ПОЛТАВСЬКИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТОРГІВЛІ
Навчально-науковий інститут денної освіти
Форма навчання денна
Кафедра комп’ютерних наук та інформаційних технологій

Допускається до захисту
Завідувач кафедри
_____ Олена ОЛЬХОВСЬКА
(підпис)

«___» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА

на тему

РОЗРОБКА ДОДАТКУ З МОНІТОРИНГУ ТА АНАЛІЗУ ДИНАМІКИ КРИПТОВАЛЮТНИХ РИНКІВ

зі спеціальності 122 Комп’ютерні науки
освітня програма «Комп’ютерні науки»
ступеня бакалавра

Виконавець роботи Медяник Євгеній Ігорович
_____ «____»_____ 2025 р.
(підпис)

Науковий керівник к.ф.-м.н., доцент, Ольховська Олена Володимирівна
_____ «____»_____ 2025 р.
(підпис)

Рецензент

ПОЛТАВА 2025 р.

ЗАТВЕРДЖУЮ
Завідувач кафедри _____Олена ОЛЬХОВСЬКА
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ І КАЛЕНДАРНИЙ ГРАФІК
ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

на тему «Розробка додатку з моніториннгу та аналізу динаміки криптовалютних ринків»
зі спеціальності 122 Комп'ютерні науки
освітня програма «Комп'ютерні науки»
ступеня бакалавр
Прізвище, ім'я, по батькові Медяник Євгеній Ігорович
Затверджена наказом ректора № 177-Н від «10» вересня 2024 р.
Термін подання студентом роботи «___»_____ 2025 р.
Вихідні дані до кваліфікаційної роботи: статті та документації з теми розробки мобільних додатків.
Зміст пояснювальної записки (перелік питань, які потрібно розробити)
ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Актуальність теми та мета дослідження
1.2. Основні задачі, що вирішуються у роботі
1.3. Визначення функціональних та нефункціональних вимог до мобільного додатку
1.4. Обґрунтування вибору платформи Android для реалізації
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Стан криптовалютного ринку та потреби в мобільному моніторингу
2.2. Аналіз існуючих мобільних додатків для моніторингу криптовалют
2.3. Підходи до аналізу фінансових даних за допомогою AI
2.4. Архітектурні підходи до розробки систем збору та обробки криптовалютних даних
2.5. Тенденції безпечної обробки даних в Android-розробці
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Обґрунтування вибору стеку технологій для реалізації додатку
3.2. Особливості інтеграції з API криптовалютних бірж
3.3. Архітектурний підхід Clean Architecture у поєднанні з MVVM та побудова блок схеми додатку
3.4. Побудова моделі даних, взаємодія з Room DB та Paging
3.5. Методологія впровадження AI-аналітики криптовалют
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Інтеграція з криптовалютними біржами через парсинг HTML
4.2. Розробка локальної бази даних для кешування даних

4.3.Реалізація взаємодії класів мобільного додатку з використанням патерну MVVM та діаграма класів
4.4. Реалізація функціоналу управління списком монет та інтеграція штучного інтелекту для аналізу криптовалют
4.5. Навігація та інтерфейс користувача
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А
Перелік графічного матеріалу: 1 аркуш блок-схем, 11 ілюстрацій.
Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали, посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Постановка задачі	Ольховська О.В.		
Інформаційний огляд	Ольховська О.В.		
Теоретична частина	Ольховська О.В.		
Практична частина	Ольховська О.В.		

Календарний графік виконання кваліфікаційної роботи

Зміст роботи	Термін виконання	Фактичне виконання
Вступ		
Вивчення методичних рекомендацій та стандартів та звіт керівнику		
Постановка задачі		
Інформаційний огляд джерел бібліотек та інтернету		
Теоретична частина		
Практична частина		
Закінчення оформлення		
Доповідь студента на кафедрі		
Доробка (за необхідністю), рецензування		

Дата видачі завдання «__» _____ 2024 р.
Здобувач вищої освіти _____ Медяник Євгеній Ігорович
(підпис)
Науковий керівник _____ к.ф.-м.н., доц. Ольховська О.В.
(підпис) (науковий ступінь, вчене звання, ініціали та прізвище)
Результати захисту кваліфікаційної роботи
Кваліфікаційна робота оцінена на _____
(балів, оцінка за національною шкалою, оцінка за ECTS)
Протокол засідання ЕК № ____ від «____» _____ 2025 р.
Секретар ЕК_____ _____
(підпис) (ініціали та прізвище)

Затверджую
Зав. кафедрою _____
к.ф.-м.н. Олена ОЛЬХОВСЬКА
«____» _____ 2024 р.

Погоджено
Науковий керівник _____
к.пед.н., Олена ОЛЬХОВСЬКА
«____» _____ 2024 р.

План

дипломного проекту з фаху
спеціальності 122 Комп'ютерні науки
освітня програма 122 Комп'ютерні науки
на тему «Розробка додатку з моніторингу та аналізу динаміки криптовалютних ринків»
Прізвище, ім'я, по батькові Медяник Євгеній Ігорович

ВСТУП
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ
1.1. Актуальність теми та мета дослідження
1.2. Основні задачі, що вирішуються у роботі
1.3. Визначення функціональних та нефункціональних вимог до мобільного додатку
1.4. Обґрунтування вибору платформи Android для реалізації
РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ
2.1. Стан криптовалютного ринку та потреби в мобільному моніторингу
2.2. Аналіз існуючих мобільних додатків для моніторингу криптовалют
2.3. Підходи до аналізу фінансових даних за допомогою AI
2.4. Архітектурні підходи до розробки систем збору та обробки криптовалютних даних
2.5. Тенденції безпечної обробки даних в Android-розробці
РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА
3.1. Обґрунтування вибору стеку технологій для реалізації додатку
3.2. Особливості інтеграції з API криптовалютних бірж
3.3. Архітектурний підхід Clean Architecture у поєднанні з MVVM та побудова блок схеми додатку
3.4. Побудова моделі даних, взаємодія з Room DB та Paging
3.5. Методологія впровадження AI-аналітики криптовалют
РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА
4.1. Інтеграція з криптовалютними біржами через парсинг HTML
4.2. Розробка локальної бази даних для кешування даних
4.3.Реалізація взаємодії класів мобільного додатку з використанням патерну MVVM та діаграма класів

4.4. Реалізація функціоналу управління списком монет та інтеграція штучного інтелекту для аналізу криптовалют
4.5. Навігація та інтерфейс користувача
ВИСНОВКИ
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ
ДОДАТОК А
Здобувач вищої освіти _____ Є.І. Медяник

(підпис)

«____» _____ 2024 р.

РЕФЕРАТ

Записка: 110 сторінок, 11 рисунків, 1 додаток, 21 літературне джерело.

Мета дипломної роботи полягає у розробці програмного забезпечення мобільного додатку для моніторингу та аналізу динаміки криптовалютних ринків.

Об'єкт розробки – мобільний додаток для моніторингу та аналізу криптовалют.

Методи дослідження – у процесі виконання роботи були застосовані сучасні підходи до проєктування та розробки Android-застосунків на основі патерну MVVM (Model-View-ViewModel) та принципів чистої архітектури. Отримання актуальних фінансових даних криптовалют реалізовано через парсинг HTML-сторінок біржі CoinMarketCap із використанням бібліотеки Jsoup у поєднанні з HTTP-клієнтом Retrofit. Збереження та кешування інформації організовано за допомогою бібліотеки Room, що виступає ORM-обгорткою над SQLite та дозволяє ефективно працювати з локальною базою даних. Реалізовано синхронізацію локальної інформації з сервером, а також підтримку пагінації великих списків активів із використанням Room Paging.

Особливістю розробленого програмного забезпечення є інтеграція генеративної моделі штучного інтелекту Gemini для проведення аналізу фінансових показників криптовалют та формування коротких висновків і рекомендацій для користувача. Взаємодія з AI-модулем здійснюється шляхом передачі параметрів криптовалюти у вигляді сформованих текстових запитів, після чого генеруються аналітичні висновки, які виводяться в інтерфейсі додатку.

У рамках дослідження було виконано наступні завдання: сформульовано ключові функціональні та нефункціональні вимоги до системи, які охоплюють потреби у стабільній роботі з великими обсягами фінансових даних, забезпеченні продуктивності, масштабованості та зручності користувацького інтерфейсу. Проведено детальний аналіз існуючих рішень для моніторингу

криптовалютних ринків, що дозволило обґрунтувати доцільність розробки власного рішення та визначити оптимальний стек технологій. Обґрунтовано вибір архітектурного підходу Clean Architecture, що забезпечує чіткий поділ відповідальностей між рівнями системи, а також застосування MVVM для спрощення управління станом додатку та його тестованості.

Розроблено повний стек програмної логіки, включаючи класи моделей, репозиторії, використання DAO для доступу до бази даних, класи бізнес-логіки (use-cases), ViewModel-компоненти, а також адаптери для роботи зі списками монет. Реалізовано багаторівневу синхронізацію даних, де при кожному оновленні відбувається запит на сервер, парсинг HTML-документів CoinMarketCap, вилучення актуальних фінансових показників та оновлення локальної бази Room.

Впроваджено інтеграцію генеративної аналітики за допомогою AI-модуля Gemini, що дозволяє у режимі реального часу проводити аналіз основних фінансових показників криптовалют та формувати рекомендації для користувача у компактній текстовій формі. Побудовано систему передачі даних між компонентами за допомогою DI-фреймворку Koin, що забезпечило слабкий зв’язок між модулями та підвищило тестованість програмного забезпечення.

Розроблено сучасний користувацький інтерфейс згідно стандартів Material Design з реалізацією зручної навігації, пагінації, підтримкою swipe-to-refresh та відображенням детальної інформації для кожного активу. Створено повний набір діаграм класів, блок-схем, а також архітектурних моделей, які відображають логіку роботи додатку та взаємодію його компонентів.

Проведено повний цикл тестування мобільного додатку, що підтвердило стабільність, коректність роботи, продуктивність і безпечність розробленої системи. Таким чином, створено програмний продукт, який може виступати основою для подальшого вдосконалення систем фінансового моніторингу з інтеграцією штучного інтелекту у процес аналізу криптовалютних ринків.

ЗМІСТ

ВСТУП

У сучасному світі цифрової економіки та стрімкого розвитку інформаційних технологій криптовалютні ринки стають невід'ємною частиною глобальної фінансової системи. Висока волатильність, децентралізована структура та постійні зміни ринкової капіталізації формують потребу в оперативному доступі до актуальних даних та аналітичної інформації. З огляду на це, мобільні застосунки є зручним та ефективним інструментом для моніторингу фінансових показників, оскільки забезпечують швидкість, доступність та персоналізовану взаємодію з користувачем.

Разом з тим, розробка таких застосунків супроводжується низкою викликів, серед яких забезпечення актуальності отриманих даних, їх локальне збереження, стабільна робота з біржовими джерелами інформації та інтеграція сучасних методів аналізу, зокрема із застосуванням штучного інтелекту. Важливим аспектом є також вибір правильної архітектури для забезпечення модульності, масштабованості та легкості підтримки програмного продукту.

Метою даної роботи є розробка мобільного Android-додатку для моніторингу та аналізу динаміки криптовалютних ринків, що дозволяє отримувати інформацію про криптовалюти з відкритих джерел, зберігати її в локальній базі даних, відображати користувачеві у зручному форматі та генерувати аналітичні висновки з використанням моделей штучного інтелекту.

Об'єктом дослідження є мобільний додаток для моніторингу та аналізу криптовалют, а предметом дослідження - процес проєктування та реалізації програмного забезпечення для моніторингу фінансових показників з використанням патернів архітектури та AI-моделей.

Для досягнення поставленої мети необхідно вирішити ряд завдань:

– провести аналіз існуючих рішень для моніторингу криптовалют та визначити їх функціональні переваги й обмеження;

- визначити ключові функціональні вимоги до мобільного застосунку;
- реалізувати механізм збору даних із біржових джерел шляхом парсингу HTML за допомогою бібліотеки Jsoup;
- забезпечити локальне зберігання даних за допомогою бази Room з підтримкою пагінації та оновлення;
- впровадити можливість створення персонального списку монет із повною деталізацією кожної позиції;
- інтегрувати генеративну AI-модель (Gemini) для формування висновків щодо динаміки кожної криптовалюти;
- побудувати архітектуру застосунку на основі шаблону MVVM із використанням Koin для ін’єкції залежностей;
- протестувати реалізовану систему на предмет відповідності функціональним та нефункціональним вимогам.

Методи дослідження включають застосування архітектурного підходу Clean Architecture, шаблону MVVM, бібліотек Retrofit, Room, Jsoup, Picasso, а також використання генеративної моделі штучного інтелекту для обробки текстових запитів.

У результаті розробки передбачається отримання повнофункціонального інструменту, який може бути використаний як аналітичний додаток для користувачів, зацікавлених у відстеженні та прогнозуванні змін криптовалютного ринку.

Пояснювальна записка включає чотири основні розділи: постановку задачі, огляд сучасного стану проблеми, теоретичне обґрунтування та практичну реалізацію програмного продукту. Така структура дозволяє логічно викласти процес дослідження та реалізації мобільного застосунку, акцентуючи увагу на актуальності теми, технологічних рішеннях, особливостях архітектури та функціональному наповненні.

.

РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ

1.1. Актуальність та мета дослідження

У сучасних умовах активного розвитку цифрових фінансів та децентралізованих економічних систем криптовалютні ринки посідають важливе місце в глобальній економіці. Щоденна динаміка вартості цифрових активів, висока волатильність та обсяги торгів вимагають оперативного доступу до актуальної інформації та ефективних засобів її аналізу. У зв’язку з цим виникає необхідність створення зручних мобільних інструментів, які забезпечують не лише моніторинг цінових показників, а й аналітичну обробку отриманих даних з метою підтримки прийняття рішень.

Актуальність обраної теми також обумовлена відсутністю на ринку універсальних мобільних застосунків, що поєднують простоту інтерфейсу, високий рівень автоматизації збору даних та інтелектуальний аналіз на основі сучасних моделей штучного інтелекту. Додатково, наявні рішення здебільшого залежать від централізованих API, які не завжди забезпечують повну чи своєчасну інформацію, що знижує їхню ефективність в умовах високих вимог до точності та швидкодії [7, 18].

З огляду на це, метою даного дослідження є розробка Android-застосунку для моніторингу та аналізу динаміки криптовалютних ринків, який реалізує:

- парсинг HTML-даних безпосередньо з біржових сторінок CoinMarketCap (за допомогою бібліотеки Jsoup);
- обробку й збереження фінансових показників у локальній базі даних (Room);
- використання шаблону MVVM та принципів Clean Architecture для забезпечення масштабованості;
- застосування моделі Generative AI (Gemini) для генерації стислих аналітичних висновків щодо конкретних активів;

- реалізацію гнучкої навігації (Navigation Component) і сучасного адаптивного UI з використанням Material Design [7, 18, 20].

Таким чином, дослідження спрямоване на вирішення практичної задачі створення інтелектуального мобільного сервісу для криптоентузіастів, інвесторів і дослідників цифрових фінансів, що забезпечує поєднання надійного функціоналу, інтуїтивного інтерфейсу та розширеної аналітики.

1.2. Основні задачі, що вирішуються в роботі

У межах даного дослідження вирішуються задачі, спрямовані на створення функціонального, надійного та зручного у використанні мобільного програмного забезпечення для моніторингу та аналізу динаміки криптовалютних ринків. Формулювання задач ґрунтується на вимогах до сучасних мобільних застосунків, принципах інженерії програмного забезпечення, а також технологічних можливостях Android-платформи [9-12, 18, 20].

До основних задач, реалізація яких забезпечує досягнення мети дослідження, належать:

- аналіз функціональних характеристик та архітектурних рішень у вже існуючих мобільних застосунках для відстеження криптовалют, визначення їх переваг та обмежень;
- формулювання функціональних та нефункціональних вимог до мобільного застосунку, з урахуванням потреб користувачів, специфіки криптовалютного ринку та вимог до безпеки даних;
- побудова загальної архітектури системи на основі принципів чистої архітектури (Clean Architecture) з використанням шаблону MVVM (Model–View–ViewModel) для досягнення модульності та масштабованості коду;

- реалізація механізмів збору даних з публічних джерел шляхом парсингу HTML-сторінок CoinMarketCap з використанням бібліотеки Jsoup та організація взаємодії з серверними API через Retrofit;
- створення моделей даних, які відображають основні характеристики криптовалютних активів, і розробка механізмів локального збереження даних у базі Room;
- впровадження можливості управління обраними монетами, включаючи додавання, перегляд та видалення записів;
- розробка інтерфейсу користувача з використанням компонентів Material Design, забезпечення підтримки пагінації та динамічного оновлення інформації;
- інтеграція генеративної моделі штучного інтелекту (Gemini) для формування стислих текстових аналітичних висновків та рекомендацій щодо ринкової динаміки конкретних активів;
- організація внутрішньої логіки застосунку на основі Koin як інструменту для ін’єкції залежностей, що сприяє гнучкості та зменшенню зв’язності компонентів;
- забезпечення засобів тестування функціональності застосунку на рівні логіки (через JUnit) та інтерфейсу користувача (з використанням Espresso).

У сукупності реалізація зазначених задач дозволяє створити сучасний програмний продукт, здатний задовольнити запити користувачів щодо ефективного відстеження й аналітики стану криптовалютного ринку в мобільному форматі.

1.3. Визначення функціональних та нефункціональних вимог до мобільного додатку

Визначення вимог до мобільного додатку є одним із ключових етапів процесу розробки програмного забезпечення. Якісна формалізація вимог дозволяє сформувати чітке уявлення про очікувану поведінку системи, забезпечити відповідність очікуванням користувача та технічним обмеженням платформи. У межах даної роботи вимоги поділяються на функціональні — що визначають безпосередні дії та можливості додатку — та нефункціональні, які описують характеристики якості програмного продукту.

Функціональні вимоги до мобільного додатку передбачають реалізацію комплексу можливостей, що забезпечують повноцінне отримання, обробку, збереження та аналіз інформації про криптовалюти. Однією з ключових функцій є доступ до актуальних ринкових даних, що реалізується через отримання HTML-контенту з криптовалютної біржі CoinMarketCap та подальше вилучення таких фінансових показників, як ринкова ціна, капіталізація, обсяг торгів та обсяг пропозиції. Парсинг і обробка даних здійснюються за допомогою бібліотеки Jsoup, що дозволяє розбирати структуру HTML-документів та формувати відповідні внутрішні структури даних, представлені у вигляді об'єктів моделей CryptoCoinModel та CryptoCurrencyModel.

Для забезпечення збереження інформації у локальному середовищі передбачено використання бази даних Room, яка виступає засобом кешування отриманих відомостей про монети. При цьому підтримується як автоматичне, так і ручне оновлення даних, що підвищує гнучкість та адаптивність додатку. Користувачі мають можливість керувати персональним списком монет, зокрема додавати нові активи, перевіряти їх наявність у локальній базі, переглядати розширену інформацію щодо кожної криптовалюти, а також за потреби видаляти раніше додані елементи.

Інтеграція з інтелектуальними аналітичними системами реалізована через генерацію коротких висновків на основі фінансових показників за допомогою моделі Gemini, результати роботи якої виводяться в інтерфейс користувача.

Графічна складова додатку побудована на компонентах Material Design, що дозволяє забезпечити сучасний та зручний інтерфейс. Навігація між функціональними елементами додатку реалізована з використанням Navigation Component. Візуалізація списків криптовалют здійснюється у вигляді адаптивного RecyclerView з підтримкою механізмів пагінації та оновлення шляхом swipe-to-refresh.

Серед нефункціональних вимог слід виокремити насамперед масштабованість, що досягається завдяки використанню архітектурних принципів Clean Architecture та шаблону MVVM, які забезпечують можливість розширення додатку без порушення існуючої логіки. Модульність системи підтримується через використання бібліотеки Koin, яка дозволяє реалізувати ін’єкцію залежностей і підвищує незалежність між компонентами, що спрощує тестування. Продуктивність забезпечується оптимізацією механізмів обробки біржових даних та використанням кешування результатів у базі, що мінімізує кількість мережевих запитів.

Зручність використання досягається завдяки інтуїтивно зрозумілому інтерфейсу, який відповідає рекомендаціям Google щодо дизайну мобільних додатків. Надійність системи ґрунтується на коректній обробці можливих помилок при обміні даними з мережею та при парсингу, а також на реалізації механізмів валідації та захисту від дублювання. Важливим аспектом є також забезпечення безпеки: обмеження доступу до функцій редагування списку монет і реалізація заходів щодо збереження цілісності локальних даних у разі збоїв або зовнішнього втручання.

Підсумовуючи викладене, до основних вимог до мобільного додатку належать:

Функціональні вимоги:

- отримання та парсинг HTML-даних з CoinMarketCap;
- формування структурованих моделей криптовалют;
- збереження даних у локальній базі з можливістю оновлення;

- управління списком обраних монет (додавання, перегляд, видалення);
- генерація аналітичних висновків за допомогою AI-моделі Gemini;
- побудова сучасного UI з підтримкою пагінації та swipe-to-refresh;
- реалізація навігації між екранами додатку.

Нефункціональні вимоги:

- масштабованість архітектури (MVVM, Clean Architecture);
- модульність і підтримка тестування (ін’єкція залежностей через Koin);
- оптимізація продуктивності за рахунок кешування;
- зручність та інтуїтивність інтерфейсу;
- стабільність роботи при обробці помилок;
- захист локальних даних та контроль доступу.

Таким чином, сформульовані вимоги забезпечують комплексне бачення необхідного функціоналу додатку та визначають критерії його ефективності, що дозволяє перейти до етапу системного проєктування і реалізації.

1.4. Обґрунтування вибору платформи Android для реалізації

Вибір платформи для реалізації мобільного додатку є критичним етапом розробки програмного забезпечення, що визначає технологічні обмеження, інструментальні засоби, потенційне охоплення користувачів та особливості розгортання застосунку. У межах даної роботи в якості цільової платформи було обрано операційну систему Android, що обумовлено низкою об’єктивних технічних, економічних та ринкових чинників.

По-перше, Android є домінуючою мобільною платформою у світі, з часткою понад 70 % глобального ринку смартфонів. Така популярність

забезпечує широкий потенціал охоплення користувацької аудиторії, включаючи як професійних трейдерів, так і приватних інвесторів, які використовують мобільні пристрої для оперативного моніторингу фінансових даних.

По-друге, відкритість платформи Android та наявність потужного інструментарію розробки (Android SDK, Android Studio, Jetpack-компоненти) дозволяє реалізовувати як стандартні, так і інноваційні функції мобільних застосунків. Зокрема, проєкт передбачає використання бібліотек Retrofit, Jsoup, Room, Koin, Navigation Component, Material Design, які є нативно підтримуваними в Android-середовищі.

По-третє, гнучкість у побудові архітектури застосунку є важливою перевагою Android. У даному проєкті реалізовано архітектуру Clean Architecture з використанням шаблону MVVM, що передбачає чітке розділення шарів відповідальності (presentation, domain, data) і є рекомендованим підходом для довготривалих, масштабованих проєктів у середовищі Android.

По-четверте, можливість ефективної роботи з мережевими джерелами даних — важлива умова для розробки додатку, який здійснює парсинг HTML-сторінок та звернення до API криптовалютних бірж. Android-платформа забезпечує повну підтримку асинхронного програмування за допомогою корутин та Flow, що реалізовано у відповідних use case застосунку.

По-п’яте, можливість реалізації UI/UX-дизайну згідно з сучасними стандартами забезпечується наявністю Material Components та підтримкою адаптивного інтерфейсу. Це дозволяє створити зручний і привабливий застосунок, який відповідає очікуванням сучасного користувача.

Крім того, Android-платформа забезпечує зручність розгортання і тестування, включаючи можливість автоматизованого тестування (JUnit, Espresso) і використання емуляторів різних типів пристроїв без необхідності додаткового ліцензування.

Таким чином, вибір платформи Android для реалізації програмного продукту є обґрунтованим з точки зору технічної доцільності, доступності

інструментів, широти охоплення користувачів та відповідності специфіці розв’язуваних задач.

РОЗДІЛ 2. ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

2.1. Стан криптовалютного ринку та потреби в мобільному моніторингу

Криптовалютний ринок сьогодні є однією з найінтенсивніше зростаючих галузей цифрової економіки. З моменту появи першої криптовалюти Bitcoin у 2009 році екосистема цифрових активів зазнала значного розвитку як з точки зору технологічної складності, так і з огляду на обсяги капіталу, задіяного у відповідних транзакціях. Станом на 2024 рік загальна ринкова капіталізація криптовалют перевищує 2 трлн доларів США, а щоденні обсяги торгів на великих біржах, таких як Binance, Coinbase, Kraken чи OKX, обчислюються десятками мільярдів доларів.

Криптовалюти, зокрема Bitcoin (BTC), Ethereum (ETH), Solana (SOL), Cardano (ADA) та інші, активно використовуються не лише як спекулятивний актив, а й у якості засобу збереження вартості, розрахунків, інструменту для смартконтрактів, фінансування стартапів (ICO/IEO), а також як складова частина Web 3.0-екосистеми. Разом з тим, ринок характеризується високим ступенем динамічності: значні коливання вартості активів можуть відбуватися упродовж кількох хвилин під впливом новинного фону, технічних збоїв або дій великих інвесторів. Наприклад, лише за одну добу курс Bitcoin може зрости або впасти на 5–10 %, що створює передумови для високих прибутків або втрат залежно від вчасності реакції трейдера чи інвестора [7, 18].

У таких умовах критично важливим є безперервний моніторинг ринкових даних у реальному часі. Користувачі потребують інструментів, які дозволяють отримувати не лише поточну вартість цифрових активів, а й супровідні метрики, зокрема обсяг торгів за 24 години, зміну ціни у відсотках, ринкову капіталізацію, співвідношення обігової та загальної емісії, показники ліквідності тощо. Більшість професійних платформ, як-от CoinMarketCap,

CoinGecko чи TradingView, надають ці дані у веб-інтерфейсі, однак цього часто недостатньо для користувачів, які очікують постійного доступу до інформації в мобільному форматі [7, 18].

Мобільні застосунки у цьому контексті виступають ефективним інструментом доступу до криптовалютних даних. Вони дозволяють реалізувати принцип «завжди на зв'язку», що є особливо важливим у випадку спонтанних змін ринку. Наприклад, у випадку раптової новини про злам великої біржі або затвердження нормативного регулювання цифрових активів у певній країні, ціни можуть змінитися миттєво. В таких випадках можливість оперативно відслідковувати ситуацію через мобільний застосунок, отримувати автоматичні оновлення і швидко приймати рішення про купівлю чи продаж є вирішальною.

Окрім базового перегляду котирувань, сучасні користувачі очікують розширеного функціоналу, зокрема:

– можливості створення індивідуального списку спостереження (watchlist), що дозволяє відстежувати лише обрані монети;

– збереження даних у локальній базі для офлайн-доступу;

– графічного відображення змін ціни та історичних даних;

– інтеграції інструментів аналізу на основі штучного інтелекту, які можуть формулювати рекомендації або попередження щодо ризиків [7, 18]..

Важливим аспектом є також аналітична складова мобільного моніторингу, яка дає змогу не лише фіксувати поточний стан активів, а й прогнозувати їхню динаміку. Інтеграція зі штучним інтелектом або машинним навчанням, як це реалізовано в межах даного проєкту за допомогою моделі Gemini, дозволяє здійснювати автоматичний аналіз ринкових метрик з генерацією стислих рекомендацій щодо доцільності інвестицій, потенційних ризиків або очікуваних змін. Такий підхід значно підвищує цінність мобільного застосунку як інструменту прийняття рішень.

На практиці існуючі рішення мають низку обмежень. Деякі з них потребують платної підписки для доступу до розширених функцій, інші — не

підтримують локальне кешування або не забезпечують достатню гнучкість у формуванні списків монет. Значна частина застосунків не має AI-модуля аналітики або не дозволяє зручно фільтрувати та сортувати інформацію за ключовими параметрами [7, 18]..

Таким чином, з урахуванням високих темпів розвитку криптовалютного ринку, його нестабільності та глобального характеру, необхідність у надійному, функціонально повному та інтелектуально орієнтованому мобільному рішенні є цілком обґрунтованою. Це зумовлює актуальність створення мобільного застосунку, який не лише забезпечує оперативний доступ до крипторинку, а й інтегрує механізми збереження, аналітики та взаємодії з користувачем через адаптивний інтерфейс і сучасні архітектурні рішення.

2.2. Аналіз існуючих мобільних додатків для моніторингу криптовалют

Розвиток криптовалютного ринку безпосередньо зумовив появу великої кількості спеціалізованих мобільних застосунків, які надають користувачам можливість моніторингу стану ринку, здійснення аналітики та управління портфелем цифрових активів. Кожен із таких додатків має свою специфіку функціонування, архітектуру, набір функціональних можливостей, відмінності в дизайні інтерфейсу, безпекових механізмах та орієнтації на цільову аудиторію. У зв’язку з цим доцільним є здійснення порівняльного аналізу найбільш розповсюджених рішень, що функціонують на ринку мобільних додатків для моніторингу криптовалют.

Однією з найбільш відомих платформ у цій галузі є мобільний застосунок CoinMarketCap, який за багатьма параметрами є еталонним для великої частини

ринку.

Рисунок 2.1. Лого застосунку CoinMarketCap

Додаток пропонує користувачам доступ до великої кількості актуальних ринкових даних, включаючи інформацію про більше ніж 10 тисяч криптовалютних активів. Він забезпечує відображення основних фінансових метрик, таких як поточна ціна, ринкова капіталізація, добовий обсяг торгів, зміну ціни за певний період, співвідношення між обіговою та максимальною емісією, динаміку ціни у вигляді графіків з різними інтервалами часу. Крім базової інформації, додаток містить новинні розділи, календар ICO/IDO, аналітичні огляди та інші сервіси інформаційної підтримки. Окремою перевагою є наявність watchlist-функціоналу, який дозволяє користувачам формувати персоналізовані списки монет для постійного моніторингу. Водночас функціонал додатку фактично обмежується агрегованою подачею даних, що ускладнює його використання для глибшого індивідуального аналізу. Додатково, повна функціональність окремих інструментів потребує створення облікового запису та авторизації користувача.

Ще одним вагомим учасником ринку є мобільний застосунок **CoinGecko**, який надає порівняльний за обсягом набір даних із ширшою деталізацією вторинних показників, таких як обсяг заблокованої ліквідності (TVL), індекси домінування ринку певних монет, соціальна активність криптоактивів, показники розробницької активності за допомогою GitHub-статистики.

Рисунок 2.2 Застосунок CoinGecko

Застосунок CoinGecko активно використовується інституційними аналітиками завдяки більш глибокій деталізації. Архітектурно застосунок побудований з орієнтацією на великі обсяги даних, однак не містить елементів автоматизованої аналітики або використання штучного інтелекту для формування прогнозних висновків. Акцент робиться на об’єктивній подачі максимально повної інформації з мінімальним залученням суб’єктивних оцінок. У той же час відсутність вбудованої AI-аналітики створює обмеження для частини користувачів, які очікують отримання не лише сирих даних, а й їх попередньої інтерпретації.

Важливим сегментом мобільних додатків є також продукти біржових платформ, таких як Binance, Kraken, Coinbase та інші.

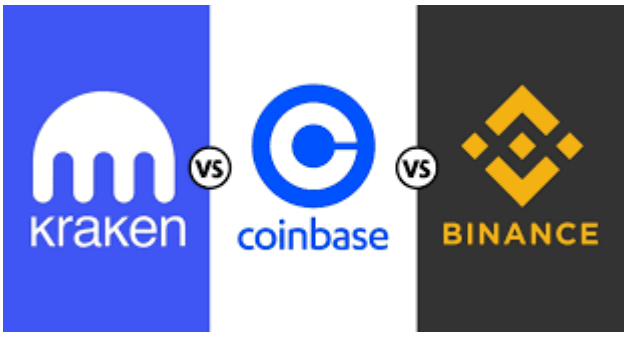

Рисунок 2.3. Лого біржових платформ Binance, Kraken, Coinbase

Ці додатки інтегрують у собі функціонал не лише моніторингу, а й здійснення фінансових операцій з купівлі, продажу, стейкінгу, лендінгу криптовалютних активів. Сервіси забезпечують реальний доступ до біржових торгів, супроводжуються модулями графічного технічного аналізу та

вбудованими торговими ботами. Особливістю біржових додатків є прив’язка до власної інфраструктури з використанням внутрішніх API, що дозволяє мінімізувати затримку оновлення інформації, але водночас обмежує спектр моніторингу переважно активами, які присутні безпосередньо на даній торговій платформі. З точки зору аналітики ринку в цілому такі додатки є менш придатними, оскільки їх основна функція полягає в реалізації торгових операцій, а не в універсальному моніторингу стану всієї ринкової екосистеми.

Окрему групу мобільних рішень формують незалежні агрегатори, які надають додаткову аналітику та функціонал управління портфелем. До них, зокрема, належать такі продукти як Blockfolio (тепер FTX App), Delta, Crypto Pro та інші. Вони дозволяють не лише отримувати ринкові дані, а й відслідковувати особистий інвестиційний портфель, відображати історію угод, надавати сповіщення про цінові зміни за встановленими порогами, синхронізуватись із гаманцями та біржами через API-з’єднання. Багато таких додатків інтегрують графічні інструменти аналізу, інформують користувача про розподіл активів за класами, проводять прості статистичні оцінки прибутковості інвестицій. Водночас їх функціонал прогнозної аналітики зазвичай обмежений або зовсім відсутній. Дані застосунки більше орієнтовані на управління власним портфелем, а не на детальну аналітичну оцінку стану ринку.

Варто відзначити, що попри розмаїття функціоналу сучасних додатків, більшість із них не пропонують користувачам систем автоматизованого аналізу ринкових даних на основі моделей штучного інтелекту. Здебільшого аналітична інтерпретація фінансових показників покладається на самостійні дії користувачів або пропонується у вигляді загальних новинних зведень без індивідуальної оцінки ризиків і перспектив окремих активів. Це створює об’єктивну нішу для розробки застосунків нового покоління, які поєднують в собі збір, обробку та інтелектуальну інтерпретацію ринкової інформації в єдиному середовищі мобільної взаємодії.

Крім цього, слід зазначити, що значна частина існуючих додатків функціонує на базі централізованих API-платформ. Це передбачає постійний обмін даними з віддаленими серверами, що підвищує навантаження на мережеві ресурси та створює додаткову залежність від стабільності зовнішніх джерел інформації. У разі перебоїв у роботі API-сервісів, обмеження швидкості запитів або зміни умов доступу користувачі таких додатків можуть втрачати доступ до актуальної інформації. Саме тому підхід, що передбачає парсинг HTML-контенту безпосередньо з відкритих біржових сторінок, може виступати альтернативною стратегією отримання даних, дозволяючи частково знизити залежність від сторонніх API.

Таблиця 1.

Аналіз переваг існуючих рішень

Характеристика	**CoinMarketCap**	**CoinGecko**	**Binance App**	**Delta**	**Blockfolio (FTX App)**	**Запропонований додаток**
Джерела даних	Власний API CoinMarketCap	Власний API CoinGecko	Власна біржа Binance	Агреговані API	API бірж та гаманців	Парсинг HTML CoinMarketCap + API
Кількість активів	10 000+	12 000+	500+ (лише Binance)	7 000+	10 000+	Необмежено (усі доступні на CoinMarketCap)
Персональні списки монет	Так	Так	Так	Так	Так	Так
Локальне кешування даних	Обмежене	Обмежене	Так	Так	Так	Повноцінне локальне кешування через Room
Оновлення даних в реальному часі	Так	Так	Так	Так	Так	Так
Портфель управління	Обмежене	Обмежене	Повне	Повне	Повне	Часткове (список обраних монет)
Аналітика з використанням AI	Немає	Немає	Немає	Немає	Немає	Є (AI-модель Gemini)
Додаткові сервіси	Новини, ICO, календари	Соціальні метрики, Github-статистика	Торгівля, стейкінг, трейдинг-боти	Портфельна статистика	Портфельна статистика, API інтеграції	Генерація AI-рекомендацій
Архітектура додатку	Моноліт	Моноліт	Моноліт	Моноліт	Моноліт	MVVM + Clean Architecture
Можливість	Обмежена	Обмежен	Частков	Частково	Частково	Є (локальне

роботи офлайн		а	о			кешування Room)

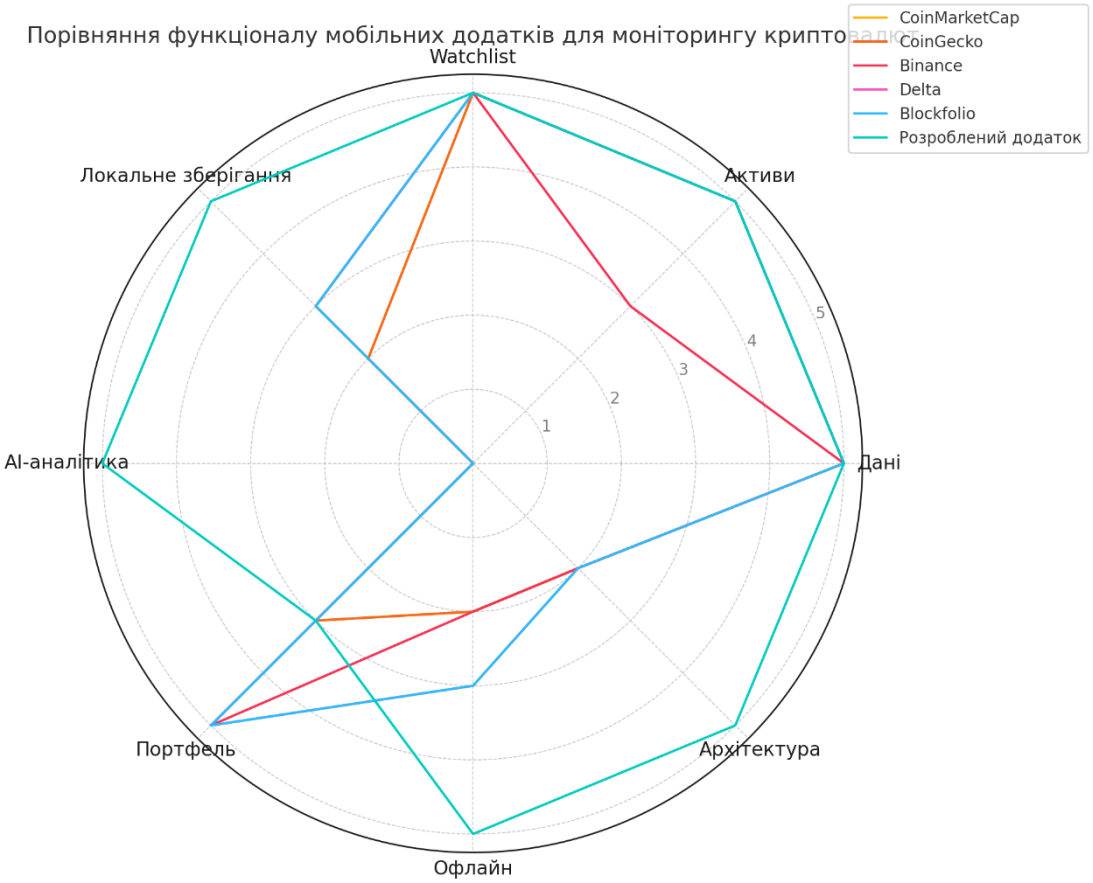

Рисунок 2.4. Діаграма порівняння функціоналу мобільних додатків

Таким чином, сучасний ринок мобільних застосунків для моніторингу криптовалют характеризується високим рівнем фрагментованості функціоналу. Окремі рішення орієнтовані на вузькоспеціалізовані задачі управління портфелем, інші — на інформаційну агрегацію, ще інші — на здійснення торгових операцій. У переважній більшості випадків відсутня глибока автоматизована аналітика, інтеграція з AI-сервісами та локальне кешування великих обсягів даних для роботи в офлайн-режимі. Це створює актуальну потребу у створенні комплексного рішення, яке поєднує в собі як гнучкі засоби збору й обробки інформації, так і елементи автоматизованої інтерпретації ринкових процесів із застосуванням моделей штучного інтелекту. Саме на такі задачі орієнтовано розроблюваний у межах даної роботи мобільний додаток.

2.3. Підходи до аналізу фінансових даних за допомогою AI

Розвиток методів штучного інтелекту (AI) значно трансформував підходи до обробки, аналізу та інтерпретації фінансових даних. Особливо актуальним це є у сфері криптовалютних ринків, які характеризуються високою складністю, нелінійною динамікою та великою кількістю факторів, що впливають на коливання вартості активів. Традиційні методи фінансового аналізу, засновані на математичній статистиці та класичних економетричних моделях, часто є недостатньо ефективними в умовах високої волатильності, інформаційного шуму та багатофакторності ринку. У цьому контексті технології штучного інтелекту, зокрема машинне навчання, глибоке навчання та моделі генеративного AI, відкривають нові можливості для підвищення точності аналізу та формування обґрунтованих інвестиційних рекомендацій.

Одним із основних напрямів застосування AI у фінансовій аналітиці є прогнозування цінових коливань активів. Використання моделей машинного навчання дозволяє виявляти складні нелінійні залежності між різними фінансовими показниками, що є складно доступним для класичних моделей. Зокрема, застосовуються методи регресійного аналізу, дерева рішень, ансамблеві методи (Random Forest, Gradient Boosting), рекурентні нейронні мережі (LSTM, GRU), які здатні обробляти часові ряди даних, враховуючи їхню історичну динаміку. Наприклад, при прогнозуванні вартості криптовалют часто використовуються часові ряди змін цін, обсягу торгів, глибини ринку, а також дані новинних стрічок та соціальних мереж як додаткові інформаційні джерела [6, 10-12, 15-16].

Важливим завданням, яке успішно вирішується із залученням AI, є також класифікація ринкових станів та оцінка ризиків. Методи класифікації

дозволяють ідентифікувати фази ринку (зростання, спад, фаза консолідації), виявляти аномальні стани та формувати сигнали щодо потенційних змін ринкових тенденцій. Для цього широко застосовуються моделі Support Vector Machine (SVM), нейронні мережі багатьох шарів (MLP), а також сучасні трансформерні архітектури, що дозволяють інтегрувати багатоджерельні вхідні дані різної природи [6, 10-12, 15-16].

Особливого розвитку набули генеративні моделі, зокрема моделі великих мовних моделей (LLM), які демонструють здатність до аналізу текстових, числових та комбінованих фінансових даних. Генеративний AI здатний формулювати короткі аналітичні висновки, надавати узагальнені рекомендації на основі комплексного аналізу даних, зменшуючи інформаційне навантаження на користувача та полегшуючи процес прийняття рішень. Саме такий підхід використовується у розробленому в межах даної роботи мобільному додатку, де реалізовано інтеграцію моделі Gemini, що генерує текстові висновки на основі поточних ринкових метрик окремих криптовалют.

Застосування AI у фінансовому аналізі також охоплює обробку нефінансових даних, що можуть впливати на ринкову динаміку. Наприклад, аналіз новинних потоків, соціальних мереж, публікацій експертів, заголовків статей та інших неструктурованих джерел інформації дозволяє формувати більш точні прогнози на основі комплексного урахування як фундаментальних, так і поведінкових факторів. Застосування методів обробки природної мови (Natural Language Processing, NLP) та семантичного аналізу текстів дозволяє системам AI оцінювати ринкові очікування, ідентифікувати позитивні або негативні настрої, а також оперативно реагувати на появу важливих новин.

Серед важливих переваг залучення AI до фінансового аналізу слід також відзначити здатність моделей самонавчатися на основі нових даних, що надходять, адаптуючись до змін ринкових умов у реальному часі. При цьому система зберігає здатність оновлювати свої внутрішні моделі прогнозування без необхідності повного ручного втручання розробників. Така властивість є

особливо цінною на ринку криптовалют, де фактори нестабільності змінюються швидко й непередбачувано [6, 10-12, 15-16].

Разом із тим, впровадження AI у фінансові аналітичні системи супроводжується низкою викликів. До них відносяться питання якості вхідних даних, необхідність регулярної актуалізації моделей, контроль над помилками моделі, уникнення ефектів переобучення, забезпечення прозорості прийнятих рішень та стабільності роботи систем в умовах високої ринкової волатильності. Також важливою є проблема етичної відповідальності при використанні рекомендацій AI у сфері фінансових рішень, оскільки модель не несе юридичної відповідальності за надані прогнози.

Таким чином, сучасні підходи до аналізу фінансових даних із використанням штучного інтелекту забезпечують істотні переваги в обробці великих обсягів ринкової інформації, її систематизації, оперативній оцінці ризиків і формуванні аналітичних висновків для підтримки прийняття інвестиційних рішень. Інтеграція генеративних AI-моделей у мобільні застосунки, такі як розроблений у межах даної роботи додаток, дозволяє перейти від традиційного пасивного моніторингу даних до активного аналітичного супроводу користувача із можливістю швидкого формування висновків навіть при значному інформаційному перевантаженні ринку криптовалют [6, 10-12, 15-16].

Таблиця 2

Можливості впровадження AI у фінансові аналітичні системи

Напрямок застосування	**Алгоритмічні методи**	**Сфери використання**	**Переваги**	**Обмеження**
Прогнозування цінових трендів	Лінійна регресія, Random Forest, Gradient Boosting, LSTM, GRU	Прогноз зміни курсу, волатильності	Врахування історичних патернів, часових залежностей	Високі вимоги до обсягу та якості історичних даних
Класифікація ринкових станів	Support Vector Machines, нейронні мережі MLP, Decision Trees	Визначення фаз ринку, виявлення аномалій	Виявлення прихованих закономірностей	Може давати помилкові класифікації при зміні ринкових умов
Обробка текстових даних (NLP)	BERT, GPT, Transformers, Word2Vec	Аналіз новин, соцмереж, заголовків	Урахування інформаційних факторів,	Мова тексту, емоційна полярність,

			новинного впливу	багатозначність
Генеративний аналіз	LLM-моделі (Gemini, GPT-4o, Claude)	Узагальнення висновків, текстова інтерпретація даних	Формування коротких, зрозумілих рекомендацій	Потреба в чіткому налаштуванні запитів (prompt engineering)
Оцінка ризиків	Байєсівські моделі, Монте-Карло симуляції	Індикатори ризику, VaR	Інтеграція багатьох факторів у прогноз	Чутливість до точності моделей вхідних даних

Аналіз криптовалюти: Bitcoin (BTC)

Приклад аналітичного висновку, який може буде згенеровано нашим програмним продуктом. Висновок: За останні 24 години Bitcoin демонструє зростання обсягу торгів на 15 %, що супроводжується підвищенням ринкової капіталізації на 2,8 %. Поточна ринкова ціна складає 67 800 USD. Актив демонструє позитивну динаміку на фоні зростання активності інституційних інвесторів та новинного фону, пов’язаного із розширенням ETF-продуктів у США.

Рекомендації: Тенденція позитивна, короткостроково можливе продовження зростання. Рекомендується утримання позицій при постійному моніторингу рівня 66 500 USD як ключової технічної підтримки. Варто враховувати потенційні новини щодо регулювання на ринках Азії.

Саме такого типу висновок є можливість моделювати у розробленому нами програмному продукті, завдяки підключенню Gemini через реалізований нами GeminiUseCase - з короткою текстовою аналітикою, обмеженою за кількістю символів та адаптованою для мобільного відображення.

Таким чином, сучасні підходи до застосування AI у фінансовому аналізі дозволяють перейти від класичних методів пасивної обробки даних до створення адаптивних, персоналізованих, високотехнологічних аналітичних систем, здатних інтегрувати численні джерела інформації, оперативно адаптуватися до змін ринкових умов та формувати зрозумілі для користувача висновки в режимі реального часу.

2.4. Архітектурні підходи до розробки систем збору та обробки криптовалютних даних

Розробка інформаційних систем для збору, обробки та аналізу криптовалютних даних вимагає застосування ефективних архітектурних підходів, які дозволяють забезпечити масштабованість, модульність, високу продуктивність та адаптивність до динамічних змін ринкових умов. Особливість предметної області криптовалютних ринків, що характеризується постійною змінністю даних, високою частотою оновлень, наявністю великої кількості різнорідних джерел інформації, а також потребою в гнучкому обробленні як структурованих, так і неструктурованих даних, обумовлює необхідність ретельного обґрунтування вибору архітектурних рішень.

Одним з базових архітектурних підходів для створення подібних систем є багаторівнева архітектура із чітким розмежуванням логічних шарів: шар доступу до джерел даних, шар обробки та трансформації інформації, шар аналітики та прийняття рішень, а також шар відображення результатів для користувача. Поділ системи на такі логічні компоненти дозволяє забезпечити слабкий зв’язок між модулями, спрощує супровід та тестування системи, а також створює передумови для подальшого її масштабування [2, 3].

У сучасних мобільних додатках широкого застосування набув архітектурний патерн MVVM (Model-View-ViewModel), що дозволяє чітко розділити бізнес-логіку, модель даних та інтерфейс користувача. У контексті розробки системи моніторингу криптовалют MVVM забезпечує централізоване управління станами відображення даних, ізоляцію логіки збору та обробки інформації в окремих модулях ViewModel, що дозволяє уникати дублювання коду та полегшує внесення змін у функціонал додатку без порушення стабільності роботи інших його компонентів [5, 20].

На рівні збору даних одним із ключових завдань є організація стабільної інтеграції з біржовими джерелами інформації. У розробленій системі реалізовано комбінацію двох основних механізмів отримання даних: робота з HTTP API за допомогою бібліотеки Retrofit та парсинг HTML-сторінок криптовалютної платформи CoinMarketCap із використанням бібліотеки Jsoup. Такий підхід дозволяє гнучко адаптуватися до змін доступності API-з'єднань, обмежень на кількість запитів, а також забезпечити резервне отримання інформації напряму з відкритих HTML-джерел. Застосування парсингу HTML в даному випадку дозволяє обробляти дані навіть у разі часткової або повної відсутності публічних API у сторонніх сервісів.

На етапі обробки отриманих даних важливо забезпечити їх структуризацію, фільтрацію, конвертацію та приведення до внутрішнього єдиного формату, що дозволяє уніфіковано працювати із різнорідними джерелами інформації. У межах розробленої системи для цього створено спеціальні моделі даних CryptoCurrencyModel та CryptoCoinModel, які відображають основні фінансові характеристики монет: назву активу, ринкову капіталізацію, добовий обсяг торгів, пропозицію в обігу, поточну ціну, а також допоміжні візуальні атрибути (наприклад, посилання на іконки валют).

З метою забезпечення локальної автономності додатку, зменшення навантаження на мережеві з'єднання та можливості офлайн-роботи реалізовано механізм локального кешування інформації з використанням ORM-бібліотеки Room. Локальна база даних дозволяє зберігати актуальні дані криптовалют, забезпечувати їхню синхронізацію із серверними джерелами, а також виконувати локальний пошук, сортування та пагінацію при відображенні списків монет.

Для гнучкого керування залежностями між компонентами системи застосовано бібліотеку Koin, що реалізує принцип інверсії залежностей (Dependency Injection). Це дозволяє централізовано керувати створенням

об’єктів, забезпечуючи модульність системи, спрощуючи її тестування та майбутнє розширення функціоналу.

Окремою складовою архітектурної побудови системи виступає модуль AI-аналітики, який інтегрує генеративну модель Gemini. Генерація стислих текстових висновків реалізована через виділений клас GeminiUseCase, який організовує запити до AI-моделі, передає фінансові дані у відповідному форматі та отримує короткі аналітичні рекомендації, що виводяться в інтерфейс додатку у вигляді блоків висновків та рекомендацій для користувача.

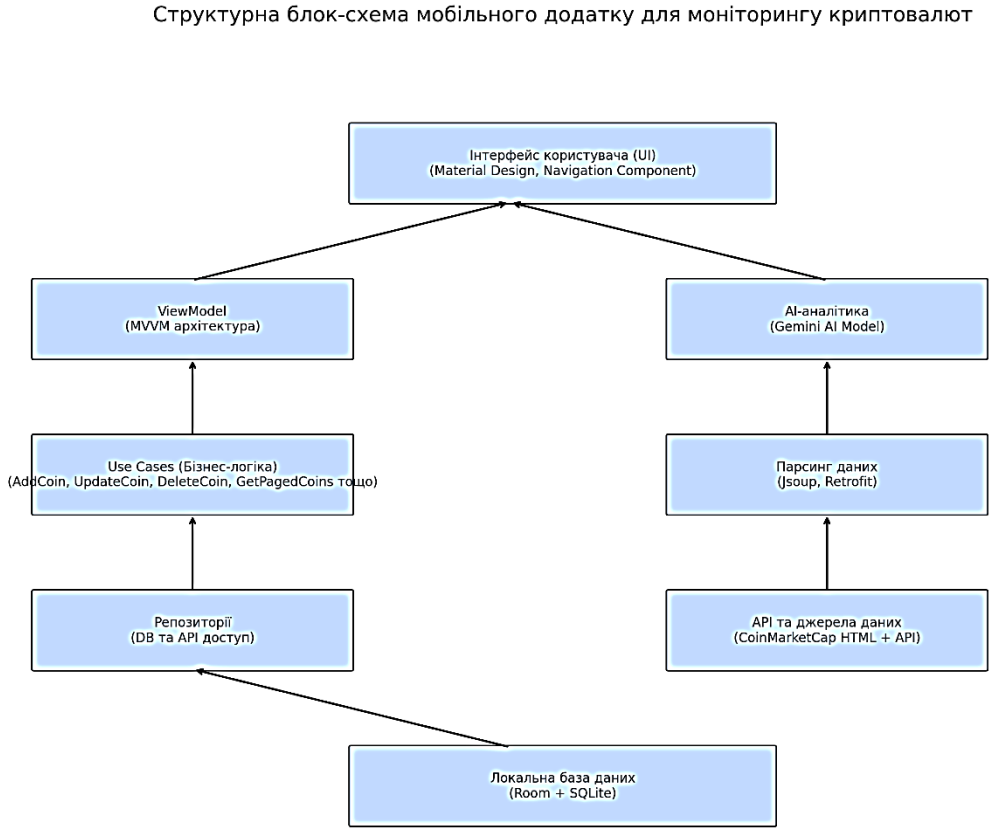

Рисунок 2.5. Структурна блок схема розробленого додатку

Загальна побудова архітектури системи, реалізованої у межах даної роботи, дозволяє забезпечити її стабільність, адаптивність до змін ринкових умов, гнучкість у додаванні нових функціональних модулів та ефективність у роботі з великими обсягами фінансової інформації. Комбінація MVVM, Clean Architecture, парсингу HTML, локального кешування Room, ін’єкції

залежностей Koin та інтеграції генеративного AI формує високотехнологічну основу для створення сучасних мобільних рішень у сфері фінансового моніторингу криптовалютних ринків.

2.5. Тенденції безпечної обробки даних в Android-розробці

Питання забезпечення безпеки даних є одним із ключових аспектів сучасної мобільної розробки, оскільки мобільні додатки, зокрема в домені фінансових технологій і криптовалют, обробляють чутливу інформацію користувачів, включаючи фінансові показники, історію транзакцій, облікові записи та персональні налаштування. У контексті Android-розробки останніми роками спостерігається активна еволюція інструментів, стандартів та рекомендацій, спрямованих на забезпечення високого рівня захисту даних як під час зберігання, так і під час їх обробки та передачі [20].

Однією з фундаментальних тенденцій є впровадження принципу “privacy by design”, що передбачає вбудовування безпекових механізмів у процес розробки ще на етапі проєктування програмної архітектури. Такий підхід вимагає ретельної оцінки кожного з етапів обробки даних, мінімізації обсягів зібраної персональної інформації та реалізації максимальної прозорості в політиках збереження і використання даних користувача [1-3].

На рівні безпечного зберігання локальних даних у середовищі Android важливою тенденцією є широке застосування шифрування баз даних. Для локальної роботи із збереженими даними, зокрема у розробленій у межах цієї роботи системі, використовується SQLite через ORM-бібліотеку Room. З метою додаткового захисту Room дозволяє інтегрувати SQLCipher — розширення для шифрування баз даних, яке застосовує AES-шифрування із 256-бітними ключами. Використання шифрування локальних баз гарантує захист даних навіть у разі фізичного доступу до пристрою сторонніми особами.

Другою важливою тенденцією є забезпечення безпеки каналів передачі даних. Сучасні Android-додатки використовують протоколи TLS (Transport Layer Security) останніх версій для захищеної передачі інформації через мережу. Бібліотека Retrofit, яка використовується для організації HTTP-запитів у розробленій системі, нативно підтримує роботу з безпечними HTTPS-з'єднаннями та дозволяє додатково налаштовувати сертифікати, політики перевірки SSL та перевірку сертифікатів серверів (Certificate Pinning), що значно знижує ризик атак типу «людина посередині» (MITM) [1-3, 9].

Окрему увагу в сучасній Android-розробці приділяють управлінню обліковими даними та токенами авторизації. Замість зберігання паролів або ключів доступу у відкритому вигляді, застосовуються безпечні сховища Android Keystore, які забезпечують апаратно підтримуване шифрування ключів. Використання Keystore гарантує, що приватні ключі не покидають межі захищеного апаратного модуля пристрою, що значно ускладнює несанкціонований доступ навіть при компрометації операційної системи.

Ще однією сучасною практикою є застосування обмежених дозволів доступу (scoped storage), запроваджених у нових версіях Android. Це дозволяє обмежити доступ додатків до файлової системи пристрою та персональних даних користувача, що зменшує ризик витоку інформації через сторонні застосунки. У сучасних додатках розробники мають працювати лише в межах виділеного контейнера власного застосунку, а доступ до зовнішніх ресурсів можливий лише за прямої згоди користувача.

Значну роль у забезпеченні безпеки займає також використання сучасних засобів ін'єкції залежностей, таких як Koin або Hilt, що дозволяють централізовано керувати створенням і передачею об'єктів, мінімізуючи ризики витоків даних при неправильному управлінні життєвим циклом компонентів. У проєкті, що реалізовано у даній роботі, застосування Koin сприяє ізоляції доступу до джерел даних, модулів бізнес-логіки та зовнішніх сервісів, знижуючи ймовірність непередбачених маніпуляцій із даними.

Окремий блок сучасних практик пов’язаний із моніторингом та аудитом безпеки мобільних застосунків. Google Play вимагає обов’язкового проходження перевірок безпеки нових застосунків, верифікації використання персональних даних, вказання цілей їх обробки та дотримання політики обмеження доступу до конфіденційної інформації. Крім того, існує розширений перелік тестів, що рекомендується проводити на етапах тестування, зокрема аналіз уразливостей OWASP Mobile Top 10.

Загальною тенденцією розвитку безпеки в мобільній Android-розробці стає комплексний підхід до забезпечення конфіденційності даних, який охоплює як захист даних у стані зберігання, так і їх захищену передачу, контроль над доступом, обмеження прав доступу до апаратних і системних ресурсів пристрою, а також впровадження прозорої політики обробки персональної інформації [1-3, 9].

Таким чином, побудова безпечних мобільних додатків у сфері моніторингу фінансових даних вимагає комплексного впровадження низки архітектурних, криптографічних, організаційних та процесних заходів. Реалізований у межах цієї роботи програмний продукт враховує зазначені тенденції, використовуючи захищене зберігання локальних даних, шифрування каналів передачі, обмеження прав доступу та сучасні методи захисту облікових даних, що у сукупності формує високий рівень інформаційної безпеки користувачів додатку.

РОЗДІЛ 3. ТЕОРЕТИЧНА ЧАСТИНА

3.1. Обґрунтування вибору стеку технологій для реалізації додатку

У процесі розробки мобільного додатку для моніторингу криптовалютних ринків було використано сучасний технологічний стек, який забезпечує стабільність, масштабованість, високу продуктивність і модульність системи.

Для організації локального зберігання даних застосовувалася бібліотека Room (androidx.room.runtime, room.ktx, room.paging), яка є сучасною ORM (Object Relational Mapping) обгорткою над вбудованою у Android системою керування базами даних SQLite. Room дозволяє забезпечити безпечний доступ до бази на основі анотаційного опису сутностей, автоматично формуючи SQL-запити на етапі компіляції. Це значно зменшує ймовірність виникнення помилок при формуванні запитів, спрощує підтримку коду та забезпечує типобезпечність. Крім того, використання Room у поєднанні з компонентом paging дозволяє реалізувати ефективне сторінкове завантаження великих списків монет, що є критично важливим для оптимізації роботи з великими обсягами криптовалютних активів.

Отримання ринкових даних з криптовалютних бірж організовано із застосуванням клієнта Retrofit (retrofit, converter.scalars). Дана бібліотека виступає універсальним HTTP-клієнтом для виконання REST-запитів, забезпечуючи зручну декомпозицію мережевих запитів у вигляді строго типізованих інтерфейсів. Особливістю застосування Retrofit у даному проєкті є його конфігурація для роботи з текстовими scalar-відповідями, що дозволяє безпосередньо отримувати HTML-документи зі сторінок криптовалютних бірж [13, 14].

Подальша обробка HTML-документів реалізована із використанням бібліотеки Jsoup (jsoup), яка є одним із найбільш популярних інструментів для

парсингу веб-контенту. Завдяки підтримці DOM-моделі, CSS-селекторів та зручних API для вилучення необхідних фрагментів HTML, Jsoup дозволяє із високою точністю отримувати окремі елементи сторінок, що містять необхідні фінансові показники монет.

Для забезпечення завантаження і відображення зображень криптовалютних активів застосовано бібліотеку Picasso (picasso), яка дозволяє асинхронно завантажувати графічні ресурси з мережі, здійснювати їх кешування у внутрішній пам’яті пристрою, а також оптимізувати завантаження з урахуванням ресурсів пристрою. Застосування Picasso гарантує високу продуктивність та плавне відображення графічних елементів в інтерфейсі.

Важливою інноваційною складовою додатку є інтеграція модуля Generative AI (generativeai), побудованого на основі генеративної моделі Gemini. Ця система штучного інтелекту дозволяє здійснювати автоматичний аналіз ринкових даних та формувати короткі текстові висновки і рекомендації на основі фінансових показників криптовалют. Інтеграція Generative AI забезпечує користувача стислою аналітикою без потреби самостійно аналізувати великі масиви числових даних.

З метою організації слабко зв’язаної архітектури застосовано бібліотеку Koin (koin.android), яка реалізує механізми інверсії керування (Inversion of Control) через впровадження залежностей (Dependency Injection). Завдяки Koin забезпечується централізоване управління залежностями між компонентами системи, що суттєво підвищує модульність, спрощує тестування та зменшує складність розширення системи у майбутньому.

Організація навігації між екранами додатку побудована на основі компонента Navigation Component (androidx.navigation.fragment.ktx, ui.ktx), який дозволяє описувати усю навігаційну логіку у вигляді єдиної діаграми переходів. Це гарантує типобезпечність передавання аргументів між екранами, спрощує підтримку складних графів переходів та забезпечує наочність усієї навігаційної структури проєкту.

Візуальна частина інтерфейсу розроблена із застосуванням стилістики Material Design (material), яка є загальноприйнятим стандартом побудови сучасних інтерфейсів у мобільних додатках. Завдяки цьому забезпечується інтуїтивно зрозумілий, адаптивний та стабільний користувацький інтерфейс [4].

Для спрощення доступу до XML-компонентів інтерфейсу застосовано View Binding (viewBinding). Цей механізм генерує типобезпечні класи доступу до елементів макетів, що дозволяє уникати використання застарілих методів findViewById() і забезпечує статичну перевірку доступу до візуальних компонентів уже на етапі компіляції.

Забезпечення якості програмного забезпечення здійснюється через автоматизоване тестування із застосуванням інструментів Espresso (androidx.test.espresso.core) та JUnit (junit). Espresso використовується для UI-тестування інтерфейсу користувача, тоді як JUnit забезпечує юніт-тестування окремих функціональних блоків логіки додатку.

Використання Kotlin Kapt (kotlin-kapt) необхідне для компіляційної підтримки анотаційних процесорів, зокрема у взаємодії з Room, де на основі анотацій генеруються класи доступу до бази даних.

Для організації безпечної передачі параметрів між фрагментами застосовано Safe Args (androidx.navigation.safeargs.kotlin). Завдяки використанню цього компонента виключається ймовірність помилок при передаванні об’єктів через Bundle, що значно підвищує безпеку навігаційних сценаріїв.

Таким чином, застосований стек технологій забезпечив побудову сучасного, масштабованого, ефективного та безпечного мобільного додатку з використанням кращих практик програмної інженерії.

Усі зазначені компоненти інтегруються у загальну архітектурну модель застосунку, що базується на принципах чистої архітектури (Clean Architecture). Така побудова забезпечує максимальну ізоляцію бізнес-логіки від зовнішніх залежностей, дозволяє легко змінювати внутрішню реалізацію модулів без

порушення роботи всієї системи, а також створює передумови для подальшого розширення функціоналу, зокрема інтеграції нових джерел даних або додаткових AI-сервісів.

Таким чином, вибір кожної технології у розробці даного мобільного додатку є обґрунтованим з позицій функціональної доцільності, архітектурної стабільності, продуктивності, безпеки та гнучкості підтримки системи в умовах швидкозмінного криптовалютного ринку.

3.2. Особливості інтеграції з API криптовалютних бірж

Однією з ключових функціональних складових розробленого мобільного додатку для моніторингу криптовалютних ринків є організація отримання актуальних фінансових даних про криптовалюти. Зважаючи на динамічність ринку, високий темп змін котирувань, появу нових активів та потребу в широкій базі аналітичних показників, особливу увагу було приділено вибору ефективної стратегії інтеграції додатку з джерелами ринкової інформації.

В загальному випадку інтеграція з криптовалютними біржами або інформаційними агрегаторами може здійснюватися за двома основними напрямами: безпосередня робота з публічними API сторонніх сервісів або опосередкована взаємодія через парсинг HTML-контенту біржових сайтів. У межах даної розробки було обґрунтовано доцільність комбінованого підходу з акцентом на парсинг HTML сторінок сервісу CoinMarketCap та роботу з HTTP-запитами через власний API-інтерфейс.

Основним джерелом фінансової інформації виступає глобальна криптовалютна платформа CoinMarketCap, яка акумулює дані з десятків криптовалютних бірж, забезпечуючи їх стандартизацію та уніфіковану подачу. Сервіс надає як власний REST API, так і веб-інтерфейс для перегляду ринкової

інформації. Водночас використання публічного API CoinMarketCap супроводжується обмеженнями за кількістю запитів, необхідністю отримання ключів доступу, а також обмеженням доступу до частини розширених фінансових показників у межах безкоштовних тарифних планів. У зв'язку із цим, з метою забезпечення повної гнучкості в отриманні необхідних ринкових параметрів, було прийнято рішення про застосування технології парсингу HTML сторінок для вилучення релевантних фінансових даних напряму з веб-інтерфейсу CoinMarketCap.

Реалізація парсингу здійснюється з використанням бібліотеки Jsoup, яка забезпечує функціональну роботу з HTML-документами у форматі DOM-моделі. Парсер дозволяє обирати необхідні елементи сторінок за CSS-селекторами, вилучати текстові значення фінансових індикаторів (ринкова капіталізація, добовий обсяг торгів, поточна ціна, кількість монет в обігу, пропозиція, відношення Vol/MktCap тощо), а також завантажувати супровідні графічні елементи (іконки монет). Даний підхід дозволяє обробляти сторінки як з окремою інформацією по кожній криптовалюті, так і агреговані списки монет із основної сторінки каталогу.

Безпосереднє формування HTTP-запитів до CoinMarketCap реалізовано на основі бібліотеки Retrofit, яка є універсальним HTTP-клієнтом із підтримкою REST-архітектури. Завдяки використанню ScalarsConverterFactory забезпечено роботу з HTML-відповідями у вигляді звичайних рядків, які передаються до парсера Jsoup для подальшого вилучення релевантних фрагментів інформації. Така архітектура дозволяє розмежувати модулі запиту даних та модулі їх обробки, зберігаючи гнучкість при зміні структури сторінок або у разі модифікації політики доступу сервісу.

Окремою важливою особливістю реалізованої інтеграції є організація роботи з пагінацією. Оскільки кількість криптовалютних активів постійно зростає, відображення всієї доступної інформації за один запит є неефективним та недопустимим з позиції продуктивності. У розробленій системі передбачено

підтримку пагінації шляхом послідовного формування запитів з параметрами сторінок (?page=N), що дозволяє контролювати обсяги даних, що завантажуються, та оптимізувати роботу із зовнішніми сервісами.

В межах роботи з API та парсингом даних реалізовано модульну структуру репозиторіїв (CryptoHtmlRepository, CryptoHtmlHtmlRepositoryImpl), що дозволяє централізовано управляти процесами запиту та отримання даних. Дана модульність забезпечує можливість легкої адаптації системи до інших джерел інформації у разі появи нових біржових платформ або зміни форматів подачі інформації на стороні CoinMarketCap.

Особливу увагу було приділено обробці помилок, які можуть виникати при роботі з зовнішніми API, зокрема при порушенні стабільності мережевого з’єднання, перевищенні ліміту запитів, змінах у структурі HTML-документів або відмовах сервісу. Для цього реалізовано відповідні механізми перехоплення виключень, перевірок статусів HTTP-відповідей та обробки некоректних даних, що дозволяє зберігати стабільність роботи мобільного застосунку навіть у випадку часткових збоїв у зовнішніх джерелах даних.

Таким чином, застосування комбінованої стратегії інтеграції із зовнішніми джерелами криптовалютних даних на базі HTTP-запитів Retrofit та парсингу HTML-контенту через Jsoup забезпечує високу гнучкість, стабільність та актуальність фінансової інформації, що надається користувачам мобільного додатку. Обрана архітектура дозволяє адаптувати систему до змін у зовнішньому середовищі та забезпечує основу для подальшого масштабування функціоналу

Для ілюстрації реалізації описаного підходу доцільно розглянути фрагмент коду з проєкту, у якому здійснюється отримання HTML-сторінки конкретної криптовалюти з веб-інтерфейсу CoinMarketCap:

```
@GET("currencies/{coin}")
suspend fun getCoinHtmlByAPI(
    @Path("coin") coinName: String
```

): Response<String>

Цей метод є частиною інтерфейсу CryptoAPI, що описує структуру HTTP-запитів для отримання HTML-даних з сервера CoinMarketCap. Метод використовує HTTP-метод GET, параметризований назвою монети (coinName), яка вставляється динамічно в URL-запит (наприклад: https://coinmarketcap.com/currencies/bitcoin). Повернення значення типу Response<String> свідчить про те, що серверна відповідь очікується у вигляді HTML-документу у форматі тексту.

Дані, отримані з API, надалі обробляються в окремому UseCase-класі GetCoinHTMLUseCase, який відповідає за вилучення структурованої інформації з HTML-контенту за допомогою парсера Jsoup. Ось приклад фрагмента, що виконує вилучення ключових метрик:

val marketCap = document.selectFirst("dt:contains(Market cap) + dd span")?.text()

val volume24h = document.selectFirst("dt:contains(Volume (24h)) + dd span")?.text()

val marketPrice = document.selectFirst("span[data-test='text-cdp-price-display']")?.text()

val coinName = document.selectFirst("span[data-role='coin-name']")?.attr("title")

Цей код виконує селективний аналіз DOM-структури HTML-сторінки. Зокрема, метод selectFirst дозволяє знайти перший елемент, що відповідає заданому CSS-селектору. Наприклад, конструкція dt:contains(Market cap) + dd span вилучає значення капіталізації ринку монети, що знаходиться у відповідному теге span, розміщеному після тега dt з вмістом "Market cap".

Отримані в результаті парсингу значення далі формують модель CryptoCoinModel, яка використовується на рівні бізнес-логіки додатку, ViewModel та відображення інтерфейсу користувача. Така реалізація дозволяє уникнути обмежень стандартного API, працювати з актуальними HTML-

сторінками без необхідності оплати API-доступу, а також оперативно оновлювати дані монет без затримки.

Загалом, приклад демонструє ефективну інтеграцію між мережевим модулем, парсером HTML-контенту та шаром бізнес-логіки. Застосування Retrofit і Jsoup у зв’язці дозволяє реалізувати високорівневу абстракцію доступу до зовнішніх даних, при цьому підтримуючи модульність та масштабованість архітектури мобільного додатку.

3.3. Архітектурний підхід Clean Architecture у поєднанні з MVVM та побудова блок-схеми додатку

У процесі розробки сучасних мобільних застосунків, особливо тих, що працюють з великими обсягами даних, інтегрують численні зовнішні сервіси, виконують аналітичні обчислення та потребують високої гнучкості у масштабуванні функціоналу, особливу актуальність набуває застосування багаторівневих архітектурних підходів. У межах реалізації даного мобільного додатку для моніторингу криптовалютних ринків було обґрунтовано доцільність поєднання архітектурних принципів Clean Architecture із патерном MVVM (Model-View-ViewModel) [3, 4, 19].

Архітектурна модель Clean Architecture забезпечує суворе розділення відповідальностей між основними логічними рівнями додатку: презентаційним, доменним та інфраструктурним. Такий підхід дозволяє досягти високої ізоляції бізнес-логіки від зовнішніх джерел даних, інтерфейсів користувача та конкретних технологічних реалізацій.

На верхньому рівні архітектури розташовується Presentation Layer — шар представлення, який взаємодіє з користувачем та забезпечує відображення даних у інтерфейсі додатку. У цьому шарі основну роль виконують ViewModel-

компоненти, що реалізують патерн MVVM. ViewModel управляє станами інтерфейсу, отримує дані з доменної логіки та готує їх до відображення, координує оновлення списків монет, відображення деталей криптовалют, обробку подій додавання чи видалення активів. Наприклад, PublicCoinsViewModel, LoadCryptoCoinsViewModel, InsertCoinsViewModel тощо відповідають за різні функціональні ділянки додатку. Такий підхід дозволяє забезпечити реактивність інтерфейсу шляхом застосування StateFlow та LiveData для відстеження змін станів.

Центральною частиною є Domain Layer — шар бізнес-логіки додатку. У ньому реалізовано основні сценарії взаємодії системи через класи UseCase. Наприклад, AddCoinUseCase, DeleteCoinUseCase, GetPagedCoinsUseCase, GetCoinHTMLUseCase, GeminiUseCase тощо інкапсулюють ключові операції доменної області, ізолюючи їх від конкретних реалізацій роботи із зовнішніми джерелами даних. Саме цей шар визначає основні правила бізнес-процесів додатку, контролює валідацію, оновлення, обробку й формування об'єктів моделей.

На рівні Data Layer знаходяться репозиторії, які забезпечують взаємодію з джерелами даних. Репозиторії (CryptoDBRepositoryImpl, CryptoHtmlHtmlRepositoryImpl) координують доступ до локальної бази даних через Room та до зовнішніх джерел інформації через Retrofit і Jsoup. Завдяки чіткій абстракції через інтерфейси CryptoDBRepository та CryptoHtmlRepository забезпечується повна заміннність реалізацій без впливу на вищі шари системи.

Особливим блоком у загальній архітектурі виступає AI Layer, який представлено інтеграцією з генеративною моделлю Gemini. Даний модуль функціонує незалежно, забезпечуючи формування аналітичних висновків на основі переданих фінансових параметрів криптовалютних активів. За допомогою класу GeminiUseCase забезпечується формування запитів до AI та обробка результатів для відображення у ViewModel.

Для інверсії залежностей у всіх шарах архітектури використовується

бібліотека Koin, яка дозволяє централізовано реєструвати всі залежності між модулями системи. Це забезпечує слабкий зв’язок між компонентами, полегшує їх тестування, підтримку та розширення системи.

Поєднання Clean Architecture та MVVM дозволяє досягти максимальної модульності та гнучкості системи, спрощує процес тестування окремих компонентів, сприяє повторному використанню коду та забезпечує легку інтеграцію додаткового функціоналу в майбутньому. Архітектура, що реалізована у даній роботі, є типовим прикладом побудови промислових Android-застосунків у сфері фінансових технологій та моніторингу даних.

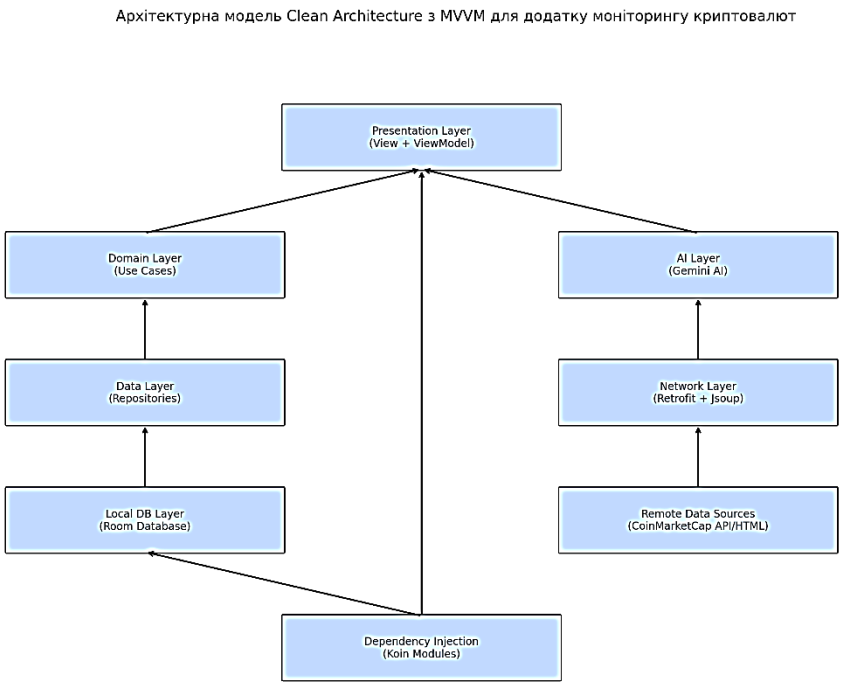

Рисунок 3.1 Архітектурна модель Clean Architecture з MVVM для розробленого додатку

Запропонована архітектурна модель мобільного додатку для моніторингу криптовалютних ринків реалізована на основі поєднання концепцій Clean Architecture та MVVM, що забезпечує чіткий поділ обов’язків між основними компонентами, гнучкість масштабування системи та високу якість супроводу програмного коду.

Presentation Layer (View + ViewModel)

На верхньому рівні розташований шар представлення. Його складовими є фрагменти інтерфейсу (MainFragment, AddCryptoFragment, DetailCryptoFragment), які забезпечують взаємодію з користувачем, а також ViewModel-класи (LoadCryptoCoinsViewModel, PublicCoinsViewModel, InsertCoinsViewModel, DeleteCoinsViewModel, GeminiViewModel), які координують отримання та обробку даних.

ViewModel-компоненти отримують дані від бізнес-логіки, зберігають їх у реактивних потоках (StateFlow) і транслюють у відповідний інтерфейс для відображення. Наприклад, у LoadCryptoCoinsViewModel реалізовано логіку завантаження списку монет, оновлення їхніх цінових показників, оновлення пагінації, координацію роботи з базою даних і зовнішніми джерелами. ViewModel-підхід дозволяє забезпечити стійкість додатку до ротацій екранів та змін стану життєвого циклу.

Domain Layer (Use Cases)

Доменний шар представлений класами UseCase, які інкапсулюють бізнес-логіку системи:

- GetCryptoCurrenciesHTMLUseCase — отримання списку криптовалют з CoinMarketCap.
- GetCoinHTMLUseCase — завантаження HTML-даних про окрему монету та парсинг фінансових показників.
- AddCoinUseCase, DeleteCoinUseCase, UpdateCoinsUseCase — збереження, видалення та оновлення монет у базі даних.
- GeminiUseCase — взаємодія зі штучним інтелектом для аналізу фінансових показників монет.
- CheckCoinExistUseCase, GetPagedCoinsUseCase — допоміжна логіка для контролю стану локальної бази.

Доменний рівень не залежить від інфраструктурних деталей і ізольований від конкретних реалізацій роботи з мережею або базою даних.

Data Layer (Repositories)

Шар даних містить конкретні реалізації репозиторіїв:

- CryptoDBRepositoryImpl — робота з локальною базою Room через DAO-інтерфейс.
- CryptoHtmlHtmlRepositoryImpl — обробка HTTP-запитів до CoinMarketCap через Retrofit та передачі отриманого HTML на подальший парсинг.

Завдяки абстрагуванню через інтерфейси CryptoDBRepository та CryptoHtmlRepository доменний шар не залежить від конкретних джерел інформації.

Network Layer (Retrofit + Jsoup)

У межах мережевого шару відбувається робота із зовнішнім ресурсом CoinMarketCap.

Для цього застосовується:

- Retrofit — бібліотека для організації HTTP-запитів (наприклад, метод getCoinHtmlByAPI() у CryptoAPI).
- Jsoup — для парсингу HTML-сторінок (наприклад, метод parseHtmlCoin() у GetCoinHTMLUseCase), де витягуються такі показники як: Market Cap, Volume 24h, Circulating Supply, Price тощо.

Використання парсингу HTML дозволяє обійти обмеження публічного API CoinMarketCap та отримати максимально широкий набір ринкових показників.

Local DB Layer (Room Database)

Уся інформація, що була зібрана та оброблена, кешується у локальній базі даних Room:

- Таблиця CoinsTable зберігає ключову інформацію по монетах.
- DAO-інтерфейси організують операції читання, запису, оновлення та видалення монет.
- Room також використовується для реалізації пагінації даних при відображенні великих обсягів інформації.

Локальна база дозволяє забезпечити роботу додатку в офлайн-режимі, зменшити навантаження на мережу та збільшити продуктивність UI.

Remote Data Sources (CoinMarketCap API/HTML)

Фактичне джерело зовнішніх фінансових даних — це ресурс CoinMarketCap, з якого здійснюється отримання HTML-контенту. З огляду на гнучкість реалізації, додаток може бути легко адаптований до інших біржових платформ.

AI Layer (Gemini AI)

Окремий рівень займає інтеграція із генеративною мовною моделлю Gemini:

- Клас GeminiUseCase передає до моделі фінансові показники монет, отримані під час парсингу.
- AI-модель генерує короткі висновки та рекомендації у текстовому вигляді, які транслюються у ViewModel та відображаються у UI.
- Такий підхід дозволяє отримати інтерактивний аналітичний супровід фінансових рішень користувача.

Dependency Injection (Koin Modules)

Усі взаємозалежності між компонентами реалізовані за допомогою Koin:

- Всі репозиторії, UseCase-класи, ViewModel-и та API-сервіси централізовано реєструються у відповідних DI-модулях.
- Це дозволяє легко тестувати окремі компоненти системи, модифікувати залежності та забезпечує слабкий зв'язок між модулями.

Таким чином, архітектура, реалізована в межах розробленого додатку, є класичним прикладом сучасної багаторівневої Android-архітектури, що дозволяє ефективно обробляти великі обсяги криптовалютних даних у реальному часі, зберігаючи при цьому стабільність, масштабованість та підтримуваність програмного забезпечення.

3.4. Побудова моделі даних, взаємодія з Room DB та Paging

Однією з важливих функціональних частин розробленого мобільного додатку є побудова гнучкої та продуктивної системи зберігання, оновлення та відображення великих обсягів криптовалютних даних. З цією метою у додатку реалізовано повноцінну роботу з локальною базою даних на основі бібліотеки Room із підтримкою пагінації та оптимізованої синхронізації даних.

Побудова моделі даних

На першому етапі було розроблено єдину систему внутрішніх моделей даних, які відображають ключові параметри кожної криптовалюти. Для цього створено два основних класи моделей: CryptoCurrencyModel та CryptoCoinModel.

Клас CryptoCurrencyModel використовується переважно для зберігання стислої інформації про монети при завантаженні списків монет з CoinMarketCap. Він містить атрибути:

- назва монети (coinName);
- поточна ринкова ціна (coinPrice);
- посилання на іконку криптовалюти (coinImg).

Клас CryptoCoinModel є більш розширеною моделлю, яка використовується для збереження детальної інформації про кожну монету. До його складу входять наступні параметри:

- ринкова капіталізація (marketCap);
- обсяг торгів за 24 години (volume24h);
- повна пропозиція (totalSupply);
- обігова пропозиція (circulatingSupply);
- відношення обсягів торгів до капіталізації (volMktCap);
- повністю розводнена вартість (fdv);
- поточна ринкова ціна (marketPrice);

- іконка монети (coinImg);
- назва монети (coinName).

Для ефективної передачі цих об'єктів між компонентами додатку в межах Android застосовано механізм @Parcelize, що забезпечує серіалізацію об'єктів моделей для передачі через фрагменти за допомогою Safe Args.

Взаємодія з Room DB

З метою локального зберігання даних у мобільному додатку застосовується база даних SQLite через обгортку ORM-бібліотеки Room. Це дозволяє зберігати списки монет, кешувати отримані дані та забезпечувати офлайн-доступ до останніх оновлень.

Структура таблиці бази даних представлена класом CoinsTable, який містить основні атрибути монети, зокрема:

- coinName (первинний ключ);
- coinPrice (ринкова ціна у форматі Float).

Взаємодія з базою даних організована через DAO-інтерфейс, що містить основні методи для:

- вставки нових монет (insert()),
- оновлення цін (updateCoins()),
- перевірки наявності монети (getCoin()),
- видалення монети (delete()),
- вибірки монет з пагінацією (getCoinsWithPagination()).

Реалізація репозиторію CryptoDBRepositoryImpl виконує всі необхідні операції з DAO на рівні бізнес-логіки, забезпечуючи єдину точку доступу до локальних даних.

Реалізація пагінації (Paging)

Зважаючи на потенційно велику кількість криптовалютних активів у базі даних, відображення усієї інформації одразу є недоцільним як з точки зору продуктивності, так і зручності користувача. Тому було реалізовано механізм пагінації, який дозволяє динамічно завантажувати дані частинами при

скролінгу списку монет.

У класі GetPagedCoinsUseCase реалізовано метод execute(), який на основі поточного номера сторінки формує запити до бази з відповідним offset. Наприклад, при кожному запиті відбувається вибірка наступних 5 монет:

```
fun execute(page: Int): Flow<List<String>> {
    return flow {
        emit(repositoryDBRepository.getAllCoinPagination(page * 5 - 5).map { it.coinName })
    }
}
```

На рівні інтерфейсу користувача для відображення списків застосовано компонент RecyclerView, інтегрований із адаптером CryptoCoinsPrivateAdapter, який отримує дані від ViewModel через StateFlow. При досягненні кінця поточного списку відбувається виклик наступної сторінки даних:

```
if (layoutManager.findLastCompletelyVisibleItemPosition() == cryptoAdapter.itemCount - 1) {
    loadCryptoCoinsViewModel.loadCoinsData()
}
```

Таким чином, реалізована система пагінації дозволяє плавно завантажувати великі обсяги даних без перевантаження оперативної пам'яті пристрою.

Синхронізація даних

Для забезпечення актуальності інформації у локальній базі періодично ініціюється оновлення даних через завантаження актуальних фінансових показників з CoinMarketCap із подальшим оновленням локальної бази методом updateCoinsDB():

```
suspend fun updateCoinsDB(coins: List<CryptoCoinModel>) {
    val transformCoinsTypeList = coins.map { coin -> coin.toCoinsTable() }
    repositoryDBRepository.updatePriceCoin(transformCoinsTypeList)
```

}

Це дозволяє автоматично синхронізувати базу відповідно до змін ринкових котирувань та відображати оновлені ціни у користувацькому інтерфейсі.

Взаємодія з бізнес-логікою

Уся логіка роботи з локальними даними повністю ізольована у бізнес-шарі UseCase-класів, що гарантує слабку залежність між архітектурними шарами. Завдяки цьому зміни у структурі бази або репозиторіїв не потребують втручання у доменний чи презентаційний рівень, що забезпечує високу стабільність системи.

Таким чином, побудова моделі даних у додатку ґрунтується на чіткій структуризації фінансових параметрів криптовалют, а застосування Room у поєднанні з механізмом пагінації дозволяє ефективно обробляти великі обсяги динамічних даних у мобільному середовищі, зберігаючи високу продуктивність та стабільність роботи додатку.

3.5. Методологія впровадження AI-аналітики криптовалют

Сучасний криптовалютний ринок характеризується високою динамікою, складною залежністю між численними економічними, технічними та соціальними факторами, що формують ринкову поведінку криптовалютних активів. У таких умовах традиційні методи статистичного аналізу часто виявляються недостатніми для швидкої та ефективної обробки великого масиву фінансових даних. Саме тому інтеграція систем штучного інтелекту (AI) в аналітичні блоки фінансових застосунків набуває особливої актуальності.

В межах реалізації розробленого додатку впроваджено окремий модуль AI-аналітики, що базується на застосуванні генеративної моделі Gemini —

представника сучасних великих мовних моделей (LLM, Large Language Models), здатних обробляти складні запити та формувати стислий узагальнений текстовий аналіз на основі переданих фінансових показників конкретної криптовалюти.

Архітектура AI-модуля у системі

Інтеграція AI-аналітики організована згідно із загальними принципами Clean Architecture та представлена окремим шаром бізнес-логіки через UseCase-клас GeminiUseCase. Основна функціональність модуля полягає у:

- прийомі детальних фінансових даних криптовалюти (ринкова капіталізація, обсяг торгів, пропозиція, ціна тощо);
- формуванні текстового запиту (prompt) до AI-моделі у спеціально структурованому форматі;
- генерації короткого аналітичного висновку на основі отриманих фінансових параметрів.

Дані для аналізу формуються з моделі CryptoCoinModel, яка містить усі основні показники, попередньо отримані шляхом парсингу HTML-сторінок CoinMarketCap.

Формування промптів для генеративної моделі

Особливістю використання генеративних AI-моделей є необхідність точного формування текстових інструкцій (prompt engineering), що описують очікуваний формат відповіді. У межах розробленого додатку сформульовано чітку структуру запиту до Gemini, який містить:

- запит на стислий аналіз стану криптовалюти;
- рекомендацію щодо подальших дій;
- обмеження обсягу тексту (до 250 символів);
- заборону використання зайвих символів і розміток для чистоти тексту;
- приклад структури відповіді: блок "Висновок" і блок "Рекомендації".

Приклад промпту, що передається до AI-моделі:

private const val ANALISE_CRYPTO_COIN =

"Проаналізуй показники криптовалюти та надай короткий висновок і рекомендацію у вигляді текстового блоку. Обмеж відповідь 250 символами.

Висновок: ...

Рекомендації: ...

Криптовалюта по якій робити аналіз:"

Такий підхід дозволяє забезпечити стандартизовану форму аналітичної відповіді, зручної для подальшого відображення в інтерфейсі користувача.

Взаємодія з AI через GeminiUseCase

Безпосередня робота з генеративною моделлю реалізована у класі GeminiUseCase. Взаємодія організована через асинхронний потік Flow, що дозволяє отримати відповідь моделі без блокування основного потоку виконання додатку:

```
fun requestGemini(coin: CryptoCoinModel): Flow<GenerateContentResponse> {
    return flow {
        emit(gemini.generateContent(ANALISE_CRYPTO_COIN + coin))
    }
}
```

Даний механізм дозволяє підключати AI-аналітику без порушення загальної реактивної моделі роботи додатку. Отримана відповідь передається до ViewModel GeminiViewModel і далі — у StateFlow для оновлення інтерфейсу.

Обробка та відображення аналітичних висновків

На рівні користувацького інтерфейсу результати AI-аналізу відображаються у DetailCryptoFragment. Після отримання відповіді від генеративної моделі, висновок та рекомендація виводяться у відповідному блоці аналізу для кожної монети. Наприклад:

Висновок: Ринкова капіталізація зростає на 2.5%, обсяги торгів залишаються стабільними. Поточна ціна 43000 USD.

Рекомендації: Утримувати актив з моніторингом підтримки 42000 USD.

Таким чином, користувач отримує компактний, узагальнений аналіз стану криптовалюти, що допомагає оперативно оцінити ринкову ситуацію без необхідності самостійного аналізу великого масиву числових показників.

Безпека та контроль AI-взаємодії

Зважаючи на специфіку генеративних моделей, в системі передбачено контроль допустимих відповідей, обмеження довжини тексту, а також застосування вбудованих політик безпеки Gemini API, які блокують некоректний або шкідливий контент згідно з політиками Google AI.

Переваги інтегрованої AI-аналітики

Інтеграція генеративного штучного інтелекту в рамках системи забезпечує:

- оперативний синтез великого обсягу фінансових показників у короткий аналітичний висновок;
- можливість персоналізації аналітики для кожної криптовалюти;
- скорочення часу аналізу для кінцевого користувача;
- підвищення зручності прийняття рішень при моніторингу активів.

Таким чином, впровадження AI-модуля на основі генеративної моделі Gemini дозволяє підвищити інтелектуальну складність додатку, забезпечуючи його інтеграцію з сучасними методами обробки фінансових даних та аналітики в реальному часі.

РОЗДІЛ 4. ПРАКТИЧНА ЧАСТИНА

4.1. Інтеграція з криптовалютними біржами через парсинг HTML

Вимоги до функціональності мобільного застосунку визначаються специфікою криптовалютного ринку, потребами цільових користувачів у постійному доступі до оновленої інформації, а також завданнями аналітичної обробки даних з подальшим формуванням висновків. У відповідності до цього, застосунок має реалізовувати низку функцій, які забезпечують повноцінне відображення, збереження, обробку та інтелектуальний аналіз даних у реальному часі.

До функціональних вимог, які реалізовано у проєкті, належать наступні:

1. Отримання даних про криптовалюти з відкритих джерел.

- Реалізовано інтеграцію з CoinMarketCap через HTTP-запити (Retrofit) з використанням ендпоінтів getCoinHtmlByAPI та getCryptoCurrenciesHTMLByAPI з класу CryptoAPI.
- Для обробки HTML-вмісту використовується бібліотека Jsoup, яка забезпечує парсинг та вилучення ключових показників: ринкова капіталізація, обсяг торгів за 24 години, ціна, загальна та обігова пропозиція.

2. Обробка та трансформація HTML-контенту в структуровані моделі даних.

- Побудовано власні моделі CryptoCurrencyModel та CryptoCoinModel, які містять усі необхідні атрибути криптовалютного активу.
- Реалізовано use cases GetCoinHTMLUseCase та GetCryptoCurrenciesHTMLUseCase, які відповідають за логіку запиту, парсингу та валідації отриманих даних.

3. Збереження даних у локальній базі з використанням ORM.

- Впроваджено бібліотеку Room, яка забезпечує зберігання даних у таблиці CoinsTable через інтерфейс DAO.

– Взаємодію з базою реалізовано в репозиторії CryptoDBRepositoryImpl, зокрема методи: addNewCoin(), checkCoinExist(), updatePriceCoin().

4. Робота зі списком обраних монет.

– Користувач має змогу додавати криптовалюти до персонального списку за допомогою UI у AddCryptoFragment.

– Перед додаванням здійснюється перевірка на наявність монети в базі (CheckCoinExistUseCase).

– Для перегляду списку реалізовано MainFragment з відображенням елементів через CryptoCoinsPrivateAdapter.

5. Інтеграція штучного інтелекту для аналітики.

– Реалізовано use case GeminiUseCase, який використовує модель GenerativeModel (Gemini API) для генерації висновків на основі показників обраної монети.

– Формат відповіді AI визначено у вигляді стислого блоку з розділом на «Висновок» і «Рекомендації», що демонструється у DetailCryptoFragment.

6. Побудова багаторівневої архітектури з використанням шаблону MVVM.

– Кожна група операцій винесена у відповідну ViewModel (LoadCryptoCoinsViewModel, InsertCoinsViewModel, GeminiViewModel тощо).

– Між шарами presentation, domain та data встановлено чіткі інтерфейсні зв’язки через репозиторії.

7. Реалізація повноцінного UI з адаптивною навігацією:

– Впроваджено бібліотеки Navigation Component для переходів між фрагментами (MainFragment, AddCryptoFragment, DetailCryptoFragment).

– Додаток використовує компоненти Material Design для створення інтуїтивно зрозумілого інтерфейсу, а також Picasso - для завантаження графіки.

8. Механізм оновлення та сортування інформації:

– Передбачено автоматичне та ручне оновлення списку криптовалют через механізм swipe-to-refresh (SwipeRefreshLayout у MainFragment).

– Сортування монет здійснюється за ринковою вартістю за спаданням для кращої візуальної аналітики.

9. Тестування та контроль стабільності додатку - у проєкті застосовано бібліотеки JUnit та Espresso (за специфікацією) для забезпечення модульного та UI-тестування окремих компонентів.

У сукупності, зазначені вимоги відображають реальну функціональну реалізацію мобільного додатку і спрямовані на забезпечення надійного та продуктивного інструменту для користувачів, зацікавлених в оперативному моніторингу та аналізі криптовалютних активів.

Таблиця 3

Функціональні вимоги та їх реалізація в коді

№	Функціональна вимога	Реалізація в коді
1	Отримання HTML-даних з криптовалютних бірж	Інтерфейс CryptoAPI (getCoinHtmlByAPI(), getCryptoCurrenciesHTMLByAPI()) з використанням Retrofit
2	Парсинг HTML-даних для вилучення фінансових показників	GetCoinHTMLUseCase, GetCryptoCurrenciesHTMLUseCase + Jsoup.parse()
3	Структурування даних у вигляді моделей	Класи CryptoCoinModel, CryptoCurrencyModel, CryptoCurrenciesModel
4	Збереження монет у локальній базі	Клас CoinsTable, інтерфейс DAO, репозиторій CryptoDBRepositoryImpl
5	Додавання нових монет до бази	AddCoinUseCase, InsertCoinsViewModel, метод addNewCoin()
6	Перевірка наявності монети в базі перед додаванням	CheckCoinExistUseCase, метод checkCoinExist()
7	Видалення криптовалют зі списку користувача	DeleteCoinUseCase, DeleteCoinsViewModel
8	Інтелектуальний аналіз криптовалюти з використанням ШІ	GeminiUseCase, GeminiViewModel, інтеграція з GenerativeModel
9	Збереження історії доданих монет і оновлення цін	UpdateCoinsUseCase, LoadCryptoCoinsViewModel, updateCoinsDB()
10	Пагінація та динамічне завантаження списків монет	GetPagedCoinsUseCase, MainFragment, AddCryptoFragment, Paging, RecyclerView.OnScrollListener
11	Навігація між екранами (головний, додавання, деталі)	Navigation Component, MainFragment, AddCryptoFragment, DetailCryptoFragment
12	Побудова архітектури MVVM із розділенням відповідальностей	ViewModels + UseCases + Repositories, ін’єкція залежностей через Koin
13	Інтерфейс користувача на	MaterialButton, SwipeRefreshLayout, Picasso, компоненти

	основі Material Design	Binding
14	Оновлення даних у реальному часі	LoadCryptoCoinsViewModel.loadCoinsData(), SwipeRefreshLayout.setOnRefreshListener()
15	Автоматизоване тестування логіки додатку	Передбачено підтримку бібліотек JUnit, Espresso

У процесі реалізації розробленого мобільного додатку одним із ключових завдань виступала організація ефективного та гнучкого збору актуальної ринкової інформації з криптовалютних бірж. Оскільки значна частина фінансових показників публікується у вигляді веб-сторінок без відкритих повнофункціональних API-доступів або з істотними обмеженнями на стороні публічних API сервісів, особливу увагу в розробці було приділено методу парсингу HTML-документів для отримання необхідних даних.

Основним джерелом ринкової інформації у реалізованій системі виступає платформа CoinMarketCap, яка акумулює та агрегує дані з численних криптовалютних бірж та формує єдину централізовану базу котирувань, обсягів торгів, капіталізації та інших фінансових індикаторів. З метою отримання доступу до повного обсягу представленої інформації у додатку реалізовано парсинг HTML-сторінок CoinMarketCap із використанням бібліотеки Jsoup.

Отримання HTML-контенту з CoinMarketCap реалізовано через інтерфейс CryptoAPI, у якому визначено структуру HTTP-запитів до конкретних сторінок криптовалют:

```
@GET("currencies/{coin}")
suspend fun getCoinHtmlByAPI(
    @Path("coin") coinName: String
): Response<String>
```

Запити до CoinMarketCap здійснюються за допомогою бібліотеки Retrofit, яка виступає транспортним клієнтом для формування HTTP-з’єднань та отримання сирих HTML-рядків у вигляді Response<String>. Перевагою такого підходу є гнучкість роботи з текстовими відповідями, незалежність від форматів API, а також можливість зчитування повної структури документа для подальшого аналізу.

Отримані HTML-сторінки передаються до парсера, реалізованого через бібліотеку Jsoup. Основна функція парсера — обробити структуру DOM-документа та вилучити з неї лише ті дані, які є критично важливими для аналізу ринкових показників монет.

Приклад вилучення фінансових метрик з HTML-контенту:

```
val marketCap = document.selectFirst("dt:contains(Market cap) + dd span")?.text()
val volume24h = document.selectFirst("dt:contains(Volume (24h)) + dd span")?.text()
val fdv = document.selectFirst("dt:contains(FDV) + dd span")?.text()
val totalSupply = document.selectFirst("dt:contains(Total supply) + dd span")?.text()
val circulatingSupply = document.selectFirst("dt:contains(Circulating supply) + dd span")?.text()
val marketPrice = document.selectFirst("span[data-test='text-cdp-price-display']")?.text()
val coinName = document.selectFirst("span[data-role='coin-name']")?.attr("title")
val coinImg = document.selectFirst("div.sc-65e7f566-0 img")?.attr("src").toString()
```

У вищенаведеному прикладі використовується синтаксис CSS-селекторів, що дозволяє гнучко та точно ідентифікувати позицію кожного необхідного елемента у структурі сторінки. Це забезпечує вилучення широкого спектру даних — від поточних цін до кількості активів в обігу та обсягів торгів.

Крім парсингу інформації для окремих монет, у додатку реалізовано парсинг таблиць зі списками монет. Для цього також застосовується Jsoup, але із використанням групових селекторів:

```
val table = document.selectFirst("table.sc-936354b2-3.tLXcG.cmc-table")
val rows = table?.select("tr") ?: return null
```

```
val cryptoCoins = rows.mapNotNull { row ->
    val coinName = row.selectFirst("p.coin-item-name")?.text()
    val marketPrice = row.selectFirst("div.sc-b3fc6b7-0 span")?.text()
    val coinImg = row.selectFirst("div.sc-65e7f566-0 img")?.attr("src").toString()
    ...
}
```

Таким чином забезпечується можливість динамічного завантаження великих списків монет із підтримкою пагінації шляхом передачі параметрів сторінок у HTTP-запитах:

```
@GET("/")
suspend fun getCryptoCurrenciesHTMLByAPI(
    @Query("page") page: Int
): Response<String>
```

У реалізації системи парсингу особлива увага приділяється стійкості до помилок. Усі операції вилучення даних обгортаються у механізми обробки винятків (runCatching { ... }), що дозволяє запобігти аварійному завершенню роботи додатку у випадку зміни структури HTML-документів чи непередбачуваних мережевих збоїв:

```
val marketPrice = runCatching {
    document.selectFirst("span[data-test='text-cdp-price-display']")?.text()
        ?: throw Exception("price is null")
}.getOrElse { throw Exception("Failed to get market price: ${it.message}") }
```

Обраний підхід до інтеграції має низку суттєвих переваг:

- повна незалежність від API-обмежень сторонніх сервісів;
- доступ до розширених фінансових показників без необхідності купівлі додаткових API-пакетів;

- оперативна адаптація до змін в структурі даних;
- універсальність і масштабованість до різних біржових платформ.

Загалом методика парсингу HTML дозволила побудувати стабільну та високоточну систему збору ринкової інформації, яка не поступається офіційним API за функціональними можливостями, а в окремих аспектах навіть перевершує їх завдяки доступу до детальніших даних.

4.2. Розробка локальної бази даних для кешування даних

У системах моніторингу фінансових ринків, де обсяги даних є значними, а частота оновлення інформації — високою, використання локального кешування відіграє надзвичайно важливу роль. Локальна база даних у мобільному застосунку забезпечує зниження мережевого навантаження, покращення швидкодії інтерфейсу, стабільну роботу при обмеженому або відсутньому доступі до мережі та формує основу для реалізації механізмів синхронізації та актуалізації ринкових даних.

В основі реалізації локальної бази даних у розробленому мобільному застосунку лежить використання реляційної СУБД SQLite, що є стандартною для Android-платформи. Для спрощення взаємодії з базою даних, підвищення безпеки роботи з SQL-запитами та зменшення ризику помилок у запитах було обрано сучасну обгортку у вигляді бібліотеки Room Persistence Library - одного з ключових компонентів Android Jetpack.

Room забезпечує:

- автоматичне формування SQL-коду на основі анотацій у коді;
- перевірку правильності запитів на етапі компіляції;
- просту інтеграцію з потоковими API (Flow, LiveData);
- підтримку асинхронної роботи з базою через suspend-функції.

Основною структурною одиницею зберігання виступає сутність CoinsTable, яка відображає кожну монету у вигляді окремого запису таблиці:

```
@Entity(tableName = "coins")
data class CoinsTable(
    @PrimaryKey val coinName: String,
    val coinPrice: Float
)
```

Таким чином, у локальній базі зберігається назва монети як унікальний ідентифікатор та поточна її ринкова ціна. Структура таблиці спрощена, оскільки детальні параметри криптовалюти (капіталізація, обсяг торгів, пропозиція тощо) завантажуються під час відкриття деталізованої сторінки активу напряму з CoinMarketCap з актуальною інформацією.

Взаємодія із таблицею здійснюється через DAO-інтерфейс DAO, у якому визначено набір базових SQL-операцій:

- додавання нової монети:

```
@Insert(onConflict = OnConflictStrategy.REPLACE)
suspend fun insert(coin: CoinsTable): Long
```

- отримання монети по назві:

```
@Query("SELECT * FROM coins WHERE coinName = :coinName LIMIT 1")
suspend fun getCoin(coinName: String): CoinsTable?
```

- оновлення цін для групи монет:

```
@Update
suspend fun updateCoins(coins: List<CoinsTable>)
```

- видалення монети:

```
@Query("DELETE FROM coins WHERE coinName = :coinName")
suspend fun delete(coinName: String)
```

- вибірка даних із підтримкою пагінації:

```
@Query("SELECT * FROM coins LIMIT 5 OFFSET :offset")
```

suspend fun getCoinsWithPagination(offset: Int): List<CoinsTable>

Для організації доступу до бази на рівні бізнес-логіки створено репозиторій CryptoDBRepositoryImpl, який інкапсулює DAO-операції та забезпечує централізовану точку взаємодії між доменним шаром і сховищем даних. Наприклад, додавання нової монети до бази відбувається через:

```
override suspend fun addNewCoin(coin: CoinsTable): Long {
    return db.insert(coin)
}
```

Для демонстраційних цілей у додатку передбачено первинне заповнення бази при першому запуску додатку, що реалізується через механізм RoomDatabase.Callback:

db.execSQL("INSERT INTO coins (coinName, coinPrice) VALUES ('Bitcoin', '100000')")

db.execSQL("INSERT INTO coins (coinName, coinPrice) VALUES ('Ethereum', '4000')")

Такий підхід дозволяє забезпечити наявність тестових даних при першій ініціалізації додатку, навіть без підключення до зовнішнього джерела даних.

Оскільки локальні дані можуть втрачати актуальність, у додатку реалізовано механізм синхронізації цінових показників, що викликається при оновленні даних у ViewModel LoadCryptoCoinsViewModel. Після завантаження актуальної інформації з CoinMarketCap локальна база оновлюється командою:

repositoryDBRepository.updatePriceCoin(transformCoinsTypeList)

Таким чином забезпечується періодичне вирівнювання локального кешу з ринковими даними.

Впровадження Room забезпечило для системи:

- стабільність при обробці великих обсягів даних;
- збереження кешу даних для офлайн-роботи;
- оптимізацію навантаження на мережу;
- ефективну інтеграцію з реактивною архітектурою додатку;

– масштабованість для розширення моделі даних у майбутньому.

Таким чином, локальна база даних, розроблена на основі Room, стала фундаментальним компонентом програмної архітектури додатку, забезпечуючи ефективне зберігання, оновлення та обробку ринкових криптовалютних даних у мобільному середовищі.

4.3. Реалізація взаємодії класів мобільного додатку з використанням патерну MVVM та діаграма класів

Побудова архітектури мобільного додатку для моніторингу криптовалютних ринків реалізована з урахуванням сучасних практик багаторівневої архітектури, поєднуючи чисту архітектуру (Clean Architecture) із патерном Model-View-ViewModel (MVVM). Такий підхід дозволив забезпечити логічне розділення відповідальностей між компонентами системи, високу модульність, ізольованість бізнес-логіки та гнучкість при розширенні функціоналу додатку.

Основна концепція побудови взаємодії класів

У межах реалізованої архітектури усі основні класи додатку розділено на кілька функціональних рівнів, які пов’язуються між собою суворо через інтерфейси, залежності та спеціалізовані сервіси. Усі залежності інжектуються в класи за допомогою бібліотеки Koin, що забезпечує контрольовану передачу залежностей та слабкий зв’язок між компонентами.

- View (Fragments) — відповідає за відображення даних користувачу.
- ViewModel (Presentation Layer) — координує запити до бізнес-логіки та формує реактивний стан для View.
- UseCase (Domain Layer) — інкапсулює бізнес-правила та обробку сценаріїв взаємодії.

- Repositories (Data Layer) — управляють доступом до даних, взаємодією з API та локальною базою.
- Models (Entity Layer) — визначають структури даних на усіх рівнях системи.
- AI Module (Gemini Layer) — окремий модуль генеративної аналітики криптовалют.

Відображення архітектури на діаграмі класів

На побудованій UML-діаграмі класів, створеній за результатами аналізу коду, відображено основні класи додатку, їх атрибути, методи та зв’язки:

Моделі даних

- CryptoCoinModel — основна доменна модель для зберігання повної фінансової інформації про кожну криптовалюту: капіталізація, обсяг торгів, пропозиція, ціна тощо.
- CryptoCurrencyModel — спрощена модель для роботи зі списками монет (назва, ціна, зображення).
- CoinsTable — Room-сутність для зберігання кешованих монет у локальній базі.

Рівень репозиторіїв

- CryptoDBRepository та CryptoHtmlRepository — визначають абстрактні інтерфейси роботи з локальною базою та зовнішніми джерелами відповідно.
- CryptoDBRepositoryImpl — реалізує логіку роботи з Room-базою через DAO.
- CryptoHtmlHtmlRepositoryImpl — реалізує роботу з CoinMarketCap шляхом HTTP-запитів та парсингу HTML-контенту.

Бізнес-логіка (Use Cases)

Кожен сценарій взаємодії користувача або системної логіки винесено у окремі класи:

- AddCoinUseCase — додавання нової монети.

- DeleteCoinUseCase — видалення монети.
- GetCoinHTMLUseCase — парсинг HTML сторінки монети.
- GetCryptoCurrenciesHTMLUseCase — парсинг списку монет.
- UpdateCoinsUseCase — оновлення цін у локальній базі.
- CheckCoinExistUseCase — перевірка існування монети у базі.
- GetPagedCoinsUseCase — реалізація пагінації монет.
- GeminiUseCase — взаємодія з генеративною AI-моделлю Gemini для створення аналітичних висновків.

  ViewModel (MVVM Presentation Layer)
- PublicCoinsViewModel — завантаження списку монет з CoinMarketCap.
- LoadCryptoCoinsViewModel — керування завантаженням кешованих монет, оновленням цін, синхронізацією.
- InsertCoinsViewModel — додавання нових монет у локальну базу.
- DeleteCoinsViewModel — видалення монет.
- GeminiViewModel — обробка запитів до AI-модуля.

  API-Шар та взаємодія з CoinMarketCap
- CryptoAPI — опис HTTP-запитів через Retrofit до CoinMarketCap, включаючи отримання HTML контенту сторінок криптовалют.

  AI-аналітика
- GeminiUseCase — реалізує підготовку запитів до моделі Gemini.
- GenerativeModel — клас роботи з API Gemini від Google для генерації аналітичних звітів.

  Зв’язки між класами
- ViewModel напряму звертається до відповідних UseCase.
- UseCase взаємодіє з відповідними репозиторіями (абстракції Repository).
- Репозиторії реалізують усю логіку роботи з базою Room та мережевими джерелами Retrofit + Jsoup.
- AI UseCase отримує сформовану модель CryptoCoinModel, передає її до GenerativeModel для генерації текстової аналітики.

Детальна UML-діаграма класів, побудована за розробленим кодом, наочно ілюструє розподіл ролей між основними модулями додатку, демонструє слабкий зв’язок між компонентами та розділення відповідальностей, що є характерною ознакою чистої архітектури.

Рис. 4.1. Діаграма класів

Таким чином, побудована система класів забезпечує стабільність, масштабованість, зручність тестування, спрощену підтримку і можливість подальшого нарощування функціоналу додатку.

4.4 Реалізація функціоналу управління списком монет та інтеграція штучного інтелекту для аналізу криптовалют

Однією з ключових функціональних складових розробленого мобільного додатку є підсистема управління списком криптовалютних активів користувача із підтримкою локального зберігання даних та інтегрованої аналітики на основі генеративної моделі штучного інтелекту. Зазначена підсистема забезпечує повний цикл роботи з персональним списком монет: додавання, видалення, оновлення, синхронізацію даних та отримання рекомендацій на основі AI-аналізу.

Функціонал додавання криптовалют до персонального списку реалізований шляхом взаємодії кількох ViewModel-компонентів, таких як PublicCoinsViewModel, InsertCoinsViewModel, DeleteCoinsViewModel та LoadCryptoCoinsViewModel, які координують усі дії користувача та узгоджують звернення до відповідних UseCase-класів. Зокрема, у класі AddCoinUseCase реалізована логіка додавання нової монети до локальної бази даних. Перед додаванням здійснюється перевірка наявності монети у сховищі, яка виконується за допомогою виклику coinExist() у PublicCoinsViewModel. Для збереження нової монети використовується метод addNewCoin() у InsertCoinsViewModel, який звертається до AddCoinUseCase, де відбувається трансформація моделі CryptoCurrencyModel до об'єкта CoinsTable і подальше збереження у локальну базу Room:

```
suspend fun addCoin(coin: CryptoCurrencyModel): Boolean {
    return cryptoDBRepository.addNewCoin(coin.toCoinsTable()) != -1L
}
```

Функціонал видалення монет організовано через DeleteCoinsViewModel, який ініціює відповідний UseCase-клас DeleteCoinUseCase для видалення

монети з бази даних. Видалення виконується безпосередньо через DAO-компонент Room за назвою монети:

```
suspend fun deleteCoinDB(coinName: String) {
    cryptoDBRepository.deleteCoin(coinName)
}
```

Оскільки курси криптовалют постійно змінюються, реалізовано періодичну синхронізацію локального кешу з актуальними даними CoinMarketCap. Процес оновлення цін починається із запуску метода loadCoinsData() у LoadCryptoCoinsViewModel. Під час виконання синхронізації кожна монета по черзі обробляється через GetCoinHTMLUseCase, де за допомогою парсингу HTML-сторінок із CoinMarketCap вилучаються оновлені фінансові показники. Після збору нових даних формується колекція оновлених об'єктів CryptoCoinModel, які перетворюються до типу CoinsTable і передаються до UpdateCoinsUseCase для масового оновлення локальної бази:

```
suspend fun updateCoinsDB(coins: List<CryptoCoinModel>) {
    val transformCoinsTypeList = coins.map { coin -> coin.toCoinsTable() }
    repositoryDBRepository.updatePriceCoin(transformCoinsTypeList)
}
```

З метою розширення аналітичної функціональності системи у додатку інтегровано модуль генеративного аналізу криптовалют на основі моделі штучного інтелекту Gemini. Механізм аналітики побудовано за допомогою окремого UseCase-класу GeminiUseCase. Під час аналізу користувачем обраної криптовалюти об'єкт CryptoCoinModel, який містить всі фінансові показники монети, передається до AI-модуля через метод requestGemini(). У межах формування запиту до моделі формулюється промпт у текстовій формі, до якого додається сформовані дані по монеті. Приклад коду формування запиту наведений нижче:

```
emit(gemini.generateContent(ANALISE_CRYPTO_COIN + coin))
```

Особливістю сформованого запиту є чітке обмеження структури відповіді

AI, де передбачається короткий блок висновку та рекомендації для користувача, а також жорсткі обмеження на обсяг та символіку відповіді для отримання компактних та чистих аналітичних висновків.

Після отримання відповіді від моделі Gemini результат обробляється у ViewModel класі GeminiViewModel. Оброблена аналітична інформація зберігається у потоковій змінній StateFlow і передається до фрагменту DetailCryptoFragment для подальшого відображення користувачеві у зручній текстовій формі. Відображення результату реалізовано наступним чином:

binding.analiseText.text = it

Інтеграція AI-аналітики в архітектуру додатку значно розширює функціональні можливості системи, дозволяючи користувачу швидко отримувати персоналізовані фінансові висновки на основі складних ринкових показників без необхідності самостійного аналізу числових даних. Синхронна взаємодія усіх описаних модулів забезпечує стабільну роботу додатку, чіткий контроль потоків даних та оптимальну організацію складної бізнес-логіки завдяки застосуванню шаблону MVVM та принципів Clean Architecture.

4.5. Навігація та інтерфейс користувача

Інтерфейс користувача є одним з центральних компонентів функціональної частини розробленого мобільного додатку, оскільки саме він забезпечує кінцевому користувачеві доступ до усіх основних функціональних можливостей системи. Реалізація інтерфейсу здійснена із застосуванням сучасних рекомендацій Material Design та бібліотеки Jetpack Navigation Component для організації внутрішньої навігації додатку.

Організація екранних компонентів структурується навколо трьох основних фрагментів, кожен з яких відповідає за окремий функціональний

аспект роботи додатку. Фрагмент MainFragment виконує роль головного екрану, на якому відображається список персональних монет користувача. Завантаження списку монет здійснюється через ViewModel LoadCryptoCoinsViewModel, який передає дані до RecyclerView з підтримкою пагінації. Відображення списку монет реалізовано з використанням адаптера CryptoCoinsPrivateAdapter. Адаптер отримує дані у вигляді списку об’єктів CryptoCoinModel, які містять назву, ринкову ціну, загальну пропозицію та графічні зображення монет. Компонент RecyclerView конфігуровано для динамічного завантаження додаткових сторінок при досягненні кінця списку, що дозволяє уникати перевантаження інтерфейсу великими обсягами даних:

```
if (layoutManager.findLastCompletelyVisibleItemPosition() == cryptoAdapter.itemCount - 1) {
    loadCryptoCoinsViewModel.loadCoinsData()
}
```

Фрагмент AddCryptoFragment відповідає за додавання нових криптовалют до персонального списку моніторингу користувача. Він відображає загальний список доступних криптовалют, який завантажується безпосередньо із CoinMarketCap шляхом парсингу HTML-сторінок за допомогою GetCryptoCurrenciesHTMLUseCase. Дані подаються до RecyclerView із застосуванням адаптера CryptoCurrencyPublicAdapter. Додавання монети до персонального списку здійснюється через інтеграцію з ViewModel InsertCoinsViewModel, яка викликає AddCoinUseCase для запису нового об’єкта до локальної бази даних Room.

Фрагмент DetailCryptoFragment надає розширену інформацію по обраній монеті. На даному екрані відображаються ключові фінансові показники активу, включаючи обсяг торгів, капіталізацію, пропозицію, поточну ринкову ціну та аналітичний висновок, сформований генеративною AI-моделлю Gemini. Завантаження AI-аналітики здійснюється через GeminiViewModel, який ініціює звернення до модуля GeminiUseCase та передає отриману відповідь у

відповідний елемент інтерфейсу для візуалізації:

```
binding.analiseText.text = it
```

Всі переходи між фрагментами реалізовано із застосуванням бібліотеки Navigation Component, яка дозволяє централізовано описувати граф переходів між екранними компонентами, автоматично забезпечуючи безпечну передачу аргументів між фрагментами. Наприклад, при переході з MainFragment до DetailCryptoFragment для передачі об’єкта монети використовується механізм Safe Args, що гарантує типобезпеку передачі об’єктів між фрагментами:

```
val navController = findNavController()
navController.navigate(
    R.id.action_mainFragment_to_detailCryptoFragment,
    bundleOf("coin" to cryptoCoinModel)
)
```

Дизайн інтерфейсу користувача побудовано відповідно до принципів Material Design. Усі основні елементи - кнопки, панелі, форми, списки - використовують стандартні компоненти Android Material для забезпечення єдиного стилю та високого рівня юзабіліті. В оформленні застосовано інтуїтивно зрозумілу навігацію, анімації переходів між фрагментами, автоматичне реагування на зміну розмірів екрана, підтримку swipe-to-refresh та пагінації.

Загальна навігація та логіка взаємодії інтерфейсу користувача реалізовані згідно принципів реактивної архітектури. Всі потоки даних передаються через StateFlow та ViewModel, що дозволяє ефективно синхронізувати дані з інтерфейсом навіть у разі зміни стану додатку або ротації екрана.

Таким чином, побудований інтерфейс користувача дозволяє реалізувати складну систему управління фінансовими даними у простій, логічній та стабільній формі, з чіткою структурованою навігацією та інтелектуальними можливостями аналітики.

На рисунку 4.2 зображено головний екран мобільного додатку, який

виконує функцію моніторингу криптовалютних активів. Інтерфейс організовано таким чином, щоб забезпечити максимальну інформативність та зручність у користуванні. Центральним елементом даного екрану є перелік криптовалют, представлений у вигляді вертикального списку карток. Кожна картка відображає назву криптовалюти, її поточну ринкову ціну, графічну іконку монети та додаткові фінансові показники. Така структура дозволяє користувачу оперативно оцінювати динаміку ринку та отримувати актуальну інформацію по кожному активу без потреби переходу до додаткових меню. Завдяки використанню пагінації реалізовано можливість поступового завантаження великої кількості монет без перевантаження інтерфейсу, а підтримка swipe-to-refresh забезпечує швидке оновлення даних у режимі реального часу.

Користувач має можливість взаємодіяти з кожною позицією списку через інтегровану кнопку «Додати», що дозволяє включити обрану криптовалюту до персонального списку моніторингу або підготувати її для подальшого аналітичного оброблення.

Актуальні ринкові дані завантажуються з відкритих джерел шляхом інтеграції з API криптовалютних бірж, зокрема через отримання HTML-даних з ресурсу CoinMarketCap з подальшим парсингом необхідної інформації. Система підтримує оновлення даних у режимі реального часу, що дозволяє користувачеві отримувати максимально свіжу інформацію при кожному оновленні інтерфейсу.

Для оптимізації швидкодії та забезпечення автономної роботи додатку запроваджено збереження обраних користувачем криптовалют у локальній базі даних, реалізованій за допомогою бібліотеки Room. Такий підхід дозволяє мінімізувати кількість мережевих запитів, забезпечити швидкий доступ до основної інформації та покращити загальну продуктивність системи.

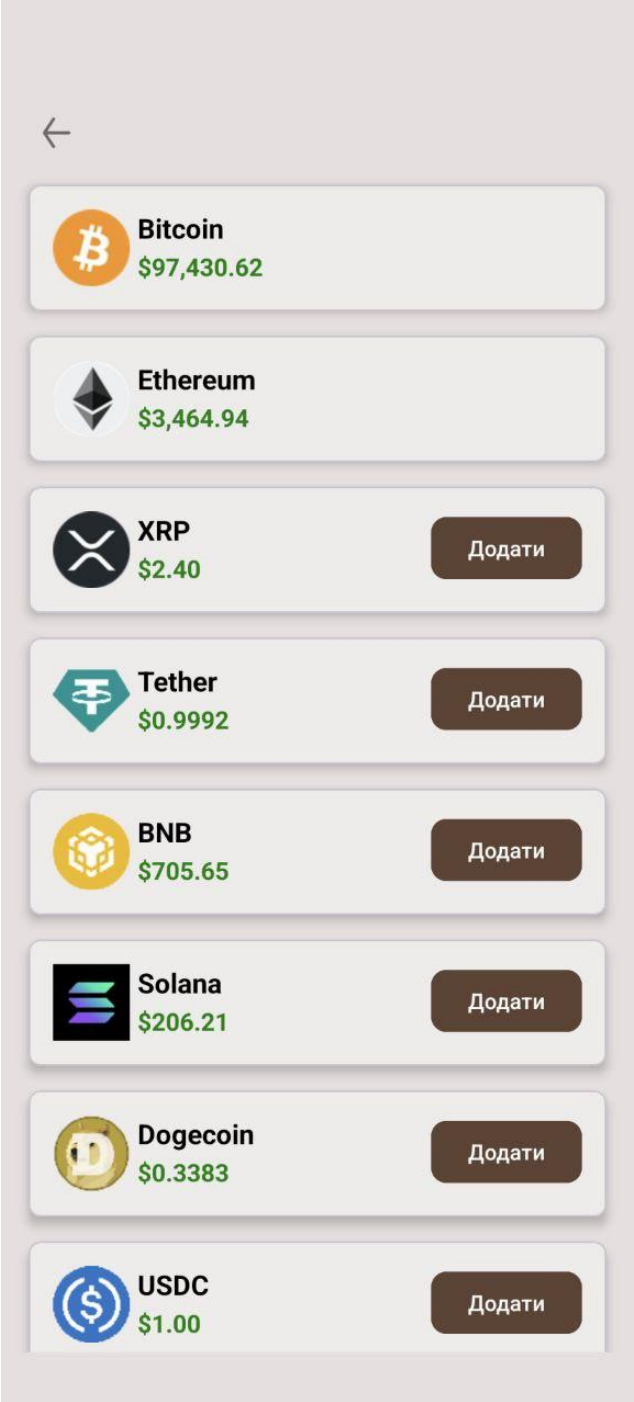

Рис. 4.2 Головний екран додатку

На рисунках 4.3.-4.4. представлено екрани детальної інформації про криптовалюту, який надає користувачеві розширену фінансову аналітику по обраному активу. Інтерфейс даного екрану структуровано таким чином, щоб забезпечити комплексний огляд основних ринкових характеристик конкретної криптовалюти.

На екрані відображається поточна ринкова ціна активу у доларах США, а також процентна зміна ціни, яка характеризує динаміку її вартості за обраний період часу. Додатково подається інформація щодо ринкової капіталізації активу, обсягу торгів за останні 24 години та кількості монет в обігу, що дозволяє отримати комплексне уявлення про ліквідність і стабільність конкретної криптовалюти.

Особливою функціональною складовою є інтегрована система штучного інтелекту на основі моделі Gemini AI 1.5, яка проводить аналітичну обробку отриманих ринкових даних. На основі аналізу основних фінансових показників система формує короткий текстовий висновок, в якому надається оцінка надійності криптовалюти, потенційних ризиків, стабільності ринкової позиції та загальних характеристик фінансової динаміки активу. Окрім висновку, формуються рекомендації, які можуть стосуватися доцільності довгострокового інвестування або проведення короткострокових торгових операцій із даною монетою.

Сформований текстовий аналіз відображається у відповідному інформаційному блоці без додаткового форматування, що забезпечує простоту сприйняття висновків користувачем. Для генерації висновків штучний інтелект використовує актуальні фінансові показники, отримані з відкритих біржових джерел у реальному часі через інтеграцію з API криптовалютних платформ.

Додатково реалізовано функціональність управління обраним списком монет: за допомогою інтегрованої кнопки користувач має можливість видалити відповідну криптовалюту зі списку персонального моніторингу, що дозволяє динамічно коригувати індивідуальний набір аналізованих активів.

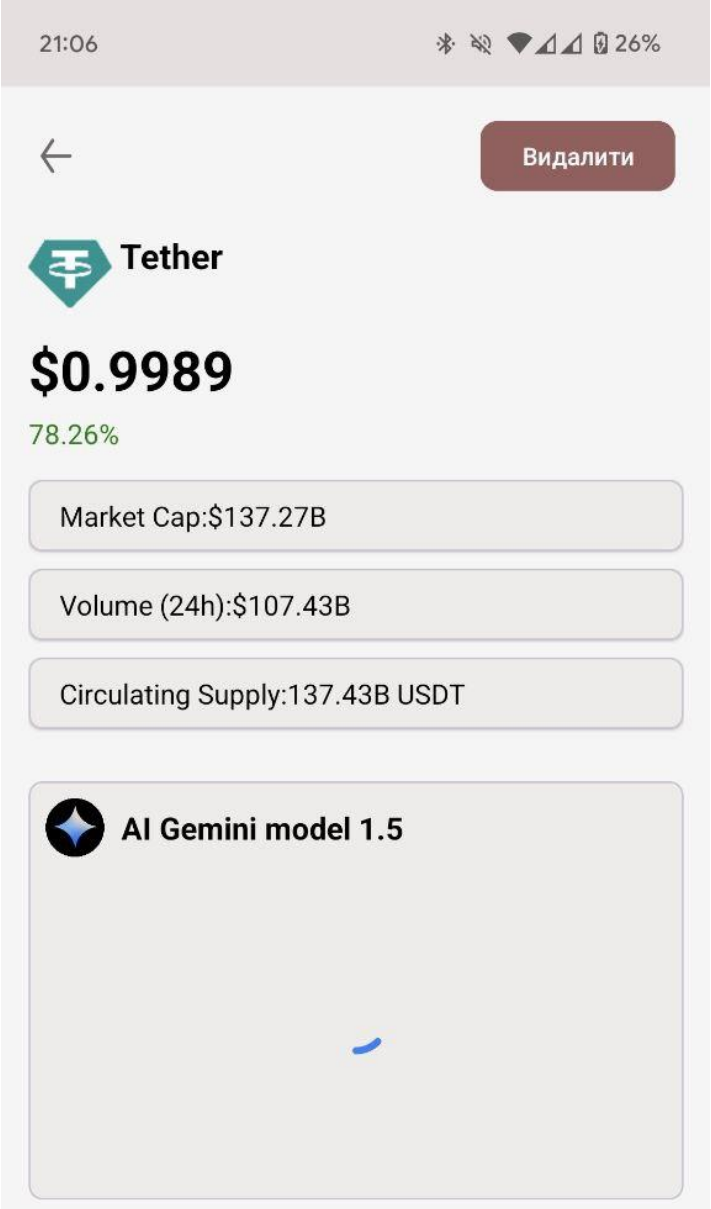

Рисунок 4.3. Інформація про криптовалюту

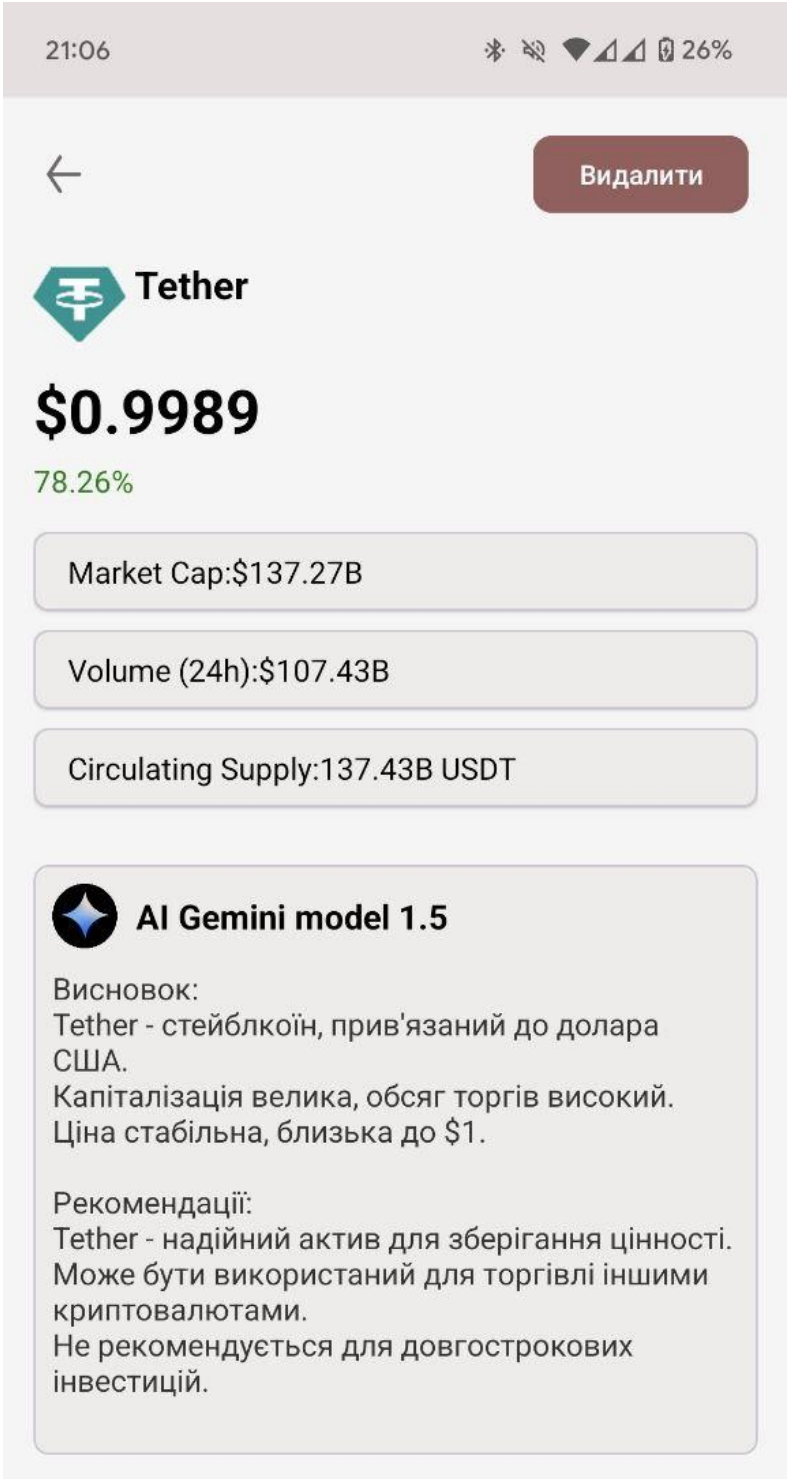

Рисунок 4.4. Інформація про криптовалюту і рекомендації на основі AI

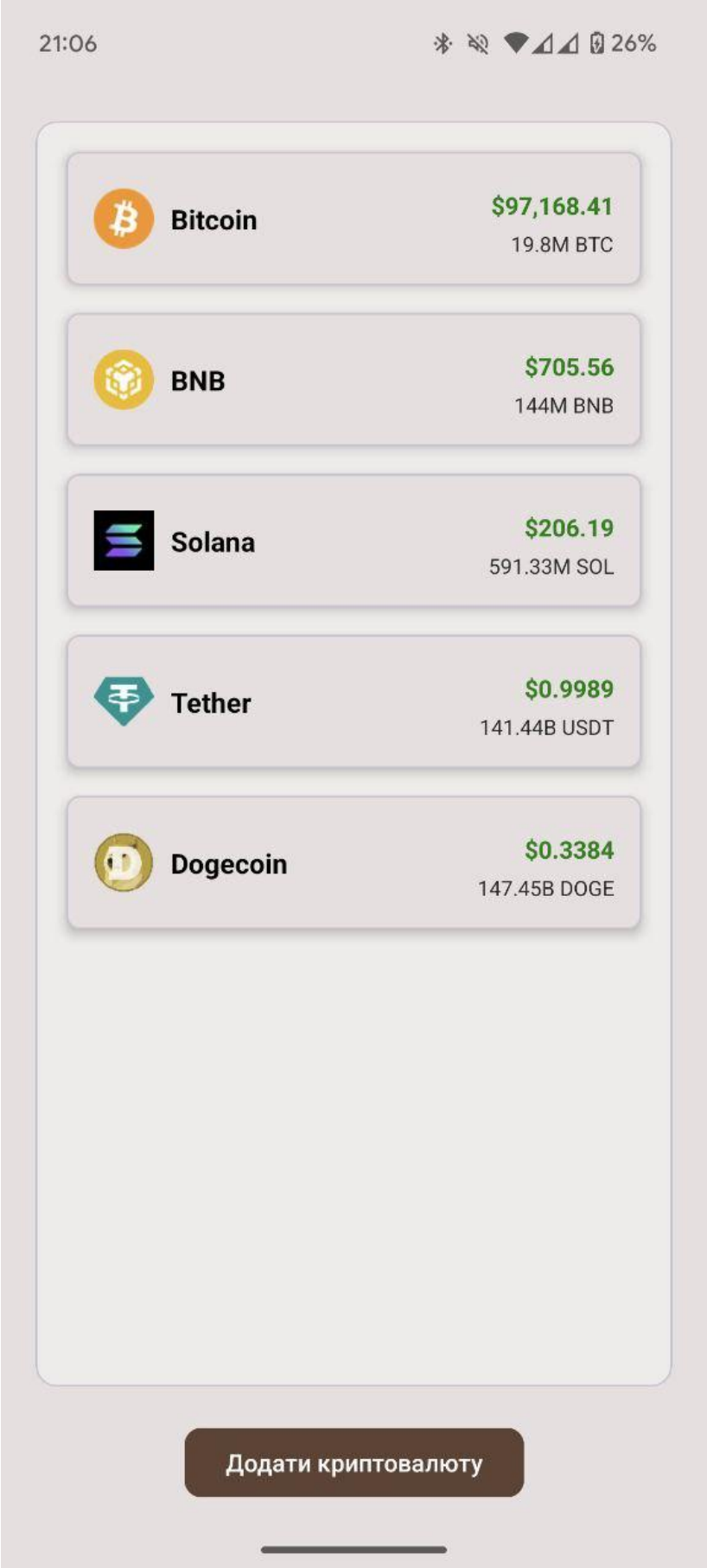

Рисунок 4.5. Екран додавання криптовалют

На рисунку 4.4 представлено екран обраного списку криптовалют, який містить ті активи, що були попередньо додані користувачем для персонального моніторингу. Даний інтерфейс дозволяє здійснювати цільове відстеження динаміки лише вибраних криптовалют, що оптимізує обсяг відображуваної інформації та підвищує зручність аналітичної роботи користувача.

У межах кожного елемента списку відображається актуальна ринкова ціна активу, виражена у доларах США, а також інформація про доступну кількість монет. Всі дані систематично оновлюються шляхом синхронізації локальної бази даних, реалізованої на основі бібліотеки Room. Такий підхід дозволяє забезпечити швидкий доступ до актуальної інформації навіть за умов тимчасової відсутності мережевого підключення, знижує навантаження на API-запити та оптимізує роботу системи при регулярному оновленні ринкових показників.

Користувачу також надається можливість розширення персонального списку криптовалют за допомогою інтегрованої кнопки «Додати криптовалюту». Перехід за даною кнопкою ініціює відкриття функціоналу додавання нових монет, де користувач має змогу обрати нові активи із загального ринкового списку для подальшого включення до персонального моніторингу.

Таким чином, функціонал персонального списку криптовалют у мобільному додатку забезпечує користувачеві зручний механізм формування власної вибірки фінансових активів для постійного моніторингу. Завдяки синхронізації з локальною базою даних реалізовано автономну та продуктивну роботу додатку з мінімальним навантаженням на мережеві ресурси. Можливість гнучкого розширення або коригування обраного списку монет створює персоналізоване середовище аналізу ринкових показників та дозволяє оперативно реагувати на зміну фінансових тенденцій, що є важливим для користувачів, які здійснюють регулярний моніторинг динаміки криптовалютних ринків.

ВИСНОВКИ

У процесі виконання дипломної роботи було розроблено мобільний Android-додаток для моніторингу та аналізу динаміки криптовалютних ринків, який дозволяє отримувати актуальні ринкові дані з відкритих джерел, зберігати їх у локальній базі даних, відображати користувачу у зручній формі та виконувати аналітичну обробку за допомогою моделей штучного інтелекту.

Під час виконання роботи були сформовані ключові вимоги до функціоналу додатку, зокрема — вимоги до ефективності обробки фінансових показників, зручності користувацького інтерфейсу, стабільності та масштабованості архітектурних рішень. Проведено порівняльний аналіз існуючих мобільних додатків для моніторингу криптовалют, що дозволило обґрунтувати вибір архітектури, технологічного стеку та інструментів реалізації програмного забезпечення.

У ході дослідження було реалізовано повноцінну архітектуру додатку з використанням чистої архітектури (Clean Architecture) та патерну MVVM, що забезпечило розділення відповідальностей між компонентами та сприяло побудові масштабованої системи. Розроблено моделі даних, репозиторії, use-case логіку, ViewModel-компоненти та механізми збереження й оновлення ринкових даних. Отримання фінансової інформації здійснюється шляхом парсингу HTML-контенту біржі CoinMarketCap за допомогою бібліотеки Jsoup у поєднанні з HTTP-клієнтом Retrofit. Локальне зберігання інформації організовано за допомогою бібліотеки Room із підтримкою пагінації для оптимізації роботи з великими обсягами даних.

Особливу увагу приділено інтеграції генеративної AI-аналітики на основі моделі Gemini, яка дозволяє здійснювати аналіз фінансових показників криптовалют та формувати короткі текстові висновки і рекомендації. Це

розширило функціональні можливості додатку, надавши користувачу не лише актуальну інформацію, а й автоматизований аналітичний супровід.

Створено зручний і сучасний користувацький інтерфейс із застосуванням стандартів Material Design та реалізацією навігаційної логіки через Navigation Component і Safe Args. Побудовано повний набір архітектурних схем і діаграм класів, що відображають логіку функціонування системи.

Проведено комплексне тестування додатку із залученням автоматизованих інструментів JUnit та Espresso, яке підтвердило коректність функціональної роботи додатку, стабільність обробки даних, ефективність алгоритмів оновлення даних, а також загальну надійність програмного забезпечення при роботі з реальними ринковими даними криптовалют.

Розроблений програмний продукт є готовим практичним рішенням для моніторингу та аналітики криптовалютних ринків і може бути використаний як основа для подальшого розвитку більш складних систем фінансового аналізу з широким залученням методів штучного інтелекту, автоматичного прогнозування та глибокої аналітики ринкових трендів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android Developers. Dependency Injection with Hilt [Електронний ресурс]. - Режим доступу: https://developer.android.com/training/dependency-injection/hilt

2. Android MVVM Architecture: Understanding ViewModel and LiveData [Електронний ресурс]. - Режим доступу: https://developer.android.com/topic/libraries/architecture/viewmodel

3. Clean Architecture. A detailed introduction by Robert C. Martin [Електронний ресурс]. - Режим доступу: https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html

4. Material Design Guidelines [Електронний ресурс]. - Режим доступу: https://material.io/design

5. Model-View-ViewModel (MVVM). Microsoft Documentation [Електронний ресурс]. - Режим доступу: https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm

6. Nature. AI integration in financial services: a systematic review [Електронний ресурс]. – Режим доступу: *https://www.nature.com/articles/s41599-025-04850-8*

7. Ninjapromo. Top 36 Crypto Tools for Trading, Analysis and Research in 2025 [Електронний ресурс]. – Режим доступу: *https://ninjapromo.io/best-crypto-tools-for-analysis-trading-research*

8. Picasso Image Loading Library for Android [Електронний ресурс]. - Режим доступу: https://square.github.io/picasso/

9. ProAndroidDev. Securing User Data in Android: Tips and Techniques [Електронний ресурс]. – Режим доступу: *https://proandroiddev.com/securing-user-data-in-android-tips-and-techniques-1b584fd5a14a*

10. ResearchGate. Applying Artificial Intelligence in Cryptocurrency Markets: A Survey [Електронний ресурс]. - Режим доступу:

https://www.researchgate.net/publication/365377903_Applying_Artificial_Intelligence_in_Cryptocurrency_Markets_A_Survey

11. ResearchGate. Cryptocurrency Analysis Using Artificial Intelligence [Електронний ресурс]. - Режим доступу: *https://www.researchgate.net/publication/373926096_Cryptocurrency_Analysis_Using_Artificial_Intelligence*

12. ResearchGate. Privacy and Security Analysis of Cryptocurrency Mobile Applications [Електронний ресурс]. - Режим доступу: *https://www.researchgate.net/publication/332374404_Privacy_and_Security_Analysis_of_Cryptocurrency_Mobile_Applications*

13. Retrofit: Type-Safe HTTP Client for Android and Java by Square, Inc [Електронний ресурс]. — Режим доступу: https://square.github.io/retrofit/

14. Retrofit2 Javadoc [Електронний ресурс]. - Режим доступу: https://square.github.io/retrofit/2.x/retrofit/

15. Sciencedirect. Anticipating cryptocurrency prices using machine learning [Електронний ресурс]. – Режим доступу: *https://arxiv.org/abs/1805.08550*

16. Sciencedirect. Artificial intelligence techniques in financial trading: A systematic review [Електронний ресурс]. – Режим доступу: *https://www.sciencedirect.com/science/article/pii/S1319157824001046*

17. Sciencedirect. Cryptocurrency Valuation: An Explainable AI Approach [Електронний ресурс]. – Режим доступу: *https://arxiv.org/abs/2201.12893*

18. Softteco. Best practices on securing Android apps [Електронний ресурс]. – Режим доступу: *https://softteco.com/blog/understanding-android-application-security*

19. UseCase Red Flags and Best Practices in Clean Architecture [Електронний ресурс]. - Режим доступу: https://engineering.teknasyon.com/usecase-red-flags-and-best-practices-in-clean-architecture-76e2f6d921eb

20. Кращі тренди розробки мобільних додатків у 2024 році [Електронний ресурс]. — Режим доступу: https://whileweb.com/uk/blog/top-trendiv-rozrobki-mobilnih-dodatkiv-u-2024-roci/

21. Ольховська О. В., Черненко О. О. Методичні рекомендації до виконання кваліфікаційної роботи для студентів спеціальності 122 Комп'ютерні науки освітня програма «Комп'ютерні науки» ступеня бакалавра. — Полтава: ПУЕТ, 2024. - 58 с. - Режим доступу: http://dspace.puet.edu.ua/bitstream/123456789/14768/1/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0%20%D0%91%D0%A0%20%D0%B4%D0%BB%D1%8F%20%D0%9A%D0%9D%202024_%D0%B1%D0%B0%D0%BA%D0%B0%D0%BB%D0%B0%D0%B2%D1%80.pdf

ДОДАТОК А
Код програми API

```
package com.student.cryptoanalitics.data.api

import okhttp3.ResponseBody
import retrofit2.Call
import retrofit2.Response
import retrofit2.http.GET
import retrofit2.http.Path
import retrofit2.http.Query

interface CryptoAPI {

    companion object {
        const val BASE_URL_CRYPTO = "https://coinmarketcap.com/"
    }

    @GET("currencies/{coin}")
    suspend fun getCoinHtmlByAPI(
        @Path("coin") coinName: String
    ): Response<String>

    @GET("/")
    suspend fun getCryptoCurrenciesHTMLByAPI(
        @Query("page") page: Int
    ): Response<String>

}


package com.student.cryptoanalitics.data

import androidx.paging.PagingSource
import com.student.cryptoanalitics.App.Companion.mylog
import com.student.cryptoanalitics.data.database.DAO
import com.student.cryptoanalitics.data.database.table.CoinsTable
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel
import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository

class CryptoDBRepositoryImpl(private val db: DAO) : CryptoDBRepository {

    override suspend fun getAllCoinPagination(offset: Int): List<CoinsTable> {
        val response = db.getCoinsWithPagination(offset)
        mylog(response)
        return response
    }

    override suspend fun addNewCoin(coin: CoinsTable): Long {
        return db.insert(coin)
```

```
    }

    override suspend fun checkCoinExist(coinName: String): CoinsTable? {
        return db.getCoin(coinName)
    }

    override suspend fun updatePriceCoin(coins: List<CoinsTable>) {
        db.updateCoins(coins)
    }

    override suspend fun deleteCoin(coinName: String) {
        db.delete(coinName)
    }

}


package com.student.cryptoanalitics.data

import com.student.cryptoanalitics.data.api.CryptoAPI
import com.student.cryptoanalitics.domain.repositories.CryptoHtmlRepository
import retrofit2.Response

class CryptoHtmlHtmlRepositoryImpl(private val cryptoAPI: CryptoAPI) : CryptoHtmlRepository
{

    override suspend fun getCoinHtml(coinName: String): Response<String> {
        return cryptoAPI.getCoinHtmlByAPI(coinName)
    }

    override suspend fun getCryptoCurrenciesHtml(page: Int): Response<String> {
        return cryptoAPI.getCryptoCurrenciesHTMLByAPI(page)
    }

}

ackage com.student.cryptoanalitics.domain.models.currencies

data class CryptoCurrenciesModel(
    val coinList: List<CryptoCurrencyModel>
)

package com.student.cryptoanalitics.domain.models.currencies

data class CryptoCurrencyModel(
    val coinName: String,
    val coinPrice: String,
    val coinImg: String
)
```

```
package com.student.cryptoanalitics.domain.models

import android.os.Parcelable
import kotlinx.android.parcel.Parcelize

@Parcelize
data class CryptoCoinModel(
    val marketCap: String? = null,
    val volume24h: String? = null,
    val fdv: String? = null,
    val volMktCap: String? = null,
    val totalSupply: String? = null,
    val circulatingSupply: String? = null,
    val marketPrice: String? = null,
    val coinImg: String? = null,
    val coinName: String
) : Parcelable

package com.student.cryptoanalitics.domain.repositories

import androidx.paging.PagingSource
import com.student.cryptoanalitics.data.database.table.CoinsTable
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel
import kotlinx.coroutines.flow.Flow
import retrofit2.Call
import retrofit2.Response

interface CryptoDBRepository {
    suspend fun getAllCoinPagination(offset: Int): List<CoinsTable>
    suspend fun addNewCoin(coin: CoinsTable): Long
    suspend fun checkCoinExist(coinName: String): CoinsTable?
    suspend fun updatePriceCoin(coins: List<CoinsTable>)
    suspend fun deleteCoin(coinName: String)
}

package com.student.cryptoanalitics.domain.repositories

import retrofit2.Response

interface CryptoHtmlRepository {

    suspend fun getCoinHtml(coinName: String): Response<String>
    suspend fun getCryptoCurrenciesHtml(page: Int): Response<String>
}

package com.student.cryptoanalitics.domain.usecases

import com.student.cryptoanalitics.data.database.table.CoinsTable
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel
import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository
```

```
class AddCoinUseCase(private val cryptoDBRepository: CryptoDBRepository) {

    suspend fun addCoin(coin: CryptoCurrencyModel): Boolean {
        return cryptoDBRepository.addNewCoin(
            coin.toCoinsTable()
        ) != -1L
    }

    private fun CryptoCurrencyModel.toCoinsTable(): CoinsTable {
        val priceParsed = coinPrice.split("$")[1].replace(",", "").toFloat()

        return CoinsTable(
            coinName = coinName,
            coinPrice = priceParsed
        )
    }

}
```

```
package com.student.cryptoanalitics.domain.usecases

import com.student.cryptoanalitics.App.Companion.mylog
import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository
import java.util.Locale

class CheckCoinExistUseCase(private val cryptoDBRepository: CryptoDBRepository) {

    suspend fun checkCoinExist(coinName: String): Boolean {
        return cryptoDBRepository.checkCoinExist(coinName) != null
    }

}
```

```
package com.student.cryptoanalitics.domain.usecases

import com.student.cryptoanalitics.App.Companion.mylog
import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository
import java.util.Locale

class CheckCoinExistUseCase(private val cryptoDBRepository: CryptoDBRepository) {

    suspend fun checkCoinExist(coinName: String): Boolean {
        return cryptoDBRepository.checkCoinExist(coinName) != null
    }

}
```

```
package com.student.cryptoanalitics.domain.usecases
```

```
import com.google.ai.client.generativeai.GenerativeModel
import com.google.ai.client.generativeai.type.GenerateContentResponse
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow

class GeminiUseCase(private val gemini: GenerativeModel) {

    fun requestGemini(coin: CryptoCoinModel): Flow<GenerateContentResponse> {
        return flow {
            emit(gemini.generateContent(ANALISE_CRYPTO_COIN + coin))
        }
    }

    companion object {
        const val DEFAULT_RESPONSE = "Виникла помилка під час аналізу"
        private const val ANALISE_CRYPTO_COIN =
            "Проаналізуй показники криптовалюти та надай короткий висновок і рекомендацію у вигляді текстового блоку. Обмеж відповідь 250 символами. Використовуй пробіли, відступи або перенесення на новий рядок для зрозумілого форматування тексту, інші символи заборонені. Уникай зайвого тексту та складних конструкцій.\n" +
                    "Відповідь у форматі:\n" +
                    "\nВисновок: тут інформація про твій аналіз без символів *,# і тд" +
                    "\nРекомендації: тут інформація про твою ррекомендацію без символів *,# і тд" +
                    "\nКриптовалюта по який робити аналіз:"
    }
}

package com.student.cryptoanalitics.domain.usecases

import com.student.cryptoanalitics.App.Companion.mylog
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository
import com.student.cryptoanalitics.domain.repositories.CryptoHtmlRepository
import org.jsoup.Jsoup
import org.jsoup.nodes.Document


class GetCoinHTMLUseCase(
    private val cryptoHtmlRepository: CryptoHtmlRepository
) {

    suspend fun getCoinData(coinName: String): CryptoCoinModel? {
        return try {
                val response = cryptoHtmlRepository.getCoinHtml(coinName)
                if (response.isSuccessful && response.body() != null) {
                    parseHtmlCoin(response.body()!!)
                } else {
                    throw Exception("Failed to fetch data: ${response.errorBody()?.string()}")
                }
            } catch (e: Exception) {
```

```
            return null
        }
    }

    private fun parseHtmlCoin(content: String): CryptoCoinModel? {
        return try {
            val document: Document = Jsoup.parse(content)

            val marketCap = document.selectFirst("dt:contains(Market cap) + dd span")?.text()
            val volume24h = document.selectFirst("dt:contains(Volume (24h)) + dd span")?.text()
            val fdv = document.selectFirst("dt:contains(FDV) + dd span")?.text()
            val volMktCap = document.selectFirst("dt:contains(Vol/Mkt Cap (24h)) + dd")?.text()
            val totalSupply = document.selectFirst("dt:contains(Total supply) + dd span")?.text()
            val circulatingSupply = document.selectFirst("dt:contains(Circulating supply) + dd span")?.text()

            val marketPrice = runCatching {
                document.selectFirst("span[data-test='text-cdp-price-display']")?.text()
                    ?: throw Exception("price is null")
            }.getOrElse { throw Exception("Failed to get market price: ${it.message}") }

            val coinName = runCatching {
                document.selectFirst("span[data-role='coin-name']")?.attr("title")
                    ?: throw Exception("name is null")
            }.getOrElse { throw Exception("Failed to get coin name: ${it.message}") }

            val coinImg = document.selectFirst("div.sc-65e7f566-0 img")?.attr("src").toString()


            val model = CryptoCoinModel(
                marketCap = marketCap,
                volume24h = volume24h,
                fdv = fdv,
                volMktCap = volMktCap,
                totalSupply = totalSupply,
                circulatingSupply = circulatingSupply,
                marketPrice = marketPrice,
                coinName = coinName,
                coinImg = coinImg
            )
            model
        } catch (e: Exception) {
            mylog(e)
            null
        }
    }


}

package com.student.cryptoanalitics.domain.usecases
```

```
import com.student.cryptoanalitics.App.Companion.mylog
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrenciesModel
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel
import com.student.cryptoanalitics.domain.repositories.CryptoHtmlRepository
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow
import org.jsoup.Jsoup
import org.jsoup.nodes.Document


class GetCryptoCurrenciesHTMLUseCase(private val cryptoHtmlRepository: CryptoHtmlRepository) {

    fun getCoinsList(page: Int): Flow<CryptoCurrenciesModel?> {
        return flow {
            try {
                val response = cryptoHtmlRepository.getCryptoCurrenciesHtml(page)
                if (response.isSuccessful && response.body() != null) {
                    emit(parseHtmlCryptoCurrencies(response.body()!!))
                } else {
                    emit(null)
                    throw Exception("Failed to fetch data: ${response.errorBody()?.string()}")
                }
            } catch (e: Exception) {
                emit(null)
                throw e
            }
        }
    }

    private fun parseHtmlCryptoCurrencies(content: String): CryptoCurrenciesModel? {
        return try {
            val document: Document = Jsoup.parse(content)

            val table = document.selectFirst("table.sc-936354b2-3.tLXcG.cmc-table")

            val rows = table?.select("tr") ?: return null

            val cryptoCoins = rows.mapNotNull { row ->
                val coinName = row.selectFirst("p.coin-item-name")?.text() // Назва монети
                val marketPrice = row.selectFirst("div.sc-b3fc6b7-0 span")?.text() // Ціна монети
                val coinImg = row.selectFirst("div.sc-65e7f566-0 img")?.attr("src").toString()

                if (coinName != null && marketPrice != null) {
                    CryptoCurrencyModel(
                        coinName = coinName,
                        coinPrice = marketPrice,
                        coinImg = coinImg
                    )
                } else {
```

```
                null
            }
        }

//          mylog(cryptoCoins)
        CryptoCurrenciesModel(cryptoCoins)
    } catch (e: Exception) {
        mylog(e)
        null
    }
}



}

package com.student.cryptoanalitics.domain.usecases

import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flow

class GetPagedCoinsUseCase(private val repositoryDBRepository: CryptoDBRepository) {

    fun execute(page: Int): Flow<List<String>> {
        return flow {
            emit(repositoryDBRepository.getAllCoinPagination(page * 5 - 5).map { it.coinName })
        }
    }

}

package com.student.cryptoanalitics.domain.usecases

import com.student.cryptoanalitics.data.database.table.CoinsTable
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository

class UpdateCoinsUseCase(private val repositoryDBRepository: CryptoDBRepository) {

    suspend fun updateCoinsDB(coins: List<CryptoCoinModel>) {
        val transformCoinsTypeList = coins.map { coin -> coin.toCoinsTable() }
        repositoryDBRepository.updatePriceCoin(transformCoinsTypeList)
    }

    private fun CryptoCoinModel.toCoinsTable(): CoinsTable {
        val priceParsed = marketPrice!!.split("$")[1].replace(",", "").toFloat()

        return CoinsTable(
            coinName = coinName,
            coinPrice = priceParsed
```

```
        )
    }

}

package com.student.cryptoanalitics.presentation.adapters.my_coins

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.squareup.picasso.Picasso
import com.student.cryptoanalitics.databinding.MycryptoItemBinding
import com.student.cryptoanalitics.domain.models.CryptoCoinModel

class CryptoCoinsPrivateAdapter(private val click: CryptoCoinsPrivateClick) :
    ListAdapter<CryptoCoinModel, CryptoCoinsPrivateAdapter.CryptoViewHolder>(
        DiffCallback
    ) {
    inner class CryptoViewHolder(private val binding: MycryptoItemBinding) :
        RecyclerView.ViewHolder(binding.root) {

        fun bind(coin: CryptoCoinModel) {
            binding.coinName.text = coin.coinName
            binding.marketPrice.text = coin.marketPrice
            binding.totalSupply.text = coin.totalSupply
            Picasso.get().load(coin.coinImg).into(binding.coinImg)

            itemView.setOnClickListener {
                click.clickCoin(coin)
            }
        }
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): CryptoViewHolder {
        val binding = MycryptoItemBinding.inflate(
            LayoutInflater.from(parent.context), parent, false
        )
        return CryptoViewHolder(binding)
    }

    override fun onBindViewHolder(holder: CryptoViewHolder, position: Int) {
        getItem(position)?.let { holder.bind(it) }
    }

    companion object {
        private val DiffCallback = object : DiffUtil.ItemCallback<CryptoCoinModel>() {
            override fun areItemsTheSame(
                oldItem: CryptoCoinModel,
                newItem: CryptoCoinModel
```

```
        ): Boolean {
            return oldItem.coinName == newItem.coinName
        }

        override fun areContentsTheSame(
            oldItem: CryptoCoinModel,
            newItem: CryptoCoinModel
        ): Boolean {
            return oldItem == newItem
        }
    }
}
```

```
package com.student.cryptoanalitics.presentation.adapters.my_coins

import android.widget.RadioButton
import com.google.android.material.button.MaterialButton
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel

interface CryptoCoinsPrivateClick {
    fun clickCoin(cryptoCoinModel: CryptoCoinModel)
}
```

```
package com.student.cryptoanalitics.presentation.adapters.public_coins

import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.DiffUtil
import androidx.recyclerview.widget.ListAdapter
import androidx.recyclerview.widget.RecyclerView
import com.squareup.picasso.Picasso
import com.student.cryptoanalitics.databinding.CryptopublicItemBinding
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel

class CryptoCurrencyPublicAdapter(private val click: CryptoCurrencyPublicClick) :
    ListAdapter<CryptoCurrencyModel,
CryptoCurrencyPublicAdapter.CryptoCurrencyViewHolder>(
        CoinsDiffCallback
    ) {

    inner class CryptoCurrencyViewHolder(private val binding: CryptopublicItemBinding) :
RecyclerView.ViewHolder(binding.root) {
        fun bind(crypto: CryptoCurrencyModel) {
            binding.coinName.text = crypto.coinName
            binding.marketPrice.text = crypto.coinPrice
            Picasso.get().load(crypto.coinImg).into(binding.cryptoImg)

            click.addCrypto(binding.addCrypto, crypto)
```

```
        }
    }

package com.student.cryptoanalitics.presentation.adapters.public_coins

import com.google.android.material.button.MaterialButton
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel

interface CryptoCurrencyPublicClick {
    fun addCrypto(button: MaterialButton, cryptoCurrencyModel: CryptoCurrencyModel)
}

package com.student.cryptoanalitics.presentation.vm

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.student.cryptoanalitics.domain.usecases.DeleteCoinUseCase
import kotlinx.coroutines.launch

class DeleteCoinsViewModel(private val deleteCoinUseCase: DeleteCoinUseCase) : ViewModel()
{

    fun deleteCoin(coinName: String) {
        viewModelScope.launch {
            deleteCoinUseCase.deleteCoinDB(coinName)
        }
    }

}

package com.student.cryptoanalitics.presentation.vm

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.usecases.GeminiUseCase
import
com.student.cryptoanalitics.domain.usecases.GeminiUseCase.Companion.DEFAULT_RESPONSE
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

class GeminiViewModel(private val geminiUseCase: GeminiUseCase): ViewModel() {

    private val _analiseCoinGemini: MutableStateFlow<String?> = MutableStateFlow(null)
    val analiseCoinGemini: StateFlow<String?> = _analiseCoinGemini


    fun requestAnaliseCoin(coinModel: CryptoCoinModel) {
        viewModelScope.launch {
            geminiUseCase.requestGemini(coinModel).collect {
```

```
                _analiseCoinGemini.value = it.text ?: DEFAULT_RESPONSE
            }
        }
    }

}

package com.student.cryptoanalitics.presentation.vm

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel
import com.student.cryptoanalitics.domain.usecases.AddCoinUseCase
import com.student.cryptoanalitics.domain.usecases.GetCoinHTMLUseCase
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch

class InsertCoinsViewModel(
    private val addCoinUseCase: AddCoinUseCase,
) : ViewModel() {

    fun addNewCoin(coin: CryptoCurrencyModel, callback: (Boolean) -> Unit) {
        viewModelScope.launch {
            callback(addCoinUseCase.addCoin(coin))
        }
    }

}

package com.student.cryptoanalitics.presentation.vm

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.student.cryptoanalitics.App.Companion.mylog
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.domain.usecases.GetCoinHTMLUseCase
import com.student.cryptoanalitics.domain.usecases.GetPagedCoinsUseCase
import com.student.cryptoanalitics.domain.usecases.UpdateCoinsUseCase
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

class LoadCryptoCoinsViewModel(
    private val getPagedCoinsUseCase: GetPagedCoinsUseCase,
    private val getCoinHTMLUseCase: GetCoinHTMLUseCase,
    private val updateCoinsUseCase: UpdateCoinsUseCase
) : ViewModel() {

    private val coinsFlowMutable: MutableStateFlow<List<CryptoCoinModel>> =
```

```
        MutableStateFlow(emptyList())
    val coinsFlow: StateFlow<List<CryptoCoinModel>> = coinsFlowMutable

    private val _isLoading = MutableStateFlow(false)
    val isLoading: StateFlow<Boolean> = _isLoading

    private val _isRefreshing = MutableStateFlow(false)
    val isRefreshing: StateFlow<Boolean> = _isRefreshing

    private var currentPage = 1

    init {
        if (currentPage == 1) {
            loadCoinsData(true)
        }
    }

    fun loadCoinsData(isRefresh: Boolean = false, immediate: Boolean = false) {
        if (_isLoading.value || _isRefreshing.value) if (!immediate) return


        if (isRefresh) {
            currentPage = 1
            coinsFlowMutable.value = emptyList()
            _isRefreshing.value = true
        } else {
            _isLoading.value = true
        }

        viewModelScope.launch(Dispatchers.IO) {
            getPagedCoinsUseCase.execute(currentPage).collect { listData ->

                val updatedCoinsList = mutableListOf<CryptoCoinModel>()

                listData.forEach { coinName ->
                    getCoinHTMLUseCase.getCoinData(coinName)?.let { coin ->
                        updatedCoinsList.add(coin)
                    }
                }

                // here updating database *********
                viewModelScope.launch(Dispatchers.IO) {
                    updateCoinsUseCase.updateCoinsDB(updatedCoinsList)
                }
                // here updating database *********

                val updatedList = coinsFlowMutable.value + updatedCoinsList
                coinsFlowMutable.value = updatedList.distinctBy { coin ->
coin.marketPrice!!.split("$")[1].replace(",", "").toFloat() }
                    .sortedByDescending { coin -> coin.marketPrice!!.split("$")[1].replace(",",
"").toFloat() }
```

```
                currentPage++

                _isLoading.value = false
                _isRefreshing.value = false
            }

        }
    }

}


package com.student.cryptoanalitics.presentation.vm

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrenciesModel
import com.student.cryptoanalitics.domain.usecases.CheckCoinExistUseCase
import com.student.cryptoanalitics.domain.usecases.GetCryptoCurrenciesHTMLUseCase
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.launch

class PublicCoinsViewModel(
    private val getCryptoCurrenciesHTMLUseCase: GetCryptoCurrenciesHTMLUseCase,
    private val checkCoinExistUseCase: CheckCoinExistUseCase
): ViewModel() {

    private val _coins = MutableStateFlow<CryptoCurrenciesModel?>(CryptoCurrenciesModel(emptyList()))
    val coins: StateFlow<CryptoCurrenciesModel?> = _coins

    private val _isLoading = MutableStateFlow(false)
    val isLoading: StateFlow<Boolean> = _isLoading

    private var currentPage = 1

    init {
        if (currentPage == 1) {
            loadCoins()
        }
    }

    fun loadCoins() {
        if (_isLoading.value) return
        _isLoading.value = true

        viewModelScope.launch(Dispatchers.IO) {
            getCryptoCurrenciesHTMLUseCase.getCoinsList(currentPage).collect{ result ->
```

```
            result?.let {
                if (it.coinList.isNotEmpty()) {
                    _coins.value = CryptoCurrenciesModel(coinList =
(_coins.value?.coinList.orEmpty() + it.coinList))
                    currentPage++
                }
            }

            _isLoading.value = false
        }
    }

}

fun coinExist(coinName: String, callback: (Boolean) -> Unit) {
    viewModelScope.launch {
        callback(checkCoinExistUseCase.checkCoinExist(coinName))
    }
}

}
```

```
package com.student.cryptoanalitics.presentation

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.google.android.material.button.MaterialButton
import com.google.android.material.snackbar.Snackbar
import com.student.cryptoanalitics.App.Companion.mylog
import com.student.cryptoanalitics.R
import com.student.cryptoanalitics.databinding.FragmentAddCryptoBinding
import com.student.cryptoanalitics.domain.models.currencies.CryptoCurrencyModel
import
com.student.cryptoanalitics.presentation.adapters.public_coins.CryptoCurrencyPublicAdapter
import com.student.cryptoanalitics.presentation.adapters.public_coins.CryptoCurrencyPublicClick
import com.student.cryptoanalitics.presentation.vm.InsertCoinsViewModel
import com.student.cryptoanalitics.presentation.vm.LoadCryptoCoinsViewModel
import com.student.cryptoanalitics.presentation.vm.PublicCoinsViewModel
import kotlinx.coroutines.launch
import org.koin.androidx.viewmodel.ext.android.activityViewModel
import org.koin.androidx.viewmodel.ext.android.viewModel

class AddCryptoFragment : Fragment(), CryptoCurrencyPublicClick {
```

```
    private lateinit var binding: FragmentAddCryptoBinding

    private val publicCoinsViewModel: PublicCoinsViewModel by viewModel()
    private val insertCoinsViewModel: InsertCoinsViewModel by viewModel()
    private val loadCryptoCoinsViewModel: LoadCryptoCoinsViewModel by activityViewModel()

    private val cryptoAdapter by lazy {
        CryptoCurrencyPublicAdapter(this)
    }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentAddCryptoBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.recyclerCrypto.adapter = cryptoAdapter

        binding.recyclerCrypto.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
                super.onScrolled(recyclerView, dx, dy)
                val layoutManager = recyclerView.layoutManager as LinearLayoutManager
                if (layoutManager.findLastCompletelyVisibleItemPosition() == cryptoAdapter.itemCount
- 1) {
                    publicCoinsViewModel.loadCoins()
                }
            }
        })

        viewLifecycleOwner.lifecycleScope.launch {
            publicCoinsViewModel.coins.collect { newsList ->
                cryptoAdapter.submitList(newsList?.coinList)
            }
        }

        viewLifecycleOwner.lifecycleScope.launch {
            publicCoinsViewModel.isLoading.collect { isLoading ->
                binding.progressBar.visibility = if (isLoading) View.VISIBLE else View.GONE
            }
        }

        binding.back.setOnClickListener {
            findNavController().navigate(R.id.action_addCryptoFragment_to_mainFragment)
        }
    }
```

```
    override fun addCrypto(button: MaterialButton, cryptoCurrencyModel: CryptoCurrencyModel) {
        publicCoinsViewModel.coinExist(cryptoCurrencyModel.coinName) {
            button.visibility = if (it) View.INVISIBLE else View.VISIBLE
        }

        button.setOnClickListener {
            insertCoinsViewModel.addNewCoin(cryptoCurrencyModel) {
                if (it) {
                    button.visibility = View.INVISIBLE

                    // update main coins list UI
                    loadCryptoCoinsViewModel.loadCoinsData(isRefresh = true, immediate = true)
                } else {
                    Snackbar.make(
                        binding.root,
                        "Крипто-токен не додано, виникла помилка",
                        Snackbar.LENGTH_LONG
                    ).show()
                }
            }
        }
    }
}


package com.student.cryptoanalitics.presentation

import android.os.Build
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import com.squareup.picasso.Picasso
import com.student.cryptoanalitics.R
import com.student.cryptoanalitics.databinding.FragmentDetailCryptoBinding
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.presentation.vm.DeleteCoinsViewModel
import com.student.cryptoanalitics.presentation.vm.GeminiViewModel
import com.student.cryptoanalitics.presentation.vm.LoadCryptoCoinsViewModel
import kotlinx.coroutines.launch
import org.koin.androidx.viewmodel.ext.android.activityViewModel
import org.koin.androidx.viewmodel.ext.android.viewModel

class DetailCryptoFragment : Fragment() {

    private lateinit var binding: FragmentDetailCryptoBinding
```

```
    private val deleteCoinsViewModel: DeleteCoinsViewModel by viewModel()
    private val geminiViewModel: GeminiViewModel by viewModel()
    private val loadCryptoCoinsViewModel: LoadCryptoCoinsViewModel by activityViewModel()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentDetailCryptoBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        val args: CryptoCoinModel? = if (Build.VERSION.SDK_INT >=
Build.VERSION_CODES.TIRAMISU) {
            arguments?.getParcelable("coin", CryptoCoinModel::class.java)
        } else {
            @Suppress("DEPRECATION")
            arguments?.getParcelable("coin")
        }

        args?.let { coin ->
            Picasso.get().load(coin.coinImg).into(binding.coinIcon)

            binding.coinName.text = coin.coinName
            binding.marketPrice.text = coin.marketPrice
            binding.priceChange.text = coin.volMktCap

            binding.circulatingSupplyValue.text =
                resources.getString(R.string.circulating_supply, coin.circulatingSupply)
            binding.marketCapValue.text = resources.getString(R.string.market_cap, coin.marketCap)
            binding.volume24hValue.text = resources.getString(R.string.volume_24h, coin.volume24h)

            binding.delete.setOnClickListener {
                deleteCoinsViewModel.deleteCoin(coin.coinName)

                // update main coins list UI after deleted coin
                loadCryptoCoinsViewModel.loadCoinsData(isRefresh = true, immediate = true)
                findNavController().navigate(R.id.action_detailCryptoFragment_to_mainFragment)
            }

            lifecycleScope.launch {
                geminiViewModel.analiseCoinGemini.collect {
                    it?.let {
                        binding.analiseText.text = it
                        binding.progressBar.visibility = View.INVISIBLE
                        binding.analiseText.visibility = View.VISIBLE
                    }
```

```
            }
        }

        geminiViewModel.requestAnaliseCoin(coin)

    }



    binding.back.setOnClickListener {
        findNavController().navigate(R.id.action_detailCryptoFragment_to_mainFragment)
    }

  }
}


package com.student.cryptoanalitics.presentation

import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.core.os.bundleOf
import androidx.fragment.app.Fragment
import androidx.lifecycle.lifecycleScope
import androidx.navigation.fragment.findNavController
import androidx.recyclerview.widget.LinearLayoutManager
import androidx.recyclerview.widget.RecyclerView
import com.student.cryptoanalitics.R
import com.student.cryptoanalitics.databinding.FragmentMainBinding
import com.student.cryptoanalitics.domain.models.CryptoCoinModel
import com.student.cryptoanalitics.presentation.adapters.my_coins.CryptoCoinsPrivateAdapter
import com.student.cryptoanalitics.presentation.adapters.my_coins.CryptoCoinsPrivateClick
import com.student.cryptoanalitics.presentation.vm.LoadCryptoCoinsViewModel
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.launch
import org.koin.androidx.viewmodel.ext.android.activityViewModel

class MainFragment : Fragment(), CryptoCoinsPrivateClick {

    private lateinit var binding: FragmentMainBinding

    private val loadCryptoCoinsViewModel: LoadCryptoCoinsViewModel by activityViewModel()

    private val cryptoAdapter by lazy {
        CryptoCoinsPrivateAdapter(this)
    }

    override fun onCreateView(
        inflater: LayoutInflater,
```

```
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentMainBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.recyclerCrypto.adapter = cryptoAdapter

        viewLifecycleOwner.lifecycleScope.launch {
            loadCryptoCoinsViewModel.isLoading.collect { isLoading ->
                binding.progressBar.visibility =
                    if (isLoading) View.VISIBLE else View.GONE
            }
        }

        viewLifecycleOwner.lifecycleScope.launch {
            loadCryptoCoinsViewModel.isRefreshing.collect { isRefreshing ->
                binding.swipeRefresh.isRefreshing = isRefreshing
            }
        }

        lifecycleScope.launch {
            loadCryptoCoinsViewModel.coinsFlow.collectLatest {
                cryptoAdapter.submitList(it)
            }
        }

        binding.swipeRefresh.setOnRefreshListener {
            if(!loadCryptoCoinsViewModel.isLoading.value) {
                loadCryptoCoinsViewModel.loadCoinsData(true)
            }else {
                binding.swipeRefresh.isRefreshing = false
            }
        }

        binding.recyclerCrypto.addOnScrollListener(object : RecyclerView.OnScrollListener() {
            override fun onScrolled(recyclerView: RecyclerView, dx: Int, dy: Int) {
                super.onScrolled(recyclerView, dx, dy)
                val layoutManager = recyclerView.layoutManager as LinearLayoutManager
                if (layoutManager.findLastCompletelyVisibleItemPosition() == cryptoAdapter.itemCount
- 1) {
                    loadCryptoCoinsViewModel.loadCoinsData()
                }
            }
        })
```

```
        binding.addCrypto.setOnClickListener {
            findNavController().navigate(R.id.action_mainFragment_to_addCryptoFragment)
        }


    }

    override fun clickCoin(cryptoCoinModel: CryptoCoinModel) {
        val navController = findNavController()
        if (navController.currentDestination?.id == R.id.mainFragment) {
            navController.navigate(
                R.id.action_mainFragment_to_detailCryptoFragment,
                bundleOf("coin" to cryptoCoinModel)
            )
        }
    }
}

package com.student.cryptoanalitics


import android.app.Application
import android.util.Log
import androidx.appcompat.app.AppCompatDelegate
import androidx.room.Room
import androidx.room.RoomDatabase
import androidx.sqlite.db.SupportSQLiteDatabase
import com.google.ai.client.generativeai.GenerativeModel
import com.google.ai.client.generativeai.type.BlockThreshold
import com.google.ai.client.generativeai.type.HarmCategory
import com.google.ai.client.generativeai.type.SafetySetting
import com.google.ai.client.generativeai.type.generationConfig
import com.student.cryptoanalitics.data.CryptoDBRepositoryImpl
import com.student.cryptoanalitics.data.CryptoHtmlHtmlRepositoryImpl
import com.student.cryptoanalitics.data.api.CryptoAPI
import com.student.cryptoanalitics.data.api.CryptoAPI.Companion.BASE_URL_CRYPTO
import com.student.cryptoanalitics.data.database.CryptoDataBase
import
com.student.cryptoanalitics.data.database.CryptoDataBase.Companion.DATABASE_NAME
import com.student.cryptoanalitics.data.database.DAO
import com.student.cryptoanalitics.domain.repositories.CryptoDBRepository
import com.student.cryptoanalitics.domain.repositories.CryptoHtmlRepository
import com.student.cryptoanalitics.domain.usecases.AddCoinUseCase
import com.student.cryptoanalitics.domain.usecases.CheckCoinExistUseCase
import com.student.cryptoanalitics.domain.usecases.DeleteCoinUseCase
import com.student.cryptoanalitics.domain.usecases.GeminiUseCase
import com.student.cryptoanalitics.domain.usecases.GetCoinHTMLUseCase
import com.student.cryptoanalitics.domain.usecases.GetCryptoCurrenciesHTMLUseCase
import com.student.cryptoanalitics.domain.usecases.GetPagedCoinsUseCase
import com.student.cryptoanalitics.domain.usecases.UpdateCoinsUseCase
import com.student.cryptoanalitics.presentation.vm.DeleteCoinsViewModel
```

```
import com.student.cryptoanalitics.presentation.vm.GeminiViewModel
import com.student.cryptoanalitics.presentation.vm.InsertCoinsViewModel
import com.student.cryptoanalitics.presentation.vm.LoadCryptoCoinsViewModel
import com.student.cryptoanalitics.presentation.vm.PublicCoinsViewModel
import org.koin.android.ext.koin.androidContext
import org.koin.android.ext.koin.androidLogger
import org.koin.core.context.startKoin
import org.koin.core.module.dsl.viewModel
import org.koin.dsl.module
import retrofit2.Retrofit
import retrofit2.converter.scalars.ScalarsConverterFactory


class App : Application() {

    override fun onCreate() {
        super.onCreate()
        AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO)

        startKoin {
            androidLogger()
            androidContext(this@App)
            modules(
                listOf(
                    repositoriesModule,
                    useCasesModule,
                    viewModelModule,
                    networkModule,
                    databaseModule,
                    artificialIntelligenceModule
                )
            )
        }
    }

    private val repositoriesModule = module {
        single<CryptoHtmlRepository> { CryptoHtmlHtmlRepositoryImpl(cryptoAPI = get()) }
        single<CryptoDBRepository> { CryptoDBRepositoryImpl(db = get()) }
    }

    private val useCasesModule = module {
        factory {
            GetCoinHTMLUseCase(get<CryptoHtmlRepository>())
        }
        factory {
            GetCryptoCurrenciesHTMLUseCase(get<CryptoHtmlRepository>())
        }
        factory {
            CheckCoinExistUseCase(get<CryptoDBRepository>())
        }
        factory {
```

```
        AddCoinUseCase(get<CryptoDBRepository>())
    }
    factory {
        GetPagedCoinsUseCase(get<CryptoDBRepository>())
    }
    factory {
        UpdateCoinsUseCase(get<CryptoDBRepository>())
    }
    factory {
        DeleteCoinUseCase(get<CryptoDBRepository>())
    }
    factory {
        GeminiUseCase(get<GenerativeModel>())
    }
}

private val viewModelModule = module {
    viewModel {
        PublicCoinsViewModel(
            getCryptoCurrenciesHTMLUseCase = get(),
            checkCoinExistUseCase = get()
        )
    }

    viewModel {
        InsertCoinsViewModel(
            addCoinUseCase = get()
        )
    }
    viewModel {
        LoadCryptoCoinsViewModel(
            getPagedCoinsUseCase = get(),
            getCoinHTMLUseCase = get(),
            updateCoinsUseCase = get()
        )
    }
    viewModel {
        DeleteCoinsViewModel(
            deleteCoinUseCase = get()
        )
    }
    viewModel {
        GeminiViewModel(
            geminiUseCase = get()
        )
    }
}

private val networkModule = module {
    single<Retrofit> {
        Retrofit.Builder()
```

```
            .baseUrl(BASE_URL_CRYPTO)
            .addConverterFactory(ScalarsConverterFactory.create())
            .build()
        }
        single<CryptoAPI> {
            get<Retrofit>().create(CryptoAPI::class.java)
        }
    }

    private val databaseModule = module {
        single<CryptoDataBase> {
            Room.databaseBuilder(
                get(),
                CryptoDataBase::class.java,
                DATABASE_NAME
            ).addCallback(object : RoomDatabase.Callback() {
                override fun onCreate(db: SupportSQLiteDatabase) {
                    super.onCreate(db)
                    db.execSQL("INSERT INTO coins (coinName, coinPrice) VALUES ('Bitcoin',
'100000')")
                    db.execSQL("INSERT INTO coins (coinName, coinPrice) VALUES ('Ethereum',
'4000')")
                }
            }).build()
        }

        single<DAO> {
            get<CryptoDataBase>().getDAO()
        }
    }

    private val artificialIntelligenceModule = module {
        single<GenerativeModel> {
            GenerativeModel(
                modelName = "gemini-1.5-flash-001",
                apiKey = BuildConfig.GEMINI_API_KEY,
                generationConfig = generationConfig {
                    temperature = 0.15f
                    topK = 32
                    topP = 1f
                    maxOutputTokens = 4096
                },
                safetySettings = listOf(
                    SafetySetting(HarmCategory.HARASSMENT,
BlockThreshold.MEDIUM_AND_ABOVE),
                    SafetySetting(HarmCategory.HATE_SPEECH,
BlockThreshold.MEDIUM_AND_ABOVE),
                    SafetySetting(HarmCategory.SEXUALLY_EXPLICIT,
BlockThreshold.MEDIUM_AND_ABOVE),
                    SafetySetting(HarmCategory.DANGEROUS_CONTENT,
BlockThreshold.MEDIUM_AND_ABOVE),
```

```
            )
        )
    }
}

companion object {
    private const val MAIN_TAG_LOG = "mycryptoapp"
    fun mylog(args: Any) {
        Log.i(MAIN_TAG_LOG, "$args")
    }
}
}

package com.student.cryptoanalitics

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.app.AppCompatDelegate
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import com.student.cryptoanalitics.App.Companion.mylog

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContentView(R.layout.activity_main)
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.nav_host_fragment)) { v,
insets ->
            val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
            insets
        }
    }
}
```