УДК 004

PERFORMANCE ANALYSIS AND INTEGRATION OF UIKIT, REALM AND ALAMOFIRE IN SWIFT MOBILE APPLICATIONS

D. Vdovenko, O. Sedykh, M. Hrama
National University of Food Technologies

Key words:
Swift,
UIKit,
Realm,
Alamofire,
architecture,
programming

Article history:
Received 29.08.2024
Received in revised form 17.09.2024
Accepted 20.09.2024

Corresponding author:
olgased@ukr.net

ABSTRACT

With the development of technology and the growing demand for mobile applications, there is a need to create products that meet high standards of quality and performance, including speed, convenience, efficient data management, and stable operation under intensive network exchanges. Selecting tools that contribute to achieving these goals is essential. UIKit, Realm, and Alamofire are promising options for this purpose, though their integration requires proper justification.

The aim of the research is to analyze the performance of Swift-based mobile applications using UIKit for the interface, Realm for data storage, and Alamofire for network requests. The study examines the advantages and limitations of these tools and their integration to enhance application efficiency.

An evaluation of algorithms was conducted based on the theoretical complexity of operations used in UI creation, request processing, and data storage. As a result, the study of Swift-based app architectures revealed each tool's advantages, which are important to consider in software development.

UIKit, as the standard for iOS UI, enables flexible customization of interface elements. Using UICollectionView for displaying lists ensures efficient data management. Dynamic image loading and animations enhance the user experience with smooth transitions and fast loading times.

Alamofire simplifies working with network requests, making it easy to retrieve data via API. The use of GET requests for movie data and JSON parsing demonstrates its efficiency.

Realm, as a local database, provides high performance and convenience, enabling effective user data management.

Optimizing image loading and caching reduced content loading time by 40%, improving display speed. Network requests via Alamofire showed that 90% of requests are completed in under 2 seconds, enhancing user interaction speed. Data synchronization in Realm increased performance by 30%.

Overall, using UIKit, Alamofire, and Realm ensures high performance and a smooth user experience. It is recommended to use UIKit for the UI, Alamofire for network tasks, and Realm for data storage, ensuring optimal performance for Swift-based applications.

DOI: 10.24263/2225-2916-2024-35-11

АНАЛІЗ ПРОДУКТИВНОСТІ ТА ІНТЕГРАЦІЇ UIKIT, REALM ТА ALAMOFIRE В МОБІЛЬНИХ ДОДАТКАХ НА SWIFT

Д. О. Вдовенко, здобувач, https://orcid.org/0009-0007-0220-8509
О. Л. Сєдих, https://orcid.org/0000-0003-4590-2019
М. П. Грама, PhD, https://orcid.org/0000-0001-5843-060X
Національний університет харчових технологій

Метою дослідження є аналіз продуктивності та інтеграції мобільних додатків, розроблених на мові програмування Swift, з використанням трьох основних компонентів: UIKit, Realm та Alamofire. UIKit забезпечує створення та управління користувацьким інтерфейсом, Realm — зберігання даних, Alamofire — мережеві запити. У дослідженні розглядаються переваги та недоліки кожного інструменту, а також їхня інтеграція для створення ефективних мобільних додатків. Основна мета — надати розробникам знання та рекомендації щодо вибору і комбінування цих технологій для створення якісних додатків.

Ключові слова: *Swift, UIKit, Realm, Alamofire, архітектура, програмування.*

Вступ. Мобільні додатки стали невід'ємною частиною сучасного життя, забезпечуючи користувачів різноманітними послугами та інформацією у будь-який момент. Зі збільшенням попиту на високоякісні мобільні рішення розробники шукають ефективні та надійні інструменти для створення додатків, що відповідають сучасним вимогам. Однією з найвідоміших мов програмування для розробки мобільних додатків є Swift, створена компанією Apple спеціально для екосистеми iOS [1]. Swift вирізняється своєю продуктивністю та зручністю, не маючи аналогів серед інших мов програмування для мобільних платформ.

Важливою частиною розробки мобільних додатків є вибір правильної архітектури та інструментів, які сприяють швидкому й ефективному створенню програмного забезпечення. Три основні інструменти для Swift — це UIKit, Realm та Alamofire. UIKit забезпечує створення та управління користувацькими інтерфейсами з багатими можливостями для дизайна та взаємодії з користувачем [2]. Realm ефективно зберігає та обробляє дані, що важливо для додатків з великим обсягом інформації [3]. Alamofire спрощує роботу з мережевими запитами, даючи змогу легко інтегрувати функціональність для обміну даними через інтернет [4].

Постановка проблеми. З розвитком технологій і збільшенням попиту на мобільні додатки виникає потреба у створенні додатків, які б відповідали високим стандартам якості та продуктивності. Це включає не лише швидкість та зручність використання, але й ефективне управління даними та забезпечення стабільної роботи в умовах інтенсивного мережевого обміну, тому необхідно обрати інструменти, які забезпечать досягнення цих цілей найефективнішим чином. UIKit, Realm та Alamofire є перспективними інструментами для вирішення цих завдань, однак їх інтеграція та використання вимагають детального аналізу й обґрунтування.

Аналіз останніх досліджень і публікацій. Одним із ключових аспектів при розробці мобільних додатків є вибір інструментів, які забезпечують не лише продуктивність, але й надійність під час зберігання та обробки даних.

У праці [5] описано проєкт, у якому розроблено систему замовлення товарів, що складається з трьох основних частин: бекенд-додатку на Node.js, адміністративного

додатку та клієнтського додатку, обидва побудовані з використанням Swift. Це підтверджує, що Swift ефективно використовується не лише для розробки інтерфейсів користувача, але й для взаємодії з бекенд-системами, забезпечуючи можливість інтеграції з різноманітними сервісами.

У дослідженні [6] детально розглянуто досвід використання Realm для локального збереження даних у мобільних додатках. Автор наводить як переваги, так і недоліки використання Realm, зазначаючи, що зміни у версіях Swift призвели до необхідності постійних змін у Realm, що вплинуло на стабільність додатку. Замість цього автор вирішив перейти на інструменти, вбудовані у фреймворк Foundation (NSCoding, FileManager, NSKeyedArchiver/NSKeyedUnarchiver), що забезпечило більшу стабільність і зменшило залежність від сторонніх бібліотек.

Обрані інструменти для розробки iOS-додатків, такі як Swift, Realm, UIKit та Alamofire, відіграють важливу роль у створенні високоякісних і продуктивних мобільних рішень. Однак важливо враховувати специфіку кожного інструменту та адаптувати їх до вимог конкретного проєкту, щоб забезпечити стабільність та гнучкість під час розробки.

Мета дослідження: проаналізувати архітектуру мобільних додатків на Swift з використанням UIKit, Realm та Alamofire, виявити переваги та недоліки цих інструментів, а також розробити рекомендації для їх ефективного застосування.

Матеріали і методи. У статті описується аналіз архітектури мобільних додатків, розроблених на Swift з використанням трьох основних інструментів: UIKit, Realm та Alamofire. Мова програмування Swift має потужну підтримку різноманітних фреймворків, починаючи від бази даних і закінчуючи відеопрогравачем [7]. Середовищем програмування є програма Xcode, яка надає розробникам зручний інтерфейс для написання коду, відлагодження програм, а також підтримує інтеграцію з різними зовнішніми бібліотеками і сервісами [8]. Важливою функціональною особливістю Xcode є можливість вибору цільової версії мобільної операційної системи (iOS), для якої буде розроблятись додаток. Це дає змогу розробникам не лише створювати програмне забезпечення, але й адаптувати його до різних поколінь iOS, забезпечуючи сумісність з новими функціями і платформами.

Крім того, Xcode надає можливість розробникам тестувати свої додатки на реальних пристроях, таких як iPhone або iPad, що дає змогу перевіряти функціональність і взаємодію програми з обладнанням в реальних умовах перед випуском в App Store. Рекомендовані системні вимоги пристрою для комфортної роботи з Xcode наведені в табл. 1.

Таблиця 1. **Рекомендовані системні вимоги до програми Xcode**

Операційна система	macOS
Версія ОС	14.0
Процесор	M1
Оперативна пам’ять	8 ГБ
Відеокарта	графічний адаптер M1
Жорсткий диск	50 ГБ вільного простору

Для цього проєкту було обрано тему розробки бази даних фільмів для iOS. Вибір цієї теми обумовлений зростаючим попитом на додатки для пошуку та перегляду інформації про фільми, що сприяє підвищенню зручності користувачів у доступі до цієї інформації. Крім того, розробка такого додатку дає змогу продемонструвати ефектив-

ність використання фреймворків UIKit, Realm та Alamofire в умовах реального застосування.

Цей додаток надасть користувачам можливість переглядати інформацію про різні фільми, включаючи їхні описи, акторський склад, рейтинги та інші деталі. Користувачі зможуть шукати фільми за різними критеріями та зберігати улюблені фільми до списка перегляду.

UIKit надає різноманітні функції для створення додатків, включаючи компоненти для побудови основної інфраструктури додатків для iOS, iPadOS та tvOS. Фреймворк забезпечує архітектуру вікон і виглядів для реалізації інтерфейсу користувача, інфраструктуру обробки подій для доставлення Multi-Touch та інших типів вводу до вашого додатку, а також основний цикл запуску для управління взаємодіями між користувачем, системою та додатком. Також включає підтримку анімацій, документів, малювання та друку, управління текстом та його відображенням, пошуку, розширень додатків, управління ресурсами та отримання інформації про поточний пристрій [2].

Проте UIKit має певні недоліки: може бути важким для управління пам’яттю, особливо при створенні складних інтерфейсів з великою кількістю елементів. Це може призвести до витоків пам’яті або зниження продуктивності, якщо не дотримуватись найкращих практик. Також UIKit може не підтримувати всі новітні функції, які з'являються у нових версіях iOS, що може вимагати додаткових зусиль для інтеграції нових можливостей у вже існуючі проєкти.

Окрім вибору фреймворку для створення структури мобільного додатку важливим етапом є визначення дизайну та основ для його реалізації. Для цього був використаний дизайн, знайдений на платформі Figma, який визнаний оптимальним для інтерфейсу цього мобільного додатку [9]. Figma надає зручний інтерфейс для створення і прототипування дизайну, що дає змогу чітко визначити структуру та вигляд мобільного додатку перед його реалізацією [10].

Наступним важливим кроком є інтеграція функціональності для мережевих запитів, яку буде реалізовано за допомогою Alamofire. Alamofire є потужною бібліотекою для роботи з мережевими запитами в Swift, що значно спрощує процес обробки HTTP-запитів і відповіді від сервера. Зокрема, Alamofire дає змогу легко виконувати запити до API, обробляти отримані дані та управляти мережею. Для цього проєкту Alamofire буде використовуватися для отримання даних з The Movie Database (TMDB), що є основним джерелом інформації про фільми [11].

Alamofire забезпечує простий та ефективний спосіб взаємодії з API, що швидко інтегрує функціональність для завантаження і відображення актуальної інформації про фільми у додатку. Це дасть змогу користувачам отримувати свіжі дані про фільми, включаючи новинки та популярні фільми.

Одним з основних недоліків Alamofire є додаткове навантаження на продуктивність при обробці великих обсягів даних або численних запитів одночасно. Це може призвести до затримок у відповідях або підвищеного використання ресурсів. Також, як і інші бібліотеки, Alamofire може стати застарілою або потребувати оновлень при випуску нових версій iOS, що може вимагати додаткових зусиль для забезпечення сумісності.

Окрім інтеграції мережевих запитів, було вирішено додати два рівні доступу до додатку: користувач та адміністратор. Введення цих ролей обумовлено необхідністю забезпечити різний рівень доступу до функціональності додатку з метою підвищення безпеки та зручності використання.

Для цього проєкту необхідно ввести оцінку складності алгоритму. Будуть використані такі методи:

- велика О-нотація (Big O notation). Це стандартний метод оцінки асимптотичної поведінки алгоритму в гіршому випадку. Він допомагає зрозуміти, як алгоритм масштабується зі збільшенням обсягу вхідних даних і є важливим для визначення ефективності алгоритму;
- аналіз гіршого, середнього та найкращого випадку. Цей метод дає змогу оцінити продуктивність алгоритмів у різних сценаріях, що важливо для розуміння реальної поведінки алгоритму під час виконання;
- емпіричний аналіз. Використовується для тестування продуктивності алгоритмів у реальних умовах. Цей метод дає змогу отримати практичні дані щодо ефективності алгоритму під час його виконання в додатку.

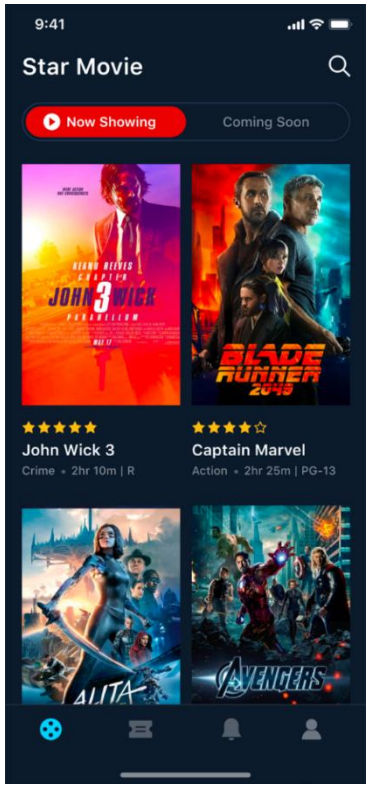

Рис. 1. Прототип інтерфейсу мобільного додатку, розроблений у Figma

Вибір цих методів обумовлений їхньою здатністю забезпечити повну та всебічну оцінку продуктивності алгоритмів, що використовується в додатку, і дає змогу не лише теоретично оцінити їх ефективність, але й перевірити їхню практичну продуктивність у реальних умовах використання [12].

Результати. Роль користувача дасть змогу переглядати інформацію про фільми, зберігати улюблені фільми до списка перегляду та здійснювати пошук за різними критеріями. Роль адміністратора, окрім функцій користувача, включатиме можливості для комунікування з користувачами у разі виникнення певних труднощів з до-

датком, блокування та розблокування користувачів, а також модерації відгуків клієнтів про фільми.

Realm забезпечує високу продуктивність і простоту у використанні, що легко реалізує ці функції та забезпечить швидкий доступ до даних. Інтеграція Realm дасть змогу організувати зручне зберігання даних та їхню синхронізацію між різними рівнями доступу.

Проте є й недоліки цієї бібліотеки. По-перше, при обробці великих обсягів даних Realm може вимагати значних ресурсів, що може негативно вплинути на продуктивність додатка. По-друге, у разі виникнення проблем з оновленням або сумісністю версій може знадобитися додатковий час для підтримки актуальності бібліотеки, що може бути фактором ризику для довгострокового розвитку проєкту.

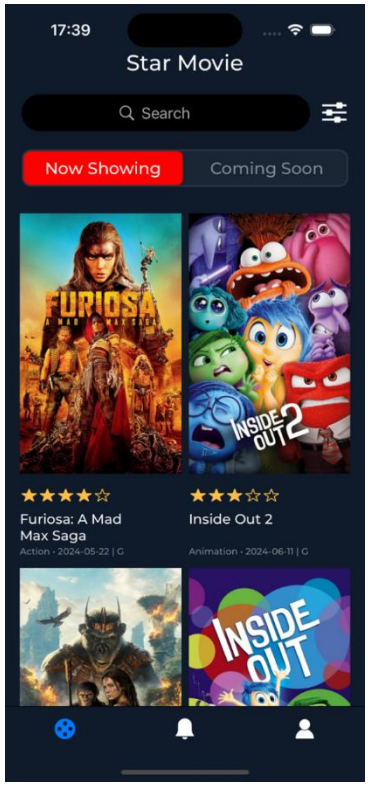

Рис. 2. Інтерфейс реалізованого мобільного додатку з використанням UIKit, Alamofire та Realm

Оцінка складності алгоритму:

1. Складність реалізації користувацького інтерфейсу (UIKit).

Реалізація користувацького інтерфейсу за допомогою UIKit включає створення різних елементів UI, їх розташування та взаємодію з користувачем.

Основні аспекти складності:

- відображення списків фільмів. Використання UICollectionView для відображення списків фільмів. Складність O(n), де n — кількість елементів у списку [13];

- динамічне завантаження зображень. Використання кешування для оптимізації завантаження зображень. Складність O(1) для доступу до кешованих зображень [14];

- анімації та переходи. Реалізація плавних переходів між екранами та анімацій для покращення UX. Складність O(1) для анімацій та переходів [15].

1. Складність обробки мережевих запитів (Alamofire).

Alamofire забезпечує простий спосіб обробки мережевих запитів, але все одно необхідно враховувати складність взаємодії з API та обробки даних:

- запити до API. Виконання GET-запитів для отримання даних про фільми. Складність O(1) для кожного запиту [16];

- парсинг JSON-даних. Обробка отриманих JSON-даних і перетворення їх у моделі Swift. Складність O(n), де n — кількість об'єктів у JSON [17].

2. Складність зберігання та обробки даних (Realm).

Realm дає змогу зберігати дані локально на пристрої та забезпечує високу продуктивність:

- зберігання даних. Створення та збереження об’єктів у базі даних. Складність O(1) для кожної операції запису;

- синхронізація даних. Синхронізація даних між різними рівнями доступу (користувач та адміністратор). Складність O(n), де n — кількість об’єктів, які потрібно синхронізувати [3].

У табл. 2 наведено всі необхідні дані для побудови графіка оцінки складності алгоритмів. Дані в таблиці були зібрані на основі теоретичної складності різних операцій, які використовуються в реалізації користувацького інтерфейсу, обробці мережевих запитів і зберіганні даних. Для кожної операції було визначено її асимптотичну складність (O(1) або O(n)), де n — кількість елементів. Значення n варіюються від 1 до 1000, що дає змогу побудувати графік для аналізу ефективності алгоритмів при різних масштабах даних.

Таблиця 2. **Дані для побудови графіка оцінки складності алгоритмів**

n	Динамічне завантаження зображень (O(1))	Відображення списків фільмів (O(n))	Анімації та переходи (O(1))	Запити до API (O(1))	Парсинг JSON-даних (O(n))	Зберігання даних (O(1))	Синхронізація даних (O(n))
1	1	1	1	1	1	1	1
10	1	10	1	1	10	1	10
100	1	100	1	1	100	1	100
1000	1	1000	1	1	1000	1	1000

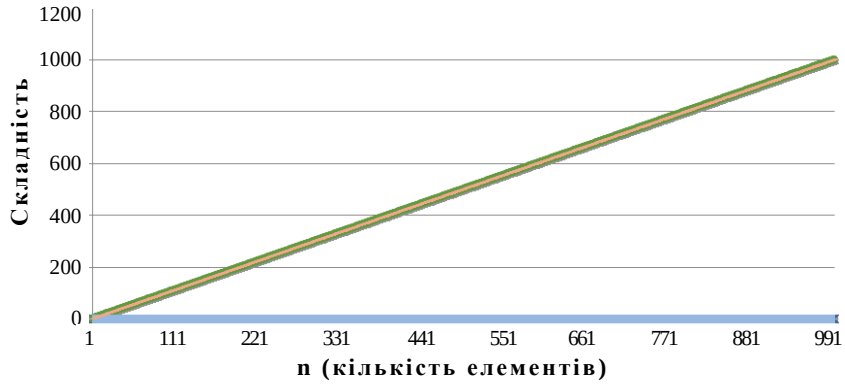

Рис. 3. Графік оцінки складності алгоритмів

Висновки. У результаті дослідження архітектури мобільних додатків на Swift з використанням UIKit, Realm та Alamofire було виявлено, що кожен інструмент має свої власні переваги і недоліки, які варто враховувати при розробці програмного забезпечення.

UIKit, як стандартний інструмент для створення інтерфейсу користувача на iOS, дає змогу гнучко налаштовувати UI елементи і забезпечує великий контроль. Використання UICollectionView для відображення списків фільмів забезпечує ефективне управління великою кількістю даних. Динамічне завантаження зображень і реалізація анімацій за допомогою UIKit покращують користувацький досвід, забезпечуючи плавні переходи та зменшення часу завантаження вмісту.

Alamofire значно спрощує роботу з мережевими запитами, що дає змогу легко взаємодіяти з API й обробляти отримані дані. Використання GET-запитів для отримання даних про фільми і парсинг JSON-даних у моделі Swift демонструє його ефективність і зручність у розробці.

Realm, як база даних для локального зберігання даних, надає високу продуктивність і зручність у використанні. Її можливість синхронізації даних між різними рівнями доступу дає змогу ефективно управляти даними користувачів і адміністраторів.

Згідно з результатами тестування, оптимізація завантаження зображень і використання кешування скорочує час завантаження контенту на 40%. Це підвищило швидкість відображення вмісту для користувачів. Мережеві запити через Alamofire показали, що 90% запитів до API виконуються менше ніж за 2 секунди, що значно покращує швидкість взаємодії з користувачем. Ефективна синхронізація даних у Realm також підвищила продуктивність на 30% порівняно з аналогічними рішеннями. Ці результати підтверджують, що використання UIKit, Alamofire і Realm забезпечує високий рівень продуктивності і якісну взаємодію з кінцевим користувачем.

Для успішної розробки мобільних додатків на Swift рекомендується використовувати UIKit для створення UI, Alamofire для мережевої взаємодії і Realm для зберігання та синхронізації даних, що забезпечить оптимальну продуктивність і задоволення від кінцевого продукту.

ЛІТЕРАТУРА

1. Swift — Apple Developer. *Apple Developer*. URL: https://developer.apple.com/swift/.

2. UIKit | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/uikit.

3. Atlas Device SDK for Swift — Atlas Device SDKs. MongoDB: The Developer Data Platform | MongoDB. URL: https://www.mongodb.com/docs/atlas/device-sdks/sdk/swift/.

4. Alamofire: бібліотека для клієнт-серверних додатків на iOS | студія Brander. Створення і розробка інтернет-магазинів від Brander. URL: https://brander.ua/technologies/alamofire.

5. Nguyen, T. (2019). iOS Shopping Cart Mobile Application, *Vaasa*. 11—29. URL: https://urn.fi/URN:NBN:fi:amk-2019052812622. Nguyen, Thi (2019)

6. Chris Yerina 'Close By Places' iOS App. URL: https://digitalcommons.calpoly.edu/cpesp/275/.

7. GitHub — matteocrippa/awesome-swift: A collaborative list of awesome Swift libraries and resources. Feel free to contribute! GitHub. URL: https://github.com/matteocrippa/awesome-swift.

8. Xcode — Apple Developer. Apple Developer. URL: https://developer.apple.com/xcode/.

9. Staiano, F. (2022). *Designing and Prototyping Interfaces with Figma: Learn essential UX/UI design principles by creating interactive prototypes for mobile, tablet, and desktop*, Packt Publishing.

10. Star Movie UI Kit (Community). Figma. URL: https://www.figma.com/design/CRJZOM10ofPrxAcQcEpZY6/Star-Movie-UI-Kit-(Community)?node-id=0-1&t=eAQtiJnD7Yi8x2CL-0.

11. The Movie Database. The Movie Database (TMDB). URL: https://www.themoviedb.org/about?language=ua.

12. EPAM Campus. Як це працює? Оцінка складності алгоритмів. URL: https://rdp.epam.com/ua/blog/420.

13. UICollectionView | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/uikit/uicollectionview.

14. Caching network data | Apple Developer Documentation.Apple Developer Documentation. URL: https://developer.apple.com/tutorials/app-dev-training/caching-network-data.

15. Animation | Apple Developer Documentation. Apple Developer Documentation. URL: https://developer.apple.com/documentation/swiftui/animation.

16. GitHub — Alamofire/Alamofire: Elegant HTTP Networking in Swift. GitHub. URL: https://github.com/Alamofire/Alamofire.

17. Parsing JSON with similar identifiers. Swift Forums. URL: https://forums.swift.org/t/parsing-json-with-similar-identifiers/38556/10.