ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка мобільного застосунку для моніторингу розкладу спортивних подій з персоналізованими сповіщеннями»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

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

_____ Олексій Жилєнков
(підпис)

Виконав: здобувач вищої освіти групи ПД-43
Олексій ЖИЛЄНКОВ

Керівник: Вячеслав САВІЦЬКИЙ
викладач кафедри

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Жилєнкову Олексію Сергійовичу

1. Тема кваліфікаційної роботи: «Розробка мобільного застосунку для моніторингу розкладу спортивних подій з персоналізованими сповіщеннями»
керівник кваліфікаційної роботи викладач кафедри Вячеслав САВІЦЬКИЙ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості, методи вилучення та обробки даних із REST-API, методи формування персоналізованих сповіщень, технічна документація фреймворків та бібліотек, додаткові ресурси.

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

1. Огляд та аналіз існуючих мобільних додатків і технологій для моніторингу розкладу спортивних подій та персоналізованих сповіщень.
2. Проєктування архітектури й інтерфейсу мобільного застосунку з використанням SwiftUI та патерну MVVM.
3. Програмна реалізація ключових модулів: інтеграція з API-Football,

відображення розкладу матчів, формування та планування локальних сповіщень.

4. Тестування застосунку.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до програмного забезпечення.
   3. Програмні засоби реалізації.
   4. Діаграма варіантів використання.
   5. Блок схема процесу роботи з матчами.
   6. Діаграма класів.
   7. Екранні форми.
   8. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз існуючих мобільних додатків і технологій для моніторингу розкладу спортивних подій та персоналізованих сповіщень.	14.03-20.03.2025	
4	Проєктування архітектури й інтерфейсу мобільного застосунку з використанням SwiftUI та патерну MVVM	21.03-31.03.2025	
5	Програмна реалізація ключових модулів: інтеграція з API-Football, відображення розкладу матчів, формування та планування локальних сповіщень.	01.04-18.04.2025	
6	Тестування застосунку	19.04-28.04.2025	
7	Оформлення роботи	29.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-18.05.2025	
9	Попередній захист роботи	19.05-30.05.2025	

Здобувач вищої освіти _____ Олексій ЖИЛЄНКОВ
(підпис)

Керівник кваліфікаційної роботи _____ Вячеслав САВІЦЬКИЙ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 61 стор., 1 табл., 10 рис., 20 джерел.

Мета роботи – покращення користувацького досвіду при моніторингу спортивних подій та керуванні нагадуваннями.

Об'єкт дослідження – процес інформування користувачів про актуальні спортивні події з урахуванням їх персональних вподобань.

Предмет дослідження – розробка та вдосконалення мобільного застосунку для відстеження футбольних матчів із підтримкою персоналізованих push-сповіщень та інтеграції з локальним збереженням користувацьких налаштувань.

Короткий зміст роботи: В роботі проаналізовано існуючі мобільні додатки та технології для моніторингу розкладу спортивних подій і персоналізованих сповіщень (зокрема Flashscore, SofaScore), а також підходи до інтеграції з REST-API. Розроблено архітектуру застосунку ScoreLine на базі SwiftUI і MVVM та програмно реалізовано ключові функціональні можливості, зокрема: завантаження та відображення розкладу матчів через API-Football [1], формування та перегляд детальної інформації про матчі (події, склади, статистика), налаштування та планування локальних нотифікацій за обраними умовами (за N хвилин до початку, при готовності складу), підтримку локалізації й тем оформлення. Проведено функціональне та модульне тестування застосунку. У роботі використано SwiftUI [3] для UI [2], Combine [7] для реактивного керування станом, UNUserNotificationCenter для нотифікацій [5]. Сферою застосування є забезпечення оперативного доступу до інформації про футбольні матчі та підвищення зручності планування спортивних активностей користувачів.

КЛЮЧОВІ СЛОВА: МОБІЛЬНИЙ ЗАСТОСУНОК, МОНІТОРИНГ СПОРТИВНИХ ПОДІЙ, ЛОКАЛЬНІ СПОВІЩЕННЯ, SWIFTUI, MVVM.

ЗМІСТ

ВСТУП **7**

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ МОНІТОРІНГУ РОЗКЛАДУ СПОРТИВНИХ ПОДІЙ З ПЕРСОНАЛІЗОВАНИМИ СПОВІЩЕННЯМИ **9**

1.1 Загальна характеристика застосунку. 9
1.2 Специфіка реалізації та технічні виклики. 11
1.3 Цільова аудиторія 11
1.4 Дослідження існуючих аналогів 12
1.4.1 FlashScore 12
1.4.2 SofaScore 14
1.4.3 LiveScore 16
1.4.4 OneFootball 18
1.4.5 FotMob 19
1.4 Визначення вимог до застосунку 21

2 ЗАСОБИ РЕАЛІЗАЦІЇ **22**

2.1 Середовище розробки 22
2.1.1 Xcode 22
2.1.1 iOS Simulator 23
2.2 Мова програмування Swift 23
2.3 Фреймворки 23
2.3.1 SwiftUI 24
2.3.2 Combine 24
2.4 Архітектура MVVM 24
2.5 Засоби мережевої взаємодії 24
2.5.1 URLSession 25
2.5.2 Codable 25
2.6 Операції з повідомленнями 25
2.7 Збереження локальних даних 25
2.8 Система управління базами даних 25
2.9 Система контролю версій 26
2.10 Інструменти для дизайну користувацького інтерфейсу 26

3 ПРОЄКТУВАННЯ **28**

3.1 Загальні принципи архітектури 28
3.2 Типи архітектури 28
3.2.1 Архітектура клієнт-сервер 28
3.2.2 MVVM 29
3.2.3 Реактивна архітектура 29
3.3 Архітектура клієнтського боку 30
3.4 Логіка взаємодії з API 31

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ **32**

4.1 Дизайн інтерфейсу 32
4.2 Реалізація клієнтської частини 38
4.3 Реалізація роботи API 43
4.4 Реалізація автоматичної зміни теми 45
4.5 Реалізація автоматичної зміни мови 46
4.6 Завантаження проєкту на GitHub 48
4.7 Тестування додатків 48
ВИСНОВКИ 50
ПЕРЕЛІК ПОСИЛАНЬ 52
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 54
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ 60

ВСТУП

Актуальність: У сучасному суспільстві мобільні пристрої стали основним каналом отримання найсвіжішої інформації, в тому числі про спортивні події. Щодня в різних лігах і турнірах по всьому світу проходять тисячі футбольних матчів, і вболівальники хочуть дізнаватися про початок гри, склади команд і важливі події безпосередньо з екранів своїх смартфонів. Однак існуючі рішення (веб-сайти, телевізійні трансляції, десктопні додатки) часто не надають кастомізованих сповіщень - користувачам доводиться постійно перевіряти розклад або покладатися на єдиний час нагадування від типового пуш-сервера.

Мобільний застосунок ScoreLine розроблений таким чином, щоб ви могли обрати власні критерії сповіщення (N хвилин до початку матчу, оголошення складів команд), що дозволяє оптимізувати інформаційний потік та підвищити залученість глядачів. Інтеграція з відкритими спортивними даними REST-API гарантує точне та своєчасне оновлення, а використання сучасних технологій SwiftUI, Combine та UNUserNotificationCenter забезпечує високу продуктивність, адаптивні інтерфейси та надійну роботу бекенду. Впровадження такого рішення допоможе покращити користувацький досвід для вболівальників та розширити можливості персоналізації спортивних цифрових продуктів.

Об'єктом дослідження є процес моніторингу розкладу футбольних матчів із формуванням персоналізованих локальних сповіщень у мобільному середовищі.

Предметом дослідження є розробка та вдосконалення мобільного застосунку для відстеження футбольних матчів із підтримкою персоналізованих push-сповіщень та інтеграції з локальним збереженням користувацьких налаштувань.

Метою роботи є покращення користувацького досвіду при моніторингу спортивних подій та керуванні нагадуваннями.

Методи дослідження: першим кроком було проаналізовано існуючі мобільні додатки та технології для моніторингу розкладу спортивних подій і побудови персоналізованих сповіщень (зокрема Flashscore, SofaScore), що дозволило

сформулювати початкові функціональні та нефункціональні вимоги до системи. Далі здійснено огляд стеку технологій: обрано SwiftUI і Combine для реалізації реактивного інтерфейсу, патерн MVVM для організації архітектури, URLSession і Codable для взаємодії з API-Football, UNUserNotificationCenter для локальних нотифікацій та UserDefaults для збереження налаштувань користувача. Дослідження способів реалізації застосунку проводилось безпосередньо під час його розробки: створено UML-діаграми, спроектовано прототип інтерфейсу в Xcode, запрограмовано ключові модулі завантаження розкладу матчів, відображення їх деталей і планування сповіщень. Паралельно виконано модульне та функціональне тестування (XCTest, XCUITest) для верифікації коректності роботи всіх компонентів, а також юзабіліті-тестування для оцінки зручності користувацького досвіду. На завершальному етапі проведено рефакторинг коду й оптимізацію продуктивності на основі отриманих результатів тестування.

Наукова новизна роботи полягає в поєднанні в одному мобільному застосунку ScoreLine механізмів динамічного планування локальних сповіщень за індивідуальними умовами, реактивного інтерфейсу на базі SwiftUI та Combine, уніфікованої MVVM-архітектури з чітким розділенням відповідальностей між мережею, бізнес-логікою та представленням даних, а також власного модуля діагностики і логування мережевих запитів і помилок, що забезпечує підвищену надійність і прозорість функціонування додатку.

Практична значущість результатів полягає в тому, що ScoreLine дозволяє вболівальникам оперативно отримувати актуальний розклад футбольних матчів із гнучким налаштуванням сповіщень, підвищуючи їхню залученість і лояльність, а також може бути адаптований під інші види спорту чи регіони завдяки модульній архітектурі та слугувати основою для комерційних і навчальних проєктів у сфері спортивних мобільних сервісів.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ МОНІТОРІНГУ РОЗКЛАДУ СПОРТИВНИХ ПОДІЙ З ПЕРСОНАЛІЗОВАНИМИ СПОВІЩЕННЯМИ

1.1 Загальна характеристика застосунку.

ScoreLine - це мобільний застосунок, призначений для швидкого та інтуїтивно зрозумілого моніторингу футбольних матчів у режимі реального часу з гнучкими локальними налаштуваннями сповіщень. На стартовому екрані користувачам одразу представляється список майбутніх футбольних матчів, включаючи дату, час, статус («Заплановано», «Пряма трансляція», «Завершено») та логотипи команд. За допомогою вбудованого календаря – зображено на рисунку 1.1, можна вибрати будь-яку дату і отримати список матчів цього дня одним дотиком, а у вкладці «Вибране» можна стежити лише за обраними командами або лігами. Веб-дані завантажуються через Football REST API, який використовує бібліотеку URLSession та механізм Codable для швидкого та безпечного розбору відповідей JSON. Завантаження статусів обробляється за допомогою комбінованої реактивної обробки, що забезпечує плавне оновлення інтерфейсу без зайвих змін у коді подання. Для сповіщень використовується UNUserNotificationCenter: застосунок генерує запити на основі вибраного передматчевого часу або появи складу команди, а iOS доставляє їх у встановлений час, навіть у фоновому режимі. Суттєвою перевагою є декларативний інтерфейс SwiftUI, який дозволяє легко адаптувати компоненти під різні розміри екранів, а архітектура MVVM[9] забезпечує чітке відокремлення бізнес-логіки від ведучого, спрощуючи розширення та супровід коду.

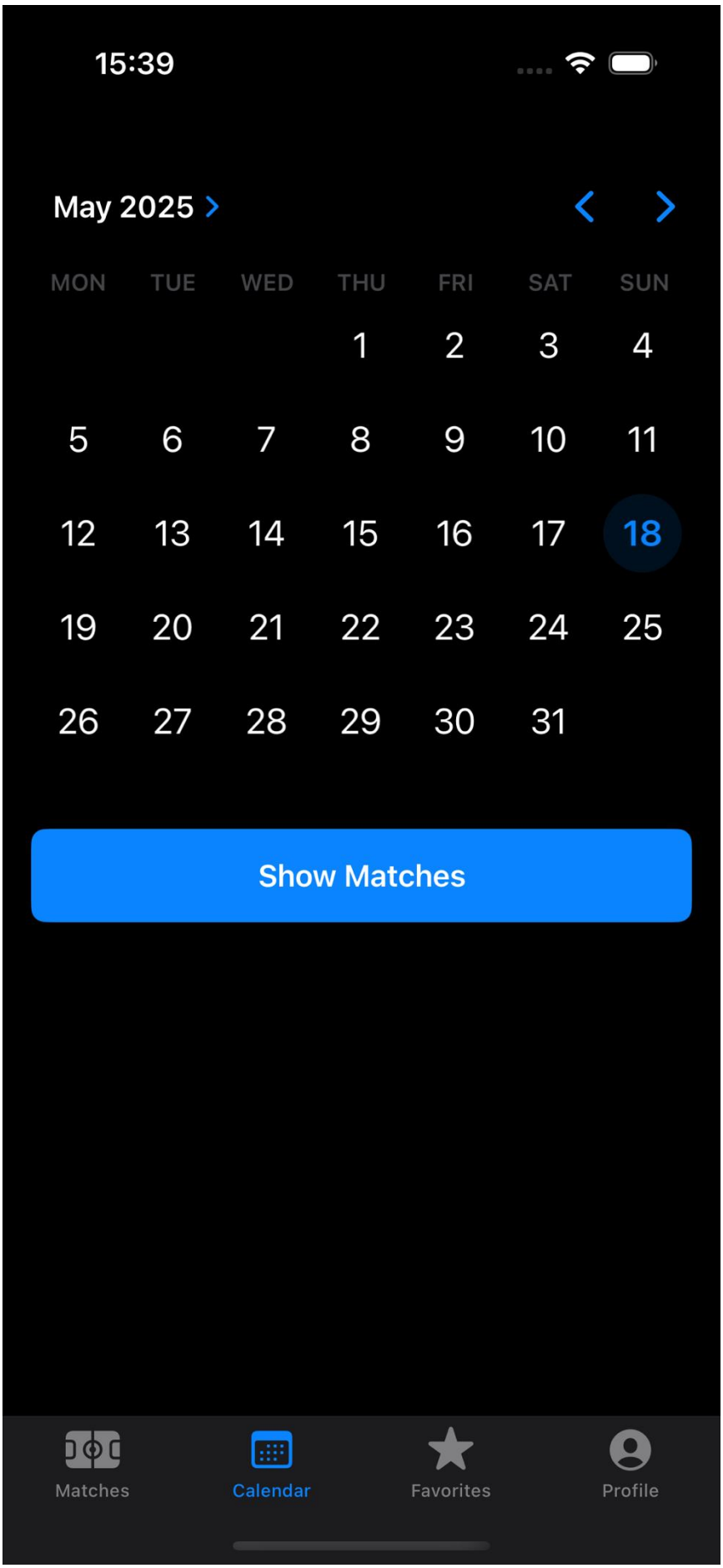


Рис. 1.1 Календар

1.2 Специфіка реалізації та технічні виклики.

Під час реалізації ScoreLine особливу увагу було приділено забезпеченню надійності фонових сповіщень та стійкості до нестабільного інтернет-з'єднання. Звернення до Football API може призводити до затримок або помилок авторизації, тому ми включили функцію кешування останніх успішних даних і механізм повторної спроби з експоненціальним відстеженням. Планування сповіщень в iOS вимагає коректного врахування часових поясів і переходу на літній час, а також обмежень на кількість запитів - ці нюанси були враховані за допомогою валідації конфігурації UNNotificationRequest та централізованої обробки. Ще одним викликом було забезпечити одночасне відображення великої кількості подій у DetailView без перевантаження інтерфейсу користувача - для цього ми використовували LazyVStack та обробляли асинхронні зображення команд через кеш-завантажувач. Не менш важливою була підготовка інтерфейсу до багатомовності: локалізація базується на Localizable.strings, що дозволяє динамічне перемикання мови в будь-який момент, що вимагає суворої перевірки констант і ключів перекладу.

1.3 Цільова аудиторія

Цільова аудиторія ScoreLine - активні футбольні вболівальники, які цінують свій час і хочуть отримувати точні сповіщення про матчі та команди, які їх цікавлять. Користувачі варіюються від аматорів, які стежать за своєю місцевою лігою, до професіоналів (тренерів, спортивних журналістів), які потребують швидкої реакції на оновлення складів і матчів при підготовці матеріалів або тренувальних занять. Користувачі, які ведуть власну статистику або беруть участь у фентезі-лігах, також можуть бути зацікавлені в додатку, оскільки гнучкість налаштувань сповіщень і швидкість оновлень полегшують відстеження результатів та їх аналіз.

1.4 Дослідження існуючих аналогів

У цьому розділі наведено детальний аналіз мобільних додатків FlashScore, SofaScore, LiveScore, OneFootball та FotMob. Кожен із них пропонує оперативне відображення розкладу футбольних матчів та базові сповіщення, однак обмежений у можливостях персоналізації, гнучкості налаштувань та адаптивності інтерфейсу у порівнянні з ScoreLine.

1.4.1 FlashScore

FlashScore - один з найпопулярніших мобільних додатків для моніторингу спортивних подій, зокрема футбольних матчів. Застосунок надає користувачам доступ до прямих трансляцій, турнірних таблиць, статистики, календарів матчів та базових push-повідомлень.

Серед переваг:

— висока стабільність;

— підтримка десятків видів спорту;

— багатомовність;

— швидке оновлення даних та зручний інтерфейс, що дозволяє переглядати велику кількість результатів одночасно.

Однак, незважаючи на загальну потужність FlashScore, є ряд недоліків, які обмежують користувацький досвід, особливо з точки зору персоналізації.

Найбільш очевидні з них:

— відсутність гнучких налаштувань сповіщень, користувачі не можуть точно вказати, коли сповіщення мають надсилатися перед матчем, або отримувати сповіщення лише після оголошення складу команди;

— узагальнений, а не персоналізований підхід, застосунок орієнтований на масову аудиторію, що ускладнює вибір лише тих матчів, які дійсно цікавлять конкретних користувачів;

— складна навігація для нових користувачів - велика кількість функцій та блоків ускладнює швидке отримання необхідної інформації, екран додатку зображений на рисунку 1.2.

Перевагами ScoreLine над FlashScore є простота використання, можливість вибору точних умов сповіщення (наприклад, за 10 хвилин до початку матчу або при оголошенні складів команд), легкий доступ до персоналізованих даних (наприклад, за 10 хвилин до початку матчу або при оголошенні складів команд), можливість доступу до персоналізованої інформації за 10 хвилин до початку матчу або при оголошенні складів команд.

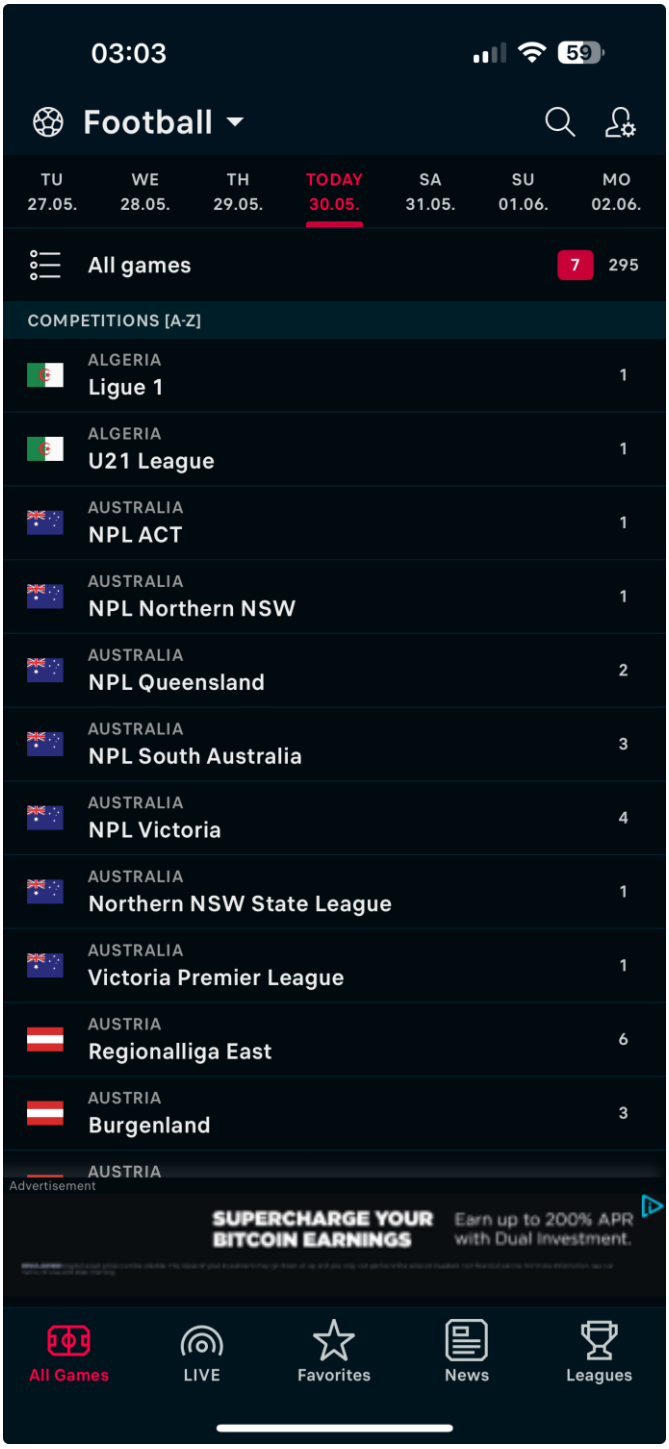


Рис. 1.2 FlashScore (головний екран)

1.4.2 SofaScore

SofaScore - це потужний мобільний застосунок для відстеження спортивних подій в режимі реального часу, який, на відміну від FlashScore, фокусується не лише на результатах, але й на глибокій статистиці, аналізі та візуалізації. Завдяки підтримці понад 20 видів спорту, включаючи футбол, баскетбол, теніс та інші, ви можете переглядати статистику матчів, порівнювати гравців, вивчати графіки активності команд, динаміку володіння м'ячем та інші аналітичні показники.

Однак, незважаючи на багатий набір функцій, SofaScore має ряд недоліків, які роблять його менш зручним для користувачів, які шукають швидкий і персоналізований досвід:

Перевантажений інтерфейс - велика кількість даних та візуалізацій ускладнює навігацію для нових користувачів, особливо на мобільних пристроях з невеликим екраном, що зображено на рисунку 1.3.

Низька гнучкість сповіщень - хоча сповіщення присутні, їхня функціональність все ще обмежена стандартними сповіщеннями, без урахування індивідуальних тригерів (наприклад, відмітка N-хвилини або поява складу).

Логіка вибору улюблених матчів є складною - користувачі повинні пройти кілька етапів, перш ніж вони зможуть налаштувати «відстеження» для окремих матчів або команд.

Відсутність простого календарного перегляду матчів - хоча є можливість переглянути розклад, немає простого способу візуалізувати матчі за датою або виділити важливі матчі.

У порівнянні з SofaScore, ScoreLine фокусується на простоті, зручності навігації та максимальній персоналізації.

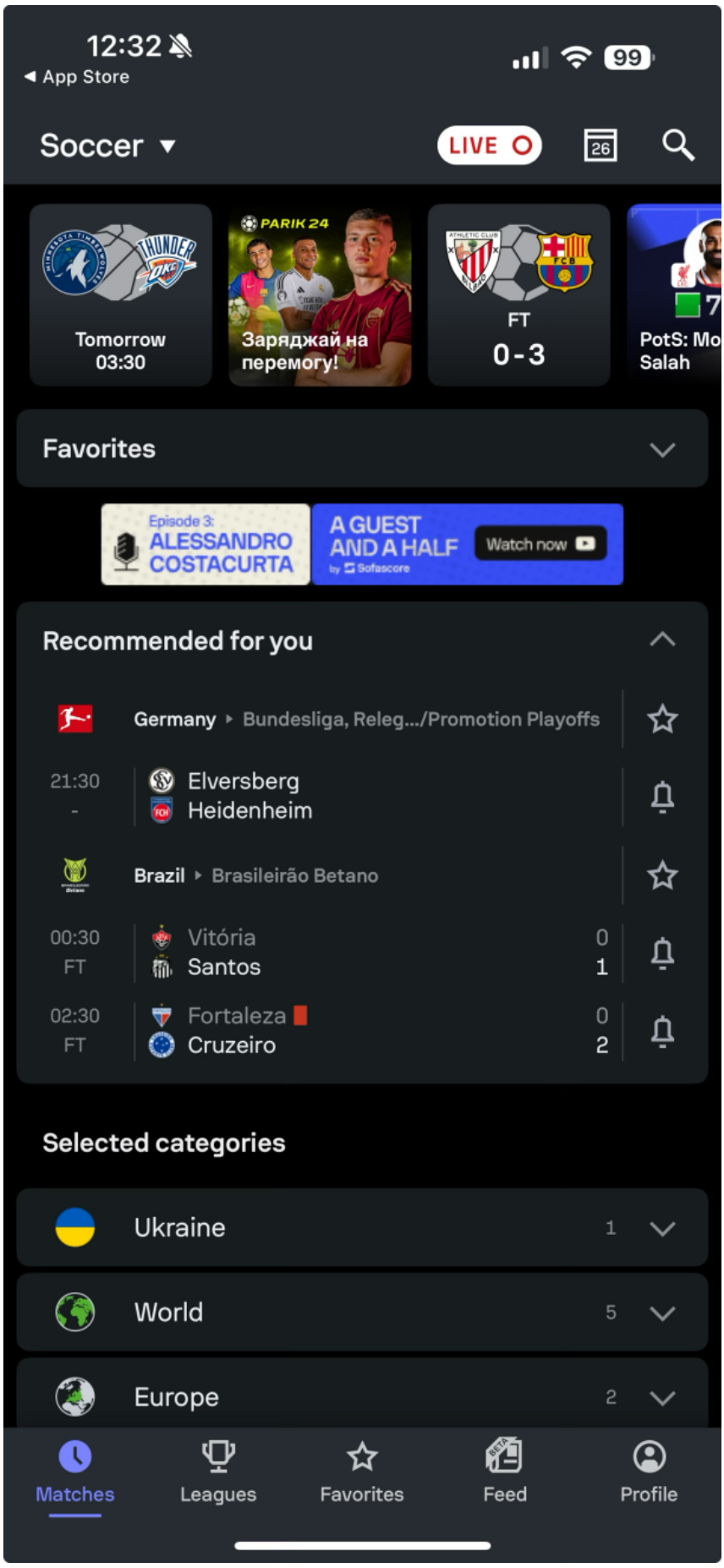


Рис. 1.3 SofaScore (головний екран)

1.4.3 LiveScore

LiveScore - один з найстаріших сервісів для відстеження спортивних результатів, пізніше трансформований у мобільний застосунок. Його головна перевага - швидке та послідовне надання результатів і статусу гри без зайвих складнощів. Застосунок має простий інтерфейс, орієнтований на результати в режимі реального часу, підтримує кілька видів спорту і дозволяє додавати команди до обраних для швидкого і зручного доступу.

У той же час, LiveScore явно поступається сучасним додаткам з точки зору дизайну та функціональності:

Обмежена функціональність – застосунок не надає детальну статистику матчів, інформацію про склад, графіки та аналітику, що є основним завданням гравців на інших ринках.

Примітивна система сповіщень - користувач отримує базові сповіщення, які неможливо налаштувати: немає можливості обрати час сповіщення або прив'язки до складу команди.

Застарілий інтерфейс - візуальна частина додатку виглядає застарілою, з невеликою підтримкою тем та елементів відповіді, зображення інтерфейсу на рисунку 1.4.

Відсутність навігації за календарем - пошук матчів за конкретними датами або подіями не є ані зручним, ані інтуїтивно зрозумілим.

На цьому тлі ScoreLine виглядає як сучасне, адаптивне та орієнтоване на користувача рішення. Застосунок пропонує не тільки гнучку систему нагадувань, але й простий, інтуїтивно зрозумілий календар, можливість зберігати улюблені команди, а також чітку структуру інтерфейсу, яка забезпечує миттєвий доступ до необхідної інформації. Технічно ScoreLine реалізовано з використанням сучасного фреймворку SwiftUI, який дозволяє безперешкодно швидко масштабувати його та адаптувати під майбутні потреби користувачів, тоді як LiveScore залишається радше класичним, але обмеженим у розвитку варіантом.

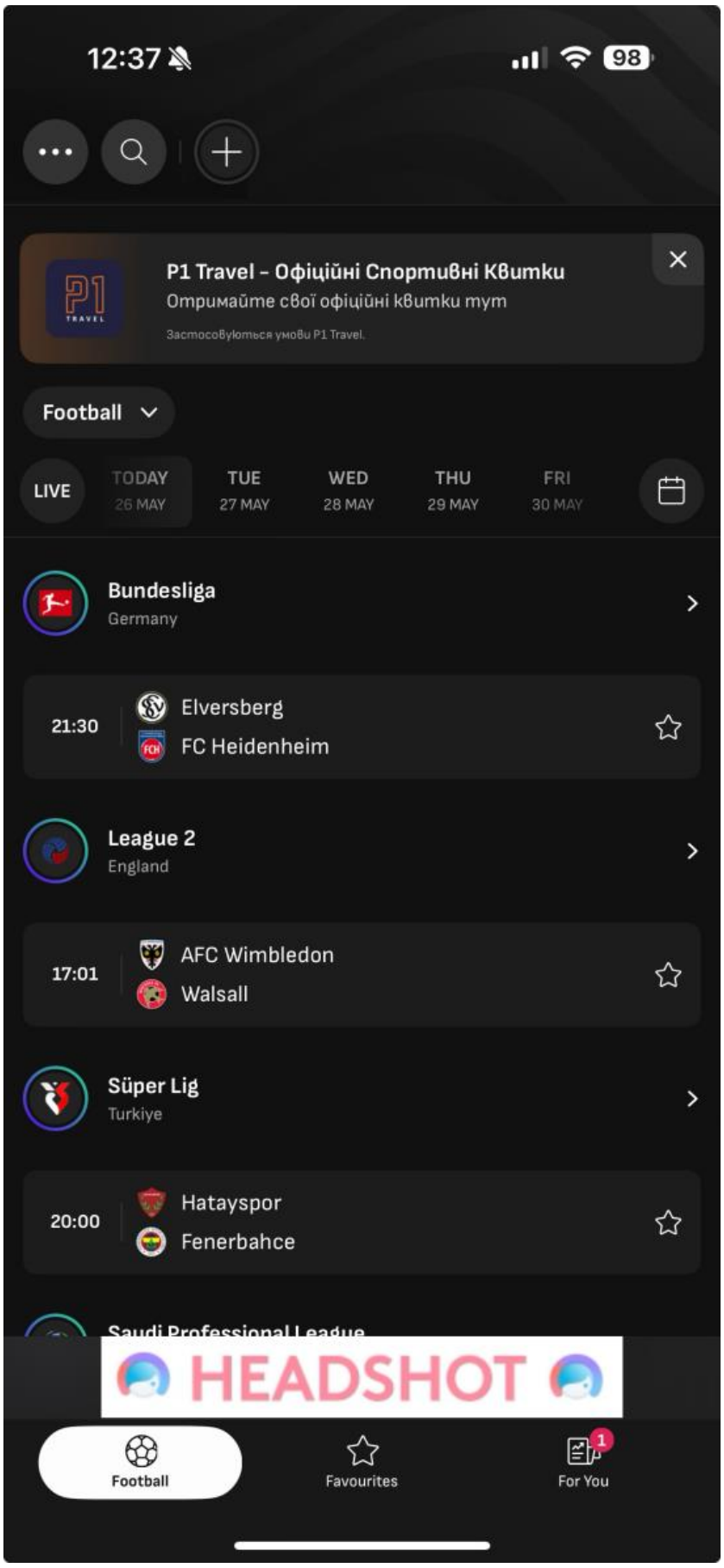


Рис. 1.4 LiveScore (головний екран)

1.4.4 OneFootball

OneFootball — це сучасний мобільний застосунок, що поєднує функції спортивного трекера та новинного ресурсу. Його головна особливість — фокус на футбольній тематиці: окрім розкладу матчів та результатів, користувачі мають доступ до статей, інтерв'ю, відеокоментарів, чуток про трансфери та аналітики з усього світу. OneFootball також підтримує базові сповіщення про матчі, новини команд та голи в режимі реального часу.

Незважаючи на сучасний вигляд і багатий контент, OneFootball має деякі очевидні обмеження для користувачів, які шукають гнучкий інструмент для індивідуального моніторингу матчів:

Контент має пріоритет над функціональністю — новини, статті та медіа займають більшу частину інтерфейсу, тому сам розклад матчів відходить на другий план і не має зручної структури або функції фільтрації.

Повідомлення недостатньо персоналізовані — хоча користувачі можуть вибрати улюблені команди, вони не можуть налаштувати час і умови отримання повідомлень; наприклад, немає повідомлень про оприлюднення складу команд.

Інформаційне перевантаження інтерфейсу — візуально застосунок схожий на новинний портал, містить велику кількість графічних блоків, що ускладнює швидкий доступ до розкладу, зображення на рисунку 1.4 відображає всю проблему.

Обмежені можливості взаємодії з розкладом матчів — можна переглядати за датою, але це не підходить для візуального аналізу або швидкого пошуку матчів у конкретному чемпіонаті.

На відміну від OneFootball, ScoreLine зосереджується на наданні зручної функції перегляду розкладу та точної інформації про важливі події без надмірного інформаційного навантаження. Він пропонує структурований розклад, персоналізовані повідомлення з можливістю налаштування умов, а також мінімалістичний і чіткий інтерфейс, придатний для будь-яких пристроїв. Таким чином, ScoreLine зосереджується на найголовнішому — наданні ефективного, швидкого та гнучкого моніторингу футбольних подій у формі, яка відповідає потребам користувачів.

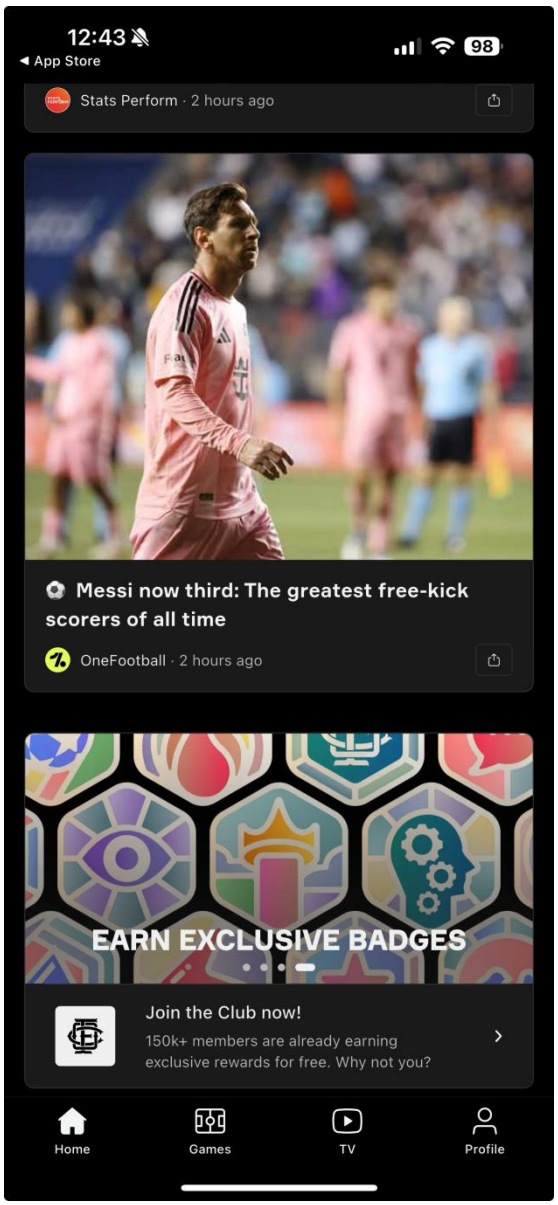


Рис. 1.5 OneFootball (головний екран)

1.4.5 FotMob

FotMob — це одна з найпотужніших і найкраще розроблених футбольних програм. Її основна увага приділяється детальним статистичним даним про матчі, текстовим трансляціям у реальному часі, новинам та аналітиці. Програма охоплює сотні ліг і чемпіонатів по всьому світу, дозволяючи користувачам отримувати повідомлення про голи, стартові склади, закінчення матчів та інші важливі події.

Незважаючи на свою популярність і потужність, FotMob має деякі обмеження в порівнянні з такими цілеспрямованими і простими рішеннями, як ScoreLine:

— занадто складний інтерфейс: дизайн інтерфейсу такий, що навіть найпростіші функції часто приховані під декількома рівнями меню; швидкий перегляд вибраних матчів або чемпіонатів може бути дуже незручним;

— стандартні сповіщення не мають гнучких налаштувань — хоча FotMob дозволяє вибрати тип події для сповіщення, неможливо встановити точний час або умови, що знижує рівень персоналізації;

— занадто великий обсяг інформації: користувачі, які хочуть лише розклад і короткий підсумок, можуть відчувати інформаційне перевантаження через аналітику, цитати, новини та текстові трансляції;

— низька адаптивність до швидкої взаємодії: відсутність зручних механізмів для швидкого додавання матчів до «Вибраного», навігації в календарі тощо.

ScoreLine вирішує ці проблеми, зосередившись на мінімалізмі, швидкій реакції та точній персоналізації. Інтерфейс максимально спрощений, але не втрачає функціональності; матчі легко додавати до обраного, переглядати в зручному календарі, а повідомлення надсилаються відповідно до обраного користувачем сценарію. Таким чином, ScoreLine не тільки надає користувачам доступ до інформації, але й контроль над нею, і все це в дуже зручній формі.

Таблиця 1.1

Порівняння існуючих програм-аналогів

Застосунок	Розклад матчів	Live-результати	Статистика	Сповіщення	Реклама
FlashScore	на тиждень	базові	базова	про початок матчу (із затримкою)	кожен екран
SofaScore	на місяць	базові	базова	про початок матчу (із затримкою)	кожна сторінка
LiveScore	тільки список	урізані	мінімальна	про початок матчу (із затримкою)	по дві на сторінку

Продовження таблиці 1.1

Порівняння існуючих програм-аналогів

OneFootball	базовий розклад	базові	базова	про початок матчу (із затримкою)	кожні 3-4 пости з новин та на кожній сторінці	середня
FotMob	календар	немає	урізана	немає	по дві на сторінку	середня
ScoreLine	окремий екран під розклад	базові	базова	за 10/30/60 хв до початку матчу або при появі складів на матч	-	низька

1.4 Визначення вимог до застосунку

Мобільний застосунок реалізує повний набір функціональних можливостей, спрямованих на персоналізований моніторинг футбольних матчів. Основні вимоги до системи включають:

— отримання актуальних даних про футбольні матчі з використанням зовнішнього API;

— відображення детальної інформації про матчі, включаючи склад команд, події, час та рахунок;

— додавання окремих матчів до списку обраного для зручного доступу;

— надсилання локальних сповіщень за індивідуально заданими умовами (наприклад, перед початком матчу або при появі складу);

— перегляд попередніх результатів матчів для аналітики та історії;

— підтримка персоналізованих налаштувань інтерфейсу, зокрема вибору мови, теми оформлення та розміру шрифтів.

Ці можливості забезпечують гнучкість, зручність використання та адаптацію застосунку під потреби кожного користувача.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Мобільний застосунок ScoreLine розроблений для операційної системи iOS з використанням сучасного технологічного стеку, з акцентом на високу продуктивність, масштабованість та простоту обслуговування коду. Основним середовищем розробки є офіційний IDE від Apple — Xcode (версія 16.1), який повністю інтегрований з усіма інструментами для створення, тестування та аналізу додатків в екосистемі Apple. Мовою програмування є Swift (версія 5.9)[2] — сучасна, безпечна та оптимізована для створення мобільних додатків мова. Для побудови інтерфейсу використовується SwiftUI[3] — декларативний фреймворк нового покоління, що дозволяє створювати адаптивні, динамічні інтерфейси з мінімальними витратами на оновлення стану. Архітектура базується на моделі MVVM (модель-представлення-модель представлення), яка забезпечує розділення обов'язків і спрощує тестування окремих компонентів.

Проєкт повністю підтримує динамічну зміну мови інтерфейсу та теми відповідно до налаштувань системи. Функціональне тестування проводилося за допомогою емуляторів iPhone 16 Pro та iPhone 16 Pro Max, щоб забезпечити адаптивність дизайну до різних розмірів екрану. В результаті було створено надійне та сучасне програмне середовище, готове до подальшого розширення функціоналу та підтримки нових версій iOS.

2.1.1 Xcode

Xcode 16.0— це офіційне інтегроване середовище розробки (IDE) від компанії Apple[4], яке використовується для розробки мобільних додатків для операційних систем iOS, iPadOS, macOS, watchOS та tvOS. Під час розробки додатка ScoreLine Xcode використовувався як основна платформа для написання коду Swift, управління ресурсами проєкту (активами, локалізацією), створення та перевірки інтерфейсу SwiftUI, виконання тестів та компіляції інсталяційного

пакета додатка. Крім того, Xcode має вбудовані інструменти для попереднього перегляду інтерфейсу користувача (preview), аналізу продуктивності додатків (інструмент Instruments) та налагодження помилок у реальному часі. У цьому середовищі також можна керувати цільовими платформами та їх версіями, включаючи імітацію різних моделей iPhone, що дозволяє гнучко тестувати додатки без використання фізичних пристроїв.

2.1.1 iOS Simulator

iOS Simulator — це вбудований компонент середовища Xcode, який імітує роботу додатків на різних пристроях Apple. Під час тестування він використовується для перевірки поведінки інтерфейсу, правильного спрацьовування сповіщень, завантаження мережевих даних, а також для оцінки адаптивності інтерфейсу до різних розмірів екрану. Зокрема, було протестовано роботу на моделях iPhone 16 Pro та iPhone 16 Pro Max.

2.2 Мова програмування Swift

Swift 5.9 — розроблена компанією Apple типова безпечна мова програмування з високою продуктивністю. Вона значно спрощує написання сучасного, сумісного та надійного коду. Вона інтегрована з SwiftUI, Combine, URLSession[10], Codable та іншими фреймворками.

2.3 Фреймворки

Фреймворки — це спеціалізовані бібліотеки або набір компонентів, які дозволяють спростити та прискорити розробку програмного забезпечення, забезпечуючи повторне використання коду та дотримання стандартів. У випадку розробки мобільного застосунку ScoreLine фреймворки виконують ключові ролі в побудові інтерфейсу, реактивному управлінні даними, взаємодії з мережею та системними службами.

2.3.1 SwiftUI

SwiftUI — декларативний фреймворк інтерфейсу користувача для створення адаптивних інтерфейсів. У поєднанні з Combine підтримує анімацію, адаптивне оновлення інтерфейсу користувача, темні/світлі теми, динамічні шрифти та локалізацію.

2.3.2 Combine

Combine — фреймворк для адаптивного програмування, що дозволяє підписуватися на зміни стану та обробляти події в режимі реального часу. Він широко використовується для комунікації між ViewModel та інтерфейсом.

2.4 Архітектура MVVM

MVVM (Model–View–ViewModel) — архітектурний шаблон, що дозволяє розділити бізнес-логіку, інтерфейс і модель даних. Покращує модульність, повторне використання коду та зручність тестування.

2.5 Засоби мережевої взаємодії

Засоби мережевої взаємодії — це інструменти, що забезпечують обмін даними між додатком та зовнішніми мережевими службами. У ScoreLine ці інструменти використовуються для надсилання HTTP-запитів до API-Football, обробки відповідей та перетворення отриманих JSON-даних у моделі, придатні для подальшого використання. Це дозволяє динамічно завантажувати розклад матчів, склад команд, статистичні дані та іншу інформацію. Крім того, правильна реалізація мережевої взаємодії забезпечує надійність, обробку помилок, повторні спроби підключення та гнучке управління тайм-аутами, що є надзвичайно важливим для роботи з нестабільним інтернет-підключенням.

2.5.1 URLSession

URLSession — API для HTTP-запитів, що дозволяє отримувати дані з API-Football. Дає змогу обробляти помилки, таймаути, статуси відповідей і заголовки.

2.5.2 Codable

Codable — протокол серіалізації/десеріалізації в Swift, який дозволяє швидко й безпечно працювати з JSON, перетворюючи його в структуровані моделі даних.

2.6 Операції з повідомленнями

UNUserNotificationCenter — це фреймворк для створення та управління локальними повідомленнями[5]. Він дозволяє планувати, відображати та реагувати на повідомлення, навіть якщо застосунок закрито.

2.7 Збереження локальних даних

UserDefaults — механізм для збереження легких параметрів (наприклад, улюблених команд, мови, типів повідомлень, тем) між сеансами запуску. Простий у використанні, достатній для задоволення потреб більшості користувачів у налаштуваннях.

2.8 Система управління базами даних

У цьому проєкті не використовується традиційна реляційна база даних, а як основне джерело даних використовується зовнішній RESTful API — API-Football. Це потужний сервіс, що надає структуровану інформацію про футбольні матчі, команди, гравців, події матчів, склади, результати тощо. Застосунок взаємодіє з цим API за допомогою HTTP-запитів і отримує відповіді у форматі JSON. Ці відповіді обробляються на клієнті і перетворюються в локальну модель даних.

Таким чином, роль системи управління базами даних у цьому проєкті виконує віддалений мережевий сервіс, який забезпечує своєчасність, надійність та

масштабованість отриманих даних. Це рішення не вимагає витрат ресурсів на локальне зберігання великого обсягу інформації та завжди використовує найсвіжіші результати. Крім того, застосунок використовує механізм локального кешування для конфігурації налаштувань користувача за допомогою UserDefaults.

2.9 Система контролю версій

GitHub — це мережева платформа для хостингу коду, заснована на системі контролю версій Git [11]. GitHub[14] швидко став однією з найвпливовіших платформ у сфері розробки програмного забезпечення, надаючи інструменти для співпраці, контролю версій та розподіленого доступу до коду[12].

Основним елементом GitHub є репозиторій для зберігання коду. Користувачі можуть створювати публічні або приватні репозиторії для своїх проєктів, що включають управління версіями за допомогою Git [11].

GitHub дозволяє користувачам «розгалужувати» (копіювати) репозиторії інших користувачів, вносити зміни та надсилати ці зміни авторам оригінальних репозиторіїв за допомогою запитів на витяг. Це ключова функція для спільної розробки відкритих проєктів[13].

2.10 Інструменти для дизайну користувацького інтерфейсу

Figma[15] — це веб-інструмент для дизайну інтерфейсів, який дозволяє командам створювати, співпрацювати та обмінюватися дизайнами інтерфейсів і прототипами. Figma була заснована Діленом Філдом та Евеном Воллесом у 2012 році і стала популярною серед дизайнерів завдяки своїй простоті у використанні, гнучкості та багатофункціональності.

Ключовою особливістю Figma є повна веб-інтеграція, що дозволяє користувачам працювати з будь-якого комп'ютера без необхідності завантажувати програму. Це забезпечує легкий доступ та співпрацю між учасниками проєкту незалежно від їхнього місцезнаходження.

Figma містить інструменти для створення інтерактивних прототипів без необхідності використання стороннього програмного забезпечення. Це дозволяє дизайнерам швидко створювати прототипи з анімацією переходів та мікроінтерактивними елементами і тестувати їх безпосередньо в середовищі Figma[18].

Figma підтримує створення та управління системами дизайну, дозволяючи дизайнерам зберігати стилі, компоненти та інші ресурси в легкодоступній бібліотеці, що спрощує координацію та повторне використання в декількох проєктах.

3 ПРОЄКТУВАННЯ

3.1 Загальні принципи архітектури

При розробці програми ScoreLine було обрано архітектурний підхід MVVM (модель-представлення-модель представлення), який дозволяє ефективно розділити логіку, інтерфейс і модель даних. Це спрощує обслуговування, тестування та розширення програми. Крім того, реалізовано чітку модульність у вигляді окремих компонентів View, ViewModel та сервісів. Архітектура REST використовується як основа для взаємодії з API, а декларативна модель SwiftUI використовується для побудови UI.

3.2 Типи архітектури

3.2.1 Архітектура клієнт-сервер

Архітектура клієнт-сервер — чітке розмежування клієнтської частини (мобільний застосунок) і серверної частини (API-Football), яка є єдиним джерелом інформації (Single Source of Truth). У ScoreLine застосунок надсилає HTTP-запити до API, отримуючи в режимі реального часу найсвіжіші дані про матчі, події та склади команд. Клієнт не зберігає ці дані локально, що робить застосунок більш легким і мінімізує потребу в синхронізації. Така архітектура також спрощує розширення та оновлення даних на сервері. Схема архітектури зображена на рисунку 3.1.

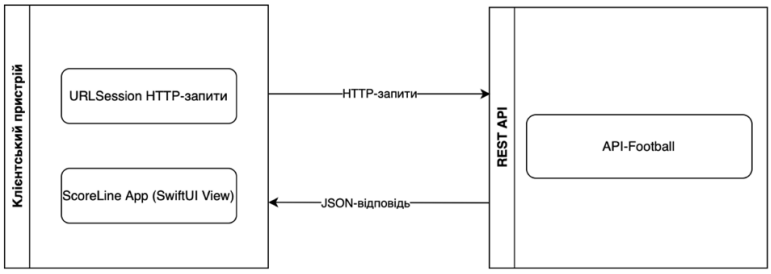


Рис. 3.1 Діаграма компонентів клієнт-серверної моделі

3.2.2 MVVM

MVVM (модель-представлення-модель представлення) — це архітектурна модель, яка забезпечує розділення представлення даних (представлення), бізнес-логіки (модель представлення) та моделі (модель). У ScoreLine кожне представлення має окремий ViewModel, схема зображена на рисунку 3.2, (наприклад, GamesViewModel, LineupsViewModel), який відповідає за завантаження даних через API, обробку помилок, перетворення моделі для відображення та управління станом (наприклад, список матчів, колекція, стан завантаження). Це дозволяє повторно використовувати логіку, легко тестувати частини ViewModel без інтерфейсу та забезпечувати розширюваність програми.

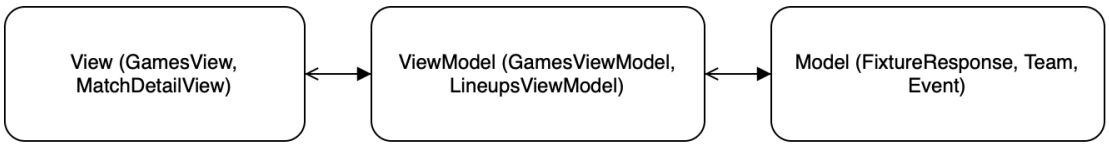


Рис. 3.2 Діаграма класів

3.2.3 Реактивна архітектура

Реактивна архітектура — реалізована за допомогою Combine, дозволяє встановити зв'язок між джерелом даних (API, внутрішній стан) та інтерфейсом користувача. ScoreLine використовує змінні @Published у ViewModel, які автоматично оновлюють інтерфейс при зміні значень, схематично зображено на рисунку 3.3. Наприклад, коли GamesViewModel отримує дані про змагання, список оновлюється в View без додаткового коду. Це забезпечує плавну роботу програми, знижує ризик несинхронізації стану та підвищує стабільність обробки асинхронного потоку даних.

Рис. 3.3 Діаграма послідовностей

3.3 Архітектура клієнтського боку

Архітектура клієнтського боку додатку ScoreLine базується на шаблоні MVVM [8](модель-представлення-модель представлення), використовує Combine для реактивного підходу та SwiftUI для макетування інтерфейсу. Всі логічні компоненти клієнтського боку чітко розподілені між відповідними модулями: View відповідає за інтерфейс користувача, ViewModel — за логічну обробку, а Model — за структуровані дані, що отримуються.

Основою побудови інтерфейсу є SwiftUI, що дозволяє формувати адаптивний та реактивний дизайн для різних пристроїв. Кожен екран, наприклад GamesView, MatchDetailView, CalendarView, реалізований як окрема структура на основі відповідного ViewModel (GamesViewModel, LineupsViewModel тощо). Дані передаються з API через службу URLSession, обробляються структурами, що відповідають протоколу Codable, і передаються до ViewModel для подальшого використання в інтерфейсі користувача.

Для взаємодії з API використовується окремий мережевий рівень, який надає послуги у формі запитів API-Football. Стан завантаження, помилки, отримані дані — все це обробляється в ViewModel і передається до View за допомогою @Published і Combine. Схема структури показана на рисунку 3.4.

Налаштування повідомлень і взаємодія з системними службами реалізовані в окремих менеджерах, таких як NotificationsManager, LineupNotifier, які також упаковують логіку планування повідомлень за допомогою UNUserNotificationCenter.

Для персоналізації додатка можна динамічно змінювати мову, тему та параметри повідомлень за допомогою UserDefaults та відповідних менеджерів (LanguageManager, AppLanguageLoader).

Такий підхід забезпечує розширюваність, тестованість та простоту обслуговування додатка.

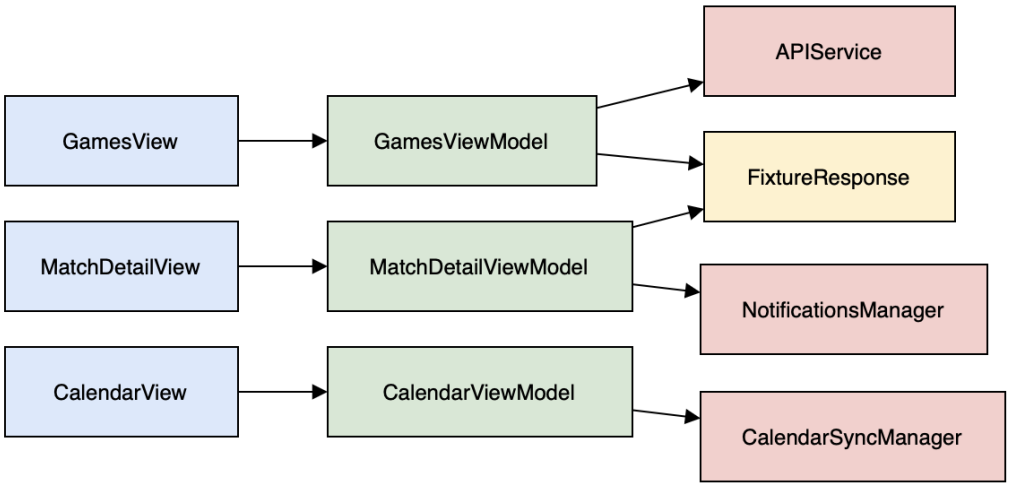


Рис. 3.4 UML Компонентна діаграма клієнтської архітектури

3.4 Логіка взаємодії з API

Застосунок ScoreLine не має окремої серверної частини, а безпосередньо взаємодіє із зовнішнім REST API (API-Football). Для забезпечення структурованості та розширюваності взаємодії в клієнтській частині реалізовано мережеву архітектуру, яка виконує роль «серверної логіки» всередині додатка.

Її основні складові:

APIService — відповідає за формування HTTP-запитів, включаючи заголовки, параметри та обробку відповідей.

EndpointBuilder — централізовано управляє маршрутизацією до різних кінцевих точок API, забезпечуючи гнучкість і масштабованість запитів.

ResponseDecoder — відповідає за перевірку стану відповіді, реєстрацію помилок і декодування JSON-даних у моделі Swift.

Кеш — використовує UserDefaults, NSCache або подібні локальні інструменти для тимчасового зберігання даних, що дозволяє зменшити кількість повторюваних запитів і прискорити завантаження.

Цей підхід реалізує всю логіку отримання, обробки та представлення даних без необхідності використання окремого сервера, що дуже корисно для одностороннього доступу до інформації.

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Дизайн інтерфейсу

Інтерфейс користувача програми ScoreLine відповідає принципам зручності використання, швидкого доступу до основної інформації та сучасних стандартів дизайну мобільних додатків. У програмі використовується SwiftUI — декларативний фреймворк, що дозволяє ефективно створювати гнучкі та адаптивні інтерфейси для пристроїв iOS. Це дає змогу швидко реалізовувати складні компоненти без надмірного повторення коду та забезпечує сумісність із майбутніми версіями системи.

Головний екран інтерфейсу зображено нижче:

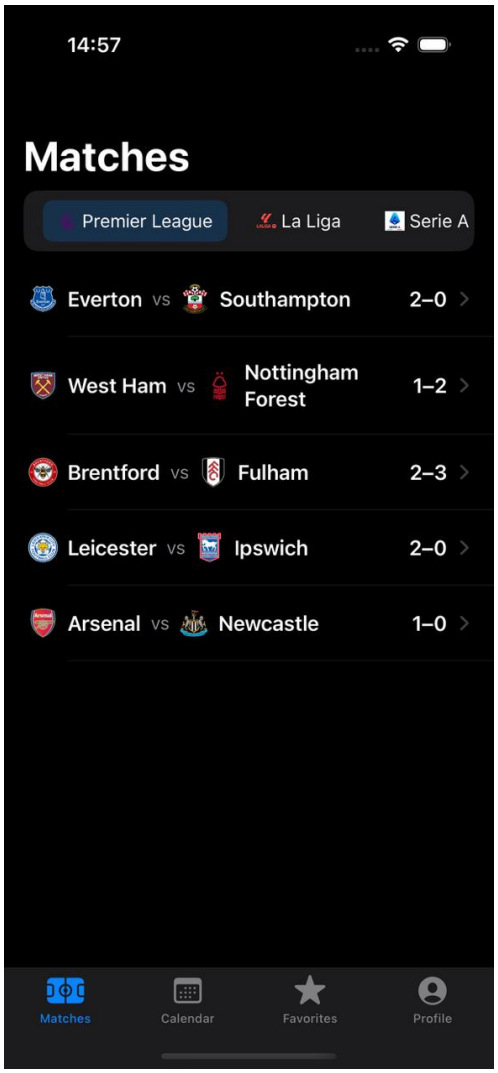


Рис. 4.1 Екранна форма головної сторінки

Головний екран — містить список майбутніх матчів, який автоматично оновлюється через API. Для кожного матчу відображаються назва команди, час початку, статус, логотип, а також можливість додати його до обраного.

Інтерфейс детальної інформації про матч — має дизайн у вигляді вкладок і надає повну інформацію про матч: події в хронологічному порядку (голи, жовті/червоні картки, заміни) - рис 4.2, повна статистика – рис 4.3, склад команди – рис 4.4, попередні результати – рис 4.5, таблиця ліги – рис 4.6.

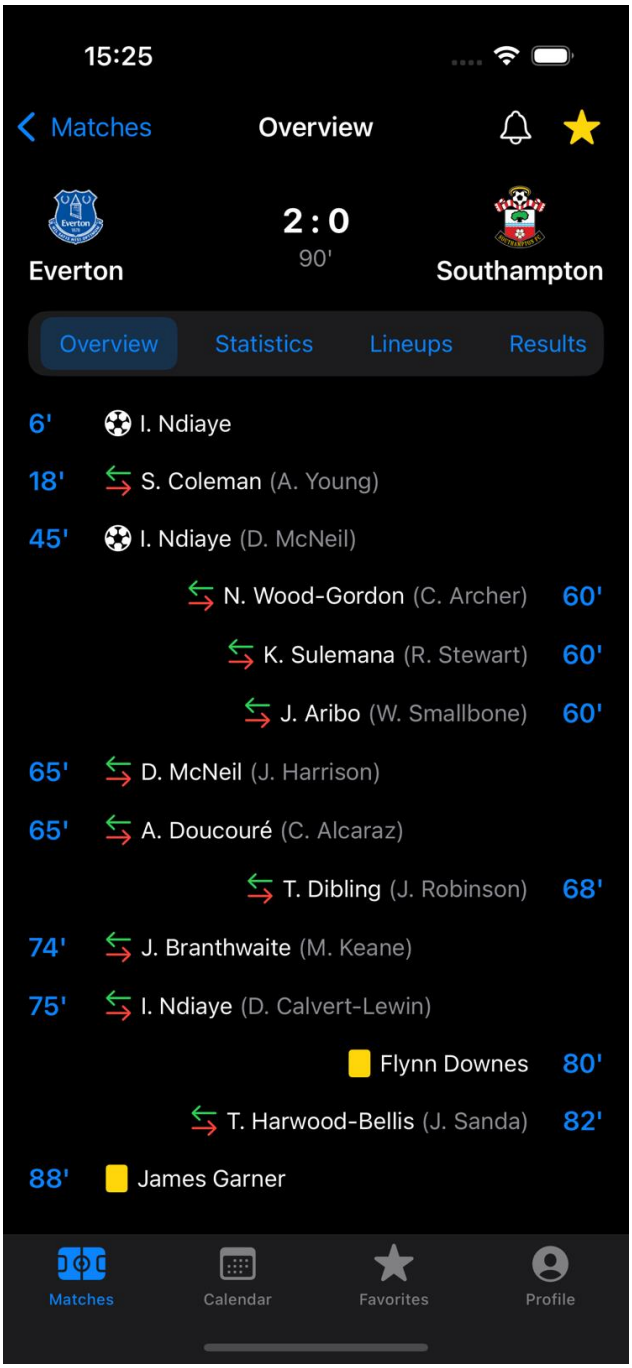


Рис. 4.2 Екранна форма інформації про матч

Рис. 4.3 Екранна форма сторінки статистики матчу

Рис. 4.4 Екранна форма складу команд

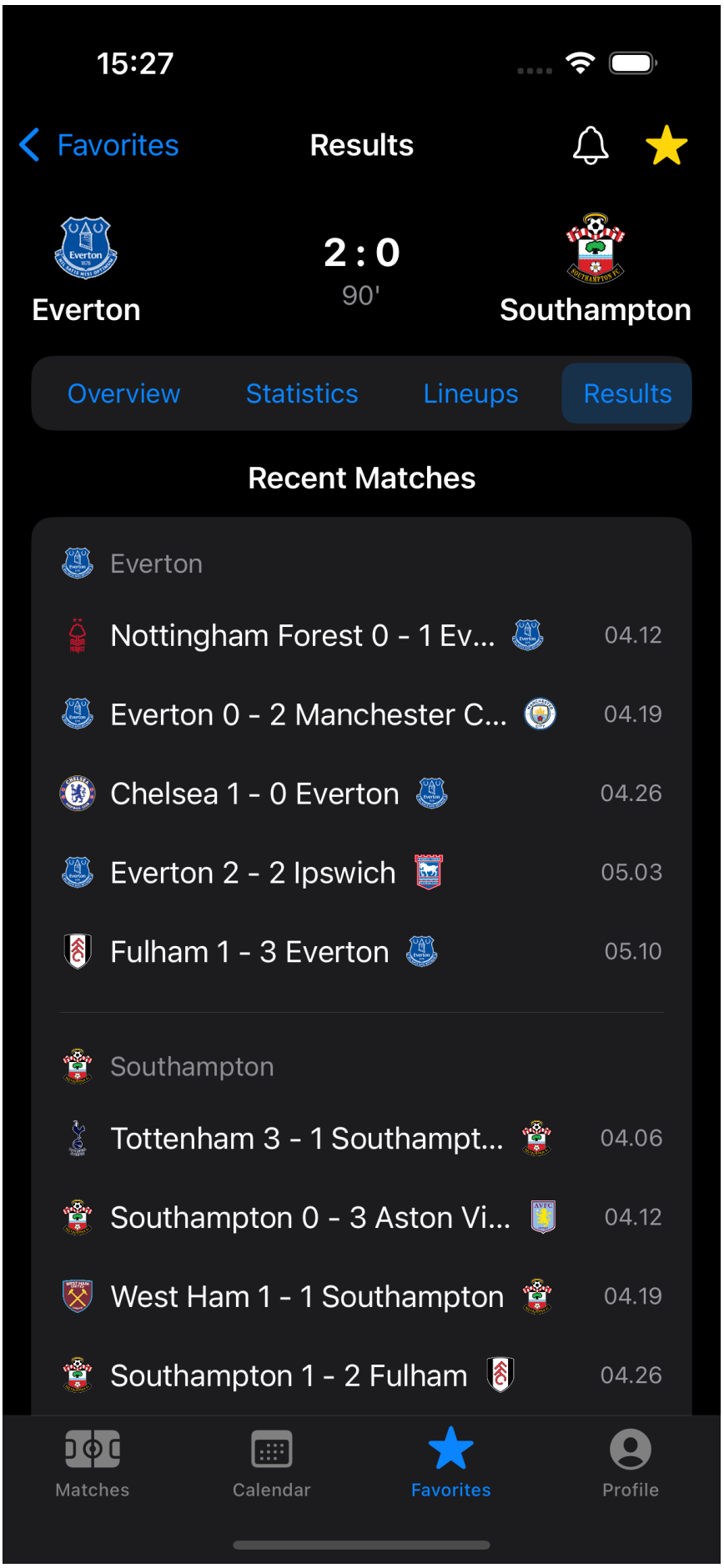


Рис. 4.5 Екранна форма попередніх результатів

Рис. 4.6 Екранна форма таблиці ліги

Налаштування — надають можливість зміни мови, управління темним/світлим режимом, активації локальних сповіщень.

Інші аспекти інтерфейсу:

Підтримка Dynamic Island — для нових моделей iPhone, сповіщення відображаються в інтерактивному режимі, без необхідності відкривати застосунок.

Іконки SF Symbols — використовуються замість звичайних зображень для кращої інтеграції з системною темою.

Дизайн — повна підтримка світлого та темного режимів. Компоненти автоматично адаптуються до поточної теми без необхідності перезапуску програми.

Особлива увага приділена оптимізації візуального відображення: всі елементи мають пропорційні проміжки, зручне розташування елементів керування[19], однаковий шрифт та розмір. Всі графічні компоненти стилізовані відповідно до єдиної візуальної теми, щоб зберегти цілісність дизайну програми використовувався @AppStorage[6]. Усі графічні компоненти стилізовані згідно з єдиною візуальною темою, що підтримує цілісність дизайну застосунку. Весь інтерфейс, включно з усіма елементами оформлення та зображеннями, динамічно адаптується до поточної системної теми пристрою. Завдяки використанню механізмів SwiftUI та можливості визначення окремих ресурсів для світлої та темної тем, застосунок виглядає органічно незалежно від обраного режиму. Це створює комфортне візуальне середовище, яке автоматично підлаштовується під уподобання користувача без потреби в ручному налаштуванні.

4.2 Реалізація клієнтської частини

Клієнтська частина додатку ScoreLine базується на архітектурній моделі MVVM (модель-представлення-модель представлення), яка забезпечує модульність, гнучкість, тестованість та спрощення розширення коду. Такий підхід дозволяє чітко розділити користувацький інтерфейс та логіку обробки даних, що значно покращує довгострокову підтримку проєкту.

Основною мовою програмування є Swift, яка є рідною мовою iOS і підтримує сучасні парадигми програмування, включаючи декларативне програмування. Інтерфейс побудований на базі SwiftUI, фреймворку, що дозволяє створювати гнучкі, адаптивні та реагуючі інтерфейси. Компоненти інтерфейсу (View) максимально розділені та можуть бути повторно використані, кожен компонент прив'язаний до відповідного ViewModel.

Кожна частина реалізована наступним чином:

View — відповідає лише за візуальне відображення стану, отриманого з ViewModel. За допомогою атрибутів @State, @Binding, @ObservedObjectта @EnvironmentObject можна ефективно реагувати на зміни даних без необхідності ручного оновлення інтерфейсу користувача.

ViewModel — виступає посередником між View та Model. Отримує дані з API за допомогою сервісу, форматує їх у відповідний для відображення вигляд та передає до View. Також відповідає за обробку подій з інтерфейсу (наприклад, обробку натискання кнопки «Додати до опцій»).

Model — представлений у вигляді Swift-структури, що відповідає формату відповіді API-Football. Усі моделі реалізують протокол Codable, що дозволяє автоматично декодувати відповіді JSON без додаткової ручної обробки.

Логіка сервісу реалізована у вигляді окремих керованих компонентів:

APIService — відповідає за формування мережевих запитів, обробку помилок, повторні спроби, запис, кешування та обробку відповідей. Надає централізований доступ до зовнішніх API та спрощує підтримку змін структури запитів.

NotificationManager — інтегрований з UNUserNotificationCenter, дозволяє планувати локальні сповіщення відповідно до заданих сценаріїв: перед матчем, коли склад доступний або після зміни статусу події. Приклад роботи зображен на рисунку 4.2.

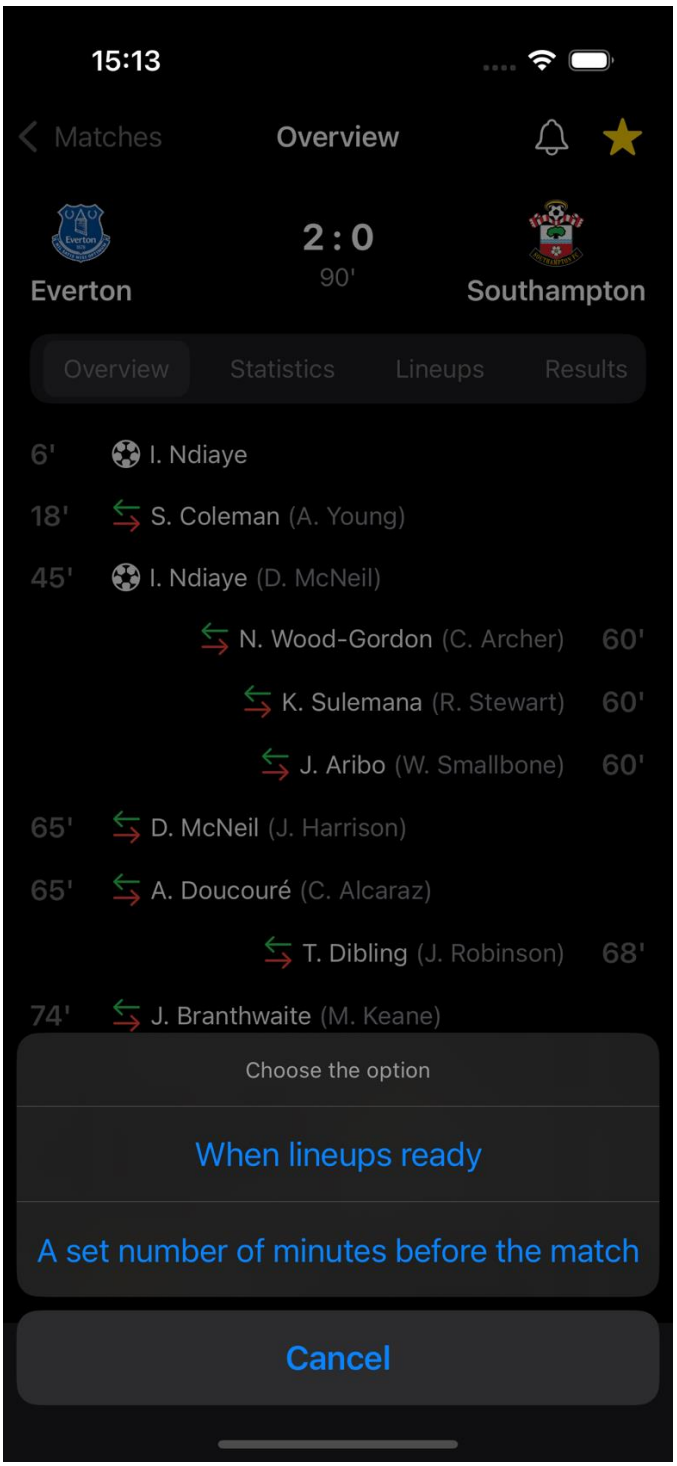


Рис. 4.2 Екранна форма з опціями вибору типу повідомлення

FavoritesManager — використовує UserDefaultsабо FileManagerдля управління збереженими матчами, вибраними користувачем, нижче наведено зображення екрану цього розділу рис 4.3.

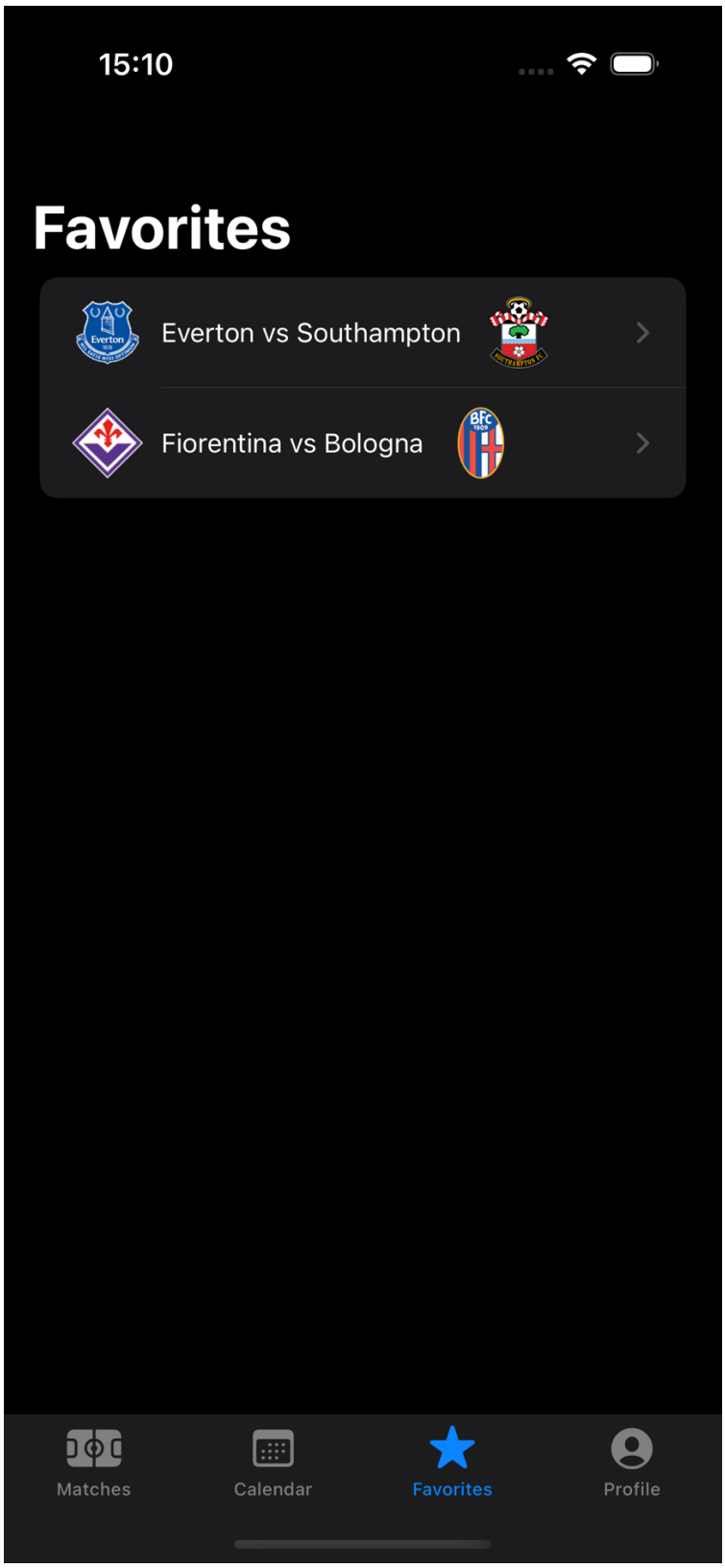


Рис. 4.3 Сторінка улюблених матчів

Вся логіка організована модульно відповідно до принципів SOLID. Код розділений на логічні частини (Views, ViewModels, Models, Services, Resources), що спрощує розширення функціоналу, написання модульних тестів і рефакторинг. Реалізовано підтримку тем (світлий/темний) і локалізації (українська/англійська).

Для складних елементів інтерфейсу використовуються анімація, власні компоненти SwiftUI і SF Symbols.

Крім того, реалізовано систему навігації за допомогою NavigationStack, яка обробляє параметри маршрутизації та підтримує глибокі посилання, що дозволяє відкривати певні відповідності або екрани за зовнішніми посиланнями. Всі компоненти додатка за порадами джерела[20] на різних розмірах екранів, включаючи Dynamic Island та різні системні конфігурації. На основі шаблону архітектури MVVM (Model-View-ViewModel) можна розділити логіку даних, представлення та взаємодію. Основною мовою програмування є Swift, що забезпечує стабільність, високу продуктивність та інтеграцію з екосистемою iOS.

Основна бізнес-логіка розподілена між такими компонентами:

View — інтерфейсні компоненти, створені за допомогою SwiftUI. Вони підписуються на відповідні ViewModel і реагують на зміни стану за допомогою механізмів @State, @ObservedObjectабо @EnvironmentObject.

ViewModel — містить логіку обробки даних, отриманих з моделі або API. Використовує Combine[7] для публікації змін і комунікації з View.

Model — сутності, що відображають структуру даних, отриманих через API (матчі, команди, події, склади тощо). Вони реалізовані як структури, що відповідають протоколу Codable, для зручності декодування JSON-відповідей.

Функціональні компоненти:

APIService — реалізує мережеву взаємодію з API-Football, формує запити, отримує дані та обробляє можливі помилки.

NotificationManager — відповідає за планування локальних сповіщень про події (початок матчу, склад команди).

FavoritesManager — реалізує функцію додавання та збереження вибраних матчів.

Під час реалізації також було приділено увагу розширюваності та повторному використанню коду. Наприклад, список матчів реалізовано як окремий компонент із введенням параметрів даних. Події матчів використовують шаблон enum + switchдля універсального відображення різних типів подій.

Код структурований у вигляді модулів: Views, ViewModels, Models, Services, що спрощує навігацію та обслуговування проєкту, струкра зображена на рисунку 4.4. Застосунок протестовано на реальних пристроях iPhone, підтримує екрани різних розмірів та динамічний інтерфейс.

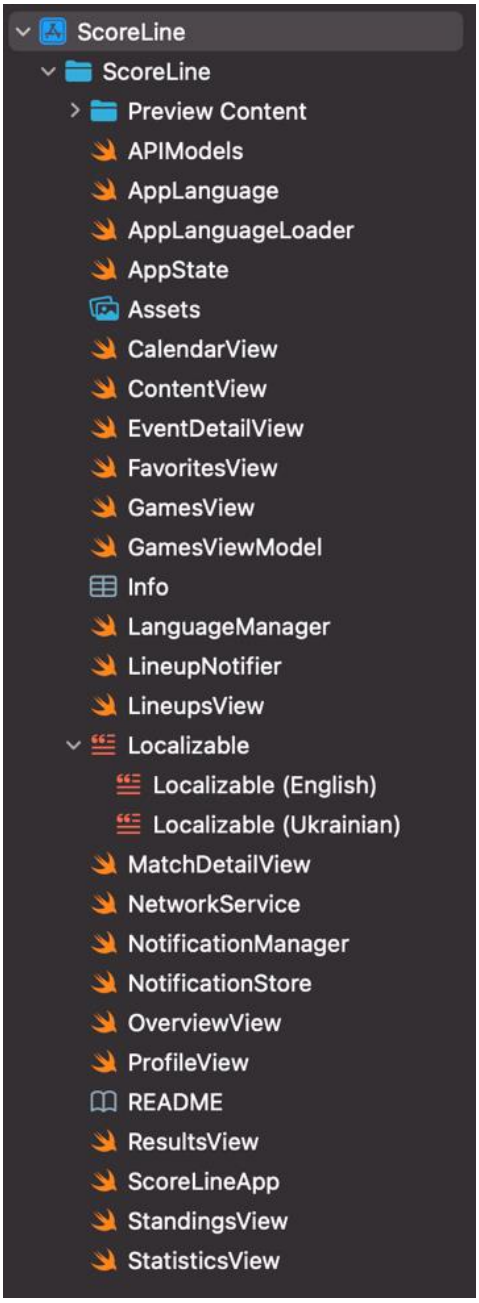


Рис. 4.4 Приклад структури SwiftUI проєкту після ініціалізації

4.3 Реалізація роботи API

У додатку ScoreLine взаємодія з API-Football відіграє ключову роль, оскільки API-Football є єдиним джерелом даних про футбольні матчі. Щоб забезпечити

стабільну, гнучку та розширювану взаємодію з API, було створено окремий рівень обслуговування, який ізольований від інтерфейсу та бізнес-логіки. Основним компонентом є **APIService**, який використовує URLSession для реалізації всіх HTTP-запитів. Він увібрав у себе логіку відправлення запитів, отримання та обробки відповідей, а також повторних спроб у разі виникнення помилок.

Файл APIService.swiftмістить єдину точку взаємодії з мережею. Наприклад, метод fetchFixtures (for teamId: Int, season: Int) використовує шлях Endpoint.fixtures (teamID: teamId, season: season)для надсилання запиту до API з метою отримання розкладу матчів команди. Цей шлях побудовано в Endpoint.swift, де кожен випадок enum повертає відповідний URL-адресу.

Нижче наведено частину коду з APIService.swift:

```
func fetchFixtures(for teamId: Int, season: Int) async throws -> [FixtureResponse] {
    let endpoint = Endpoint.fixtures(teamID: teamId, season: season)
    let request = endpoint.request
    let (data, response) = try await URLSession.shared.data(for: request)
    try validate(response: response)
    let decoded = try JSONDecoder().decode(FixtureListResponse.self, from: data)
    return decoded.response
}
```

Ця функція виконує асинхронний мережевий запит, перевіряє стан відповіді, а потім декодує отриманий JSON у модель FixtureListResponse, яка містить масив FixtureResponse. У разі помилки цей метод генерує виняток, який обробляється у ViewModel. Він підтримує різні типи запитів (GET, POST) і динамічно генерує URL-адреси на основі даних у файлі Endpoint.swift, який містить список доступних кінцевих точок у вигляді переліку.Наприклад, Endpoint.fixtures (teamID: Int)або Endpoint.lineups (fixtureID: Int)генерують повний URL-адресу з параметрами.

Отримані дані передаються до ResponseHandler, який перевіряє код стану відповіді, веде журнал або повідомляє про помилки. Потім дані передаються до

декодера, який за допомогою Codableавтоматично перетворює JSON у відповідну модель (FixtureResponse, TeamResponse, EventResponse тощо).

Для кожного типу запиту в APIServicестворюється окрема функція, яка викликається в ViewModel асинхронно. Наприклад, метод fetchFixtures (for:) або fetchLineups (fixtureID:). Це дозволяє чітко контролювати процес обробки та повторне використання логіки. У разі помилки або переривання мережевого з'єднання сервіс може повернути дані з кешу (NSCache, UserDefaults) або відобразити відповідне повідомлення.

Таким чином, реалізація API в додатку ScoreLine відповідає принципам інкапсуляції, повторного використання та стабільності. Весь код API логічно розміщений у каталозі Services, що полегшує розширення підтримки нових запитів у майбутньому.

4.4 Реалізація автоматичної зміни теми

Застосунок ScoreLine повністю підтримує автоматичну зміну теми відповідно до системних налаштувань пристрою. Темні та світлі теми можна застосовувати динамічно, без додаткового перемикання з боку користувача. Завдяки SwiftUI зміна теми відбувається автоматично: всі основні кольори інтерфейсу (текст, фон, елементи навігації) налаштовуються відповідно до активної теми за допомогою системних кольорів (Color.primary, Color.secondary, Color(.systemBackground) тощо). Це забезпечує високу контрастність, читабельність та відповідність рекомендаціям Apple щодо доступності.Крім кольорів, зображення в ресурсі Assets.xcassets мають дві версії — світлу та темну. Наприклад, логотипи команд, піктограми тегів та елементи фону мають окремі версії, які автоматично замінюються при зміні теми. Це дозволяє реалізувати цілісний дизайн без втрати візуальної якості.

Наступні приклади коду ілюструють використання системних кольорів у фоні:

.background (Color (.systemBackground)), або у тексті:

.foregroundColor (Color.primary).

Таке використання гарантує, що всі елементи відповідають стилю обраної теми. Тестування зміни теми проводилося в різних конфігураціях: ручне перемикання теми в системних налаштуваннях та автоматична зміна режиму (наприклад, залежно від часу доби).У всіх випадках застосунок стабільно адаптується, забезпечуючи естетичність і зручність обох версій. Таким чином, інтерфейс додатка є дуже гнучким, повністю адаптується до індивідуальних налаштувань користувача, підвищуючи комфорт використання та персоналізацію. Нижче наведено рисунок як саме це реалізовано у проєкті.

4.5 Реалізація автоматичної зміни мови

Ця програма підтримує кілька мов інтерфейсу та автоматично налаштовується відповідно до мови системи пристрою. Під час першого запуску програма автоматично визначає мову системи користувача та завантажує відповідні локалізовані рядки.

Локалізовані файли зберігаються в окремому файлі .stringsy проєкті Xcode (наприклад, Localizable.strings (Ukrainian), приклад зображен на рисунку 4.5, та Localizable.strings (English)).

Кожна текстова константа в додатку викликається за допомогою функції NSLocalizedString, яка забезпечує динамічне відображення перекладеного інтерфейсу.

Нижче наведено приклад реалізації цієї функції в проєкті:

Text (NSLocalizedString («match_details», comment: «»)).

Така реалізація дозволяє не тільки змінювати мову відповідно до мови системи, але й розширювати застосунок на інші мови в майбутньому шляхом додавання нових локалізованих файлів.

Крім тексту, підтримується локалізація дати, формату часу та повідомлень. Використання DateFormatterразом з localeзабезпечує правильне відображення дати відповідно до регіональних налаштувань.

Ця функція була протестована на пристроях з різними мовами інтерфейсу (українська, англійська) — весь текст, дати та повідомлення відображаються відповідно до системних налаштувань, без необхідності ручного вибору мови користувачем.

Таким чином, реалізація автоматичної зміни мови підвищила зручність використання програми ScoreLine та забезпечила локалізований досвід для широкого кола користувачів.

```
"Matches" = "Матчі";
"Profile" = "Профіль";
"Favorites" = "Улюблені";
"Notifications" = "Сповіщення";
"Match Details" = "Деталі матчу";
"Lineups" = "Склади";
"Statistics" = "Статистика";
"Overview" = "Огляд";
"Results" = "Результати";
"Standings" = "Турнірна таблиця";
"Interface Style" = "Стиль інтерфейсу";
"Theme" = "Тема";
"System" = "Системна";
"Light" = "Світла";
"Dark" = "Темна";
"Language" = "Мова";
"Calendar" = "Календар";
"Show Matches" = "Показати матчі";
"Match Details" = "Деталі матчу";
"Choose the option" = "Обери опцію";
"A set number of minutes before the match" = "Поставити кількість хвилин до матчу";
"Cancel" = "Відмінити";
"Select when to be notified" = "Оберіть коли ви хочете щоб вас повідомили";
"Minutes before match" = "Хвилин до матчу";
"Set Notification" = "Встановити сповіщення";
"Match has not started yet" = "Матч ще не розпочався";
"minutes match starts" = "хвилин матч розпочнеться";
"After" = "Через";
"NS" = "Не розпочався";
"HT" = "Перерва";
"FT" = "Закінчився";
"Starting Lineups" = "Стартові Склади";
"Substitutes" = "Лава запасних";
"Lineups will appear here once available" = "Склади з'являться коли будуть доступні";
"Standings not available" = "Таблиця не доступна";
"When lineups ready" = "Коли будуть готові склади";
"Loading recent matches..." = "Завантаження останніх матчів…";
"Invalid URL" = "Недійсний URL";
"API key not found" = "Ключ API не знайдено";
"Error: %@" = "Помилка: %@";
"Recent Matches" = "Останні матчі";
"Team 1" = "Команда 1";
"Team 2" = "Команда 2";
"No matches today for " = "На сьогодні матчів немає у";
/* Metric types */
"Statistics" = "Статистика";
"Top stats" = "Основні показники";
"Shots" = "Удари";
"Passes" = "Передачі";
```

Рис. 4.5 Приклад структури файлу локалізації на українську мову

4.6 Завантаження проєкту на GitHub

Щоб додати проєкт SwiftUI до існуючого репозиторію на GitHub, потрібно виконати кілька кроків. Ці кроки включають ініціалізацію Git у проєкті, додавання файлів до репозиторію та завантаження змін на GitHub:

— спочатку відкрийте термінал у кореневому каталозі проєкту;

— потім виконайте наступні команди, щоб ініціалізувати репозиторій Git: git init;

— далі додайте URL-адресу існуючого репозиторію GitHub як віддалений репозиторій: https://github.com/MastaBata/ScoreLine;

— додайте всі файли проєкту до індексу Git: git add . ;

— наступним кроком створіть коміт, що описує зміни в проєкті: git commit -m «Project loaded»;

— останнім кроком завантажте зміни до віддаленого репозиторію на GitHub: git push -u origin master.

4.7 Тестування додатків

Для забезпечення стабільності та надійності мобільного додатку ScoreLine ми провели комплексне тестування, що включало модульне тестування, інтеграційне тестування та функціональне тестування.

Модульне тестування охоплювало всі компоненти додатку, включаючи ViewModel, API-сервіси, обробку даних та локальні сервіси сповіщень.

Були протестовані такі методи: fetchFixtures (for:season:), fetchLineups (fixtureID:), toggleFavoriteтощо. Для цього використовувалися тестові випадки XCTest[16], що імітують відповіді API.

Інтеграційне тестування призначене для перевірки взаємодії між різними модулями: завантаження даних з API, відображення даних у вікні, взаємодія з локальними налаштуваннями, зміна теми та мови [6]. Було перевірено, що натискання на матч у списку відкриває правильну сторінку з детальною

інформацією, включаючи події матчу, склад команд та результати, а після перезапуску додатка збережені матчі залишаються у списку та матчі на які мають надходити повідомлення. Також залишається обрана тема та мова.

Функціональне тестування охоплює основні сценарії використання: перегляд розкладу матчів, додавання до обраного, отримання сповіщень, автоматичне оновлення інтерфейсу відповідно до змін системи (мова, тема, дата тощо).

Ручне тестування проводилося на реальних пристроях (iPhone 16, iPhone 16 Pro Max, iOS 15+) з різними мовами інтерфейсу та регіональними налаштуваннями, а також перевірялася адаптивність інтерфейсу до темної/світлої теми.

Таким чином, комплексне тестування підтвердило відповідність функції технічним вимогам, стабільність роботи програми та зручність використання для кінцевих користувачів.

ВИСНОВКИ

На основі завершення роботи з сертифікації кваліфікації було розроблено сучасний мобільний застосунок ScoreLine, що підтримує персоналізовані сповіщення та призначений для моніторингу розкладу футбольних матчів.

У процесі роботи було проведено аналіз конкурентних продуктів, сформульовано функціональні та нефункціональні вимоги до системи, обрано відповідні інструменти розробки та архітектурні методи.

Реалізовано масштабований застосунок, що підтримує завантаження даних через API-Football, локальне кешування, динамічне відображення подій матчів, статистичну інформацію, склад команд та результати.

Користувальницький інтерфейс побудовано відповідно до принципів UX/UI-дизайну, з підтримкою адаптивного дизайну темних та світлих тем. Особливу увагу приділено локалізації інтерфейсу та багатомовній підтримці.

Комплексне тестування підтвердило стабільність роботи програми, відповідність її функціональних можливостей технічним вимогам та зручність у використанні.

Програма має великий потенціал для подальшого розвитку, зокрема додавання нових типів повідомлень, інтеграції з календарем, розширення статистичної інформації або впровадження аналітики.

Результати роботи можуть бути використані як основа для створення комерційних або освітніх продуктів для спортивних уболівальників та користувачів, які бажають отримувати актуальну інформацію в зручному форматі.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Жилєнков О.С., Савіцький В.А. Розробка мобільного застосунку «Моніторинг розкладу спортивних подій з персоналізованими сповіщеннями» з використанням SwiftUI. Матеріали всеукраїнської науково-технічної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15.04.2025, ДУІКТ,

с. (подано до друку).

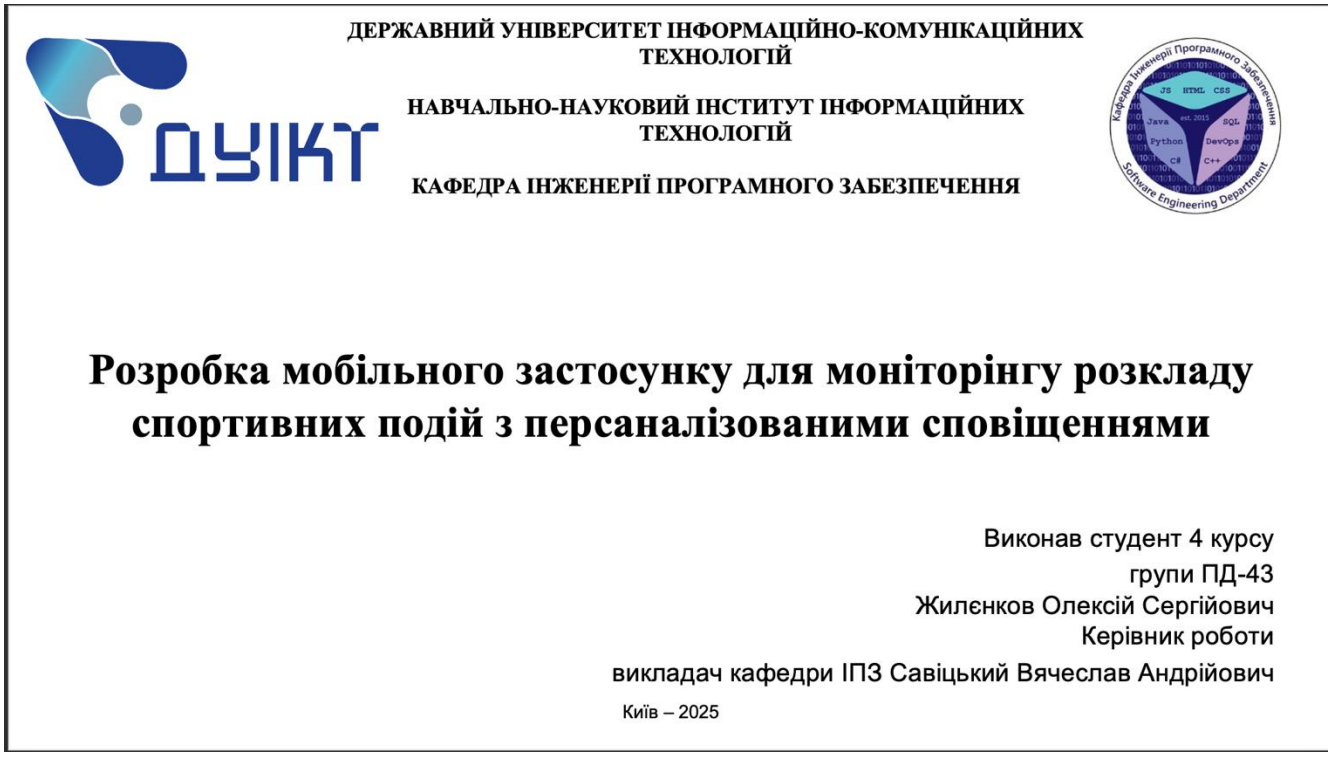
2. Жилєнков О.С., Савіцький В.А. Застосунок для моніторингу розкладу спортивних подій з персоналізованими сповіщеннями. Матеріали всеукраїнської науково-технічної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15.05.2025, ДУІКТ, с. (подано до друку).

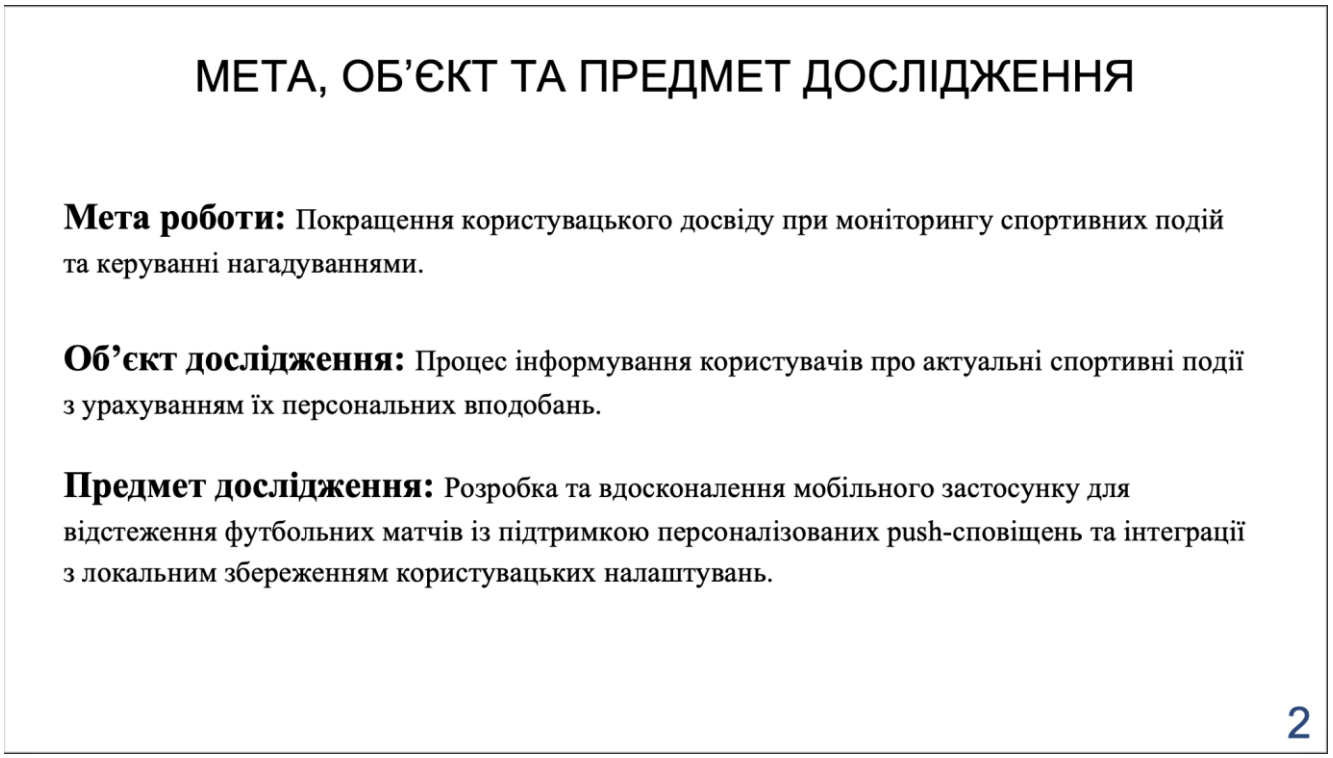
ПЕРЕЛІК ПОСИЛАНЬ

1. API-Football.URL: https://www.api-football.com/documentation-v3 (дата звернення 15.04.2025).
2. Apple Inc. "The Swift Programming Language". URL: https://docs.swift.org/swift-book/ (дата звернення 15.04.2025).
3. Apple Inc. "SwiftUI Documentation". URL: https://developer.apple.com/documentation/swiftui (дата звернення 15.04.2025).
4. Apple Inc. "Xcode 16 Release Notes". URL: https://developer.apple.com/documentation/xcode-release-notes/xcode-16-release-notes (дата звернення 15.04.2025).
5. Apple Inc. "UNUserNotificationCenter". URL: https://developer.apple.com/documentation/usernotifications/unusernotificationcenter (дата звернення 15.04.2025).
6. Apple Inc. "AppStorage Property Wrapper". URL: https://developer.apple.com/documentation/swiftui/appstorage (дата звернення 15.04.2025).
7. Apple Inc. "Combine Framework". URL: https://developer.apple.com/documentation/combine (дата звернення 15.04.2025).
8. Apple Developer. "UserDefaults". URL: https://developer.apple.com/documentation/foundation/userdefaults (дата звернення 15.04.2025).
9. Raywenderlich.com. "SwiftUI MVVM Tutorial". URL: https://www.raywenderlich.com/1000693-mvvm-with-combine-tutorial-for-ios (дата звернення 15.04.2025).
10. Apple Developer. "URLSession Documentation". URL: https://developer.apple.com/documentation/foundation/urlsession (дата звернення 15.04.2025).

11. GitHub Docs. "About GitHub". URL: https://docs.github.com/en/get-started/using-github/about-github (дата звернення 15.04.2025).
12. GitHub Docs. "GitHub for Mobile". URL: https://docs.github.com/en/get-started/using-github/github-mobile (дата звернення 15.04.2025).
13. Git SCM. "Git Documentation". URL: https://git-scm.com/doc (дата звернення 15.04.2025).
14. Atlassian. "Version Control with Git". URL: https://www.atlassian.com/git (дата звернення 15.04.2025).
15. Medium. "Understanding MVVM in iOS". URL: https://medium.com/@abhimuralidharan/mvvm-in-ios-swift-52ef499e7c4e (дата звернення 15.04.2025).
16. UX Planet. "Principles of Good Mobile App Design". URL: https://uxplanet.org/principles-of-mobile-app-design-1fc8e4d5f5c5 (дата звернення 15.04.2025).
17. Nielsen Norman Group. "Mobile UX Design". URL: https://www.nngroup.com/topic/mobile-ux/ (дата звернення 15.04.2025).
18. Firebase Docs. "Cloud Messaging". URL: https://firebase.google.com/docs/cloud-messaging (дата звернення 15.04.2025).
19. iOS Dev Tools. "Top Swift Libraries". URL: https://iosdev.tools/ (дата звернення 15.04.2025).
20. OpenAI. "Best Practices for API Integration". URL: https://platform.openai.com/docs/guides/gpt-best-practices (дата звернення 15.04.2025).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ





ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати наявні мобільні застосунки для моніторингу футбольних матчів та виявити їх основні переваги і недоліки.
2. Визначити вимоги до функціональності, структури та інтерфейсу майбутнього застосунку з урахуванням потреб цільової аудиторії.
3. Розробити архітектуру мобільного застосунку для iOS з використанням мови програмування Swift та фреймворку SwiftUI.
4. Реалізувати механізм отримання даних про футбольні матчі через зовнішні API (наприклад, API-Football).
5. Здійснити інтеграцію персональних push-сповіщень про початок матчів та появу стартових складів.
6. Реалізувати локальне збереження обраних матчів та налаштувань користувача.
7. Забезпечити синхронізацію з Apple Calendar та (опціонально) Siri Shortcuts.
8. Провести тестування працездатності та зручності використання застосунку на мобільному пристрої.
9. Оцінити результати розробки та сформулювати висновки щодо ефективності запропонованого рішення.

3

АНАЛІЗ АНАЛОГІВ

Застосунок	Розклад матчів	Live-результати	Статистика	Сповіщення	Реклама
FlashScore	на тиждень	базові	базова	про початок матчу (із затримкою)	кожен екран
SofaScore	на місяць	базові	базова	про початок матчу (із затримкою)	кожна сторінка
LiveScore	тільки список	урізані	мінімальна	про початок матчу (із затримкою)	по дві на сторінку
OneFootball	базовий розклад	базові	про початок матчу (із затримкою)	кожні 3-4 пости з новин та на кожній сторінці	середня
FotMob	календар	немає	немає	по дві на сторінку	низька
ScoreLine	окремий екран під розклад	базові	за 10/30/60 хв до початку матчу або при появі складів на матч	-	

4

ВИМОГИ ДО ДОДАТКУ

Функціональні:

1) Отримання даних про футбольні матчі.
2) Відображення інформації про матч.
3) Додавання матчів до обраного.
4) Надсилання локальних сповіщень.
5) Синхронізація з Apple Calendar (iOS).
6) Відображення попередніх результатів.
7) Підтримка персоналізованих налаштувань інтерфейсу

Нефункціональні:

1) Адаптивність під різні розміри екрану.
2) Портативність під різні моделі приладів.

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

Swift

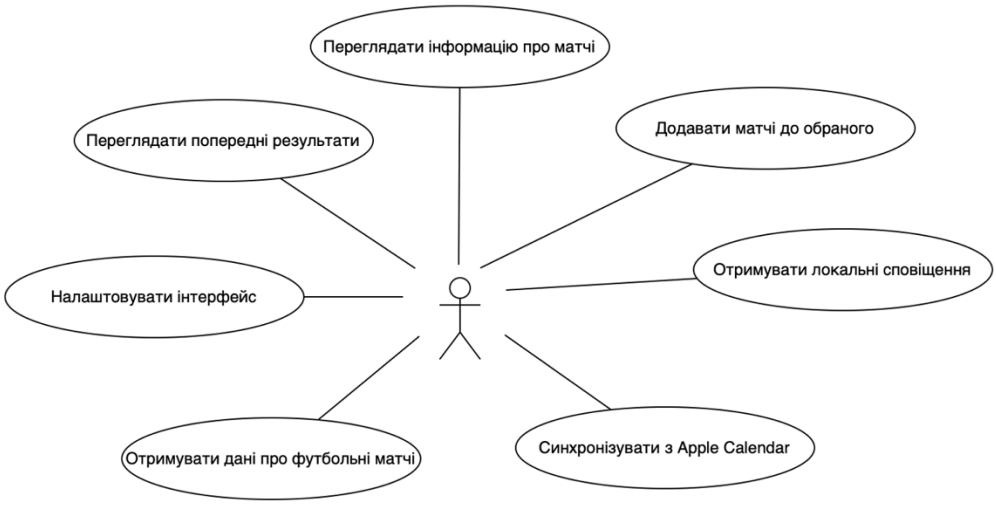
SwiftUI

Xcode

SF Symbols

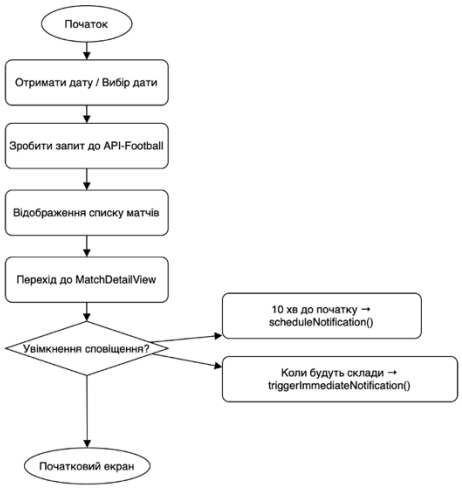
6

Діаграма варіантів використання



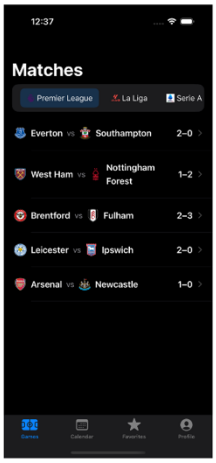
7

Блок схема процесу роботи з матчами

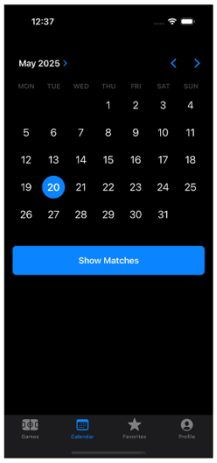


8

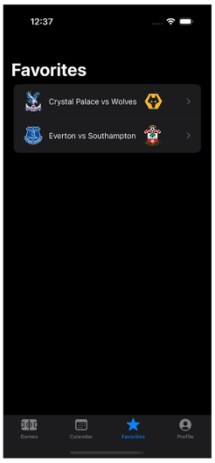
ЕКРАННІ ФОРМИ



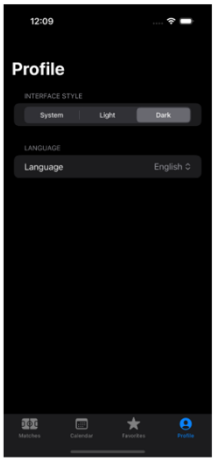
Головний екран (список матчів)



Календар



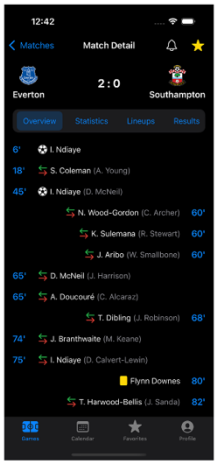
Список улюбленних матчів



Профіль (налаштування мови та інтерфейсу)

9

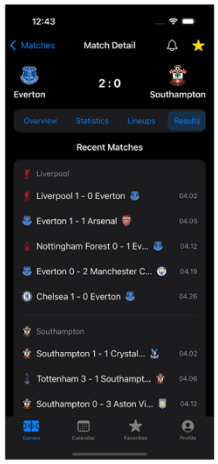
ЕКРАННІ ФОРМИ



Огляд матчу

Статистика

Склади команд



Останні матчі

Таблиця ліги

10

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Жилєнков О.С., Савіцький В.А. Розробка мобільного застосунку «Моніторинг розкладу спортивних подій з персоналізованими сповіщеннями» з використанням SwiftUI. Матеріали всеукраїнської науково-технічної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15.04.2025, ДУІКТ, с. (подано до друку).

2. Жилєнков О.С., Савіцький В.А. Застосунок для моніторингу розкладу спортивних подій з персоналізованими сповіщеннями. Матеріали всеукраїнської науково-технічної конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15.05.2025, ДУІКТ, с. (подано до друку).

11

ВИСНОВКИ

1) Проведено аналіз популярних мобільних застосунків для моніторингу футбольних матчів, що дозволило виявити ключові функціональні характеристики та недоліки, які були враховані під час розробки власного рішення.
2) На основі визначених вимог було сформульовано архітектуру застосунку, побудовану на Swift/SwiftUI із застосуванням патерну MVVM, що забезпечує розділення відповідальностей та легку масштабованість.
3) Реалізовано повноцінну інтеграцію з API-Football для отримання актуальної інформації про матчі, склади, рахунки, події та статистику — із подальшим відображенням у зручному інтерфейсі користувача.
4) Розроблено систему персональних push-сповіщень, яка підтримує два типи нагадувань: за обрану кількість хвилин до початку матчу та при появі складів. Сповіщення реалізовані через UNUserNotificationCenter з локальним збереженням налаштувань.
5) Застосунок підтримує локальне збереження обраних матчів та типів сповіщень у UserDefaults, що дозволяє користувачеві зберігати персоналізований досвід.
7) У результаті тестування було підтверджено коректну роботу всіх основних сценаріїв взаємодії користувача, а також відповідність реалізованого функціоналу поставленим вимогам і задачам дипломної роботи.

12

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Вихідний код файла ScoreLineApp

```
import SwiftUI

import UIKit



@main

struct ScoreLineApp: App {

@StateObject private var appState = AppState()

@StateObject private var gamesViewModel =
GamesViewModel()

@StateObject private var languageManager =
LanguageManager()

private let lineupNotifier = LineupNotifier()

@AppStorage("appTheme") private var appThem
e: String = "system"



init() {

let tabBarAppearance = UITabBarAppearance()

tabBarAppearance.configureWithDefaultBackgrou
nd()

tabBarAppearance.backgroundColor =
UIColor.systemGray6



UITabBar.appearance().standardAppearance =
tabBarAppearance

if #available(iOS 15.0, *) {

UITabBar.appearance().scrollEdgeAppearance =
tabBarAppearance

}

UITabBar.appearance().tintColor = nil

UITabBar.appearance().unselectedItemTintColor
= nil

UITabBar.appearance().isTranslucent = true

}



var body: some Scene {

WindowGroup {

ContentView()

.environmentObject(appState)

.environmentObject(gamesViewModel)

.environmentObject(languageManager)

.preferredColorScheme(

appTheme == "light" ? .light :

appTheme == "dark" ? .dark :

nil

)

.onAppear {

NotificationManager.shared.requestAuthorization()



let settings = NotificationStore.shared.load()

for setting in settings {

switch setting.type {
```

```
case "before10min":

if let fixture =
gamesViewModel.allFixtures.first(where:
{ $0.fixture.id == setting.fixtureId }),

let matchDate =
ISO8601DateFormatter().date(from:
fixture.fixture.date) {

let notificationDate =
matchDate.addingTimeInterval(-10 * 60)

NotificationManager.shared.scheduleNotification(

id: "match-\(fixture.fixture.id)",

title: NSLocalizedString("Match Reminder",
comment: ""),

body: String(

format: NSLocalizedString("⏰ %@ vs %@ starts in
10 minutes!", comment: ""),

fixture.teams.home.name,

fixture.teams.away.name

),

date: notificationDate

)

}

case "lineups":

if let fixture =
gamesViewModel.allFixtures.first(where:
{ $0.fixture.id == setting.fixtureId }) {

lineupNotifier.register(fixture: fixture)

}

default:

break

}
}
}
}
}
```

Вихідний код файла GamesView

```
import SwiftUI

struct GamesView: View {

@EnvironmentObject var appState: AppState

@EnvironmentObject var viewModel:
GamesViewModel

@Environment(\.colorScheme) private var colorS
cheme

@State private var selectedLeague: String = ""

@State private var fixtures: [FixtureResponse] =
[]

@State private var allFixtures: [FixtureResponse]
= []

@State private var isLoading = false

@State private var loadError: String? = nil

@State private var availableLeagues: [String] = []

private let leagueFilters: [String] = [

"Premier League", "La Liga", "Bundesliga",
```

```
"Serie A", "Ligue 1",
"UEFA Champions League", "UEFA Europa League", "UEFA Europa Conference League"
]
private let leagueIDMap: [String: Int] = [
"Premier League": 39,
"La Liga": 140,
"Bundesliga": 78,
"Serie A": 135,
"Ligue 1": 61,
"UEFA Champions League": 2,
"UEFA Europa League": 3,
"UEFA Europa Conference League": 848
]
private let invertLeagues: Set<String> = [
"UEFA Champions League",
"UEFA Europa League",
"UEFA Europa Conference League"
]

private let isoFrac: ISO8601DateFormatter = {
let f = ISO8601DateFormatter()
f.formatOptions = [.withInternetDateTime, .withFractionalSeconds]
return f
}()
private let isoNoFrac: ISO8601DateFormatter = {
let f = ISO8601DateFormatter()
f.formatOptions = [.withInternetDateTime]
return f
}()
private let fallbackFmt: DateFormatter = {
let f = DateFormatter()
f.locale = Locale(identifier: "en_US_POSIX")
f.dateFormat = "yyyy-MM-dd'T'HH:mm:ssZ"
return f
}()
private let timeFmt: DateFormatter = {
let f = DateFormatter()
f.dateFormat = "HH:mm"
f.locale = Locale.current
return f
}()
private let displayDateFmt: DateFormatter = {
let f = DateFormatter()
f.dateStyle = .medium
f.timeStyle = .none
return f
}()

var body: some View {
NavigationView {
VStack(spacing: 0) {
```

```
leagueSlider

contentView

}

.navigationTitle("Matches")

.onAppear(perform: fetchData)

.onChange(of: appState.selectedDate) { _,
_ in fetchData() }

}

}


private var leagueSlider: some View {

ScrollView(.horizontal, showsIndicators: false) {

HStack(spacing: 12) {

ForEach(availableLeagues, id: \.self) { league in

Button(action: {

selectedLeague = league

filterFixtures()

}) {

HStack(spacing: 4) {

leagueLogo(league)

Text(league)

.font(.subheadline)

.foregroundColor(.primary)

}

.padding(.vertical, 8)

.padding(.horizontal, 12)

.background(selectedLeague == league ?
Color.blue.opacity(0.2) : Color.clear)

.cornerRadius(8)

}

}

}

.padding(.horizontal)

.padding(.vertical, 8)

}

.background(Color(UIColor.secondarySystemBack
ground).cornerRadius(12))

.padding(.horizontal)

.padding(.bottom, 8)

}


@ViewBuilder

private var contentView: some View {

if isLoading {

ProgressView("Loading
\(selectedLeague)…").padding()

Spacer()

} else if let err = loadError {

Text(err)

.foregroundColor(.red)

.multilineTextAlignment(.center)

.padding()

Spacer()

} else if fixtures.isEmpty {
```

```
let d = appState.selectedDate

let msg = Calendar.current.isDateInToday(d)

? "No matches today for ‘\(selectedLeague)’"

: "No matches for ‘\(selectedLeague)’ on
\(displayDateFmt.string(from: d))"

Text(msg)

.foregroundColor(.secondary)

.padding()

Spacer()

} else {

List(fixtures) { match in

let ts = match.fixture.date

let start = isoFrac.date(from: ts)

?? isoNoFrac.date(from: ts)

?? fallbackFmt.date(from: ts)

?? Date.distantPast

NavigationLink(

destination: MatchDetailView(fixture: match)

.environmentObject(viewModel)

) {

MatchRow(match: match, startDate: start,
timeFormatter: timeFmt)

}

}

.listStyle(PlainListStyle())

.listRowInsets(.init())

.listRowSeparator(.visible)

.listRowSeparatorTint(.gray)

}

}


@ViewBuilder

private func leagueLogo(_ league: String)
-> some View {

if let id = leagueIDMap[league],

let url = URL(string: "https://media.api-
sports.io/football/leagues/\(id).png") {

AsyncImage(url: url) { phase in

if let image = phase.image {

let icon = image

.resizable()

.aspectRatio(contentMode: .fit)

.frame(width: 16, height: 16)

if invertLeagues.contains(league) && colorScheme
== .dark {

icon.colorInvert()

} else {

icon

}

} else {

Color.clear.frame(width: 16, height: 16)

}

}

}

}
```

```
private func fetchData() {
isLoading = true
loadError = nil
allFixtures = []
fixtures = []
let fmt = DateFormatter()
fmt.dateFormat = "yyyy-MM-dd"
let dateStr = fmt.string(from:
appState.selectedDate)

NetworkService.shared.fetchFixtures(date: dateStr,
league: nil) { result in
DispatchQueue.main.async {
switch result {
case .success(let list):
self.allFixtures = list
viewModel.allFixtures = list
self.availableLeagues = leagueFilters.filter
{ name in
guard let lid =
leagueIDMap[name] else { return false }
return list.contains { $0.league?.id == lid }
}
if availableLeagues.isEmpty {
availableLeagues = leagueFilters
}
if !availableLeagues.contains(selectedLeague) {
selectedLeague = availableLeagues.first ?? ""
}
filterFixtures()
case .failure(let error):
loadError = "Failed to load:
\(error.localizedDescription)"
}
isLoading = false
}
}
}

private func filterFixtures() {
guard let lid =
leagueIDMap[selectedLeague] else {
fixtures = allFixtures
return
}
fixtures = allFixtures.filter { $0.league?.id == lid }
}
}

private struct MatchRow: View {
let match: FixtureResponse
let startDate: Date
let timeFormatter: DateFormatter
```

```
var body: some View {
HStack(spacing: 8) {
teamView(match.teams.home)
Text("vs").font(.subheadline).foregroundColor(.secondary)
teamView(match.teams.away)
Spacer()
infoView
}
.padding(.vertical, 8)
}

@ViewBuilder
private func teamView(_ team: TeamInfo) -> some View {
if let url = URL(string: team.logo) {
AsyncImage(url: url) { phase in
if let image = phase.image {
image
.resizable()
.aspectRatio(contentMode: .fit)
.frame(width: 24, height: 24)
} else {
Color.gray.opacity(0.3)
.frame(width: 24, height: 24)
}
}
Text(team.name).font(.headline)
}

@ViewBuilder
private var infoView: some View {
if let h = match.goals.home, let a = match.goals.away {
VStack(alignment: .trailing, spacing: 2) {
Text("\(h)–\(a)").font(.headline)
switch match.fixture.status.short {
case "1H", "2H", "ET":
if let elapsed = match.fixture.status.elapsed {
Text("\(elapsed)'")
.font(.subheadline)
.foregroundColor(.gray)
} else {
Text("Live")
.font(.subheadline)
.foregroundColor(.gray)
}
case "HT", "B":
Text("Half Time")
.font(.subheadline)
.foregroundColor(.gray)
default:
EmptyView()
```

```
}

}

} else {

Text(timeFormatter.string(from: startDate))

.font(.subheadline)

.foregroundColor(.gray)

}

}

}

struct GamesView_Previews: PreviewProvider {

static var previews: some View {

GamesView()

.environmentObject(AppState())

.environmentObject(GamesViewModel())

}

}
```