Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем

# РОЗРОБКА IOS-ДОДАТКУ З ВИКОРИСТАННЯМ БАГАТОПОТОЧНОСТІ

**Текстова частина до курсової роботи**
**за спеціальністю «Інженерія програмного забезпечення»**

Керівник курсової роботи
ст. викладач Борозенний С. О.

_____
*(підпис)*
“____” _____ 2021 р.

Виконав студент 4 р. н.
Мокрий М. В.
“____” _____ 2021 р.

Київ – 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем

доцент, к.ф.-м.н.

О. П. Жежерун

_____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Мокрому Михайлу Вікторовичу 4-го курсу факультету інформатики

ТЕМА Розробка iOS-додатку з використанням багатопоточності

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

1 Аналіз предметної області

2 Проектування архітектури додатку

3 Огляд API та знайомство з багатопоточністю

4 Розробка додатку

Висновки

Список літератури

Дата видачі „___” _____ 2021 р. Керівник _____

Завдання отримав _____

**Тема:** Розробка iOS-додатку з використанням багатопоточності

**Календарний план виконання роботи:**

| № п/п | Назва етапу | Термін виконання | Примітка |
|---|---|---|---|
| 1. | Отримання завдання на курсову роботу. | 09.11.2020 | |
| 2. | Аналіз предметної області. | 28.11.2020 | |
| 3. | Знайомство з багатопоточним програмуванням та вибір API. | Грудень 2020 | |
| 3. | Проектування архітектури додатку. | Січень 2021 | |
| 4. | Розробка додатку. | Лютий 2021 | |
| 5. | Написання текстової частини роботи. | Березень 2021 | |
| 6. | Створення презентації та написання доповіді. | 01.04.2021 | |
| 7. | Захист курсової роботи. | 19.04.2021 | |

Студент Мокрий М. В.

Керівник Борозенний О. С.

“_____”_____

# Зміст

# Анотація

Курсова робота присвячена створенню мобільного додатку на базі операційної системи iOS 13 з використанням елементів багатопоточного програмування. Для його розробки була використана мова програмування Swift 5, інтегроване середовище розробки Xcode, менеджер залежностей Swift Package Manager, архітектура шаблону MVVM, Jikan API, а також бібліотеки з кодом у відкритому доступі SDWebImage та SDWebImageSwiftUI.

Основні функції системи: перегляд списків аніме відсортованих за різними критеріями та додаткової інформації про кожне з них, можливість додавання вибраного аніме до списку обраних.

Ключові слова: iOS, Xcode, Swift, Apple, API, JSON, iPhone, SPM, MVVM, мобільний додаток, багатопоточність.

# Вступ

Сучасний світ постійно розвивається, як і розвиваються мобільні технології. У сьогоденні неможливо уявити людину, яка б не використовувала мобільного пристрою. Він слугує для розв’язання багатьох задач. Але що саме робить цей мобільний пристрій таким потрібним і популярним?

На мою думку, відповідь на це питання полягає у існуванні широкого спектра мобільних додатків, які постійно допомагають користувачеві в вирішенні будь-яких ситуацій. Саме тому кількість мобільних додатків просто величезна. Але не всі вони є корисними кінцевому користувачу, багато з них просто копіюють інші додатки, не надаючи ніяких нових функцій, а деякі просто існують задля того, щоб заробляти гроші на показі постійної реклами всередині додатку.

Метою курсової роботи є створення сучасного мобільного застосунку, використовуючи технології і методи, прийняті в колі iOS-розробників. Він повинен мати інтуїтивно зрозумілий інтерфейс, бути зручним у використанні та виконувати поставлені перед ним функції. Основними завданнями даного дослідження є:

- аналіз предметної області та створення функціональних вимог;
- вибір архітектури та її реалізація в самому додатку;
- опис обраного API;
- дослідження багатопоточності;
- створення мобільного застосунку для операційної системи iOS 13.

Об’єктом дослідження є процес розробки мобільного додатку з використанням багатопоточності.

Предметом дослідження є сам мобільний застосунок.

У цій роботі висвітлюється тематика, яка популяризує японську анімаційну культуру. Застосунок створений згідно функціональних вимог і відповідає критеріям сучасного застосунку, а також використовує методи багатопоточності і правильні архітектурні рішення у своєму проектуванні.

Робота складається з п’яти розділів.

Перший розділ призначений для аналізу предметної області і визначення основних функціональних вимог, згідно яких буде будуватися функціонал додатку.

Другий розділ присвячений дослідженню основних методів проектування, вибору архітектурного шаблону та його детального опису. Також у розділі розглядаються принципи чистої архітектури.

У третьому розділі обґрунтовується вибір API та описуються основні запити вибраного API, які використовуються безпосередньо в самому мобільному застосунку.

Четвертий розділ присвячений опису можливостей багатопоточного програмування та його безпосередньої реалізації в самому додатку.

В останньому розділі наводиться реалізація графічного інтерфейсу користувача, опис структури класів та їх взаємодія один з одним. Також приділяється увага використаним при розробці додатковим бібліотекам.

У результаті виконання дослідження створено програмний продукт, який призначений для пошуку аніме, відсортованого за різними критеріями, з можливістю його подальшого збереження у список обраних та перегляду детальної інформації про нього.

# 1 Аналіз предметної області

## 1.1 Характеристика предметної області

Обраною темою є створення мобільного застосунку на базі операційної системи iOS з використанням елементів багатопоточності. Додаток представляє з себе декілька різних екранів, на яких відображається списки аніме, відсортованих за різними критеріями, а саме: популярність, рейтинг серіалів, рейтинг фільмів, прем’єра аніме. Користувач може переходити на сторінку з додатковою інформацією про конкретне аніме, дивитися його трейлер та переглядати список усіх персонажів з можливістю більш детального ознайомлення з кожним із них. Також він може формувати особистий список обраних аніме, який відображається на окремому екрані додатку і зберігається в пам’яті телефону, додавати до нього нове аніме та видаляти вже додане. Користувачеві доступний пошук за назвою по цьому списку обраних. У мобільному застосунку використані методи багатопоточності, які дозволяють розподіляти ресурси системи між усіма наявними задачами, що позитивно впливає на роботу графічного інтерфейсу і додатку в цілому.

## 1.2 Функціональні вимоги

Спираючись на предметну область і мету проекту, доречно було б зазначити функціональні вимоги, які були поставлені переді мною при розробці мобільного додатку, а саме:

- Можливість завантажувати дані з мережі, використовуючи API;
- Можливість перегляду списку аніме, відфільтрованого за одним з параметрів;
- Можливість довантажувати аніме зі списку за допомогою пагінації;
- Можливість відображення більш детальної інформації про конкретне аніме;
- Можливість перегляду трейлера конкретного аніме;
- Можливість перегляду списку персонажів конкретного аніме;
- Можливість перегляду більш детальної інформації про конкретного персонажа аніме;
- Можливість додання аніме до списку обраних, або його видалення з цього списку;
- Можливість пошуку аніме по списку обраних;
- Можливість збереження список обраних аніме у пам’ять телефону;
- Можливість відображення декількох екранів зі списками аніме, відфільтрованих по одному з критеріїв, і екран зі списком обраних;
- Можливість автоматичної зміни фону додатку в залежності від обраної системної теми;
- Адаптація під різні розміри екрану.

# 2 Проектування

## 2.1 Архітектура iOS-додатку

У сучасному світі існує багато підходів і патернів, за допомогою яких створюються архітектурні рішення, які дозволяють полегшити розробку і подальшу підтримку мобільних застосунків, а також допомагають розробникам краще орієнтуватися в коді, роблячи його більш читабельним та зрозумілим, розділяючи на різні рівні логіки. У маленьких додатках вибір архітектури, на перший погляд, є не таким важливим, так як логіка таких додатків є інтуїтивно зрозумілою, а кількість функцій, яку він виконує можна перелічити на пальцях. Але проблеми починаються тоді, коли в цьому мобільному застосунку потрібно розширювати функціонал. З невірно підібраною архітектурою, це дуже часто неможливо зробити без переписування значної частини коду і змінення внутрішньої логіки, тому архітектура в маленькому додатку також грає важливу роль. На відміну від додатків маленьких розмірів, у великих додатках зі складною логікою і об’ємним функціоналом ситуація набагато серйозніша. Правильний вибір архітектури є найважливішим чинником при розробці. На цьому етапі закладається фундамент додатку, від якого залежить його успішність. Архітектура повинна робити додаток здатним до розширення, полегшувати його підтримку і тестування та відокремлювати різні рівні логіки.

Саме тому необхідно ретельно продумати архітектуру мобільного додатку на етапі його проектування задля полегшення подальшої розробки. Для цього існують шаблони проектування, перевірені часом і досвідом багатьох розробників. Вони допомагають звести до мінімуму кількість потенційних помилок, з якими може стикнутися програміст під час розробки архітектури мобільного додатку.

## 2.2 Принципи чистої архітектури

Дуже часто недостатньо просто вибрати шаблон проектування і дотримуватися його, особливо в складних, комплексних проектах, перед цим потрібно замислитись про його загальну структуру. Користуючись принципами чистої архітектури[1], описаними Робертом Мартіном в своїй книзі "Чиста архітектура. Мистецтво розроблення програмного забезпечення", можна значно покращити розробку мобільного додатку. Вони надають деякі корисні інструкції і правила, щодо належного розділення компонентів на рівні та зменшення зв’язку між ними. У майбутньому це дасть змогу замінити застарілі частини системи на нові з мінімальними втратами для усієї системи.

Архітектура дає змогу розділити проект на три шари, а саме:

- Шар даних (англ. Data layer).

  Цей шар відповідальний за отримання даних з бази даних, або API, обробку цих даних і їх відображення в відповідних структурах.

- Шар домену (англ. Domain layer).

  Шар, в якому зосереджена бізнес логіка додатку, є проміжним рівнем між шаром даних і шаром представлення. Він повністю незалежний від них, тому може бути повторно використаний в різних проектах.

- Шар представлення (англ. Presentation layer).

  Шар, відповідальний за правильне відображення даних у графічному інтерфейсі додатку, а також взаємодії з користувачем.

Розділення проекту на три шари використовувалось при розробці мого додатку. Це дає можливість згрупувати компоненти по різним шарам в поєднанні з використанням шаблону проектування, що допомагає краще розуміти, підтримувати, тестувати та повторно використовувати написаний код.

## 2.3 Шаблон проектування MVVM

При розробці мобільних додатків на базі iOS використовуються різні шаблони проектування, а саме: MVC, Cocoa MVC, MVP, MVVM, VIPER та інші. Багато шаблонів мають, як свої переваги, так і власні недоліки. З часом архітектурні рішення, використані в деяких шаблонах, застарівають, або виникають нові потреби, які ці шаблони просто не можуть покрити. Вони перестають виконувати своє основне призначення — полегшення розробки мобільних застосунків, створюючи зайві проблеми для розробників. Це зовсім не означає, що ці шаблони потрібно забути, їх все ще можна використовувати в різноманітних проектах, але на зміну їм з'являються нові шаблони, створенні задля того, щоб вирішити всі ті проблеми, присутні у старіших архітектурних шаблонах проектування. До цих нових шаблонів можна віднести MVVM та VIPER.

Для розробки мобільного додатку було вирішено вибрати шаблон MVVM[2], який є одним з найновіших патернів проектування. Його схема проектування зображена на рисунку 2.2.1. Шаблон з'явився для обходу обмежень MVC та MVP та увібрав у себе їхні сильні сторони. Основною його відмінністю від Cocoa MVC, шаблону розробленому компанією Apple, є те, що він відокремлює логіку додатку, зв'язок з моделлю даних, від візуального представлення, що дозволяє змінювати ці компоненти окремо один від одного і покращує підтримку та тестування коду, у той час, як у Cocoa MVC, контролер настільки залучений в життєвий цикл вигляду, що утворює з ним одне ціле. У колі iOS-розробників його навіть жартома називають як Massive View Controller.

MVVM розділяється на три основні компоненти:

- Модель (англ. Model)
- Модель вигляду (англ. View Model)
- Вигляд (англ. View)

Модель відображає дані застосунку, отримані з бази даних, API, або інших джерел. Варто зазначити, що модель просто зберігає дані, не виконуючи жодну бізнес-логіку мобільного додатку.

Модель вигляду — головний компонент архітектури шаблону MVVM. Він пов’язує модель і вигляд через механізм прив’язки даних. Модель вигляду незалежна від самого вигляду, що значно розвантажує його та покращує тестування окремих компонентів. Також в ній може бути зосереджена внутрішня логіка застосунку. Модель вигляду викликає зміни в моделі, самостійно оновлюючи її внутрішні дані, які відображаються у вигляді.

Вигляд показує дані з моделі у графічному інтерфейсі, через який відбувається взаємодія користувача з мобільним додатком. За допомогою моделі вигляду, який є посередником між моделлю та виглядом, які нічого не знають один про одного, цими даними можна маніпулювати. Тобто зв’язок між цими компонентами є мінімальним.

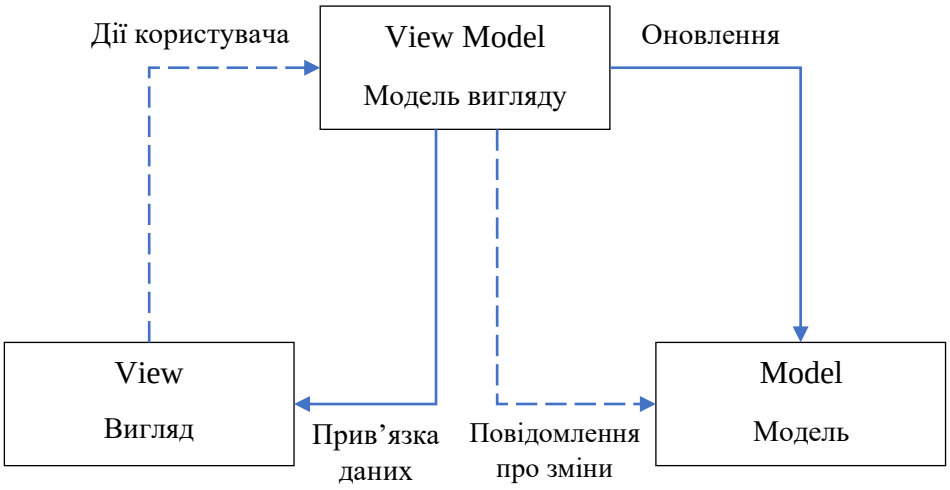


Рисунок 2.2.1. Схема взаємодії в MVVM

Шаблон MVVM містить в собі всі риси хорошої архітектури, а саме простоту використання, можливість тестування та розподіл обов’язків між усіма компонентами. Тому поданий шаблон проектування був використаний при розробці мобільного додатку.

# 3 API

## 3.1 Обґрунтування вибору

Кожен працюючий з даними додаток повинен звідкись ці дані брати. Найчастіше вони надходять або з локальної бази даних, або з зовнішнього API, програмного інтерфейсу додатку, який дозволяє напряму звертатися до віддаленого сервера, отримувати від нього дані та передавати їх.

Мій додаток містить механізми роботи з даними, які ми отримуємо з мережі, їх декодування з формату JSON та перетворення у відповідні структури даних за допомогою протоколу Codable[3], спеціального інструменту для кодування або декодування даних, а також зберігання на диск та передачі по мережі.

Після довгих роздумів було вирішено взяти Jikan API, яке повністю задовольняє функціональні потреби мого додатку, має багатий функціонал і зручну документацію. Він відкриває доступ до великої кількості інформації, містить багато різних типів запитів з параметрами, а також є повністю відкритим, тобто не вимагає авторизації, чи ключа доступу.

Як було зазначено вище, тематикою мого застосунку є японська анімаційна культура. Jikan API дозволяє отримувати список з різноманітними аніме серіалами та фільмами, відсортованими за різними параметрами з пагінацією по сторінкам. Більш детальна інформація про кожне аніме, як і інформація про всіх його персонажів, доступна за іншим запитом з використанням унікального ідентифікатора аніме. Також є можливість отримати детальнішу інформацію про кожного персонажа, знаючи його власний унікальний ідентифікатор. У цьому API реалізовано набагато ширший функціонал, і воно має набагато більше доступних запитів, ніж було взято, але більшість з них були відкинуті на етапі проектування мобільного додатку, тому що не знайшли свого застосування.

## 3.2 Jikan API

На етапі проектування будо вирішено взяти Jikan API, який має багатий функціонал і дозволяє отримати всю необхідну інформацію для мого мобільного додатку. Варто зазначити, що API є повністю відкритим для кожного пересічного користувача, тому воно має обмеження в кількості запитів задля забезпечення захисту від зловживання. В офіційній документації[4] зазначено, що діє обмеження 30 запитів за хвилину та 2 запити за секунду, чого більш ніж достатньо для мого мобільного додатку. Нижче описана додаткова інформація та самі запити, які використовувались в застосунку.

Головний шлях API: "https://api.jikan.moe/v3"

Запит 1:

Кінцевий шлях: "/top/anime/{page}/{subtype}"

Опис: Запит повертає відфільтрований та відсортований по параметру subtype список з 50-ти аніме, з можливістю пагінації по цьому списку.

Параметри:

page – поточна сторінка пагінації.

subtype – сортування за різними критеріями:

- bypopularity: Відсортовані аніме за популярністю.
- movie: Відсортовані аніме фільми за рейтингом.
- tv: Відсортовані аніме серіали за рейтингом.
- airing: Відсортовані аніме за прем’єрою.

Запит 2:

Кінцевий шлях: "/anime/{id}(/request)"

Опис: Запит повертає всю інформацію про поточне аніме за унікальним ідентифікатором, тобто параметром id, також є можливість робити додаткові запити з використанням необов’язково параметру request.

Параметри:

id – унікальний ідентифікатор аніме.

request – необов’язковий параметр для додаткової інформації:

- character_staff: Список усіх персонажів поточного аніме.

Запит 3:

Кінцевий шлях: "/character/{id}"

Опис: Запит повертає наявну детальну інформацію про персонажа аніме за його унікальним ідентифікатором, параметром id.

Параметри:

id – унікальний ідентифікатор персонажа.

Запити, використані мною, повертають інформацію, яка успішно декодується з формату JSON у структури даних, з якими вже йде взаємодія в мобільному додатку. Вони передаються з шару даних у шар домену і під кінець надходять до моделі вигляду, яка вже напряму взаємодіє з графічним інтерфейсом, правильно відображуючи їх користувачеві.

# 4 Багатопоточність в iOS

## 4. 1 Загальні відомості

Сучасний більш-менш серйозний мобільний додаток просто неможливо представити без використання багатопоточності. Вона дозволяє розмежувати групи задач та виконувати їх на різних потоках, розпаралелювати складні для системи обчислення, задля зменшення часу їх виконання, покращувати загальний архітектурний дизайн, використовуючи блокування потоків та інші операції. Але, на мою думку, найголовніша перевага використання багатопоточності — це розвантаження головного потоку, який відповідає за відображення графічного інтерфейсу, системних повідомлень, обробку жестів і взаємодію з користувачем. Якщо на цей потік додати ресурсні задачі, наприклад, завантаження даних з мережі, то робота користувацького інтерфейсу може суттєво сповільнитися, що негативно вплине на загальні враження користувача. Щоб цього не сталося iOS представляє багато інструментів для цих цілей, але ми зупинимось на одному, а саме GCD[5].

Grand Central Dispatch (GCD) — бібліотека, надана розробниками від Apple, для паралельного виконання завдань в декількох потоках. Вона складається з виконуваних операцій і черг, які містять в собі ці операції. У кожної черги є своя пріоритетність, чим вона вище, тим більше процесорного часу виділяється під задачі для цієї черги. Крім черг, які можна самостійно створювати, система iOS дає в розпорядження розробника глобальні черги DispatchQueue. Всього їх шість — головна черга, на якій виконуються всі операції, пов’язані з графічним інтерфейсом, і п’ять фонових паралельних глобальних черг з різною пріоритетністю, від більшої до меншої: userInteractive, userInitiated, default, utility, background. Задачі в цих чергах можуть виконуватися синхронно чи асинхронно (послідовно чи паралельно). А для завантаження даних з мережі найкращою буде черга з пріоритетом обслуговування default або utility.

## 4. 2 Реалізація в додатку

В додатку було використано два типи черг для різних видів задач, а саме:

- DispatchQueue.main.async
- DispatchQueue.global(qos: .default).async

Перша черга виконується на головному потоці і, як вже було зазначено раніше, за допомогою неї ми оновлюємо графічний інтерфейс після того, як з моделі вигляду надійшли дані, які потрібно відобразити на екрані, або при додаванні конкретного аніме до списку обраних. Також варто зазначити, що задачі черги виконуються асинхронно, щоб не блокувати основний потік, бо це призведе до аварійного завершення роботи додатку.

Друга черга є однією з глобальних паралельних фонових черг і має пріоритет, або скорочено qos (англ. quality of service) — default, який рекомендовано для мережевих запитів. Він оброблює дані швидше аніж utility, при цьому не конкуруючи за надмірні ресурси додатку. Головною метою цієї черги є розвантаження основного потоку за допомогою передачі задач, виконання яких потребує багато ресурсів та часу, на інші потоки. Глобальна черга використовується для того, щоб асинхронно завантажити дані з запиту API в модель вигляду, після чого передаючи їх за допомогою замикань на чергу головного потоку, яка відображає їх в вигляді.

Хотілось би наголосити, що розробник не може повністю контролювати роботу потоків, тому система сама вирішує, які ресурси і який процесорний час надавати для виконання задачі в залежності від її пріоритетності. Це дуже полегшує розробку, тому що не потрібно думати на якому потоці, що запустити — GCD автоматично керує цим процесом. У додатку були реалізовані методи багатопоточного програмування, які пришвидшують його роботу, не залишаючи користувачеві місця для негативних вражень від графічного інтерфейсу і взаємодії з ним.

# 5 Структура iOS-додатку

## 5. 1 Графічний інтерфейс користувача

Правильний графічний інтерфейс є одним з найважливіших факторів, які впливають на успіх додатку. Це саме той кінцевий продукт з яким взаємодіє користувач, тому він має бути досконало продуманим, не містити спірних дизайнерських рішень і бути зручним й інтуїтивним у використанні. Дизайн графічного інтерфейсу не повинен відволікати користувача від основних функцій, які виконує програмний продукт.

Було вирішено не відходити від основних принципів Human Interface Guidelines[6] при розробці iOS-додатку, а саме:

- Естетична цілісність;
- Послідовність;
- Чітка взаємодія;
- Контроль користувача.

Тому були використані стандартні графічні компоненти інтерфейсу, в поєднані зі стандартними шрифтами і кольорами, що гармонійно вписуються в загальний дизайн операційної системи iOS 13 та використовуються в офіційних додатках Apple.

Під час запуску додатку, користувач бачить перед собою екран завантаження (див. рис. 5.1.1), що показується до тих пір, поки додаток заходиться в стані Inactive.

Рисунок 5.1.1 Екран завантаження

Після того, як всі основні ресурси, потрібні для роботи додатку завантажилися, користувач потрапляє на одну з головних сторінок, на якій відображається список з найбільш популярних аніме, відсортованих у порядку спадання популярності, як показано на рисунку 5.1.2, де вказано тип цього аніме, його назва, рейтинг, дата виходу першої і останньої серії, обкладинка і кількість епізодів. Користувач має змогу переходити на інші сторінки зі списками, в яких елементи відсортовані по іншим критеріям  за допомогою нижньої панелі вкладок, а саме: "По рейтингу аніме фільмів", "По рейтингу аніме серіалів", "За прем’єрою" та аніме, які були додані в обране і збережені в пам’яті телефону. Списки категорій завантажуються з мережі інтернет і мають пагінацію по сторінкам, як зазначено в функціональних вимогах. Перехід на нову сторінку пагінації відбувається непомітно для користувача за допомогою протоколу UITableViewDataSourcePrefetching[7] , який дозволяє завантажувати дані раніше, ніж вони будуть потенційно відображені в списку.

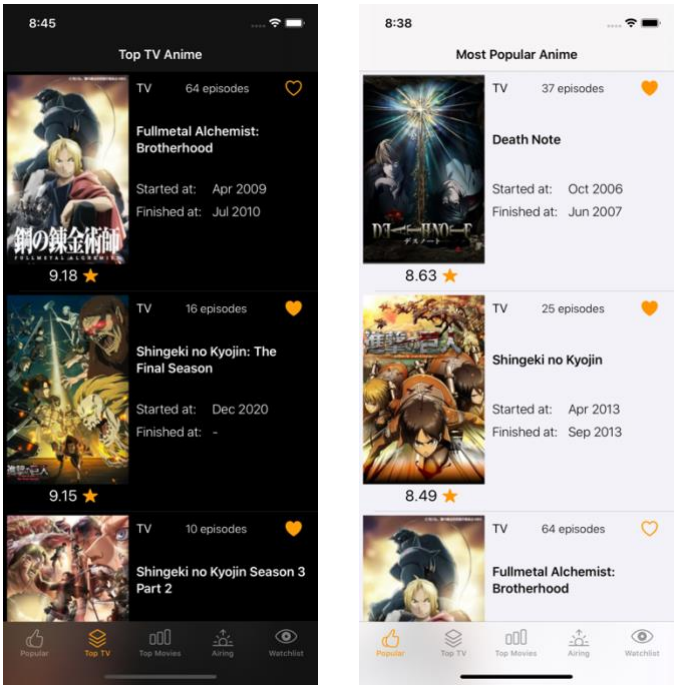


Рисунок 5.1.2 Одна з головних сторінок
зі списком аніме (темна і світла теми)

Натиснувши на аніме зі списку, користувач переходить на його сторінку, де відображається більш детальна інформація (див. рис. 5.1.3). Вона включає в себе всю інформацію, доступну з попереднього екрана, в поєднанні з новою: його популярність, переклад назви англійською мовою, якщо він присутній, першоджерело аніме, кількість користувачів, які його оцінили, вікові обмеження, довжину кожної серії/фільму, жанри, студії, які займалися його створенням, а також детальний опис цього аніме. З цієї сторінки є можливість перейти на ще одну сторінку, натиснувши на кнопку "Characters List", на якій відображається список персонажів, або перейти в сервіс відеохостингу YouTube та подивитися трейлер цього аніме за наданим посиланням.

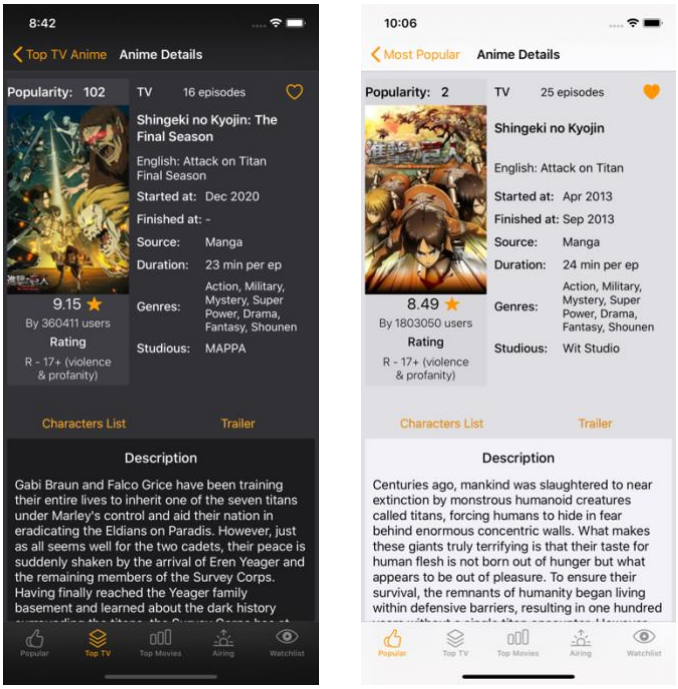


Рисунок 5.1.3 Сторінка з детальною інформацією про конкретне аніме (темна і світла теми)

На наступній сторінці зі списком усіх персонажей, яку показано на рисунку 5.1.4, є інформація про ім’я персонажу, його роль (головна або допоміжна) і портрет. Цей список не має пагінації тому завантажується одразу одним запитом до API, бо кількість персонажів не може бути дуже великою, саме тому необхідність реалізації пагінації є зайвою. З цієї сторінки користувач може перейти на сторінку з більш детальним описом персонажу. До цього опису відноситься його повне ім’я, наявні прізвиська, якщо воно присутнє, та загальна біографія персонажа.

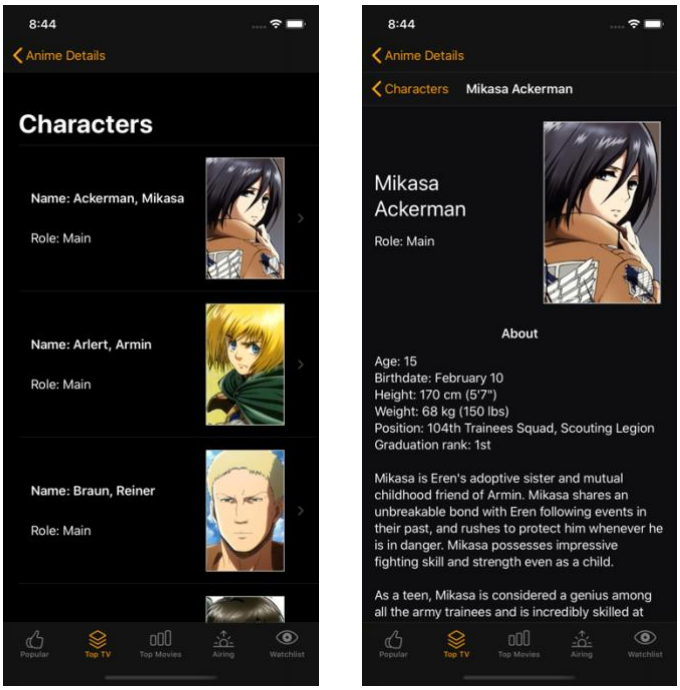


Рисунок 5.1.4 Сторінки зі списком персонажів і детальним описом кожного з них

Окремо хочу звернути увагу на сторінку, на якій відображаються список обраних користувачем аніме. Вона відрізняється від сторінки зі списком аніме, відсортованих за різними категоріями. Як видно на рисунку 5.1.5, перед цим списком є поле пошуку, яке дозволяє шукати збережені в ньому аніме, що є єдиною відмінністю цієї сторінки від інших.

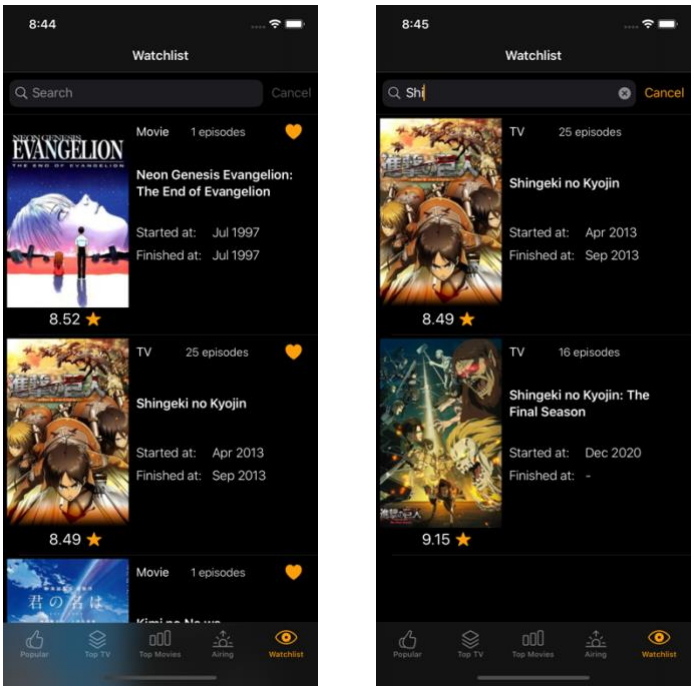


Рисунок 5.1.5 Екран зі списком обраного та пошук по ньому

Варто наголосити, що переходи між сторінками відбуваються за допомогою навігаційної панелі UINavigationBar, яка в поєднанні з нижньою панеллю вкладок UITabBar створює потужний та зручний інструмент навігації між сторінками, який і був повною мірою використаний у мобільному додатку. Також не можна не зазначити, що застосунок підтримує нововведення iOS 13, дозволяючи змінити колір інтерфейсу системи і додатків, які підтримують цю технологію, в залежності від активної теми, яких є дві — темна і світла. Як можна побачити на рисунках 5.1.2 та 5.1.3, завдяки використанню класичних кольорів, елементи графічного інтерфейсу гармонійно відображаються в обох варіаціях системної теми.

Мобільний додаток створений з урахуванням наявних мобільних пристроїв, починаючи від iPhone SE 2, тому графічний інтерфейс адаптований для кожного з них, про що можна переконатися на рисунку 5.1.6.

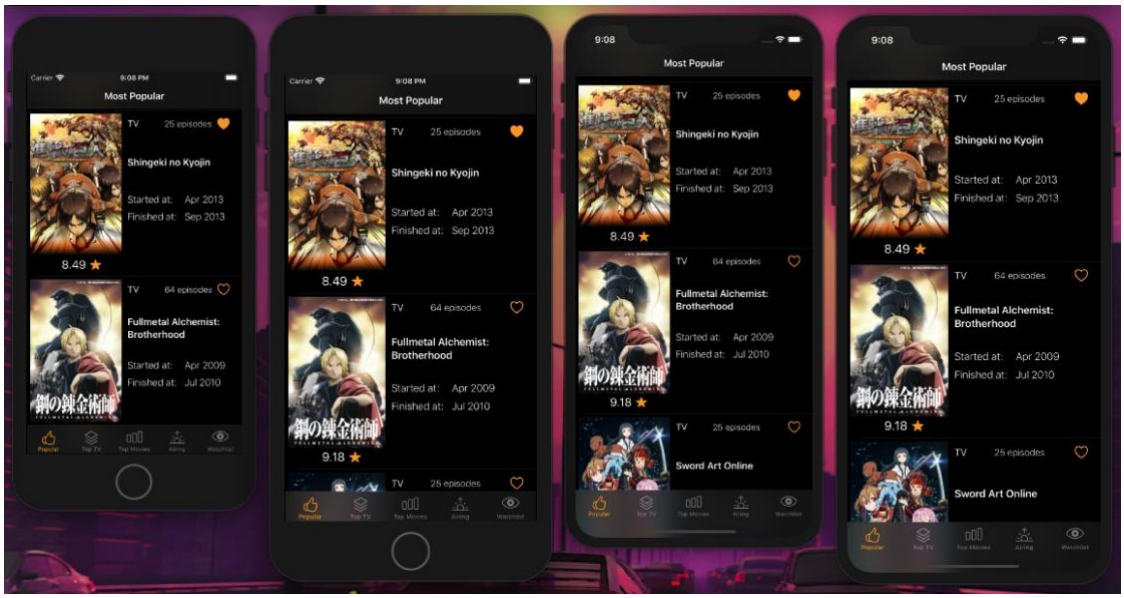


Рисунок 5.1.6 Вигляд додатку на різних пристроях

## 5. 2 Структура класів та їх взаємодія

### 5. 2. 1 Основні компоненти

Структура класів додатку побудована за допомогою принципів чистої архітектури, тому в кореневій директорії проекту є три папки з назвами шарів: Presentation, Domain і Data, а також декілька допоміжних класів і папка з медіа-ресурсами додатку. Всього проект містить 29 класів і структур, які розташовуються в 8-ми директоріях. Давайте розглянемо їх більш детально, провівши короткий опис усіх наявних класів та директорій, де ці класи та структури розміщені.

Класи AppDelegate та SceneDelegate відповідають за загальний стан додатку, поточну сцену — екран, де знаходиться користувач, а також керування цими екранами. Ці класи є невідмінною складовою кожного мобільного додатка, написаного для операційної системи iOS.

Директорія Assets створена для організації медіа-ресурсів мобільного додатку. Вона дозволяє зберігати графічні зображення, іконки з різною роздільною здатністю для коректного відображення на екранах різних розмірів і безпосереднього використання в коді проекту. Також в ній можна розмістити іконки мобільного додатку, які будуть показуватися в системі iOS в налаштуваннях, в повідомленнях, а також на екрані встановлених застосунків.

Директорії Presentation, Domain, Data, в яких розміщені основні класи додатку, задля більш зручної організації проекту, яка робить його структуру чистішою і зрозумілішою для розробника. Це так звані шари додатку, які описувалися раніше. Кожен з них ми по черзі розглянемо та коротко проаналізуємо їх зміст.

### 5. 2. 2 Шар даних

Шар даних (директорія Data) має внутрішні директорії Network і Model , а також два інші класи. Директорія Network містить всередині класи, які відповідають за отримання даних з мережі за допомогою API, їх декодування з формату JSON та перетворення у відповідні структури з директорії Model. Директорія Model є невід'ємною частиною архітектури шаблону MVVM. У ній розміщуються структури даних, потрібних для подальшої роботи додатку.

Вміст директорії:

- Клас ListRepository

Один з головних елементів шару даних. Відповідає за підпорядкування даних, отриманих з класу HTTPService, а також збереження деяких даних у локальний файл мобільного пристрою та можливість подальшого доступу до них. Цей клас і його методи використовується в шарі домену .

- Клас PersistenceManager

Допоміжний клас, створений задля того, щоб зберігати у пам'яті додатку деякі дані. Він взаємодіє з класом ListRepository та HTTPService.

Вміст внутрішньої директорії Model:

- Структура CharactersList

Допоміжна структура, створена для того, щоб декодувати дані API про всіх персонажів і їх відображення в списку структур CharacterInfo.

- Структура CharacterInfo

Структура, що уособлює собою сутність елемента списку персонажів і містить поля з цими даними.

- Структура CharacterDetails

Відображає додаткові дані про конкретного персонажа за запитом API, декодувавши ці дані з мережі і записавши їх у відповідну структуру.

- Структура ItemsList

Допоміжна структура, створена для того, щоб декодувати дані API (запит 1) і забезпечити їх відображення в списку структур ListItem.

- Структура ListItem

Структура, що відображає дані одного рядка зі списку аніме, записані у відповідних полях.

- Структура Item

Відображає додаткові дані про конкретне аніме за запитом API, декодувавши ці дані з мережі і записавши їх у відповідну структуру.

- Структура ItemInfo

Допоміжна структура, яка відображає дані про категорії та студії, які займаються розробкою аніме.

Вміст внутрішньої директорії Network:

- Перелік ListApi

Перелік, який дозволяє записувати запити API у зручному форматі без повного приписання шляху запиту.

- Перелік ListError

Перелік, який реалізовує власний клас помилок. Він активно використовується в додатку.

- Перелік ServiceResponse

Обгортка, створена задля того, щоб передавати дані в замиканнях[8] (англ. closures). Містить два стани: успіх і невдача. При успіху передає будь-які типи даних, а при невдачі — ListError.

- Клас HTTPRequester

Клас, який місить в собі метод завантаження даних API в додаток за допомогою класу URLSession[9].

- Клас HTTPService

Клас, за допомогою якого реалізовуються методи прив’язки даних, отриманих з класу HTTPRequester у відповідних структурах з директорії Model.

### 5. 2. 3 Шар домену

Шар домену (директорія Domain) є важливою складовою архітектури додатку. У нашому випадку він представлений одним класом, а саме:

- Клас UseCase

Клас отримує дані з ListRepository, задля їх подальшого використання в моделі вигляду, а також реалізує додаткову бізнес-логіку додатку.

### 5. 2. 4 Шар представлення

Цей шар (директорія Presentation) представлений двома storyboards[10] — основним і стартовим, а також двома внутрішніми директоріями View, ViewModel, які є основними компонентами шаблону архітектури MVVM.

Вміст внутрішньої директорії View:

- Клас ListViewController

Клас, який містить реалізацію UITableViewController, контролера, керуючого представленням списку, або таблиці на екрані додатку. В нашому випадку, це список аніме.

- Клас ListItemViewCell

Імплементація UITableViewCell візуального відображення одного рядка зі списку ListViewController з інформацією про конкретне аніме.

- Клас ListSavedViewController

Клас, який містить реалізацію UITableViewController, а саме списку з аніме, збережених в пам’ять мобільного пристрою і доданих до обраного. На відміну від ListViewController, він має додаткову пошукову стрічку UISearchBar, яка дозволяє користувачу шукати обрані аніме по їх назві.

- Клас ItemViewController

Клас контролю екрану, де показується вся інформація про конкретне аніме. Він успадкований від стандартного класу UIViewController та використовується для контролю UIView, яка являє собою екран, обробку подій та відображення в ньому деяких даних.

- Клас CharacterListViewController

Клас контролю екрану, де показується список з персонажами цього аніме. Успадкований від UIViewController та містить в собі реалізацію зв’язку UIKit і SwiftUI за допомогою UIHostingController, що дозволяє інтегрувати структури, написані на SwiftUI, нового фреймворку для написання користувацького інтерфейсу, у звичайний контролер екрану.

- Структура CharacterDetailsView

Структура, написана на SwiftUI, у якій показується екран з детальним описом конкретного персонажу.

- Структура CharacterRowView

Структура, написана за допомогою SwiftUI, яка дозволяє візуально відображати один рядок зі списку CharactersListView з інформацією про конкретного персонажа.

- Структура CharactersListView

Структура, що відображає список з усіх персонажів конкретного аніме, написана на SwiftUI, а також імплементацію навігації по цьому списку, тобто переходу від екрану списку персонажів до екрану з додатковими відомостями про цього персонажа.

Вміст внутрішньої директорії ViewModel:

- Клас ListViewModel

Реалізація зв’язку списку моделей ListItem з виглядом ListViewController за допомогою механізму прив’язки даних, у нашому випадку цей механізм — замикання. Містить в собі екземпляр класу UseCase, який пов’язує рівень представлення і рівень моделі. У класі використовуються методи UseCase, для отримання початкових даних і даних, при подальшій пагінації списку, а також інші методи, які реалізують додаткову внутрішню логіку моделі вигляду.

- Клас ListSavedViewModel

Клас, що реалізовує зв’язок списку моделей ListItem з її виглядом ListSavedViewController за допомогою замикань. Містить в собі екземпляр класу UseCase, з методом отримання списку обраних аніме, які локально зберігаються на телефоні. Також у класі зосереджена логіка пошуку по назві аніме, при цьому результати пошуку записуються в окремий список

ListItem та показується тільки тоді, коли користувач починає використовувати панель пошуку.

- Клас ItemViewModel

Клас, за допомогою якого відбувається зв’язок між моделлю ListItem та додатковими полями моделі Item з контролером вигляду ItemViewController. Використовує методи екземпляру UseCase для додавання аніме в список обраних, його видалення з цього списку, завантаження додаткових даних про конкретне аніме. Зберігає в собі дані, передані при переході з екрану ListViewController до ItemViewController.

- Клас ListItemViewModel

Клас, який пов’язує модель ListItem з виглядом ListItemViewCell. В ньому міститься методи екземпляру класу UseCase для додавання аніме в список обраних, а також його видалення з цього списку.

- Клас CharactersViewModel

Клас, який реалізує зв’язок між списком моделей CharacterInfo та виглядом CharacterListViewController. У середині цього класу використовуються метод екземпляру класу UseCase, щоб завантажувати весь список персонажей, так як цей тип запиту API не має пагінації.

- Клас CharacterDetailsViewModel

Реалізація зв’язку моделі CharacterDetails з виглядом CharacterDetailsView. Метод з UseCase застосовується, для того щоб отримати додаткову інформації про конкретного персонажа.

Вміст директорії:

- Storyboard LaunchScreen

Візуальне представлення екрану під час завантаження самого додатку і основних ресурсів, потрібних для його роботи.

- Storyboard Main

Основний інструмент для представлення графічного інтерфейсу додатку, в якому контролери вигляду зв’язані один з одним за допомогою переходів UIStoryboardSegue між цими екранами з використанням навігаційної панелі та нижньої панелі вкладок (див. рис. 5.2.4).

Рисунок 5.2.4 Переходи між екранами в storyboard

## 5. 3 Додаткові бібліотеки

При розробці додатку були використані дві додаткові бібліотеки, встановлені за допомогою менеджеру залежностей Swift Package Manager[11]. Він дозволяє iOS-розробникам додавати до проекту сторонні бібліотеки.

Бібліотеки, які були використані:

- SDWebImage

  Бібліотека, яка забезпечує асинхронне завантаження зображень по url з можливістю їх подальшого кешування.

- SDWebImageSwiftUI

  Бібліотека, для завантаження зображень в середовищі SwiftUI. Вона працює на основі SDWebImage.

# Висновки

Результатом виконання цієї роботи стало створення самостійного програмного продукту, який дозволяє користувачеві в повному обсязі використовувати всі його технічні можливості і виконувати поставлені перед ним задачі. Додаток стане в нагоді людям, які хочуть знайти собі нове аніме для перегляду, керуючись його рейтингом, популярністю, статусом, тощо, а також матимуть можливість додати його до списку обраних.

В ході розробки були досліджені і проаналізовані основні методи та засоби розробки програмного забезпечення під операційну систему iOS. Також був розроблений зручний і інтуїтивно зрозумілий графічний інтерфейс з адаптацією під різні розміри екрану, що дозволяє користуватися цим застосунком на будь-якому мобільному пристрої Apple, який підтримує операційну систему iOS 13 і вище.

Варто зауважити, що всі функціональні вимоги, які ставилися переді мною, були успішно виконані. Даний програмний продукт був розроблений за допомогою мови програмування Swift і інтегрованого середовища розробки Xcode, яке дозволяє в повному обсязі використовувати усі необхідні засоби та інструменти для успішної розробки мобільного програмного забезпечення.

Перспективами розвитку даного мобільного додатку є збільшення кількості функціоналу, який дозволить в повністю використати всі можливості Jikan API.

# Список літератури

1. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin. – Upper Saddle River: Prentice Hall, 2017. – 1432 с.

2. The MVVM Pattern [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10).

3. Codable [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/swift/codable.

4. Jikan API [Електронний ресурс] – Режим доступу до ресурсу: https://jikan.docs.apiary.io.

5. Grand Central Dispatch [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/DISPATCH.

6. Human Interface Guidelines [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.

7. UITableViewDataSourcePrefetching [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/uikit/uitableviewdatasourceprefetching.

8. Closures [Електронний ресурс] – Режим доступу до ресурсу: https://docs.swift.org/swift-book/LanguageGuide/Closures.html.

9. URLSession [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/foundation/urlsession.

10. iOS Storyboards: Getting Started [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://www.raywenderlich.com/5055364-ios-storyboards-getting-started.

11. Swift Package Manager [Електронний ресурс] – Режим доступу до ресурсу: https://swift.org/package-manager/.

# Додаток А (обов’язковий). Лістінг класу HTTPRequester

```
class HTTPRequester{

    static var shared = HTTPRequester()
    private init(){}

    func requestData(api: ListApi,  completion: @escaping (ServiceResponse<Data>) -> ()) {
        guard let url = URL(string: api.url) else{
            completion(.failure(.network(.urlError)))
            return
        }
        let task = URLSession.shared.dataTask(with: url) { (data, response, error) in
            guard error == nil && data != nil  else {
                completion(.failure(.network(.requestDataError)))
                return
            }
            completion(.success(data!))
        }
        task.resume()
    }
}
```

# Додаток Б (обов’язковий). Лістінг методу з класу HTTPService

Метод getListRequest перетворює завантажені дані з мережі у список структур ListItem.

```
func getListRequest(postsApi: ListApi, completion: @escaping (ServiceResponse<[ListItem]>) -> ()){

    HTTPRequester.shared.requestData(api: postsApi) { (response) in
        switch response {
        case .success(let json):
            do{
                let results: [ListItem] = try JSONDecoder().decode(ItemsList.self, from:json).itemsList
                self.dbManager.setListItems(items: results)
                completion(.success(results))
            } catch {
                completion(.failure(.decodingError))
                return
            }
        case .failure(let error):
            completion(.failure(error))
        }
    }
}
```

# Додаток В (обов’язковий). Лістінг структури ItemsList

```
struct ItemsList: Decodable{

    var itemsList: [ListItem] = []

    private enum RootKeys: String, CodingKey{
        case top

        enum DataKeys: String, CodingKey{
            case id = "mal_id"
            case title
            case image = "image_url"
            case type
            case episodes
            case start = "start_date"
            case end = "end_date"
            case members
            case score
        }
    }


    init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: RootKeys.self)
        var dataCr = try container.nestedUnkeyedContainer(forKey: .top)
        while !dataCr.isAtEnd {
            let infoCr = try dataCr.nestedContainer(keyedBy: RootKeys.DataKeys.self)
            let item = ListItem(
                id: try infoCr.decode(UInt?.self, forKey: .id),
                title: try infoCr.decode(String?.self, forKey: .title),
                type: try infoCr.decode(String?.self, forKey: .type),
                episodes: try infoCr.decode(UInt?.self, forKey: .episodes),
                start: try infoCr.decode(String?.self, forKey: .start),
                end: try infoCr.decode(String?.self, forKey: .end),
                members: try infoCr.decode(UInt?.self, forKey: .members),
                score: try infoCr.decode(Float?.self, forKey: .score),
                image: try infoCr.decode(String?.self, forKey: .image))
            self.itemsList.append(item)
        }
    }
}
```

# Додаток Г (обов’язковий). Лістінг класу UseCase

```
class UseCase{

    private let repository: ListItemsRepository = ListItemsRepository()

    func executeItemsList(request: ListApi, completion: @escaping (ServiceResponse<[ListItem]>) ->()){
        repository.getList(query: request, completion: completion)
    }

    func executeItem(request: ListApi, completion: @escaping (ServiceResponse<Item>) ->()){
        repository.getItem(query: request, completion: completion)
    }

    func executeCharactersList(request: ListApi, completion: @escaping (ServiceResponse<[CharacterInfo]>) ->()){
        repository.getCharactersList(query: request, completion: completion)
    }

    func executeCharacter(request: ListApi, completion: @escaping (ServiceResponse<CharacterDetails>) ->()){
        repository.getCharacter(query: request, completion: completion)
    }


    func getListItems(items: [ListItem]) -> [ListItem]{
        return items
    }


    func getSavedListItems(completion: @escaping (ServiceResponse<[ListItem]>) ->()){
        repository.readFrom(fileName: "saved.json", completion: completion)
    }

    func saveListItems(items: [ListItem], completion: @escaping (ServiceResponse<[ListItem]>) ->()){
        repository.saveListItem(data: items, withName: "saved.json", completion: completion)
    }
}
```

# Додаток Д (обов’язковий). Лістінг класу CharactersViewModel

```
class CharactersViewModel {

    // MARK: - Variables
    private lazy var useCase: UseCase = UseCase()
    var characters: [CharacterInfo] = []


    // MARK: - Functions
    func loadListCharacters(id: UInt, completion: @escaping (ListError?) -> ()) {

        DispatchQueue.global(qos: .default).async{
            let requestBody = ListApi.getItem(id: id, details: .characters_staff)

            self.useCase.executeCharactersList(request: requestBody){(response) in
                switch response {
                case .success(let data):
                    self.characters = data
                    completion(nil)
                case .failure(let error):
                    completion(error)
                    print(error)
                }
            }
        }
    }
}
```