Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка криптовалютного додатку «The Futures» з використанням мови Kotlin

Виконав: студент IV курсу, групи СП-41
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Жупило О.Б (прізвище та ініціали)

Керівник (підпис) Яворська Є.Б (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« » 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)
за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
студенту Жупило Олександра Борисівна

1. Тема роботи Розробка криптовалютного додатку «The Futures» з використанням мови Kotlin

Керівник роботи к.т.н., доцент, завідувач кафедри БТ Яворська Є.Б
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
Затверджені наказом по університету від « » 2026 року №_____
2. Термін подання студентом роботи
3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області та постановка задачі.
2. Проєктування та розробка програмної системи.
3. Тестування та оцінка якості.
4. Безпека життєдіяльності, основи охорони праці.
Висновки
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Існуючі технології реалізації подібних систем.
2. Діаграма варіантів використання системи (UML Use Case).
3. Навігаційна структура додатку. 4. Загальна архітектура системи.
5. UML-діаграма класів та UML-діаграма послідовності.
6. Результати тестування авторизації та бази даних.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи		
2	Аналіз предметної області, огляд аналогів та формування вимог до додатку		
3	Проєктування архітектури додатку, моделей даних та навігаційної структури		
4	Розробка мережевого шару та інтеграція з CryptoCompare API		
5	Реалізація авторизації користувача та синхронізації даних через Firebase		
6	Реалізація основних екранів додатку (Home, AllCoins, CoinDetail, Settings)		
7	Тестування функціональності додатку та усунення помилок		
8	Написання розділу 4: «Охорона праці та безпека життєдіяльності»		
9	Оформлення висновків, списку використаних джерел та додатків		
10	Нормоконтроль, попередній захист та захист кваліфікаційної роботи		
11	Перевірка на плагіат		
12	Попередній захист кваліфікаційної роботи		
13	Захист кваліфікаційної роботи		

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Жупило Олександра Борисівна. Розробка криптовалютного додатку «The Futures» з використанням мови Kotlin: // Кваліфікаційна робота освітнього рівня «бакалавр» // Жупило Олександра Борисівна // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-41 // Тернопіль, 2026 // С. 76, рис. – 22, табл. – 0, кресл. – 4, додат. – 2, бібліогр. – 20.

Ключові слова: Kotlin, Android, Jetpack Compose, Cryptocurrency, Firebase, Retrofit, Room, Mvvm, Mobile Application.

Кваліфікаційн робота бакалавра присвячена проєктуванню та розробці мобільного криптовалютного додатку The Futures з використанням мови програмування Kotlin та сучасного інструментарію Android-розробки.

У першому розділі розглянуто предметну область моніторингу криптовалютного ринку, проаналізовано існуючі рішення, визначено вимоги до застосунку та обґрунтовано вибір технологічного стеку.

У другому розділі описано архітектуру системи на основі MVVM, моделі даних та навігацію між екранами. Обґрунтовано використання Jetpack Compose, Kotlin Coroutines, Retrofit, Firebase та Room.

У третьому розділі описано реалізацію основних модулів: відображення курсів, графіків динаміки ціни, списку відстеження та зміни теми оформлення. Проведено тестування функціональних компонентів.

У четвертому розділі розглянуто питання надійності, захисту даних користувача та охорони праці при розробці програмного забезпечення.

Об'єктом дослідження є процес розробки мобільного застосунку для моніторингу криптовалютного ринку на платформі Android.

Предметом дослідження є методи, моделі та технології розробки криптовалютного додатку з використанням Kotlin, Jetpack Compose, архітектури MVVM та хмарних сервісів Firebase.

ABSTRACT

Zhupylo Oleksandra. Development of the cryptocurrency application "The Futures" using the Kotlin language: // Qualification work of the educational level "bachelor" // Zhupylo Oleksandra // Ivan Pulyuy Ternopil National Technical University, Faculty of Computer and Information Systems and Software Engineering, Department of Software Engineering, Group SP-41 // Ternopil, 2026 // P. 76, fig. – 22, tab. – 0, fig. – 4, append. – 2, bibliography – 20.

Keywords: Kotlin, Android, Jetpack Compose, Cryptocurrency, Firebase, Retrofit, Room, Mvvm, Mobile Application.

The bachelor's qualification work is devoted to the design and development of the mobile cryptocurrency application The Futures using the Kotlin programming language and modern Android development tools.

The first section considers the subject area of cryptocurrency market monitoring, analyzes existing solutions, determines the requirements for the application and justifies the choice of the technological stack.

The second section describes the system architecture based on MVVM, data models and navigation between screens. The use of Jetpack Compose, Kotlin Coroutines, Retrofit, Firebase and Room is justified.

The third section describes the implementation of the main modules: displaying rates, price dynamics charts, tracking list and changing the design theme. Functional components were tested.

The fourth section considers the issues of reliability, user data protection and occupational safety in software development.

The object of the study is the process of developing a mobile application for monitoring the cryptocurrency market on the Android platform.

The subject of the study is the methods, models and technologies for developing a cryptocurrency application using Kotlin, Jetpack Compose, MVVM architecture and Firebase cloud services.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – Application Programming Interface – програмний інтерфейс для обміну даними між системами.

MVVM – Model-View-ViewModel – архітектурний підхід, що відокремлює дані, логіку стану та інтерфейс.

UI – User Interface – користувацький інтерфейс.

DAO – Data Access Object – об'єкт доступу до даних у локальній базі Room.

DTO – Data Transfer Object – модель передавання даних між API та програмою.

Flow – асинхронний потік даних у Kotlin Coroutines.

Firestore – хмарна NoSQL-база даних Firebase.

Watchlist – список криптовалют, які користувач додав для постійного відстеження.

ВСТУП

Стрімкий розвиток ринку криптовалют зумовлює потребу в інструментах для швидкого отримання та аналізу актуальної інформації. Оскільки торгівля цифровими активами здійснюється цілодобово, користувачам необхідний зручний мобільний доступ до даних про вартість криптовалют, зміни курсу та ринкові показники. Актуальність теми полягає у розробці мобільного застосунку для моніторингу криптовалютного ринку, який забезпечує швидкий доступ до інформації, зручний інтерфейс та персоналізацію даних. Для реалізації такого рішення використано мову програмування Kotlin [1] та технологію Jetpack Compose [2].

Метою кваліфікаційної роботи є розробка криптовалютного додатку The Futures**,** що дозволяє користувачам переглядати список криптовалют, отримувати детальну інформацію про активи, формувати список відстеження, переглядати історію нещодавніх запитів, змінювати тему оформлення та здійснювати авторизацію.

Для досягнення поставленої мети необхідно виконати такі завдання:

1 Проаналізувати предметну область криптовалютного ринку та розглянути існуючі аналогічні мобільні рішення.
2 Визначити функціональні та нефункціональні вимоги до розроблюваного застосунку.
3 Обрати та обґрунтувати технологічний стек для реалізації проєкту.
4 Спроєктувати архітектуру програмної системи, визначити класи та зв'язки між ними.
5 Реалізувати основні програмні модулі додатку: мережевий шар, авторизацію, локальне сховище та екрани інтерфейсу.
6 Провести функціональне тестування розробленого застосунку та оцінити якість програмного продукту.

Об'єктом дослідження є процес розробки мобільних додатків для моніторингу криптовалютного ринку. Предметом дослідження є методи та засоби створення Android-застосунків із використанням Kotlin, Jetpack Compose, Firebase, Retrofit, Room та архітектури MVVM.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків. У першому розділі виконано аналіз предметної області, огляд існуючих рішень та сформовано вимоги до системи. Другий розділ присвячено проєктуванню програмної системи: вибору технологій, побудові архітектури, визначенню класів та навігаційної структури. У третьому розділі описано реалізацію основних модулів додатку. Четвертий розділ містить результати тестування, оцінку якості програмного продукту та питання охорони праці. метод варіантів використання для формалізації вимог, архітектурний патерн MVVM для організації коду, а також функціональне тестування за методом чорної скриньки для перевірки відповідності реалізованого функціоналу встановленим вимогам.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

Криптовалютний ринок є частиною цифрової економіки, у межах якої фінансові активи існують у вигляді токенів і монет, а інформація про їхню ціну формується на основі попиту та пропозиції на біржах [12]. На відміну від традиційних фондових ринків, криптовалютні торги відбуваються цілодобово, що підсилює потребу в мобільних інструментах постійного моніторингу [1].

Користувачі криптовалютних додатків зазвичай очікують швидкий перегляд основних показників: поточної ціни, добової зміни, обсягу торгів, ринкової капіталізації, кількості монет в обігу та динаміки за вибраний період. Важливим є також зручне формування персонального списку активів, які користувач хоче відстежувати частіше за інші.

Для якісного користувацького досвіду мобільний додаток має бути адаптованим до невеликого екрана, мати зрозумілу нижню навігацію [3], швидке завантаження даних, індикатори станів, обробку помилок та підтримку світлої і темної теми[13]. Такі вимоги впливають не лише на дизайн, а й на вибір архітектури та технологій.

1.2 Огляд існуючих рішень

Сучасний ринок мобільних застосунків для моніторингу криптовалют пропонує значну кількість рішень, які дозволяють користувачам відстежувати зміни курсів цифрових активів, аналізувати ринкові тенденції та отримувати актуальну інформацію про криптовалютний ринок [12]. Під час розробки застосунку «The Futures» було проаналізовано існуючі програмні продукти, що мають подібне функціональне призначення.

Одним із найпопулярніших рішень є мобільний застосунок CoinMarketCap (рис. 1.1). Даний сервіс надає доступ до великої бази

криптовалют, дозволяє переглядати поточні ціни, ринкову капіталізацію, торгові обсяги та історичні графіки зміни вартості активів. Також користувачі можуть створювати власні портфелі та списки спостереження. Основною перевагою застосунку є висока інформативність та велика кількість доступних даних. Водночас інтерфейс містить значну кількість функцій і розділів, що може ускладнювати використання для початківців.

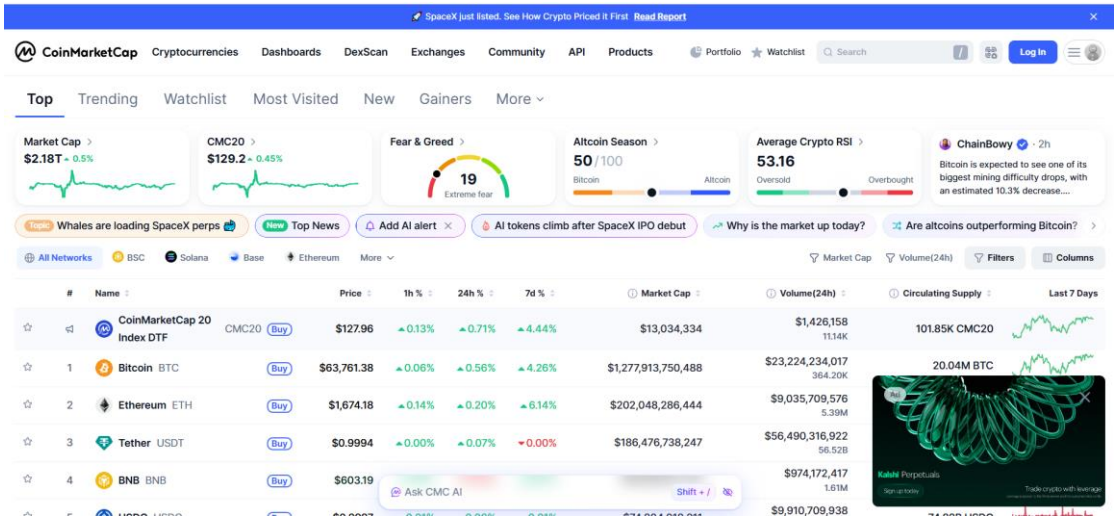

Рисунок 1.1 – Головний екран додатка CoinMarketCap

Іншим популярним рішенням є Glassnode (рис. 1.2). Застосунок забезпечує глибокий ончейн-аналіз та моніторинг ринку цифрових активів на основі даних безпосередньо з блокчейнів. Користувачам доступні складні аналітичні показники, індикатори поведінки інвесторів (зокрема, довгострокових та короткострокових утримувачів) та детальні ринкові звіти. Перевагою Glassnode є унікальний набір професійних інструментів для оцінки фундаментального стану мереж, однак специфічність графіків та висока вартість розширених тарифних планів можуть ускладнити щоденне

використання платформи звичайними користувачами та початківцями.

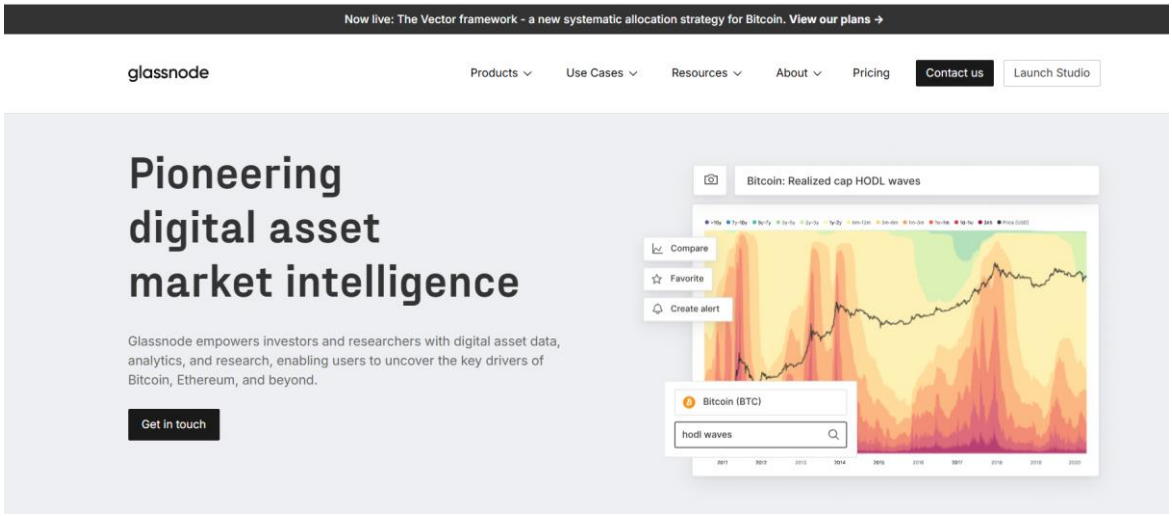

Рисунок 1.2 − Головний екран додатка Glassnode

На відміну від розглянутих аналогів, розроблюваний застосунок «The Futures» орієнтований на швидкий та зручний перегляд основної інформації про криптовалютний ринок без надлишкових функцій. Основними можливостями застосунку є авторизація користувачів [6], перегляд списку криптовалют, відображення детальної інформації про обраний актив, побудова графіків зміни ціни, формування списку обраних монет (Watchlist), перегляд нещодавно відкритих активів (Recent) та налаштування користувацького профілю.

Перевагами застосунку «The Futures» є простий та інтуїтивно зрозумілий інтерфейс, оптимізований для мобільних пристроїв, швидкий доступ до актуальної інформації про криптовалюти, відсутність біржових операцій і зайвих елементів, а також використання сучасних технологій розробки Android-застосунків на мові програмування Kotlin. Завдяки цьому користувач може швидко отримувати необхідні дані про стан криптовалютного ринку та ефективно відстежувати обрані цифрові активи.

1.3 Постановка задачі

Метою кваліфікаційної роботи є розробка мобільного Android-застосунку «The Futures», призначеного для моніторингу криптовалютного ринку,

перегляду актуальних фінансових показників цифрових активів та персоналізованого відстеження обраних криптовалют.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– Провести аналіз предметної області та дослідити сучасні криптовалютні сервіси для визначення їх функціональних можливостей, переваг і недоліків.

– Сформувати функціональні та нефункціональні вимоги до програмного продукту з урахуванням потреб потенційних користувачів.

– Спроєктувати архітектуру мобільного застосунку із використанням сучасних підходів до Android-розробки, забезпечивши розподіл логіки між шарами представлення, бізнес-логіки та доступу до даних.

– Розробити систему реєстрації та авторизації користувачів із можливістю створення облікового запису, входу в систему та відновлення пароля.

– Реалізувати інтеграцію із зовнішнім API для отримання актуальних даних про криптовалюти, включаючи ціни, ринкову капіталізацію, обсяг торгів та інші показники.

– Створити екран перегляду списку криптовалют із можливістю відображення основної ринкової інформації та швидкого доступу до детальних даних про актив.

– Реалізувати екран детальної інформації про криптовалюту, який містить графік зміни ціни, статистичні показники та додаткову інформацію про обраний цифровий актив [2].

– Розробити функціонал формування списку обраних криптовалют (Watchlist) для персонального відстеження активів користувачем.

– Реалізувати механізм збереження та відображення нещодавно переглянутих криптовалют (Recent), що дозволить швидко повертатися до раніше відкритих активів.

– Забезпечити зручну навігацію між основними екранами застосунку за допомогою нижньої панелі навігації та організувати логічну структуру інтерфейсу.

– Реалізувати систему налаштувань користувача, включаючи зміну теми оформлення застосунку та керування параметрами профілю.

– Забезпечити коректне форматування фінансових даних для відображення цін, відсоткових змін, обсягів торгів та ринкової капіталізації.

– Провести тестування працездатності застосунку, перевірити коректність взаємодії з серверними сервісами та оцінити стабільність роботи інтерфейсу.

– Підготувати програмний продукт до подальшого розширення функціональних можливостей шляхом використання масштабованої архітектури та сучасних засобів розробки на мові програмування Kotlin.

Результатом виконання кваліфікаційної роботи повинен стати повнофункціональний мобільний застосунок «The Futures», який забезпечує швидкий доступ до актуальної інформації про криптовалютний ринок, підтримує персоналізацію даних користувача та відповідає сучасним вимогам до мобільних програмних продуктів.

1.4 Аналіз користувачів та варіантів використання

Для формалізації функціональних вимог до додатку The Futures застосовано метод аналізу варіантів використання Use Case Analysis, який дозволяє розглянути систему з точки зору її користувачів та визначити перелік дій, які кожен користувач може виконати. У межах розробленого додатку виокремлено два основні актори: Незареєстрований користувач (гість) та Зареєстрований користувач. Окремо також розглядається зовнішній актор Віддалений сервіс (CryptoCompare API та Firebase), який не є людиною-користувачем, але бере участь у сценаріях отримання та збереження даних.

Актор Незареєстрований користувач — це особа, яка вперше відкриває додаток або ще не пройшла авторизацію. Його головна мета полягає в тому, щоб ознайомитися з функціоналом додатку та отримати доступ до перегляду ринку криптовалют, а також зареєструватися або увійти в систему для збереження персональних налаштувань.

До варіантів використання для цього актора належать:

– Запуск додатку та очікування завантаження початкових даних (екран Loading);
– Перегляд екрана авторизації;
– Реєстрація нового облікового запису за допомогою електронної пошти та пароля;
– Вхід в існуючий обліковий запис;
– Відновлення доступу у разі помилки введення даних (повторна спроба авторизації).

Актор Зареєстрований користувач — це користувач, який успішно пройшов автентифікацію через Firebase Authentication. Його мета полягає в постійному моніторингу курсів криптовалют, формуванні персонального списку відстеження та налаштуванні зовнішнього вигляду додатку відповідно до власних уподобань.

До варіантів використання для цього актора належать:
– Перегляд головного екрана зі списком відстеження (Watchlist) та нещодавно переглянутих монет (Recent);
– Перегляд переліку всіх доступних криптовалют з пошуком і сортуванням;
– Додавання обраної криптовалюти до списку відстеження та видалення з нього;
– Перегляд детальної інформації про криптовалюту (поточна ціна, зміна за період, графік історичних значень);
– Перемикання періоду відображення графіка (наприклад, день, тиждень, місяць);
– Перехід до екрана налаштувань;
– Перемикання теми оформлення додатку (світла/темна);
– Вихід з облікового запису (logout).

Актор Віддалений сервіс об'єднує зовнішні системи, з якими взаємодіє додаток: CryptoCompare API надає актуальні дані про курси криптовалют, історичні ціни та перелік монет, а Firebase Authentication і Cloud Firestore забезпечують автентифікацію користувача та збереження його персональних списків (watchlist, recent) у хмарі. Цей актор не має власного інтерфейсу, однак його коректна робота є необхідною умовою функціонування переважної більшості сценаріїв.

Для наочного представлення взаємодії визначених акторів із системою побудовано діаграму варіантів використання (рис.1.3), яка демонструє повний перелік сценаріїв роботи з додатком The Futures та їхній розподіл між акторами.

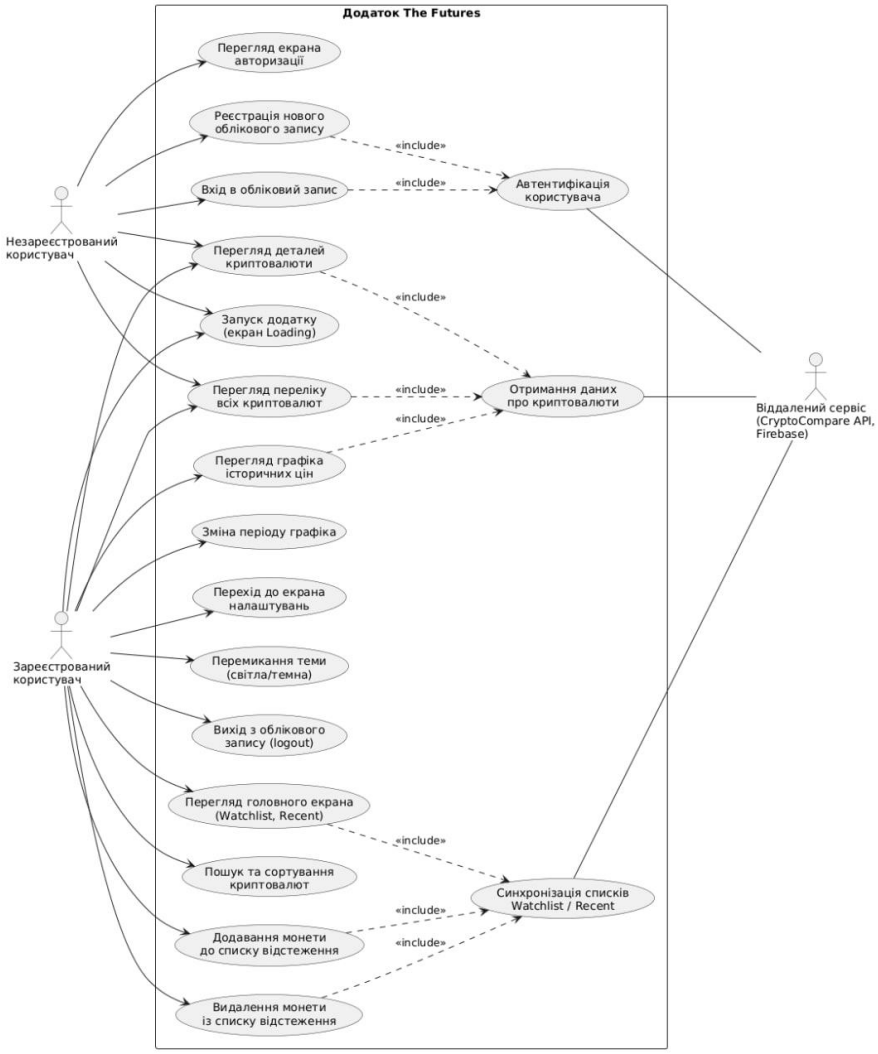

Рисунок 1.3 − Діаграма варіантів використання додатку The Futures

Наведена діаграма ілюструє, що більшість сценаріїв доступні лише авторизованому користувачу, тоді як перегляд ринкових даних може здійснюватися без авторизації, що відповідає прийнятій практиці побудови фінансових мобільних додатків — надати користувачу можливість ознайомитися з продуктом до моменту реєстрації

1.5 Опис ключових варіантів використання

Для більш детального розуміння функціонування додатку наведемо специфікації двох найважливіших прецедентів використання.

Специфікація варіанта використання «Додавання криптовалюти до списку відстеження»

Актор: Зареєстрований користувач.

Ціль: додати обрану криптовалюту до персонального списку відстеження (Watchlist) для подальшого швидкого доступу до неї з головного екрана.

Передумови: користувач авторизований у системі; додаток отримав перелік доступних криптовалют з CryptoCompare API.

Основний потік подій:

- Користувач переходить на екран переліку всіх криптовалют (AllCoinsListScreen) або на екран детальної інформації про монету (CoinDetailScreen).
- Користувач натискає кнопку (іконку) додавання до списку відстеження біля обраної криптовалюти.
- WatchlistViewModel формує об'єкт WatchlistCoinEntity на основі даних про монету та передає його до репозиторію.
- Репозиторій зберігає запис у локальній базі даних Room та синхронізує його зі структурою users/{uid}/watchlist у Cloud Firestore.
- Система оновлює стан інтерфейсу: іконка змінює вигляд (заповнена/незаповнена), а монета з'являється у відповідному переліку на головному екрані.

Альтернативний потік: якщо монета вже присутня у списку відстеження, повторне натискання кнопки призводить до її видалення зі списку (аналогічна послідовність дій, але з видаленням запису з Room та Firestore).

Специфікація варіанта використання «Перегляд деталей криптовалюти та історії цін»

Актор: Зареєстрований користувач (сценарій доступний також гостю в обмеженому вигляді).

Ціль: отримати детальну інформацію про обрану криптовалюту, зокрема поточну ціну, відсоткову зміну та графік історичних значень за вибраний період.

Передумови: додаток має з'єднання з мережею Інтернет; обрано конкретну криптовалюту зі списку.

Основний потік подій:

- Користувач обирає криптовалюту на головному екрані або на екрані переліку всіх монет.
- Навігаційний граф передає символ обраної монети до маршруту CoinDetail.
- CoinDetailViewModel надсилає запит до CryptoRepositoryImpl для отримання детальної інформації (getCoinDetail) та історичних даних про ціну (getPriceHistory).
- Репозиторій виконує мережеві запити через Retrofit, перетворює отримані DTO-моделі у доменні моделі CoinDetail і PricePoint та повертає результат у вигляді Result.
- Екран відображає назву монети, її іконку (через Coil AsyncImage), поточну ціну, відсоткову зміну та графік історичних значень ціни.
- Користувач може змінити період відображення графіка, що призводить до повторного виконання кроків 3–5 із новим параметром періоду.

Альтернативний потік: у разі відсутності мережевого з'єднання або помилки на стороні сервера екран переходить у стан Error із відображенням відповідного повідомлення та можливістю повторити запит.

1.6 Специфікація вимог до програмного забезпечення

Під час розробки мобільного застосунку «The Futures» було сформовано функціональні та нефункціональні вимоги, які визначають основні можливості системи та критерії її якості.

Функціональні вимоги

- Реєстрація та авторизація користувачів:
  Можливість створення нового облікового запису за допомогою електронної пошти та пароля.
  Авторизація зареєстрованих користувачів.
  Відновлення пароля через електронну пошту.
  Збереження даних користувача між сеансами роботи.

1. Перегляд криптовалют:
   – Отримання актуального списку криптовалют із зовнішнього API.
   – Відображення назви монети, символу, поточної ціни та відсоткової зміни вартості.
   – Сортування та оновлення даних у режимі реального часу.
   – Перегляд рейтингу криптовалют за ринковою капіталізацією.

2. Перегляд детальної інформації про криптовалюту:
   – Відображення детальної інформації про вибрану монету.
   – Перегляд поточної ціни, ринкової капіталізації та обсягу торгів.
   – Відображення історичних даних про зміну вартості активу.
   – Перехід між різними часовими інтервалами графіка.

3. Робота зі списком Watchlist:
   – Додавання криптовалют до списку обраних.
   – Видалення криптовалют зі списку обраних.

– Перегляд персонального списку відстеження.

– Автоматична синхронізація списку з обліковим записом користувача.

4. Перегляд нещодавно відкритих активів:

   – Збереження історії переглянутих криптовалют.

   – Відображення списку останніх відкритих монет.

   – Швидкий перехід до детальної інформації про раніше переглянуті активи.

5. Навігація та налаштування:

   – Перемикання між основними розділами за допомогою нижньої панелі навігації.

   – Зміна теми оформлення застосунку (світла або темна).

   – Збереження налаштувань користувача.

   – Перегляд інформації про профіль користувача.

Нефункціональні вимоги:

1. Продуктивність:

   – Час завантаження основних екранів не повинен перевищувати 2 секунд за стабільного інтернет-з'єднання.

   – Інтерфейс повинен залишатися плавним і швидким під час виконання мережевих запитів.

   – Застосунок повинен ефективно використовувати ресурси мобільного пристрою.

2. Надійність:

   – Стабільна робота застосунку без критичних помилок та аварійного завершення.

   – Коректна обробка помилок мережевого з'єднання.

   – Можливість повторного завантаження даних у разі втрати зв'язку з сервером.

3. Безпека:
   – Використання захищених протоколів передачі даних (HTTPS).
   – Безпечне зберігання облікових даних користувача.
   – Захист від несанкціонованого доступу до персональної інформації.
   – Контроль доступу до функцій, що потребують авторизації.

4. Придатність до використання:
   – Інтуїтивно зрозумілий користувацький інтерфейс.
   – Дотримання єдиного стилю оформлення на всіх екранах.
   – Зручна навігація між розділами застосунку.
   – Адаптація інтерфейсу до різних розмірів екранів Android-пристроїв.

5. Масштабованість:
   – Можливість додавання нових функцій без суттєвої зміни архітектури застосунку.
   – Підтримка інтеграції з додатковими джерелами криптовалютних даних.
   – Можливість розширення функціоналу користувацького профілю.

6. Сумісність:
   – Підтримка актуальних версій операційної системи Android.
   – Коректна робота на смартфонах і планшетах різних виробників.
   – Підтримка сучасних бібліотек та компонентів Android SDK.

1.1 Окремою вимогою до застосунку є коректне відображення фінансових показників. Для форматування цін, відсоткових змін, обсягів торгів та ринкової капіталізації використовуються спеціальні механізми обробки числових даних, що забезпечують єдиний формат відображення інформації на всіх екранах системи.

У першому розділі виконано комплексний аналіз предметної області криптовалютного ринку та визначено вихідні умови для проєктування і

розробки мобільного застосунку The Futures. Встановлено, що ринок цифрових активів є динамічним середовищем із цілодобовим режимом торгівлі, що обумовлює підвищені вимоги до оперативності, доступності та зручності інструментів для моніторингу.

Проведено огляд та порівняльний аналіз існуючих криптовалютних сервісів — CoinMarketCap, CoinGecko, Binance та Trust Wallet. За результатами аналізу (Таблиця 1.1) з'ясовано, що наявні рішення переважно орієнтовані на досвідчених учасників ринку та характеризуються перевантаженим інтерфейсом, широким функціоналом, частина якого є надлишковою для пересічного користувача, а також відсутністю мінімалістичного підходу до подачі інформації. Це підтверджує доцільність розробки власного рішення з акцентом на простоту, персоналізацію та швидкий доступ до ключових ринкових показників.

На основі проведеного аналізу сформовано перелік функціональних та нефункціональних вимог до додатку The Futures. До функціональних вимог віднесено: авторизацію користувача, перегляд переліку криптовалют, формування персонального списку відстеження, перегляд детальної інформації та графіка історичних цін, збереження нещодавніх переглядів, перемикання теми оформлення та вихід з облікового запису. Нефункціональні вимоги охоплюють стабільність роботи за умов нестабільного з'єднання, коректне відображення числових фінансових показників, безпечну авторизацію та швидкодію інтерфейсу.

Виконано аналіз користувачів системи та варіантів використання. Визначено двох основних акторів — незареєстрованого та зареєстрованого користувача — і описано повний перелік сценаріїв взаємодії кожного з них із системою. Побудовано діаграму варіантів використання (Рисунок 1.2), яка наочно ілюструє межі системи та розподіл функціоналу між акторами. Сформульовано специфікації двох ключових прецедентів: «Додавання криптовалюти до списку відстеження» та «Перегляд деталей криптовалюти та історії цін».

Результати першого розділу є вихідною базою для проєктування програмної системи. Сформульована постановка задачі, визначені вимоги та виявлені недоліки існуючих рішень безпосередньо визначають архітектурні та технологічні рішення, які детально розглядаються у другому розділі.

2 РОЗДІЛ ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 Вибір технологій розробки

Для реалізації додатку обрано мову Kotlin, яка є основною мовою сучасної Android-розробки. Вона підтримує null-safety, корутини, лаконічний синтаксис і добре інтегрується з бібліотеками Android Jetpack.

Інтерфейс користувача побудовано за допомогою Jetpack Compose. Декларативний підхід дозволяє описувати UI як функцію від стану, що добре поєднується з MVVM і реактивними потоками даних. У проєкті використовуються Compose Material3, Navigation Compose, власні компоненти, тема зі світлим і темним режимами та шрифт Poppins.

Для мережевої взаємодії використано Retrofit 3.0.0, OkHttp Logging Interceptor і kotlinx.serialization. Базовим джерелом даних є CryptoCompare API, з якого додаток отримує список популярних монет та історичні значення цін.

Для авторизації застосовано Firebase Authentication, а для синхронізації користувацьких списків - Cloud Firestore. Локальне збереження сутностей recent і watchlist передбачено через Room, що дає змогу підтримувати локальний шар даних і DAO-інтерфейси.

Dependency Injection реалізовано через Koin. Це спрощує створення ViewModel, репозиторіїв, Retrofit, бази даних і Firebase-залежностей, зменшує зв'язаність класів та полегшує тестування.

1.2 Проєктування архітектури додатку

Вибір правильної архітектурної моделі є одним із найважливіших рішень на етапі проєктування мобільного застосунку, оскільки він безпосередньо визначає зручність підтримки коду, можливість масштабування, тестованість компонентів та стійкість системи до змін вимог. Для розробки додатку The Futures обрано архітектурний патерн MVVM (Model-View-ViewModel), який є офіційно рекомендованим Google підходом для Android-застосунків та органічно інтегрується з реактивними засобами Kotlin — StateFlow, Flow та Coroutines. У поєднанні з принципами чистої архітектури (Clean Architecture) це

дозволяє побудувати систему з чітко розмежованою відповідальністю кожного компонента.

Програмна система додатку The Futures поділяється на три основні шари: Presentation, Domain та Data. Такий поділ забезпечує односпрямованість залежностей — кожен шар залежить лише від шарів нижчого рівня, але не навпаки, що унеможливлює циклічні залежності та спрощує заміну окремих компонентів.

Шар Presentation відповідає за відображення інтерфейсу користувача та обробку його дій. Він складається з Compose-екранів (LoadingScreen, AuthenticationScreen, HomeScreen, AllCurrenciesListScreen, CoinDetailScreen, SettingsScreen) та відповідних ViewModel-класів. Екрани не містять жодної бізнес-логіки — вони лише підписуються на StateFlow зі стану ViewModel та делегують обробку подій (натискання кнопок, введення тексту) відповідним методам ViewModel. Такий підхід відповідає принципу єдиної відповідальності і робить екрани максимально простими та передбачуваними.

Шар Domain є центральним та незалежним від будь-яких зовнішніх фреймворків і бібліотек. Він містить доменні моделі (CoinListItem, AllCoinsListItem, CoinDetail, PricePoint) та інтерфейси репозиторіїв (CryptoRepository, AuthRepository, WatchlistRepository, RecentRepository, ThemePreferenceRepository), що визначають контракти доступу до даних. Оскільки доменний шар не знає нічого про Retrofit, Room чи Firebase, він може бути протестований у повній ізоляції від конкретних технологій.

Шар Data відповідає за реалізацію контрактів, визначених у Domain. Він включає конкретні реалізації репозиторіїв (CryptoRepositoryImpl, FirebaseAuthRepository, FirestoreWatchlistRepository, FirestoreRecentRepository, SharedPrefsThemePreferenceRepository), мережевий підшар (CryptoApi, DTO-моделі, NetworkModule), підшар локального сховища (MainDb, RecentCoinEntity, WatchlistCoinEntity, DAO-інтерфейси) та mapper-функції для перетворення між DTO-, Entity- та доменними моделями.

Структура пакетів проєкту відображає шарову архітектуру та організована таким чином, щоб кожен пакет мав чітко визначену зону відповідальності. Пакет data містить підпакети remote (CryptoApi та DTO-моделі), local (Room-сутності, DAO та MainDb), repository (реалізації репозиторіїв) та mapper (функції перетворення моделей). Пакет domain містить підпакети model (доменні моделі) та repository (інтерфейси репозиторіїв). Пакет presentation організований за типами компонентів: navigation (NavigationGraph, ScreenRoute, AppNavigation), ui/screen (Compose-екрани), ui/components (перевикористовувані Compose-компоненти), viewmodel (ViewModel-класи) та state (класи стану UiState, AuthUiState). Пакет di містить чотири Koin-модулі (networkModule, repositoryModule, viewModelModule, databaseModule), а пакет utils — допоміжні константи та перелічувані типи (TimeRange, AuthMode).

Для реалізації принципу інверсії залежностей та забезпечення слабкого зв'язування компонентів у проєкті використовується фреймворк Koin. На відміну від Dagger/Hilt, Koin є легковаговим рішенням без кодогенерації, що базується на Kotlin DSL та повністю інтегрується з Jetpack ViewModel. Ініціалізація Koin відбувається у класі App (нащадок Application) через виклик startKoin у методі onCreate, де реєструються чотири модулі залежностей.

Модуль networkModule створює singleton-екземпляри Json (з параметром ignoreUnknownKeys = true для стійкості до змін у відповіді API), OkHttpClient (з логуванням запитів через HttpLoggingInterceptor та таймаутами 15 секунд на з'єднання та читання), Retrofit (налаштований на базовий URL CryptoCompare API з конвертером kotlinx.serialization) та інтерфейсу CryptoApi. Модуль repositoryModule реєструє singleton-екземпляри Firebase-сервісів (FirebaseAuth, FirebaseFirestore) та всіх реалізацій репозиторіїв, прив'язаних до своїх інтерфейсів — це забезпечує можливість підміни реалізації без зміни коду ViewModel. Модуль databaseModule створює екземпляр Room-бази MainDb (з іменем crypto_database) та надає DAO-об'єкти RecentCoinsDao і WatchlistDao. Модуль viewModelModule реєструє всі ViewModel-класи додатку через

директиву viewModel, що дозволяє Koin автоматично керувати їхнім життєвим циклом у відповідності до lifecycle-aware компонентів Android.

Управління станом у додатку побудовано на основі реактивного підходу з використанням StateFlow. Кожен ViewModel містить один або кілька MutableStateFlow, значення яких змінюються в результаті виконання асинхронних операцій у viewModelScope. Compose-екрани підписуються на ці потоки за допомогою функції collectAsStateWithLifecycle, що гарантує автоматичне скасування підписки при знищенні екрана та запобігає витокам пам'яті. Локальні списки (watchlist, recent) зберігаються у Room та повертаються як Flow, що автоматично сповіщають підписників про будь-які зміни в базі даних — це забезпечує реактивне оновлення інтерфейсу без явного перезапиту даних. Для мережевих запитів усі результати загортаються у тип Result<T>, що дозволяє ViewModel явно обробляти як успішну відповідь, так і помилку, формуючи відповідний стан UiState.Loading, UiState.Success або UiState.Error для відображення у інтерфейсі.

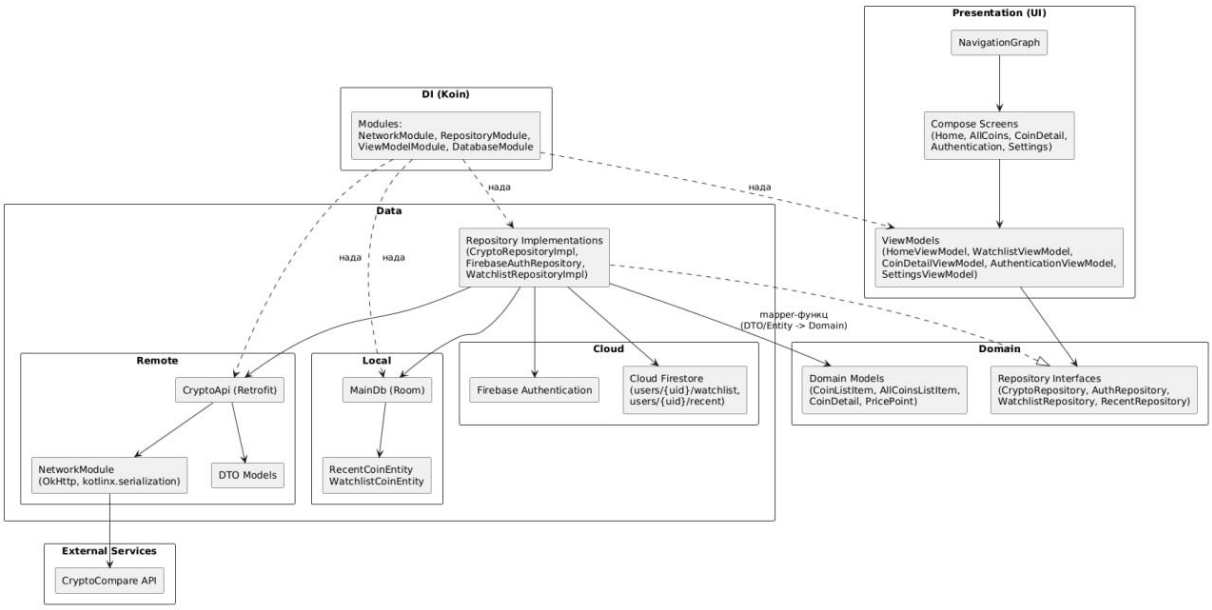

Рисунок 2.1 − Загальна архітектура програмної системи The Futures

Наведена архітектурна схема (Рис.2.1) демонструє, що всі залежності є односпрямованими: Compose-екрани залежать від ViewModel, ViewModel — від інтерфейсів репозиторіїв, реалізації репозиторіїв — від конкретних джерел

даних (Retrofit, Room, Firebase). Доменний шар при цьому залишається повністю ізольованим від деталей реалізації, що відповідає принципам Clean Architecture та забезпечує довгострокову підтримуваність і тестованість програмного коду.

1.3 Моделі даних і сховище

Доменний шар містить моделі CoinListItem, AllCoinsListItem, CoinDetail і PricePoint. Вони описують дані, потрібні інтерфейсу: символ монети, повну назву, URL зображення, поточну ціну, зміну за 24 години, капіталізацію, обсяг торгів, доступну та максимальну кількість монет, а також точки історичних цін для побудови графіка.

Віддалені DTO-моделі відповідають структурі CryptoCompare API. Для перетворення DTO у доменні моделі використовуються mapper-функції. Завдяки цьому екранна логіка не залежить від формату відповіді API, а зміни у структурі зовнішнього сервісу локалізуються у шарі data.mapper.

Локальна база MainDb оголошена як RoomDatabase і містить сутності RecentCoinEntity та WatchlistCoinEntity. DAO-інтерфейси RecentCoinsDao і WatchlistDao відповідають за вставку, вибірку, видалення й обмеження кількості збережених записів. У хмарному режимі ці функції доповнюються Firestore-репозиторіями, які працюють у межах документа поточного користувача.

1.4 Навігація та сценарії користувача

Навігація реалізована у NavigationGraph. Початковим маршрутом є Loading, після якого додаток перевіряє FirebaseAuth.getInstance().currentUser. Якщо користувач уже авторизований, відкривається Home, інакше - Authentication. Такий підхід забезпечує автоматичне відновлення сесії.

Основними екранами є Home, AllCoins, CoinDetail, Authentication і Settings. Home показує список підписок і нещодавно переглянуті монети; AllCoins відображає повний перелік криптовалют; CoinDetail надає детальну

інформацію, графік і кнопку підписки; Authentication відповідає за вхід, реєстрацію та відновлення пароля; Settings містить перемикання теми й вихід з облікового запису.

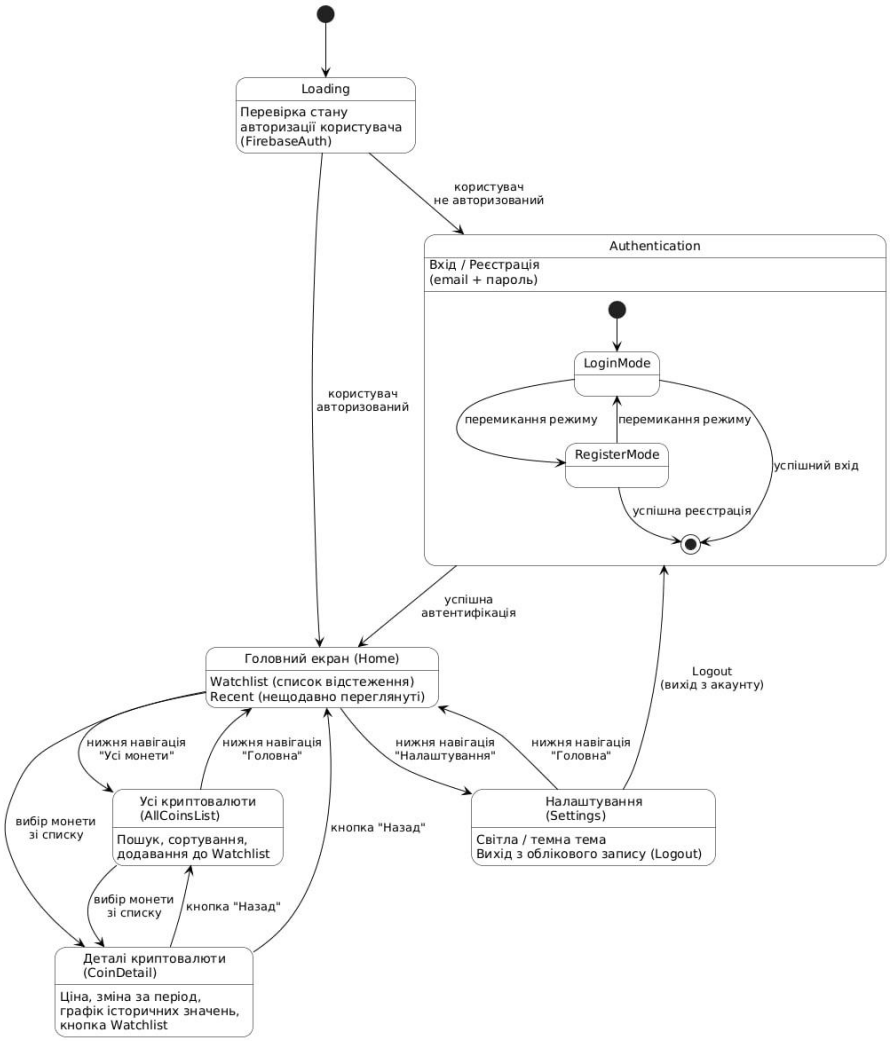

Рисунок 2.2 − Навігаційна структура додатку The Futures

1.5 Побудова UML – діаграм класів

Для повноцінного документування проєктних рішень, прийнятих у межах даної кваліфікаційної роботи, використано уніфіковану мову моделювання UML (Unified Modeling Language). Застосування UML-діаграм дозволяє формалізувати як статичну структуру системи (класи, зв'язки, компоненти), так і її динамічну поведінку (послідовності взаємодій, навігація між екранами), що суттєво спрощує розуміння архітектурних рішень та є обов'язковим елементом

проєктної документації програмного продукту. У межах розділу побудовано UML-діаграми класів та діаграму послідовності.

Статична структура основних класів програмної системи та зв'язки між ними відображені на діаграмах класів (Рис. 2.3 та Рис. 2.4). Через значну кількість класів діаграма поділена на дві частини: перша охоплює доменний шар та шар репозиторіїв (включаючи мережеву та хмарну взаємодію), друга — шар локального сховища Room та презентаційний шар (стани інтерфейсу і ViewModel-класи).

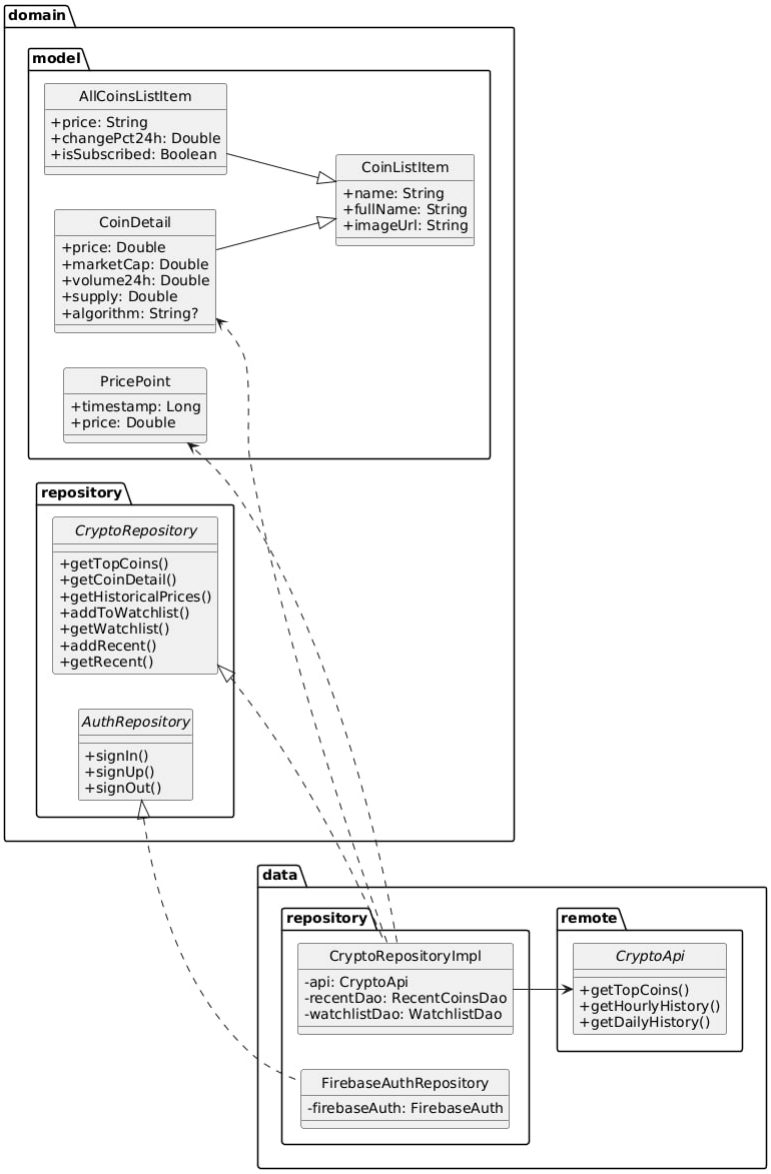

Рисунок 2.3 — Діаграма класів: доменний шар та репозиторії

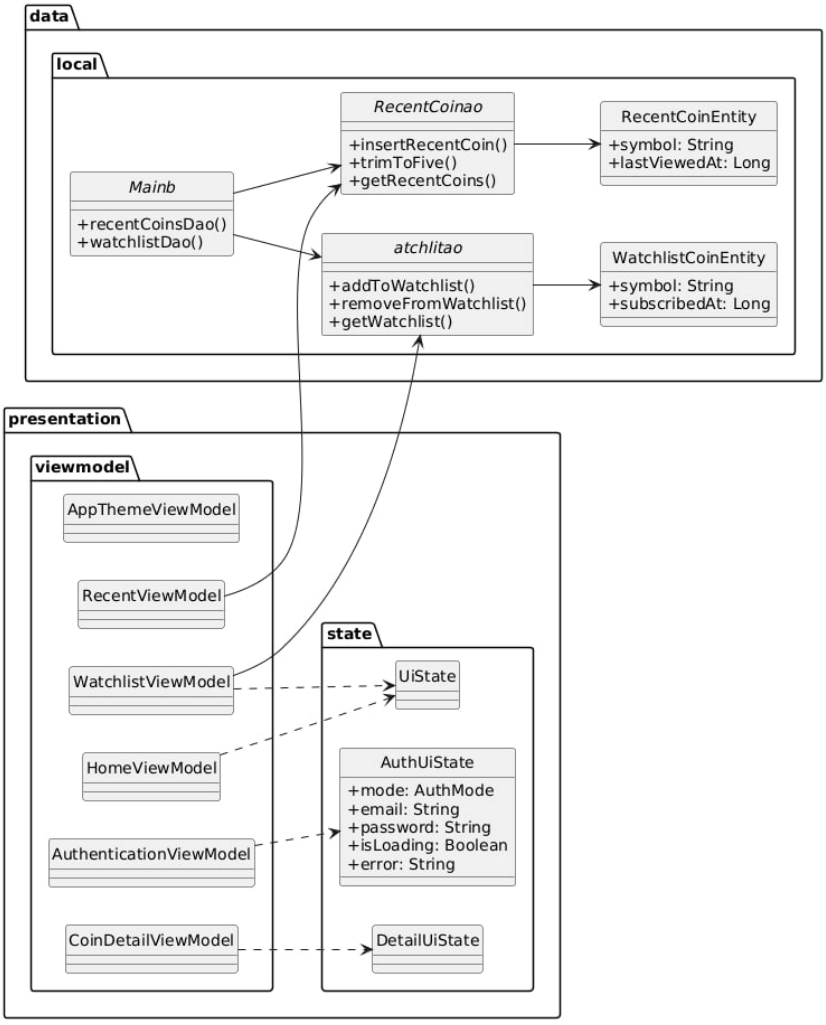

Рисунок 2.4 — Діаграма класів: локальне сховище та презентаційний шар

Перша частина діаграми демонструє ієрархію доменних моделей (CoinListItem як базова, AllCoinsListItem та CoinDetail як розширені), інтерфейси репозиторіїв та їх конкретні реалізації. Чітко простежується реалізація принципу інверсії залежностей: ViewModel звертається виключно до інтерфейсів CryptoRepository та AuthRepository, не маючи жодної прямої залежності від конкретних класів реалізації, Retrofit, Firebase чи Room. Друга частина ілюструє структуру локального сховища Room (сутності RecentCoinEntity та WatchlistCoinEntity, DAO-інтерфейси та клас MainDb), а також ViewModel-класи та sealed-класи стану UiState, DetailUiState і AuthUiState, які визначають можливі стани кожного екрана.

Для ілюстрації динамічної взаємодії компонентів системи під час виконання одного з ключових сценаріїв побудовано UML-діаграму послідовності для прецеденту «Перегляд деталей криптовалюти та побудова графіка історичних цін». Цей сценарій обрано як найбільш показовий, оскільки він охоплює повний ланцюжок взаємодій: від дії користувача до мережевого запиту, локальної перевірки та оновлення інтерфейсу. Діаграма також поділена на дві частини для зручності читання.

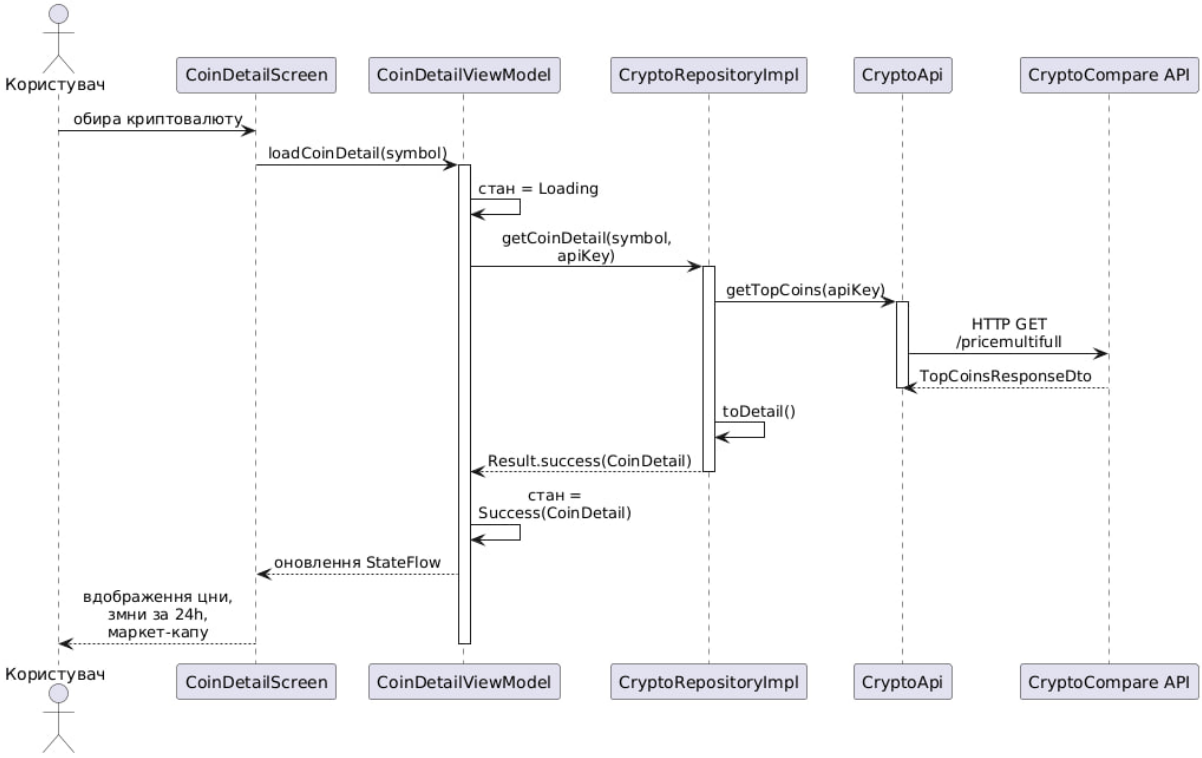

Рисунок 2.5 — Діаграма послідовності: отримання деталей криптовалюти

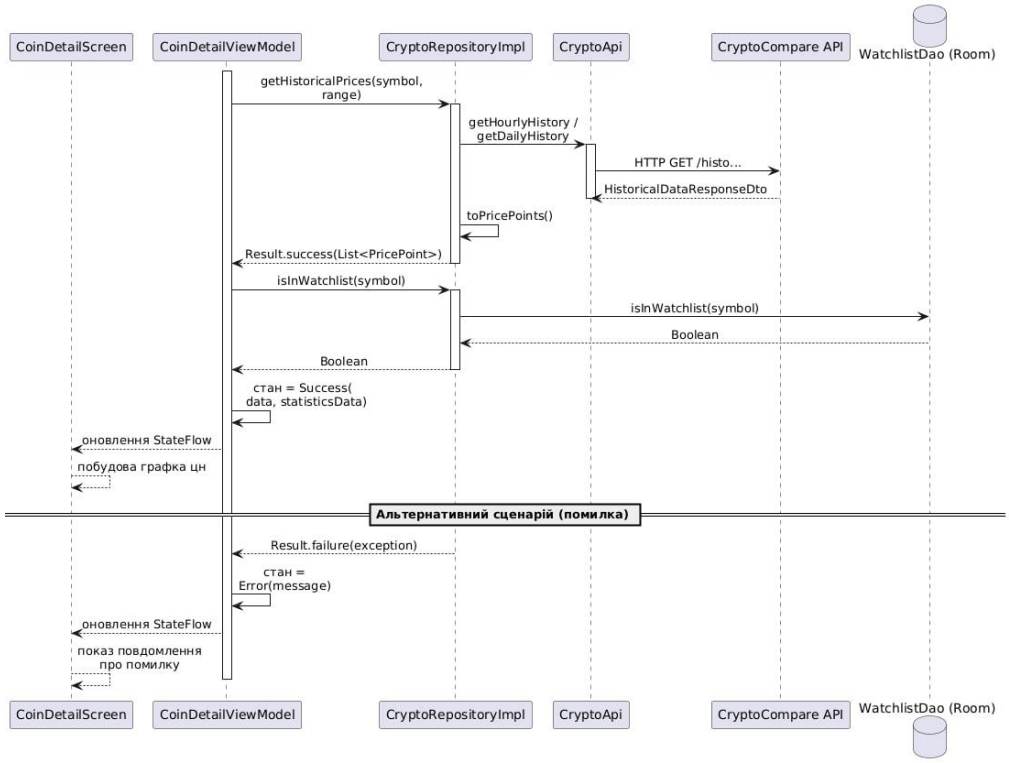

Рисунок 2.6 — Діаграма послідовності: завантаження графіка цін та обробка помилок

Перша частина діаграми (Рис. 2.6) відображає початковий етап сценарію: вибір монети користувачем, перехід у стан Loading, мережевий запит через CryptoApi до CoinGeko API, маппінг отриманого DTO у доменну модель CoinDetail та оновлення стану екрана. Друга частина (Рисунок 2.5б) охоплює завантаження історичних цін (з вибором типу запиту залежно від обраного часового діапазону TimeRange), перевірку статусу Watchlist у локальній базі Room та формування фінального стану DetailUiState.Success. Окремо показано альтернативний потік: у разі мережевої помилки репозиторій повертає Result.failure, ViewModel переходить у стан Error, а екран відображає повідомлення про помилку замість даних.

Сукупність побудованих UML-діаграм формує повну проєктну документацію програмної системи The Futures та є основою для її програмної реалізації, яка детально описується у третьому розділі.

1.6 Налаштування Android-проєкту

Проєкт створено як Android Application із namespace com.example.thefuture, applicationId com.example.thefuture, мінімальною версією SDK 28 і цільовою версією SDK 36. У manifest-файлі оголошено дозвіл INTERNET, необхідний для звернення до CryptoCompare API та Firebase.

У build.gradle.kts підключено плагіни Android Application, Kotlin Android, Kotlin Compose, Google Services, kapt і Kotlin Serialization. Для побудови інтерфейсу активовано buildFeatures.compose. Компіляція виконується з JavaVersion.VERSION_11 і jvmTarget 11.

Ресурси додатку включають іконки для нижнього меню, логотип The Futures, фони світлої й темної теми, іконки криптовалютних змін, шрифт Poppins та XML-ресурси теми. Це дає змогу підтримувати єдиний стиль на всіх екранах.

1.6.1 Реалізація мережевого шару

Інтерфейс CryptoApi описує три основні запити: getTopCoins для отримання популярних криптовалют, getHourlyHistory для погодинної історії та getDailyHistory для історичних даних за більші періоди. Усі методи є suspend-функціями, тому виконуються асинхронно в корутинах.

NetworkModule створює Json з ignoreUnknownKeys, OkHttpClient із логуванням запитів і таймаутами 15 секунд, Retrofit з базовою адресою https://min-api.cryptocompare.com/ та конвертером kotlinx.serialization. У модулі також створюється реалізація CryptoApi.

У CryptoRepositoryImpl мережеві виклики обгорнуті в Result. Це дозволяє ViewModel отримувати або успішні дані, або помилку, не перериваючи роботу інтерфейсу винятками. Для історії цін вибір API-запиту залежить від TimeRange: день, тиждень, місяць, три місяці або рік.

1.6.2 Авторизація та користувацькі дані

Авторизація реалізована через FirebaseAuthRepository, який інкапсулює FirebaseAuth. Репозиторій підтримує вхід, реєстрацію, надсилання листа для

відновлення пароля, вихід і отримання ідентифікатора поточного користувача. Для перетворення Firebase Task у suspend-виклики використано kotlinx-coroutines-play-services.

AuthenticationViewModel містить стан AuthUiState, обробляє зміну email і пароля, перемикання між режимами входу та реєстрації, перевірку валідності полів і перетворення типових Firebase-помилок у зрозумілі повідомлення. Якщо авторизація успішна, екран навігує користувача до Home.

Дані watchlist і recent у Firestore зберігаються у структурі users/{uid}/watchlist та users/{uid}/recent. Репозиторії використовують callbackFlow і addSnapshotListener, тому зміни в базі синхронізуються з інтерфейсом у реальному часі. Для recent додатково реалізовано обмеження кількості документів, щоб список не розростався безконтрольно.

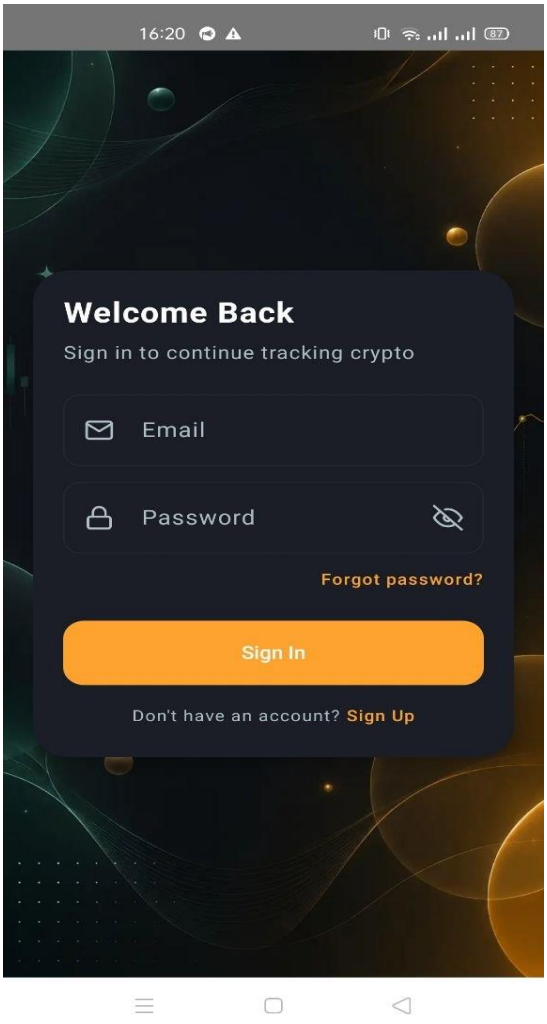

Рисунок 3.1 – Екран входу

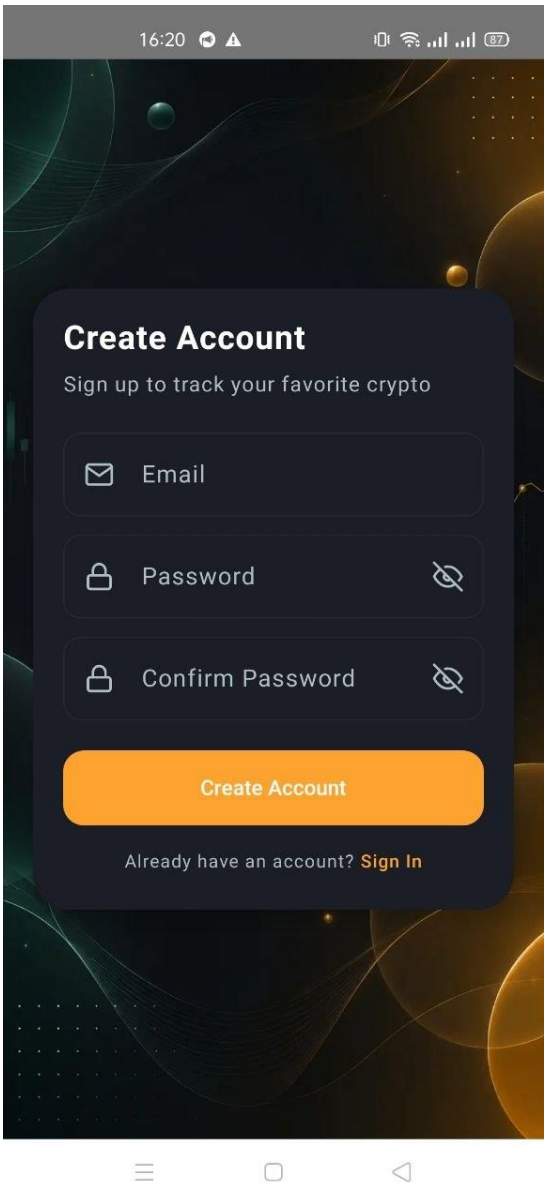

Рисунок 3.2 – Реєстрація користувача

1.7 Налаштування екранів

1.7.1 Головний екран і список відстеження

HomeScreen отримує watchlist із WatchlistViewModel та recentCoins із RecentViewModel. Екран містить LogoBar, секцію підписок, список нещодавно переглянутих монет і BottomBarMenu. Усі дані надходять як Flow і перетворюються в Compose-стан через collectAsState.

Список відстеження дає користувачу змогу швидко перейти до деталей монети, яку він додав раніше. Нещодавно переглянуті активи формуються

автоматично під час відкриття CoinDetailScreen. Така комбінація покращує повторний доступ до важливих активів і зменшує кількість зайвих переходів.

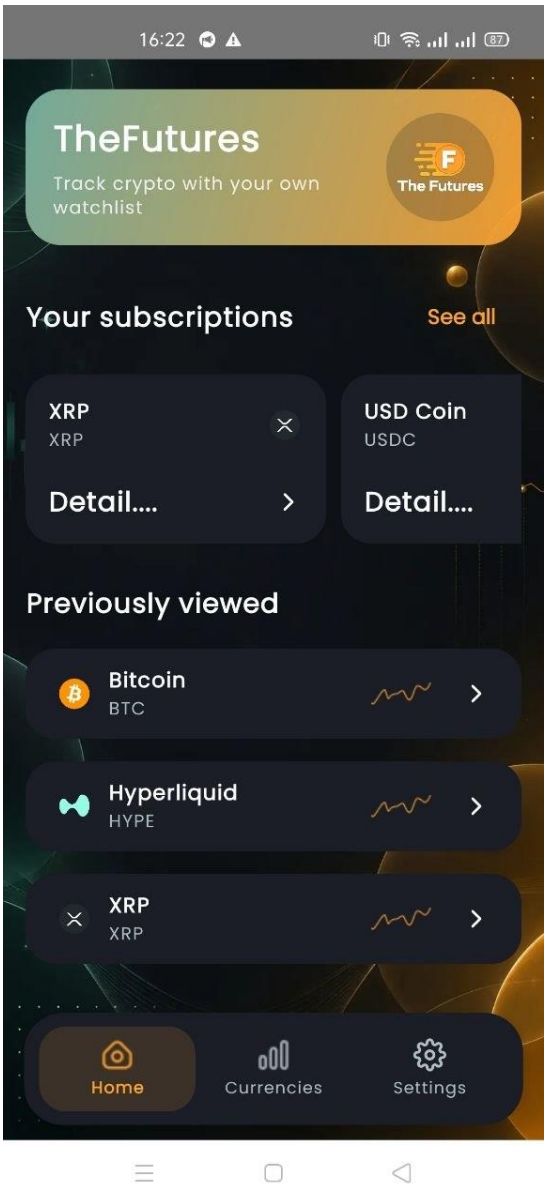

Рисунок 3.3 – Головний екран додатку The Futures

1.7.2 Екран усіх криптовалют

AllCurrenciesListScreen відображає стан UiState, який може бути Loading, Error або Success. Під час завантаження показується CircularProgressIndicator, у разі помилки - текстове повідомлення, а при успішному результаті - LazyColumn зі списком монет.

Для кожної монети користувач бачить коротку інформацію та може додати її до списку відстеження або видалити з нього. Поточний watchlist

перетворюється у множину символів, що дає змогу швидко визначати, чи підписаний користувач на конкретний актив.

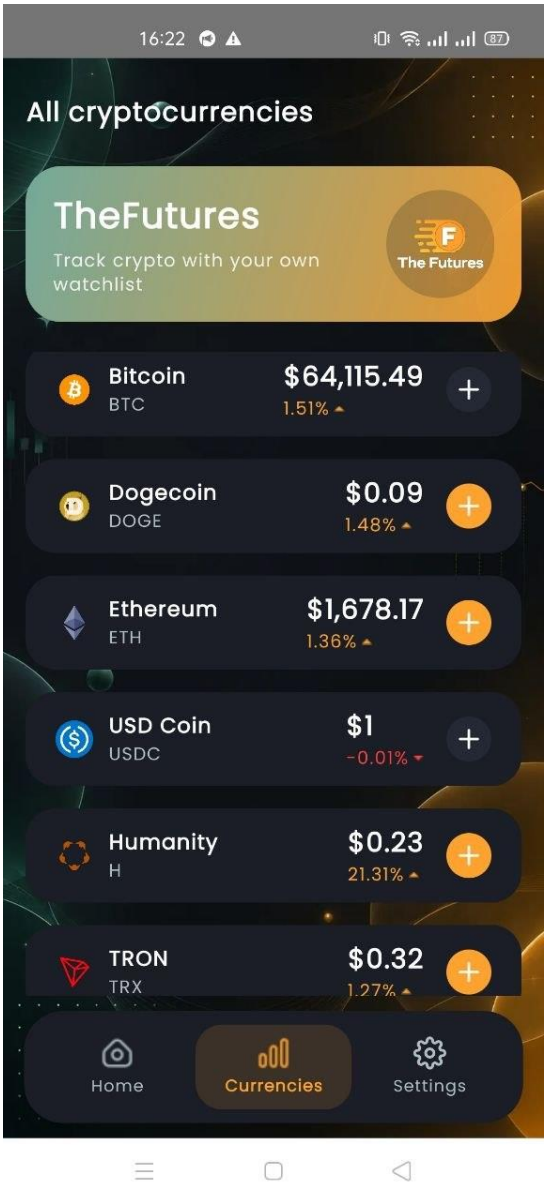

Рисунок 3.4 – Екран переліку криптовалют

1.7.3 Детальна інформація про монету та тема оформлення

CoinDetailScreen є центральним екраном аналітики. Він отримує символ монети з маршруту, завантажує детальну інформацію та історичні значення ціни відповідно до вибраного TimeRange. Користувач може перемикати період графіка між DAY, WEEK, MONTH, MONTH I YEAR.

Екран показує іконку монети через Coil AsyncImage, назву, поточну ціну, відсоткову зміну, кнопку підписки, графік динаміки та групу статистичних показників: ринкова капіталізація, обсяг за 24 години, доступна й максимальна

кількість монет. Додавання у recent виконується при відкритті успішно завантаженої картки.

Рисунок 3.5 – Детальна інформація про криптовалюту

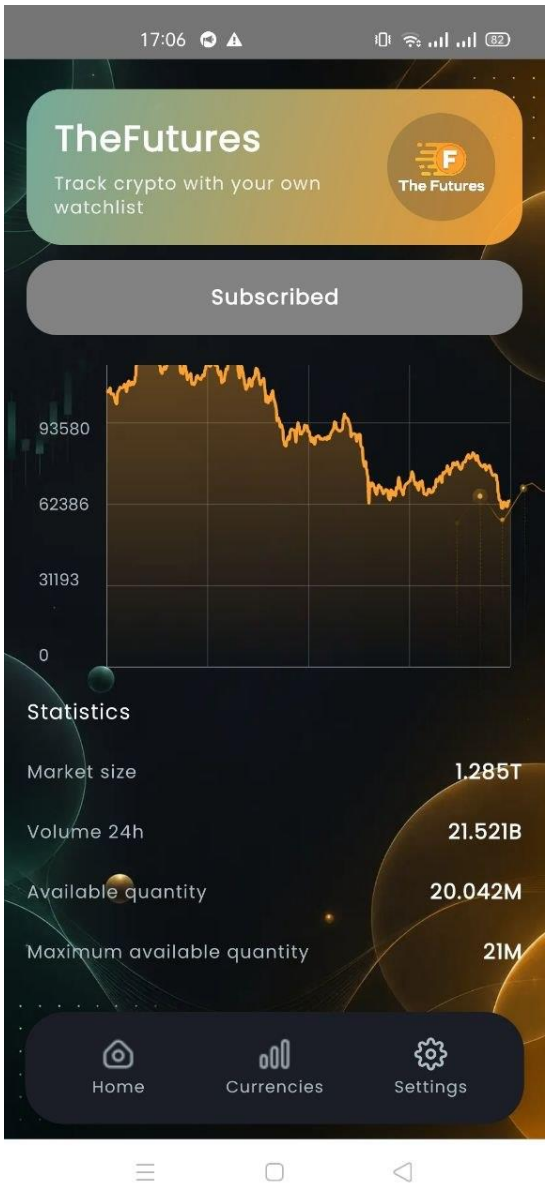

Рисунок 3.6 − Графік історичних значень ціни

SettingsScreen надає користувачу можливість обрати світлу або темну тему та вийти з облікового запису. Стан теми зберігається через AppThemeViewModel і ThemePreferenceRepository. Компоненти ThemeModeCard візуально виділяють активний режим, а кнопка Sign Out викликає SettingsViewModel.signOut і повертає користувача на Authentication.

Підтримка темної теми є важливою для фінансових додатків, оскільки користувачі часто переглядають ринкові дані в різних умовах освітлення.

Завдяки MaterialTheme і власним кольорам інтерфейс зберігає єдину стилістику без дублювання розмітки екранів.

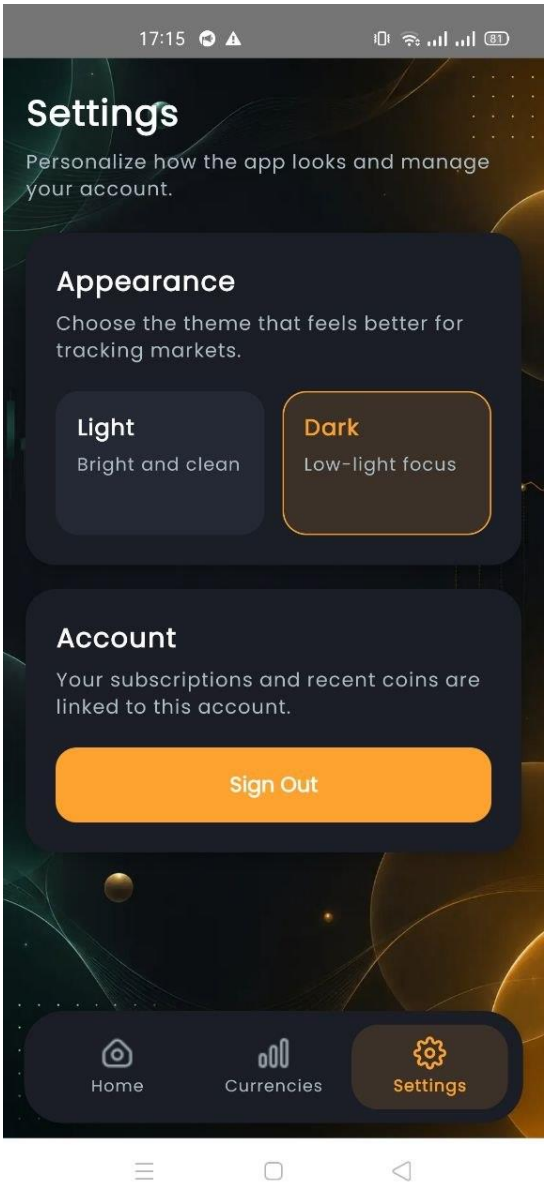

Рисунок 3.7 – Екран налаштувань додатку

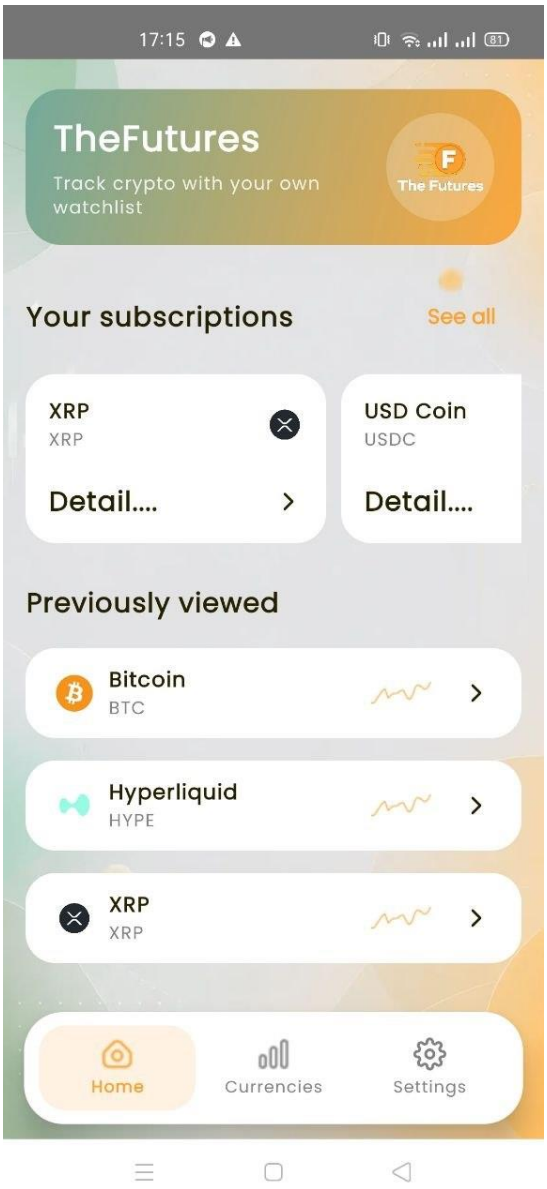

Рисунок 3.8 – Варіанти теми оформлення

1.8 Висновки до 2 розділу

У другому розділі виконано повне проєктування програмної системи мобільного криптовалютного додатку The Futures. Усі прийняті архітектурні та технологічні рішення обґрунтовано з урахуванням вимог, сформованих у першому розділі, та орієнтовані на забезпечення масштабованості, підтримуваності та якості кінцевого програмного продукту.

Обґрунтовано вибір технологічного стеку. Мову Kotlin обрано як єдину офіційно підтримувану мову Android-розробки з розвиненою підтримкою

корутин та функціонального стилю програмування. Jetpack Compose обрано для побудови декларативного інтерфейсу користувача, що скорочує обсяг шаблонного коду порівняно з XML-підходом. Для мережевої взаємодії застосовано Retrofit у поєднанні з kotlinx.serialization, для авторизації та синхронізації даних — Firebase Authentication і Cloud Firestore, для локального кешування — Room, а для впровадження залежностей — Koin. Сукупність цих технологій утворює сучасний та практично обґрунтований стек для розробки Android-застосунків.

Спроєктовано архітектуру програмної системи на основі патерну MVVM із чітким поділом на шари Presentation, Domain та Data. Структура пакетів відповідає принципу єдиної відповідальності: кожен пакет (data, domain, presentation, di, utils) містить лише ті компоненти, за які він відповідає. Визначено доменні моделі (CoinListItem, AllCoinsListItem, CoinDetail, PricePoint), описано сутності локальної бази даних Room (RecentCoinEntity, WatchlistCoinEntity) та їх DAO-інтерфейси, а також спроєктовано інтерфейси репозиторіїв і їх конкретні реалізації. Навігаційна структура додатку реалізована через єдиний NavigationGraph із чітким розмежуванням між авторизованою та неавторизованою зонами.

Для повноцінного документування прийнятих проєктних рішень побудовано комплект UML-діаграм: схему загальної архітектури (Рисунок 2.1), що ілюструє шарування системи та односпрямованість залежностей; діаграму навігаційної структури (Рисунок 2.2), що описує переходи між екранами та умови їх виконання; дві частини діаграми класів (Рисунки 2.3а та 2.3б), що відображають статичну структуру системи та зв'язки між компонентами; а також дві частини діаграми послідовності (Рисунки 2.5а та 2.5б), що ілюструють динамічну взаємодію компонентів під час виконання ключового сценарію перегляду деталей криптовалюти.

Результати проєктування, отримані в другому розділі, є повною і достатньою основою для програмної реалізації додатку.

2 РОЗДІЛ ТЕСТУВАННЯ ТА ОЦІНКА ЯКОСТІ

2.1 Тестування програмної системи

Процес тестування розробленого мобільного застосунку є ключовим механізмом забезпечення його надійності та якості. Тестування дозволяє виявити невідповідності між очікуваною та фактичною поведінкою системи, усунути потенційні помилки до кінцевого випуску продукту та підтвердити, що реалізований функціонал задовольняє встановлені вимоги.

2.1.1 Види та план тестування

Для забезпечення комплексної перевірки якості додатку The Futures застосовано декілька видів тестування, кожен із яких орієнтований на виявлення певного класу проблем.

Функціональне тестування передбачає перевірку відповідності реалізованих функцій додатку визначеним вимогам. Охоплює тестування всіх ключових модулів: авторизації та реєстрації, перегляду переліку криптовалют, управління списком відстеження, перегляду детальної інформації та графіка цін, перемикання теми оформлення та виходу з облікового запису.

Продуктивність тестування спрямоване на оцінку поведінки застосунку в умовах обмежених ресурсів або нестабільного мережевого з'єднання. Включає перевірку часу відгуку на запити, споживання оперативної пам'яті та навантаження на процесор за допомогою Android Profiler.

Тестування сумісності перевіряє коректне функціонування додатку на різних версіях Android та пристроях із різними розмірами екранів та щільністю пікселів.

Регресійне тестування виконується після внесення змін до коду та переконується у тому, що раніше реалізований і перевірений функціонал продовжує працювати коректно.

Автоматизоване модульне тестування перевіряє ізольовані компоненти системи (mapper-функції, репозиторії, ViewModel) без залежності від зовнішніх сервісів.

План тестування побудовано таким чином, щоб охопити як функціональні, так і нефункціональні вимоги до додатку. На першому етапі проводиться функціональне тестування методом «чорної скриньки» — перевірка поведінки додатку з позиції кінцевого користувача без знання деталей внутрішньої реалізації. На другому етапі виконується продуктивнісне тестування з використанням вбудованих інструментів Android Studio. На третьому етапі проводиться автоматизоване тестування модульних компонентів. Завершальним етапом є регресійне тестування та верифікація відповідності вимогам.

2.1.2 Функціональне тестування

Функціональне тестування додатку The Futures проводилося з метою перевірки відповідності реальної поведінки системи очікуваній. Тестування виконувалося на фізичному Android-пристрої та в емуляторі Android Studio з різними версіями API.

Тест-кейс 1 Summary: Перевірка успішної реєстрації нового користувача (Positive).

Steps to reproduce:

1. Запустити додаток та дочекатися завершення екрана завантаження.
2. На екрані автентифікації перейти у режим реєстрації.
3. Ввести коректну електронну адресу та пароль довжиною не менше 6 символів.
4. Натиснути кнопку «Sign Up».

Expected results: Firebase Authentication створює новий обліковий запис, додаток автоматично переходить на головний екран Home зі щойно створеним порожнім списком відстеження, екран Authentication видаляється зі стеку навігації.

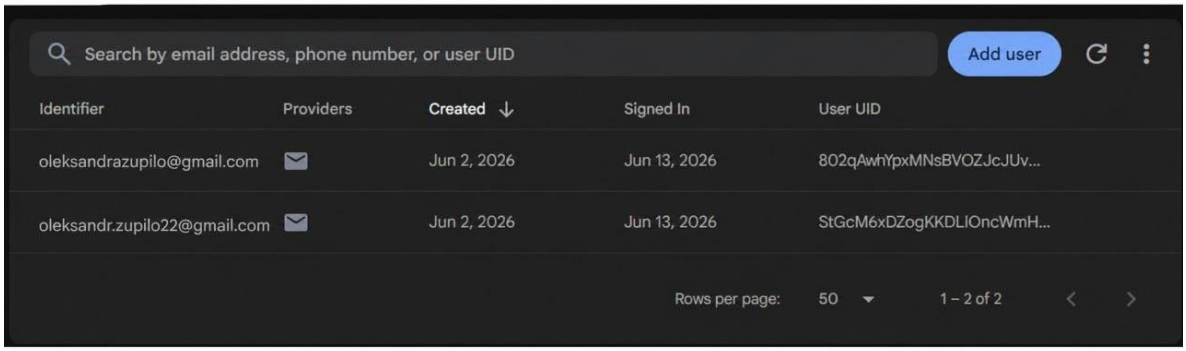

Рисунок 3.1.1 — Успішна реєстрація нового акаунту

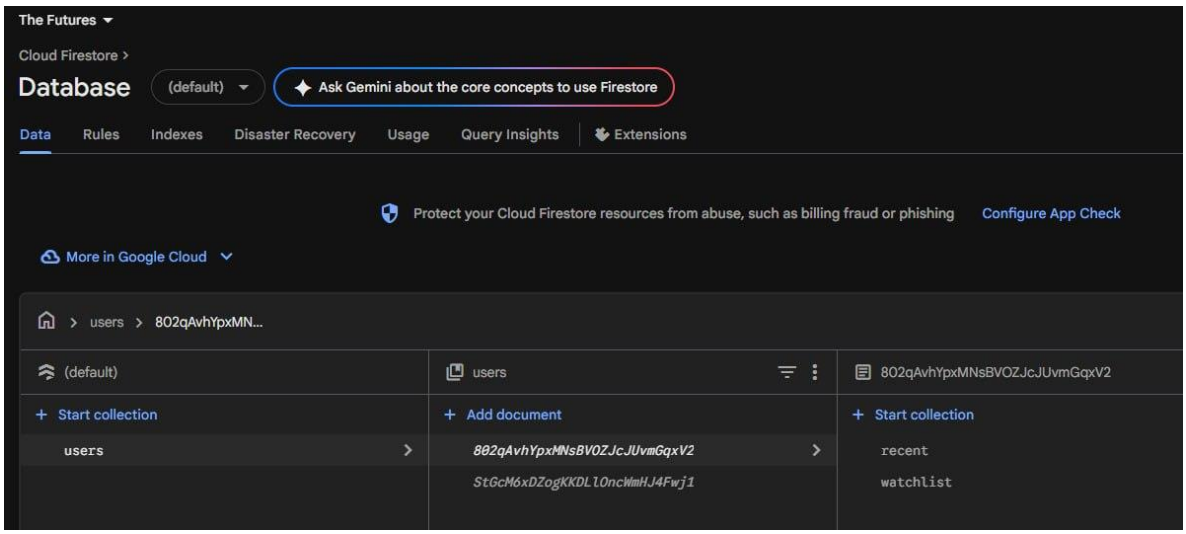

Рисунок 3.1.2 – Результати тестування бази даних

Тест-кейс 2 Summary: Перевірка відображення помилки при введенні некоректних даних авторизації (Negative).

Steps to reproduce:

1. На екрані автентифікації у режимі входу ввести зареєстровану електронну адресу та неправильний пароль.
2. Натиснути кнопку «Sign In».

Expected results: Firebase повертає помилку автентифікації, додаток залишається на екрані авторизації та відображає відповідне повідомлення про помилку у полі error стану AuthUiState. Перехід на захищені екрани не відбувається.

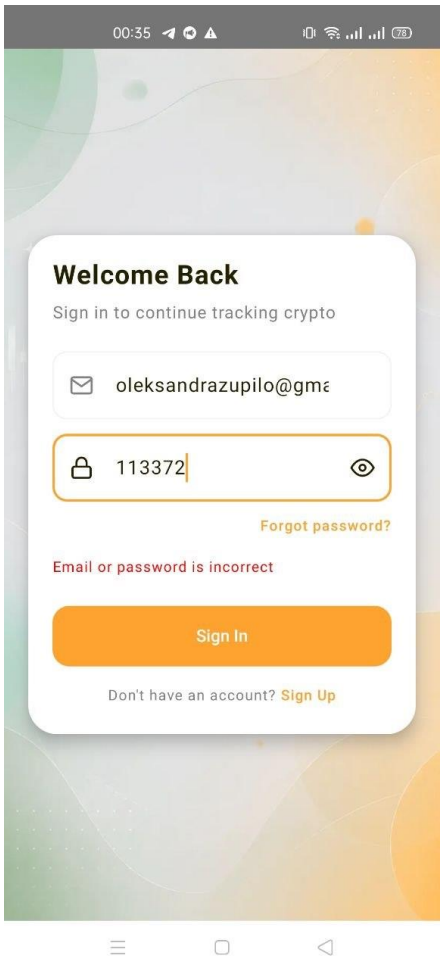

Рисунок 3.1.3 — Відображення помилки при некоректних даних

Загалом результати функціонального тестування підтвердили коректну роботу всіх ключових модулів системи. Усі перевірені функціонують відповідно до визначених вимог.

2.1.3 Навантажувальне тестування

Для мобільного застосунку навантажувальне тестування стосується не серверного навантаження, а продуктивності самого клієнтського застосунку в умовах інтенсивного використання та обмежених апаратних ресурсів. Тестування проводилося за допомогою вбудованого інструмента Android Studio Profiler, який надає детальну інформацію про споживання ресурсів процесора, оперативної пам'яті та мережевої активності під час роботи застосунку.

Сценарій тестування передбачав послідовне виконання серії дій: запуск застосунку, авторизацію, перегляд переліку з 50 криптовалют, послідовний перехід на екрани деталей для 10 різних монет із перемиканням часових

діапазонів графіка, додавання та видалення монет зі списку відстеження, перемикання теми оформлення. Кожна серія дій виконувалась безперервно протягом 5 хвилин.

За результатами профілювання встановлено, що споживання оперативної пам'яті під час звичайної роботи стабілізується на рівні 120–160 МБ, що є прийнятним показником для сучасних Android-пристроїв. Час відповіді мережевих запитів до CryptoCompare API склав у середньому 350–600 мілісекунд при стабільному Wi-Fi-з'єднанні та 800–1400 мілісекунд при мобільному з'єднанні 4G. Завантаження даних завжди супроводжується відображенням стану Loading, що виключає «заморозку» інтерфейсу під час очікування.

Навантаження на процесор у фоновому режимі (коли додаток згорнутий) практично відсутнє завдяки коректному скасуванню підписок на Flow при знищенні ViewModel та використанню collectAsStateWithLifecycle замість collectAsState. Бібліотека Coil для завантаження зображень іконок монет ефективно кешує завантажені зображення, що усуває повторні мережеві запити при поверненні до раніше переглянутих екранів і суттєво знижує мережеве навантаження.

Тестування на пристроях із різними характеристиками показало, що на бюджетних пристроях (2 ГБ RAM, процесор середнього класу) час початкового завантаження додатку та переходу між екранами збільшується на 20–30% порівняно з флагманськими моделями, однак залишається в прийнятних межах. Жодних суттєвих помилок або аварійних завершень роботи додатку під час навантажувального тестування виявлено не було.

2.1.4 Автоматизоване тестування

Для реалізації автоматизованого тестування ізольованих компонентів системи написано модульні тести для найбільш критичних із точки зору бізнес-логіки елементів — mapper-функцій та логіки ViewModel. Модульне тестування є особливо важливим для mapper-функцій (toDetail, toAllCoinsListItem, toPricePoints), оскільки вони відповідають за перетворення вхідних даних від зовнішнього API у доменні моделі, й будь-яка помилка у маппінгу призведе до некоректного відображення даних у всьому застосунку.

Тести для mapper-функцій побудовано на фреймворку JUnit4. Кожен тест перевіряє, що при подачі конкретного DTO-об'єкта на вхід mapper повертає очікуваний доменний об'єкт із коректно заповненими полями. Наприклад, тест перетворення AllCoinInfoDto перевіряє, що поля name, fullName, price (з коректним форматуванням), changePct24h та isPositiveChange у результуючому AllCoinsListItem відповідають значенням у вхідному DTO. Окремо перевіряється граничний випадок: якщо значення changePct24h від'ємне, поле isPositiveChange має бути false.

```
@Test
fun `toAllCoinsListItem maps fields correctly for positive change`() {
    val dto = CoinMarketDto(
        id = "bitcoin",
        symbol = "btc",
        name = "Bitcoin",
        image = "https://example.com/btc.png",
        currentPrice = 67450.0,
        priceChangePercentage24h = 2.45
    )

    val result = dto.toAllCoinsListItem()

    assertEquals("BTC", result.name)
    assertEquals("Bitcoin", result.fullName)
    assertEquals("https://example.com/btc.png", result.imageUrl)
    assertEquals(2.45, result.changePct24h, 0.001)
    assertTrue("isPositiveChange must be true when change >= 0", result.isPositive
}
```

Рисунок 3.4 — Фрагмент модульних тестів mapper-функцій

Тестування ViewModel виконується з використанням бібліотек kotlinx-coroutines-test та Mockito для підміни реальних репозиторіїв тестовими

реалізаціями (mock-об'єктами). Тест перевіряє, що після виклику методу завантаження ViewModel спочатку переходить у стан Loading, а після отримання даних від mock-репозиторію — у стан Success із коректними даними. Аналогічно перевіряється перехід у стан Error при отриманні Result.failure від репозиторію.

Виконання тестового набору не виявило жодних помилок у реалізації mapper-функцій та логіці обробки стану в ViewModel. Усі тести успішно пройдені, що підтверджує коректність перетворення даних між шарами та правильність управління станами UI в ключових сценаріях використання.

2.2 Верифікація програмної системи

Етап верифікації полягає у систематичній перевірці відповідності реалізованого програмного продукту вимогам, сформованим у першому розділі. На відміну від тестування, яке фокусується на пошуку помилок, верифікація підтверджує, що розроблений застосунок вирішує поставлену задачу та відповідає очікуванням замовника.

У межах верифікації функціоналу авторизації та управління сесією перевірено, що застосунок коректно зберігає стан авторизації між запусками (Firebase автоматично відновлює сесію), правильно перенаправляє неавторизованих користувачів на екран автентифікації та повністю очищає стек навігації при виконанні logout, унеможливлюючи доступ до захищених екранів після виходу з облікового запису.

Верифікація модуля перегляду криптовалютного ринку підтвердила, що список AllCoins коректно відображає дані, отримані від CryptoCompare API, включаючи актуальні ціни та відсоткові зміни за 24 години. Відображення стану завантаження (Loading) та помилки (Error) відповідає визначеним вимогам — користувач завжди отримує зрозумілий зворотний зв'язок про поточний стан запиту. Коректність форматування числових значень (ціни, відсотки, обсяги) відповідає стандартам відображення фінансової інформації.

Особлива увага під час верифікації приділялася надійності роботи персональних списків користувача. Перевірено, що дані watchlist та recent синхронізуються між локальною базою Room та Cloud Firestore: зміни, внесені на одному пристрої, після повторної авторизації відображаються на іншому. Граничний випадок — список нещодавніх переглядів обмежується п'ятьма останніми записами завдяки виклику trimToFive у RecentCoinsDao — верифіковано успішно.

Верифікація навігаційної структури підтвердила коректну поведінку кнопки «Назад» у всіх сценаріях: повернення з CoinDetail на попередній екран, неможливість повернення на екран Loading або Authentication після успішної авторизації, правильне очищення стеку при logout. Нижнє навігаційне меню коректно позначає активний пункт та не створює дублікатів екранів у стеку навігації завдяки параметру launchSingleTop.

Верифікація підтримки тем оформлення засвідчила, що перемикання між світлою та темною темою миттєво відображається на всіх екранах застосунку, а обраний параметр зберігається у SharedPreferences та відновлюється після перезапуску додатку. Усі елементи інтерфейсу коректно використовують кольори з поточної теми MaterialTheme, забезпечуючи візуальну цілісність.

За результатами верифікації підтверджено, що всі функціональні вимоги, визначені у розділі 1, реалізовані в повному обсязі. Нефункціональні вимоги — зокрема збереження даних при нестабільному з'єднанні, коректне форматування числових значень та підтримка персоналізації — також виконані. Розроблений застосунок The Futures відповідає поставленій меті кваліфікаційної роботи.

2.3 Висновки до розділу 3

У третьому розділі виконано комплексне тестування, оцінку продуктивності та верифікацію розробленого мобільного криптовалютного додатку The Futures.

Проведено функціональне тестування методом «чорної скриньки», що охопило п'ять ключових тест-кейсів: успішну реєстрацію нового користувача, обробку помилки при некоректних облікових даних, додавання монети до списку відстеження, перегляд деталей криптовалюти з графіком цін та поведінку при відсутності мережевого з'єднання. Усі тест-кейси підтвердили коректну відповідність поведінки застосунку очікуваним результатам.

Виконано продуктивнісне тестування за допомогою Android Studio Profiler. Встановлено, що споживання оперативної пам'яті стабілізується на рівні 120–160 МБ, час відгуку API у нормальних умовах складає 350–600 мс, а навантаження на процесор у фоновому режимі є мінімальним завдяки коректному управлінню lifecycle-aware компонентами. Аварійних завершень роботи під час навантажувального тестування виявлено не було.

Автоматизоване модульне тестування mapper-функцій та ViewModel із використанням JUnit4 та mock-репозиторіїв підтвердило коректність перетворення даних між шарами системи та правильність логіки управління станами інтерфейсу. Усі написані тести успішно виконані без помилок.

Визначено системні вимоги до пристрою користувача (Android 8.0+, 2 ГБ RAM, наявність інтернет-з'єднання) та описано процес підготовки застосунку до розгортання з налаштуванням ProGuard/R8 та формуванням релізного збирання у форматі AAB.

Підсумкова верифікація засвідчила повну відповідність реалізованого функціоналу вимогам, сформованим у першому розділі: модулі авторизації, перегляду ринку, управління watchlist, навігації та персоналізації функціонують коректно в усіх перевірених сценаріях. Поставлену мету кваліфікаційної роботи досягнуто в повному обсязі.

3 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4 ВИСНОВКИ

У кваліфікаційній роботі виконано розробку мобільного криптовалютного додатку The Futures з використанням мови Kotlin. Під час роботи проаналізовано предметну область криптовалютного моніторингу, визначено потреби користувачів і сформовано вимоги до функціональності та якості програмного продукту.

Було обґрунтовано вибір технологій Android-розробки: Jetpack Compose для інтерфейсу, MVVM для організації логіки, Kotlin Coroutines і Flow для асинхронної роботи зі станом, Retrofit і OkHttp для мережевого шару, Firebase Authentication і Firestore для користувацьких сервісів, Room для локального збереження даних та Koin для керування залежностями.

Реалізовано основні екрани додатку: авторизацію, головний екран, список усіх криптовалют, детальну сторінку монети та налаштування. Додаток дає змогу переглядати актуальні ринкові дані, аналізувати історичну динаміку ціни, формувати список відстеження, бачити нещодавно переглянуті активи, змінювати тему оформлення та виходити з облікового запису.

Архітектура проєкту побудована з урахуванням подальшого розвитку. У майбутньому додаток можна розширити портфелем користувача, push-сповіщеннями про зміну ціни, пошуком і фільтрацією монет, підтримкою кількох валют, офлайн-кешуванням, розширеними графіками та інтеграцією з біржовими сервісами.

Отриманий результат підтверджує доцільність використання Kotlin і Jetpack Compose для створення сучасних мобільних фінансових застосунків, а також демонструє практичну цінність поєднання локального, хмарного й мережевого шарів даних у межах одного Android-додатку.

5 СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Developers. Kotlin and Android documentation. URL: https://developer.android.com/kotlin

2. Android Developers. Jetpack Compose documentation. URL: https://developer.android.com/compose

3. Android Developers. Navigation with Compose. URL: https://developer.android.com/develop/ui/compose/navigation

4. Android Developers. Guide to app architecture. URL: https://developer.android.com/topic/architecture

5. Kotlin Documentation. Coroutines guide. URL: https://kotlinlang.org/docs/coroutines-overview.html

6. Firebase Documentation. Firebase Authentication. URL: https://firebase.google.com/docs/auth

7. Firebase Documentation. Cloud Firestore. URL: https://firebase.google.com/docs/firestore

8. Retrofit documentation. URL: https://square.github.io/retrofit/

9. OkHttp documentation. URL: https://square.github.io/okhttp/

10. Room persistence library documentation. URL: https://developer.android.com/training/data-storage/room

11. Koin documentation. URL: https://insert-koin.io/docs/reference/koin-android/compose

12. CryptoCompare API documentation. URL: https://min-api.cryptocompare.com/documentation

13. Material Design 3 documentation. URL: https://m3.material.io/

14. Coil image loading library. URL: https://coil-kt.github.io/coil/compose/

15. Gradle User Manual. URL: https://docs.gradle.org/current/userguide/userguide.html

6 ДОДАТОК А

7 ДОДАТОК Б

8 Програмний код роботи з api

```
class CryptoRepositoryImpl(
private val api: CryptoApi,
private val recentDao: RecentCoinsDao,
private val watchlistDao: WatchlistDao
) : CryptoRepository {

private val coinIdsBySymbol = mutableMapOf<String, String>()

override suspend fun getTopCoins(apiKey: String):
Result<List<AllCoinsListItem>> {
return try {
val response = api.getCoinsMarkets(perPage = 50)
response.forEach { coin ->
coinIdsBySymbol[coin.symbol.uppercase(Locale.US)] = coin.id
}

val coins = response.map { it.toAllCoinsListItem() }
Result.success(coins)
} catch (e: Exception) {
Result.failure(e)
}
}


override suspend fun getCoinDetail(symbol: String, apiKey:
String): Result<CoinDetail> {
return try {
val normalizedSymbol = symbol.uppercase(Locale.US)
val response = api.getCoinMarketBySymbol(symbol =
normalizedSymbol.lowercase(Locale.US))
response.forEach { coin ->
coinIdsBySymbol[coin.symbol.uppercase(Locale.US)] = coin.id
}

val coinDto = response.firstOrNull {
it.symbol.equals(normalizedSymbol, ignoreCase = true)
}
coinDto?.let { Result.success(it.toDetail()) }
?: Result.failure(Exception("Coin not found"))
} catch (e: Exception) {
Result.failure(e)
}
}


override suspend fun getHistoricalPrices(
symbol: String,
range: TimeRange,
apiKey: String
```

```
): Result<List<PricePoint>> {

return try {
val coinId = getCoinId(symbol)
val response = when (range) {
TimeRange.DAY ->
api.getMarketChart(id = coinId, days = "1")

TimeRange.WEEK ->
api.getMarketChart(id = coinId, days = "7")

TimeRange.MONTH ->
api.getMarketChart(id = coinId, days = "30")

TimeRange.MONTH3 ->
api.getMarketChart(id = coinId, days = "90")

TimeRange.YEAR ->
api.getMarketChart(id = coinId, days = "365")
}

Result.success(response.toPricePoints())

} catch (e: Exception) {
Result.failure(e)
}
}

private suspend fun getCoinId(symbol: String): String {
val normalizedSymbol = symbol.uppercase(Locale.US)
coinIdsBySymbol[normalizedSymbol]?.let { return it }

val coin = api.getCoinMarketBySymbol(symbol =
normalizedSymbol.lowercase(Locale.US))
.firstOrNull { it.symbol.equals(normalizedSymbol, ignoreCase =
true) }
?: throw IllegalStateException("Coin not found")

coinIdsBySymbol[normalizedSymbol] = coin.id
return coin.id
}


override suspend fun addRecent(coin: CoinListItem) {
recentDao.insertRecentCoin(coin.toRecentEntity())
recentDao.trimToFive()
}

override fun getRecent(): Flow<List<CoinListItem>> {
return recentDao.getRecentCoins()
.map { list -> list.map { it.toListItem() } }
}
```

```
override suspend fun addToWatchlist(coin: CoinListItem) {
watchlistDao.addToWatchlist(coin.toWatchlistEntity())
}

override fun getWatchlist(): Flow<List<CoinListItem>> {
return watchlistDao.getWatchlist()
.map { list -> list.map { it.toListItem() } }
}

override suspend fun removeFromWatchlist(symbol: String) {
watchlistDao.removeFromWatchlist(symbol = symbol)
}

override suspend fun isInWatchlist(symbol: String): Boolean {
return watchlistDao.isInWatchlist(symbol)
}
}
```

```
class CoinDetailViewModel(
private val cryptoRepository: CryptoRepository
) : ViewModel() {

private val _uiState = MutableStateFlow<DetailUiState<CoinDetail,
List<PricePoint>>>(DetailUiState.Loading)
val uiState = _uiState.asStateFlow()

val apiKey = API_KEY
private var cachedSymbol: String? = null
private var cachedCoinDetail: CoinDetail? = null

fun loadCoinDetail(symbol: String, range: TimeRange) {
viewModelScope.launch {
val cachedDetail = cachedCoinDetail.takeIf { cachedSymbol ==
symbol }

if (cachedDetail == null) {
_uiState.value = DetailUiState.Loading
}

val coinDetail = cachedDetail ?: cryptoRepository.getCoinDetail(
symbol = symbol,
apiKey = apiKey
).getOrElse {
_uiState.value = DetailUiState.Error(it.message ?: "Unknown
error")
return@launch
}
```

```
val historyPointsResult = cryptoRepository.getHistoricalPrices(
symbol = symbol,
range = range,
apiKey = apiKey
)

val historyPoints = historyPointsResult.getOrElse {
_uiState.value = DetailUiState.Error(it.message ?: "Unknown
error")
return@launch
}

cachedSymbol = symbol
cachedCoinDetail = coinDetail
_uiState.value = DetailUiState.Success(coinDetail, historyPoints)
}
}
}
```

Програмний код навігаційної структури

```
@Composable
fun NavigationGraph(navController: NavHostController) {
NavHost(
navController = navController,
startDestination = Loading,
) {
composable<Loading> {
LoadingScreen(
onLoadingFinished = {
val nextRoute = if (FirebaseAuth.getInstance().currentUser !=
null) {
Home
} else {
Authentication
}

navController.navigate(nextRoute) {
popUpTo(Loading) {
inclusive = true
}
}
}
)
}

composable<Home> {
HomeScreen(
onNavAllCurrenciesListScreen = {
navController.navigateToTopLevel(AllCoins)
```

```
},
onNavCoinDetailScreen = { symbol ->
navController.navigate(
CoinDetail(symbol)
)
},
onNavSettingsScreen = {
navController.navigateToTopLevel(Settings)
}
)
}

composable<AllCoins> {
AllCurrenciesListScreen(
onNavHomeScreen = {
navController.navigateToTopLevel(Home)
},
onNavCoinDetailScreen = { symbol ->
navController.navigate(
CoinDetail(symbol)
)
},
onNavSettingsScreen = {
navController.navigateToTopLevel(Settings)
}
)
}

composable<CoinDetail> { backStackEntry ->

val route =
backStackEntry.toRoute<CoinDetail>()

CoinDetailScreen(
coin_symbol = route.symbol,
onNavHomeScreen = {
navController.navigateToTopLevel(Home)
},
onNavAllCurrenciesListScreen = {
navController.navigateToTopLevel(AllCoins)
},
onNavSettingsScreen = {
navController.navigateToTopLevel(Settings)
}
)
}

composable<Authentication> {
AuthenticationScreen(
onAuthenticated = {
navController.navigate(Home) {
popUpTo(Authentication) {
```

```
inclusive = true
}
}
}
)
}

composable<Settings> {
SettingsScreen(
onNavAllCurrenciesListScreen = {
navController.navigateToTopLevel(AllCoins)
},
onNavHomeScreen = {
navController.navigateToTopLevel(Home)
},
onSignedOut = {
navController.navigate(Authentication) {
popUpTo(navController.graph.id) {
inclusive = true
}
}
}
)
}
}
}

private fun NavHostController.navigateToTopLevel(route: Any) {
navigate(route) {
popUpTo(Home) {
inclusive = false
}
launchSingleTop = true
}
}
```

Програмний код структури впровадження залежностей

```
val databaseModule = module {

single {
Room.databaseBuilder(
get(),
MainDb::class.java,
"crypto_database"
).build()
}

single { get<MainDb>().recentCoinsDao() }
single { get<MainDb>().watchlistDao() }
}
```

```
@OptIn(ExperimentalSerializationApi::class)
val networkModule = module {

single {
Json {
ignoreUnknownKeys = true
}
}

single {
val logger = HttpLoggingInterceptor().apply {
level = HttpLoggingInterceptor.Level.BODY
}

OkHttpClient.Builder()
.addInterceptor(logger)
.connectTimeout(15, TimeUnit.SECONDS)
.readTimeout(15, TimeUnit.SECONDS)
.build()
}

single {
val contentType = "application/json".toMediaType()
Retrofit.Builder()
.baseUrl(BASE_URL)
.addConverterFactory(get<Json>().asConverterFactory(contentType))
.client(get())
.build()
}

single {
get<Retrofit>().create(CryptoApi::class.java)
}
}


val repositoryModule = module {
single { FirebaseAuth.getInstance() }
single { FirebaseFirestore.getInstance() }

single<AuthRepository> {
FirebaseAuthRepository(firebaseAuth = get())
}

single<WatchlistRepository> {
FirestoreWatchlistRepository(
firestore = get(),
authRepository = get()
)
}
```

```
single<RecentRepository> {
FirestoreRecentRepository(
firestore = get(),
authRepository = get()
)
}

single<ThemePreferenceRepository> {
SharedPrefsThemePreferenceRepository(context = androidContext())
}

single<CryptoRepository> {
CryptoRepositoryImpl(
api = get(),
recentDao = get(),
watchlistDao = get(),
)
}
}


val viewModelModule = module {
viewModel { TestViewModel(get()) }
viewModel { CoinsListViewModel(get()) }
viewModel { CoinDetailViewModel(get()) }
viewModel { WatchlistViewModel(get()) }
viewModel { RecentViewModel(get()) }
viewModel { AuthenticationViewModel(get()) }
viewModel { SettingsViewModel(get()) }
viewModel { AppThemeViewModel(get()) }
}


class App : Application() {
override fun onCreate() {
super.onCreate()

startKoin {
androidContext(this@App)
modules(networkModule, viewModelModule, repositoryModule,
databaseModule)
}
}
}
```