ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ НАВЧАЛЬНО-НАУКОВИЙ
ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «**ІНТЕГРАЦІЯ СУЧАСНИХ ТЕХНОЛОГІЙ У ПРОЦЕС РОЗРОБКИ ДОДАТКІВ ДЛЯ IOS ТА MACOS**»

на здобуття освітнього ступеня бакалавра

зі спеціальності 123 – Комп'ютерна інженерія
(код, найменування спеціальності)

освітньо-професійної програми _____
(назва)

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

_____ Данило ДЕНИСЮК
(підпис) Ім'я, ПРІЗВИЩЕ здобувача

Виконав: здобувач вищої освіти групи КІД-41

Данило Денисюк
Ім'я, ПРІЗВИЩЕ

Керівник: Світлана Куфтеріна
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра комп'ютерної інженерії

Ступінь вищої освіти _____

Спеціальність 123-Комп'ютерна інженерія

Освітньо-професійна програма _____

ЗАТВЕРДЖУЮ
Завідувач кафедрою _____
_____ Ім'я, ПРІЗВИЩЕ
«____» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Денисюку Данилу Эвгеновичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: _____

керівник кваліфікаційної роботи _____,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)

затверджені наказом Державного університету інформаційно-комунікаційних технологій

від «___» _____ 20__ р. № ___

2. Строк подання кваліфікаційної роботи

«___» _____ 20__ р.

3. Вихідні дані до кваліфікаційної роботи:

Онлайн-документація компанії Apple: посилання [https://developer.apple.com/]

Книга "Advanced iOS App Architecture" автор Джош Берлин та Рене Кешо

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Ознайомлення з сучасними технологіями для розробки додатків для iOS і macOS;
2. Дослідження користі використання архітектур, особливості адаптації додатків до різних версій iOS та macOS та інтеграції сучасних технологій у реальних проектах;
3. Створення додатку для операційних систем iOS і macOS з використанням сучасних технологій.

5. Перелік ілюстративного матеріалу: *презентація*

6. Дата видачі завдання «___» _____ 20__ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	20.01-27.02.2024	Виконано
2	Обробка матеріалу	27.02-02.03.2024	Виконано
3	Розробка теоретичної частини	02.03-20.04.2024	Виконано
4	Розробка практичної частини	20.01-15.04.2024	Виконано
5	Висновки за результатом аналізу	15.04-30.04.2024	Виконано
6	Розробка презентації	30.04-09.05.2024	Виконано
7	Передзахист	14.05-17.05.2024	Виконано
9	Здача в деканат	25.05-01.06.2024	Виконано

Здобувач вищої освіти _____ Наталія Лащевська
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Світлана Куфтеріна
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 74 стор., 33 рис., 0 табл., 11 джерел.

Мета роботи – розбір та дослідження технологій та інструментів для створення додатків для платформ iOS та macOS.

Об'єкт дослідження – додатки для платформ iOS та macOS.

Предмет дослідження – впровадження новітніх технологій та поглядів на проектування програм для платформ iOS та macOS.

Методи дослідження – методи теорії інформації, метод оптимального управління.

Короткий зміст роботи: досліджено та використано на практиці інструменти, сучасні технології та обґрунтовано вибір оптимальних архітектурних рішень і особливостей адаптації додатків до різних версій систем у розробці додатків для iOS та macOS.

Галузь використання – розробка кросплатформених нативних додатків для платформ iOS та macOS.

КЛЮЧОВІ СЛОВА:

- iOS та macOS додатки
- Бібліотека
- Фреймворк
- Розробка

ABSTRACT

The text part of the qualification work for the degree of bachelor's degree: 74 p., 33 figures, 0 tables, 11 sources.

The purpose of the work is to analyze and study technologies and tools for creating applications for iOS and macOS platforms.

Object of research - applications for iOS and macOS platforms.

The subject of the study is the introduction of the latest technologies and views on the design of applications for iOS and macOS platforms.

Research methods - methods of information theory, optimal control method.

Summary of the work: research and practical application of tools, modern technologies, and justification of the choice of optimal architectural solutions and features of application adaptation to different versions of systems in the development of applications for iOS and macOS.

Scope: development of cross-platform native applications for iOS and macOS platforms.

KEYWORDS:

- iOS and macOS apps
- Library
- Framework
- Development

ДЕРЖАВНИЙ УНІВЕРСИТЕТ

ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут _____

ПОДАННЯ
ГОЛОВІ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра (магістра)

Направляється здобувач(ка) _____ до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю _____
(код, найменування спеціальності)
освітньо-професійної програми _____
(назва)
на тему: «_____».

Кваліфікаційна робота і рецензія додаються.
Директор ННІ _____ _____
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач(ка) _____
_____
_____
_____

Все це дозволяє оцінити виконану кваліфікаційну роботу здобувача(ки) _____ на оцінку «_____» та присвоїти йому(їй) кваліфікацію _____.

Керівник кваліфікаційної роботи _____ _____
(підпис) *(Ім'я, ПРІЗВИЩЕ)*
«____» _____ 20___ року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач(ка) *прізвище та ініціали* допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою _____ _____ _____
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА

на кваліфікаційну бакалаврську (магістерську) роботу

здобувача(ки) вищої освіти _____

(прізвище, ім'я, по батькові)

на тему «_____»

Актуальність.

_____

_____

_____

Позитивні сторони.

1.

2.

3.

Недоліки.

1.

2.

Відзначені зауваження не впливають на загальну позитивну оцінку кваліфікаційної бакалаврської (магістерської) роботи.

Висновок: *кваліфікаційна бакалаврська (магістерська) робота заслуговує оцінку "_____", а здобувач(ка) _____ заслуговує присвоєння кваліфікації: ……..*

Рецензент: _____ _____

науковий ступінь, вчене звання *підпис* *Ім'я, ПРІЗВИЩЕ*

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ 5
ВСТУП 5
ІНСТРУМЕНТИ ТА СУЧАСНІ ТЕХНОЛОГІЇ У РОЗРОБЦІ ДОДАТКІВ ДЛЯ iOS ТА macOS 5
1.1 Основні особливості та переваги мови програмування Swift її сучасні тенденції використання у розробці додатків. 5
1.2 Інтегроване середовище розробки (IDE) Xcode та інтеграція інструментів та сервісів платформи Apple. 9
1.3 Огляд та рекомендаціїї використання Cocoa та Cocoa Touch у розробці додатків.13
1.4 Використання Interface Builder та інших інструментів та технологій для розробки інтерфейсу користувача. 16
1.5 Сучасні тренди Progressive Web Apps (PWA), Privacy and Security у розробці додатків. 19
ВИБІР АРХІТЕКТУРИ ТА ОСОБЛИВОСТІ АДАПТАЦІЇ ДОДАТКІВ ДО РІЗНИХ ВЕРСІЙ iOS ТА macOS 22
2.1 Основні принципи архітектури додатків для iOS та macOS 22
2.2 Рекомендації з побудови ефективної архітектури додатків для платформ Apple 25
2.3. Способи адаптації інтерфейсу користувача до різних розмірів екранів. 27
2.4. Забезпечення стабільності та безпеки управління версіями коду та сумісність з різними версіями операційних систем. 29
ВИКОРИСТАННЯ СУЧАСНИХ МЕТОДИК ТА ТЕХНОЛОГІЙ НА ПРИКЛАДІ ДОДАТКУ Future News 35
3.1. Опис програмної реалізації та використаних технологій. 35
3.2. Приклад роботи додатку Future News. 58
3.3. Запуск додатку на macOS та iPadOS. 63
ВИСНОВКИ 5
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

iOS	iPhone Operation System
macOS	Mach Operation System
ARC	Automatic Reference Counting
MRC	Manual Reference Counting
JSON	JavaScript Object Notation
NS	NexTSTEP
англ.	англійською
ПЗ	програмне забезпечення

ВСТУП

Зростаюча популярність мобільних пристроїв і комп’ютерів з операційними системами iOS і macOS призвела до зростання попиту на програми, призначені для різних цілей, таких як робота, навчання, спілкування та розваги. Однак розробники стикаються з різними проблемами під час створення програмного забезпечення для цих платформ, включаючи інтеграцію передових технологій, оптимізацію продуктивності, змінні вимоги ринку та вдосконалення платформи. Таким чином, існує потреба у вдосконаленні методів і технологій, що використовуються в розробці додатків для iOS і macOS, щоб інтегрувати сучасні технології, підвищувати продуктивність і пропонувати користувачам інноваційні функції.

Об'єктом мого дослідження є інтеграція сучасних технологій у сфері розробки додатків для платформ iOS і macOS.

Предмет дослідження охоплює поглиблений аналіз методів і технологій, що використовуються для створення програмного забезпечення для цих платформ, з кінцевою метою визначення найбільш ефективних підходів і методологій для їх інтеграції.

Метою моєї наукової роботи є ретельний аналіз і дослідження сучасних технологій, які використовуються в розробці додатків для платформ iOS і macOS, з метою визначення найбільш вигідних і ефективних підходів для їх впровадження. Моя основна мета полягає в тому, щоб визначити перешкоди та можливості для вдосконалення додатків на цих платформах, а також сформулювати пропозиції щодо оптимізації процесу розробки та створення

додатків за допомогою використання сучасних методологій і технологій.

Поставлені завдання моєї роботи передбачають ознайомлення з сучасними технологіями для розробки додатків для iOS і macOS, аналіз особливостей їх використання, оцінку сильних і слабких сторін, а також розкриття можливостей для посилення розробки та інтеграції цих технологій у реальні проекти.

Структура моєї роботи включає вступ, три розділи, висновок та список використаних джерел. В кожному розділі буде детально розглянуто певний аспект дослідження, проведено аналіз та надано відповідні висновки.

РОЗДІЛ 1

ІНСТРУМЕНТИ ТА СУЧАСНІ ТЕХНОЛОГІЇ У РОЗРОБЦІ ДОДАТКІВ ДЛЯ iOS ТА macOS

1.1 Основні особливості та переваги мови програмування Swift її сучасні тенденції використання у розробці додатків

Вибір мови програмування - це перший аспект, щоб почати писати додатки, програми, тощо на будь-якій платформі, і Apple у цьому плані не виключення. Тому я почну з огляду мов програмування. На відміну від інших платформ, мова програмування Swift, яка створена самою Apple, є основною мовою, що використовується для написання програм для платформ Apple. Swift — це багатопарадигмова мова програмування, розроблена Apple для заміни застарілої мови Objective-C. Перша версія була представлена у 2014 році на WWDC 2014. Ця мова була створена Крісом Літером і натхненна такими мовами, як Objective-C, C# і Python.

Тепер ознайомимося з функціями та перевагами мови Swift. Ця мова має досить чистий і стислий синтаксис, що робить код зрозумілим і легким для читання. Це сприяє швидкому розвитку та супроводу проектів. Особливу увагу розробники цієї мови приділяють безпеці. Swift використовує статичну типізацію, має механізми безпеки для запобігання помилкам під час виконання програми та має власні механізми керування пам’яттю(ARC, MRC).

Оскільки код компілюється в байт-код, Swift забезпечує високу швидкість виконання програм. Швидкість має вирішальне значення для розробки мобільних

додатків.

Мова може легко взаємодіяти з існуючими базами коду Objective-C, що дозволяє розробникам поступово переходити на нову мову без необхідності переписувати весь свій код.

Swift розроблено та підтримується Apple, що робить мову широко підтримуваною та актуальною для розробників. Але не лише Apple розвиває своє творіння. Мова Swift має спільноту розробників, які створюють бібліотеки для її підтримки. Наприклад, будь-хто може записати свої думки або повідомити про помилки на офіційній сторінці Swift на GitHub.

Swift замінює свого попередника, Objective-C. Розглянемо, чому компанія Apple вирішила перейти на нову мову. Спочатку використаємо приклад, щоб проаналізувати синтаксис цих двох мов на прикладі з Рисунку 1.1.

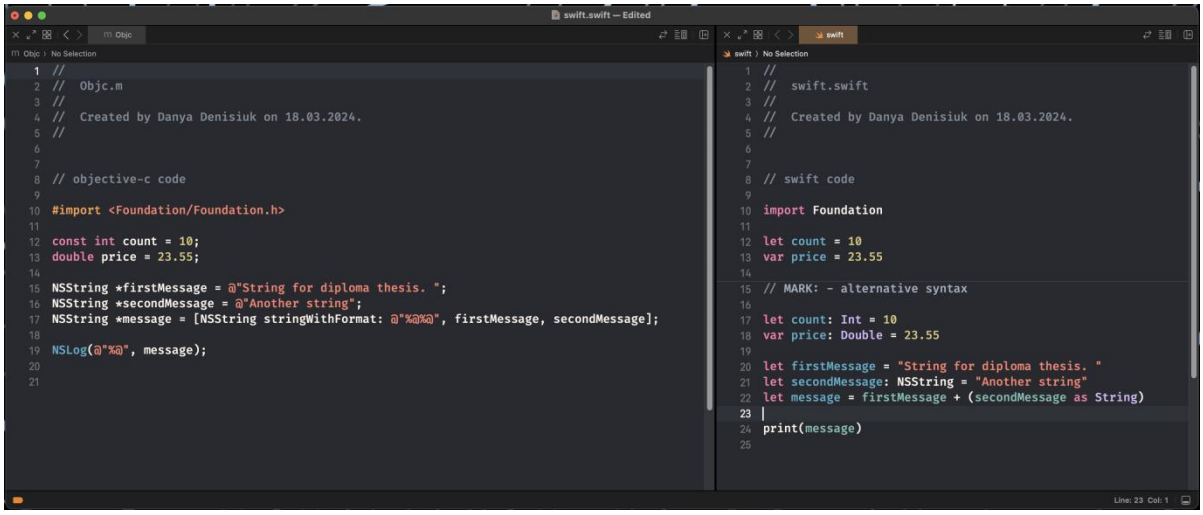


Рисунок 1.1 - Наглядний приклад у різниці синтаксису

Розглянемо відмінності синтаксису на прикладі Рисунку 1.1:

1. Порівнюючи рядки 1 і 2, ми бачимо, що тепер немає необхідності явно вказувати тип константи, хоч це робити досі можна, а в деяких випадках і

необхідно, компілятор сам розуміє переданий йому тип, але потрібно вказати, чи це константа чи змінна.

2. Крім того, хоча це все ще можливо, Apple рекомендує нам, щоб ми відмовились від спадщини C і більше не писали щоразу крапку з комою ";".

Для кращого розуміння переваг та недоліків обох мов, детальніше порівняємо Swift і Objective-C. Аспекти порівняння:

1. У порівнянні з Objective-C, який має більш складний і менш інтуїтивно зрозумілий синтаксис, Swift має більш сучасний, простий і стислий синтаксис, який робить код легшим для розуміння та читання.

2. Swift забезпечує вищий рівень безпеки завдяки статичним типам, що допомагає уникнути помилок під час виконання програми. Для порівняння, механізми безпеки Objective-C менш розвинені.

3. Swift відомий своєю високою швидкістю виконання, що робить його більш привабливим для розробки ефективних і швидких програм. Для порівняння, Objective-C може бути менш продуктивним у деяких випадках.

4. Swift активно підтримується Apple і має велике та активне співтовариство розробників, яке надає підтримку, допомогу та відкриті рішення для будь-яких проблем і питань, що виникають у процесі розробки. Однак Objective-C також користується підтримкою та великою спільнотою розробників, особливо враховуючи його довгу історію.

5. Swift має більш розширені функції для розробки інноваційних і сучасних програм, включаючи SwiftUI, Combine та інші функції. Для порівняння, Objective-C може бути обмежений у цьому відношенні.

В даний час Apple має власне бачення розвитку своїх систем і написання програмного забезпечення для них. Починаючи з 2019 року, Apple зосереджується на парадигмі програмування, яка є досить новою для неї, оскільки раніше Objective-C не пропонував «свободи кількох парадигм», як Swift, тому все було в основному написано в об’єктно-орієнтованому векторі. Зараз Apple активно розвиває наступні аспекти:

1. SwiftUI та Combine: використання SwiftUI та Combine для розробки інтерфейсів користувача та керування асинхронними подіями стає все більш поширеним. SwiftUI забезпечує декларативний підхід до створення інтерфейсів, спрощуючи розробку та підтримку коду, а Combine дозволяє ефективно обробляти потоки даних і події в реактивній парадигмі.

2. Кросплатформенність: за допомогою Swift розробники можуть використовувати загальний код або спільні компоненти для створення єдиної бази коду для програм, які працюють на різних платформах, таких як iOS, macOS, watchOS і tvOS.

3. Штучний інтелект і машинне навчання: Swift стає все більш популярним у сфері штучного інтелекту та машинного навчання. Такі фреймворки, як Core ML, дозволяють розробникам використовувати моделі машинного навчання безпосередньо у своїх програмах, створюючи нові можливості для розробки інтелектуальних програм.

4. Функціональне програмування: використання функціонального програмування в Swift дозволяє писати чистіший та ефективніший код. Функціональні підходи, такі як використання функцій вищого порядку, замикань і

незмінних структур даних, допомагають покращити якість і простоту коду.

5. Розвиток мобільних технологій. За останні роки мобільні технології зазнали значних змін і вдосконалень. Використання Swift дозволяє розробникам використовувати нові функції платформи Apple у своїх програмах.

Розглянемо перспективи розробки додатків з використанням нових версій мови програмування Swift.

Swift продовжує розвиватися та вдосконалюватися, надаючи розробникам нові можливості та зручності в процесі створення програмного забезпечення.

Кожна нова версія Swift приносить нові функції та можливості, розширюючи можливості розробника. Нові версії мови можуть містити покращення продуктивності, нові API та інструменти, підтримку нових технологій і функцій на платформах Apple, а також покращення машинного навчання та штучного інтелекту.

На момент написання цієї дипломної роботи була випущена нова версія мови Swift 5.10, і Apple покращила багатопотоковість за допомогою @MainActor, але це вже більш глибока тема.

1.2 Інтегроване середовище розробки (IDE) Xcode та інтеграція інструментів та сервісів платформи Apple

Розглянемо середовище розробки Xcode. Xcode — це інтегроване середовище розробки (IDE), розроблене Apple для розробки програмного забезпечення для платформ iOS, macOS, watchOS і tvOS. Xcode має широкий набір функцій, які допомагають розробникам створювати високоякісні програми.

Він, як зазначає компанія Apple[1], має:

1. Редактор коду, який підтримує автозавершення, підсвічування синтаксису кольором, можливість швидкого переходу між файлами та інші корисні функції.

2. Інтерфейс користувача, котрий містить різноманітні інструменти розробки інтерфейсу, такі як Interface Builder і SwiftUI, які дозволяють розробникам легко створювати та налаштовувати інтерфейси користувача.

3. Тестування інтеграції: Xcode має вбудовані інструменти для автоматизованого та ручного тестування додатків, включаючи модульне, функціональне та інші типи тестування.

4. Налагодження коду: забезпечує налагодження коду та підтримує точки зупинки, перегляд стека викликів, моніторинг змінних та інші інструменти.

5. Інструменти аналізу та оптимізації: Xcode містить інструменти для аналізу продуктивності програми, виявлення проблем продуктивності та оптимізації для підвищення продуктивності.

6. Інтеграція зі службами Apple: Xcode може легко інтегруватися з іншими службами Apple, такими як App Store Connect, TestFlight і iTunes Connect, для розгортання, тестування та керування програмами.

7. Підтримка контролю версій: підтримує інтеграцію з різними системами контролю версій, такими як Git і Subversion, забезпечуючи зручний контроль версії коду.

8. Локалізація та ресурси: Інструменти для локалізації програм і керування такими ресурсами, як зображення, звуки та інші файлові ресурси.

Щодо інтерфейсу Xcode (приклад на рис. 1.2). Він розділений на кілька

основних вікон і панелей, які надають розробникам доступ до різноманітних інструментів для створення та розробки програм для платформ Apple.

Основні елементи інтерфейсу:

1. Навігатор проекту: Ця панель дозволяє переглядати та керувати всіма файлами та ресурсами вашого проекту. Він містить такі важливі компоненти, як файли коду, зображення, звуки, ресурси інтерфейсу, налаштування проекту тощо.

2. Редактор коду: це вікно, де ви пишете та редагуєте код. Редактор коду має багато корисних функцій, таких як автозаповнення, швидкий доступ до документації, перевірка помилок, відстеження змін тощо.

3. Вікно консолі (консоль налагодження): тут під час роботи програми відображаються повідомлення про помилки, повідомлення про налагодження та інша інформація.

4. Панель інспектора: ця панель дозволяє налаштувати властивості та параметри вибраного об’єкта в проекті. Він змінюється залежно від контексту, показуючи різні налаштування для різних типів об’єктів.

5. Панель інструментів: це панель із кнопками та іншими інструментами, які забезпечують швидкий доступ до основних функцій Xcode, таких як збереження проектів, запуск і налагодження програм, вибір сценаріїв тощо.

6. Вікно конфігурації проекту: це вікно, у якому можна налаштувати різні параметри проекту, такі як цільовий пристрій, версія платформи, параметри збірки тощо.

7. Панель навігації: Ця панель забезпечує швидкий доступ до різноманітних функцій та інструментів Xcode, таких як навігація назад і вперед, швидкий пошук,

вибір макета тощо.

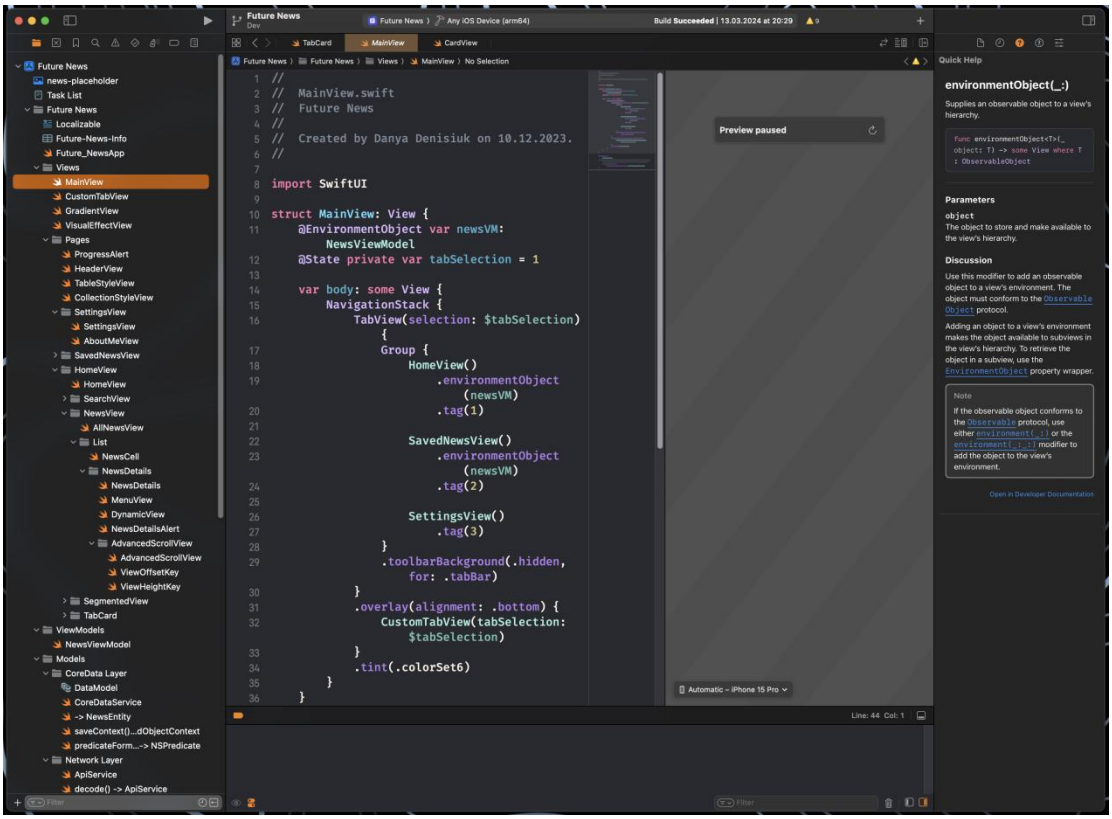


Рисунок 1.2 - Приклад відкритого вікна Xcode

Розглянемо перспективи розвитку Xcode. Майбутні доповнення та вдосконалення середовища розробки Xcode можуть ще більше спростити і покращити процес створення програм для пристроїв Apple. Для тих початківців, очікуються додаткові вдосконалення вбудованого редактора коду та інтегровані туторіали, які дозволять швидше розібратись з Xcode. Це може передбачати покращення можливостей автозавершення коду, забезпечення швидкого пошуку та навігації в проектах, а також інтеграцію з системами контролю версій.

Крім того, очікуються вдосконалення в області інтегрованого редактора візуального інтерфейсу користувача Interface Builder, який дозволить розробникам створювати більш складні та динамічні інтерфейси за допомогою простого,

зручного інтерфейсу. Майбутні вдосконалення можуть включати нові методи роботи з автоматичним макетом і обмеженнями, підтримку нових функцій і компонентів інтерфейсу, а також можливість візуалізації інтерфейсу на різних пристроях і роздільній здатності.

Також, майбутні версії Xcode можуть містити нові інструменти та функції, які підтримуватимуть поточний стан розробки програмного забезпечення та технологій. Це може включати розширену підтримку машинного навчання та штучного інтелекту, нові інтерфейси та методи роботи з доповненою та віртуальною реальністю, а також підтримку нових мов і методів програмування.

1.3 Огляд та рекомендаціїї використання Cocoa та Cocoa Touch у розробці додатків

Одними з найважливіших компонентів розробки програмного забезпечення для платформ Apple є Cocoa та Cocoa Touch. Вони надають розробникам різноманітні інструменти та компоненти, які ті можуть використовувати для створення програм графічного інтерфейсу користувача та взаємодії з різними системними службами.

Основними компоненти є:

Інтерфейс користувача:

1.AppKit (для macOS) / UIKit (для iOS): надає різноманітні класи та інструменти для створення графічних інтерфейсів, таких як вікна, кнопки, тексти, таблиці, засоби перегляду, шрифти та інші компоненти інтерфейсу.

2. Автоматичний макет: він автоматично розташовує та впорядковує

компоненти інтерфейсу відповідно до умов і обмежень, що дозволяє створювати адаптивні та адаптивні дизайни.

Робота над даними:

1. Core Data: структура для роботи з базами даних, яка дозволяє зберігати, керувати та використовувати інформацію в програмних програмах.

2. Foundation: містить класи, які полегшують роботу з рядками, масивами, словниками, датами, числами та іншими базовими типами даних.

Мультимедіа:

1. AVFoundation: надає інструменти для роботи з аудіо та відео, включаючи запис, відтворення та обробку мультимедійних даних.

2. Core Animation: дозволяє створювати складні та анімовані інтерфейси користувача за допомогою шарів, переходів і ефектів.

Комунікації:

1.NSURLSession (для iOS і macOS) / NSURLSessionDataTask (для iOS) / NSURLConnection (для macOS): полегшує виконання мережевих запитів і вилучення даних з Інтернету.

2.WebSocket: підтримка веб-сокетів, що дозволяє реалізувати зв’язок між клієнтом і сервером.

Спілкування з системою:

1. UserDefault: дозволяє зберігати та отримувати налаштування користувача за замовчуванням, як-от налаштування програми та стан інтерфейсу.

2. NotificationCenter: центр, який реалізує зв’язок між різними частинами програми або між різними програмами.

Створюючи програми для платформ Mac і iOS за допомогою фреймворків Cocoa і Cocoa Touch, важливо дотримуватися кількох правил, щоб забезпечити якість, ефективність і змінюваність проекту, а саме потрібно:

1. Використовувати архітектури проектування, такі як MVC або MVVM. Це сприяє структурованому підходу до проектування програми та полегшує розробку та підтримку програми.

2. Використовувати Auto Layout для створення універсальних і адаптивних інтерфейсів, ефективних на різних пристроях і екранах.

3. Уникати надмірного використання ресурсів пристрою, оптимізуйте спосіб завантаження та роботи з даними, щоб забезпечити ефективну та безперебійну роботу програми.

4. Використовувати тестування для оцінки функціональності та стабільності програми. Автоматизуйте тести, які забезпечують стабільну якість коду та усувають помилки.

5. Використовувати документації та інструменти для співпраці: ознайомтеся з офіційною документацією Apple і скористайтеся ресурсами спільноти розробників для вирішення проблем і отримання вказівок.

6. Оновлювати на регулярній основі програми та використовувати найновіші версії фреймворків Cocoa та Cocoa Touch, щоб використовувати нові функції та покращення.

7. Стежити за новими функціями та інноваціями в екосистемі Apple і реагувати на них, щоб підтримувати актуальність і якість своїх програм.

1.4 Використання Interface Builder та інших інструментів та технологій для розробки інтерфейсу користувача

Створюючи програми для платформ MacOS та iOS, розробники можуть використовувати різні технології та інструменти, щоб полегшити процес розробки та розширити можливості програм. Як було вказано вище, Swift має велику активну спільноту, яка також розробляє власні бібліотеки та фреймворки. Розглянемо пропрієтарні та сторонні бібліотеки та фреймворки. Почнемо з інструментів написаних компанією Apple.

Interface Builder — це візуальний інтерфейс, який полегшує створення компонентів інтерфейсу користувача програми.

Основними перевагами використання Interface Builder є:

1. Можливість створювати інтерфейс користувача шляхом перетягування компонентів у вікні редагування, що дозволяє розробникам негайно спостерігати за результатами своїх зусиль.

2. Велика кількість готових компонентів інтерфейсу, таких як кнопки, тексти, поля введення, зображення та інші, які можна легко включити та персоналізувати.

3. Простий метод створення обмежених інтерфейсів і використання автоматичного макету для створення портативного та універсального дизайну, який працюватиме на екранах різного розміру.

4. Легке керування ресурсами програми, такими як зображення, сімейство шрифтів і текстові рядки. Крім того, інтегровані інструменти локалізації полегшують створення кількох версій програми для різних мов і культур.

5. Можливість попереднього перегляду та тестування інтерфейсу користувача безпосередньо в середовищі розробки, що дає змогу швидко оцінити його зовнішній вигляд і функціональність.

6. Легкий спосіб підключення коду до самого Interface Builder, як показано на Рисунку 1.3, прив’язавши елементи інтерфейсу до класів і методів програми, це дозволить вам реалізувати функціональність інтерфейсу в коді вашої програми.

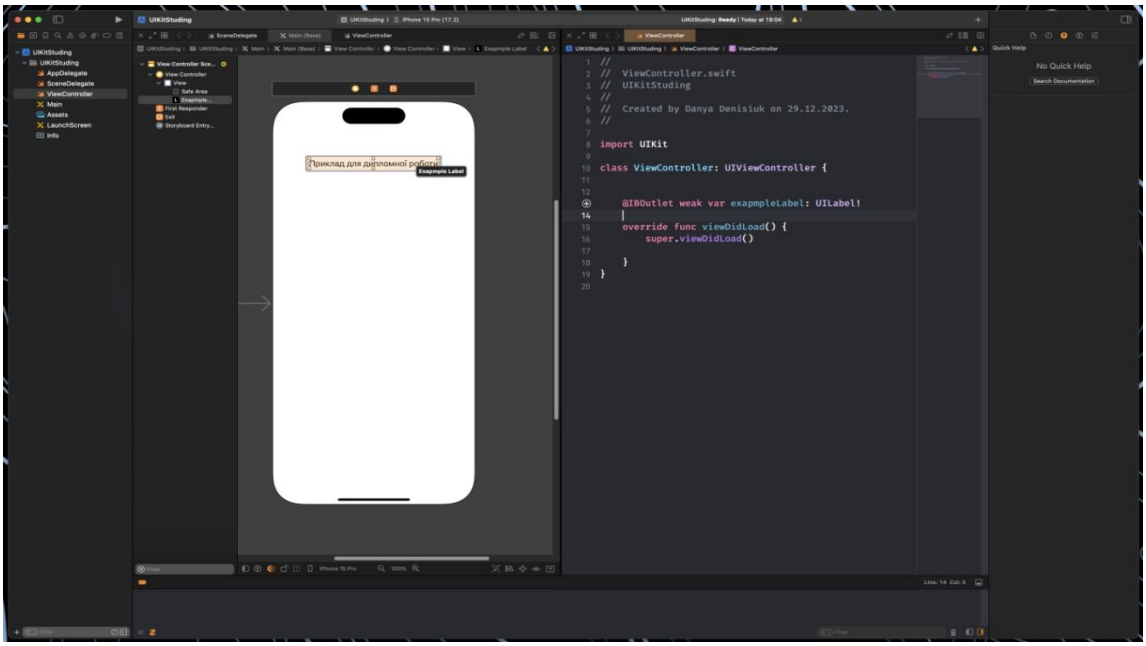


Рисунок 1.3 - Приклад інтеграції Interface Builder та коду

Крім того, можна відзначити сторонні бібліотеки та фреймворки, найпопулярніші з яких:

1. Firebase: мобільна та веб-платформа для розробки баз даних, яка пропонує такі послуги, як аналітика, автентифікація користувачів, бази даних у реальному часі, зберігання файлів тощо.

2. GraphQL: мова для запитів API, яка дозволяє розробникам отримувати лише необхідну інформацію, ця мова зменшує обсяг мережевого трафіку та покращує продуктивність програм.

3. SnapKit: стороння оболонка, яка забезпечує більш зручний дизайн інтерфейсу.

4. RXSwift: сторонній фреймворк реактивного програмування в Swift, який полегшує керування даними та хід подій програмі.

5.Realm: Мобільна база даних, яка забезпечує простий і швидкий спосіб зберігання та керування інформацією у вашому додатку.

Щоб підключити сторонні бібліотеки, ви повинні використовувати Swift Package Manager або CocoaPods для завантаження та використання сторонньої бібліотеки. Їх використання подібне до пропрієтарних рішень.

Звичайно, використання сторонніх бібліотек має певні мінуси. Хоч і можна знайти вирішення проблем на GitHub, додаючи та використовуючи все більше сторонніх бібліотек у проекті, але це погано для великих програм. Бо, як правило, за оновлення, виправлення помилок та підтримку бібліотеки, в першу чергу, відповідає спільнота. Як наслідок, якщо проект має серйозний зв’язок із певною бібліотекою, і ця третя сторона дуже мало оновлює бібліотеку, зрештою доведеться переписувати значну частину коду, деінтегруючи старі залежності від сторонньої бібліотеки.

Наприклад, раніше згадувалось 2 фреймворки: RXSwift і Combine. Ці фреймворки використовуються для додавання реактивного підходу до вашої програми. Оскільки Combine став доступним лише у 2019 році, до цього часу розробники шукали вихід, так як потрібно було знайти рішення для включення реактивного підходу в їх додаток і почали масово використовувати RXSwift. Але згодом розробники, які використовували RXSwift, все ж таки почали

переписувати свої додатки на фреймворк Combine. Так як Combine є фреймворком, розробленим Apple, це забезпечує певну впевненість у тому, що ця технологія, ймовірно, буде підтримуватися. Обставини ідентичні з Alomafire, KingFisher та багатьма іншими сторонніми бібліотеками та інструментами, які обговорювались раніше. У результаті, коли розробник заощаджує кілька днів на написання рішення і використовує для цього сторонні бібліотеки, з часом він виратратить сотні годин, щоб «розірвати залежність» проекту від конкретної бібліотеки.

1.5 Сучасні тренди Progressive Web Apps (PWA), Privacy and Security у розробці додатків

Обговоримо сучасні тенденції розробки додатків на прикладі різних сфер, включаючи прогресивні веб-додатки (PWA), темний режим, конфіденційність і безпеку.

Сучасні технології розробки додатків для iOS і macOS надають розробникам масу можливостей для творчості та інновацій. Використання цих принципів і методів дозволяє створювати програми, які не тільки задовольняють запити користувачів, але й мають найвищі стандарти якості та продуктивності.

Сьогодні розробка програм для iOS і macOS стає все більш складною і різноманітною, але є кілька важливих тенденцій які впливають на розробку програмного забезпечення для цих платформ. Однією з найпопулярніших тенденцій є зростання популярності веб-додатків. Розробники все частіше використовують такі веб-технології, як JavaScript, HTML і CSS, для створення

програм, які можна запускати в браузері на різних пристроях, включаючи iPhone, iPad і Mac. Це сприяє швидкій розробці та розповсюдженню програм. Сучасні програми розвиваються з дедалі більшою пристрастю до прогресивних веб-програм: PWA. Ця технологія дає змогу створювати додатки, які можна запускати безпосередньо у браузері, але мають функції, подібні до нативних програм. Використання технологій веб-розробки для створення PWA дозволяє розробникам створювати програми, які можна запускати на різних пристроях, включаючи iOS і macOS.

Однією з переваг PWA є їх доступність на всіх платформах, що зменшує час і зусилля, необхідні для створення окремих версій програм для кожної платформи. Крім того, PWA можна оновлювати безпосередньо через мережу, цей процес спрощений і гарантує, що користувачі завжди матимуть останню версію.

Хоча PWA сприяють економії коштів, пов’язаних із розробкою та підтримкою програм, які є кросплатформними, вони можуть не мати всіх функцій, наявних у нативних програмах. Наприклад, відсутність взаємодії з операційною системою та відсутність доступу до апаратних функцій пристроїв. Інші проблеми включають продуктивність і стабільність, ці проблеми особливо актуальні під час роботи в автономному режимі.

«Apple - дорівнює безпека». важливо забезпечити високий рівень конфіденційності та безпеки даних у програмах, а також у будь-якій сучасній програмі, створеній на платформах Apple. Зрештою, якщо ви спробуєте отримати доступ до даних користувача без його дозволу,то програма не матиме доступу до Apple AppStore. Користувачі вважають, що Apple захистить їхні особисті дані та

не використовуватиме їх без їхнього дозволу. Тому розробники повинні звертати увагу на ці принципи та враховувати їх під час створення та вдосконалення програм. Щоб зберегти конфіденційність даних користувача, розробники можуть використовувати різні методи. Наприклад, дуже важливо включити в програму політику конфіденційності, яка описує дані, які збираються та як вони використовуються, а також можливість, на розсуд користувача, стерти всі особисті дані. Також важливо враховувати принципи обмеження доступу до даних і важливість шифрування конфіденційної інформації, щоб запобігти несанкціонованому доступу.

Що стосується безпеки даних, розробники повинні бути знайомі з останніми тенденціями та найкращими методами кібербезпеки. Це передбачає регулярне впровадження нового програмного забезпечення для усунення вразливостей, використання безпечних методів автентифікації та авторизації, а також спостереження за діяльністю користувачів для виявлення підозрілої поведінки.

Загалом, підтримка високого рівня конфіденційності та безпеки даних у програмах є дуже важливим для розробників. Вони повинні активно намагатися запобігти потенційній небезпеці та вжити всіх практичних заходів для захисту особистої інформації користувачів.

РОЗДІЛ 2

ВИБІР АРХІТЕКТУРИ ТА ОСОБЛИВОСТІ АДАПТАЦІЇ ДОДАТКІВ ДО РІЗНИХ ВЕРСІЙ iOS ТА macOS

2.1 Основні принципи архітектури додатків для iOS та macOS

Проектуванням архітектури програми має вирішальне значення для розробки програмного забезпечення. Дизайн додатків для операційних систем Apple має низку специфічних особливостей, пов’язаних із використовуваними фреймворками та бібліотеками. Існує два основні підходи до написання програм: UIKit (або AppKit для macOS) і SwiftUI. У кожному з них передбачається використання певних архітектурних підходів. Наприклад, фреймворк SwiftUI насамперед рекомендується використовувати разом з архітектурою MVVM (Model-View Model-View), яикй обговоримо пізніше. Крім того, потрібно пам’ятати про інші аспекти архітектури, окрім структуризації дизайну самої програми, а саме використання патернів, таких як: Observer, Delegate, Singleton та інші, використання Dependency Injection залежностей та інших.

Як вже згадувалось раніше, перший крок у виборі архітектури структурування проекту базується на обраному фреймворкі: UIKit (або AppKit для macOS) або SwiftUI. Обговоримо деякі фундаментальні принципи, які служать керівництвом для розробників, які хочуть створювати ефективні та стабільні програми для iOS і macOS.

Одним із основних принципів є модель-представлення-контролер (MVC), який полегшує розрізнення між логікою програми та її поданням і даними.

Model-View-Controller (MVC)[2] є одним із найпопулярніших архітектурних шаблонів у розробці програмного забезпечення, включаючи програми для платформи Apple. Ця архітектура сприяє збільшенню читабельності та зручності обслуговування коду в майбутньому. Ця архітектура в основному використовується разом із фреймворком UIKit.

Розглянемо кожен з компонентів архітектури MVC:

1. Модель (Model) - відповідає за дані та логіку програми.

2. Вид (View) - полягає в тому, щоб показати цю інформацію користувачеві.

3. Контролер (Controller) - відповідає за обробку взаємодії користувача з програмою та керування потоком даних між моделлю та представленням.

Незважаючи на популярність MVC і його широке використання в розробці для комп'ютерів Apple, є недоліки. У великих і складних програмах може виникнути проблема з розрізненням відповідності між моделлю, представленням і контролером, що може призвести до «переплутаного» коду та збільшення складності. Крім того, MVC може призвести до контролерів, які мають багато логіки, що не завжди є найефективнішим підходом до підтримки акуратного та легко змінюваного коду, але спочатку розглянемо як працює фреймворк UIKit.

В UIKit перспектива елемента інтерфейсу користувача включена в спадкоємця UIViewController, який також є ,так би мовити, полотном, на якому розташовані спадкоємці UIView. В результаті виявляється, що наш контролер і представлення тісно пов’язані, тому чим складніше «полотно», тим більшим стане наш контролер.

Apple визнає проблеми з MVC в UIKit і пропонує альтернативні методи розробки програм для своїх платформ. Як приклад:

1. Model-View-ViewModel (MVVM) — це шаблон архітектури, який Apple активно просуває для створення програм. Це відокремлює логіку представлення даних (ViewModel) від їх відображення (View), це дає змогу мати менше контролерів (зазвичай контролери в MVVM мають меншу функціональність, ніж у MVC) і простіші моделі, але щоб досягти цього, потрібно додати у проект реактивне програмування за допомогою Combine або RXSwift.

2. Шаблон Coordinator - цей шаблон полегшує керування навігацією та координацію переходів між різними екранами програми, зменшує залежність від контролерів і полегшує перехід на нові екрани.

3. Архітектура Redux або Flux — це форма архітектури, яка є похідною від застосування управління станом і поділу компонентів на чисті ізольовані компоненти. Незважаючи на те, що ці компоненти не вбудовані в структуру Apple, ці компоненти все одно можна використовувати для створення ефективних і простих в управлінні програм.

4. Ще одна важлива концепція — це Clean Architecture, яка дозволяє розділити програму на рівні та зменшити залежність між цими рівнями за допомогою абстракцій. Це полегшує зміну та еволюцію програмного забезпечення в майбутньому. Це ефективна архітектура, яка з’явилась не так давно, але яку легко використовувати з фреймворками як SwiftUI так і UIKit. Однак слід визнати, що ця архітектура в першу чергу підходить для великих проектів, які мають значну кількість коду, оскільки використання багатьох різних абстракцій може зменшити

читабельність коду.

5. Впровадження залежностей (DI) — ще один важливий принцип, який полегшує вашу здатність краще керувати залежностями між класами та забезпечує більшу гнучкість і зменшення коду.

2.2 Рекомендації з побудови ефективної архітектури додатків для платформ Apple

Щоб створити ефективну архітектуру для платформ Apple, необхідно врахувати кілька важливих аспектів. Обговоримо декілька з них:

1. Спочатку потрібно використовувати такі шаблони проектування, як MVC, MVVM або Clean Architecture, щоб створити впорядкований і легко змінюваний код. Ці шаблони полегшують співпрацю між різними частинами програми, зберігаючи при цьому модульність проекту.

2. Важливо розбити додаток на окремі модулі за функціоналом, це полегшить розробку та супровід. Цей метод сприяє більш ефективному управлінню різними аспектами програми та зменшує залежність між компонентами. Хорошим тоном вважається використання коду для повторного використання іншими розробниками.

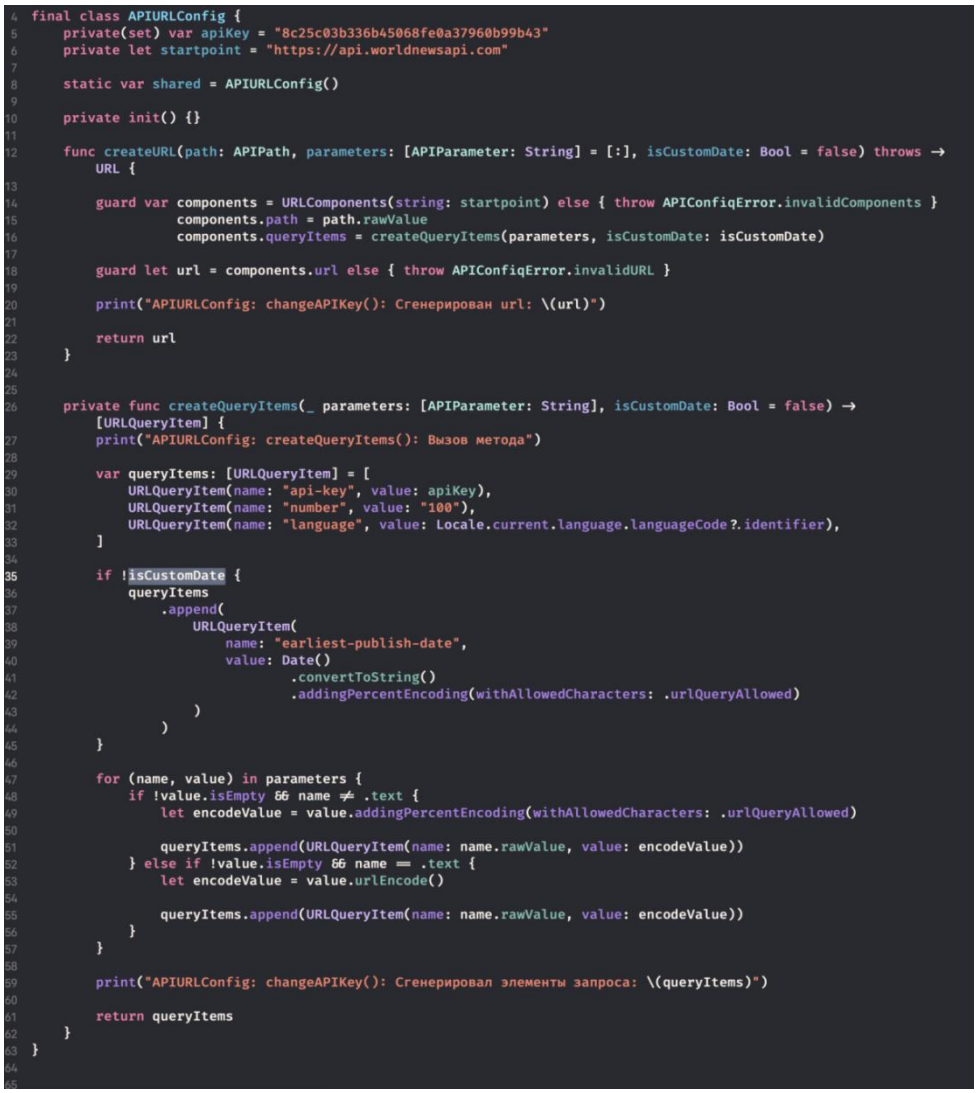
3. Важливо використовувати абстракції, щоб зберегти чистоту коду та зменшити залежність між компонентами. Це сприяє більш пластичному та змінюваному коду, який легше тестувати та підтримувати.

4. Продуктивність і оптимізацію пам'яті. Під час циклу розробки програми потрібно уникати надмірного навантаження на ресурси та знаходити ефективні

методи для виконання завдань, щоб забезпечити швидку та ефективну роботу програми.

5. Тестування програм на різних етапах розробки, включаючи функціональне, модульне та інтеграційне тестування. Це полегшує виявлення та виправлення помилок на ранніх стадіях розробки та гарантує послідовну роботу програми.

Тепер обговоримо шаблони на прикладі найпопулярнішого представника - шаблону Singleton. Цей шаблон дозволяє створювати класи, які мають лише один екземпляр, що корисно для об'єктів, які повинні бути доступні в усій програмі.
Ось зразок коду, який використовує шаблон Singleton, який реалізувано додатку Future News, який буде розглядатись в розділі 3.



```swift
final class APIURLConfig {
    private(set) var apiKey = "8c25c03b336b45068fe0a37960b99b43"
    private let startpoint = "https://api.worldnewsapi.com"

    static var shared = APIURLConfig()

    private init() {}

    func createURL(path: APIPath, parameters: [APIParameter: String] = [:], isCustomDate: Bool = false) throws ->
        URL {

        guard var components = URLComponents(string: startpoint) else { throw APIConfiqError.invalidComponents }
                components.path = path.rawValue
                components.queryItems = createQueryItems(parameters, isCustomDate: isCustomDate)

        guard let url = components.url else { throw APIConfiqError.invalidURL }

        print("APIURLConfig: changeAPIKey(): Сгенерирован url: \(url)")

        return url
    }


    private func createQueryItems(_ parameters: [APIParameter: String], isCustomDate: Bool = false) ->
        [URLQueryItem] {
        print("APIURLConfig: createQueryItems(): Вызов метода")

        var queryItems: [URLQueryItem] = [
            URLQueryItem(name: "api-key", value: apiKey),
            URLQueryItem(name: "number", value: "100"),
            URLQueryItem(name: "language", value: Locale.current.language.languageCode?.identifier),
        ]

        if !isCustomDate {
            queryItems
                .append(
                    URLQueryItem(
                        name: "earliest-publish-date",
                        value: Date()
                                .convertToString()
                                .addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)
                    )
                )
        }

        for (name, value) in parameters {
            if !value.isEmpty && name != .text {
                let encodeValue = value.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed)

                queryItems.append(URLQueryItem(name: name.rawValue, value: encodeValue))
            } else if !value.isEmpty && name == .text {
                let encodeValue = value.urlEncode()

                queryItems.append(URLQueryItem(name: name.rawValue, value: encodeValue))
            }
        }

        print("APIURLConfig: changeAPIKey(): Сгенерировал элементы запроса: \(queryItems)")

        return queryItems
    }
}
```

Рисунок 2.1 - Приклад реалізації патерну Singleton

Як побачимо, цей клас-одинак відповідає за створення URL-посилання до API, детальніше про те, що таке URL та API поговоримо в **Розділі 3.**

2.3. Способи адаптації інтерфейсу користувача до різних розмірів екранів

Перетворення інтерфейсу користувача на різні версії операційних систем має вирішальне значення для розробників програмного забезпечення. Це важливо для забезпечення простої та приємної взаємодії між користувачами та програмами, незалежно від версії операційної системи, яку вони використовують.

Одним із основних способів зміни інтерфейсу є використання стандартів і

рекомендацій, наданих розробникам виробником операційної системи. Наприклад, Apple дає розробникам правила дизайну, які спрямовують їх на загальний і узгоджений інтерфейс користувача [3]. Важливо уникати використання компонентів інтерфейсу, які більше не актуальні або змінилися в нових версіях операційної системи. Це полегшує сумісність із майбутніми оновленнями та запобігає проблемам із зображенням.

Динамічний дизайн є ще одним важливим компонентом еволюції інтерфейсу. Це полегшує створення інтерфейсу, який буде автоматично адаптуватися до різних розмірів екрана та пристроїв, це забезпечить належний вигляд інтерфейсу незалежно від пристрою.

Використання автоматичного макета та інших обмежень є корисним для створення адаптивного інтерфейсу користувача для різних пристроїв і розмірів екранів. Auto Layout полегшує розміщення правил, які вказують роз ташування та розміркомпонентів інтерфейсу, які автоматично адаптуються до змін розміру екрана чи орієнтації пристрою.

Обмеження визначають взаємозв’язки між компонентами інтерфейсу, наприклад відстань між компонентами, їхній розмір і положення на екрані. Їх можна створювати за допомогою інтерфейсу користувача як інструменту проектування, наприклад Interface Builder у Xcode, або за допомогою коду.

Використовуючи правила автоматичного макетування та дизайну, розробники можуть створити інтерфейс, який добре відображатиметься та працюватиме на пристроях різного розміру, включаючи iPhone, iPad та Mac. Наприклад, за допомогою обмежень можна вказати, що кнопка завжди повинна бути

розташована посередині екрана, або що текстове поле має адаптуватися до розміру екрана пристрою.

Розробка програмного забезпечення для підтримки екранів різного розміру та роздільної здатності має вирішальне значення для розробників програмного забезпечення. Доступно кілька методів і підходів, які допомагають забезпечити ефективне відтворення та функціональність на пристроях із різною роздільною здатністю екрана. Одним із методів є використання адаптивного дизайну. Цей підхід використовується для створення дизайну, який буде автоматично адаптуватися до різних розмірів екрану та роздільної здатності. Інший метод полягає в ретельному проектуванні та розміщенні компонентів інтерфейсу. Розробникам слід розглянути, як розташувати елементи, щоб вони були легко сумісні з екранами різних розмірів. Важливо уникати надмірного навантаження на екран невідповідними компонентами та забезпечити чітку ієрархію відображення.

2.4. Забезпечення стабільності та безпеки управління версіями коду та сумісність з різними версіями операційних систем.

Контроль версії коду має вирішальне значення для розробки програмного забезпечення. Це полегшує ефективне керування змінами коду, відстеження прогресу проекту та забезпечення сумісності з різними версіями операційних систем.

Для керування версіями коду розробники часто використовують програмне забезпечення для контролю версій, наприклад Git. За допомогою Git розробники можуть створювати копії свого коду ("коміти"), вести журнал змін між комітами

та комбінувати різні гілки розробки проекту. Також, для забезпечення сумісності з різними версіями операційних систем розробники можуть використовувати директиви препроцесора та умовну компіляцію у своєму коді, як показано на Рисунку 2.2. Це полегшує включення або виключення певних частин коду залежно від версії операційної системи, з якою він пов’язаний.

Наприклад, якщо розробник створює програму для iOS і macOS, він може використовувати умовну компіляцію, як показано на Рисунку 2.2, щоб включити код, який є специфічним для певних версій цих операційних систем. Це полегшує асоціацію з різними версіями систем і використання нових функцій, які доступні лише в останніх версіях.

Як правило, контроль версій коду та процес забезпечення сумісності з різними версіями операційної системи є найважливішими для розробки програмного забезпечення. Використання систем контролю версій і умовної компіляції допомогло розробникам ефективно керувати розвитком своїх проектів і забезпечити належну роботу програм на різних платформах і версіях операційних систем.

Визначаючи мінімально необхідну версію iOS і macOS, розробники зазвичай враховують кілька факторів. По-перше, вони вивчають поширення конкретних версій операційних систем серед користувачів. Для iOS це надзвичайно важливо через часті оновлення, якими користуються юзери. Далі розробники оцінюють нові функції або вдосконалення API, які доступні в кожній новій версії, і чи важливі ці функції для програми, яку вони розробляють. Як правило, мінімально необхідна версія призначена для охоплення великої частини користувачів, але все

одно використовуються новіші функції API, коли це можливо. Розробники також можуть використовувати такі інструменти, як діагностична інформація, щоб визначити, які версії операційної системи вони можуть підтримувати, не жертвуючи значною частиною своєї цільової аудиторії.

Якщо деякі функції програм не підтримуються певними версіями операційних систем, розробники можуть розглянути можливість розробки альтернативних рішень. Це може включати створення функцій, які можуть повернутися назад, або використання альтернативних програм чи бібліотек для забезпечення сумісності з різними версіями операційних систем, як показано на прикладі на Рисунку 2.2. Наприклад, якщо нова функція ще не доступна в старіших версіях iOS, розробники можуть розглянути можливість використання сторонніх бібліотек або власних алгоритмів для досягнення тієї ж функціональності.

```
func reloadWidgets() {
    if #available(iOS 14, macOS 11, *) {
        WidgetCenter.shared.reloadAllTimelines()
    } else {
        updateTodayViewWidgetData()
    }
}
```

Рисунок 2.2 - Приклад доступності нових версій у додатку

Також, розробники беруть до уваги оновленні правила людського інтерфейсу, або як в оригіналі Human Interface Guideline[3], які можуть змістовно змінити підхід та стиль користування додатком. Якщо коротко, адже це дуже велика тема, рекомендації щодо людського інтерфейсу (HIG) або Human Interface Guidelines — це набір правил, принципів і вказівок, створених компанією Apple, які описують найкращий спосіб проектування та розробки інтерфейсів користувача для їхніх продуктів, включаючи iOS, macOS, watchOS і tvOS. Приклад макету Human Interface Guidelines зображено на Рисунку 2.3.

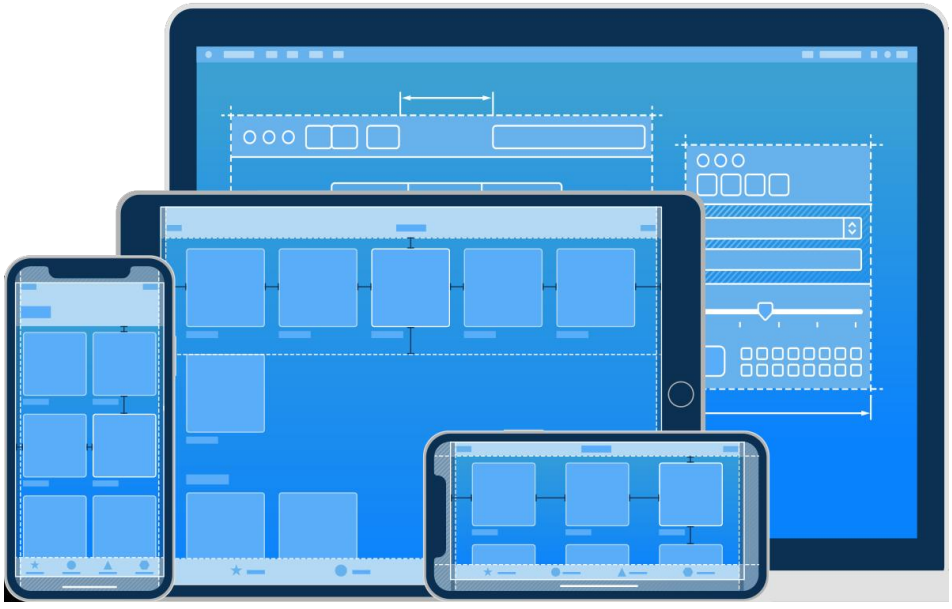


Рисунок 2.3 - Приклад макету Human Interface Guidelines

Ці правила включають різноманітні аспекти дизайну інтерфейсу, включаючи розміщення компонентів у програмі, використання кольорів, типографіки, анімації та значків, а також взаємодію з користувачем за допомогою різних жестів

і дій.

Основною метою HIG є сприяння одноманітності та простоті використання на платформах Apple, а також забезпечення високої якості та позитивного досвіду для всіх користувачів. Розробники, які дотримуються цих правил, можуть створювати програми, які прості для розуміння та використання широким колом користувачів.

Тепер розглянемо, як розробники досягають стабільності та безпеки під час оновлень. Це є одним із найважливіших обов'язків розробників операційних систем, платформ і програм. Випускаючи нові версії системи або програмного забезпечення, важливо переконатися, що вони не призводять до помилок або збоїв у роботі системи чи програмного забезпечення, не порушують безпеку користувачів.

Щоб забезпечити стабільність оновлень, розробники ретельно тестуватимуть нові версії перед їх випуском. Це передбачає перевірку функціональності програми, її взаємодії з іншими програмами та пристроями, а також можливі помилки чи проблеми з безпекою. Потім вони мають додати важливі механізми для відновлення даних і вирішення проблем, які дозволяють користувачам відновити свої дані, якщо під час оновлення виникнуть проблеми. Це може включати автоматичне резервне копіювання, яке йде перед оновленням, або можливість повернути вашу систему до попередньої версії.

Отже, виявлення та усунення проблем і помилок, пов'язаних з оновленнями, має важливе значення для забезпечення безперебійної роботи як операційних систем, так і програм. Під час оновлень можуть виникати різні проблеми, зокрема

проблеми із програмами сторонніх розробників, невдалі оновлення або випадкові помилки, які виникають через зміни в операційній системі чи самій програмі. Розробники використовують різні підходи до виявлення проблем і помилок під час оновлень, включаючи тестування на різних пристроях і конфігураціях, відстеження звітів про помилки користувачів, аналіз журналів подій і діагностичних даних. Виявивши проблеми, розробники дослідять причину проблеми та розроблять план усунення. Це може включати випуск виправлень або оновлень, які вирішують виявлені проблеми, або роботу з іншими розробниками чи постачальниками для вирішення проблем сумісності.

РОЗДІЛ 3

ВИКОРИСТАННЯ СУЧАСНИХ МЕТОДИК ТА ТЕХНОЛОГІЙ НА ПРИКЛАДІ ДОДАТКУ Future News

3.1. Опис програмної реалізації та використаних технологій

Зрештою, розглянемо головну частину роботи. Розглянемо додаток, який використовує деякі з найновіших фреймворків, бібліотек і методів програмування.

Додаток Future News — це додаток для пошуку та читання новин, яка має на меті надавати користувачам актуальну та цікаву інформацію. Основна мета програми — надати користувачам простий досвід читання новин.

Future News пропонує зручний спосіб збереження важливих статей у базу даних для подальшого перегляду із можливістю пошуку за ключовими словами чи темами, а також підтримує обмін цікавими новинами з іншими. Ця програма надає доступ до різноманітних джерел новин, а також допомагає користувачам знайти саме ту інформацію та деталі, які його цікавлять.

Дизайн архітектури додатку, написаний на основі SwiftUI з використанням архітектури MVVM (Model-View-ViewModel), є сучасним з точки зору ефективності. У цьому дизайні різні компоненти відповідають за свої власні обов’язки, що дає змогу створювати організований і легко зрозумілий код. Схематичне зображения архітектури можна побачити на Рисунку 3.1.

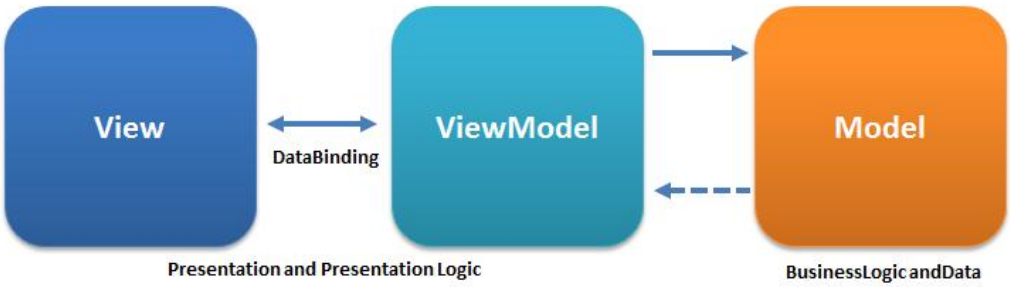


Рисунок 3.1 - Схематичне зображення архітектури MVVM

Розглянемо дизайн MVVM більш детально. MVVM [4], який також відомий як Model-View-ViewModel, — це шаблон проектування програмного забезпечення, який використовується для створення інтерфейсів користувача. Вперше він був випущений як частина концепції шаблонної архітектури, яка була призначена для використання з платформою WPF під час розробки платформи в 2005 році. MVVM складається з трьох основних частин: моделі, представлення та моделі перегляду.

Почнемо з моделі. Модель — це концептуальний компонент, який містить усю бізнес-логіку, обробку даних і взаємодію із зовнішніми ресурсами, такими як сервери або бази даних. Його можна реалізувати у формі класів або структур, які представляють структуровані дані та методи їх обробки. Приклад структуризації на Рисунку 3.2.

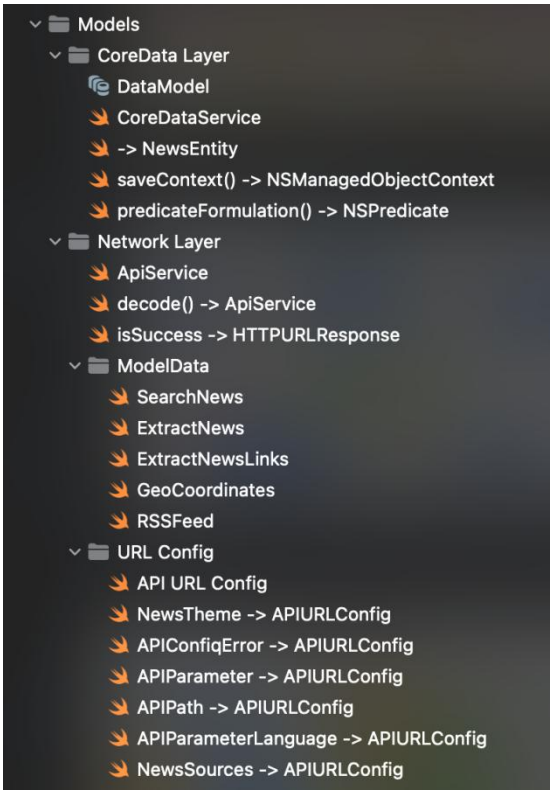


Рисунок 3.2 - Приклад структуризації Моделей у проекті

Наступним компонентом є View. У SwiftUI інтерфейс користувача описується декларативно за допомогою різних компонентів і макетів. View відповідає за представлення інформації та спілкування з користувачем. Він отримує інформацію від ViewModel і переглядає її простим для розуміння способом. Приклад структуризації на Рисунку 3.3.

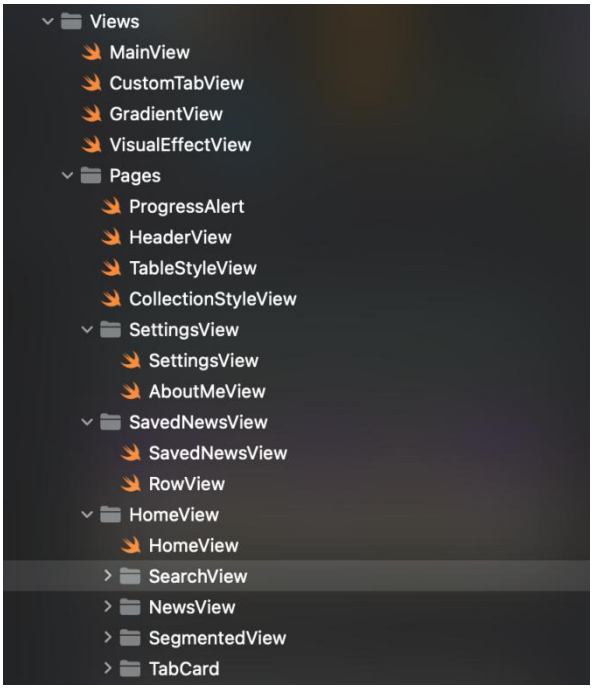


Рисунок 3.3 - Приклад структуризації Видів у проекті

Останнім компонентом є ViewModel. ViewModel використовується для зберігання та обробки даних, пов’язаних із View, а також для виконання логіки, пов’язаної з взаємодією користувача. Це відокремлює логіку представлення від даних і дозволяє підключати різні типи даних до різних точок зору.

MVVM полегшує керування складними програмами, розділяючи відображення логіки та даних. Це полегшує тестування, обслуговування та розширення програм. Це об’єднання створює потужну комбінацію дизайну MVVM із SwiftUI, яка ідеально підходить для розробки додатків на платформі Apple.

Проект «Future News» використовує кілька методів і шаблонів, які призначені для сприяння розвитку та вдосконалення програми.

Почнемо з шаблону проектування Singleton, який використовується для створення окремого екземпляра об’єкта, наприклад менеджерів даних, до яких можна отримати доступ із будь-якого місця в програмі. Приклад на Рисунку 3.4.

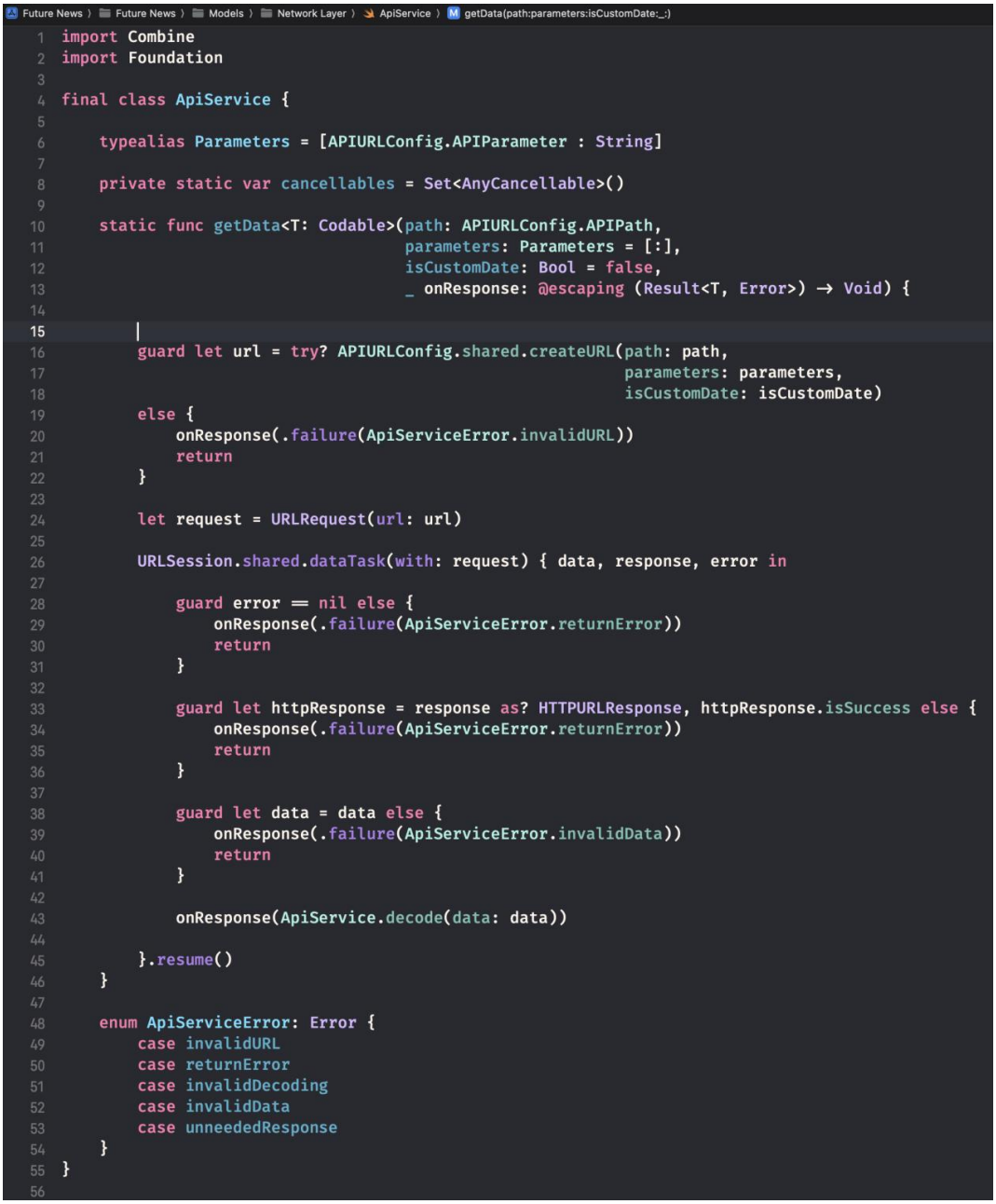


Рисунок 3.4 - Приклад використання патерну Singleton у додатку

Технологія створення мережі базується на фреймворку для URLSessions, який використовується для забезпечення взаємодії з сервером через мережу. Це

полегшує прийом і передачу даних по мережі, що забезпечує зв'язок між додатком і сервером.

CoreData використовується для полегшення взаємодії між локальною базою даних і даними програми. Це зручно для зберігання та впорядкування великих обсягів інформації, як-от архівних новин або налаштувань користувача.

Для програмування, яке є асинхронним і має реактивний підхід, використовується фреймворк Combine, цей фреймворк полегшує керування потоками даних і подіями. Це полегшує обробку асинхронних дій швидше та ефективніше.

Використання SwiftUI для створення інтерфейсу користувача також має велике значення. Цей декларативний підхід полегшує створення ефективних і привабливих інтерфейсів, зменшує кількість необхідного коду та спрощує процес розробки.

Зрештою, Grand Central Dispatch (GCD) використовується для ефективного розподілу та спрямування завдань між різними потоками, ця процедура підвищує ефективність програми та її продуктивність.

Почнемо розгляд інтсрументів з фреймворку SwiftUI [5], про який згадувалось вже декілька разів. Це новий фреймворк від Apple, який полегшує створення інтерфейсів користувача в програмах для всіх платформ iOS, macOS, watchOS і tvOS. Одним із основних атрибутів SwiftUI є використання декларативного стилю опису інтерфейсу. Замість традиційного імперативного підходу, який передбачає опис кожного кроку створення та зміни інтерфейсу, SwiftUI дозволяє описати бажаний вигляд інтерфейсу в кожному стані, а

фреймворк автоматично визначить, як цього досягти. На Рисунку 3.5 наведено приклад коду структури цього фреймоворку.

```swift
//  MainView.swift
//  Future News
//
//  Created by Danya Denisiuk on 10.12.2023.
//

import SwiftUI

struct MainView: View {
    @EnvironmentObject var newsVM: NewsViewModel
    @State private var tabSelection = 1

    var body: some View {
        NavigationStack {
            TabView(selection: $tabSelection) {
                Group {
                    HomeView()
                        .environmentObject(newsVM)
                        .tag(1)

                    SavedNewsView()
                        .environmentObject(newsVM)
                        .tag(2)

                    SettingsView()
                        .tag(3)
                }
                .toolbarBackground(.hidden, for: .tabBar)
            }
            .overlay(alignment: .bottom) {
                CustomTabView(tabSelection: $tabSelection)
            }
            .tint(.colorSet6)
        }
    }
}
```

Рисунок 3.5 - Приклад використання SwiftUI

SwiftUI надає різноманітні модифікатори та макети, які можна використовувати для створення різних інтерфейсів, включаючи кнопки, текстові поля та складні списки, усі з яких підтримуються анімацією. Крім того, SwiftUI поєднується з іншими технологіями від Apple, такими як Combine, яка використовується для реалізації реактивного програмування, і Core Data, яка використовується для керування базою даних.

Однією з головних переваг SwiftUI є простота використання та створення. Декларативний підхід до програмування дозволяє створювати інтерфейс з

мінімальною кількістю коду, що робить процес розробки швидше і дешевше. Крім того, SwiftUI автоматично покращує інтерфейс для різних пристроїв і режимів, що забезпечує узгоджену роботу користувача на всіх платформах. SwiftUI відкрив нові можливості для розробників. Це призвело до збільшення простоти створення та продуктивності інтерфейсів. Вибираючи цей фреймворк, ми маємо впевненість у можливості використовувати архітектуру MVVM без великої кількості концептуалізацій, оскільки цей фреймворк створений з урахуванням реактивного програмування, яке задовольняє основній вимозі MVVM.

Розглянемо використану базу даних CoreData [6]. Це структура, яка використовує базу даних для роботи на пристроях iOS, macOS, watchOS і tvOS, що полегшує розробникам можливість створювати та керувати об’єктами даних у програмах. Ці об’єкти можна використовувати для зберігання, отримання, оновлення або видалення даних. Використання та конфігурація бази даних цілком можливе в графічному редакторі, як показано в прикладі на рисунку 3.6.

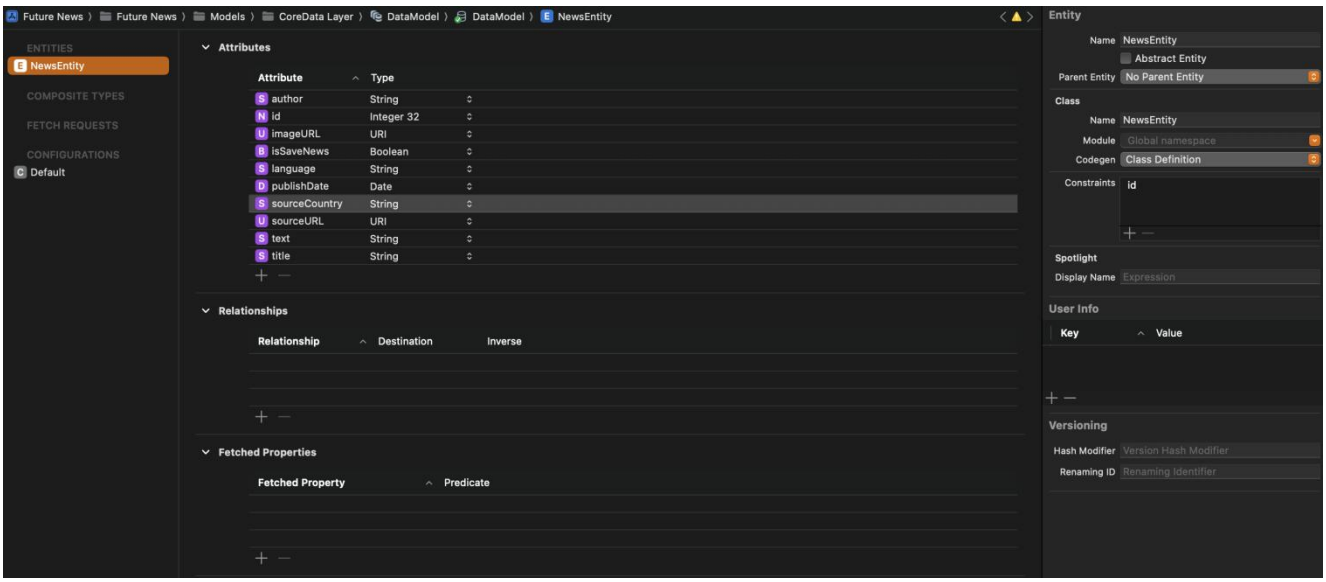


Рисунок 3.6 - Приклад налаштованої бази даних CoreData у проекті

Одним із основних атрибутів CoreData є здатність використовувати об’єктно-орієнтований підхід до роботи з даними. Розробникам не потрібно мати справу зі складними операторами SQL або безпосередньо отримувати доступ до бази даних. Натомість вони можуть створювати моделі даних, які складаються з класів або структур, і CoreData автоматично синхронізує ці об’єкти з базою даних.

Крім того, CoreData надає можливість використовувати різні типи зберігання даних, наприклад SQLite, XML або в пам’яті, залежно від вимог програми. Він також має внутрішню підтримку даних керування версіями, що дає змогу змінювати базу даних без втрати інформації чи підтримки сумісності з попередніми версіями.

Окрім зберігання даних, CoreData також містить різноманітні складні запити, які можна виконувати до даних у різних потоках, що зробить його потужним інструментом для роботи з даними, які зберігаються в програмах для платформи Apple.

Тепер обговоримо, як Apple адаптувала досить старий інструмент CoreData, який в основному написаний на Objective-C, до співпраці з новим фреймоворком SwiftUI. Заповітна корпорація із Купертіно виконала важливе завдання на всі сто. Обговоримо тему більш детально.

CoreData можна поєднувати з SwiftUI, щоб полегшити обробку даних, які постійно зберігаються в програмах, без написання своїх реактивних доповнень. Основна концепція полягає в тому, що SwiftUI полегшує відображення даних, пов’язаних із CoreData, в інтерфейсі користувача, подібно до того, як CoreData

може ретроспективно надавати всю відповідну інформацію View.

Одним із основних методів для використання CoreData реактивним способом у SwiftUI, є використання @FetchRequest. Цей інструмент полегшує автоматичне реактивне вилучення даних із CoreData та оновлення інтерфейсу, коли дані змінюються. Наприклад, можна створити представлення, яке визначає @FetchRequest як засіб отримання списку об’єктів CoreData та відображення їх у інтерфейсі. Крім того, за допомогою @FetchRequest можна описати атрибути, які використовуються для визначення того, як дані повинні бути організовані або агреговані при доступі з бази даних, а також критерії, які використовуються для вибору об’єктів із бази даних, що дозволить виконати основну фільтрацію даних за допомогою конкретних умов для цілей, як показано на прикладі Рисунок 3.1 (8).

Інші методи, такі як NSManagedObject і NSFetchRequest, можна використовувати для прямого доступу до даних CoreData у програмі на SwiftUI. Є можливість створювати та виконувати запити до бази даних CoreData, отримувати NSManagedObjects і використовувати їх у своєму SwiftUI-додатку.

Однією з головних переваг використання CoreData з SwiftUI є можливість використовувати декларативний метод створення інтерфейсу користувача разом із потужним менеджером даних, який надає CoreData. Це надає розробникам швидке створення інтерфейсів, які автоматично оновлюватимуться під час зміни даних, і швидку роботу зі збереженими даними в програмах SwiftUI.

Ось приклад роботи з використанням CoreData та SwiftUI на Рисунку 3.7 .

Рисунок 3.7 - Приклад використання @FetchRequest

Тепер детально обговоримо весь код на прикладі рисунка 3.7 У рядках 12 і 13 оголошується властивість з обгорткою @FetchRequest, яка є сама по собі інформацією з бази даних. Далі залежно від умови, створюємо або таблицю, або колекцію, але в обох випадках передаються дані типу FetchedRequest<NewsEntity> до інших видів в ієрархії. У рядку 12 викликаємо статичний метод fetchSaveNews. Цей метод використовується для впорядкування даних із бази даних. Він необхідний, тому що база даних буде повертати нам всі

дані, які тільки в нього є, але цей метод, який можна розглянути на Рисунку 3.1 (9), фільтрує та сортує так як нам потрібно. Розглянемо код методу.

```
19      static func fetchSaveNews() -> NSFetchRequest<NewsEntity> {
20          let request = NSFetchRequest<NewsEntity>(entityName: "NewsEntity")
21              request.sortDescriptors = [ NSSortDescriptor(keyPath: \NewsEntity.publishDate, ascending: false) ]
22              request.predicate = NSPredicate(format: "isSaveNews == true")
23
24          return request
25      }
```

Рисунок 3.8 - Приклад реалізації методу для налаштування NSFetchRequest

Обговорюючи процедуру з рисунка 3.8, ми можемо побачити дескриптори та предикати, які були згадані раніше, і в цьому випадку вони необхідні для того, щоб контекст нашої бази даних фільтрував та повертав лише збережені новини (запис: “isSaveNews = = true”) і впорядкував їх за датою публікації в sortDescriptors.

Це досить простий спосіб отримання завжди актуальної інформації з бази даних.

Обговоримо реактивну частину мого додатку. Незважаючи на те, що ми вже обмежено обговорювали фреймворк Combine, ми повинні повернутися до фреймворку та обговорити його більш детально. Combine [7] — це фреймворк від Apple, який надає потужні інструменти для реактивного програмування в програмах для платформ iOS, macOS, watchOS і tvOS. Основна концепція реактивного програмування полягає у створенні коду, який є асинхронним і реагує на зміни даних.

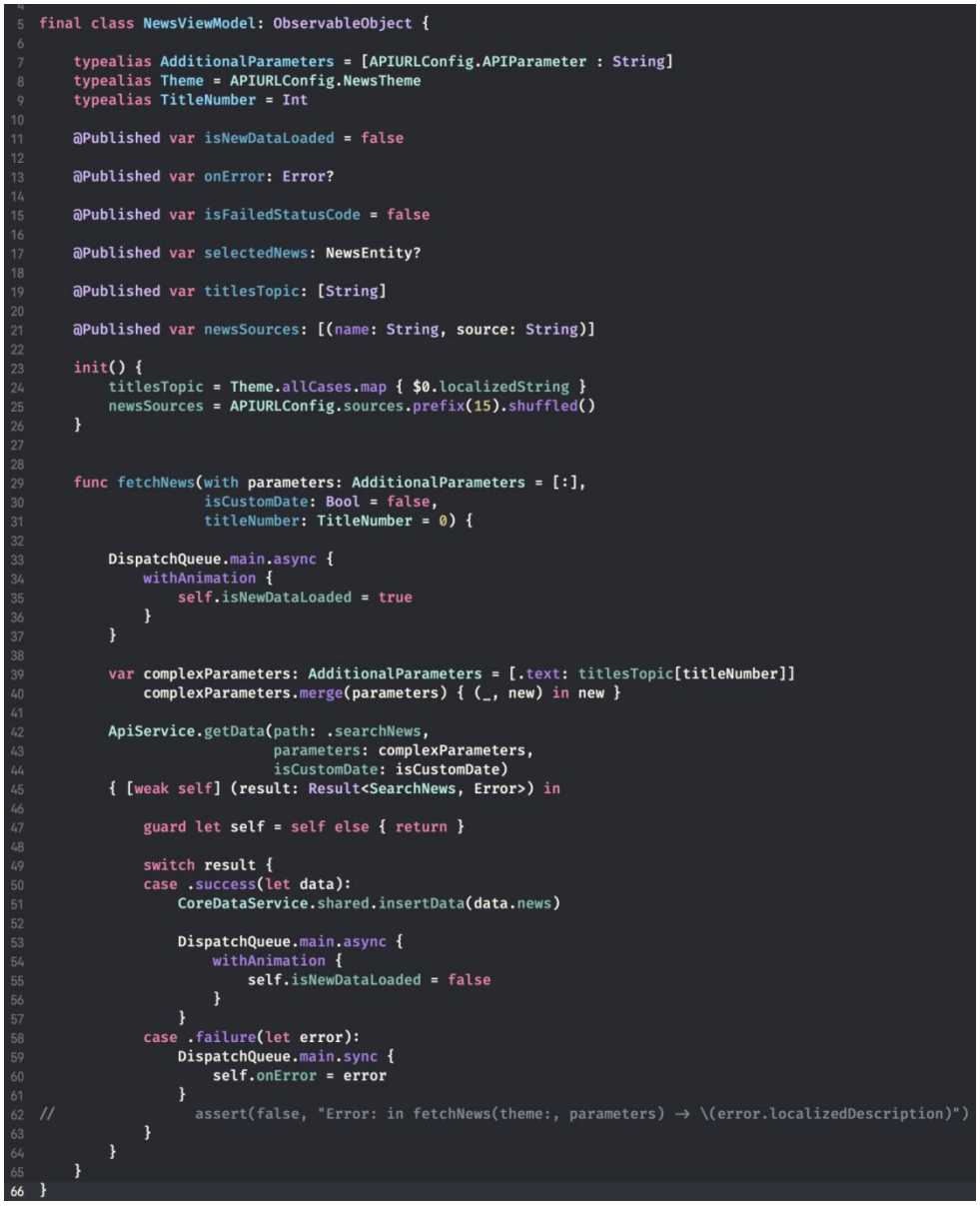
Combine надає кілька класів операторів і подій, які дозволяють об’єднувати, перетворювати та оцінювати потоки даних і подій. Це походить від концепції надсилання та отримання подій, ці події представлені видавцями (Publishers) і підписниками (Subscribers). Підписники підписуються на видавців, щоб

отримувати події, які вони публікують.

Однією з головних переваг Combine є можливість створювати складні потоки даних і реагувати на зміни в реальному часі. Крім того, він допомагає в управлінні асинхронним кодом, що робить його більш зрозумілим і простим у розробці та підтримці.

Тепер ми маємо елементарне розуміння того, як Combine пов’язаний з усіма іншими бібліотеками та фреймворками Apple. Combine — це той самий золотий ключ, який Apple використовує для об’єднання кількох технологій, зокрема: SwiftUI, CoreData та Networking. Якби не Combine, написання ViewModel для програми зайняло б набагато більше часу, оскільки Apple створила протокол ObservableObject для класів, які створюватимуть видавців (@Published), які створюватимуть View Models.

Давайте розглянемо реалізацію ViewModel у програмі на прикладі коду на Рисунку 3.9.



```swift
final class NewsViewModel: ObservableObject {

    typealias AdditionalParameters = [APIURLConfig.APIParameter : String]
    typealias Theme = APIURLConfig.NewsTheme
    typealias TitleNumber = Int

    @Published var isNewDataLoaded = false

    @Published var onError: Error?

    @Published var isFailedStatusCode = false

    @Published var selectedNews: NewsEntity?

    @Published var titlesTopic: [String]

    @Published var newsSources: [(name: String, source: String)]

    init() {
        titlesTopic = Theme.allCases.map { $0.localizedString }
        newsSources = APIURLConfig.sources.prefix(15).shuffled()
    }


    func fetchNews(with parameters: AdditionalParameters = [:],
                   isCustomDate: Bool = false,
                   titleNumber: TitleNumber = 0) {

        DispatchQueue.main.async {
            withAnimation {
                self.isNewDataLoaded = true
            }
        }

        var complexParameters: AdditionalParameters = [.text: titlesTopic[titleNumber]]
            complexParameters.merge(parameters) { (_, new) in new }

        ApiService.getData(path: .searchNews,
                           parameters: complexParameters,
                           isCustomDate: isCustomDate)
        { [weak self] (result: Result<SearchNews, Error>) in

            guard let self = self else { return }

            switch result {
            case .success(let data):
                CoreDataService.shared.insertData(data.news)

                DispatchQueue.main.async {
                    withAnimation {
                        self.isNewDataLoaded = false
                    }
                }
            case .failure(let error):
                DispatchQueue.main.sync {
                    self.onError = error
                }
//                assert(false, "Error: in fetchNews(theme:, parameters) → \(error.localizedDescription)")
            }
        }
    }
}
```

Рисунок 3.9 - Приклад коду ViewModel

Як бачимо, існує кілька видавців, які пов’язані з обгорткою властивості @Published, які є реактивними та передають дані до View. Як відомо, функціональність Combine не обмежується на рівні цієї реалізації. Майже всі основні інструменти Swift мають додаткові компоненти для реактивного програмування, такі як колекції, URLSession, які будуть обговорюватися пізніше, та інші.

Тепер обговоримо основний функціонал програми, а саме: запити за

інформацією в Інтернет. Робота з мережею, або Networking, має вирішальне значення у функціональності розглянутої програми. Основний функціонал додатка передбачає надсилання запитів до сервера, отримання відповідей і зберігання цієї інформації.

Для використання мережі в iOS і macOS доступні різні методи, як-от використання URLSession та сторонні рішення, наприклад Alamofire або Moya. URLSession [8] — це готова мережева утиліта, яка надає функції надсилання HTTP-запитів, отримання відповідей і керування сеансами зв'язку.

Коли маєте справу з мережею, дуже важливо враховувати асинхронний характер операцій, оскільки підключення до сервера може зайняти час, а відповіді можуть бути затриманими. Це може негативно вплинути на взаємодію з користувачем, тому вкрай важливо мати точне керування даними та виправляти помилки під час зв'язку із сервером.

Під час використання мережі також важливо дотримуватися принципів безпеки, зокрема захисту даних від несанкціонованого доступу та шифрування конфіденційної інформації. Щоб уникнути збору конфіденційної інформації, усі запити на HTTP мають виконуватися за допомогою безпечного протоколу (наприклад, HTTPS).

Тепер обговоримо службу, яка надає дані, а саме News API. NewsAPI [9] – це сервіс, який надає доступ до великої кількості новин з різних джерел по всьому світу. News API дозволяє отримувати доступ до поточних новин із багатьох джерел, включаючи провідні публікації, блоги, інформаційні агентства та інший вміст, призначений для вашої цільової аудиторії.

News API також має додаткові функції, які дозволяють аналізувати та обробляти дані новин. Для використання в програмі я використовую безкоштовну пробну версію API, тому під час роботи з цією конкретною версією API є деякі обмеження. Крім того, важливо розуміти, що цей API надає дані як JSON. В свою чергу [10] JSON (або JavaScript Object Notation) — це формат тексту, який базується на синтаксисі JavaScript і має просту структуру, яка складається з пар ключ-значення, де ключі — рядки, а значення можуть бути будь-якими об’єктами, зокрема числами, логічними значеннями, масивами або інші структурами JSON. Зразок вигляду пакету JSON приведено на Рисунку 3.10.

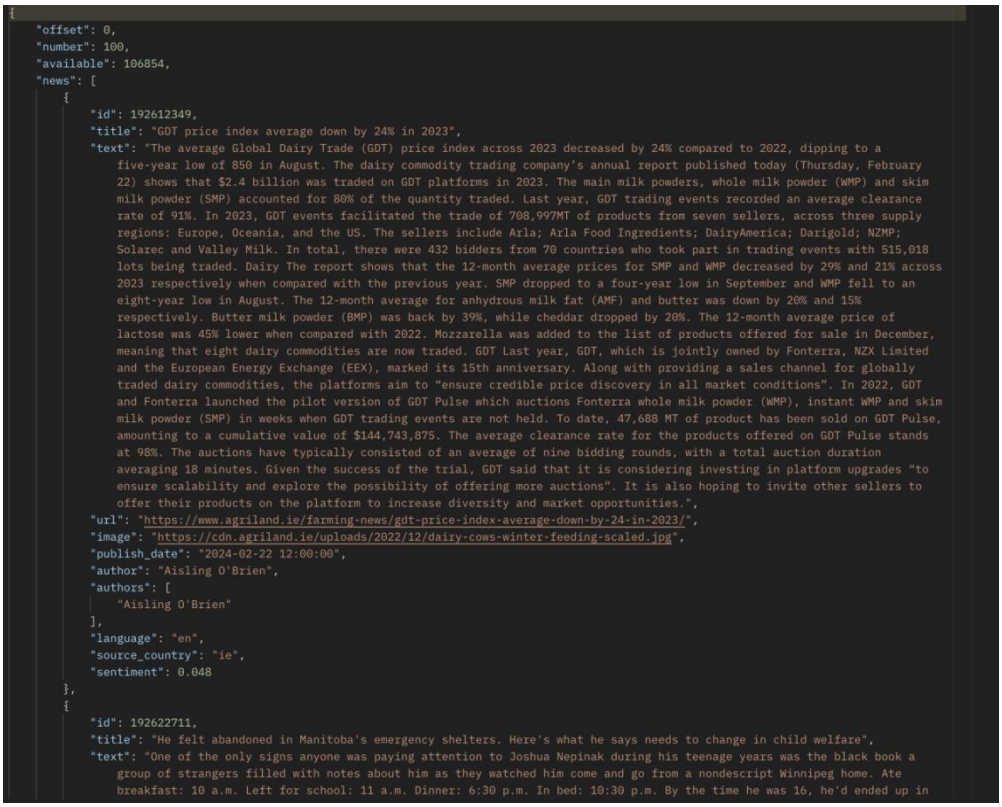


Рисунок 3.10 - Приклад JSON

Щоб отримати необхідну інформацію від сервера, потрібно зв’язатися з ним. Для цього я використовую URL-запит де вказую URL-адресу. URL-адреса [11] – це адреса ресурсу, яка вказує на розташування ресурсу в Інтернеті. Його

призначення – знаходити веб-сайти, файли, зображення, відео та інші інтернет-ресурси.

URL-адреса складається з кількох компонентів, включаючи протокол, який використовується для передачі даних. Далі можете побачити URL-адрес, який створюється одним з моїх класів-одинаків(Singleton) на Рисунку 3.11

https://api.worldnewsapi.com/search-news?api-key=8c25c03b336b45068fe0a37960b99b43&number=100&language=en&earliest-publish-date=2024-02-22%252018:25:20&text=All%2520news

У цьому прикладі зазначено:

1. Протокол передачі даних - HTTPS.
2. Ім'я хоста - api.worldnewsapi.com.
3. Шлях до ресурсу на сервері - search-news.
4. Мій API-ключ - 8c25c03b336b45068fe0a37960b99b43.
5. Параметри запиту:

- number=100 - кількість новин, яку повинен повернути сервер.
- language=en - мова новин.
- earliest-publish-date=2024-02-22 - діапазон дат публцікації.
- text=All%2520news - пошук по тексту “All news”. Також тут можна побачити якісь символи, це називається URL-кодування, воно потрібне для того щоб заміняти різні символи, по типу: пробіл, “!”, “?”, “,”, “.” та багато інших на закодовані форматом ASCII записи.

Повернемось до URLSession. Знаючи що таке JSON та URL, можемо розглянути приклад реалізації мережевого слою в моєму додатку на Рисунку 3.1.

Рисунок 3.11 - Приклад реалізації мережевого слою

Як очевидно, класом, відповідальним за мережевий рівень, є Singleton. Цей клас має лише один метод, який генерує URL-адресу, цей метод створює тіло запиту та викликає метод завантаження даних за допомогою методу dataTask(). Нарешті, в тілі закриття ми використовуємо захисний механізм і додаткову

прив’язку для оцінки помилок і статус-коду двохсотої серії. Після завершення цього процесу ми викликаємо метод, який декодує JSON у зрозумілий для Swift тип. Для того, щоб використовувати цю інформацію з JSON, її потрібно перетворити у формат, знайомий Swift, до декодування це просто серія байтів типу Data.

Для декодування потрібно виповнити певні послідовні дії, а саме:

1) Створити тип з властивостями з однаковими іменами як у key в тілі JSON;
2) Підписати тип на протокол Codable та реалізувати його;
3) Визвати статичний метод decode(_: T.Type , from: Data) типу JSONDecoder та передати в параметри потрібні дані.

За підсумком ми повинні отримати тип подібний прикладу зображеному на Рисунку 3.12.

```swift

import Foundation

struct ExtractNews: Codable {
    let title: String
    let text: String
    let url: String
    let image: String
    let publishDate: String
    let author: String
    let language: String
    let sentiment: Double
    let entities: [Entity]

    enum CodingKeys: String, CodingKey {
        case title = "title"
        case text = "text"
        case url = "url"
        case image = "image"
        case publishDate = "publish_date"
        case author = "author"
        case language = "language"
        case sentiment = "sentiment"
        case entities = "entities"
    }
}

// MARK: - Entity
struct Entity: Codable {
    let type: String
    let name: String
    let latitude: Double
    let longitude: Double
    let locationType: String

    enum CodingKeys: String, CodingKey {
        case type = "type"
        case name = "name"
        case latitude = "latitude"
        case longitude = "longitude"
        case locationType = "location_type"
    }
}

```

Рисунок 3.12 - Приклад реалізації моделі даних

Тепер подивимось, як виконується запит до серверу. Вище ми вже розглядали клас NewsViewModel, який являється ViewModel в моєму проекті. В цьому класі є метод fetchNews() в якому є параметри для того, щоб налаштувати запит на сервер. Тепер подивимось як виклакається цей метод на приклалі коду головного View на Рисунку 3.13.

```
1
2  import SwiftUI
3
4  @main
5  struct Future_NewsApp: App {
6      @ObservedObject private var newsVM = NewsViewModel()
7
8      var body: some Scene {
9          WindowGroup {
10             MainView()
11                 .onAppear {
12                     newsVM.fetchNews()
13                 }
14                 .environmentObject(newsVM)
15                 .environment(\.managedObjectContext,
16                              CoreDataService.shared.context)
17         }
18     }
19 }
20
```

Рисунок 3.13 - Приклад виклику методу fetchNews()

На рядках 10-12 на Рисунку 3.13, ми можемо побачити модифікатор із фреймворку SwiftUI. Цей модифікатор потрібен для того щоб запускати якісь процеси після того як View з’явиться на екрані і так як це головна View нашого проекту, то вона з’являється лише один раз при запуску. Тому кожного разу як запускається проект лента новин буде оновлюватись, визиваючи метод fetchNews(). Але це не єдиний виклик методу fetchNews() у проекті, він є в деяких місцях, як наприклад, на Рисунку 3.14. Виклик fetchNews() знаходиться в структурі AllNewsView та визивається цей метод у випадку коли ми оновлюємо ленту новин.

```
76             .refreshable {
77                 newsVM.fetchNews(titleNumber: selectedIndex)
78             }
79         }
80 }
```

Рисунок 3.14 - Приклад виклику методу fetchNews() в інших частинах проекту

Тепер обговоримо багатопотоковість більш детально та те, як я використав її у своєму проекті, але спочатку трохи теорії. Багатопотоковість (або паралельність) — це концепція програмування, яка дозволяє виконувати кілька кодів (потоків) одночасно в одній програмі. Кожен потік може виконувати власні функції, не залежачи від інших потоків.

Багатопотоковість є особливо корисною у випадках, коли програма повинна обробляти кілька дій одночасно або взаємодіяти з різними джерелами даних. Наприклад, веб-сервер може використовувати багатопотоковість для обслуговування запитів від кількох клієнтів одночасно. Крім того, у графічних інтерфейсах можна використовувати багатопотоковість для одночасного оновлення інтерфейсу користувача та обробки введених користувачем даних.

У Swift є кілька методів роботи з багатопотоковістю, які дозволяють створювати потоки та керувати ними в програмах. Деякі з найбільш значущих механізмів є:

1. Grand Central Dispatch (GCD): це найпопулярніший механізм для потоків у Swift. GCD простий і ефективний у створенні потоків, які виконуються в асинхронному середовищі. Він використовує черги, щоб спрямовувати завдання до призначених місць і виконувати їх у відповідних потоках.

2. Async/await оператори: це новий метод, реалізований у Swift 5.5., який дає можливість виконувати виклики функцій, які є асинхронними, у синхронний спосіб, це прояснить і спростить код, зробивши його більш розумілим.

3. Клас NSOperation походить від фреймворку Foundation. Він полегшує

створення та керування завданнями вищого рівня, які можна планувати відповідно до їх пріоритету та залежностей.

Багато офіційних бібліотек і фреймворків від Apple тісно пов’язані одна з одною, і багатопотоковість у цьому контексті не є чимось незвичайним. Подібно до async/await, Apple включила методи з атрибутом async у раніше обговорений клас URLSession, і ці методи представлені методами:

data(for:)

download(from:)

upload(_: to: completionHandler:)

Крім того, потрібно розуміти, чи може бібліотека виконуватись асинхронно у фоновому режимі чи ні. Наприклад, CoreData не може бути виконано поза основним потоком. Основний потік (також званий UI-потоком) є потоком, який виконує програму. Це основний потік, у якому міститься код для інтерфейсу користувача (UI), і розглядаються такі події, як натискання кнопок або жести. Основна мета основного потоку — забезпечити постійне спілкування з користувачем. Він відповідає за відображення інтерфейсу користувача, обробку взаємодії користувача та оновлення візуальних компонентів. Оскільки будь-яка тривалість або блокування в основному потоці може призвести до заминок та деградації роботи додатку, рекомендується виконувати важкі обчислення та операції введення/виведення в інших потоках.

Також, потрібно відмітити, що у фреймворку SwiftUI є елемент інтерфейсу, який потрібен для завантаження зображень з мережі, назва йому AsyncImage. Цей

елемент отримує URL-адресу зображення та асинхронно завантажує його. Як приклад роботи з таким елементом інтерфейсу, слугує Рисунок 3.15.

```
fileprivate struct VerticalViewRow: View {
    @State var item: NewsEntity

    var body: some View {
        HStack {
            AsyncImage(url: item.imageURL_) { image in
                image
                    .resizable()
                    .scaledToFill()
                    .shadow(radius: 10)
                    .frame(width: 90, height: 90)
                    .clipped()
            } placeholder: {
                Image("news-placeholder")
                    .resizable()
                    .scaledToFill()
                    .shadow(radius: 10)
                    .frame(width: 90, height: 90)
                    .clipped()
            }
            .clipShape(RoundedRectangle(cornerRadius: 10))

            VStack(alignment: .leading) {
                Text(item.title_)
                    .lineLimit(1)
                    .font(.system(size: 23))
                    .bold()

                Text(item.text_)
                    .fontWeight(.medium)
                    .font(.system(size: 17))
                    .lineLimit(2)

                Text(item.publishDate_.publishingDifference())
                    .font(.system(size: 15))
            }
        }
        .padding()
        .background {
            RoundedRectangle(cornerRadius: 7)
                .fill(.colorSet3)
                .shadow(radius: 3)
        }
    }
}
```

Рисунок 3.15 - Приклад використання елементу AsyncImage

AsyncImage отримує URL-адресу зображення та завантажує його асинхронно. Якщо завантаження не вдається, буде показано альтернативне зображення, яке вводиться вручну.

3.2. Приклад роботи додатку Future News

Тепер розглянемо конкретний приклад використання програми. Користувач починає процес на своєму пристрої, як зображено на Рисунку 3.16. Коли юзер

вперше запускає додаток, попередньо завантажуються новини, і найактуальніші із них відображаються. Користувач може вибрати тему, яка його цікавить, і потім переглянути список статей на цю тему.

Рисунок 3.16 - Приклад першого запуску додатку

Після завантаження, інтерфейс виглядає подібно до Рисунку 3.17.

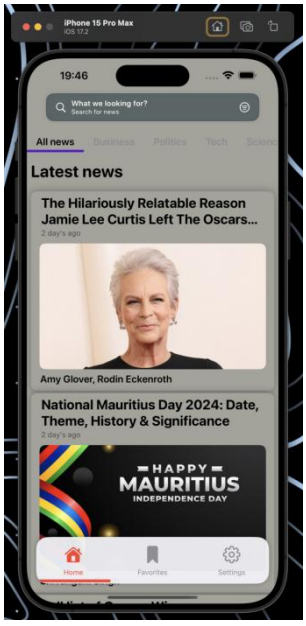


Рисунок 3.17 - Приклад вже завантаженої ленти

Користувач може натиснути на цікаву йому новину та прочитати статю. Інтерфейс, у свою чергу, виглядає як показано на Рисунку 3.18

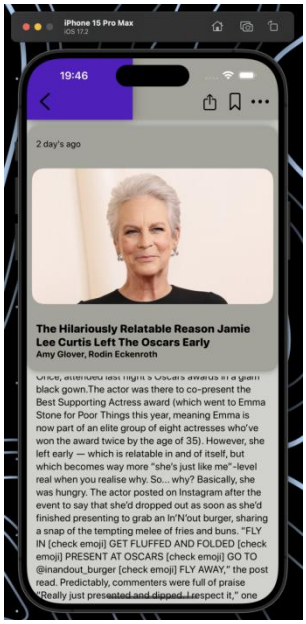


Рисунок 3.18 - Приклад відкритої новини

Потім користувач може натиснути на кнопку “Зберегти”, щоб дана новина записалась у базу данних із значенням true для атрибуту isSaveNews.

Після цього, на головному екрані з’явиться новий елемент інтерфейсу, де будуть показані останні збережені новини. Приклад на Рисунку 3.19.

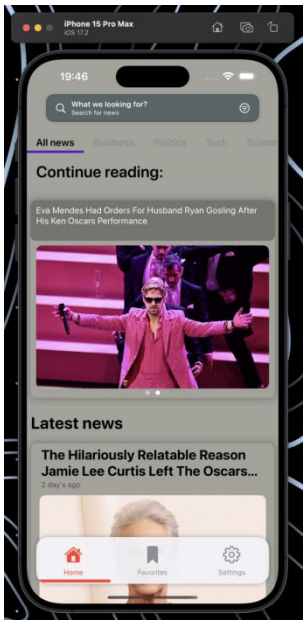


Рисунок 3.19 - Приклад вигляду нового елементу інтерфейсу на головному екрані

Перемкнувшись на екран “Favourites”, користувач може переглянути свої збережені новини. Також натиснувши кнопку у правому верхньому кутку змінити види зображення. Приклад функціоналу на Рисунку 3.20.

Рисунок 3.20 - Приклад вигляду збережених новин

Якщо користувач вирішує здійснити пошук конкретної новини, він натискає на View зверху головного екрану, і вводить певні характеристики для пошуку, як сам текст пошуку, дата публікацій в діапазоні “From-To” та джерело, як показано на Рисунку 3.21.

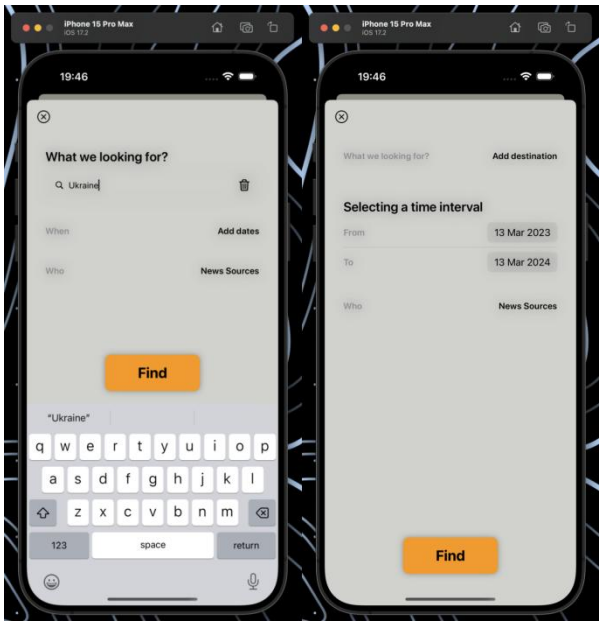


Рисунок 3.21 - Меню пошуку

Натискаєте на кнопку пошуку. Пошук починається. Зображення процесу завантаження на Рисунку 3.22

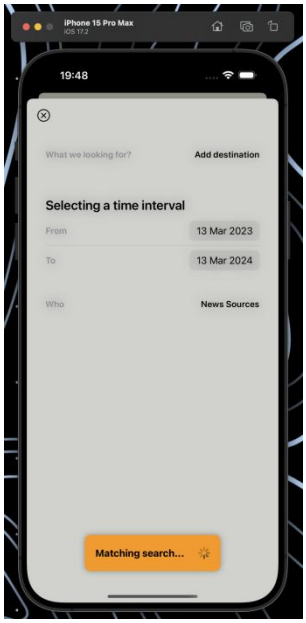


Рисунок 3.22 - Приклад пошуку новин

Після вдалого завантаження, ористувач переглядає результати пошуку та обирає ту новину, яка його цікавить. Він може прочитати повний текст новини, як було показано раніше, та переглянути зображення, які супроводжують цю статтю.

В пошуку теж є 2 види відображення новин. Приклад як виглядає цей View на Рисунку 3.23.

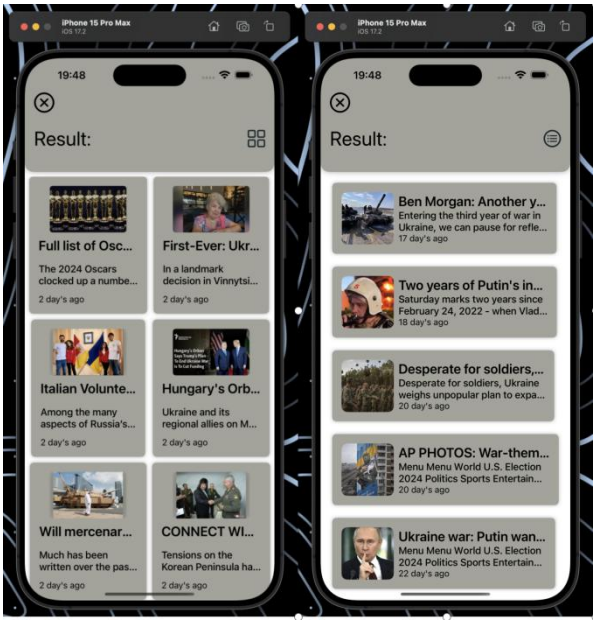


Рисунок 3.23 - Приклад пошуку новин

Таким чином, додаток "Future News" надає користувачам зручні інструменти для пошуку, перегляду та збереження новин, що робить його важливим додатком для отримання актуальної інформації.

3.3. Запуск додатку на macOS та iPadOS

Так як ми писали проект за домопогою фреймворку SwifUI, ми можемо запустити цей проект на інших платформах. Для того щоб запустити його, потрібно в проекті вибрати додаткові платформи, як показано на Рисунку 3.24.

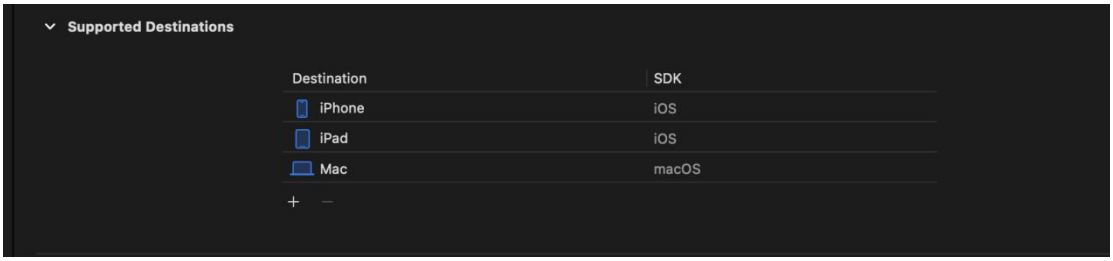


Рисунок 3.24 - Приклад налаштування проекту для запуску на інших платформах

Також потрібно відмітити, що за допомогою директив компілятора або за допомогою інтсрументів по типу класа UIUserInterfaceIdiom можна налаштувати певні шматки коду під певну операційну систему. Після цього ми матимемо більш закінчений дизайн або можливість використання функціональності, яка притаманна саме тому девайсу на якому запущено додаток, без проблем та помилок у роботі самого додатку. Як приклад в додатку, реалізовано використання идиом(приклад на рис 3.25), для того щоб новини були розміщені в ряд по 2 елементи, а не 1, як це реалізовани на версії для смартфону.

```
38              LazyVStack {
39                  if UIDevice.current.userInterfaceIdiom == .pad ||
                        UIDevice.current.userInterfaceIdiom == .mac {
40                      LazyVGrid(columns: [ GridItem(.flexible()),
41                                           GridItem(.flexible()) ],
42                                spacing: 15) {
43
44                          ForEach(items, id: \.self) { element in
45                              NewsCell(selectedNews: element, idiom: [.pad, .mac])
46                                  .onTapGesture {
47                                      newsVM.selectedNews = element
48                                      isActive.toggle()
49                                  }
50                                  .id(element.id_)
51                          }
52                      }
53                  } else {
54                      ForEach(items, id: \.self) { element in
55                          NewsCell(selectedNews: element, idiom: [.phone])
56                              .padding(.bottom, 6)
57                              .onTapGesture {
58                                  newsVM.selectedNews = element
59                                  isActive.toggle()
60                              }
61                              .id(element.id_)
62                      }
63                  }
64              }
65              .padding(.horizontal, 7)
66          }
```

Рисунок 3.25 - Приклад використання UIUserInterfaceIdiom у додатку

Тепер розглянемо як працює додаток на macOS, на прикладі Рисунку 3.26.

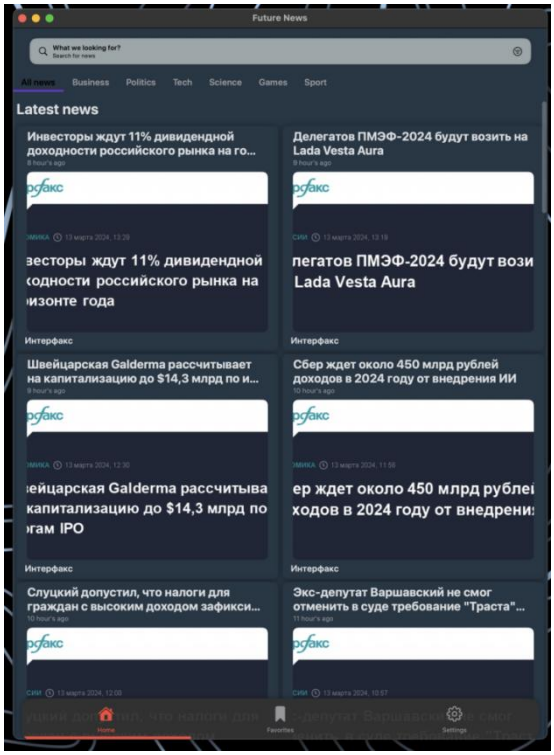


Рисунок 3.26 - Приклад роботи на операційній системі macOS

Так, є певна різниця у вигляді, але без ніяких змін у кодовій базі працює весь функціонал додатку.

І ось приклад на Рисунку 3.27, як працює додаток на iPadOS.

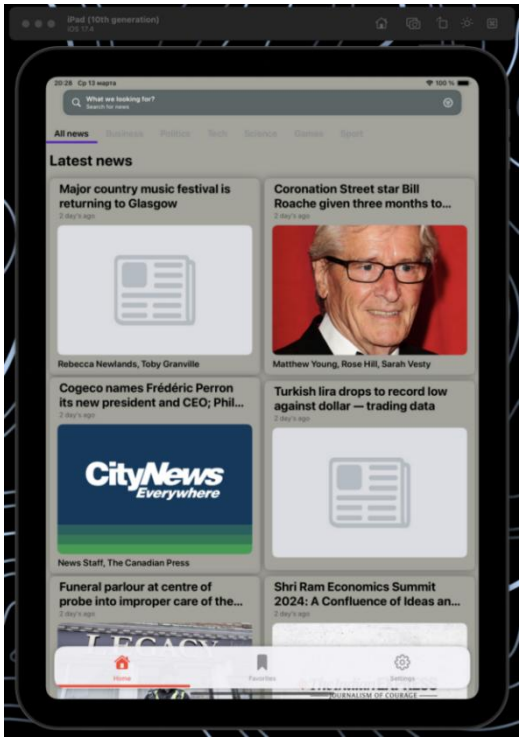


Рисунок 3.27 - Приклад роботи на операційній системі iPadOS

ВИСНОВКИ

В роботі отримані наступні результати:

1. При ознайомлення з сучасними технологіями для розробки додатків для iOS і macOS було розглянуто мову програмування Swift, інтегроване середовище розробки Xcode, інструменти Cococa та CocoaTouch, Interface Builder та інших технологій для розробки інтерфейсу користувача та сучасні тренди Progressive Web Apps(PWA), Privacy and Security у розробці додатків. Під час аналізу, ми порівнювали стару мову програмування Objective-C та нову Swift, і чому Apple вирішила перейти на Swift. Було досліджено, як розробники впливають на підтримку нової мови програмування та майбутнє цієї мови. Було досліджено, яку небезпеку може нести за собою використання сторонніх бібліотек. Опираючись на власні дослідження та на документацію від компанії Apple, була проаналізована основна функціональність Xcode, її інтерфейс, інструменти та можливість використання сторонніх інструментів, на прикладі GitHub Copilot. Аналізуючи інструменти Cococa та CocoaTouch, були виявлені основні особливості, функціонал та специфічність роботи на операційних системах macOS та iOS. Було досліджено мінуси та плюси використання PWA, та які обмеження ця технологія за собою несе. Досліджено вплив з боку компанії Apple на створення додатків та їх безпечності і приватності опрацьованих даних для юзера.

2. Була досліджена користь при використанні архітектурних рішень, аналізована особливость адаптації додатків до різних версій iOS та macOS та інтеграція сучасних технологій у реальних проектах. Було дослідженно вплив використання архітектури на підтримку кодової бази та розвиток програм.

Дослідження проводилось на основі архітектур MVC(Model-View-Controller) та MVVM(Model-ViewModel-View). Досліджено можливості забезпечення стабільності та безпеки управління версіями коду та сумісність з різними версіями операційних систем. Розглянуто інструкції для компілятора, атрибути “#available” та багато іншого.

3. Було створено додаток для операційних систем iOS і macOS з використанням концепцій, технологій та підходів розглянутих раніше. Було проведено опис програмної реалізації сучасних технологій та на основі досліджених тем, рекомендацій, методик та використаних технологій, таких як: CoreData, Grand Central Dispatch, SwiftUI, etc. Приведено приклади роботи додатку на різних пристроях, та можливість легко змінювати інтерфейс додатку на різних операційних системах за допомогою UIUserInterfaceIdiom.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Онлайн-документація компанії Apple [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://developer.apple.com/xcode]

(дата звернення 20.03.2024) - Xcode;

2) Онлайн-енциклопедія Вікіпедія [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://ru.wikipedia.org/wiki/Model-View-Controller]

(дата звернення 18.03.2024) - Model-View-Controller;

3) Онлайн-документація компанії Apple [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://developer.apple.com]

(дата звернення 20.03.2024) - Human Interface Guildelines;

4) Онлайн-енциклопедія Вікіпедія [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://ru.wikipedia.org/wiki/Model-View-ViewModel]

(дата звернення 18.03.2024) - Model-View-ViewModel;

5) Онлайн-документація компанії Apple [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://developer.apple.com/swiftui]

(дата звернення 20.03.2024) - SwiftUI;

6) Онлайн-документація компанії Apple [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://developer.apple.com/coredata]

(дата звернення 20.03.2024) - CoreData;

7) Онлайн-документація компанії Apple [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://developer.apple.com/combine]

(дата звернення 20.03.2024) - Combine;

8) Онлайн-документація компанії Apple [Інтернет-ресурс]: [Веб-сайт]. – Режим

9) доступу: [https://developer.apple.com/urlsession]

(дата звернення 20.03.2024) - URLSession;

10) Онлайн-сервіс новин [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://worldnewsapi.com/console/#Profile]

(дата звернення 20.03.2024) - NewsAPI;

11) Онлайн-енциклопедія Вікіпедія [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://ru.wikipedia.org/wiki/JSON]

(дата звернення 18.03.2024) - JSON;

12) Онлайн-енциклопедія Вікіпедія [Інтернет-ресурс]: [Веб-сайт]. – Режим доступу: [https://ru.wikipedia.org/wiki/URL]

(дата звернення 18.03.2024) - URL;