КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ БІБЛІОТЕКИ ФІЛЬМІВ НА iOS-ПЛАТФОРМІ

Виконав студент 4-го курсу

Олександр ТОЦЬКИЙ _____

(підпис)

Науковий керівник:

доцент, кандидат фіз.-мат.наук

Максим ВЕРЕС _____

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем

Протокол 10 від 25.05.2022р.

Завідувач кафедри
Олександр ПРОВОТАР _____

(підпис)

Київ – 2022

РЕФЕРАТ

Обсяг роботи 43 сторінки, 12 ілюстрацій, 14 джерел посилань.

АРХІТЕКТУРНІ ШАБЛОНИ, БАЗА ДАНИХ REALM, БІБЛІОТЕКА ФІЛЬМІВ, ІНТЕРФЕЙС КОРИСТУВАЧА, МОБІЛЬНИЙ ДОДАТОК, МОВА СВІФТ.

Об’єктом роботи є використання та ефективне застосування бібліотек мови Swift та інших допоміжних засобів для розробки мобільного додатку на платформі iOS. Предметом роботи є мобільний додаток на платформі iOS, що реалізовує в собі зручний пошук та ознайомлення з бібліотекою фільмів.

Метою роботи є аналіз існуючих аналогів вирішення цієї проблеми, ознайомлення з розробкою мобільних додатків на різних платформах та створення на основі аналізу мобільного додатку «Бібліотека фільмів» для перегляду та вибору фільмів, якими цікавиться споживач, що може набути широкого застосування серед користувачів мобільної платформи iOS.

Методи розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Xcode, мова програмування Swift, бібліотека для побудови графічного інтерфейсу UIKit, база даних Realm, сервіс для API запитів - The Movie Database.

Результати роботи: виконано загальний огляд засобів створення мобільного додатку, проаналізовано переваги та недоліки використання цих засобів під час розробки, розроблено мобільний додаток «Бібліотека фільмів», який у подальшому може набуто практичного характеру та бути поширений за допомогою мобільного магазину додатків, AppStore, серед користувачів мобільних пристроїв.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

IDE – Integrated development environment, комплексне програмне рішення для розробки програмного забезпечення;

iOS – мобільна операційна система від компанії Apple Inc;

macOS – Unix-подібна операційна система, розроблена компанією Apple Inc;

ЦП – центральний процесор;

БД – база даних;

UX, UI – User Experience, User Interface, користувацький досвід, яким чином користувач взаємодіє з інтерфейсом, та наскільки зрозумілий та зручний додаток для нього; користувацький інтерфейс, оформлення додатку: кольори, шрифти, іконки та кнопки, відповідно;

JSON – JavaScript Object Notation, текстовий формат обміну даними між комп’ютерами, який дає змогу описувати об’єкти та інші структури даних;

API – Application Programming Interface.

ВСТУП

Оцінка сучасного стану об’єкта розробки. З розвитком технологій наше життя заполонили всі можливі мобільні пристрої, роль яких сьогодні складно недооцінити, адже за допомогою мобільного телефону ми можемо будь-якої миті зв'язатися зі своїми друзями, родичами чи колегами по роботі, щоб оперативно дізнатися чи передати інформацію. Всі можливі корисні програми, функції та опції, якими оснащені сучасні стільникові телефони роблять цей невеликий за розміром пристрій настільки багатофункціональним, що неможливо ними не захоплюватися. Тому розробка нових мобільних додатків буде актуальна ще на багато років вперед, адже спростити життя користувача або внести в його життя використання нових технологій і є сенсом мобільної розробки.

Актуальність роботи та підстави для її виконання. Кількість користувачів мобільних додатків збільшується з кожним днем в умовах розвитку інформаційного суспільства. Тому і зростає попит на використання зручних та корисних додатків, які допоможуть людині досягти поставленої задачі, в нашому випадку – це пошук та ознайомлення з бібліотекою фільмів для подальшого перегляду.

Перегляд фільмів дозволяє відволіктись від рутини та задовольняє загальну потребу у яскравих емоціях. В часи пандемії, яка потрохи вщухає, та війни, коли ця потреба вкрай важлива – це ідея є дуже актуальною.

Існуючі рішення цієї задачі або вкрай складні, тобто мають занадто багато функціоналу, або мають не зручний користувацький інтерфейс, який

зараз відіграє вирішальну роль у виборі мобільного додатку. Тому створення простого додатку, який надавав би можливість легко і зручно знаходити фільми – є надзвичайно корисною задачею.

Мета й завдання роботи. Метою роботи є розробка мобільного додатку, який був б цікавим для користувачів та заохочував б їх до подальшого використання. Для досягнення цієї мети поставлено такі завдання:

- розглянути існуючі додатки, що вирішують цю проблеми, визначити їх перевагу та недоліки;
- ознайомитися з відмінностями розробки мобільного додатку на різних платформах;
- обрати архітектуру мобільного додатку;
- спроектувати, розробити додаток та описати процес його створення.

Об’єкти, методи й засоби розроблення. Об’єктом розробки мобільного додатку «Бібліотека фільмів» на платформі iOS є використання API сервісу «The Movie Database», існуючих бібліотек та фреймворків, які надає мова Swift.

В якості інструменту створення програмного застосунку було обрано Xcode IDE – інтегроване середовище розробки, створена компанією Apple, яка дозволяє створювати додатки для таких платформ, як iOS, macOS. Вона містить у собі багату кількість корисних для розробників інструментів, наприклад, Interface Builder – додаток, що використовується для створення графічних інтерфейсів користувача, або вбудована підтримка керування

вихідним кодом за допомогою системи контроля версій і прокотолу Git, що дозволяє зручно працювати з віддаленими репозиторіями.

У якості мови програмування було обрано Swift через його швидкість, сучасність (ця мова була створена у 2014 році), легкий синтаксис, який дозволяє чітко та стисло висловлювати складні ідеї.

Можливі сфери застосування. Мобільний додаток «Бібліотека фільмів» може мати широке практичне застосування серед більшості користувачів мобільної платформи iOS для полегшення пошуку фільмів, у яких вони зацікавлені.

РОЗДІЛ 1 ПІДСТАВИ ТА ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

Для розробки мобільного застосунку були використані наступні технології:

- «природний» (native) шлях розробки, а саме розробка під мобільну платформу iOS;
- Swift, як мова програмування;
- бібліотека UIKit для побудови графічного інтерфейсу додатку.

Середовищем розробки обрав Xcode, тому що це безкоштовна IDE, розроблена компанією Apple Inc., яка пропонує своїм користувачам широкий вибір інструментів для комфортної розробки та подальшого дослідження коректної роботи застосунку.

1.1 Опис проблеми

За останні декілька років людство стикнулося з великою пандемією COVID-19, через яке життя більшості людей змінилося. Вона принесла багато змін у те, як ми живемо, а разом з тим невизначеність, фінансовий тиск та соціальну ізоляцію. Інформаційне перевантаження та дезінформація негативно впливала на наш мозок на початку цієї хвороби, адже ніхто не очікував, що буде далі. Разом з тим, здоров’я майже кожної третьої людини, яка перехворіла на COVID-19, значно погіршилося, згідно з дослідженнями Оксфордського університету [1]. Їм діагностували проблеми, пов’язані з

неврологічним або психіатричним станом. Найпоширенішими психічними розладами були тривожні розлади, розлади настрою (наприклад, депресія), розлади зловживання психоактивними речовинами та безсоння.

Тому важливо дотримуватися стратегій, направлених на покращення нашого психічного стану, які допомагають контролювати наше життя. До цього відносяться такі банальні правила, як правильний сон, активні фізичні заняття та, звісно, ж фокус на позитивних думках. Як раз тут, на мою думку, і стає актуальною ідея перегляду фільмів, яка, на перший погляд, може бути і не така очевидна.

Доведено, що перегляд комедій або романтичних фільмів покращує настрій і здоров'я. Багато сертифікованих лікарів дійсно погодяться, що сміх розширює наші кровоносні судини, і тому він допомагає, знижуючи кров'яний тиск та кількість гормонів стресу, зміцнюючи імунну систему та зменшуючи тривожність. Педіатри першими скажуть вам зі свого досвіду та знань у роботі з дітьми, що багато досліджень показали, що інтенсивний сміх протягом 10-15 хвилин має той самий ефект на здоров'я серця, як і під час фізичних вправ [2].

Сумні фільми, як правило, можуть пригнічувати нас ще більше, якщо в нас наявні симптоми депресії. Однак інші дослідження показали, що сумні та депресивні фільми викликають у глядачів почуття вдячності до власного життя та близьких, змушуючи їх відчувати себе трохи задоволеніше, ніж раніше. За іронією долі, трагедії змушують вас більше цінувати власне життя.

Фільми дають нам шанс втекти та повністю розслабитися. Це дає вам відпочинок від поточних проблем і турбот, дозволяючи інвестувати в щось інше, ніж у власні проблеми. Фільми знижують рівень тривоги і стресу,

вивільняючи кортизон і дофіні в мозку. Коли фільм закінчується, людина може повернутися до своїх турбот, але вже готова розглянути їх з нової точки зору.

Саме тому я подумав, що створення зручного інструменту у вигляді мобільного додатку для пошуку фільмів дозволить знизити кількість наростаючого стресового навантаження, яке виникає в житті кожного із нас.

1.2 Огляд існуючих мобільних додатків

Існує чимало додатків, які надають схожу функціональність, але більшість із них має або занадто складний користувацький інтерфейс, де користувачу надається занадто багато можливостей, або цей інтерфейс не відповідає теперішнім стандартам, що може злякати користувача з самого початку.

Що ж повинно бути в «ідеальному» додатку, до якого ми будемо намагатися наблизитися? По-перше, використання нових технологій, які полегшують розробку та одночасно надають гарний інтерфейс. По-друге, наявність пошуку фільмі, яка теж є важливою особливістю. Якщо користувач має в голові якийсь фільм, він одразу захоче спробувати його знайти. По-третє, коротка інформація про фільм, щоб користувач одразу міг зрозуміти, чи зацікавлений він в ньому взагалі чи ні.

Я розглянув два популярні мобільні додатки, які можна скачати з мобільного магазину AppStore: Netflix та My Movies & TV series (рис. 1).

Обидва додатки мають усі перелічені вище вимоги: можна переглядати популярні фільми та шукати ті, в яких ви зацікавлені.

З плюсів, які можна відзначити: мобільний додаток Netflix має дуже привабливий користувацький інтерфейс, адже це одна із найпопулярніших стрімінгових платформ, вона повинна відповідати потребам користувачів; також окремо хотілося б відзначити пошук, який одразу ж пропонує запити, який на даний момент є популярними на цій платформі; щоб ж стосується другого додатку – My Movies & TV series, то він має детальну інформацію про фільми та акторів, які знімаються в них, також окрім фільмів, тут можна переглядати серіали.

Рисунок 1 – Вигляд мобільних аналогів:

а – додаток Netflix; б – додаток My Movies & TV series

Що ж стосується мінусів, які мені вдалося помітити: Netflix надає фільми та серіали, які виробляються тільки цією платформою, а також на початку користувач може розгубитися, незважаючи на цікавий дизайн – інформації дуже багато, і потрібно розібратися у функціональності додатку. У другому додатку пошук працює не так швидко, як хотілося, він має деяку затримку, а також має достатньо застарілий дизайн, який потребує оновлення. Також важливим мінусом є наявність реклами, яка відволікає від процесу пошуку фільмів.

Загалом, можна сказати, що у існуючих аналогах є потрібний функціонал, але немає єдиного додатку, який мав би зручний користувацький інтерфейс та усі перелічені вимоги, який зробив би пошук та перегляд фільмів максимально комфортним.

1.3 Відмінність розробки мобільного додатку на різних платформах

Android і iOS є двома очевидними лідерами мобільних операційних систем. Але, за даними видавництва SensorTower [3], на перший квартал 2022 року, незважаючи на те, що Google Play Market має вдвічі більше завантажень, Apple App Store приносить майже у 2 рази більше доходу (рис. 2).

Існує 3 основних підходи для розробки мобільних додатків: гібридна, кросплатформена та нативна.

Гібридна використовує WebView – специфічні для платформи компоненти, що використовуються для відображення веб-вмісту безпосередньо всередині програми замість стандартного браузера (Safari або Chrome). Таким чином, програми працюватимуть порівняно однаково на всіх пристроях. Щоб створити нативний інтерфейс для кожної платформи, можна використовувати такі технології, як, наприклад, Ionic.

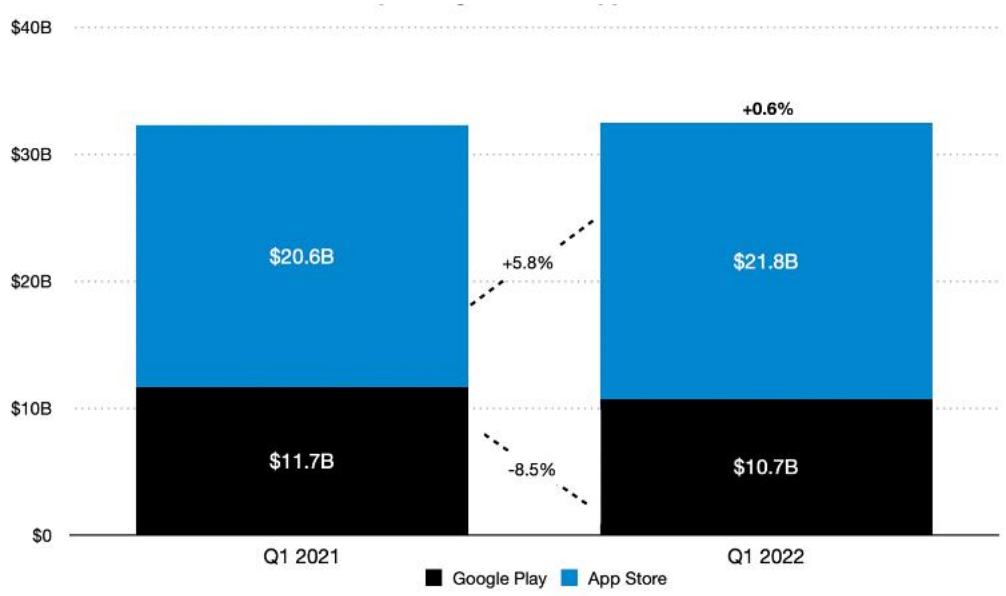

Рисунок 2 – Порівняння витрат у мобільних магазинах Google Play та App Store

Ionic — це фреймворк на основі AngularJS, тому він має цілий набір переваг та підтримки спільноти відомого фреймворку. Більше того, основні розробники фреймворків Angular 2 і 4 обіцяють багато покращень і для Ionic. Також тут добре реалізована важлива можливість розробляти гібридні програми з оригінальним інтерфейсом. Наприклад, він повторно використовує елементи DOM для обробки однієї з причин поганої мобільної продуктивності. І Ionic використовує ngCordova для надання доступу до вбудованих функцій, таких як камера або GPS, як і будь-який інший фреймворк.

Незважаючи на численні переваги інструментів для майже нативної гібридної розробки, у багатьох аспектах рішення далеко не ідеальне. Найпоширенішими його недоліками є: низька продуктивність, повільніші взаємодії, занадто загальний UX, обмежений доступ до апаратного забезпечення пристрою та функцій ОС.

Кроссплатформенність ж рухається іншим шляхом. Міжплатформений підхід використовує єдиний набір інструментів для доставки програм на кількох платформах. На відміну від гібридної розробки, яка поєднує як нативні, так і веб-компоненти, кросплатформна техніка передбачає використання невеб- та автономних інструментів. Найпопулярніші рішення для кросплатформної розробки включають Xamarin і React Native.

У Xamarin вихідний код пишеться за допомогою C# та .Net framework. Потім він перехресно компілюється в нативний код окремо для будь-якої платформи (iOS, Android) [4]. Однак такі фреймворки як Xamarin мають певний недолік, так як інструменти сторонніх розробників не можуть забезпечити негайну підтримку останніх версій iOS та Android, оскільки для впровадження змін та/або впровадження нових плагінів потрібен деякий час.

Нативна розробка передбачає використання специфічних для платформи мов програмування, комплектів для розробки програмного забезпечення, середовища розробки та інших інструментів, наданих постачальниками ОС, насамперед iOS або Android.

Приклади нативних програм iOS включають велику різноманітність офіційних і сторонніх клієнтів для багатьох популярних веб-сервісів, соціальних мереж, а також користувацькі аналоги програм iOS за замовчуванням. Хоча деякі власники сервісів все ще віддають перевагу

гібридним додаткам, щоб заощадити гроші, надаючи єдиний додаток для систем Android і iOS, інші прагнуть максимізувати продуктивність і якість за допомогою нативної розробки. Розробка нативних мобільних додатків пропонує безліч унікальних переваг, недоступних для гібридного та кросплатформного програмного забезпечення. Нижче наведено кілька переваг, пов'язаних із розробкою програм, підтримкою та маркетингом, а також переваги нативних мобільних програм iOS, пов'язані з UX:

a) Серед гібридних, рідних і веб-додатків нативне програмне забезпечення забезпечує найкращу продуктивність на певній платформі.
b) Якщо вони задовольняють вимогам і опубліковані згідно з інструкціями Apple, нативні програми iOS отримують офіційну підтримку з App Store, що корисно для розробки їхніх оновлень, а також для їхнього поширення та обслуговування клієнтів.
c) Нативні програми для iPhone створені з використанням набору програмного забезпечення Apple для розробки програмного забезпечення і, таким чином, мають привабливі інтерфейси користувача, які відповідають загальному стилю операційної системи та інших рідних програм iOS.
d) Вбудовані програми iPhone можуть отримати доступ до всього набору функцій, що надаються як iOS, так і апаратним забезпеченням смартфона, включаючи обчислювальну потужність, камеру, функції штучного інтелекту тощо.
e) Нативні програми для iOS пропонують кращий користувальницький досвід, оскільки вони оптимізовані для певних моделей пристроїв і не повинні підтримувати велику

різноманітність можливих моделей, брендів, версій операційної системи, роздільної здатності екрана тощо, порівняно з рідним програмним забезпеченням Android.

f) Додатковою перевагою від підтримки Apple є підвищена безпека та якість рідних програм iOS у порівнянні з їхніми гібридними чи веб-програмами, що забезпечується регулярними виправленнями продуктивності та безпеки, які надаються безпосередньо Apple. Також важливо відмітити, що мобільні додатки на iOS більше безпечні, ніж на Android – це пов’язано з більш важким етапом перевірки та верифікації додатку на етапі додавання його до мобільного магазину.

Однак основним недоліком, пов’язаним із розробкою нативних програм для iOS є те, що вона вимагає більше часу та коштів, а отримане програмне забезпечення зазвичай охоплює лише частину можливої цільової аудиторії в порівнянні з гібридним або веб-програмним забезпеченням.

Як висновок, можна переконатися, що хоча нативні програми для iOS мають певну частку недоліків, включаючи високу вартість та зменшену цільову аудиторію, вони також пропонують незаперечні переваги як для розробників, так і для користувачів. Найважливіші з них – це покращена продуктивність і користувальницький досвід, а також офіційна підтримка від Apple. Зрештою, «нативний» підхід до розробки програмного забезпечення iOS призводить до більшої задоволеності клієнтів, збільшення кількості завантажень, кращих рейтингів у App Store і, як наслідок, забезпечення кращої прибутковості програми.

1.4 Переваги та особливості мови Swift

Swift - багатопарадигмальна мова програмування, створена за сучасними вимогами до безпеки, продуктивності та шаблонів проектування програмного забезпечення - була представлена Apple у 2014 році для розробки додатків iOS, watchOS, tvOS, macOS як логічна заміна Objective-C, який мав недоліки та був дещо застарілим. Хоча Swift продовжує деякі концепції Objective-C, такі як розширювальне (extensible) програмування, цей підхід відрізняється протокол-орієнтованим дизайном і статичним типом типів.

Згідно з індексом Tiobe [5], який показує, яку мову програмування варто використовувати при запуску нового проекту, Swift знаходиться на 12-му місці. Swift також може похвалитися багатьма зручними інструментами сторонніх розробників і стає все більш популярним вибором, особливо для невеликих програм і стартапів.

Мова Swift, яку часто називають «Objective-C, без C», у багатьох аспектах перевершує свого попередника. Згідно з офіційним прес-релізом [6], «Swift поєднує продуктивність та ефективність компільованих мов із простотою та інтерактивністю популярних мов».

Нижче наведені основні переваги мови Swift:

a) Швидкий та оптимальний процес розробки

   Чиста й виразна мова зі спрощеним синтаксисом та граматикою, Swift легше читати й писати. Відповідно, для створення додатків iOS за допомогою Swift зазвичай потрібно менше часу. Зосереджена на

продуктивності та швидкості мова спочатку була розроблена, щоб перевершити свою попередницю: початковий реліз стверджував, що продуктивність збільшилася на 40 відсотків у порівнянні з Objective-C. Крім того, Swift був побудований за допомогою компілятора LLVM, який перекладає мову асемблера в машинний код і оптимізує код, прискорюючи розробку.

Яскравим прикладом цієї переваги є додаток Lyft: компанія повністю переписала свій додаток для iOS за допомогою Swift. У той час як стара кодова база складалася з близько 75 000 рядків коду, версія Swift відтворила ту саму функціональність з менш ніж третиною цього.

b) Зменшення обсягу пам'яті

Коли ви створюєте програму, ви використовуєте багато стороннього коду – часто відкритих фреймворків або бібліотек, скомпільованих у код вашої програми. Ці бібліотеки можуть бути статичними або динамічними. Статичні бібліотеки заблоковані в коді під час їх компіляції, тобто вони стають частиною вашого файлу виконання, таким чином збільшуючи його розмір і час завантаження. З іншого боку, динамічні бібліотеки існують за межами вашого коду і завантажуються лише за потреби.

c) Автоматичне управління пам'яттю за допомогою ARC

Swift використовує автоматичний підрахунок посилань (Automatic Reference Count) – технологію, спрямовану на додавання функції збирання сміття, яка раніше не була представлена в iOS. Автоматичний підрахунок посилань визначає, які екземпляри більше не

використовуються, і позбавляється від них від вашого імені. Це дозволяє підвищити продуктивність програми без затримки пам’яті чи ЦП.

d) Велика спільнота та відкритий код мови

Як зазначив старший віце-президент Apple з розробки програмного забезпечення Крейг Федерігі: «Щоб стати основною мовою для наступних 20 років програмування в нашій галузі, ми бачили можливість open-source як важливого елементу, який допоможе Swift досягти свого потенціалу».

Дійсно, завдяки потужній корпоративній підтримці з боку Apple та IBM, Swift швидко завоювала одне з найактивніших та найактивніших спільнот із відкритим кодом. Тенденції прийняття, про які говорилося раніше, є яскравим прикладом. Крім того, Swift входить до п’ятірки найпопулярніших мов на GitHub, після Go, TypeScript і Rust (рис. 3).

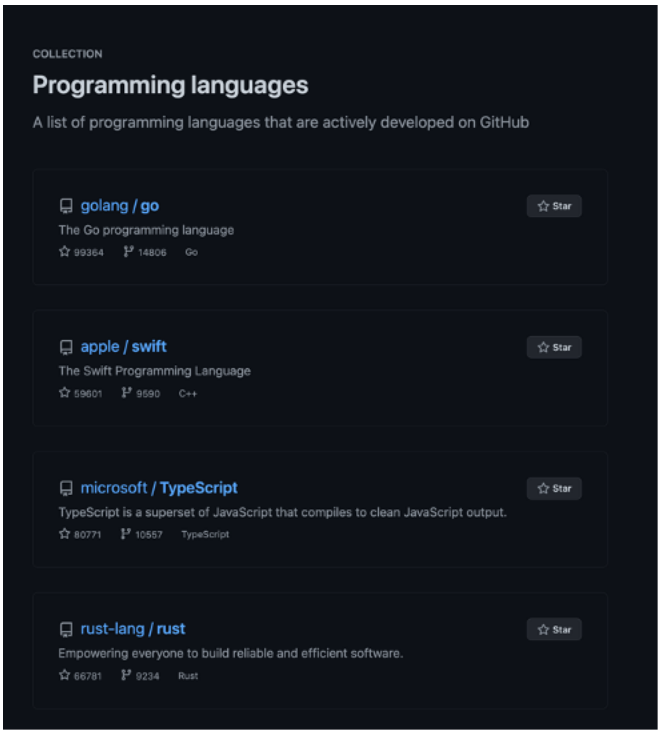

Рисунок 3 – Популярність мови Swift на сайті GitHub

1.5 Вибір архітектури додатку

Шаблони проектування завжди допомагали у створенні керованого, тестованого, багаторазового та оптимізованого програмного забезпечення. Як правило, це допомагає модулювати програмне забезпечення так, що кожен компонент є окремим і несе єдину відповідальність. Крім того, вони значно покращують читабельність коду, що відіграє велику роль у передачі програмного коду. Крім того, процес розробки програмного забезпечення різко пришвидшується завдяки вже перевіреним парадигмам проектування. Спочатку мобільні додатки були занадто малі, щоб відповідати дизайну або архітектурним шаблонам, і тому вони звикли дотримуватись найпростіших.

У наш час, оскільки мобільні додатки стають все більшими необхідно розглянути шаблони дизайну, перш ніж переходити у режим розробки.

1.5.1 MVC (Movie – View – Controller)

Почнемо з найпростішого і найуживанішого. Apple через свої зразки кодів завжди рекомендувала слідувати MVC – Model, View, Controller. Але форма Apple MVC дещо змінена, щоб краще адаптуватися до мобільних додатків. Загальна його форма представлена таким чином (рис. 4).

В основному він складається з представлення, контролера та моделі. Представлення можна уявити як інтерфейс користувача, який відображається йому в певний момент часу. Модель – це дані, які відображаються в компонентах Представлення. Контролер є містком між ними. Контролер володіє представленням і пов’язує з ним модель.

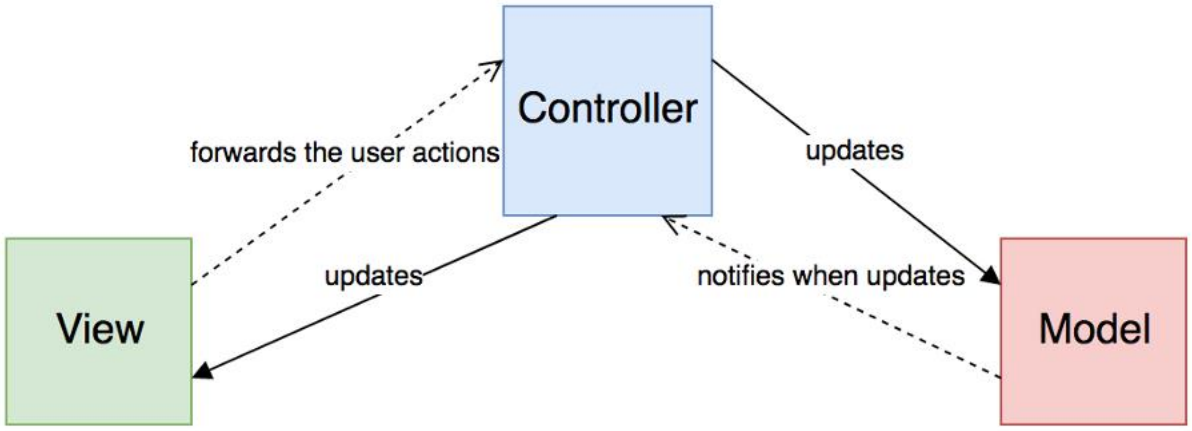

Рисунок 4 – Шаблон проектування MVC

Представлення несе відповідальність за показ користувачам свого інтерфейсу користувача. Зрештою, представлення передає дії, які виникли з інтерфейсу користувача, до контролера. Дії або ініційовані користувачем,

наприклад, користувач може натиснути на кнопку на екрані, щоб розпочати дію, або, які автоматично генеруються, наприклад, може завчасно оновлюватися вміст екрану. Відповідальність контролера в основному полягає в тому, щоб отримувати ці дії від Представлення і виконувати їх. Контролер під час обробки може оновлювати Модель. Іноді цей процес оновлення займає деякий час. Модель повідомляє Контролера, коли він завершує процес оновлення, після чого Представлення отримує сповіщення від Контролера про оновлення свого інтерфейсу користувача оновленими даними. Представлення і Модель ніколи не спілкуються один з одним напряму - вони спілкуються через контролер. Це єдина відмінність, якою володіє Apple MVC в порівнянні з оригінальним MVC.

На перший погляд, все виглядає достатньо просто на зрозуміло, але ця простата приносить деякі проблеми і труднощі. Раніше з цими проблемами не стикалися, поки мобільні додатки не стали коротшими та простішими. Але тепер, коли пристрої iOS є потужнішими, ніж настільні комп’ютери, користувачі очікують, що мобільні програми будуть достатньо потужними, щоб повністю використовувати ресурси пристрою. Це призводить до проблеми збільшення кодової бази проекту, а звідти вже випливає проблема керування всіма компонентами.

По-перше, проблема, яку викликає MVC, полягає в тому, що він не дотримується парадигми єдиної відповідальності і, як наслідок, викликає MVC – Massive View Controller. Відповідно до MVC, контролер є посередником між представленням і моделлю. Це змушує контролера виконувати численні обов’язки. Очевидно, код стає дуже складним для

управління та розуміння. Ця проблема трохи вирішена, коли концепція дочірніх контролерів була введена в iOS.

По-друге, MVC ускладнює тестування модульних тестів. Завдяки прямому зв’язку Контролер-Представлення, потрібно налаштувати все таким чином, щоб виконувалася лише бізнес-логіка для створення дійсного результату тесту.

MVC як і раніше добре працює в багатьох випадках, коли користувальницький інтерфейс простіший, а контролер повинен нести менше відповідальності, але він з тріском дає збій для складних інтерфейсів користувача.

1.5.2 MVP (Model – View – Presenter)

MVP — це розширення MVC Apple, де відповідальність контролера перегляду розподіляється між Представленням та Пред’явником (рис. 5). Він розглядає контролер як представлення. Представлення в свою чергу зберігає відповідальність за обробку інтерфейсу користувача, тоді як Пред’явник займається фактичною бізнес-логікою. Таким чином модульне тестування спрощується, оскільки всі модульні тести записуються через Пред’явника, де обробка, пов’язана з переглядом, недоступна. У певному сенсі це вигідно у порівняні з MVC від Apple, він добре розподіляє відповідальність, а тестування модульних тестів є менш стомлюючою роботою. Однак це зменшує швидкість розробки, оскільки впровадження Пред’явника і зв’язування його між шарами вимагає додаткової роботи.

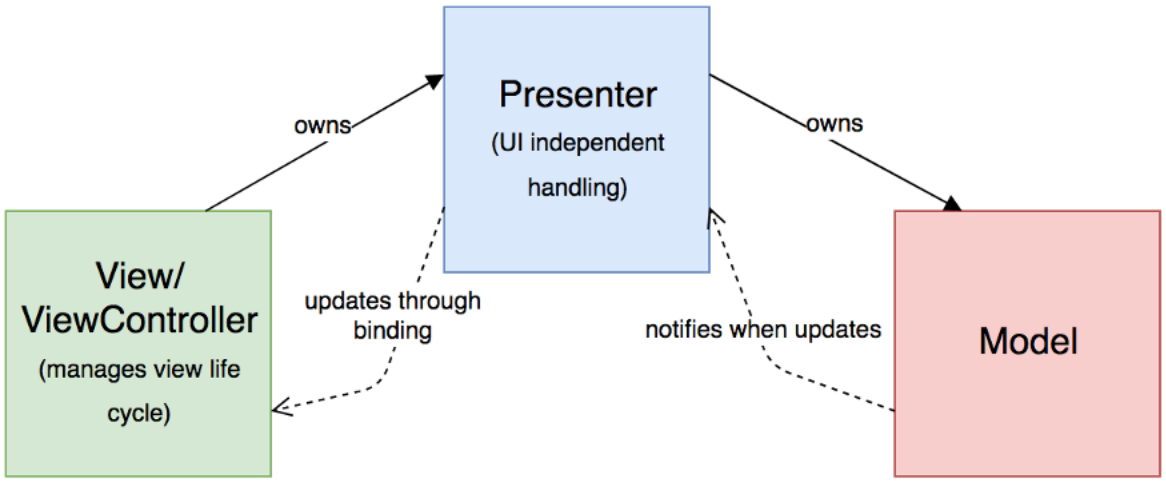

Рисунок 5 – Шаблон проектування MVP

Отже, як висновок було обрано шаблон проектування MVP, який є удосконаленням класичного шаблону MVC, який дозволяє відокремлювати логіку наших Представлень та Контролерів, що полегшує тестування нашого додатку та дозволяє вносити значні зміни у логіці, не переписуючи всі інші класи.

РОЗДІЛ 2 ОСОБЛИВОСТІ РОЗРОБКИ

При створенні мобільного додатку важливо переконатися, що кожен компонент виконує свою функцію. Невелика проблема на етапі створення архітектури для мобільного додатка може мати далекосяжні наслідки для життєздатності кінцевого продукту. Популярні мобільні додатки розуміють цей принцип – вони створені з урахуванням стабільності та функціональності та щодня демонструють ці концепції. Додатки, які не враховують архітектуру розробки мобільних додатків у своєму плануванні, як правило, не живуть довго в жодному з магазинів додатків, оскільки користувачі швидко демонструють своє невдоволення. Також важливим етапом планування та розробки є доступний та зрозумілий інтерфейс, адже користувачі оцінюють додаток в першу ж хвилину, саме це є вирішальним моментом – чи буде він надалі користуватися ним чи піде шукати інший, більш зручний додаток для використання.

2.1 Розробка структури додатку

Архітектура мобільного додатка — це набір різних структурних елементів та їхніх інтерфейсів. На їх основі можна створювати добре структуровані та зручні мобільні додатки. Перш ніж приступити до написання коду, важливо отримати чітке уявлення про те, які функції повинні бути вбудовані у вашу програму, щоб вона відповідала потребам клієнтів. Після

того, як вони чітко визначені, ми можемо створити мобільну архітектуру для свого додатка.

Усі ресурси, вкладені в створення бездоганної мобільної архітектури, неодмінно окупляться в майбутньому. Ми зможемо ефективно усувати неполадки та легко масштабувати програму Такі принципи розробки програмного забезпечення, як SOLID, KISS (будь простим, дурним) і DRY (не повторюйся), слід враховувати при розробці архітектури програми, щоб уникнути проблем у майбутньому та досягти найкращих результатів.

Крім того, необхідно дотримуватися типу архітектури Clean [7]. Існує два основних принципи чистої архітектури: правило залежності і принцип абстракції. По-перше, правило залежності визначає, що кожен рівень програми не залежить від інших шарів або зовнішньої програми. По-друге, згідно з принципом абстракції, внутрішній рівень має справу з бізнес-логікою, тоді як зовнішній рівень містить деталі реалізації.

Підбиваючи підсумки перерахованого вище, при розробці мобільного додатку було дотримано усіх вимог.

По-перше, виконано правило залежності - кожен рівень програми не залежить від інших шарів або зовнішньої програми. Як описувалось у пункті 1.5 «Вибір архітектури додатку», мною було обрано шаблон MVP. Кожна складова відповідає за свою роботу:

- Модель відповідає за зберігання даних, які будуть відображатися, в моєму випадку це були Фільми, Актори та Фільми за результатами пошуку, які зберігалися внаслідок запиту до API The Movie Database або з бази даних Realm.

- Представлення відповідає за показ інтерфейсу користувачу та подальшу обробку подій, пов’язаних з діями користувача. Саме представлення відповідає за відображення переліку категорій фільмів, популярних акторів, пошукові запити на екрані «Пошуку фільмів» та навіть за відображення обраних фільмів на екрані «Подивитися пізніше»
- Пред’явник є відповідальним за оновлення даних на Представленні або за навігацію – перехід між екранами, якщо користувач захоче перейти до іншої секції.

По-друге, враховувалися розміри усіх можливих пристроїв та можливість відсутності мережі Інтернет, розроблені відповідні механізми, які запобігали б виникненню незрозумілої поведінки додатку. Навіть якщо користувачі мають хорошу швидкість Інтернету більшість часу, ми повинні бути готові до найгірших сценаріїв. У деяких випадках їх підключення може бути переривчастим, і ми повинні визнати це під час розробки діаграми архітектури мобільного додатка.

2.2 Розробка програмного інтерфейсу додатку

Доступний та зрозумілий інтерфейс для користувача – запорука успіху будь-якого мобільного додатку. Користувач повинен відкрити додаток і одразу зрозуміти, яку проблему хотів вирішити розробник, щоб зробити життя користувача легше.

Завдяки детальній та поширеній документації компанії Apple [8] – Human Interface Guidelines, розробляти доступні інтерфейси стало набагато легше, адже ти одразу розумієш, яких критеріїв ти повинен дотримуватися.

Додаток «Movie Library» було розроблено з використанням компонентів UIKit, фреймворку, що визначає загальні елементи інтерфейсу. Елементи UIKit гнучкі та знайомі. Вони адаптовані, що дає змогу створити єдиний додаток, який чудово виглядає на будь-якому пристрої iOS, і вони автоматично оновлюються, коли система вносить зміни у зовнішній вигляд. Елементи інтерфейсу, надані UIKit, поділяються на три основні категорії:

- Панелі. З їх допомогою можна вказувати користувачу, де він знаходяться у нашому додатку, який забезпечує навігацію, та може містити кнопки чи інші елементи для ініціювання дій та передачі інформації.
- Представлення. Містять основний вміст, який користувачі бачать у програмі, наприклад текст, графіку, анімацію та інтерактивні елементи.
- Елементи керування. Вони ініціюють дії та передають інформацію. Прикладами елементів керування є кнопки, перемикачі, текстові поля та індикатори прогресу.

Одним із важливих аспектів розробки мобільного додатку є підтримка двох видів інтерфейсу – темної та світлої теми мобільного телефону (рис. 6). Адже більшість досліджень доводить, що використання темної теми менш шкодить нашому зору, адже деяким людям з вадами зору краще працювати в темному режимі, але ще є і такий фактор, як банальна зручність у використанні.

Нещодавнє дослідження [9] показало, що ми дійсно заощаджуємо багато енергії, лише якщо перемикаєтеся зі світлового режиму на 100% яскравості. Дослідники виявили, що перемикання в темний режим дає від 3% до 9% економії. Ще одна корисність темного режиму, про яку ми так часто чуємо, полягає в тому, що він відсікає шкідливе синє світло. Синє світло - це високоенергетичний спектр видимого світла з найкоротшою довжиною хвилі. Найбільшим природним джерелом синього світла для людини є Сонце, але наші телефони також випромінюють невелику кількість синього світла. Згідно з дослідженням Harvard Health, надмірне вплив синього світла може пригнічувати вироблення меланіну нашим організмом, гормону, необхідного для повноцінного сну вночі [10].

При побудові інтерфейсу користувача використовувалася сучасна API, надана компанією Apple, яка була вперше представлена на Всесвітній

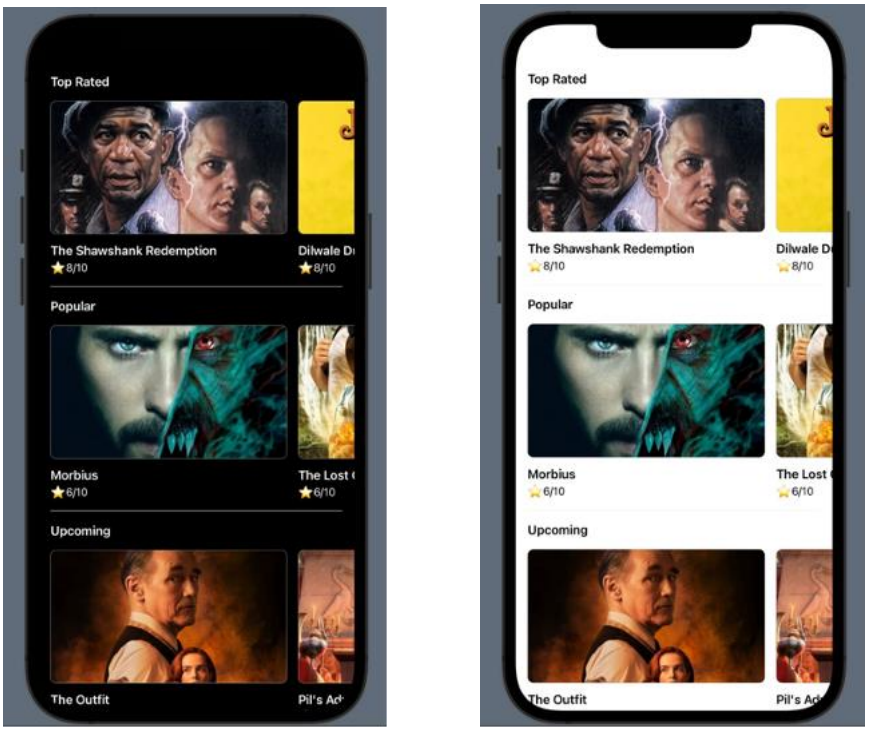

Рисунок 6 – Дві теми мобільного додатку:

а – темна; б - світла

конференції розробників Apple (WWDC) – UICompositionalLayout. Його попередник UICollectionFlowLayout досить добре працює з простими макетами, але оскільки проекти стають все більш неоднорідними, виникає потреба у створенні власних макетів, що має свою частку проблем. Шаблонний код і проблеми, пов’язані з перебудовою осередків – це лише кілька проблем, які роблять створення розширених проектів громіздким. Але представлене API спрощує процес розробки складних макетів у програмах розробників.

UICompositionalLayout — це декларативний тип API, який дозволяє нам створювати великі макети, з’єднуючи менші групи макетів [11]. Композиційні макети мають ієрархію, яка складається з: елементу, групи, розділу та секції. Цей інструмент допомагає розробляти різні види колекцій або таблиць, в

залежності від нашої потреби. З його допомогою, наприклад, були створені такі елементу інтерфейсу, як колекція популярних акторів:

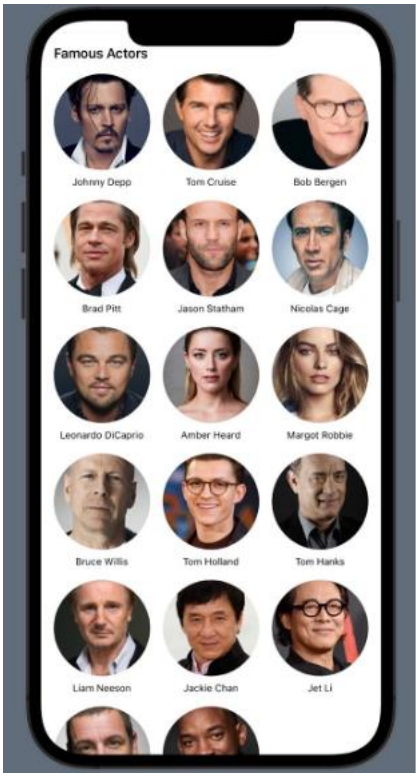

Рисунок 7 – Вигляд інтерфейсу, розробленого за допомогою UICompositionalLayout

Також важливо відмітити, що розробка усього інтерфейсу відбувалася програмним шляхом. Інтегрована середа розробки Xcode надає зручний інструмент побудови UI – Interface Builder – за допомогою якого можна будувати Представлення, просто перетягуючи вікна, кнопки, текстові поля та інші об’єкти на полотно дизайну, яке має назву Storyboard, щоб створювати функціонуючий інтерфейс користувача. Але цей підхід має певний перечень недоліків, а саме:

- Не підтримує анімації, адже їх можна реалізувати тільки програмним шляхом;

- Важко переглядати властивості UI елементів, що викликає певні складності при пошуку помилок чи інших проблем;
- Затримки у виконанні програмі, якщо Storyboard погано організований;
- При роботі більше, ніж одного розробника виникають проблеми при роботі з одними і тими самими екранами, адже представлення Storyboard – це XML файл, який змінює свої властивості від кожної зміни.

З появою нового декларативного фреймворку SwiftUI виникла можливість попереднього перегляду Представлення, над яким ведеться робота, що полегшує роботу з інтерфейсом користувача та пришвидшує розробку, адже можна одразу побачити, який в тебе прогрес, та чи виникли якісь помилки на етапі проектування інтерфейсу. Раніше без цього інструменту доводилися кожен раз компілювати програму, що займало достатню кількість часу через велику кількість елементів та екранів.

2.3 Робота та використання API сервісу The Movie Database

Для роботи з API було обрано базу даних The Movie Database [12]. Вирішальними факторами вибору саме цією бази даних були:

- Велика кількість даних. Починаючи з 2008 року, кількість даних у базі зростає з кожним днем, оскільки її використовує більше 400 000 розробників і компаній, вона є чудовим джерелом даних.
- Зображення чіткої якості. Вона пропонує одну із найкращих добірок зображень високої якості, що є ваговим фактором у виборі API, адже

користувач не повинен отримувати дискомфорт від перегляду зображень.

- Надійність платформи. Кожен день цим сервісом користуються мільйони людей, а обробляється понад 3-х мільярдів запитів, тому можна бути впевненим у надійності та безпекою даних, що є одним із вагомих факторів у розробці iOS-додатку.

Для використання цього API потрібно зареєструватися на сервісі, адже для виконання запитів у кожного розробника повинен бути свій власний api-key (зашифрований, у вигляді рядка ключ, який є унікальним для кожного розробника). Після отримання ключа, треба було ознайомитися з документацією, яка, на щастя, є дуже докладною, з відповідними прикладами та JSON-відповідями.

Для подальшої комфортної роботи було використано додаток Postman, який дозволяє робити REST-запити. Перевагою використання саме цього інструменту був дружній інтерфейс, легкість у використанні та зручний вивід JSON-відповідей. Адже обізнаність у форматі відповіді відіграє основну роль у розробці інтернет-менеджера, який буде виконувати запити та обробляти відповіді у потрібній нам формі.

Розглянемо основні базові виклики нашого менеджера, які будуть необхідні в нашому додатку:

1. Метод GET, звертання на /movie/(category) – повертає масив фільмів, вибраної категорії, або повертає помилку з докладним описом.

2. Метод GET, звертання на /movie/(id_of_movie) – повертає фільм та детальну інформацію за його ID, або повертає помилку з докладним описом.

3. Метод GET, звертання на /movie/trending/person/week – повертає масив популярних за тиждень людей з інформацію про них, або повертає помилку з докладним описом.

Багато завдань програмування включають надсилання даних через мережеве з’єднання, збереження даних на диск або передачу даних до API та служб. Ці завдання часто вимагають кодування та декодування даних у проміжний формат та з нього під час передачі даних. Стандартна бібліотека Swift визначає стандартизований підхід до кодування та декодування даних. Для цього ми використаємо протокол Decodable, який дозволяє кодувати та декодувати наші користувацькі типи.

Кожна наш модель буде використовувати протокол Decodable, адже при його використанні нам потрібно лише ознайомитися зі структурою JSON-запиту, щоб правильно створити назвати змінні, дані яких нам будуть потрібні для подальшої роботи. Для довгих назв в нас створений власний JSON-Decoder, з відповідної decoding-стратегією, а саме конвертація зі «снейк-кейсу». Надалі наші моделі будуть розуміти, як їм правильно конвертуватися з JSON-об’єкта в Swift модель, і нам залишається лише правильно зробити запит до сервісу The Movie Database. Для цього існує клас URLSession, який відповідає за обробку API-запитів, який ми можемо налаштовувати за власним розсудом.

2.4 Використання бази даних Realm

Робити кожен раз Networking-запити – не є хорошою практикою, тому потрібно десь зберігати дані, щоб можна було швидко звертатися до них. Тому один із аспектів, який ми повинні з'ясувати, — це як зберігати та шукати великі обсяги даних. Ми, ймовірно, будемо використовувати базу даних. Найпоширенішими варіантами баз даних iOS є SQLite і CoreData, і відносно новіша БД під назвою Realm. У порівнянні з іншими аналогами Realm є дуже швидкою та легкою, ніж, наприклад, CoreData, яка має велику кількість нюансів, які треба враховувати, плюс використовує більше пам'яті та займає більше місця [13].

Підключається Realm за допомогою SPM (Swift Package Manager) — це інструмент для керування розповсюдженням вихідного коду, спрямований на те, щоб спростити використовувати сторонніх бібліотек. Інструмент безпосередньо вирішує проблеми компіляції та зв'язування пакетів Swift, керування залежностями, керування версіями та підтримку гнучких моделей розповсюдження та співпраці.

Ось ключові моменти, які варто пам'ятати, працюючи з цією базою даних:

a) Realm: екземпляри Realm є серцевиною нашого фреймворку. Це наша точка доступу до базової бази даних, як-от контекст керованого об'єкта CoreData. Ми створюємо екземпляри за допомогою ініціалізатора Realm().

b) Об'єкт: Це наша модель Realm. Щоб створити модель, ми створюємо підклас Object і визначаємо поля, які ми хочемо зберегти як властивості за

допомогою мітки @objc та ключого слово dynamic. Наприклад, ми використовуємо:

```
class Movie: Object {
    @objc dynamic name: String
    @objc dynamic year: Int
}
```

c) Зв’язки: ми створюємо зв’язки «один до багатьох» між об’єктами, оголошуючи властивість типу об’єкта, на який ми хочемо посилатися. Ми можемо створювати зв’язки «багато до одного» та «багато до багатьох» за допомогою властивості типу List.

d) Транзакції запису: будь-які операції в базі даних, наприклад створення, редагування або видалення об’єктів, повинні виконуватися в межах запису за допомогою виклику write(_:) для екземплярів Realm.

e) Запити: для отримання об’єктів із бази даних ми використовуємо запити. Найпростіша форма запиту — це виклик objects() для екземпляра Realm і передача в клас об’єкта, який ми шукаємо. Якщо наші потреби в пошуку даних складніші, ми можемо використовувати предикати, поєднувати наші запитів і впорядковувати результати.

f) Результати. Результати – це тип контейнера, який автоматично оновлюється, який ми отримуєте із запитів об’єктів. Вони мають багато подібності зі звичайними масивами, включаючи синтаксис індексу.

Realm включає в себе Realm Browser для читання та редагування баз даних. Формат бази даних Realm є зрозумілим і читабельним без додаткових засобів.

Використовуючи такий інструмент, як Realm, ми реалізували зберігання улюблених фільмів користувача, які відображаються на окремому екрані «Watch Later» (рис.8).

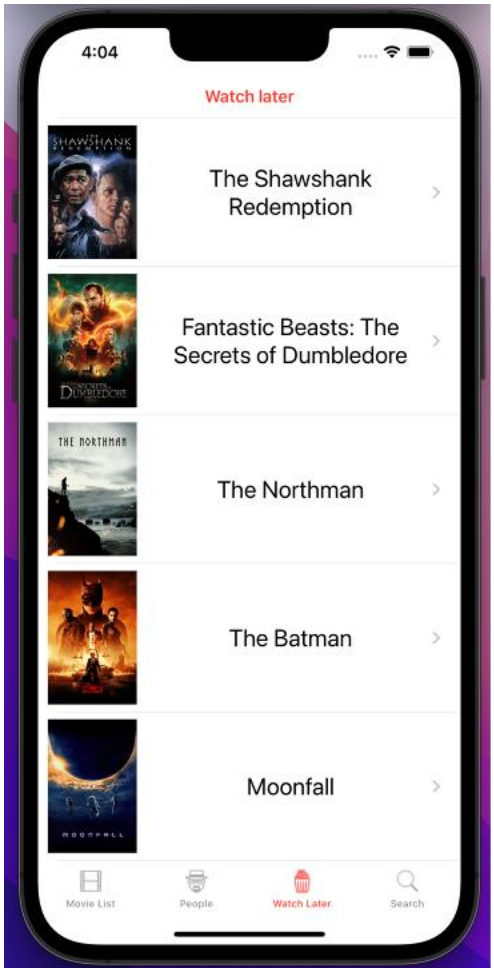

Рисунок 8 – Вигляд екрану улюблених фільмів користувача

2.5 Опис сценаріїв додатку

Для опису сценаріїв, за якими працює мобільний додаток «Movie Library», було створено мережу Петрі (рис. 9). Завдяки їй можна побачити поведінку та опис дії користувача в залежності від його подальших дій [14].

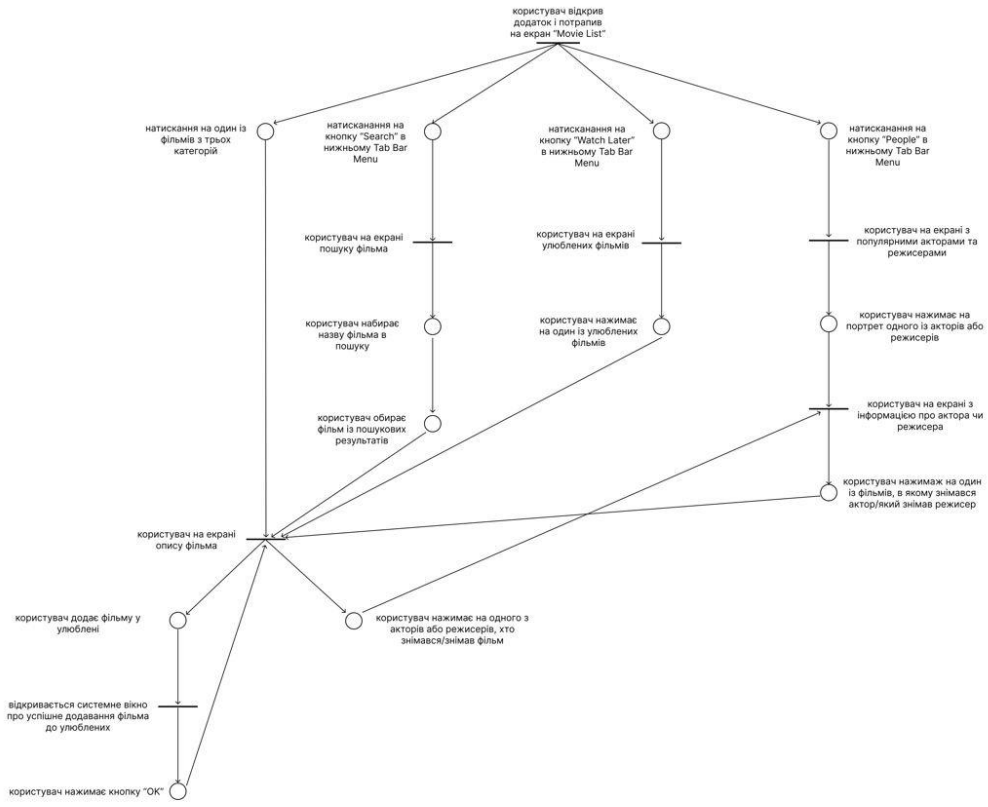

Рисунок 9 – Побудована мережа Петрі для опису сценарію додатку

Як можна побачити на малюнку, спочатку користувач опиняється на першому екрані «Movie List», де знаходяться три каруселі з наборами фільмів. При натисканні на будь-який із фільмів, користувач переходить на екран із

детальною інформацією про фільм, де знаходяться коротка інформація про нього, у вигляді опису, року виходу, рейтингу та акторського складу. Користувач може додати фільм у улюблені за допомогою кнопки «+» у правому верхньому куті. Також при натисканні на портрет одного із акторів, користувач перейде на екран з описом актора, а також списку найпопулярніших фільмів, у яких він знімався. При натисканні на фільм, користувач переходить на екран з детальною інформацією про обраний фільм.

Також у нижній частині екрану знаходиться Tab Bar Menu, за допомогою якого користувач може переходити між основними екранами в будь-який момент.

При натисканні на кнопку «People», користувач опиняється на екрану з найпопулярнішими акторами за тиждень (рис. 7), натискання на портрети яких веде на екран з детальною інформацією про них.

При натисканні на кнопку «Watch List», користувач опиняється на екрані обраних фільмів (рис. 8), які попередньо були додані сюди з екрану детального опису фільму. Користувач може видаляти їх за допомогою свайпа вліво.

При натисканні на кнопку «Search» користувач переходить на екран пошуку фільмів (рис. 10). Згори знаходиться сам пошук, який видає запити в залежності від тексту, набраного користувачем у текстовому полі. При натисканні на результат пошуку переходимо на екран з детальною інформацією про фільм.

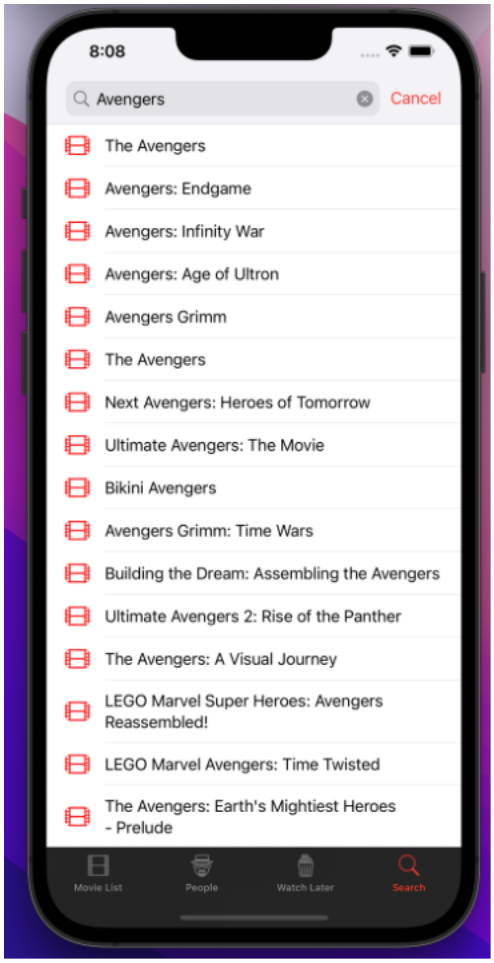

Рисунок 10 – Екран пошуку з результати пошуку

ВИСНОВКИ

У роботі був представлений опис відмінностей розробки мобільного додатку на різних платформах, було розібрано переваги та недоліки кожного із них, а також аргументовано вибір розробки «нативним» методом. Проаналізовано основні відмінності розробки мовою Swift та описано ключові переваги перед іншими мовами, які мають суттєвий характер.

Досліджено реалізовані рішення актуальної проблеми, за результатами було виявлено переваги, які має включати майбутній додаток, а також недоліки, які слід уникнути при розробці.

Було аргументовано вибір поточної архітектури додатку, проведено порівняння з більш популярним аналогом, описані плюси та мінуси використання обох шаблонів.

В результаті було розроблено мобільний додаток на платформі iOS, що реалізовує зручний перегляд та пошук фільмів, які цікавлять користувача. Детально описані основні етапи створення додатку, зокрема побудова інтерфейсу користувача, були розглянуті основні акценти, на які варто звернути увагу при розробці. Також були представлені технології, які використовуються для створення додатку, та обґрунтовано, чому було обрано саме їх.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 6-month neurological and psychiatric outcomes in 236 379 survivors of COVID-19: a retrospective cohort study using electronic health records [Електронний ресурс]. – Режим доступу: URL: – https://www.thelancet.com/journals/lanpsy/article/PIIS2215-0366(21)00084-5/fulltext – Назва з екрану.
2. How watching movies can benefit our mental health [Електронний ресурс]. – Режим доступу: URL: – https://psychcentral.com/blog/how-watching-movies-can-benefit-our-mental-health - Назва з екрану.
3. Global App Revenue Growth Was Flat in Q1 2022, While Usage Grew Nearly 5% [Електронний ресурс]. Режим доступу: URL: – https://sensortower.com/blog/app-revenue-and-downloads-q1-2022 – Назва з екрана.
4. Pros and Cons of Mobile Development with Xamarin [Електронний ресурс]. Режим доступу: URL: – https://www.iflexion.com/blog/xamarin-pros-and-cons – Назва з екрану.
5. TIOBE Index for May 2022 [Електронний ресурс]. – Режим доступу: URL: – https://www.tiobe.com/tiobe-index/ – Назва з екрана.
6. Apple Releases iOS 8 SDK with Over 4000 New APIs [Електронний ресурс]. Режим доступу: URL: – https://www.apple.com/newsroom/2014/06/02Apple-Releases-iOS-8-SDK-With-Over-4-000-New-APIs/ – Назва з екрана.
7. Robert C. Martin Clean Architecture: A Craftsman’s Guide to Software Structure and Design [Електронний ресурс]. / Pearson; 1st edition 2017 – 154 с.

8. Human Interface Guidelines [Електронний ресурс]. Режим доступу: URL: - https://developer.apple.com/design/human-interface-guidelines/ – Назва з екрана.
9. Dark mode might not be a silver bullet for improved battery life [Електронний ресурс]. Режим доступу: URL: – https://www.androidauthority.com/dark-mode-study-battery-2026759/ – Назва з екрана.
10. Blue light has a dark side [Електронний доступ]. Режим доступу: URL: - https://www.health.harvard.edu/staying-healthy/blue-light-has-a-dark-side - Назва з екрана.
11. UICollectionViewCompositionalLayout [Електронний ресурс]. – Режим доступу: URL: – https://developer.apple.com/documentation/uikit/uicollectionviewcompositionallayout – Назва з екрану.
12. The Movie Database API [Електронний ресурс]. – Режим доступу: URL: – https://developers.themoviedb.org/3/getting-started/introduction – Назва з екрану.
13. Todorov M. Building Modern Swift Apps with Realm Database [Електронний ресурс]. / M. Todorov. – 2018 – 10 с.
14. Стеценко І.В. Моделювання систем: навч. Посібник. [Електронний ресурс, текст] / І. В. Стеценко. – М-во освіти і науки України, Черкас. держ. технол. ун-т – Черкаси: ЧДТУ, 2010 – 100 с.