено алгоритм обробки самих даних, до того як вони будуть перетворені в об’єкт.

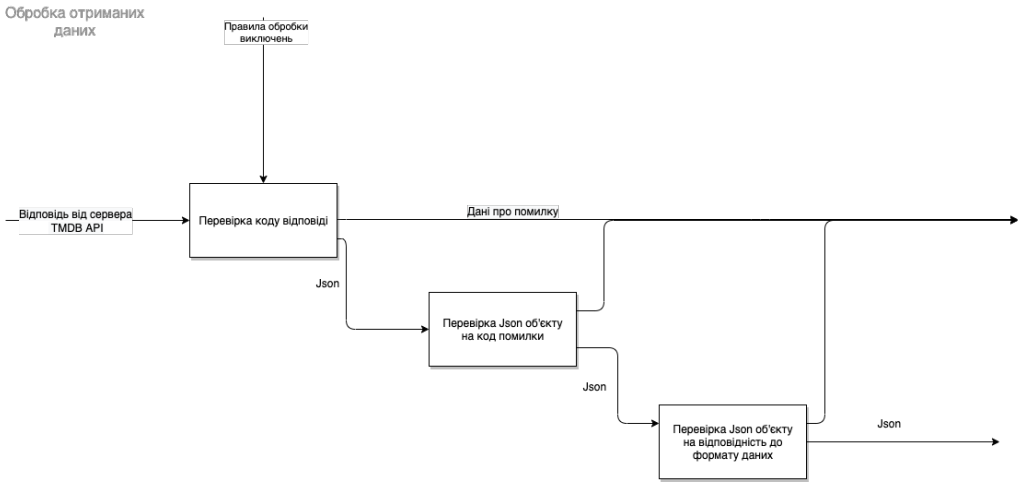

Рис 4.10 Діаграма IDEF0 – 4ій рівень декомпозиції Активності “Обробка отриманих даних”

Очевидно, що діаграма IDEF0 дає дуже зручний спосіб відображення системи як в загальному вигляді так і з описом найдрібніших деталей.

4.4 Діаграма класів та сховища даних

Створена діаграма класів(Рис 4.11) відображає сутності з якими працює система та інформацію яку користувач може побачити у додатку.

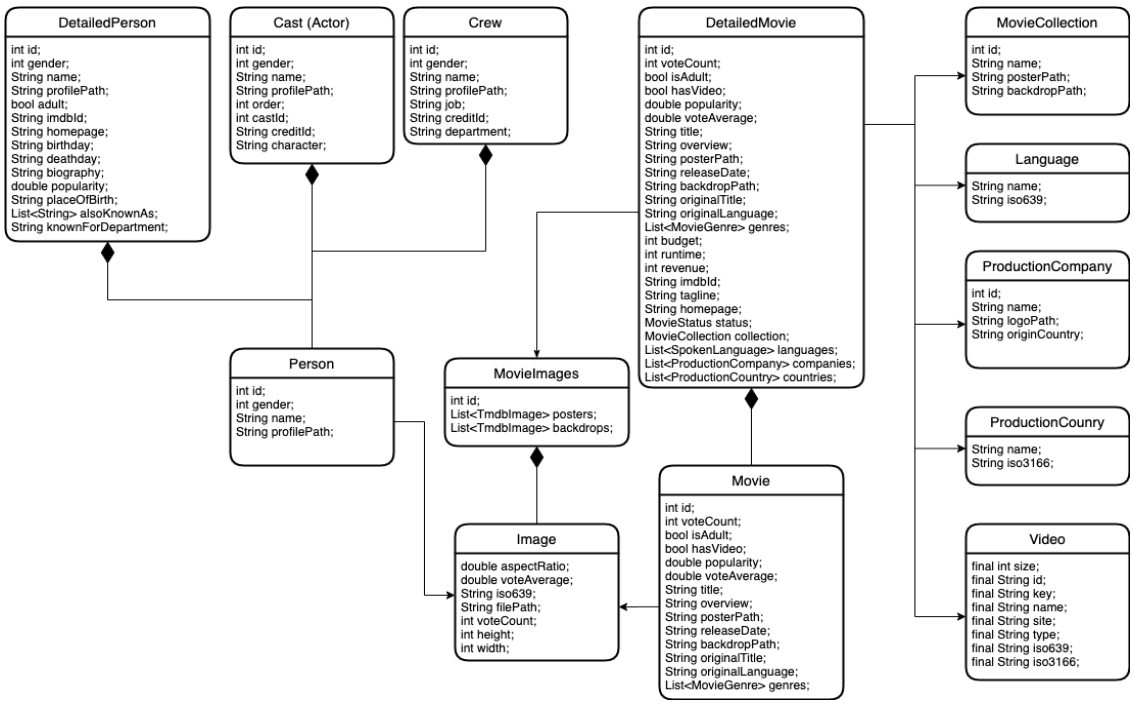

Рис 4.11 Діаграма класів

Схема бази даних складається всього із двох сутностей(Рис 4.12).

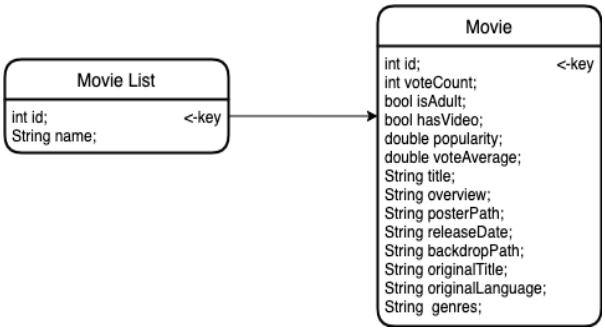

Рис 4.12 Схема бази даних

Оскільки дана система передбачає великий потік відображеної інформації за короткий час і інформація постійно оновлюватиметься сенсу в її зберіганні немає, а також по умовах користування сервісом TMDB API не можна зберігати певні дані довше ніж на певний термін. Тому в сховищі зберігаються тільки дані про збережені користувачем стрічки.

РОЗДІЛ 5
РОЗРОБКА ПРОГРАМНОГО РІШЕННЯ

5.1 Опис використаних сторонніх бібліотек

Для розробки програми пошуку фільмів було використано кілька сторонніх бібліотек, щоб спростити сам процес розробки і не “винаходити колесо” знову [20].

Provider – це пакет, написаний в 2018 році Ремі Руслі. Він дозволяє помістити будь-який об'єкт, в дерево віджетів і відкрити до нього доступ для будь-якого іншого елемента (нащадка). Також Provider допомагає керувати часом життя станів програми, ініціалізувавши їх з даними і виконуючи очистку після того, як вони будуть видалені з дерева віджетів. Provider ідеально підходить для реалізації компонентів BLoC архітектури або може служити основою для інших рішень в управління станом. Дана бібліотека також може просто застосовуватися для впровадження залежностей - термін, що має на увазі передачу даних у віджети таким чином, який дозволяє послабити зв'язаність і полегшити тестування коду. Нарешті, Provider може працювати з набором спеціалізованих класів, завдяки яким використовувати його ще зручніше.

Url launcher – бібліотека яка призначена для виконання різноманітних завдань у зв’язку з іншими програмами, встановленими на телефоні користувача. Завдяки цьому пакету можна без труднощів створити заготовку імейлу і відкрити поштовий клієнт користувача із заповненими даними про тему, отримувача і тд... Можна відкривати різні веб посилання, які маючи аналог додатку в смартфоні користувача, відкриються саме через додаток а не через веб браузер. В даному додатку бібліотека використовується для відкриття трейлерів фільмів в додатку YouTube або ж у веб браузері.

Firebase Crashlytics – пакет що допомагає відстежувати помилки в програмі заснований на однойменному сервісі від Google.

Carousel slider – UI бібліотека, що допомагає в створені красивого ефекту прокрутки власних компонент інтерфейсу. Дана бібліотека має метод *CarouselSlider.builder* що забезпечує ефективність відображення інформації і економить оперативну пам'ять смартфону. Зокрема Carousel slider може контролюватись спеціальним класом Carousel controller, що позволяє гнучко налаштовувати специфіку відображення елементів. В додатку цю бібліотеку в дії можна побачити на головному екрані в категорії популярних фільмів.

Equatable – дана бібліотека сильно спрощує життя для розробників що працюють із мовою програмування Dart. В даній мові програмування об'єкти порівнюються не тільки за типом і параметрами а й за хеш кодом об'єкту. Така специфіка мови призводить до того що два об'єкти одного класу маючи однакові параметри будуть рівні тільки у випадку якщо це один і той самий об'єкт. Ця бібліотека перевизначає оператор рівності для об'єкту і дозволяє визначити параметри для порівнювання, що вирішує вищезгадану проблему.

Flutter localization – це пакет, що дозволяє тримати всі тексти, які використовуються в програмі, в єдиному місці і, залежно від локалізації програми, перевизначити текст, що відображається на екрані користувача, під різні мови. Даний пакет дозволить в майбутньому додати підтримку різних мов для додатку.

http – ця бібліотека містить набір функцій та класів високого рівня, які полегшують роботу із HTTP-ресурсами. Він не залежить від платформи і може використовуватися як в командному рядку, так і в браузері. В даному програмному рішенні на основі цього пакету були побудовані класи-репозиторії для витягування даних із TMDB API.

5.2 Опис розроблених програмних модулів

Додаток використовує TMDB API для отримання інформації про акторів та кінострічки. Оскільки для того щоб побудувати якісну та швидку систему отримування даних із сервера потрібно використовувати окрему бібліотеку, що буде займатись тільки операціями спілкування із серверною частиною, перетворювати JSON стрічки у визначені об’єкти та виконувати різноманітні операції із отриманими даними.

У спеціалізованому збірнику бібліотек для мови Dart вдалось знайти кілька реалізацій потрібного функціоналу, проте на мою думку ці бібліотеки не покривали увесь функціонал що використовується у програмному рішенні, не були засновані на принципі Dependency Injection, що призведе до складнощів у тестуванні програми і не надавали можливість відстежувати помилки та неправильні дані, які теоретично можуть прийти зі сторони серверу. Саме тому було прийнято рішення розробити власну бібліотеку для такого функціоналу.

Створений пакет має таку структуру:

- base – папка в якій знаходяться в основному абстрактні класи для полегшення і уніфікації створення інших класів.
- models – папка в якій зберігаються всі моделі даних що використовуються у бібліотеці.
- repositories – папка в якій розташовані всі функціональні класи для витягування та перетворення об’єктів із мережі. Також для збереження об’єктів в локальному сховищі даних.
- services – папка для класів що працюють із складнішими операціями, такими як пагінація, об’єднання інформації із кількох об’єктів в один, відправка веб запитів та відслідковування похибок.

5.3 Розробка та опис інтерфейсу користувача

Створення якісного інтерфейсу вимагає значно більшого, ніж просто і дотримання деяких інструкцій. Воно припускає реалізацію принципу "інтереси користувача вище усього" і відповідну методологію розробки всього програмного продукту. При розробці мобільного додатку я керувався принципами простоти, зручності використання та природного інтерфейсу.

Інтерфейс повинний бути простим. При цьому мається на увазі не зменшення кількості функціоналу, а забезпечення легкості у його вивченні та використанні. Крім того, інтерфейс повинний надавати доступ до всього переліку можливостей, передбачених додатком. Розробка ефективного інтерфейсу покликана збалансувати такі цілі.

Природний інтерфейс - такий, котрий не змушує користувача суттєво змінювати звичні для нього способи рішення задачі. Це, зокрема , означає, що повідомлення і результати, видавані додатком, не повинні вимагати додаткових пояснень[7].

Один з можливих шляхів підтримки простоти - представлення на екрані інформації, мінімально необхідної для виконання користувачем чергового кроку завдання. Зокрема, потрібно уникати багатослівних назв, чи назв із надто загальним значенням. Непродумані чи надлишкові фрази заважають користувачу поглинати інформацію.

Естетична привабливість - проектування візуальних компонентів є найважливішою складовою розробки програмного інтерфейсу. Коректне візуальне представлення використовуваних об'єктів забезпечує передачу дуже важливої додаткової інформації про і взаємодію різних об'єктів.

Огляд інтерфейсу користувача

При запуску застосунку, користувач побачить перед собою, екран з актуальними категоріями кінострічок(Рис 5.1). З даного екрану він відразу ж може підшукати собі фільм. Також натиском на категорію користувач має можливість переглянути список фільмів що входять у цю категорію, а натиском на постер перейти на екран детальної інформації про стрічку.

При натисненні на заголовок категорії, користувач відкриє екран із фільмами які входять в ту категорію(Рис 5.2). На даному екрані він має можливість детальніше оглянути відповідні кінострічки.

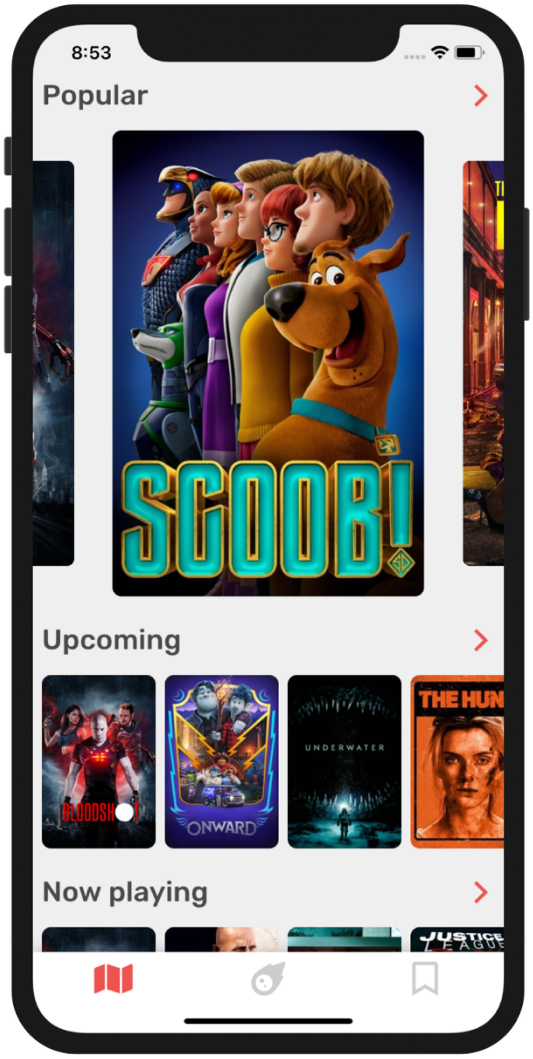

Рис. 5.1 Головний екран програми

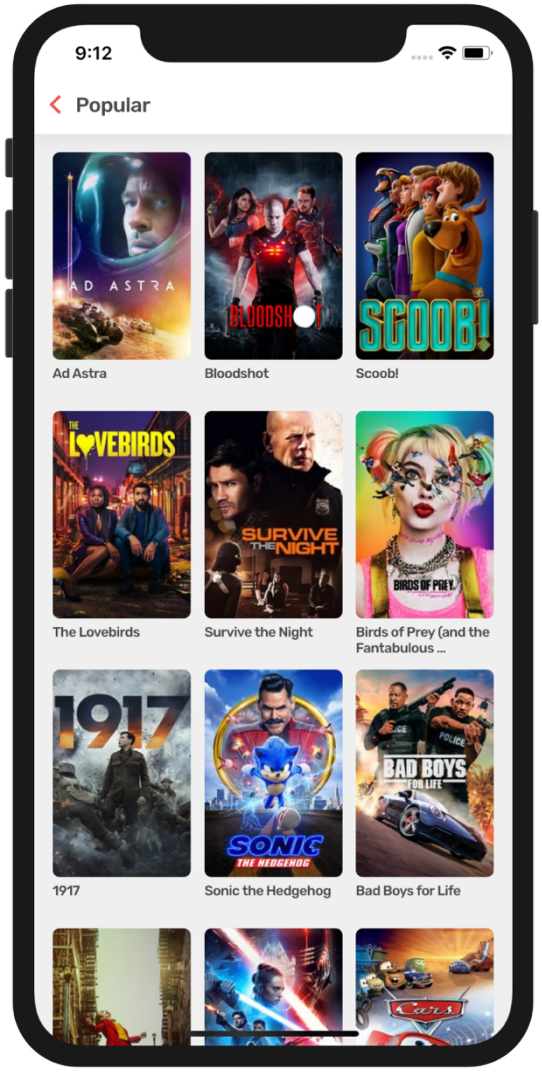

Рис 5.2 Список фільмів категорії

При натисненні на постер фільму, користувач відкриє екран із деталями про вибраний фільм, де він зможе ознайомитись із описом, трейлером, акторським складом та іншою інформацією(Рис 5.3). Зокрема по кліку на зображення відео відкриється трейлер даної стрічки. При натисненні на кнопку із іконкою серця можна додати фільм у список улюблених, при повторному натисненні – забрати. Аналогічно із інша кнопка добавляє фільм у список переглянути пізніше(Рис 5.4).

Рис 5.3 Детальний опис фільму

Рис 5.4 Додавання фільму у список

При натисненні на заголовок “Actors & Crew” відкриється екран із списком акторів та знімального персоналу вибраного фільму(Рис 5.5).

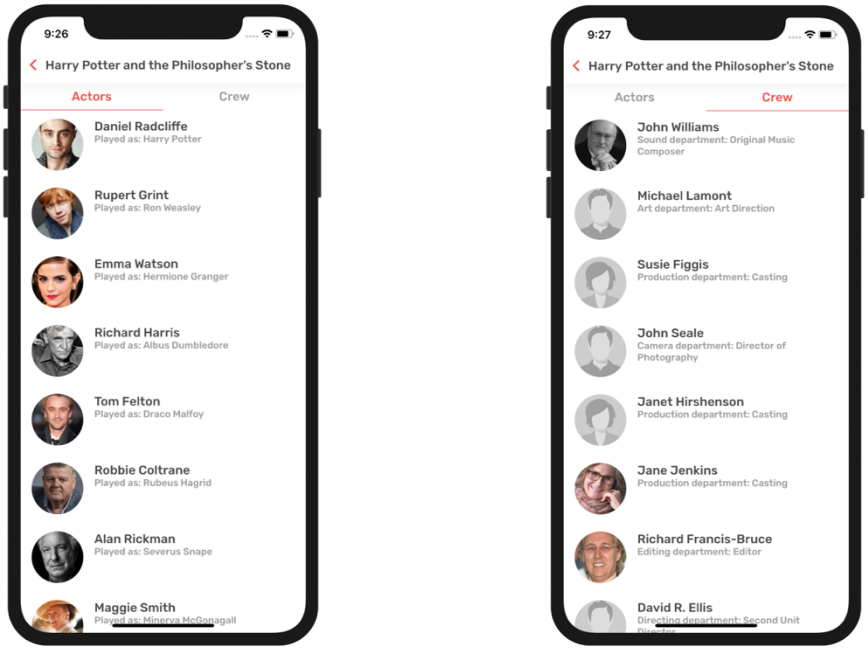

Рис 5.5 Списки акторів фільму

При натисненні на іконку персони, користувач відкриє екран із деталями про вибрану людину де він зможе ознайомитись із його фільмографією як актора/акторки, або як людини що приймала участь у зйомці фільму(Рис 5.6).

Рис 5.6 Сторінка деталей про актора

По свайпу вгору відкривається інформація про фільмографію людини, де можна знайти фільми в яких вона грала як актор та роль яку вона грала, або якщо це людина що працювала над фільмом – відділ в якому вона працювала та посаду. Якщо дана людина і знімалась у фільмі і приймала участь у зніманні фільмів у правому верхньому куті екрану буде кнопка, що позволить переключатись між фільмами в яких вона була актором і стрічками над якими вона працювала.

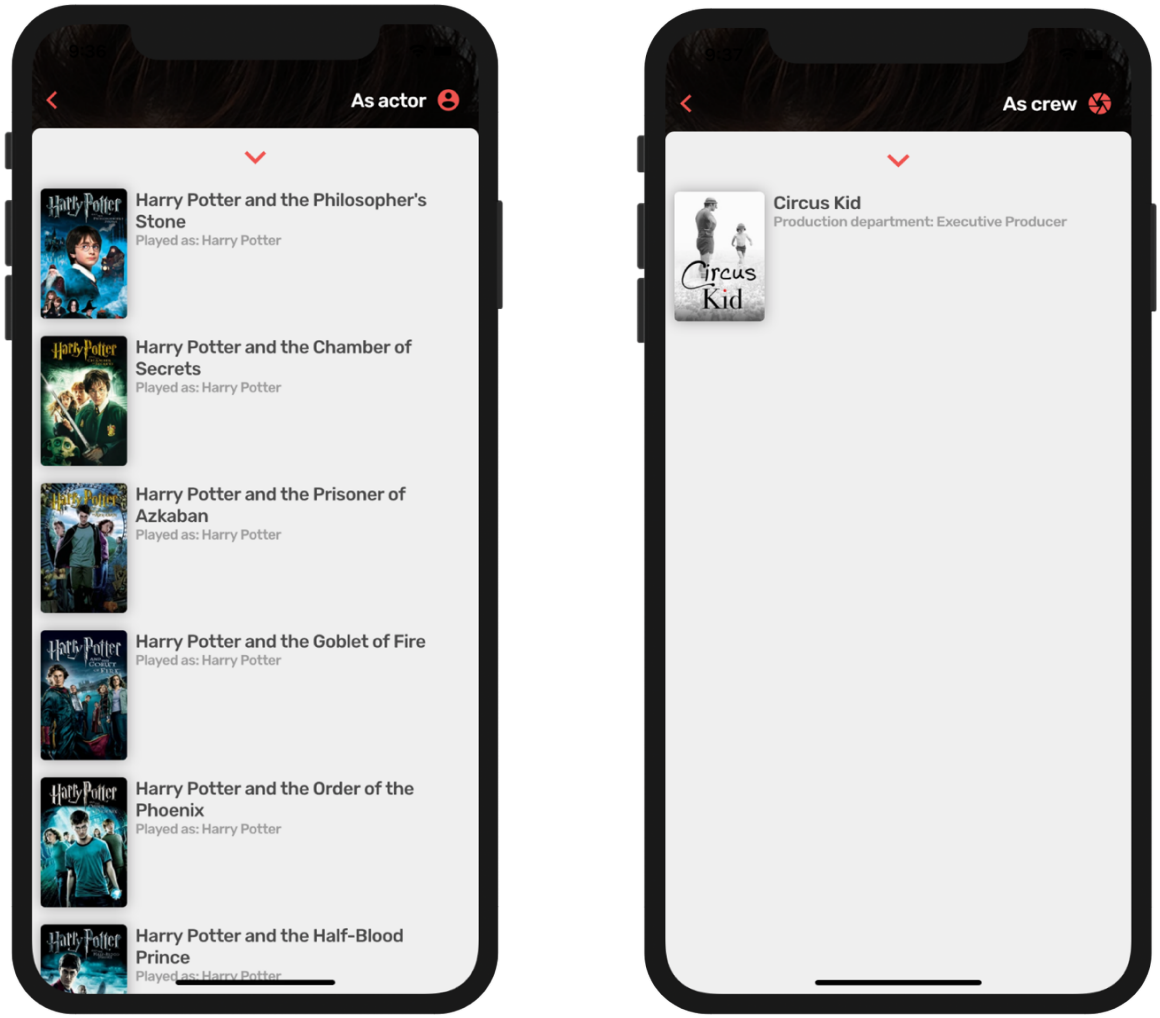

Рис 5.7 Фільмографія актора

У середній вкладці користувач може знайти меню пошуку, що позволяє користувачу легко відшукати потрібну йому інформацію(Рис 5.8). При натиснені на крайній лівий значок у полі пошуку, можна поміняти тип пошуку фільм або людина. При натисненні на значок у формі букви ‘x’ користувач може стерти його введений текс у полі.

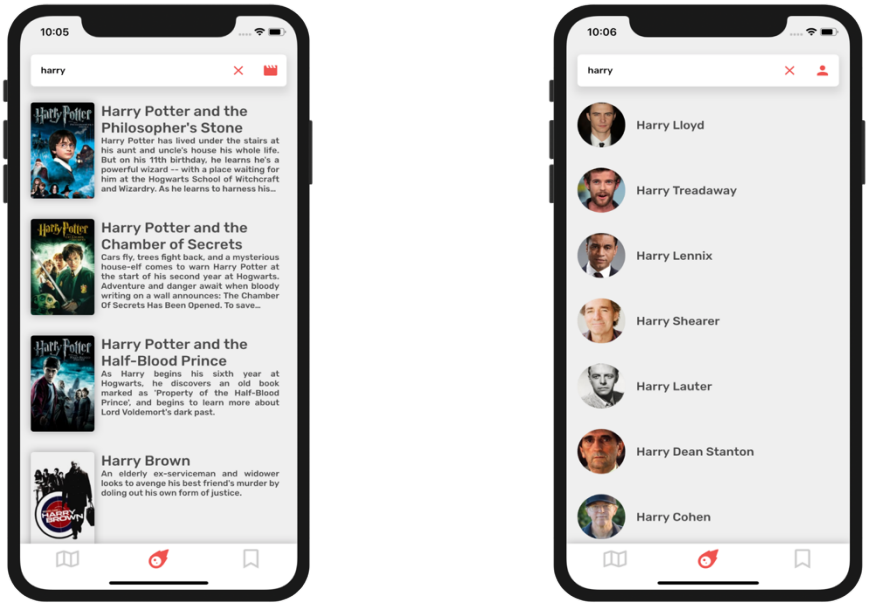

Рис 5.8 Меню пошуку

На вкладці зі знаком прапорця користувач знайде списки збережених фільмів а також актуальну версію програми.(Рис 5.9).

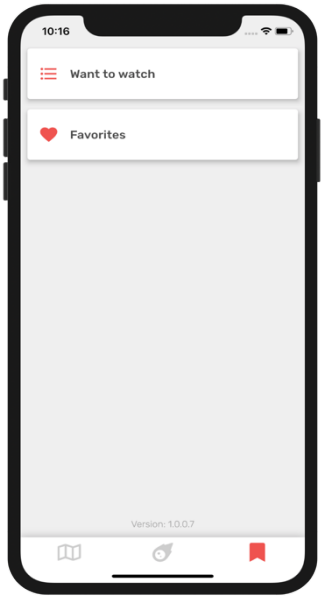

Рис 5.9 Меню списків збережених фільмів

Натиснувши на список користувач зможе побачити збережені фільми, які він додав до нього, зокрема свайпнувши з ліва на право він зможе видалити фільм із списку або ж натиснувши на іконку корзини згори він зможе очистити список, після підтвердження того, що він дійсно хоче це зробити.

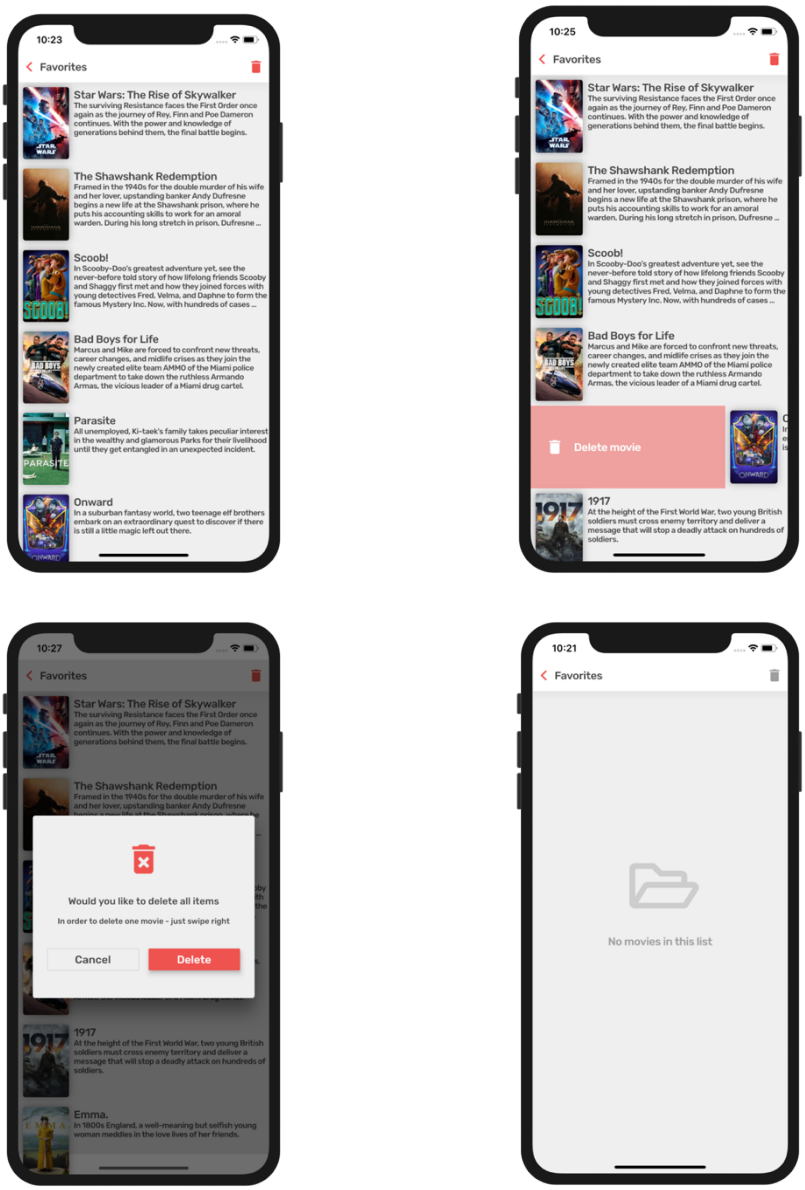

Рис 5.10 Меню списку фільмів

Отже додаток отримав достатньо органічний інтерфейс із інтуїтивно зрозумілим управлінням – це означає що практично будь-яка людина з зможе користуватись застосунком.

РОЗДІЛ 6
ЕКСПЕРИМЕНТАЛЬНА ЧАСТИНА

6.1 Встановлення додатку

Для встановлення додатку потрібно завантажити виконуваний файл із розширенням .apk на ваш Android смартфон. Є можливість розмістити додаток на сторінках найбільших програмних магазинів PlayMarket та AppStore, після чого користувачі зможуть легко завантажити додаток, проте ці опції платні. Зокрема через політику компанії Apple єдиний спосіб встановити дану програму на свій смартфон – через магазин програм AppStore.

Вимоги для встановлення та користування системою:

- Смартфони Apple IPhone 5s і новіші під управліннм IOS.
- ОС Android 5.0 Lollipop і вище або IOS 10 і вище.
- Доступ до Інтернету.
- Швидкість Інтернет-з’єднання 512 Кбіт/сек і вище.

6.2 Перенесення проекту на іншу машину

Для розробки мобільних додатків за допомогою Flutter SDK потрібно встановити одну із підтримуваних IDE, що мають велику кількість плагінів, які позволяють зручно розробляти програмний продукт.

На сьогоднішній день є 2 середовища розробки, що підтримують дану технологію розробки на високому рівні: VSCode, Android Studio.

Для встановлення VSCode потрібно:

1. Перейти за посиланням https://code.visualstudio.com/download і скачати пакет для вашої операційної системи (MacOS/Windows/Linux).
2. Подвійним натиском лівої кнопки миші відкрити установочний пакет та слідувати інструкціям по встановленню.

Оскільки ми розробляємо під мобільні платформи Android та IOS потрібно встановити програмне забезпечення для компілювання додатків під нативну платформу – компіляція під Android системи відбувається за допомогою Android SDK а під IOS за допомогою XCode.

Інструкція по встановленню Android SDK:

1. Якщо у вас встановлена система Android Studio – у вас уже автоматично було встановлено Android SDK і вам не потрібно виконувати наступні кроки.
2. Для завантаження окремого пакета інструментів розробки без IDE необхідно скористатися сайтом Android Developer https://developer.android.com/studio/index.html. Спускаємося до самого низу сторінки і знаходимо розділ “Command line tools only” - можна також скористатися пошуком по сторінці. Після скачування відповідного пакета для вашої операційної системи, досить розпакувати в будь-яке зручне для вас місце.
3. Для отримання доступу до всього функціоналу Android SDK досить запустити пакетний файл android що знаходиться у розпакованому архіві, якщо ви працюєте під Windows. Або ж в терміналі перейти в розпакований архів та виконати команду «android» в папці вашого SDK для запуску на Linux і MacOS. Вам буде запропоновано встановити інструменти для різних версій API, чим більше інструментів встановлено - тим краще. Завантаження пакетів може зайняти багато часу в залежності від вашої системи і швидкості інтернет-з'єднання.

Система XCode дає можливість розробляти програмні продукти під IOS та MacOS, проте для роботи із системою потрібно мати пристрій під управлінням MacOS, дане середовище тісно пов’язане із операційною самою системою і через політику компанії Apple не є можливим встановлення під інші системи.

Інструкція по встановленню XCode:

1. Відкрийте стандартний магазин програм “AppStore”.
2. Введіть в пошуку XCode.
3. Натисніть на кнопку “Інсталювати”.

Для встановлення Flutter SDK потрібно:

1. Перейти за посиланням https://flutter.dev/docs/get-started/install і клікнути на кнопку із вашою операційною системою.
2. Скачати зіп архів.
3. Розпакувати його у зручному для вас місці.
4. Або замість попередніх кроків просто клонувати git репозиторій командою “git clone https://github.com/flutter/flutter.git -b stable”.
5. Для коректної та зручної роботи із Flutter потрібно додати його в список змінних середовища (PATH) із шляхом: “Шлях до папки в якому розпакований архів/flutter/bin”.
6. Запускаємо команду в будь-якій консолі команду “flutter doctor”, якщо якийсь компонент встановлено неправильно або ж щось не так на екрані консолі появляться надписи червоного кольору із причиною проблеми.

Коли всі підготовчі кроки для роботи проекту виконані потрібно завантажити сам проект. Сам проект знаходиться на віддаленому репозиторії сервісу GitHub. Щоб завантажити проект потрібно зробити кілька простих кроків:

1. Перейти за посиланням https://github.com/Kroha1999/movie_app
2. Під назвою сховища натисніть Клонувати або завантажити.
3. Щоб клонувати репозиторій за допомогою HTTPS, у розділі "Клонувати з HTTPS" скопіюйте посилання. Щоб клонувати сховище за допомогою ключа SSH, включаючи сертифікат, виданий органом сертифікації SSH вашої організації, натисніть

кнопку Використовувати SSH, а потім скопіюйте посилання із для копіювання проекту із SSH ключем.

4. Відкрийте термінал.
5. Змініть поточне робоче розміщення на місце, де ви хочете клонувати проект.
6. Введіть “git clone” та вставте URL, який ви скопіювали в пункті 3.
7. Нажміть кнопку “Enter” для виконання копіювання проекту

Для відкриття проекту:

1. Відкрийте систему розробки програмного забезпечення.
2. Виберіть пункт відкрити папку.
3. Відкрийте скопійований проект.
4. Можна працювати із проектом

Для запуску проекту:

1. На Андроїд в налаштуваннях системи знаходимо в меню системи, і нажимаємо на пункт “Номер збірки” доки не побачимо поп-ап із підтвердженням що ви розробник.
2. Переходимо в налаштування смартфону, пункт “Для розробників” і включаємо “Відладку по USB”.
3. Підключаємо мобільний девайс до комп’ютера за допомогою USB кабелю, якщо на телефоні з’являються вікна із запитом на дозволи - надаємо всі дозволи. Альтернативно можна використовувати емулятор мобільної операційної системи.
4. В консолі переходимо в папку із проектом і запускаємо команду “flutter pub get” для завантаження всіх сторонніх пакетів проекту.
5. Запускаємо проект за допомогою команди “flutter run”.
6. Проект після компіляції відобразиться на екрані смартфону.

6.3 Тестування

Тестування додатку - це важливий етап, на якому перевіряється поведінка програми на великій кількості вхідних даних, включаючи невірні, та перевірка поведінки додатку при неочікуваних даних із сервера.

Для тестування системи застосовувалося функціональне тестування, тобто тестування програмного забезпечення з метою перевірки реалізації функціональних вимог. Тестування додатку проводилося як вручну, так і з використанням модульних тестів.

Модульні тести - це, як правило, автоматизовані тести, написані та запущені розробниками програмного забезпечення, щоб переконатися, що розділ програми відповідає його дизайну та веде себе за призначенням].

Інструментальні модульні тести - тести, за допомогою яких тестується логіка, яка використовуються в програмі. Їх виконання проходить на фізичному пристрої або емуляторі[5].

Велика частина логіки додатка пов'язана зі взаємодією системи та зовнішніх сервісів, тому здебільшого були використані інструментальні модульні тести.

Для модульного тестування використовувалася бібліотека flutter_test. Бібліотека призначена для тестування програм на мові Dart і має велику кількість можливостей: поділ даних і логіки тесту, угруповання тестів, використання анотацій для опису тестів, оброблення винятків, що виникли в тесті, і т.д.

Оскільки в процесі реалізації програми було прийнято рішення самому створити бібліотеку для роботи із TMDB API, було написано велику кількість коду, в якому легко допустити помилки. У зв'язку з цим було вирішено приділити особливу увагу тестуванню логічних компонент, призначеного для роботи із сервером TMDB.

Таблиця 6.1

Опис проведених тестів роботи додатку із TMDB API

№	Аспект роботи	Дії	Результат	Тест успішно виконаний?
1	Отримання інформації про категорію фільмів (популярні, зараз у кіно…)	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри.	Отриманий JSON із списком відповідної категорії фільмів. У відповіді також додана сторінка пагінації якій відповідає запит та максимальна загальна кількість сторінок для даного запиту.	Так
2	Отримання інформації про конкретний фільм	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри та ID фільму.	Отриманий JSON із інформацією про заданий фільм.	Так
3	Отримання інформації про список особу	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри та ID особи.	Отриманий JSON із інформацією про шукану особу.	Так
4	Отримання інформації про людей які знімали(сь) у ньому	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри та ID фільму.	Отриманий JSON із інформацією про акторів (cast) та знімальну групу (crew).	Так
5	Отримання інформації про подібні фільми	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри та ID фільму.	Отриманий JSON із списком подібних фільмів, у відповіді також додана сторінка пагінації якій відповідає запит та максимальна загальна кількість сторінок для даного запиту.	Так

Продовження таблиці 6.1

6	Отримання інформації про трейлери конкретного фільму	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри та ID фільму.	Отриманий JSON із списком посилань на трейлери	Так
7	Отримання інформації про фільм за пошуковим запитом	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри та запит введений користувачем.	Отриманий JSON із списком фільмів що найбільш відповідають запиту користувача, у відповіді також додана сторінка пагінації якій відповідає запит та максимальна загальна кількість сторінок для даного запиту.	Так
8	Отримання інформації про актора за пошуковим запитом	Відправити GET запит на відповідний URL, із додаванням ключа доступу в параметри та запит введений користувачем.	Отриманий JSON із списком акторів що найбільш відповідають запиту користувача, у відповіді також додана сторінка пагінації якій відповідає запит та максимальна загальна кількість сторінок для даного запиту.	Так

Таблиця 6.2

Опис проведених ручних тестів додатку

№	**Аспект роботи**	**Дії**	**Результат**	**Тест успішно виконаний?**
1	Пошук фільму за користувацьким запитом	1. Перейти на сторінку пошуку. 2.Натиснути на поле пошуку та ввести запит “harry”. 3.Натиснути кнопку пошуку на клавіатурі.	Відбувається показ фільмів які містять в назві слово “harry” – в даному випадку перші три фільми були із серії “Harry Potter”.	Так

Продовження таблиці 6.2

2	Пошук людини за користувацьким запитом	1. Перейти на сторінку пошуку. 3. Збоку поля пошуку натиснути на кнопку переключення типу пошуку. 2.Натиснути на поле пошуку та ввести запит “bruce”. 3.Натиснути кнопку пошуку на клавіатурі.	Відбувається показ людей які мають в імені фразу “bruce” – в даному випадку перший актор був “Bruce Willis”.	Так
3	Збереження фільму в список улюблених	1. Відкрити будь-який фільм. 3. Натиснути на кнопку “додати в улюблені”. 2.Повернутись на головну сторінку та перейти на вкладку збережених фільмів. 3.Відкрити вкладку улюблені.	Відбувається збереження інформації про фільм в локальне сховище, і відображення фільму у списку улюблених.	Так
4	Видалення фільмів із списку улюблених	1. Відкрити вкладку збережених фільмів. 2. Свайпнути по фільму який потрібно видалити.	Відбувається видалення фільму із БД	Так
5	Видалення всіх фільмів із списку улюблених	1. Відкрити вкладку збережених фільмів. 2. Натиснути на кнопку корзини у верхньому правому куті програми. 3. Підтвердити свій намір видалити всі елементи списку.	Відбувається видалення всіх фільмів із списку із БД	Так
6	Робота при відсутності інтернету	1. Відкрити додаток. 2. Відключити інтернет на телефоні. 3. Попробувати відкрити будь який список на головній сторінці.	Відображається плейсхолдер який повідомляє користувача що потрібно підключитись до інтернету і кнопка “Try Again” яка дозволяє перезагрузити сторінку після перепідключення до мережі.	Так

Продовження таблиці 6.2

7	Відкриття трейлеру	1. Відкрити будь-який фільм, що має трейлер. 2. Натиснути на вікно трейлеру.	Користувач переспрямовується до додатку YouTube (за наявності), або в мобільний браузер на сторінку трейлеру для його перегляду.	Так
8	Перегляд фільмографії особи	1. Відкрити сторінку будь-якої особи. 2. Свайпнути вгору для перегляду фільмографії.	Користувачу відображається список фільмів особи. При наявності кнопки у верхньому правому кутку, можна переключитись на фільми в яких людина брала участь як знімальна група	Так

6.4 Крашлітика (Crashlytics)

В додатках існує висока ймовірність збою навіть після його ручного та автоматичного тестування. Це може бути пов'язано з такими факторами, як несумісність з різними рівнями API, розмірами екрану, пам'яттю телефону, доступністю апаратних датчиків і налаштуванням індивідуальних пристроїв. Крашлітика допомагає вам в діагностиці таких проблем.

Збої роблять користувачів сумними і сердитими. Вони швидш за все видалять додаток, якщо таке відбуватиметься часто. Потім погано оцінять застосунок на платформі з якої вони його завантажили, та залишать негативний відгук і що найгірше встановлять додаток конкурента. На вже переповненому і конкурентному ринку додатків, дуже важливо для успіху програми, щоб користувачі були задоволені.

Практично неможливо щоб в додатку не було збоїв, тому важливо мінімізувати кількість падінь додатку. Під час розробки та тестування часто можна й не зіткнутися із певними збоями, але при застосуванні крашлітики, коли користувачі будуть активно його використовувати, ви почнете бачити всі відстежені проблеми. Використання надійного і потужного інструменту

для створення звітів про помилки вкрай важлива для збору інформації про додатки, що дає уявлення про те, як можна виправити ту чи іншу проблему, щоб користувачі залишилися задоволені[8].

В дану мобільну програму я додав підтримку Firebase Crashlytics, вона відстежуватиме збої на різних пристроях користувачів та присилатиме детальну інформацію про них. Крім цього було імплементовано відстеження збоїв зі сторони TMDB API, оскільки цей сервіс є стороннім треба враховувати, що з часом дещо може помінятись і крашлітика зможе від разу попередити про зміни у вигляді відстежених неправильних результатів.

Можливості Firebase Crashlytics:

- Крашлітика зчитує інформацію про збої в смартфоні, отримує контекстну інформацію, підкреслює важливість того чи іншого збою та підраховує кількість подібних збоїв на різних девайсах, щоб можна було як найшвидше визначити першопричину.
- Сервіс крашлітики пропонує корисні поради, які висвітлюють поширені проблеми зі стабільністю та надає ресурси, що полегшують пошук, усунення та вирішення проблем.
- Firebase Crashlytics будує детальні графіки про помилки їх частоту та критичність. За допомогою цього інструменту можна легко визначити яка проблема критична на даний момент і за яку потрібно братись в першу чергу.
- Сервіс позволяє отримувати сповіщення в реальному часі про нові події, регресуючі проблеми та все більші проблеми, які можуть потребувати негайної уваги.

На рисунку 6.1 зображений основний екран консолі крашлітики, де ми можемо побачити загальну інформацію про проблеми, вибрати операційну систему для якої показуватимуться результати, відфільтрувати по різним признакам.

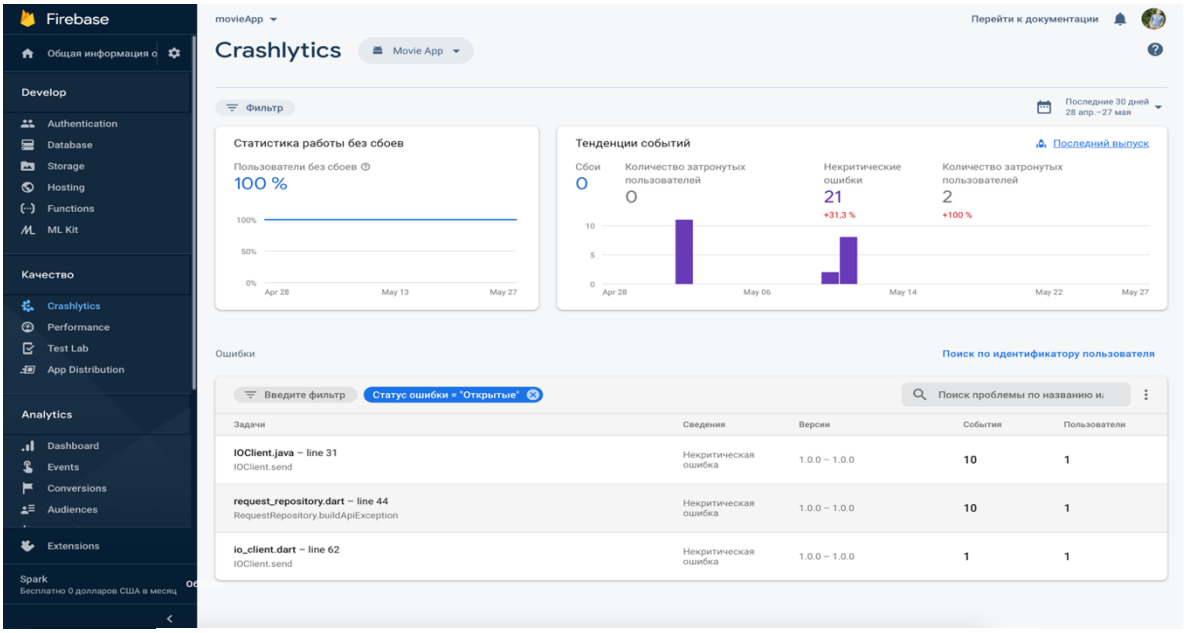

Рис 6.1 Головне вікно консолі Firebase Crashlitics

На рисунку 6.2 можна побачити детальний екран тої чи іншої проблеми, вона відобразить не тільки графік частоти відтворення проблеми але й марки й моделі телефонів, їх характеристики, кількість вільної памяті під час виникнення проблеми.

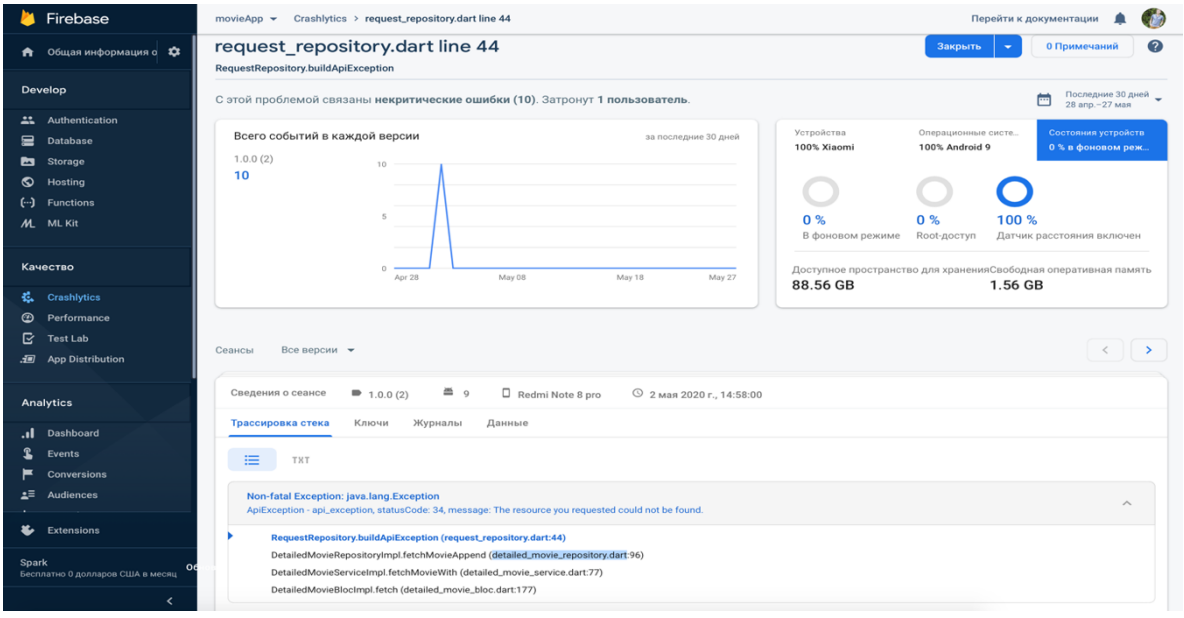

Рис 6.2 Інформація про конкретну відстежену проблему

Отже зрозуміло, що крашлітика – це дуже потужний інструмент для відстеження та попередження помилок і який дозволить ефективно вирішити проблему, як тільки вона появиться.

РОЗДІЛ 7
ЕКОНОМІЧНА ЧАСТИНА

7.1 Економічна характеристика програмного продукту

Метою дипломної роботи є розробка системи пошуку фільмів, що надає користувачу можливість спростити процес пошуку і вибору фільмів, створювати списки збережених та вподобаних картин, пошуку людей, що брали участь в створенні кінострічки та перегляду їх фільмографії. Система надає широкий вибір кінопродукції, яка є також розподілена за жанрами, акторами, вподобаннями та іншими характеристиками, що гарантує користувачу вирішення його проблеми з пошуком фільму в найкоротші терміни.

Основними економічними факторами, що впливають на доцільність використання системи, є грошові витрати на її розробку та впровадження, а також час на проектування, розробку та тестування. Також частина грошових витрат може піти на створення та поширення реклами на відповідних платформах, щоб забезпечити швидкий приріст користувачів додатку [21].

Розроблювана система відповідає вимогам до сучасного програмного забезпечення:

- багатофункціональність;
- зручний інтерфейс;
- продуктивність;
- надійність виконання.

На сьогоднішній день розробка систем, які допомагають у пошуку кінострічок є дуже актуальною, оскільки попит на різноманітне програмне забезпечення зростає кожного дня в умовах розвитку кіноіндустрії та популярності мобільних додатків. При розвитку системи більший прибуток гарантуватиметься меншим фінансовим забезпеченням.

7.2 Гіпотеза щодо потреби розроблення продукту

На сьогодні фільми є невід'ємною частиною людського життя. Кожна людина час від часу відпочиває переглядаючи різні кінострічки, в тому числі велика кількість людей сформувала свій смак щодо фільмів і зустрічалась із складністю вибору наступної картини для перегляду. Чималу роль у вирішенні цієї проблеми відіграють різноманітні сайти для пошуку фільмів, але зачасту їх проблема у великому виборі і малому напрямі на інтереси і вподобання того чи іншого користувача, рідко зустрічається можливість побачити не тільки фільмографію актора, а й людини що знімала стрічку.

Даний програмний продукт розробляється з метою надати користувачу зручний спосіб для вибору картини, пошуку людей з кінематографу та інформації про них а також збереження фільмів в списки і все це в єдиному місці. Систему переважно буде використовувати як молодь, так і люди іншого віку – які зацікавленні в наявності зручної системи яка буде знаходитися на їхньому смартфоні. Необхідність розробки такого продукту досить значна, оскільки дана проблема актуальна в умовах карантину і попит на програмні продукти зростає щодня та всі люди люблять переглядати фільми.

7.3 Оцінювання та аналіз факторів середовища

Цей розділ передбачає оцінювання та аналіз факторів зовнішнього та внутрішнього середовищ групою експертів. Фактори зовнішні оцінюються за шкалою [-5;5]. Фактори внутрішні оцінюються за шкалою [0;5].

Сума ваг факторів становить одиницю, тобто рівень вагомості для кожного елемента визначається за допомогою коефіцієнтів. Результати наведено у таблиці 7.1.

Таблиця 7.1

Результати експертного оцінювання впливу факторів зовнішнього та внутрішнього середовищ

Фактори	**Середня експертна оцінка, бали**	**Середня вагомість факторів**	**Зважений рівень впливу, бали**
1	2	3	4
Фактори зовнішнього середовища			
Споживачі	5	0,11	0,55
Постачальники	1	0,1	0,1
Конкуренти	-4	0,1	-0,4
Державні органи влади	1	0,05	0,05
Інфраструктура	4	0,06	0,24
Законодавчі акти	0	0,1	0
Профспілки, партії та інші громадські організації	0	0,05	0
Система економічних відносин в державі	4	0,06	0,24
Організації-сусіди	-3	0,01	-0,03
Міжнародні події	4	0,01	0,04
Міжнародне оточення	3	0,03	0,09
Науково-технічний прогрес	5	0,07	0,35
Політичні обставини	2	0,06	0,12
Соціально-культурні обставини	3	0,05	0,15
Особливості міжнародних економічних відносин	4	0,02	0,08
Рівень техніки та технологій	5	0,04	0,2
Стан економіки	3	0,08	0,24
Загальна сума		1	2,02

Продовження таблиці 7.1

Фактори внутрішнього середовища			
1	2	3	4
Цілі	5	0,11	0,55
Структура	5	0,16	0,8
Завдання	5	0,07	0,35
Технологія	5	0,2	1
Працівники	4	0,21	0,84
Ресурси	5	0,25	1,25
Загальна сума		1	4,79

Отже, зважений рівень впливу зовнішнього середовища є позитивним та становить 2,02. Найбільш позитивно впливають на продукт зацікавлені споживачі, науково-технічний прогрес, рівень техніки та технологій і стан економіки. Негативними факторами є конкуренти, організації-сусіди. У внутрішньому середовищі сумарний зважений рівень впливу становить 4,79.

За результатами оцінювання та аналізування факторів зовнішнього та внутрішнього середовищ можна зробити висновок, що розробка системи є актуальною та користується попитом на сучасному ринку.

7.4 Бюджетування

На етапі бюджетування необхідно визначити собівартість продукту, який розробляється та економічно обґрунтувати доцільність вибору однієї із стратегій. Бюджет матеріальних витрат наведено в таблиці 7.2.

Таблиця 7.2

Бюджет матеріальних витрат (робоча техніка)

Назва матеріалів та комплектуючих	Марка, тип, модель	Фактична кількість, шт.	Ціна за одиницю, грн.	Разом, грн.
Комп'ютер	Apple MacBook Pro 15 (MJLQ2)	1	48000,0	48000,0
Тестовий девайс (Android)	Xiaomi Redmi Note 9S	1	6499,0	6499,0
Тестовий девайс (IOS)	Apple iPhone Xr 64GB (MRY52)	1	19999,0	19999,0
Програмне забезпечення	xCode	1	0	0
Програмне забезпечення	VS Code	1	0	0
Монітор	Philips (245E1S)	1	4899,0	4899,0
Модем	TP-LINK (TL-WR841N)	1	555,0	555,0
Разом:				138 452,0

Систему будуть розробляти троє осіб. Бюджет витрат на оплату праці наведений у таблиці 7.3. Час розробки системи розрахований на 4 місяці, це 88 робочих днів.

Таблиця 7.3

Бюджет витрат на оплату праці

Посада, спеціальність	Кількість працівників, осіб	Час роботи, дні	Денна заробітна плата, грн.	Сума витрат на оплату праці, грн.
Основна заробітна плата				
Розробник	1	90	1200,0	108 000,0
Дизайнер	1	55	600,0	33 000,0
Тестувальник ПЗ	1	35	400,0	14 000,0
Разом:				155 000,0
Додаткова заробітна плата				
Розробник	1	10	1800,0	18 000,0
Дизайнер	1	8	800,0	6400,0
Тестувальник ПЗ	1	5	500,0	2500,0
Разом:				26 900,0

Бюджет обов’язкових відрахувань та податків наведений у таблиці 7.4.

Визначаємо суму єдиного внеску на соціальне страхування кожної категорії працівників:

$$ЄСВ_{програміста} = (108\,000{,}0 + 18\,000{,}0) * 0{,}22 = 27\,200{,}0 \text{ (грн)}$$

$$ЄСВ_{дизайнера} = (33\,000{,}0 + 6400{,}0) * 0{,}22 = 8008{,}0 \text{ (грн)}$$

$$ЄСВ_{тестувальника\ ПЗ} = (14\,000{,}0 + 2500{,}0) * 0{,}22 = 3630{,}0 \text{ (грн)}$$

Визначаємо суму податку з доходів фізичних осіб для кожної категорії працівників:

$$ПДФО_{програміста} = (108\,000{,}0 + 18\,000{,}0) * 0{,}18 = 22\,680{,}0 \text{ (грн)}$$

$$ПДФО_{дизайнера} = (33\,000{,}0 + 6400{,}0) * 0{,}18 = 7092{,}0 \text{ (грн)}$$

$$ПДФО_{тестувальника\ ПЗ} = (14\,000{,}0 + 2500{,}0) * 0{,}18 = 2970{,}0 \text{ (грн)}$$

Визначаємо суму військового збору для кожної категорії працівників:

$$ВЗ_{програміста} = (108\,000{,}0 + 18\,000{,}0) * 0{,}015 = 1890{,}0 \text{ (грн)}$$

$$ВЗ_{дизайнера} = (33\,000{,}0 + 6400{,}0) * 0{,}015 = 553{,}6 \text{ (грн)}$$

$$ВЗ_{тестувальника\ ПЗ} = (14\,000{,}0 + 2500{,}0) * 0{,}015 = 247{,}5 \text{ (грн)}$$

Таблиця 7.4

Бюджет обов’язкових відрахувань та податків

Посада, спеціальність	Сума основної заробітної плати	Сума додаткової заробітної плати	Разом витрат на оплату праці	Сума ЄСВ *,грн	Сума податку з доходів фізичних осіб, грн.	Сума військового збору, грн
Розробник	108 000,0	18 000,0	126 000,0	27 200,0	22 680,0	1890,0
Дизайнер	33 000,0	6400,0	39 600,0	8008,0	7092,0	553,6
Тестувальник ПЗ	14 000,0	2500,0	16 500,0	3630,0	2970,0	247,5
Разом:	155 000,0	26 900,0	182 100,0	38 838,0	32 742,6	2691,1

Крім прямих виробничих витрат можуть виникати загальновиробничі витрати (змінні та постійні), які не можна напряму віднести до конкретного виду продукції. Бюджет загальновиробничих витрат наведено в таблиці 7.5.

До адміністративних витрат відносяться загальногосподарські втрати, спрямовані на обслуговування та управління підприємством. Витрати на збут – це витрати пов'язані з реалізацією продукції. Бюджет адміністративних витрат та витрат на збут наведено в таблиці 7.6.

Таблиця 7.5

Бюджет загальновиробничих витрат

Статті витрат	**Сума, грн.**
Змінні загальновиробничі витрати, у т.ч.:	
- заробітна плата допоміжного персоналу;	20 000,0
- витрати на МШП;	2000,0
- витрати на електроенергію та технологічні цілі;	1500,0
Разом змінних витрат:	23 500,0
Постійні загальновиробничі витрати, у т.ч.:	
- комунальні послуги;	12 000,0
- витрати на оренду;	45 000,0
- інші постійні витрати;	10 000,0
Разом постійних витрат:	67 000,0
Разом загальновиробничих витрат:	90 500,0

Таблиця 7.6

Бюджет адміністративних витрат та витрат на збут

Статті витрат	**Сума, грн.**
Адміністративні витрати, у т.ч.:	
- заробітна плата адміністративного персоналу;	8500,0
- витрати на МШП;	1000,0
- витрати на сплату податків і зборів;	1500,0
- інші адміністративні витрати;	450,0
Разом адміністративних витрат:	11 450,0
- Витрати на збут (рекламу);	26 000,0
Разом витрат на збут:	26 000,0

Витрати на виробництво продукції у вартісному виразі формують її виробничу собівартість. Цей показник відображає зростання продуктивності праці, економію ресурсів, технічний прогрес.

Зведений кошторис витрат на розробку проектного рішення наведено в таблиці 7.7.

Таблиця 7.7

Зведений кошторис витрат на розробку проектного рішення (продукту)

Статті витрат	**Разом, грн.**
Матеріали та комплектуючі вироби	138 452,0
Основна заробітна плата	155 000,0
Додаткова заробітна плата	26 900,0
Відрахування на соціальне страхування	38 838,0
Загальновиробничі витрати, у т.ч.:	
- змінні;	23 500,0
- постійні;	90 500,0
Разом виробничих витрат:	473 190,0
Адміністративні витрати	11 450,0
Витрати на збут	26 000,0
Разом виробничих і операційних витрат:	510 640,0

Ціну продукту визначаємо за формулою:

$$Ц = СБ * Р + СБ \quad (1)$$

де Ц – ціна програмного продукту, грн.

СБ – собівартість програмного продукту, грн.

Р – рентабельність %.

При рентабельності 40% вартість програмного продукту буде становити

Ц = 510 640,0 * 0,4 + 510 640,0 = 735 321,6 грн

Бюджет фінансових результатів наведено в таблиці 7.8.

Таблиця 7.8

Бюджет фінансових результатів

Показники	Сума, грн.
Дохід від реалізації продукції	735 321,6
Податок на додану вартість (20 %)	147 064,32
Чистий дохід від реалізації продукції	588 257,28
Собівартість реалізованої продукції	473 190,0
Валовий прибуток	115 067,28
Операційні витрати:	
- адміністративні витрати	11 450,0
- витрати на збут	26 000,0
Фінансовий результат	77 567,28
Податок на прибуток (18%)	32 742,6
Чистий прибуток (збиток)	44 824,69

Таким чином, кінцевим результатом розробки інформаційної системи є одержання прибутку в сумі 44 824,69 грн.

7.5 Вибір стратегії

З першої групи стратегічних альтернатив розвитку обираю стратегію розроблення нового продукту, адже розробляється нове програмне забезпечення, яке забезпечить засоби вирішення потреб користувача.

З другої групи стратегічних альтернатив, я обираю стратегію розвитку продукту, з огляду на те що створюю нову систему для існуючого сегменту ринку та споживачів.

Загальна сума виробничих витрат становить 473 190,0 грн. Однак на підприємстві мають місце також адміністративні витрати – 11 450,0 грн і витрати на збут – 26 000,0 грн, що є рекламою і гарантує більший попит додатку.

Враховуючи припустиму рентабельність в 40% було визначено ціну продукту – 735 321,6 грн. Обчислено що від реалізації однієї продукції

отримаємо чистий прибуток 44 824,69 гривень. Порівнюючи отримані дані з загальною комерційністю подібних систем, можна зробити висновок щодо актуальності задачі із розробки інформаційної системи пошуку фільмів подій.

Після проведення економічного аналізу можна зробити висновок, що дійсно існує потреба у розробці нового проекту, опираючись на те, що є потенційні клієнти, які зацікавлені в даному рішенні та готові за це заплатити.

ВИСНОВКИ

У результаті виконання дипломної роботи був розроблений додаток для пошуку фільмів із розширеним функціоналом. Код додатку був написаний із використанням архітектури BLoC і мови програмування Dart, із застосуванням фреймворку Flutter. Оскільки програма буде запускатись на великій кількості різних мобільних пристроїв – була впроваджена система відстеження збоїв та помилок додатку для швидкого вирішення можливих проблем.

Дизайн додатку був створений у лаконічному стилі з інтуїтивно зрозумілим інтерфейсом, щоб користувач не задумувався як знайти потрібну йому функцію.

Дана програма має хороші перспективи в розвитку. На разі створений найпотрібніший функціонал, що допомагає вирішити основні цілі, проте є ще дуже багато функціоналу якого можна додати в програму. Зокрема можна дозволити користувачам: робити власні списки фільмів, робити списки акторів, групувати і відображати фільмів за колекціями, зберігати дані користувача на віддаленому сервері для можливості повернути дані, якщо додаток був випадково видалений, та багато інших можливостей. Завдяки використанню правильної архітектури – розробка кожної одиниці нового функціоналу забере відносно не багато часу.

Перш за все, для зручного завантаження додатку потрібно додати його у найбільші магазини додатків Play Market та AppStore. На початку додаток цілком може бути безкоштовним, проте із зростом кількості завантажень можна буде додати контекстну рекламу або ж зробити спеціальні підписки із розширеним функціоналом за окрему плату.

Очевидно, що розроблена програма підтримки пошуку фільмів має досить багато перспектив зокрема серед молоді.

СПИСОК ЛІТЕРАТУРИ

1. Windmill E. Flutter in Action / Eric Windmill., 2020. – 310 с. – (Manning Publications).
2. Бэнкс А. React и Redux: функциональная разработка 2018 / А. Бэнкс, Е. Порселло., 2018. – 336 с. – (Питер).
3. Мартін Р. Чиста архітектура: мистецтво розробки програмного забезпечення / Роберт Мартін.. – 416 с.
4. Савіна Г. Фактори зовнішнього та внутрішнього впливу на рівень ефективності управління підприємством / Г. Савіна, Т. Скібіна. // ДКС-центр. – 2016. – №12.
5. Сем К. Тестування програмного забезпечення / К. Сем, Д. Фолк, Е. К. Нгуен., 2001. – 544 с. – (Діасофт).
6. Vandad Nahavandipoor iOS 8 Swift Programming Cookbook / Vandad Nahavandipoor – Boston: O'Reilly Media., 2014. – 902 p.
7. Брюханова Г. Комп'ютерні дизайн-технології / Галина Брюханова., 2018. – 180 с. – (Центр навчальної літератури).
8. Crashlytics [Електронний ресурс]. – 2011. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Crashlytics.
9. Офіційна документація мови програмування Dart [Електронний ресурс] // Google. – 2015. – Режим доступу до ресурсу: https://dart.dev/guides.
10. Офіційна документація фреймворку Flutter [Електронний ресурс] // Google. – 2017. – Режим доступу до ресурсу: https://flutter.dev/docs.
11. Матрёнин А. Обзор архитектур управления состоянием на Flutter [Електронний ресурс] / Антон Матрёнин. – 2019. – Режим доступу до ресурсу: https://dou.ua/lenta/articles/flutter-architecture/.
12. Офіційний збірник бібліотек мови dart [Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/.

13. How flutter works under the hood [Електронний ресурс] // ElevateX. – 2019. – Режим доступу до ресурсу: https://buildflutter.com/how-flutter-works/.
14. Основні поняття кросплатформного програмування [Електронний ресурс] – Режим доступу до ресурсу: https://cutt.ly/fyKRuua
15. Операційна система IOS [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/IOS.
16. Операційна система Android [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/Android.
17. Zorrilla M. Flutter Lifecycle for Android and iOS Developers [Електронний ресурс] / Mariano Zorrilla // Medium. – 2019. – Режим доступу до ресурсу: https://medium.com/flutter-community/flutter-lifecycle-for-android-and-ios-developers-8f532307e0c7.
18. Chathurika H. Why You should use Google Flutter [Електронний ресурс] / Harshani Chathurika. – 2019. – Режим доступу до ресурсу: https://uxplanet.org/why-you-should-use-google-flutter-42f2c6ba036c.
19. BLoC паттерн [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://habr.com/ru/post/475404/.
20. Packages for Dart language [Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/.
21. Довгалюк Н. В. Методологія визначення та методика аналізу економічної ефективності [Електронний ресурс] / Наталія Валеріївна Довгалюк. – 2010. – Режим доступу до ресурсу: shorturl.at/xPY06.

ДОДАТОК А

Приклад коду моделі даних.

```
import 'package:dart_tmdb_api/api/base/dao.dart';
import 'package:dart_tmdb_api/api/models/enums/genres.dart';
import 'package:equatable/equatable.dart';
import 'package:flutter/material.dart';

class Movie extends Equatable {
  const Movie({
    @required this.id,
    @required this.title,
    @required this.genres,
    @required this.isAdult,
    @required this.hasVideo,
    @required this.overview,
    @required this.voteCount,
    @required this.popularity,
    @required this.posterPath,
    @required this.releaseDate,
    @required this.voteAverage,
    @required this.backdropPath,
    @required this.originalTitle,
    @required this.originalLanguage,
  });

  final int id;
  final int voteCount;
  final bool isAdult;
  final bool hasVideo;
  final double popularity;
  final double voteAverage;
  final String title;
  final String overview;
  final String posterPath;
  final String releaseDate;
  final String backdropPath;
  final String originalTitle;
  final String originalLanguage;
  final List<MovieGenre> genres;

  @override
  List<Object> get props => [
        id,
        title,
        genres,
        isAdult,
        hasVideo,
        overview,
        voteCount,
        popularity,
        posterPath,
        releaseDate,
        voteAverage,
        backdropPath,
        originalTitle,
        originalLanguage,
      ];
}


class MovieDao implements StorageDao<Movie> {
  const MovieDao();

  static const id = 'id';
```

```
 static const title = 'title';
 static const isAdult = 'adult';
 static const hasVideo = 'video';
 static const overview = 'overview';
 static const genreIds = 'genre_ids';
 static const voteCount = 'vote_count';
 static const popularity = 'popularity';
 static const posterPath = 'poster_path';
 static const releaseDate = 'release_date';
 static const voteAverage = 'vote_average';
 static const backdropPath = 'backdrop_path';
 static const originalTitle = 'original_title';
 static const originalLanguage = 'original_language';


@override
 Movie fromMap(final Map<String, dynamic> map) => Movie(
       id: map[id],
       title: map[title],
       isAdult: map[isAdult],
       hasVideo: map[hasVideo],
       overview: map[overview],
       voteCount: map[voteCount],
       posterPath: map[posterPath],
       releaseDate: map[releaseDate],
       backdropPath: map[backdropPath],
       originalTitle: map[originalTitle],
       originalLanguage: map[originalLanguage],
       popularity: map[popularity]?.toDouble(),
       voteAverage: map[voteAverage]?.toDouble(),
       genres: List<int>.from(map[genreIds]).map(MovieGenreExt.fromId).toList(),
     );


@override
 Map<String, dynamic> toMap(final Movie movie) => {
       id: movie.id,
       title: movie.title,
       isAdult: movie.isAdult,
       hasVideo: movie.hasVideo,
       overview: movie.overview,
       voteCount: movie.voteCount,
       posterPath: movie.posterPath,
       releaseDate: movie.releaseDate,
       backdropPath: movie.backdropPath,
       originalTitle: movie.originalTitle,
       originalLanguage: movie.originalLanguage,
       popularity: movie.popularity,
       voteAverage: movie.voteAverage,
       genreIds: movie?.genres?.map<int>((genre) => genre.id)?.toList(),
     };

 List<Movie> fromJson(final Map<String, dynamic> json) {

   final results = List<Map<String, dynamic>>.from(json['results']);

   if (results == null || results.isEmpty) return [];

   return fromList(results);
 }

 List<Movie> fromList(final List<Map<String, dynamic>> list) =>
     list.map(fromMap).toList();
}
```

ДОДАТОК Б

Приклад коду репозиторію даних.

```
import 'package:dart_tmdb_api/api/base/repository.dart';
import 'package:dart_tmdb_api/api/models/movie/movie.dart';
import 'package:dart_tmdb_api/api/repositories/api_exception.dart';
import 'package:dart_tmdb_api/api/repositories/list/list_args.dart';
import 'package:dart_tmdb_api/api/repositories/movies/movies_result.dart';
import 'package:dart_tmdb_api/api/repositories/request_repository.dart';
import 'package:dart_tmdb_api/api/services/api_service.dart';

abstract class MoviesRepository implements Repository {
  Future<MoviesResult> fetchMovies(
    final MoviesListArgs moviesList, {
    final int page = 1,
  });
}

class MoviesRepositoryImpl extends RequestRepository implements MoviesRepository {
  MoviesRepositoryImpl({
    @required ApiClient client,
    JsonCodec jsonCodec = const JsonCodec(),
    MovieDao movieDao = const MovieDao(),
  })  : assert(jsonCodec != null),
        assert(movieDao != null),
        assert(client != null),
        _client = client,
        _movieDao = movieDao,
        _jsonCodec = jsonCodec;

  static const missingException = 'missing_movies_information';

  final ApiClient _client;
  final MovieDao _movieDao;
  final JsonCodec _jsonCodec;

  @override
  Future<MoviesResult> fetchMovies(
    MoviesListArgs moviesList, {
    int page = 1,
  }) async {
    try {
      final uri = requestUri(
        moviesList.apiRequest,
        pathParameters: moviesList.pathParams,
        queryParameters: {'page': '$page'}..addAll(moviesList.queryParams),
      );

      final response = await _client.get(uri);
      final Map<String, dynamic> json = _jsonCodec.decode(response.body);

      if (response.statusCode != HttpStatus.ok) throw buildApiException(json);

      final movies = _movieDao.fromJson(json);
      final numberOfPages = json['total_pages'];
      final numberOfMovies = json['total_results'];

      if (numberOfPages == null || numberOfMovies == null) {
        throw _buildInfoException(json);
      }

      return MoviesResult.success(movies, numberOfPages, numberOfMovies);

    } catch (e, stack) {
```

```
    final exception = e.runtimeType == ApiException
        ? e
        : ApiException(
            e,
            statusCode: null,
            stackTrace: stack,
            statusMessage: 'Failed to fetch ${moviesList.name} movies',
          );

    return MoviesResult.failed(exception);
  }
 }

 @override
 void dispose() => _client.close();

 ApiException _buildInfoException(
  final Map<String, dynamic> json,
 ) =>
     ApiException(
       missingException,
       stackTrace: StackTrace.current,
       statusCode: json['status_code'] ?? -1,
       statusMessage: json['status_message'] ?? '',
     );
}
```

ДОДАТОК В

Приклад коду сервісу.

```
import 'package:dart_tmdb_api/api/base/disposable.dart';
import 'package:dart_tmdb_api/api/models/movie/movie.dart';
import 'package:dart_tmdb_api/api/repositories/list/list_args.dart';
import 'package:dart_tmdb_api/api/repositories/movies/movies_repository.dart';
import 'package:dart_tmdb_api/api/repositories/movies/movies_result.dart';
import 'package:dart_tmdb_api/api/services/exception_tracker.dart';
import 'package:flutter/foundation.dart';

abstract class MoviesService implements Disposable {
  bool get canFetchMore;
  MoviesResult loadCachedMovies();
  Future<MoviesResult> fetchMoreMovies(final MoviesListArgs moviesList);
  Future<MoviesResult> fetchMovies(final MoviesListArgs moviesList);
}

class MoviesServiceImpl extends MoviesService {
  MoviesServiceImpl({
    @required MoviesRepository moviesRepository,
    ExcepionTracker tracker = const BaseExceptionTracker(),
  })  : assert(moviesRepository != null),
        assert(tracker != null),
        _tracker = tracker,
        _moviesRepository = moviesRepository;

  final MoviesRepository _moviesRepository;
  final ExcepionTracker _tracker;

  int _page = 1;
  int _numberOfPages = 0;
  int _numberOfMovies = 0;
  List<Movie> _movies = [];

  @override
```

```
  bool get canFetchMore => _page < _numberOfPages;

  @override
  MoviesResult loadCachedMovies() =>
      MoviesResult.success(_movies, _numberOfPages, _numberOfMovies);

  @override
  Future<MoviesResult> fetchMovies(final MoviesListArgs moviesList) async {
    _page = 1;

    final result = await _moviesRepository.fetchMovies(moviesList);

    if (result.hasException) {
      _tracker?.trackApiException(result.exception);
      return result;
    }

    _numberOfMovies = result.numberOfMovies;
    _numberOfPages = result.numberOfPages;
    _movies = result.value;

    return MoviesResult.success(
      _movies,
      _numberOfPages,
      _numberOfMovies,
    );
  }


  @override
  Future<MoviesResult> fetchMoreMovies(final MoviesListArgs moviesList) async {

    if (!canFetchMore) return null;

    _page++;

    final result = await _moviesRepository.fetchMovies(moviesList, page: _page);

    if (result.hasException) {
      _tracker?.trackApiException(result.exception);
      _page--;

      return result;

    }

    _numberOfMovies = result.numberOfMovies;
    _numberOfPages = result.numberOfPages;

    _movies.addAll(result.value);

    return MoviesResult.success(_movies, _numberOfPages, _numberOfMovies);

  }

  @override
  void dispose() => _moviesRepository.dispose();

}
```

ДОДАТОК Г

Приклад коду блоку.

```
import 'package:dart_tmdb_api/api/models/movie/movie.dart';
import 'package:dart_tmdb_api/api/repositories/list/list_args.dart';
import 'package:dart_tmdb_api/api/services/movies/movies_service.dart';
import 'package:equatable/equatable.dart';
import 'package:flutter/foundation.dart';
import 'package:movie_app/core/extensions/list_extension.dart';
import 'package:movie_app/core/network/request_synchronizer.dart';
import 'package:movie_app/core/network/sink_wrapper.dart';
import 'package:movie_app/core/services/connectivity/connectivity_service.dart';
import 'package:movie_app/src/blocs/list_bloc.dart';
import 'package:rxdart/rxdart.dart';

@immutable
class MoviesState extends Equatable implements ListState<Movie> {
  const MoviesState._({
    @required this.name,
    @required this.values,
    @required this.isFetching,
    @required this.hasInternet,
    @required this.hasException,
  });

  const MoviesState.waiting({
    final List<Movie> movies,
  }) : this._(
          values: movies,
          name: 'waiting',
          isFetching: true,
          hasInternet: true,
          hasException: false,
        );

  const MoviesState.fetched(
    final List<Movie> movies,
  ) : this._(
          name: 'fetched',
          values: movies,
          isFetching: false,
          hasInternet: true,
          hasException: false,
        );

  const MoviesState.failed(
    final List<Movie> movies,
  ) : this._(
          name: 'failed',
          values: movies,
          isFetching: false,
          hasInternet: true,
          hasException: true,
        );

  const MoviesState.noInternet(
    final List<Movie> movies,
  ) : this._(
          values: movies,
          isFetching: false,
          hasInternet: false,
          hasException: false,
          name: 'no internet',
        );
```

```
  bool get hasMovies => values?.isNotEmpty ?? false;

  @override
  final String name;
  @override
  final List<Movie> values;
  @override
  final bool isFetching;
  @override
  final bool hasInternet;
  @override
  final bool hasException;

  @override
  List<Object> get props => [name, values, isFetching, hasInternet, hasException];

  @override
  String toString() => name;

  @override
  int get length => values.notNullLength;
}

abstract class MoviesBloc implements ApiListBloc<MoviesListArgs, MoviesState> {}

class MoviesBlocImpl extends MoviesBloc with RequestSynchronizer {
  MoviesBlocImpl({
    @required MoviesListArgs moviesList,
    @required MoviesService moviesService,
    @required ConnectivityService connectivityService,
  })  : assert(connectivityService != null),
        assert(moviesService != null),
        assert(moviesList != null),
        _moviesList = moviesList,
        _moviesService = moviesService,
        _connectivityService = connectivityService;

  MoviesListArgs _moviesList;
  final MoviesService _moviesService;
  final ConnectivityService _connectivityService;
  final _moviesPublisher = PublishSubject<MoviesState>();

  @override
  MoviesListArgs get args => _moviesList;

  @override
  Stream<MoviesState> get state => _moviesPublisher;

  @override
  MoviesState get initialState =>
      MoviesState.waiting(movies: _moviesService.loadCachedMovies().value);

  bool get _hasInternet => _connectivityService.hasInternet;

  Sink<MoviesState> get _moviesSink => SinkWrapper(_moviesPublisher);

  @override
  Future<void> fetch([final bool shouldRefresh = false]) => executeRequest(() async {
        await _connectivityService.updateConnectivityStatus();
        final cached = shouldRefresh ? [] : _moviesService.loadCachedMovies().value;

        if (!_hasInternet) {
          _moviesSink.add(_buildNoInternetState(cached));

          return ConnectivityStatus.disconnected();
        }
```

```
      if (cached.isBlank) _moviesSink.add(const MoviesState.waiting());

      final fetched = await _moviesService.fetchMovies(_moviesList);

      final state = fetched.hasException
          ? _buildFailedState(cached)
          : _buildFetchedState(fetched.value);

      _moviesSink.add(state);

      return fetched;
    });

  @override
  Future<void> fetchMore() => executeRequest(() async {
      if (!_hasInternet) {
        final cached = _moviesService.loadCachedMovies();
        _moviesSink.add(_buildNoInternetState(cached.value));

        return ConnectivityStatus.disconnected();
      }

      final result = await _moviesService.fetchMoreMovies(_moviesList);

      if (result == null) return result;

      if (result.hasException) {
        final cachedResult = _moviesService.loadCachedMovies();
        _moviesSink.add(_buildFailedState(cachedResult.value));
      } else {
        _moviesSink.add(_buildFetchedState(result.value));
      }

      return result;
    });

  @override
  void forceLoadingState() => _moviesSink.add(const MoviesState.waiting(movies: []));

  @override
  void setArgs(final MoviesListArgs list) {
    if (list == null || list == _moviesList) return;
    _moviesList = list;
  }

  @override
  void dispose() {
    _moviesPublisher.close();
    _moviesService.dispose();
  }

  MoviesState _buildFailedState(final List<Movie> movies) =>
      MoviesState.failed(movies ?? []);

  MoviesState _buildNoInternetState(final List<Movie> movies) =>
      MoviesState.noInternet(movies ?? []);

  MoviesState _buildFetchedState(final List<Movie> movies) =>
      MoviesState.fetched(movies ?? []);
}
```

ДОДАТОК Ґ

Приклад коду користувацького інтерфейсу.

```
import 'package:dart_tmdb_api/api/models/enums/api_image_sizes.dart';
import 'package:dart_tmdb_api/api/models/movie/movie.dart';
import 'package:dart_tmdb_api/api/repositories/list/list_args.dart';
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';
import 'package:movie_app/core/extensions/list_extension.dart';
import 'package:movie_app/core/mixins/list_view_scroll_mixin.dart';
import 'package:movie_app/core/navigation/router.dart';
import 'package:movie_app/core/ui/pull_to_refresh.dart';
import 'package:movie_app/core/ui/scroll_to_top.dart';
import 'package:movie_app/generated/l10n.dart';
import 'package:movie_app/src/blocs/movies_bloc.dart';
import 'package:movie_app/src/components/fade_api_image.dart';
import 'package:movie_app/src/components/header.dart';
import 'package:movie_app/src/components/placeholders/placeholder.dart';
import 'package:movie_app/src/components/spinner.dart';
import 'package:movie_app/src/mixins/page_opener.dart';
import 'package:movie_app/src/utils/styles/color_styles.dart';
import 'package:movie_app/src/utils/styles/decoration_styles.dart';
import 'package:movie_app/src/utils/styles/text_styles.dart';

class MoviesPage extends StatefulWidget {
  const MoviesPage(this.bloc);

  final MoviesBloc bloc;

  @override
  _MoviesPageState createState() => _MoviesPageState();
}

class _MoviesPageState extends State<MoviesPage> with ListViewScrollMixin, PageOpener {
  ScrollController controller;
  ScrollPosition pageProgressPosition;

  @override
  void initState() {
    super.initState();
    controller = ScrollController();
    widget.bloc.fetch();
  }

  static const _gridDelegate = SliverGridDelegateWithFixedCrossAxisCount(
    childAspectRatio: 0.55,
    crossAxisSpacing: 12,
    mainAxisSpacing: 10,
    crossAxisCount: 3,
  );

  @override
  void dispose() {
    widget.bloc.dispose();
    controller.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return DecoratedBox(
      decoration: const BoxDecoration(color: ColorStyles.white),
      child: SafeArea(
        bottom: false,
        child: DecoratedBox(
```

```
decoration: const BoxDecoration(color: ColorStyles.movieListBackround),
child: Stack(
  children: <Widget>[
    StreamBuilder<MoviesState>(
      stream: widget.bloc.state,
      initialData: widget.bloc.initialState,
      builder: (context, snapshot) {
        final state = snapshot.data;
        // TODO(Bohdan): think what to do if the internet dissapears during the scroll
        // probably popup which infirm user and catch socket exceptions from FadeInImage
        if (!state.hasInternet) {
          return ActionPlaceholder.noInternet(
            message: S.of(context).noInternet,
            onButtonTap: () => widget.bloc.fetch(),
          );
        }

        if (state.hasException && state.values.isBlank) {
          return ActionPlaceholder.error(
            onButtonTap: () => widget.bloc.fetch(),
          );
        }

        if (state.isFetching && state.values.isBlank) {
          return LoadingIndicator(color: ColorStyles.red.withOpacity(0.6));
        }

        return PullToRefresh(
          onRefresh: widget.bloc.fetch,
          color: ColorStyles.white,
          backgroundColor: ColorStyles.red,
          displacement: 66,
          notificationPredicate: (notification) => listenWhenReachedBottom(
            notification,
            widget.bloc.fetchMore,
            bottomPadding: 1000,
          ),
          child: ScrollToTop(
            offset: 15,
            scrollController: controller,
            maxScrollTime: const Duration(seconds: 5),
            scrollToTop: ScrollUp(size: MediaQuery.of(context).size.width / 9),
            body: GridView.builder(
              primary: false,
              shrinkWrap: false,
              controller: controller,
              itemCount: state.values.length,
              physics: const ClampingScrollPhysics(),
              padding: const EdgeInsets.only(right: 16, left: 16, top: 68),
              gridDelegate: _gridDelegate,
              itemBuilder: (context, index) => MovieGridCell(
                movie: state.values[index],
                list: widget.bloc.args,
                onTap: () => openMovie(
                  context,
                  movie: state.values[index],
                  list: widget.bloc.args,
                ),
              ),
            ),
          ),
        );
      },
    ),
    Header(
      onLeftChildTap: closePageCall(context),
      childColor: ColorStyles.red,
```

```
              centerChildType: HeaderCenterChild.title,
              leftChildType: HeaderLeftChild.backButton,
              title: widget.bloc.args.localized(context),
            ),
          ],
        ),
      ),
    ),
  );
 }
}

class MovieGridCell extends StatelessWidget {
 const MovieGridCell({this.movie, this.onTap, this.list});

 final Movie movie;
 final MoviesListArgs list;
 final VoidCallback onTap;

 @override
 Widget build(BuildContext context) => Column(
      mainAxisAlignment: MainAxisAlignment.end,
      crossAxisAlignment: CrossAxisAlignment.start,
      children: <Widget>[
       Expanded(
        child: GestureDetector(
         onTap: onTap,
         child: Hero(
          tag: '${movie.id}${list.name}',
          child: ClipRRect(
           clipBehavior: Clip.antiAlias,
           borderRadius: BorderRadius.circular(6.0),
           child: Stack(
            children: <Widget>[
             const SizedBox(
              width: double.infinity,
              height: double.infinity,
              child: DecoratedBox(
               decoration: BoxDecoration(color: ColorStyles.posterPlaceholder),
              ),
             ),
             FadeApiImage(
              movie.posterPath,
              fit: BoxFit.cover,
              size: ApiSize.w185,
              width: double.infinity,
              height: double.infinity,
              fadeInCurve: Curves.easeInOut,
              fadeOutCurve: Curves.easeInOut,
              failedPlaceholder: Image.asset(
               'res/assets/movie_placeholder.png',
               color: ColorStyles.moviePlaceholder,
              ),
             ),
            ],
           ),
          ),
         ),
        ),
       ),
       const SizedBox(height: 5),
       SizedBox(
        height: 30,
        child: Text(
         '${movie.title}',
         maxLines: 2,
         overflow: TextOverflow.ellipsis,
```

```
            style: TextStyles.standart(fontWeight: FontWeight.w600, fontSize: 12),
          ),
        )
      ],
    );
}

class ScrollUp extends StatelessWidget {
  const ScrollUp({this.size = 55});

  final double size;

  @override
  Widget build(BuildContext context) {
    return Center(
      child: SizedBox(
        width: size,
        height: size,
        child: DecoratedBox(
          decoration: BoxDecoration(
            color: ColorStyles.movieListBackround,
            boxShadow: DecorationStyles.scrollUpShadow,
            borderRadius: BorderRadius.circular(size / 2),
          ),
          child: Icon(Icons.keyboard_arrow_up, color: ColorStyles.red, size: size),
        ),
      ),
    );
  }
}
```