СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВОЛОДИМИРА ДАЛЯ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ПРОГРАМУВАННЯ

Пояснювальна записка

до бакалаврської дипломної роботи

(освітньо-кваліфікаційний рівень)

на тему: Розробка мобільного додатку для відстеження інвестиційного портфелю для криптовалютних активів

Виконав: студент 4 курсу, групи ІПЗ -19д

спеціальності 121 Інженерія прогрмного забезпечення
(шифр і назва спеціальності)

Желнов Микита Олегович
(прізвище та ініціали)

Керівник доц., к.т.н. Іванов В.Г
(прізвище та ініціали)

Рецензент доц., д.т.н. Лифар В. О.
(прізвище та ініціали)

Київ - 2023

ЗМІСТ

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи бакалавру складається з: 43 сторінок основного тексту, 14 рисунків, 1 додатку, 14 використаних джерел.

Метою даної бакалаврської дипломної роботи є розробка мобільного додатку для Android, який допоможе інвесторам відстежувати свої криптовалютні активи (криптовалютне портфоліо) та аналізувати ситуацію на ринку, прорахочуючи прибуток та можливі ризики.

Об’єктом дослідження є розробка мобільного застосунку за допомогою мов програмування Java та TypeScript для аналізу криптовалютного портфоліо.

Головні завдання роботи:

1. Огляд предметної області.
2. Провести аналіз існуючих рішень.
3. Розробити архітектуру проєкту.
4. Реалізувати архітектуру проєкту.
5. Реалізувати інтерфейс користувача додатку.
6. Провести перевірку створеного інструменту.

Методи дослідження – аналіз предметної області буде виконаний з акцентом на ієрархічне представлення об'єктів та сформованими логічними зв'язки між процесами.

Ключові слова: ПРОЄКТ, ПОРТФОЛІО, КРИПТОВАЛЮТА, JAVA, KOTLIN, ANDROID STUDIO, МОБІЛЬНИЙ ДОДАТОК, ШИФРУВАННЯ ДАНИХ

ВСТУП

У сучасному світі інвестиції у криптовалютні активи стають все більш популярними. Швидкий розвиток технологій, зростання обсягів цифрових активів та нові можливості фінансового вкладання привертають увагу широкого кола інвесторів. Однак, зростання популярності криптовалют також супроводжується збільшенням складності управління та моніторингу інвестиційного портфоліо.

В рамках цього контексту виникає необхідність у розробці мобільного додатку, який допомагатиме інвесторам ефективно відстежувати та аналізувати свої криптовалютні активи. Метою даної дипломної роботи є розробка такого мобільного додатку для відстеження інвестиційного портфелю з фокусом на криптовалютні активи.

Основною метою додатку є забезпечення інвесторів інструментом, який дозволяє легко відстежувати та аналізувати їхні інвестиційні позиції в криптовалютних активах. Додаток буде надавати користувачам зручні та цілеспрямовані функції для моніторингу вартості активів, отримання статистики, аналізу та визначення потенційних ризиків та доходності. Крім того, він буде забезпечувати зручний інтерфейс для додавання та оновлення інвестиційних позицій, а також отримання актуальної інформації про криптовалютні ринки.

У роботі буде використано сучасні технології та інструменти для розробки мобільних додатків, зокрема, мову програмування Java та Kotlin, популярну платформу для розробки Android, а також безкоштовний API та сервіс для отримання актуальних даних про криптовалютні ринки.

Очікується, що розроблений мобільний додаток буде корисним інструментом для інвесторів, які мають інтерес до криптовалютних активів. Він дозволить їм ефективно керувати своїми інвестиціями, отримувати актуальну інформацію та здійснювати обґрунтовані рішення щодо їхнього портфоліо.

У дипломній роботі будуть розглянуті основні аспекти розробки

мобільного додатку для відстеження інвестиційного портфоліо криптовалютних активів, включаючи аналіз вимог, проектування архітектури, реалізацію функціональності та тестування додатку.

Основними завданнями дипломної роботи є:

1. Аналіз вимог і визначення функціональності мобільного додатку.

2. Проектування архітектури додатку та вибір необхідних технологій.

3. Реалізація функціональності додатку згідно з встановленими вимогами.

4. Тестування та валідація додатку для забезпечення його коректної роботи та надійності.

5. Оцінка ефективності та використання додатку через проведення експериментів та оцінку результатів.

Очікується, що результати дипломної роботи дадуть корисні висновки та рекомендації щодо розробки та використання мобільних додатків для відстеження інвестиційного портфоліо криптовалютних активів.

РОЗДІЛ 1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

Мобільні додатки для криптовалютного портфоліо є невід'ємною частиною сучасного інвестиційного середовища. Вони надають користувачам зручний і простий спосіб управління криптовалютними активами на мобільних пристроях, таких як смартфони та планшети. Ці додатки дозволяють інвесторам отримувати у реальному часі інформацію про стан їхнього портфоліо, відстежувати ціни криптовалют та отримувати нагадування про важливі події на ринку.

Основні функції мобільного додатку для криптовалютного портфелю:

1. Управління портфелем: Додаток надає можливість користувачам додавати, видаляти та редагувати свої криптовалютні активи у портфелі. Користувачі можуть відстежувати кількість монет, вартість та відсоток зміни для кожного активу.

2. Отримання інформації в реальному часі: Додаток повинен мати можливість отримувати оновлену інформацію про ціни криптовалют з надійних джерел даних. Це дозволяє користувачам відстежувати останні зміни в цінах та приймати обґрунтовані рішення щодо свого портфоліо.

3. Графіки та аналіз: Додаток повинен надати графіки цін криптовалют та індикатори, що допомагають користувачам аналізувати рух ринку. Такі інструменти, як лінійні графіки, свічкові графіки та індикатори технічного аналізу, допомагають інвесторам зробити інформовані рішення.

4. Безпека та приватність: Враховуючи важливість безпеки криптовалютних активів, додаток повинен мати надійний механізм захисту, такий як, шифрування даних.

5. Торгівельні можливості: Деякі мобільні додатки для криптовалютного портфелю надають користувачам можливість здійснювати торгівельні операції безпосередньо з додатку. Це дозволяє інвесторам

купувати та продавати криптовалюти без необхідності використовувати окрему торговельну платформу.

Мобільні додатки для криптовалютного портфоліо стали незамінним інструментом для інвесторів, які бажають мати зручний та швидкий доступ до своїх криптовалютних активів. Ці додатки надають широкий спектр функцій, що дозволяють відстежувати стан портфеля, аналізувати ринок та здійснювати торгівельні операції.

РОЗДІЛ 2 ІСНУЮЧІ РІШЕННЯ

У цьому розділі проведемо огляд різних наявних аналогів мобільних додатків, призначених для криптовалютного інвестиційного портфоліо. При дослідженні цих аналогів звернемо увагу на їх функціональність, зручність використання, наявність додаткових функцій та можливостей, які можуть бути корисними для інвесторів.

Аналоги мобільних додатків для криптовалютних інвестиційних портфоліо повинні включати функції, такі як і в кваліфікаційній роботі:

1. Відстеження портфоліо: Додатки, які дозволяють користувачам відстежувати свої криптовалютні активи, переглядати їх поточну вартість, зміни цін та баланси.

2. Графіки та аналітика: Деякі додатки надають графіки, діаграми та аналітику, що допомагають в оцінці ринкової динаміки, трендів та ризиків.

3. Сповіщення: Додатки можуть надсилати сповіщення користувачам про значні зміни в цінах або інших важливих подіях, що стосуються їх портфеля.

4. Інтеграція з біржами: Деякі додатки можуть підтримувати підключення до різних криптовалютних бірж для отримання актуальних даних про торгові пари та можливість здійснювати торгівельні операції.

5. Безпека: Важливим аспектом є захист персональних даних, які зберігають криптовалютні активи користувача.

Досліджуючи існуючі аналоги, є можливість отримати об'єктивну оцінку їх переваг та недоліків, що допоможе покращити власний застосунок для криптовалютних інвестицій.

2.1. Blockfolio

Blockfolio [7] є одним з найпопулярніших мобільних додатків для криптовалютного портфоліо. Це потужний інструмент, який дозволяє користувачам відстежувати, управляти та аналізувати їх інвестиційними криптовалютними портфоліо.

Основна функціональність Blockfolio включає наступні можливості:

1. Відстеження портфоліо: Blockfolio дозволяє користувачам додавати свої криптовалютні активи та відстежувати їх поточну вартість та зміни в реальному часі. У ньому можна налаштувати сповіщення про цінові рухи та отримувати оновлення про інвестиції.

2. Користувацькі налаштування: Blockfolio дозволяє налаштувати свої власні списки обраного, створювати власні категорії активів і настроювати вигляд портфоліо під свої потреби. Також є можливість персоналізувати сповіщення та події, що пов'язані зі станом вашого портфоліо.

3. Аналітика та графіки: Blockfolio надає користувачам детальну аналітику та графіки, які допомагають в оцінці ринкової динаміки та трендів. Також є можливість переглядати історичні дані, графіки цін, обсягів торгів та інші важливі показники.

4. Новини та оновлення: Додаток постійно оновлюється з новинами та оглядами з криптовалютного ринку. Додаток надсилає останні новини, аналітику та коментарі експертів, що допомагає інвесторам бути в курсі останніх подій.

5. Підтримка різних бірж: Blockfolio [7] інтегрується з численними криптовалютними біржами, що дозволяє користувачам отримувати актуальну інформацію про свої торгові операції, баланси та історію торгів.

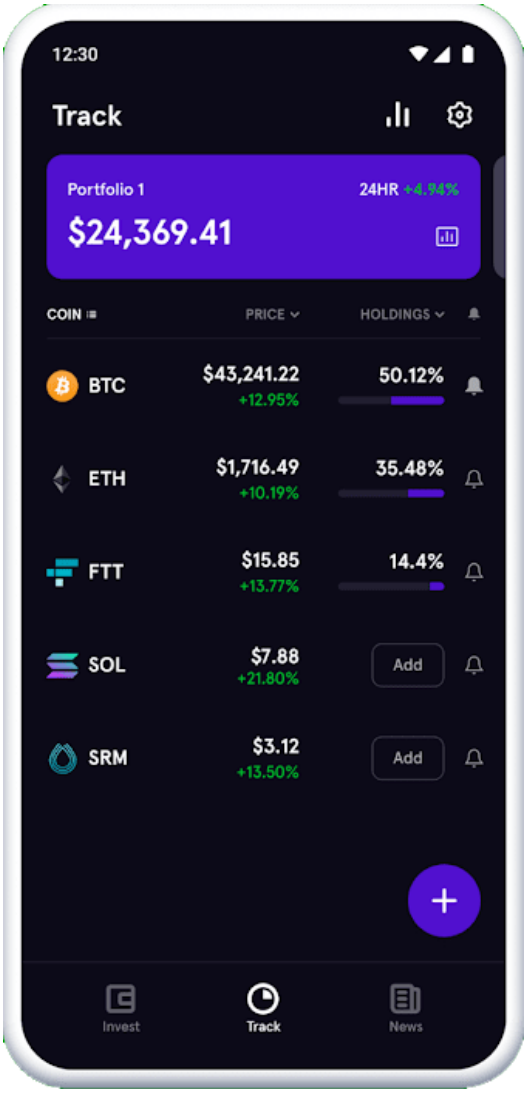

Рис. 2.1 – Інтерфейс Blockfolio

2.2. Coinbase

Coinbase [8] – це одна з найбільших і найпопулярніших криптовалютних платформ у світі, яка надає послуги з купівлі, продажу, зберігання та обміну криптовалют. Цей мобільний додаток дозволяє користувачам легко управляти своїм криптовалютним портфоліо та здійснювати операції на ринку криптовалют.

Основні особливості Coinbase включають:

1. Купівля та продаж криптовалют: Додаток Coinbase дозволяє користувачам придбавати і продавати різні криптовалюти, такі як Bitcoin

(BTC), Ethereum (ETH), Litecoin (LTC) та багато інших. Він пропонує простий та зручний інтерфейс для здійснення торгівлі на ринку.

2. Зберігання активів: Coinbase забезпечує безпечне зберігання криптовалютних активів своїх користувачів у віртуальних гаманцях. Вони використовують різні методи захисту, включаючи холодне зберігання (cold storage) для максимальної безпеки.

3. Підтримка багатьох криптовалют: Coinbase підтримує широкий спектр криптовалютних активів, що дозволяє користувачам диверсифікувати своє портфоліо та обмінюватися різними криптовалютами безпосередньо в додатку.

4. Графіки та аналітика: Додаток Coinbase надає доступ до детальних графіків цін та статистики ринку криптовалют. Користувачі можуть аналізувати ринкові тренди та отримувати актуальну інформацію для прийняття розумних інвестиційних рішень.

5. Інтеграція з банківськими рахунками: Coinbase [8] дозволяє зв'язати свій банківський рахунок з додатком, що дозволяє легко поповнювати свій рахунок та здійснювати операції купівлі-продажу криптовалют.

6. Безпека та регулювання: Coinbase дотримує високих стандартів безпеки та дотримання правил та регуляцій криптовалютного ринку. Вони працюють у відповідності з відповідними органами регулювання та використовують різні заходи для захисту активів та персональних даних користувачів.

Додаток Coinbase є популярним вибором серед багатьох інвесторів криптовалют, оскільки він пропонує зручний інтерфейс, надійну безпеку та широкий спектр функціональних можливостей для управління криптовалютними активами.

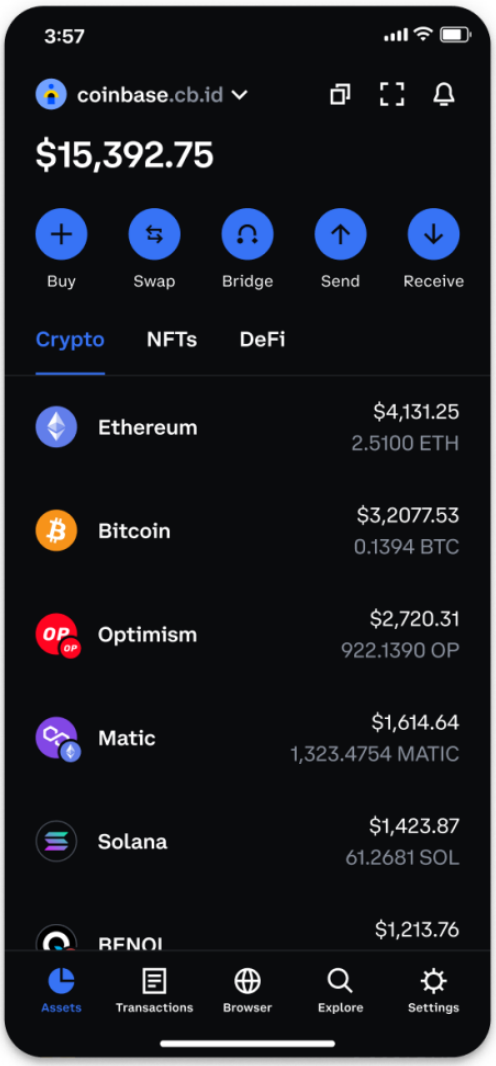

Рис. 2.2 – Інтерфейс Coinbase

2.3. Binance

Binance [9] – це одна з найбільших та найпопулярніших криптовалютних бірж у світі, яка пропонує широкий спектр послуг для торгівлі криптовалютами та управління криптовалютним портфелем. Binance привертає увагу інвесторів своєю високою ліквідністю, широким вибором криптовалютних активів та високоякісними торговельними інструментами.

Основні особливості Binance:

1. Торгівля криптовалютами: Binance надає можливість користувачам купувати, продавати та торгувати різними криптовалютами. Платформа

підтримує широкий вибір криптовалютних пар і запроваджує різні типи замовлень, такі як ринкові, лімітні та стоп-лімітні замовлення, що дозволяють інвесторам ефективно управляти своїми торговими операціями.

2. Binance Coin (BNB) [9]: Binance випустила власну криптовалюту під назвою Binance Coin (BNB). BNB може бути використаний для оплати комісій на біржі та отримання знижок. Також, BNB може бути використаний для участі в ініціативах платформи, таких як запуск нових проектів через Binance Launchpad.

3. Binance Smart Chain (BSC): Binance має власну блокчейн-платформу під назвою Binance Smart Chain. Вона надає можливість запуску та використання децентралізованих додатків (DApps) та смарт-контрактів. BSC також підтримує взаємодію з іншими блокчейнами, що розширює можливості користувачів для участі в екосистемі DeFi (децентралізованих фінансів).

4. Безпека та регулювання: Binance приділяє велику увагу безпеці своїх користувачів та дотриманню вимог регуляторних органів. Вони використовують різні заходи безпеки, включаючи двофакторну аутентифікацію, шифрування даних та системи моніторингу.

5. Мобільний додаток: Binance [9] надає мобільний додаток для iOS та Android, що дозволяє користувачам зручно торгувати криптовалютами та отримувати доступ до свого портфеля з будь-якого місця та в будь-який час.

Binance є популярним вибором для торгівлі криптовалютами та управління криптовалютним портфелем завдяки своїй надійності, розширеному функціоналу та високому рівню ліквідності.

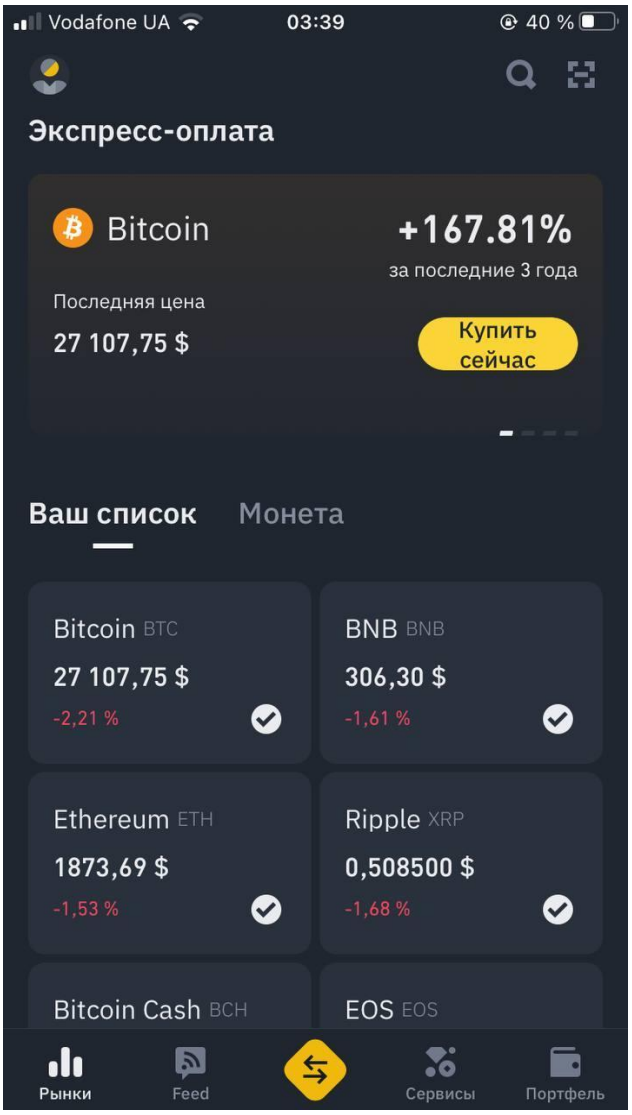

Рис. 2.3 – Інтерфейс Binance

2.4. Kraken

Kraken [10] – це одна з провідних криптовалютних бірж у світі, яка надає можливість торгувати різними криптовалютами та управляти криптовалютним портфоліо. Заснована у 2011 році, Kraken пропонує широкий вибір торгових пар та різні інструменти для зручної торгівлі інвесторам.

Основні особливості Kraken:

1. Торгівля криптовалютами: Kraken дозволяє користувачам торгувати різними криптовалютами з використанням різних торгових пар, включаючи

криптовалюту до фіатної валюти та криптовалюту до криптовалюти. Платформа підтримує різні типи замовлень, включаючи ринкові, лімітні та стоп-лімітні замовлення, що дозволяє інвесторам здійснювати точні торгові операції згідно з їхніми стратегіями.

2. Зручний інтерфейс та аналітика: Kraken надає зручний та інтуїтивно зрозумілий інтерфейс, який допомагає користувачам легко орієнтуватися по платформі та здійснювати торгові операції. Крім того, біржа надає різні аналітичні інструменти та графіки, які допомагають інвесторам аналізувати ринок та приймати обґрунтовані рішення.

3. Безпека та надійність: Kraken приділяє велику увагу безпеці своїх користувачів та забезпечує заходи безпеки, такі як двофакторна аутентифікація, шифрування даних та холодне сховище для зберігання криптовалютних активів. Крім того, біржа підлягає регулюванню та дотримується стандартів безпеки, що забезпечує додаткову надійність та довіру інвесторів.

4. Підтримка клієнтів: Kraken надає широкий спектр підтримки клієнтів, включаючи онлайн-чат, електронну пошту та телефонну підтримку. Команда підтримки готова допомогти користувачам у вирішенні будь-яких питань чи проблем, пов'язаних з платформою.

5. Мобільний додаток: Kraken [10] пропонує мобільний додаток для iOS та Android, що дозволяє інвесторам зручно торгувати та відстежувати свої інвестиції з мобільного пристрою. Це дозволяє користувачам мати доступ до свого портфоліо та ринку криптовалют у будь-який час і з будь-якого місця.

Kraken є популярною та надійною криптовалютною біржею, яка надає інвесторам можливість торгувати криптовалютами та управляти криптовалютним портфоліо.

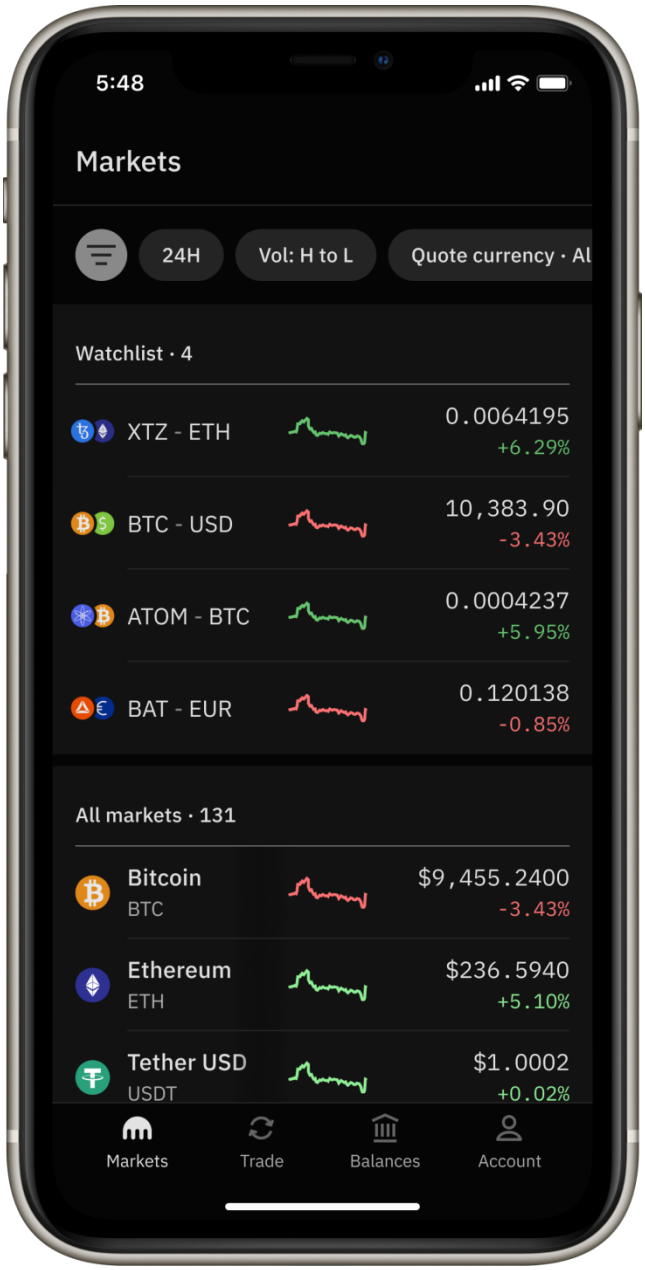

Рис. 2.4 – Інтерфейс Kraken

РОЗДІЛ 3 АРХІТЕКТУРА ЗАСТОСУНКУ

3.1. Трирівнева архітектура

Для розробки проєкту було вибрана трирівнева архітектура [11] застосунку, що дозволяє чітко розділити функціональність на трьох рівнях: презентаційний (інтерфейс), бізнес-логіку та доступ до даних. Ця архітектура забезпечує модульність, розширюваність та легкість управління проєктом.

На першому рівні, презентаційному, знаходяться компоненти, що відповідають за відображення інтерфейсу користувача. Це можуть бути активності, фрагменти або інші елементи, що забезпечують взаємодію з користувачем. На цьому рівні відбувається обробка подій та передача даних до наступного рівня.

Другий рівень, бізнес-логіка, відповідає за обробку логічних операцій та бізнес-правил. Тут розміщені компоненти, що виконують операції над даними, проводять розрахунки, валідацію та прийняття рішень. Цей рівень забезпечує основну функціональність додатку та забезпечує правильну обробку даних.

Третій рівень, доступ до даних, відповідає за збереження та доступ до даних у базі даних або інших джерелах даних. Тут розміщені компоненти, що виконують операції зчитування, запису та маніпуляції даними. Цей рівень забезпечує надійність та ефективність роботи з даними.

Трирівнева архітектура [11] (рис 3.1) дозволяє зменшити залежність між компонентами, покращити перевикористання коду, забезпечити легкість тестування та підтримки проєкту. Кожен рівень може бути розроблений та модифікований незалежно, що спрощує розробку та розширення функціональності проєкту.

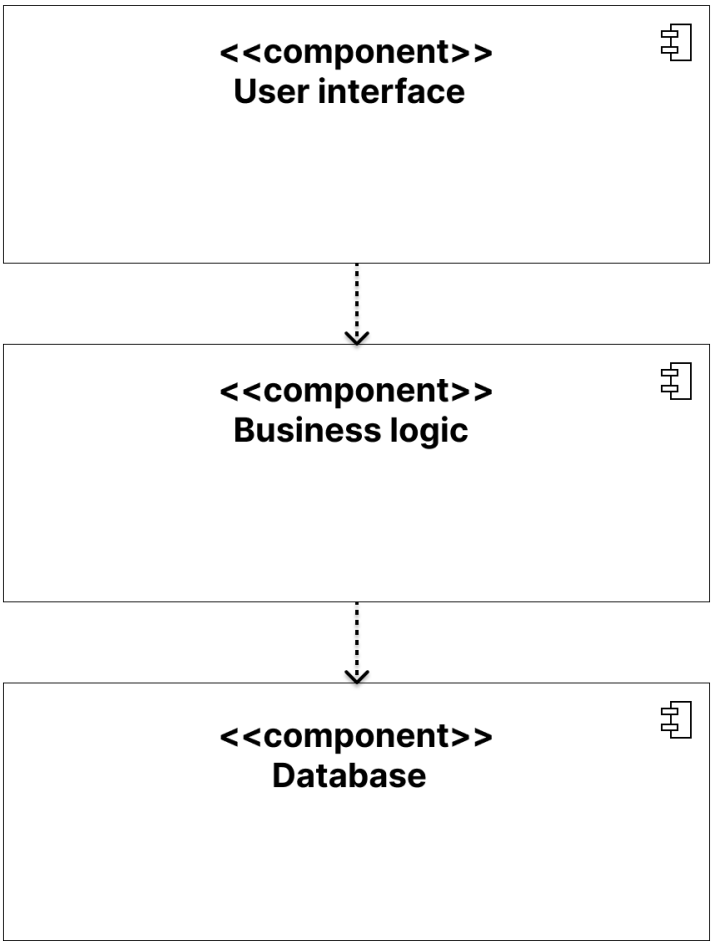

Рис. 3.1 – Трирівнева архітектура

Завдяки трирівневій системі, застосунок буде більш зручним для користувачів та забезпечуватиме правильний підхід до створення криптопортфоліо.

Презентаційний рівень відповідає за інтуїтивно зрозумілий та естетичний інтерфейс, що дозволяє користувачам легко знаходити необхідні функції, переходити між різними розділами та екранами, а також виконувати різноманітні дії, такі як перегляд балансу, виконання торгівельних операцій, отримання статистичних даних та інше.

Бізнес-логічний рівень забезпечує обробку логіки та операцій, пов'язаних з криптовалютним портфоліо, включаючи розрахунки, валідацію та забезпечення безпеки.

Рівень доступу до даних відповідає за збереження, отримання та маніпуляції з даними криптовалютного портфоліо.

Трирівнева архітектура дозволяє розмежувати функціональність та відповідальності між різними компонентами, що забезпечує чітку структуру проєкту та полегшує розробку, тестування та підтримку. Користувачі зможуть швидко зорієнтуватись у функціоналі додатку, завдяки ясному розподілу ролей між рівнями. Крім того, така архітектура сприяє підтримці та розширенню додатку, оскільки зміни в одному рівні не мають безпосереднього впливу на інші рівні.

Застосування трирівневої архітектури [11] до створення криптопортфоліо забезпечує не лише зручність для користувачів, але й правильний підхід до зберігання, обробки та захисту криптовалютних активів. Він дозволяє створити масштабований, надійний та безпечний інструмент для управління криптовалютними інвестиціями, що задовольняє вимоги сучасних користувачів та стандарти безпеки в цій сфері.

3.2. Логіка мобільного застосунку

Розглянемо логіку та рівень даних у контексті трирівневої моделі розробки бізнес-додатків.

Додаток можна умовно розділити на два модулі (рис. 3.2). Перший модуль, який називається portfolio service, відповідає за зберігання та обробку даних про користувачів та їхні портфоліо, другий модуль, який називається marketdata provider, відповідає за завантаження та зберігання даних про котирування активів.

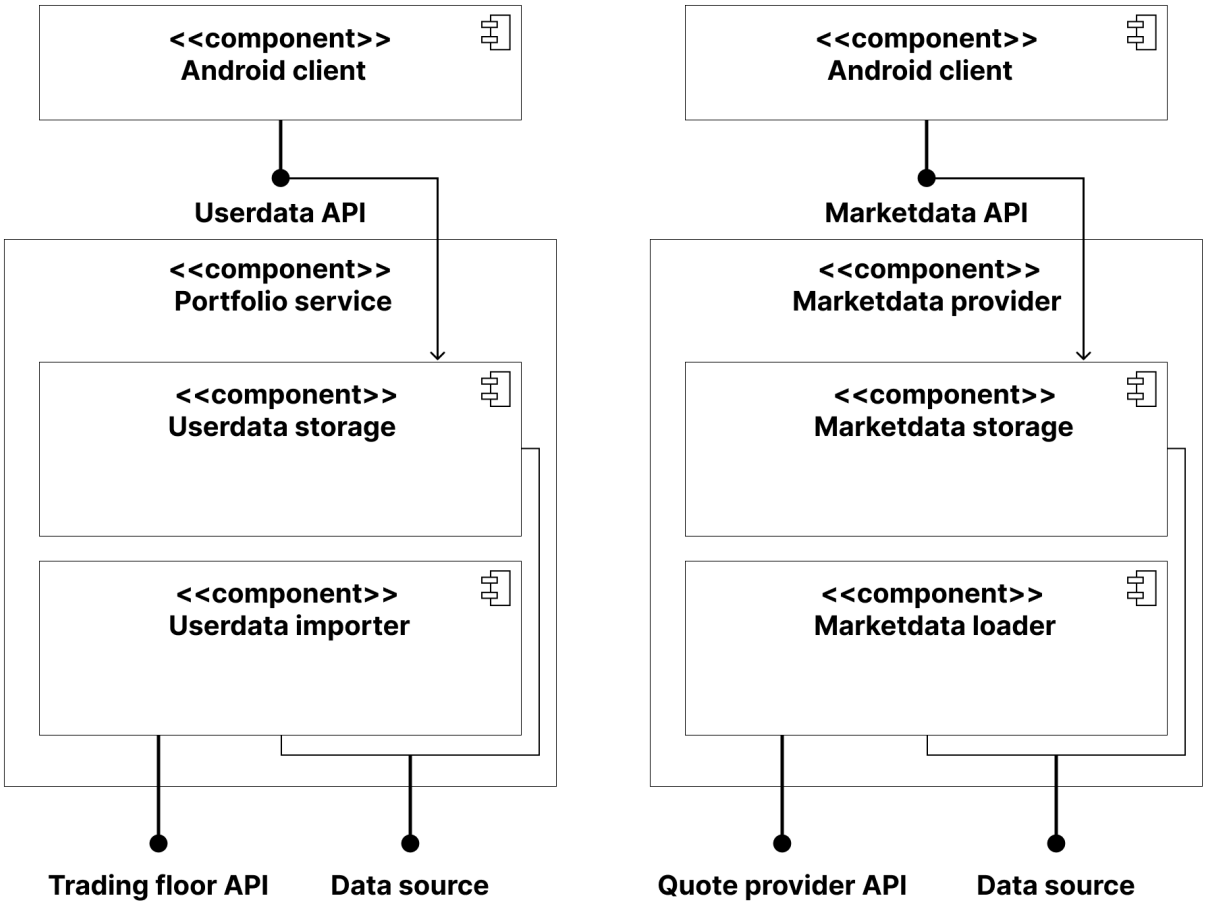

Рис. 3.2 – Модулі додатку

Більш детально розглянемо архітектуру кожного сервісу. Portfolio service має шарувату архітектуру (рис. 3.3), яка є стандартом у подібних додатках. Він містить такі шари:

- **Шар контролерів**.

Використовується для забезпечення зв'язку між компонентом інтерфейсу користувача та компонентом логіки у трьохрівневій моделі розробки програмних додатків. Його основна мета полягає в установленні інтерфейсу програмного забезпечення (API), який забезпечує взаємодію з серверною частиною програми.

У шарі контролерів визначаються методи та функції, які дозволяють обробляти запити користувача та передавати їх до відповідних компонентів бізнес-логіки. Цей шар відповідає за обробку вхідних даних, валідацію, виконання необхідних операцій та повернення відповідей користувачу. Він

встановлює точку входу для взаємодії з програмним додатком, де обробка запитів та передача даних між різними компонентами відбувається за допомогою визначеного API.

Однією з важливих переваг використання шару контролерів є можливість визначити різні контролери для різних компонентів інтерфейсу користувача або різних видів запитів. Це дозволяє забезпечити чітку розділеність відповідальностей та покращити організацію коду. Крім того, шар контролерів дозволяє легко розширювати функціональність програмного додатку шляхом додавання нових контролерів або розширення існуючих.

Використання шару контролерів сприяє забезпеченню гнучкості, модульності та перевикористання коду. Він допомагає впроваджувати принципи чистої архітектури та розділяти логіку рівня інтерфейсу користувача від бізнес-логіки, що спрощує розробку, тестування та підтримку програмного додатку.

- **Шар логіки.**

Використовується для моделювання предметної області. У цьому контексті використовується анемічна модель предметної області, що означає, що всі дії, пов'язані з об'єктами предметної області, виносяться в окремі сервіси. Такий підхід дозволяє спростити кодування, особливо в ситуаціях, коли взаємодія між об'єктами не вимагає складної логіки.

Застосування анемічної моделі предметної області допомагає покращити розподіл відповідальностей та розділити логіку обробки даних від самого об'єкта. Кожен сервіс в шарі бізнес-логіки відповідає за конкретний аспект обробки даних і надає відповідні методи для виконання операцій над об'єктами.

Крім того, використання анемічної моделі полегшує серіалізацію об'єктів. Оскільки всі дії пов'язані з об'єктами винесені в сервіси, об'єкти можуть бути передані безпосередньо між сервісами без необхідності передавати всю логіку разом з об'єктом.

Загалом, використання шару бізнес-логіки дозволяє покращити модульність, зрозумілість та перевикористання коду. Кожен сервіс виконує конкретну функцію, що сприяє зменшенню залежностей між компонентами та полегшує розвиток та супровід програмного додатку.

- **Шар репозиторіїв.**

Він відповідає за забезпечення взаємодії між додатком і базою даних, а також збереження та отримання даних з бази. Цей шар має на меті абстрагувати деталі роботи з базою даних, що дозволяє розробникам працювати з даними на вищому рівні абстракції.

У шарі репозиторіїв визначаються методи для здійснення різноманітних операцій з базою даних, таких як створення, читання, оновлення та видалення записів. Ці методи реалізують бізнес-логіку додатка і виконують запити до бази даних, використовуючи відповідні мовні конструкції та бібліотеки для роботи з базою даних.

Шар репозиторіїв також відповідає за обробку помилок, пов'язаних з доступом до бази даних, і встановлення відповідного рівня абстракції для забезпечення безпеки та цілісності даних. Він дозволяє розробникам працювати з даними зручним способом, не хвилюючись про деталі реалізації бази даних.

Шар репозиторіїв часто використовується разом з іншими шарами архітектури додатка, такими як шар бізнес-логіки та шар інтерфейсу користувача. Він сприяє розподілу обов'язків та дотриманню принципів чистої архітектури, що полегшує тестування, розширення та підтримку додатка.

Загалом, шар репозиторіїв є важливим елементом розробки програмного додатка, оскільки він дозволяє ефективно працювати з базою даних і забезпечує гнучкість та незалежність від конкретної реалізації бази даних. Використання шару репозиторіїв сприяє покращенню якості та швидкості розробки додатків, а також спрощує підтримку та розширення системи в майбутньому.

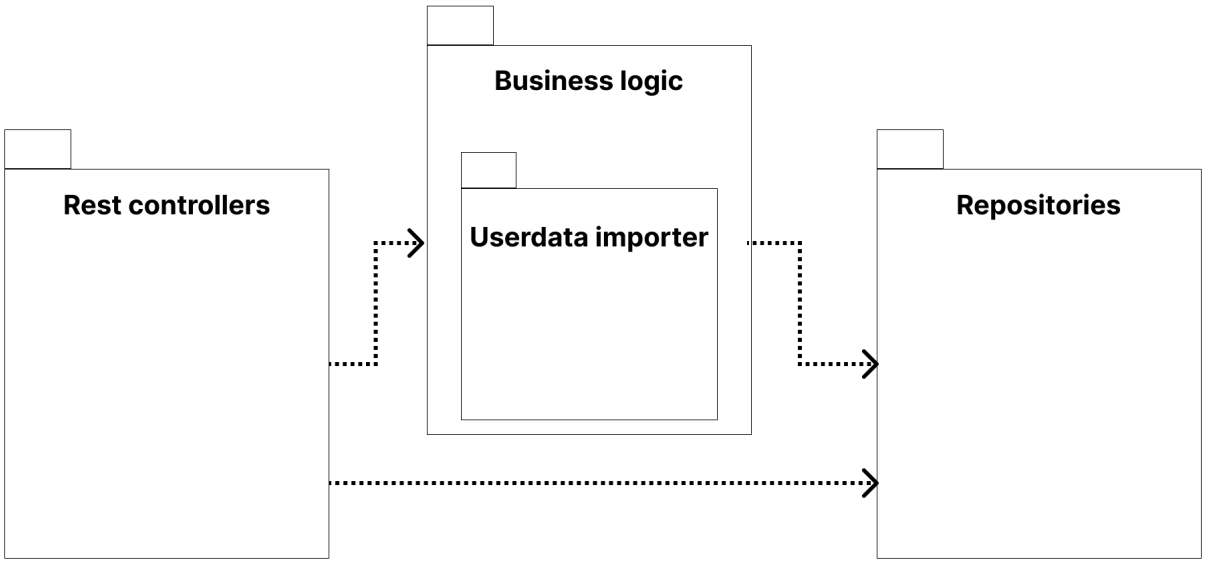

Рис. 3.3 – Архітектура Portfolio service

У шарі логіки на рисунку 3.3 зображено компонент Userdata important, котрий відповідає за імпорт даних про фінансову діяльність користувача. Розглянемо його архітектуру.

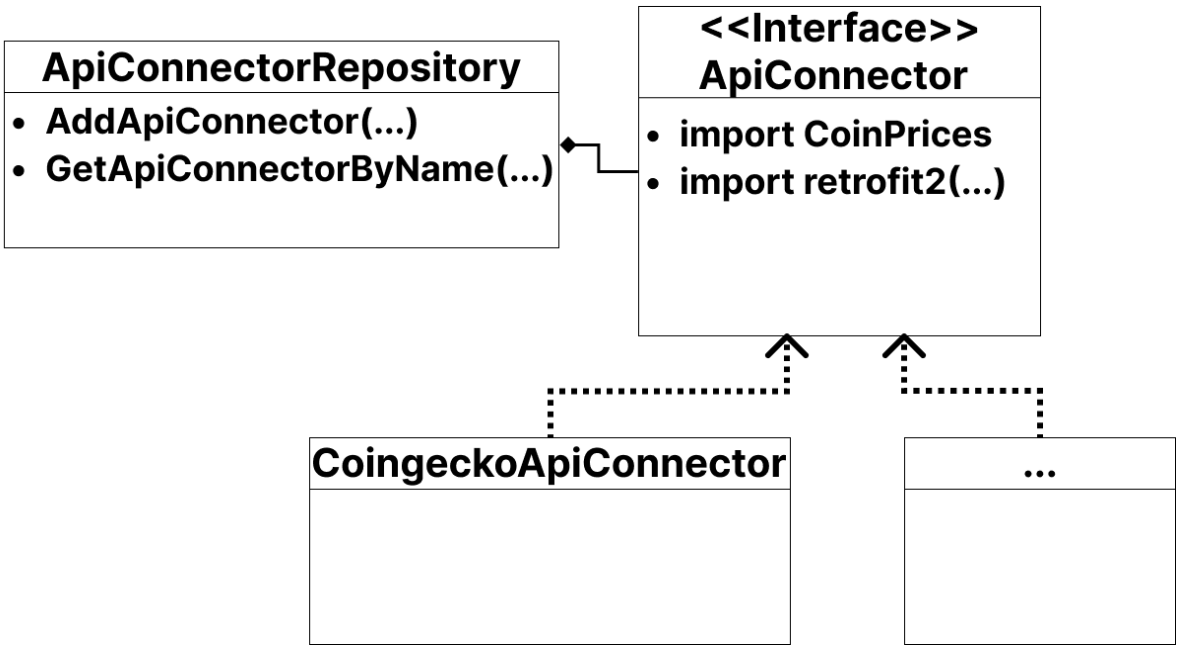

Рис. 3.4 – Архітектура Userdata important

Marketdata provider має таку ж саму шарувату архітектуру як і у portfolioservice.

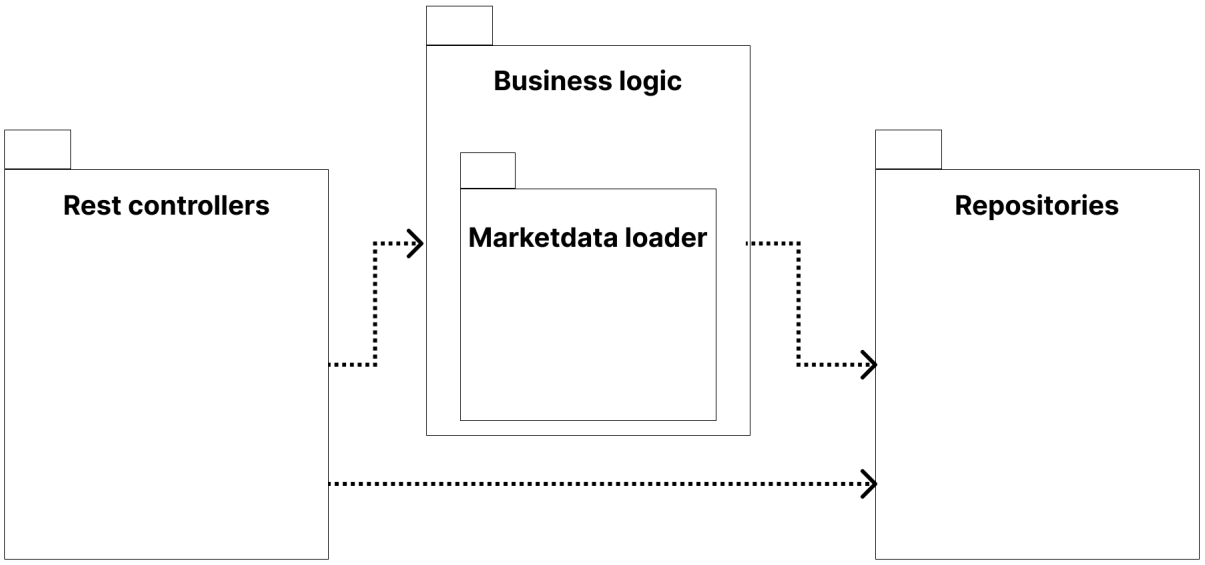

Рис. 3.5 – Архітектура Marketdata provider

РОЗДІЛ 4 ШИФРУВАННЯ

4.1. Загальні відомості

У сучасному світі криптовалют стає все більше та більше, а разом з цим зростає і необхідність в безпеці та захисті цих цінних активів. Оскільки криптовалюти зберігаються та обробляються у цифровому форматі, вони стають схильними до ризику злому або крадіжки. Це робить безпеку мобільних застосунків для криптовалютних портфелів надзвичайно важливою.

Шифрування є ключовим елементом забезпечення безпеки в мобільних застосунках для криптовалютних портфоліо. Воно дозволяє зашифровувати дані, щоб лише авторизовані користувачі мали доступ до них. Шифрування використовує різні алгоритми та ключі для перетворення звичайного тексту в нерозбірливий код, який може бути розкодований лише знанням правильного ключа. Це гарантує, що навіть якщо зловмисники отримають доступ до зашифрованих даних, вони не зможуть розкодувати їх без ключа.

У мобільних застосунках для криптовалютних портфоліо шифрування використовується для захисту конфіденційності приватних ключів, паролів та інших чутливих даних. Воно забезпечує надійний бар'єр між користувачем та зловмисниками, запобігаючи несанкціонованому доступу до криптовалютних активів.

Найпопулярніші види шифрування даних [14] у мобільних застосунках криптовалютних портфоліо включають:

1. AES (Advanced Encryption Standard): AES є одним з найбільш використовуваних алгоритмів симетричного шифрування. Він забезпечує надійний рівень шифрування даних та захист від несанкціонованого доступу.

2. RSA (Rivest-Shamir-Adleman): RSA є алгоритмом асиметричного шифрування, який використовується для захисту приватних ключів та обміну даними між користувачами. Він забезпечує безпеку комунікації та захист від перехоплення даних.

3. ECC (Elliptic Curve Cryptography): ECC є асиметричним шифруванням, яке використовується для створення ключів та підпису даних. Воно відоме своєю високою ефективністю та міцністю шифрування при низькому розмірі ключа.

4. HMAC (Hash-based Message Authentication Code): HMAC є алгоритмом хешування, який використовується для перевірки цілісності та автентифікації повідомлень. Він забезпечує захист від модифікації даних під час передачі.

5. PGP (Pretty Good Privacy): PGP є протоколом шифрування, який використовується для безпечного обміну повідомленнями та захисту даних. Він використовує комбінацію симетричного та асиметричного шифрування для забезпечення конфіденційності та цілісності даних.

Ці шифрувальні методи використовуються для забезпечення безпеки даних у мобільних застосунках криптовалютного портфоліо. Важливо, щоб застосунки використовували надійні алгоритми шифрування та дотримувалися кращих практик безпеки для захисту конфіденційності та безпеки криптовалютних активів користувачів.

4.2. Шифрування AES

Для даного мобільного застосунку було вибрано шифрування AES. Розглянемо його більш детально.

AES (Advanced Encryption Standard) [12] є одним з найбільш популярних і надійних алгоритмів симетричного шифрування, який використовується для захисту даних у мобільних застосунках криптовалютного портфоліо. Він був прийнятий у 2001 році як заміна застарілого стандарту DES (Data Encryption Standard) і став промисловим стандартом для шифрування даних.

Основні особливості AES полягають у його ефективності, швидкості та надійності шифрування. Алгоритм AES використовує блокову криптографію, де дані поділяються на блоки фіксованої довжини і обробляються окремо.

За замовчуванням AES [12] використовує блоки довжиною 128 біт, але також підтримується шифрування з більшою довжиною блоків (192 та 256 біт).

AES використовує симетричний ключовий обмін, що означає, що той самий ключ використовується як для шифрування, так і для розшифрування даних. Ключ може мати різну довжину (128, 192 або 256 біт), що впливає на рівень безпеки шифрування. Чим довший ключ, тим складніше його перебрати шляхом злому.

Однією з переваг AES є його швидкодія. Використовуючи ефективні математичні операції, AES може шифрувати та розшифровувати дані з високою швидкістю, що робить його ідеальним для мобільних пристроїв з обмеженими ресурсами.

AES також відомий своєю надійністю та стійкістю до атак. Він використовує сучасні криптографічні методи, які ускладнюють можливість злому шифру. Багато організацій та урядів використовують AES для захисту конфіденційних даних.

У мобільних застосунках криптовалютного портфоліо AES використовується для шифрування приватних ключів, паролів та інших конфіденційних даних, що забезпечує їх безпеку під час зберігання та передачі через мережу.

4.3 Архітектура шифрування AES

Рівняння шифрування AES [12] складається з різних кроків, включаючи SubBytes, ShiftRows, MixColumns та AddRoundKey. Більш детально розглянемо загальний опис кожного з кроків:

1. SubBytes:

У кроці SubBytes, кожен байт вхідного блоку замінюється на відповідний байт з S-Box таблиці заміни. S-Box використовується для заміни значень байтів на основі їх розташування у таблиці.

2. ShiftRows:

У кроці ShiftRows, байти в кожному рядку вхідного блоку зсуваються вліво на певну кількість позицій. Кожному рядку відповідає свій власний зсув.

3. MixColumns:

У кроці MixColumns, кожна колонка вхідного блоку множиться на певну матрицю заміщення. Ця матриця заміщення змішує байти в колонках, забезпечуючи додаткову криптографічну стійкість.

4. AddRoundKey:

У кроці AddRoundKey, до кожного байта вхідного блоку додається відповідний раундовий ключ. Раундові ключі генеруються на основі основного ключа шифрування і використовуються для поетапного шифрування блоків.

Рівняння для кожного з цих кроків мають математичні операції, що включають побітові операції, множення у полі Галуа та додавання. Деталі кожного кроку можуть бути складними для представлення у вигляді одного рівняння, оскільки вони включаються у послідовність дій.

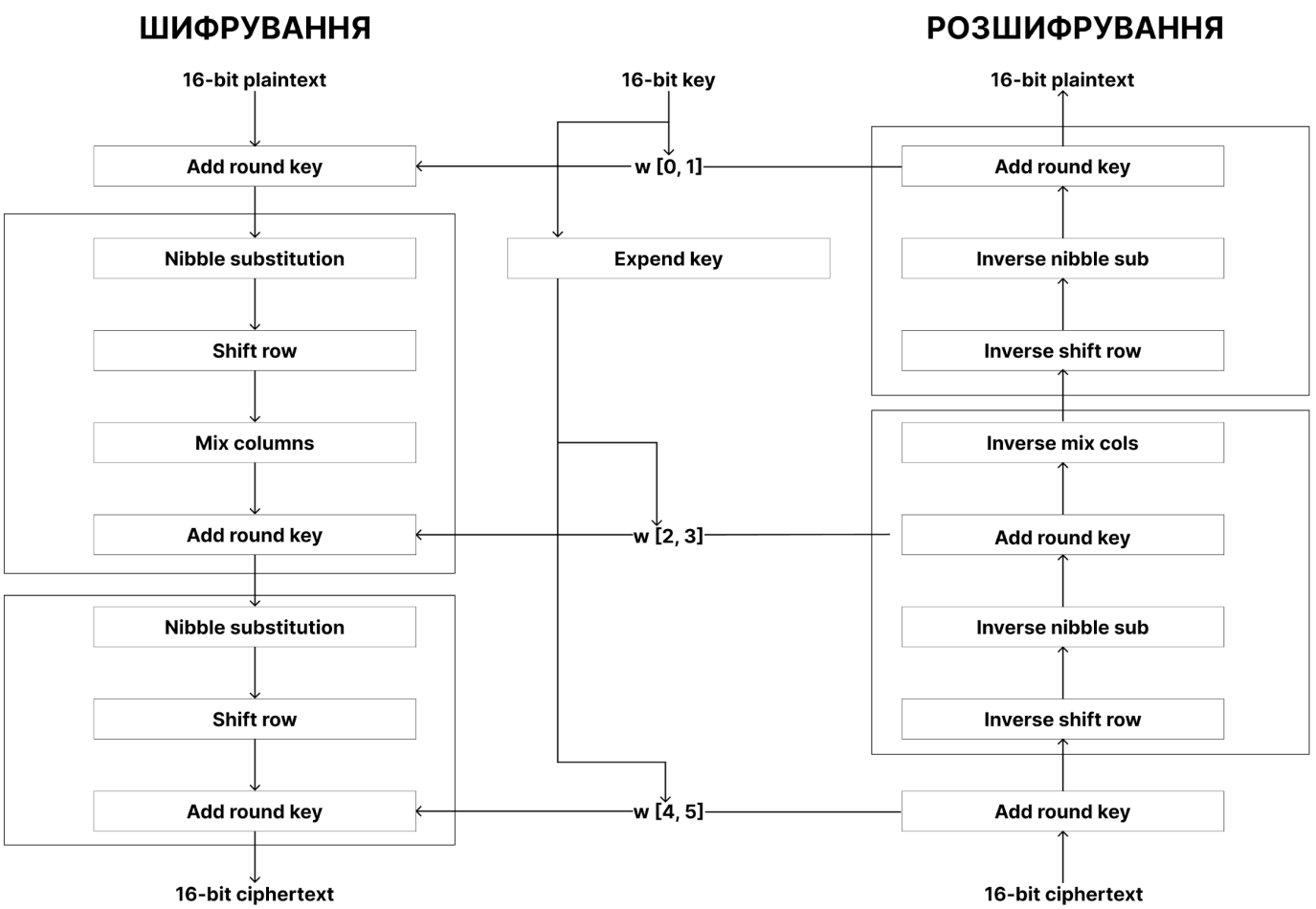

Рис. 4.1 – Архітектура шифрування AES

РОЗДІЛ 5 АРХІТЕКТУРА МОБІЛЬНОГО ДОДАТКУ

У даному розділі розглянемо інструменти та методи розробки мобільного додатку для відстеження інвестиційного портфоліо для криптовалютних активів.

5.1. Середовище розробки ANDROID STUDIO

Android Studio [1] – це інтегроване середовище розробки (IDE) для платформи Android. Воно надає розширені інструменти і функціональні можливості для створення, редагування, збирання і налагодження Android-додатків. Android Studio базується на популярній IDE IntelliJ IDEA і розроблене спеціально для роботи з платформою Android.

Головні переваги Android Studio включають наступне:

1. Розширений редактор коду: Android Studio має потужний редактор коду, який надає підсвічування синтаксису, автодоповнення, перехід до визначення функцій і багато іншого. Це спрощує написання коду і полегшує виявлення помилок.

2. Візуальний редактор макетів: За допомогою Android Studio можна використовувати візуальний редактор макетів, який дозволяє створювати і редагувати інтерфейс користувача за допомогою перетягування та розміщення компонентів.

3. Вбудована симуляція і тестування: Android Studio [1] надає засоби для симуляції різних пристроїв Android, що дозволяє перевіряти та налагоджувати додатки безпосередньо на комп'ютері. Крім того, воно має вбудовані інструменти для автоматизованого тестування додатків.

4. Інтеграція з Gradle: Android Studio використовує систему збирання проектів Gradle, що дозволяє ефективно управляти залежностями, збирати додатки та керувати конфігураціями проекту.

5.2. Gradle

Градл (Gradle) [3] – це система збирання проектів та автоматизації робочих процесів, яка широко використовується в розробці Android-додатків.

Вона дозволяє визначити і керувати залежностями проекту, налаштовувати різні варіанти збирання, виконувати тестування, розгортання і багато іншого.

Основні переваги Gradle [3] для розробки Android-додатків включають наступне:

1. Декларативний синтаксис: Gradle використовує декларативний синтаксис на основі DSL (Domain-Specific Language), що дозволяє легко визначати та керувати завданнями збирання проекту.

2. Масштабованість: Gradle розроблений з урахуванням масштабованості, що дозволяє ефективно керувати проектами будь-якої складності, включаючи великі та складні структури проектів.

3. Широкі можливості налаштування: Завдяки гнучкій системі плагінів та конфігураційних параметрів, Gradle дозволяє налаштовувати різні варіанти збирання, включаючи різні варіанти залежностей, ресурсів, розгортання тощо.

4. Інтеграція з Android Studio: Gradle [3] має глибоку інтеграцію з Android Studio, що дозволяє зручно керувати проектами, налаштовувати залежності та збирати додатки безпосередньо з IDE.

Android Studio та Gradle спільно створюють потужні інструменти для розробки Android-додатків, що дозволяють розробникам зручно та ефективно створювати, тестувати та розгортати свої додатки на платформі Android.

5.3. База даних PostgreSQL

PostgreSQL [4] є потужною реляційною базою даних, яка використовується у проєктах Android для зберігання та керування даними. PostgreSQL забезпечує надійну, безпечну та масштабовану базу даних з великою кількістю функцій, які полегшують роботу з даними.

Ось кілька способів, які дозволяють взаємодіяти з базою даних PostgreSQL у проєктах Android:

1. JDBC (Java Database Connectivity): JDBC є стандартним інтерфейсом Java для взаємодії з реляційними базами даних, включаючи PostgreSQL. Також можна використовувати JDBC для встановлення з'єднання з базою даних, виконання SQL-запитів, отримання результатів та керування транзакціями.

2. ORM (Object-Relational Mapping): ORM-бібліотеки, такі як Room, Hibernate або JOOQ, надають абстракцію бази даних, яка дозволяє працювати з об'єктами Java та Kotlin замість прямих SQL-запитів. Це полегшує розробку та управління базою даних PostgreSQL у проєктах Android.

3. Бібліотеки доступу до даних: Деякі сторонні бібліотеки, такі як Exposed або jOOQ, надають високорівневий інтерфейс для роботи з базою даних PostgreSQL у Kotlin. Вони дозволяють використовувати мову Kotlin для виконання SQL-запитів, опису схеми бази даних та взаємодії з об'єктами даних.

4. PostgreSQL API: PostgreSQL [4] також надає власний API, який може бути використаний для взаємодії з базою даних безпосередньо з Android-проєктом. Також можна використовувати PostgreSQL API для створення з'єднання з базою даних, виконання запитів та керування даними.

5.4. Java

Java [2] є однією з основних мов програмування, яка використовується у розробці додатку для платформи Android.

Основні ролі Java у розробці Android включають наступне:

1. Основна мова програмування: Java є основною мовою програмування для розробки додатків на платформі Android. Вона надає розробникам широкий набір інструментів, бібліотек і фреймворків, які сприяють швидкому і ефективному розробленню додатків.

2. Платформна незалежність: Однією з ключових переваг Java є її платформна незалежність. Це означає, що код, написаний на Java, може працювати на різних операційних системах, включаючи Android. Це дозволяє розробникам створювати додатки, які можуть працювати на різних пристроях Android з різними версіями операційної системи.

3. Широкий набір бібліотек та фреймворків: Для розробки Android-додатків на Java доступний великий вибір бібліотек та фреймворків. Наприклад, Android SDK, що включає в себе різноманітні інструменти та бібліотеки, розроблені спеціально для роботи з Android. Також існує багато сторонніх бібліотек, які допомагають розробникам вирішувати конкретні завдання швидше та ефективніше.

Java [2] відіграє ключову роль у розробці Android-додатків, надаючи розробникам потужні інструменти та різноманітні можливості. Її популярність, платформна незалежність та широкий вибір бібліотек роблять її однією з найпоширеніших мов програмування у цій сфері.

5.5. Kotlin

Kotlin [5] – це сучасна, статично типізована мова програмування, що працює на віртуальній машині Java (JVM). Вона була розроблена компанією JetBrains із метою створення мови, яка була б безпечною, експресивною та сумісною з Java. Kotlin став офіційною мовою програмування для розробки Android-додатків в 2017 році, і з тих пір став дедалі популярнішим серед

розробників.

Ось кілька ключових рис Kotlin та його роль у створенні Android-проєктів:

1. Сумісність з Java: Kotlin повністю сумісний з Java, що дозволяє розробникам поступово мігрувати існуючі проекти Java на Kotlin без проблем. Це означає, що можна використовувати існуючий код на Java, бібліотеки та фреймворки в проектах Kotlin, а також використовувати код Kotlin з Java-проектами.

2. Безпечність та нульова безпека: Kotlin пропонує безпеку типів на рівні мови, завдяки чому більше шансів уникнути помилок під час виконання програми. Він також надає підтримку для нульової безпеки, що допомагає уникнути NullPointerException, поширеної проблеми в Java.

3. Синтаксис інтуїтивно зрозумілий: Kotlin має чистий та лаконічний синтаксис, що дозволяє писати код більш зрозуміло та експресивно. Він пропонує такі функціональні можливості, як лямбда-вирази, функції-розширення та операції над нульовими значеннями.

4. Корутини та асинхронне програмування: Kotlin має вбудовану підтримку корутинів, що дозволяє легко виконувати асинхронні операції та уникати блокування основного потоку. Це особливо корисно для розробки Android-додатків, де потрібно виконувати мережеві виклики, завантаження файлів та інші операції, які можуть зайняти багато часу.

5. Розширення Android-фреймворку: Kotlin [5] пропонує деякі розширення для Android-фреймворку, що полегшують розробку. Наприклад, він має нульовий оператор для безпечного виклику методів, розширення для роботи з макетами (англ. layouts) та декларативну специфікацію інтерфейсу користувача за допомогою Android Jetpack.

5.6. Type Script

TypeScript [6] – мова програмування, яка розширює стандартну мову JavaScript, додаючи до неї статичний типізм та інші функціональні можливості. Вона використовується для розробки веб-додатків, включаючи бекенд частини.

Основна роль TypeScript у створенні бекенду полягає в наступному:

1. Статичний типізм: TypeScript надає можливість використовувати статичний типізм у коді. Це означає, що можна визначити типи змінних, параметрів та повернутих значень. Це дозволяє виявляти помилки під час компіляції, забезпечуючи більшу надійність та підтримку в розробці бекенду.

2. Підтримка сучасних функцій: TypeScript [6] підтримує сучасні функціональні можливості JavaScript, такі як лямбда-функції, високорівневі функції, асинхронні функції та інші. Це дозволяє розробникам писати чистий, структурований та ефективний код для бекенду.

3. Підтримка модульності: TypeScript надає підтримку модульності, що дозволяє організувати код бекенду у логічні модулі. Це допомагає підтримувати код, робить його більш зрозумілим та зручним для розробників.

4. Інтеграція з інструментами розробки: TypeScript інтегрується з різними інструментами розробки, такими як IDE (наприклад, Visual Studio Code), компілятори та засоби перевірки коду. Це спрощує процес розробки, забезпечуючи автодоповнення, перевірку синтаксису та інші корисні функції.

5. Розширені можливості рефакторингу: TypeScript надає розширені можливості рефакторингу коду, такі як перейменування змінних, вилучення методів, витягування інтерфейсів тощо. Це допомагає зробити код бекенду більш чистим, модульним та ефективним.

Усі ці функції та можливості TypeScript роблять його потужним інструментом у розробці бекенду. Він сприяє покращенню надійності, підтримці сучасних функцій, організації коду та зручній інтеграції з іншими інструментами розробки.

5.7. Visual Studio Code

Visual Studio Code (VS Code) [13] – це безкоштовний редактор коду, розроблений компанією Microsoft. Він став популярним серед розробників завдяки своїм потужним можливостям, розширюваності та хорошій продуктивності. Ось деякі важливі аспекти, що стосуються Visual Studio Code:

1. Крос-платформений редактор: Visual Studio Code доступний для Windows, macOS та Linux. Це дозволяє розробникам використовувати його на будь-якій платформі свого вибору.

2. Розширюваність: VS Code має велику екосистему розширень, що дозволяє розробникам налаштувати редактор під свої потреби. Розширення додають нові функціональні можливості, підтримку мов програмування, інтеграцію зі сторонніми сервісами та інше.

3. Підтримка багатьох мов програмування: VS Code [13] надає підтримку для широкого спектра мов програмування, включаючи JavaScript, TypeScript, Python, C#, Java, Go, PHP та багато інших. Редактор надає підсвічування синтаксису, автодоповнення, перевірку помилок та інші інструменти, щоб полегшити розробку.

4. Вбудовані функції та інструменти: Visual Studio Code має ряд корисних функцій, таких як вбудований термінал, систему керування версіями, засоби для налагодження коду, інтеграцію з системами контролю версій (наприклад, Git) та інші.

5. Спільна робота та розширені можливості: VS Code підтримує роботу в команді, дозволяючи спільно редагувати код, обмінюватись коментарями та використовувати інші інструменти для співпраці. Крім того, редактор має вбудовану інтеграцію з популярними сервісами розробки, такими як GitHub.

6.Налаштування та зовнішній вигляд: VS Code дозволяє розробникам налаштовувати редактор, включаючи теми оформлення, шрифти, розташування панелей і т.д. Користувачі можуть також використовувати різні розширення для покращення вигляду та робочого потоку.

Visual Studio Code є популярним вибором серед розробників завдяки своїй легкості використання, розширюваності та потужним можливостям. Він підтримує широкий спектр мов програмування та надає багато корисних інструментів, що полегшують процес розробки програмного забезпечення.

РОЗДІЛ 6 ТЕСТУВАННЯ ДОДАТКУ

Апробація додатку є важливим етапом в процесі розробки програмного забезпечення. Це процес тестування та перевірки функціональності, якості та відповідності додатку вимогам та очікуванням користувачів перед його випуском на ринок. Апробація дозволяє переконатися, що додаток працює належним чином, не має помилок та забезпечує задоволення потреб користувачів.

Під час апробації додатку проводяться різноманітні тести, які мають на меті перевірити різні аспекти його роботи. Це включає функціональні тести, які перевіряють, чи працюють всі функції додатку правильно; тести на навантаження, що дозволяють визначити, як додаток поводиться при великому обсязі даних або одночасному доступі користувачів; тести на безпеку, які перевіряють стійкість додатку до атак та забезпечують захист конфіденційності та цілісності даних.

Апробація додатку включає також тестування на різних пристроях, щоб переконатися, що додаток працює стабільно та оптимально на різних мобільних пристроях. Крім того, проводяться тести на взаємодію з іншими додатками та системами, щоб забезпечити сумісність та інтеграцію зовнішніх компонентів.

Метою апробації додатку є впевненість у його готовності до випуску на ринок. Після успішного проходження апробації можна впевнено стверджувати, що додаток відповідає всім вимогам та очікуванням, що він працює стабільно, безпечно та забезпечує задоволення користувачів.

6.1. Головний екран

Головний екран мобільного додатку пропонує користувачеві швидкий та зручний доступ до основної інформації про його криптовалютний портфель. Ця сторінка відображає зведену інформацію про різні валюти в портфоліо, включаючи їх поточну ціну та зміни ціни за останній тиждень.

На головному екрані користувач може бачити графік, який ілюструє

зміни ціни валют протягом останнього тижня. Цей графік дозволяє швидко оцінити тенденції зростання або падіння цін і дати уявлення про стан портфеля.

Окрім цього, головний екран може включати додаткову інформацію, таку як загальна вартість портфоліо, процентний приріст або зниження, а також важливі новини або повідомлення, пов'язані з криптовалютами та ринком.

Цей головний екран створений з метою надати користувачу швидкий огляд його криптовалютного портфоліо, відображаючи ключову інформацію та тренди змін цін. Він допомагає зрозуміти стан та ефективність портфоліо та дозволяє користувачеві приймати обґрунтовані рішення щодо управління своїми криптовалютними інвестиціями.

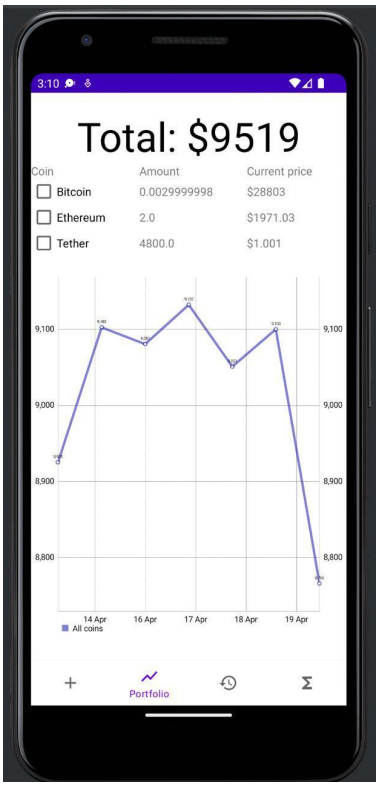

Рис. 6.1 – Головний екран додатку

6.2. Аналіз графіків

Головний екран мобільного додатку пропонує користувачу не тільки перегляд загальної інформації про криптовалютний портфель, але й можливість порівняти графіки окремих валют. За допомогою чекбоксів,

розташованих поряд з кожною валютою, користувач може вибрати конкретні валюти, які бажає порівняти.

Після вибору валют, користувач може переглянути їх графіки, які відображають зміни цін протягом останнього тижня. Цей функціонал дозволяє швидко порівняти тенденції зростання або падіння цін між різними валютами в портфелі. Він надає користувачеві можливість зробити об'єктивні порівняння та прийняти інформовані рішення щодо свого криптовалютного портфоліо.

Ця функція покликана допомогти користувачеві отримати додаткову аналітичну інформацію та легше виявити залежності між різними валютами. Вона розширює можливості користувача в оцінці та аналізі його криптовалютного портфоліо, що сприяє більш ефективному управлінню інвестиціями та прийняттю обґрунтованих рішень.

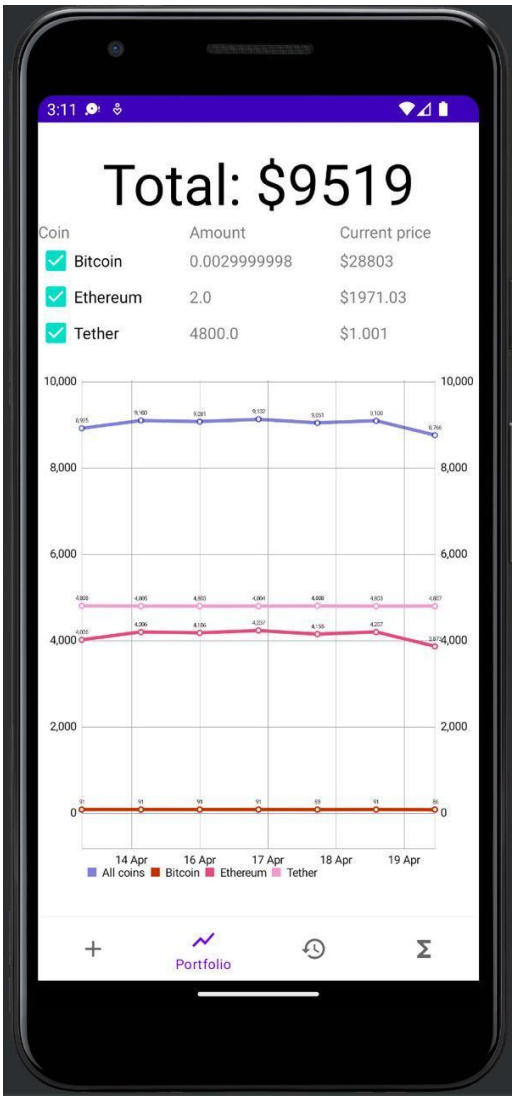

Рис 6.2 – Порівняння графіків

6.3. Оновлення портфоліо

На вкладці Add є можливість додавати оновлення до криптовалютного портфоліо, пов'язані з купівлею або продажем. У ній можна вказати яку конкретну валюту було куплено або продано, кількість одиниць, за яку ціну це було зроблено та дату операції. Це дає можливість точно відстежувати торгові дії і зберігати відповідні записи в портфоліо.

Додавання цих даних дозволяє зручно вести облік криптовалютного портфоліо та відстежувати зміни вартості криптовалютних активів.

Такий підхід дозволяє зберігати точні дані про торгові операції, а також проводити аналіз портфоліо на основі історичних даних. Це допоможе

отримати зрозумілу картину криптовалютного портфоліо, розрахувати поточний баланс, виявити прибуткові або збиткові операції та зробити осмислені фінансові рішення на основі цих даних.

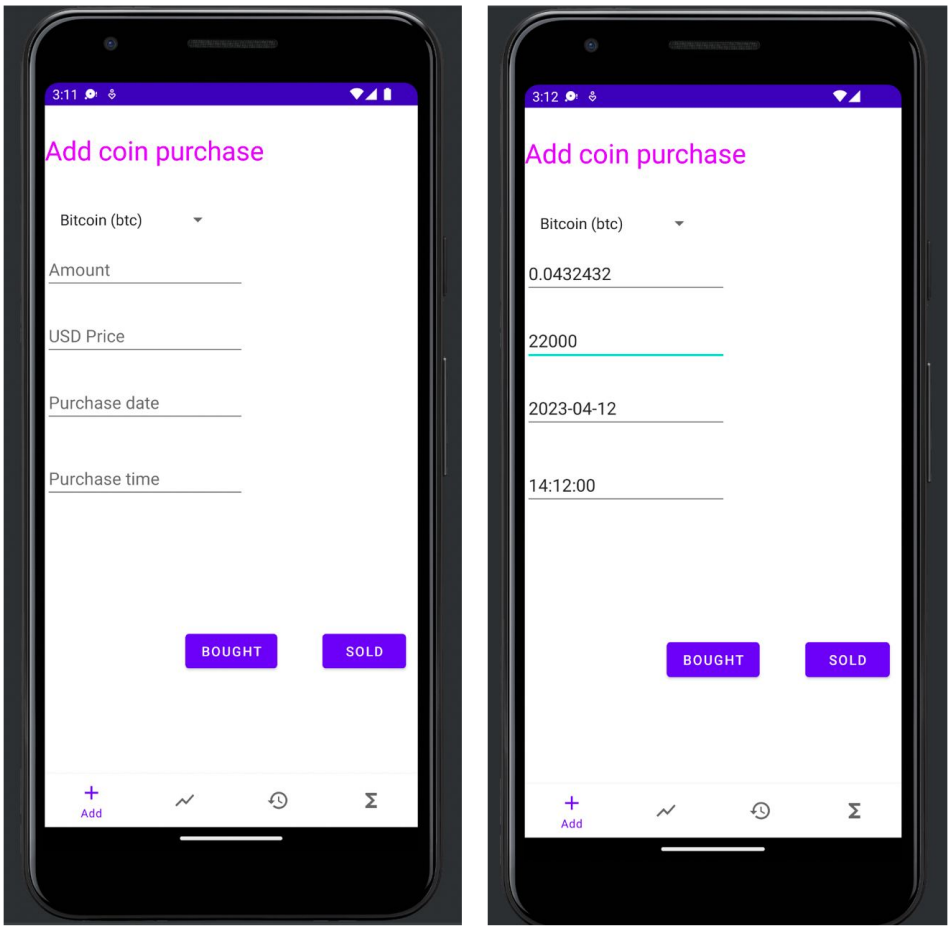

Рис. 6.3 – Оновлення портфоліо

6.4. Оракул. Алгоритми прогрозування

Остання вкладка в додатку називається "Оракул". Вона дозволяє отримати прогноз ціни на конкретну криптовалюту в майбутньому, використовуючи історичні дані. На основі цих даних, система здійснює екстраполяцію, що дозволяє приблизно визначити напрямок, в якому може рухатися ціна у майбутньому.

Важливо зазначити, що система надає можливість експериментувати з різними алгоритмами екстраполяції, можна вибрати різні підходи, включаючи використання штучного інтелекту, щоб прогнозувати майбутні зміни цін на

криптовалюти. Це дозволяє зробити свої власні дослідження та спробувати різні алгоритми, щоб отримати найбільш точний та надійний прогноз.

Використання "Оракула" допоможе зробити більш обґрунтовані рішення щодо криптовалютних інвестицій. Засновуючись на прогнозах цін, можна визначити оптимальний час для купівлі або продажу активів, а також виробити більш стратегічний підхід до управління криптовалютним портфелем.

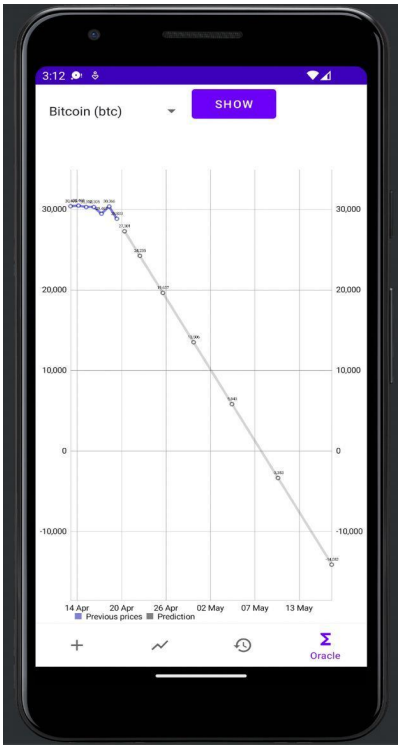

Рис. 6.4 – Вкладка «Оракул»

ВИСНОВОК

У даній кваліфікаційній роботі було досліджено та розроблено мобільний додаток для криптовалютного портфоліо. Мета полягала в створенні зручного та функціонального інструменту, який дозволяє користувачам ефективно відстежувати та керувати своїми криптовалютними інвестиціями.

У процесі розробки додатку, були ретельно проаналізовані вимоги та потреби користувачів. В процесі створення додатку враховано основні функціональні можливості, які були визначені, такі як перегляд портфоліо по різних валютах, графіки зміни цін, додавання та оновлення транзакцій, аналіз інформації про криптовалюти, а також прогнозування майбутніх змін цін.

Додаток включає зручний та привабливий інтерфейс, що дозволяє користувачам легко орієнтуватися у додатку та здійснювати необхідні операції. Також було приділено особливу увагу безпеці, використовуючи надійний метод шифрування для захисту конфіденційності даних користувачів.

У результаті роботи, було створено функціональний мобільний додаток для криптовалютного портфоліо, який буде корисним інструментом для інвесторів та трейдерів. Додаток дозволяє користувачам ефективно керувати своїми інвестиціями, відстежувати зміни цін та отримувати прогнози для прийняття обґрунтованих рішень.

У майбутньому, планується продовжувати розвивати додатку, додавши нові функціональні можливості, такі як інтеграція з різними біржами, детальніша статистика та аналітика, а також розширення можливостей прогнозування майбутніх змін цін.

В цілому, робота по створенню мобільного додатку для криптовалютного портфоліо була успішною, додаток може стати незамінним помічником для інвесторів у сфері криптовалют, надаючи їм зручний та надійний інструмент для управління їхніми активами.

СПИСОК ЛІТЕРАТУРИ

1. Package Search Official Website Android studio — 2022 — URL: https://developer.android.com/studio
2. Getting Started with Java — 2022 — URL: https://dev.java/learn/getting-started/
3. Android Gradle plugin release notes — 2023 — URL: https://developer.android.com/build/releases/gradle-plugin
4. PostgreSQL 15.3 Documentation — 2023 — URL: https://www.postgresql.org/docs/current/
5. Kotlin 1.8.21 — 2023 — URL: https://github.com/JetBrains/kotlin/releases/tag/v1.8.21
6. TypeScript for JavaScript Programmers — 2023 — URL: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html
7. Official Website Blockfolio — 2023 — URL: https://ftx.com/
8. Official Website Coinbase — 2023 — URL: https://www.coinbase.com/
9. Official Website Binance — 2023 — URL: https://www.binance.com/ru-UA
10. Official Website Kraken — 2023 — URL: https://www.kraken.com/uk-ua
11. Guide to app architecture — 2023 — URL: https://developer.android.com/topic/architecture
12. The Design of Rijndael: AES - The Advanced Encryption Standard, 2023. с. 120 — 152.
13. Package Search Official Website Visual Studio Code — 2023 — URL: https://code.visualstudio.com/
14. Secure Mobile Development: How to Secure Mobile Applications. J. D. Glaser, 2014.с.56 — 98.

ДОДАТОК А

```
package com.example.ktportfolio
import CoinPrices
import android.graphics.Color
import android.os.Bundle
import android.util.Log
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.CheckBox
import android.widget.TableLayout
import android.widget.TableRow
import android.widget.TextView
import androidx.fragment.app.Fragment
import com.example.ktportfolio.api.Coin
import com.example.ktportfolio.api.WeekPrice
import com.example.ktportfolio.storage.Portfolio
import com.github.mikephil.charting.charts.LineChart
import com.github.mikephil.charting.components.AxisBase
import com.github.mikephil.charting.components.XAxis
import com.github.mikephil.charting.data.Entry
import com.github.mikephil.charting.data.LineData
import com.github.mikephil.charting.data.LineDataSet
import com.github.mikephil.charting.formatter.ValueFormatter
import kotlinx.coroutines.GlobalScope
import kotlinx.coroutines.launch
import java.text.SimpleDateFormat
import java.util.*
```

```
class PortfolioFragment : Fragment() {
    private val coinSet = mutableSetOf<String>()

    private var coinList: List<Coin>? = null

    private var portfolio: List<Portfolio>? = null

    private var week: List<WeekPrice>? = null

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_portfolio, container, false)
    }

    override fun onStart() {
        super.onStart()
        fetchPortfolio()
    }

    private fun fetchPortfolio() {
        val cthis = this
        GlobalScope.launch {
            val activity = requireActivity()

            val coinsList = Accessor.getCoinsList()
            val currentCoinsPrice = Accessor.getCurrentCryptoPrice(null)
            val week = Accessor.getWeekPrices()
```

```
            val portfolio = Accessor.getPortfolio(activity.applicationContext)
            val total = Accessor.getTotal(activity.applicationContext)

            activity.runOnUiThread(kotlinx.coroutines.Runnable {
                activity.findViewById<TextView>(R.id.total).text = "Total:
\$$total"
                drawPortfolio(coinsList, portfolio, currentCoinsPrice)
                cthis.coinList = coinsList
                cthis.portfolio = portfolio
                cthis.week = week
                cthis.drawGraph()
            })
        }
    }

    private fun drawPortfolio(coinList: List<Coin>, portfolio: List<Portfolio>,
prices: List<CoinPrices>) {
        val activity = requireActivity()
        val view = requireView()

        val table = view.findViewById<TableLayout>(R.id.purchase_table)
        val coinNameMap = coinList.associateBy({ it.symbol }, { it.name })
        val coinIdMap = coinList.associateBy({ it.symbol }, { it.id })
        val coinPriceMap = prices.associateBy({ it.coin }, { it.prices[0].price })

        while (table.childCount > 1) {
            val row = table.getChildAt(1)
            table.removeView(row)
        }
```

```
    for (item in portfolio) {
        val row = TableRow(activity)

        val cb = CheckBox(activity)
        cb.text = coinNameMap[item.coin]
        cb.setOnClickListener { onCoinCheckBoxClick(it) }
        row.addView(cb)

        val amount = TextView(activity)
        amount.text = item.sum.toString()
        row.addView(amount)

        val price = TextView(activity)
        val p = coinPriceMap[coinIdMap[item.coin]]
        price.text = "\$$p"
        row.addView(price)

        table.addView(row)
    }
}

private fun onCoinCheckBoxClick(view: View?) {
    val cb = view as CheckBox;
    val coin = cb.text.toString().lowercase()
    if (cb.isChecked) {
        coinSet.add(coin)
    } else {
        coinSet.remove(coin)
    }
```

```
    this.drawGraph()
}

private fun drawGraph() {
    Log.i("graph", coinList.toString())
    Log.i("graph", portfolio.toString())
    Log.i("graph", week.toString())
    if (coinList == null || portfolio == null || week == null) {
        return
    }
    val coinNameMap = coinList!!.associateBy({ it.id }, { it.symbol })
    val portfolioMap = portfolio!!.associateBy({ it.coin }, { it.sum })
    val allEntries = ArrayList<Entry>()
    val byCoinEntries = mutableMapOf<String, ArrayList<Entry>>()

    for (day in week!!) {
        val ts = day.timestamp.toFloat()
        var sum = 0.0f
        for (coinPrice in day.prices) {
            val found = portfolioMap[coinNameMap[coinPrice.coin]]
            if (found != null) {
                val dayCoinTotal = found.times(coinPrice.price).toFloat()
                sum += dayCoinTotal
                if (byCoinEntries[coinPrice.coin] == null) {
                    byCoinEntries[coinPrice.coin] = ArrayList()
                }
                byCoinEntries[coinPrice.coin]!!.add(Entry(ts, dayCoinTotal))
            }
        }
        allEntries.add(Entry(ts, sum))
```

```
        }

        val chart = requireView().findViewById<LineChart>(R.id.idGraphView)
        class XDateFormatter:ValueFormatter() {
            override fun getAxisLabel(value: Float, axis: AxisBase?): String {
                val format = SimpleDateFormat("dd MMM")
                return format.format(Date(value.toLong()))
            }
        }
        chart.xAxis.position = XAxis.XAxisPosition.BOTTOM
        chart.xAxis.valueFormatter = XDateFormatter()
        chart.description.text = ""

        val allSets = ArrayList<LineDataSet>()
        allSets.add(createDataSet(allEntries,
androidx.appcompat.R.color.material_blue_grey_800, "All coins"))
        for (item in byCoinEntries) {
            if (!coinSet.contains(item.key)) {
                continue
            }
            val rnd = Random()
            allSets.add(createDataSet(
                item.value,
                Color.argb(255, rnd.nextInt(256), rnd.nextInt(256),
rnd.nextInt(256)),
                "${item.key.capitalize()}",
            ))
        }

        val lineData = LineData(allSets.toList())
```

```
        chart.data = lineData
        chart.invalidate()
    }

    private fun createDataSet(entries: ArrayList<Entry>, color: Int, label:
String): LineDataSet {
        val dataSet = LineDataSet(entries, label)
        dataSet.setCircleColor(color)
        dataSet.color = color
        dataSet.valueTextSize = 5.0f
        dataSet.lineWidth = 3.0f
        return dataSet
    }
}
import androidx.annotation.NonNull;
import androidx.room.DatabaseConfiguration;
import androidx.room.InvalidationTracker;
import androidx.room.RoomOpenHelper;
import androidx.room.RoomOpenHelper.Delegate;
import androidx.room.RoomOpenHelper.ValidationResult;
import androidx.room.migration.AutoMigrationSpec;
import androidx.room.migration.Migration;
import androidx.room.util.DBUtil;
import androidx.room.util.TableInfo;
import androidx.room.util.TableInfo.Column;
import androidx.room.util.TableInfo.ForeignKey;
import androidx.room.util.TableInfo.Index;
import androidx.sqlite.db.SupportSQLiteDatabase;
import androidx.sqlite.db.SupportSQLiteOpenHelper;
import androidx.sqlite.db.SupportSQLiteOpenHelper.Callback;
```

```
import androidx.sqlite.db.SupportSQLiteOpenHelper.Configuration;
import java.lang.Class;
import java.lang.Override;
import java.lang.String;
import java.lang.SuppressWarnings;
import java.util.Arrays;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

@SuppressWarnings({"unchecked", "deprecation"})
public final class AppDatabase_Impl extends AppDatabase {
  private volatile PurchaseDao _purchaseDao;

  @Override
  protected SupportSQLiteOpenHelper
createOpenHelper(DatabaseConfiguration configuration) {
    final SupportSQLiteOpenHelper.Callback _openCallback = new
RoomOpenHelper(configuration, new RoomOpenHelper.Delegate(1) {
      @Override
      public void createAllTables(SupportSQLiteDatabase _db) {
        _db.execSQL("CREATE TABLE IF NOT EXISTS `Purchase` (`uid`
INTEGER PRIMARY KEY AUTOINCREMENT, `coin` TEXT NOT NULL,
`amount` REAL NOT NULL, `purchase_usd_price` REAL NOT NULL,
`purchase_date` INTEGER NOT NULL)");
        _db.execSQL("CREATE TABLE IF NOT EXISTS room_master_table
(id INTEGER PRIMARY KEY,identity_hash TEXT)");
```

```
    _db.execSQL("INSERT OR REPLACE INTO room_master_table
(id,identity_hash) VALUES(42, 'd16376327778a34d9624302e15b1ca24')");
  }

  @Override
  public void dropAllTables(SupportSQLiteDatabase _db) {
   _db.execSQL("DROP TABLE IF EXISTS `Purchase`");
   if (mCallbacks != null) {
    for (int _i = 0, _size = mCallbacks.size(); _i < _size; _i++) {
     mCallbacks.get(_i).onDestructiveMigration(_db);
    }
   }
  }

  @Override
  protected void onCreate(SupportSQLiteDatabase _db) {
   if (mCallbacks != null) {
    for (int _i = 0, _size = mCallbacks.size(); _i < _size; _i++) {
     mCallbacks.get(_i).onCreate(_db);
    }
   }
  }

  @Override
  public void onOpen(SupportSQLiteDatabase _db) {
   mDatabase = _db;
   internalInitInvalidationTracker(_db);
   if (mCallbacks != null) {
    for (int _i = 0, _size = mCallbacks.size(); _i < _size; _i++) {
     mCallbacks.get(_i).onOpen(_db);
```

```
        }
      }
    }

    @Override
    public void onPreMigrate(SupportSQLiteDatabase _db) {
      DBUtil.dropFtsSyncTriggers(_db);
    }

    @Override
    public void onPostMigrate(SupportSQLiteDatabase _db) {
    }

    @Override
    protected RoomOpenHelper.ValidationResult
onValidateSchema(SupportSQLiteDatabase _db) {
      final HashMap<String, TableInfo.Column> _columnsPurchase = new
HashMap<String, TableInfo.Column>(5);
      _columnsPurchase.put("uid", new TableInfo.Column("uid",
"INTEGER", false, 1, null, TableInfo.CREATED_FROM_ENTITY));
      _columnsPurchase.put("coin", new TableInfo.Column("coin", "TEXT",
true, 0, null, TableInfo.CREATED_FROM_ENTITY));
      _columnsPurchase.put("amount", new TableInfo.Column("amount",
"REAL", true, 0, null, TableInfo.CREATED_FROM_ENTITY));
      _columnsPurchase.put("purchase_usd_price", new
TableInfo.Column("purchase_usd_price", "REAL", true, 0, null,
TableInfo.CREATED_FROM_ENTITY));
      _columnsPurchase.put("purchase_date", new
TableInfo.Column("purchase_date", "INTEGER", true, 0, null,
TableInfo.CREATED_FROM_ENTITY));
```

```
        final HashSet<TableInfo.ForeignKey> _foreignKeysPurchase = new
HashSet<TableInfo.ForeignKey>(0);
        final HashSet<TableInfo.Index> _indicesPurchase = new
HashSet<TableInfo.Index>(0);
        final TableInfo _infoPurchase = new TableInfo("Purchase",
_columnsPurchase, _foreignKeysPurchase, _indicesPurchase);
        final TableInfo _existingPurchase = TableInfo.read(_db, "Purchase");
        if (! _infoPurchase.equals(_existingPurchase)) {
          return new RoomOpenHelper.ValidationResult(false,
"Purchase(com.example.ktportfolio.storage.Purchase).\n"
                  + " Expected:\n" + _infoPurchase + "\n"
                  + " Found:\n" + _existingPurchase);
        }
        return new RoomOpenHelper.ValidationResult(true, null);
      }
    }, "d16376327778a34d9624302e15b1ca24",
"5492822a5fefc8718ed15209ac2cf40f");
    final SupportSQLiteOpenHelper.Configuration _sqliteConfig =
SupportSQLiteOpenHelper.Configuration.builder(configuration.context)
        .name(configuration.name)
        .callback(_openCallback)
        .build();
    final SupportSQLiteOpenHelper _helper =
configuration.sqliteOpenHelperFactory.create(_sqliteConfig);
    return _helper;
  }

  @Override
  protected InvalidationTracker createInvalidationTracker() {
```

```
    final HashMap<String, String> _shadowTablesMap = new
HashMap<String, String>(0);
    HashMap<String, Set<String>> _viewTables = new HashMap<String,
Set<String>>(0);
    return new InvalidationTracker(this, _shadowTablesMap, _viewTables,
"Purchase");
  }

  @Override
  public void clearAllTables() {
    super.assertNotMainThread();
    final SupportSQLiteDatabase _db =
super.getOpenHelper().getWritableDatabase();
    try {
      super.beginTransaction();
      _db.execSQL("DELETE FROM `Purchase`");
      super.setTransactionSuccessful();
    } finally {
      super.endTransaction();
      _db.query("PRAGMA wal_checkpoint(FULL)").close();
      if (!_db.inTransaction()) {
        _db.execSQL("VACUUM");
      }
    }
  }

  @Override
  protected Map<Class<?>, List<Class<?>>> getRequiredTypeConverters() {
    final HashMap<Class<?>, List<Class<?>>> _typeConvertersMap = new
HashMap<Class<?>, List<Class<?>>>();
```

```
    _typeConvertersMap.put(PurchaseDao.class,
PurchaseDao_Impl.getRequiredConverters());
    return _typeConvertersMap;
  }

  @Override
  public Set<Class<? extends AutoMigrationSpec>>
getRequiredAutoMigrationSpecs() {
    final HashSet<Class<? extends AutoMigrationSpec>>
_autoMigrationSpecsSet = new HashSet<Class<? extends
AutoMigrationSpec>>();
    return _autoMigrationSpecsSet;
  }

  @Override
  public List<Migration> getAutoMigrations(
      @NonNull Map<Class<? extends AutoMigrationSpec>,
AutoMigrationSpec> autoMigrationSpecsMap) {
    return Arrays.asList();
  }

  @Override
  public PurchaseDao purchaseDao() {
    if (_purchaseDao != null) {
      return _purchaseDao;
    } else {
      synchronized(this) {
        if(_purchaseDao == null) {
          _purchaseDao = new PurchaseDao_Impl(this);
        }
```

```
            return _purchaseDao;
          }
        }
      }
}
    import android.database.Cursor;
    @SuppressWarnings({"unchecked", "deprecation"})
    public final class PurchaseDao_Impl implements PurchaseDao {
      private final RoomDatabase __db;

      private final EntityInsertionAdapter<Purchase>
    __insertionAdapterOfPurchase;

      private final EntityDeletionOrUpdateAdapter<Purchase>
    __deletionAdapterOfPurchase;

      private final SharedSQLiteStatement __preparedStmtOfClear;

      public PurchaseDao_Impl(RoomDatabase __db) {
        this.__db = __db;
        this.__insertionAdapterOfPurchase = new
    EntityInsertionAdapter<Purchase>(__db) {
          @Override
          public String createQuery() {
            return "INSERT OR ABORT INTO `Purchase`
    (`uid`,`coin`,`amount`,`purchase_usd_price`,`purchase_date`) VALUES
    (?,?,?,?,?)";
          }

          @Override
```

```
    public void bind(SupportSQLiteStatement stmt, Purchase value) {
      if (value.getUid() == null) {
        stmt.bindNull(1);
      } else {
        stmt.bindLong(1, value.getUid());
      }
      if (value.getCoin() == null) {
        stmt.bindNull(2);
      } else {
        stmt.bindString(2, value.getCoin());
      }
      stmt.bindDouble(3, value.getAmount());
      stmt.bindDouble(4, value.getPurchaseUsdPrice());
      stmt.bindLong(5, value.getDate());
    }
  };
  this.__deletionAdapterOfPurchase = new
EntityDeletionOrUpdateAdapter<Purchase>(__db) {
    @Override
    public String createQuery() {
      return "DELETE FROM `Purchase` WHERE `uid` = ?";
    }

    @Override
    public void bind(SupportSQLiteStatement stmt, Purchase value) {
      if (value.getUid() == null) {
        stmt.bindNull(1);
      } else {
        stmt.bindLong(1, value.getUid());
      }
```

```
    }
  };
  this.__preparedStmtOfClear = new SharedSQLiteStatement(__db) {
    @Override
    public String createQuery() {
      final String _query = "DELETE FROM purchase";
      return _query;
    }
  };
}

@Override
public void insertAll(final Purchase... purchase) {
  __db.assertNotSuspendingTransaction();
  __db.beginTransaction();
  try {
    __insertionAdapterOfPurchase.insert(purchase);
    __db.setTransactionSuccessful();
  } finally {
    __db.endTransaction();
  }
}

@Override
public void delete(final Purchase purchase) {
  __db.assertNotSuspendingTransaction();
  __db.beginTransaction();
  try {
    __deletionAdapterOfPurchase.handle(purchase);
    __db.setTransactionSuccessful();
```

```
    } finally {
      __db.endTransaction();
    }
  }

  @Override
  public void clear() {
    __db.assertNotSuspendingTransaction();
    final SupportSQLiteStatement _stmt = __preparedStmtOfClear.acquire();
    __db.beginTransaction();
    try {
      _stmt.executeUpdateDelete();
      __db.setTransactionSuccessful();
    } finally {
      __db.endTransaction();
      __preparedStmtOfClear.release(_stmt);
    }
  }

  @Override
  public List<Purchase> getAll() {
    final String _sql = "SELECT * FROM purchase ORDER BY
purchase_date DESC";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    __db.assertNotSuspendingTransaction();
    final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
    try {
      final int _cursorIndexOfUid =
CursorUtil.getColumnIndexOrThrow(_cursor, "uid");
```

```
      final int _cursorIndexOfCoin =
CursorUtil.getColumnIndexOrThrow(_cursor, "coin");
      final int _cursorIndexOfAmount =
CursorUtil.getColumnIndexOrThrow(_cursor, "amount");
      final int _cursorIndexOfPurchaseUsdPrice =
CursorUtil.getColumnIndexOrThrow(_cursor, "purchase_usd_price");
      final int _cursorIndexOfDate =
CursorUtil.getColumnIndexOrThrow(_cursor, "purchase_date");
      final List<Purchase> _result = new
ArrayList<Purchase>(_cursor.getCount());
      while(_cursor.moveToNext()) {
        final Purchase _item;
        final Integer _tmpUid;
        if (_cursor.isNull(_cursorIndexOfUid)) {
          _tmpUid = null;
        } else {
          _tmpUid = _cursor.getInt(_cursorIndexOfUid);
        }
        final String _tmpCoin;
        if (_cursor.isNull(_cursorIndexOfCoin)) {
          _tmpCoin = null;
        } else {
          _tmpCoin = _cursor.getString(_cursorIndexOfCoin);
        }
        final float _tmpAmount;
        _tmpAmount = _cursor.getFloat(_cursorIndexOfAmount);
        final float _tmpPurchaseUsdPrice;
        _tmpPurchaseUsdPrice =
_cursor.getFloat(_cursorIndexOfPurchaseUsdPrice);
        final long _tmpDate;
```

```
        _tmpDate = _cursor.getLong(_cursorIndexOfDate);
        _item = new
Purchase(_tmpUid,_tmpCoin,_tmpAmount,_tmpPurchaseUsdPrice,_tmpDate)
;
        _result.add(_item);
      }
      return _result;
    } finally {
      _cursor.close();
      _statement.release();
    }
  }

  @Override
  public List<Purchase> loadAllByCoin(final ArrayList<String> coins) {
    StringBuilder _stringBuilder = StringUtil.newStringBuilder();
    _stringBuilder.append("SELECT * FROM purchase WHERE coin IN (");
    final int _inputSize = coins.size();
    StringUtil.appendPlaceholders(_stringBuilder, _inputSize);
    _stringBuilder.append(") ORDER BY purchase_date DESC");
    final String _sql = _stringBuilder.toString();
    final int _argCount = 0 + _inputSize;
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql,
_argCount);
    int _argIndex = 1;
    for (String _item : coins) {
      if (_item == null) {
        _statement.bindNull(_argIndex);
      } else {
        _statement.bindString(_argIndex, _item);
```

```
      }
      _argIndex ++;
    }
    __db.assertNotSuspendingTransaction();
    final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
    try {
      final int _cursorIndexOfUid =
CursorUtil.getColumnIndexOrThrow(_cursor, "uid");
      final int _cursorIndexOfCoin =
CursorUtil.getColumnIndexOrThrow(_cursor, "coin");
      final int _cursorIndexOfAmount =
CursorUtil.getColumnIndexOrThrow(_cursor, "amount");
      final int _cursorIndexOfPurchaseUsdPrice =
CursorUtil.getColumnIndexOrThrow(_cursor, "purchase_usd_price");
      final int _cursorIndexOfDate =
CursorUtil.getColumnIndexOrThrow(_cursor, "purchase_date");
      final List<Purchase> _result = new
ArrayList<Purchase>(_cursor.getCount());
      while(_cursor.moveToNext()) {
        final Purchase _item_1;
        final Integer _tmpUid;
        if (_cursor.isNull(_cursorIndexOfUid)) {
          _tmpUid = null;
        } else {
          _tmpUid = _cursor.getInt(_cursorIndexOfUid);
        }
        final String _tmpCoin;
        if (_cursor.isNull(_cursorIndexOfCoin)) {
          _tmpCoin = null;
        } else {
```

```
          _tmpCoin = _cursor.getString(_cursorIndexOfCoin);
        }
        final float _tmpAmount;
        _tmpAmount = _cursor.getFloat(_cursorIndexOfAmount);
        final float _tmpPurchaseUsdPrice;
        _tmpPurchaseUsdPrice =
_cursor.getFloat(_cursorIndexOfPurchaseUsdPrice);
        final long _tmpDate;
        _tmpDate = _cursor.getLong(_cursorIndexOfDate);
        _item_1 = new
Purchase(_tmpUid,_tmpCoin,_tmpAmount,_tmpPurchaseUsdPrice,_tmpDate)
;
        _result.add(_item_1);
      }
      return _result;
    } finally {
      _cursor.close();
      _statement.release();
    }
  }

  @Override
  public List<Portfolio> getPortfolio() {
    final String _sql = "SELECT coin, SUM(amount) as sum FROM purchase
GROUP BY coin";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    __db.assertNotSuspendingTransaction();
    final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
    try {
      final int _cursorIndexOfCoin = 0;
```

```
      final int _cursorIndexOfSum = 1;
      final List<Portfolio> _result = new
ArrayList<Portfolio>(_cursor.getCount());
      while(_cursor.moveToNext()) {
        final Portfolio _item;
        final String _tmpCoin;
        if (_cursor.isNull(_cursorIndexOfCoin)) {
          _tmpCoin = null;
        } else {
          _tmpCoin = _cursor.getString(_cursorIndexOfCoin);
        }
        final float _tmpSum;
        _tmpSum = _cursor.getFloat(_cursorIndexOfSum);
        _item = new Portfolio(_tmpCoin,_tmpSum);
        _result.add(_item);
      }
      return _result;
    } finally {
      _cursor.close();
      _statement.release();
    }
  }

  @Override
  public int getTotal() {
    final String _sql = "SELECT SUM(amount * purchase_usd_price) FROM
purchase";
    final RoomSQLiteQuery _statement = RoomSQLiteQuery.acquire(_sql, 0);
    __db.assertNotSuspendingTransaction();
    final Cursor _cursor = DBUtil.query(__db, _statement, false, null);
```

```
      try {
        final int _result;
        if(_cursor.moveToFirst()) {
          _result = _cursor.getInt(0);
        } else {
          _result = 0;
        }
        return _result;
      } finally {
        _cursor.close();
        _statement.release();
      }
    }

    public static List<Class<?>> getRequiredConverters() {
      return Collections.emptyList();
    }
}
```