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

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

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка мобільного додатку для виконання операцій з криптовалютою із використанням технології Firebase на мові програмування Java

Виконав(ла): студент(ка) 4 курсу, групи СП-42
спеціальності 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

Осадця Р. Б.
(підпис) (прізвище та ініціали)

Керівник Бойко І. В.
(підпис) (прізвище та ініціали)

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

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

Рецензент
(підпис) (прізвище та ініціали)

Тернопіль
2024

АНОТАЦІЯ

Кваліфікаційна робота бакалавра за спеціальністю 121 - Інженерія програмного забезпечення на тему “Розробка мобільного додатку для виконання операцій з криптовалютою із використанням технології Firebase на мові програмування Java”. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп’ютерно - інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СП-42. Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню “бакалавр” містить: 70 с., 50 рис., 4 додатки, 23 джерела.

Мета даної кваліфікаційної роботи полягає у проведенні аналізу, розробці і реалізації Android додатку для виконання операцій з криптовалютою. Для цього було проведено аналіз вже існуючих програмних систем та, базуючись на результатах дослідження потреб користувачів, спроектовано та сконструйовано власний мобільний додаток.

В результаті було створено програмну систему, що дозволяє крипто-інвесторам вводити та зберігати записи своїх транзакцій легко та зручно у своєму мобільному телефоні.

Враховуючи неймовірну популярність криптовалюти в останні роки, цей мобільний додаток має практичну користь для великої кількості людей і може стати необхідним для кожного з нас вже дуже скоро.

Ключові слова: Android, Kotlin, Android Studio, Firebase, Криптовалюта, Інвестиції.

ANNOTATION

Bachelor's qualifying work on the specialty 121 - Software engineering on the topic "Development of a mobile application for performing operations with cryptocurrency using Firebase technology in the Java programming language." Ternopil National Technical University named after Ivan Pulyuya, faculty of computer and information systems and software engineering, department of software engineering, group SP-42. The explanatory note to the qualification work for obtaining the bachelor's degree contains: 70 pages, 50 images, 4 appendices, 23 sources.

The purpose of this qualification work is to analyze, develop and implement an Android application for performing transactions with cryptocurrency. For this, an analysis of already existing software systems was carried out and, based on the results of the research of user needs, an own mobile application was designed and constructed.

The result was a software system that allows crypto-investors to enter and store their transaction records easily and conveniently on their mobile phone.

Considering the incredible popularity of cryptocurrency in recent years, this mobile application has practical benefits for a large number of people and may become a necessity for all of us very soon.

Keywords: Android, Kotlin, Android Studio, Firebase, Cryptocurrency, Investment.

ЗМІСТ

ВСТУП

У наш час мобільні додатки відіграють надзвичайно важливу роль і стали найбільш зручним інструментом для економії часу і ресурсів для кожного з нас. Зазвичай вони мають більший функціонал, ніж вебсайти, оскільки вони тісніше пов’язані з операційною системою нашого пристрою, тому і надають змогу до більш зручних налаштувань.

Об’єктом розробки у дипломній роботі є створення мобільного додатку на Android, що передбачає можливість отримувати інформацію з ринку криптовалют, а також створювати власне портфоліо на основі вибраних транзакцій користувача. Основні функції розроблюваного додатку включають отримання актуальної інформації про ціни на криптовалюту, аналіз ринкових тенденцій і відстеження змін у цінах. Використовувати програмну систему можна як для навчальних, так і для інвестиційних цілей. Даний додаток розроблено з використанням мови програмування Java за допомогою інтегрованого середовища розробки Android Studio, що є одним із передових інструментів для створення програм такого призначення. Також для забезпечення зручності користувачів використано технологію Firebase, яка дозволяє зберігати та обробляти дані в реальному часі, а також надає можливість авторизації користувача за допомогою логіну та паролю, таким чином забезпечуючи високий рівень надійності та швидкодії додатку.

Мета даної дипломної роботи полягає у проведенні аналізу, розробці і реалізації Android додатку для виконання операцій з криптовалютою з використанням технології Firebase та написанні виконуваного коду на мові програмування Java. Основним завданням є аналіз ринку криптовалют та потреб користувачів у цій сфері, а потім проектування та конструювання інтерфейсу та функціональності нашого додатку. Заключним етапом є аналіз отриманих результатів. Загальний обсяг роботи також включає розробку інтуїтивного та привабливого інтерфейсу, який дозволить легко орієнтуватися в додатку і швидко виконувати потрібні операції.

Розроблена програмна система спеціалізована не тільки для підвищення комфорту користувачів, а і й збільшення потенційного прибутку для інвесторів. У сучасному світі, де володіння криптоактивами стає все більш популярним, дуже важливо володіти найактульнішою інформацією для прийняття правильних рішень, що дозволить ефективно управляти своїми фінансами та доходами, мінімізуючи ризики та максимізуючи можливості заробітку на даному багатоконкурентному ринку.

1 РОЗРОБКА ПРОГРАМНОЇ ЧАСТИНИ

1.1 Аналітичний огляд

1.1.1 Аналіз предметної області

Криптовалюта - це цифрові гроші, які працюють на основі застосування технології блокчейн. На відміну від номінальною валюти, такої як гривня, євро чи долар, немає центрального банківського органу, що відповідає за емісію фінансових активів, таким чином цей актив є цілком децентралізованим.

На сьогоднішній день криптовалюта, як явище, розвинулася настільки різноманітно, що ми можемо розраховуватися нею за товари та послуги так само легко, як це робимо за допомогою стандартної дебетової картки. Незважаючи на це, більшість людей купують цифрові монети з метою інвестування, оскільки цей ринок має дуже високу ефективність і може принести великі прибутки своїм вкладникам.

Ще у 1980-і роки з’явилася ідея створити цифрові гроші. Її розвивали американські криптографи Девід Чаум і Стефан Брендс. Вони пояснили, як працює анонімна система цифрових платежів, і запропонували перші протоколи цифрових грошей [1].

Якщо говорити про принцип роботи криптовалюти, то основна суть технології полягає в тому, що вся інформація про транзакції зберігається в блокчейні, який реалізується у вигляді ланцюжків блоків, кожен з яких містить набір транзакцій та підписів, а також хеш попереднього блоку [2].

Технологія блокчейн допомагає користувачам виконувати транзакції напряму і без посередників, таких як банк, оскільки кожен користувач має доступ до всієї історії транзакцій.

Один з основних плюсів використання криптовалюти полягає в тому, що вона може служити платіжною валютою по всьому світу і не обмежується кордонами країн або юридичними обмеженнями. Будь-хто може переказати її знаходячись в

Україні в інший кінець світу, і це відбудеться дуже швидко, незалежно від суми, яку буде переказано.

Для зберігання криптовалют зазвичай використовують криптогаманці. Цей спосіб є одним з найбезпечніших. Криптогаманці поділяються на два види: гарячі (онлайн) та холодні (офлайн). За допомогою цих інструментів люди можуть безпечно зберігати та керувати своєю криптовалютою. Серед найпопулярніших онлайн гаманців можна виділити MetaMask, інтерфейс якого зображений на рисунку 1.1.

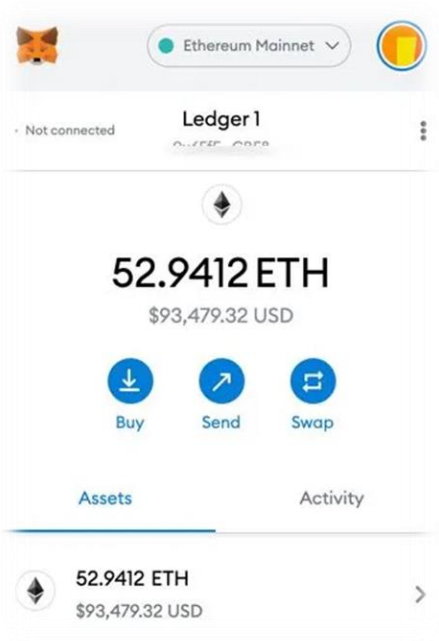

Рисунок 1.1 - Додаток MetaMask

Якщо йдеться про офлайн гаманці, то вони не потребують підключення до мережі для доступу до ваших коштів, а тому вони менш вразливі до хакерських атак. До найбільш популярного холодного гаманця можна віднести Ledger Nano S Plus, який зображений на рисунку 1.2.

Рисунок 1.2 - Холодний гаманець Ledger Nano S Plus

Основною задачею всіх гаманців є забезпечення максимальної безпеки для активів. Іноді цей захист настільки сильний, що люди, які втратили свій пароль від гаманця, більше ніколи не зможуть відновити доступ.

Після вибору криптогаманця часто виникає питання, які саме криптовалюти слід купувати. Базуючись на історії тренду, можна сказати, що завжди варто мати у своєму портфелі стабільні та перевірені часом монети. Наприклад, можна розглядати Bitcoin та Ethereum. Також варто розширити свій портфель активами з середньою капіталізацією, що приносить більший ризик, проте і потенційний прибуток буде більшим.

Одним з найважливіших питань для людей, які користуються криптовалютою, є те, як слідкувати за поточними трендами та тим, як змінюється цінність їхнього портфелю.

На даний момент цю проблему вирішують спеціальні програми для керування криптопортфелем. Це можуть бути як вебсайти, так і мобільні додатки для операційної системи Android чи IOS. Ці складні криптотрекери надають користувачам можливість відстежувати їхні грошові дані та записувати транзакції. Також вони допомагають налаштовувати свої портфелі відповідно до своїх

конкретних потреб. Більшість платформ дають змогу вручну вносити виконані транзакції або можливість зв’язатись з криптобіржею для автоматичних оновлень.

1.1.2 Постановка задачі

У сучасному світі технології змінюються і розвиваються надзвичайно швидко. На даному етапі найбільш зручним інструментом для кожного з нас є мобільний телефон. Кожен користувач хоче, щоб всі програми на його пристрої працювали швидко та без будь-яких помилок, і найкращим рішенням є нативний мобільний додаток.

Основним завданням даної роботи є аналіз ринку криптовалют та потреб цільової аудиторії, і після чого - проектування та конструювання програмної частини додатку.

Даний додаток повинен мати такі функції, як інтерфейс авторизації, алгоритм пошуку криптоактивів за назвою та додавання їх до свого портфеля. Крім того, користувач повинен мати змогу відстежувати зміни, які відбулися у його портфелі протягом останньої доби та за весь час.

Важливо зберігати всі дані користувачів конфіденційно та забезпечити максимальну безпеку для даних, оскільки ніхто не хотів би, щоб треті особи мали доступ до інформації про наявність тих чи інших активів у інвестора.

Однією з ключових особливостей розробленої системи є інтеграція технології Firebase у додаток. Це дозволяє досягти високого рівня безпеки, швидкості та надійності проведених нами операцій.

Можливість входу в додаток за допомогою облікового запису Google або комбінації електроної пошти та пароля дозволить користувачеві створювати кілька портфелів, пов’язаних з різними обліковими записами. Інтеграція технології Firebase Firestore дозволить користувачеві зберігати його дані віддалено, що є

більш зручним методом і виключає ризик втрати даних при зміні пристрою користувача або проблемах з операційною системою. Також дуже актуальним активом наразі є NFT токени. NFT – це основані на блокчейні цифрові записи про право власності й автентичність, пов’язані з певним мультимедійним об’єктом [3]. Оскільки NFT зараз набуває більшої популярності, наш додаток буде містити окремий список з найбільш трендових з NFT токенів на даний момент.

Функція пошуку монет за назвою дозволить зекономити час інвестора, а графічне відображення тренду кожної монети зробить наш інтерфейс зрозумілим та зручним.

Дана робота має практичний потенціал у сфері мобільних фінансових додатків і відкриває широкі можливості для розвитку даної галузі та полегшення повсякденної роботи користувачів.

1.1.3 Аналіз аналогічних проектів

Існує велика кількість програм та веб-сайтів, які допомагають керувати криптовалютним портфелем. Багато з них можуть похвалитись своєю великою кількістю різноманітного функціоналу, але часто це не означає, те, що користуватись цим додатком зручно користувачеві.

Вибір найкращого додатка для криптовалютного портфоліо відіграє велику роль, оскільки від цього залежатимуть наші інвестиційні рішення, які ми прийматимемо.

Найперше, при виборі найзручнішої програми для себе, потрібно звернути увагу на операційну систему, якою ви зазвичай користуєтеся. Оскільки найкращі додатки можуть відрізнітись для різних операційних систем, ми проаналізуємо аналогічні проекти саме для Android.

Однією з найважливіших характеристик при виборі є надійність та репутація криптотрекера. Потрібно брати до уваги лише ті проекти, які добре зарекомендували себе та мають велику кількість відгуків та масштабне ком’юніті навколо себе. Це важливо, оскільки не хочеться в якийсь момент прокинутися зранку і зрозуміти, що всі твої дані про транзакції просто зникли.

Не менш важливим для додатку для відстеження криптовалют є зручний інтерфейс. Чітка навігація дозволить користувачеві швидко отримувати доступ до необхідної їм інформації без витрати часу на так званий “Онбординг”.

Грунтуючись на цих критеріях, можна зробити висновок, що найбільш популярним і зручним додатком на даний момент є додаток CoinMarketCap. Інтерфейс додатку можна побачити на рисунку 1.3.

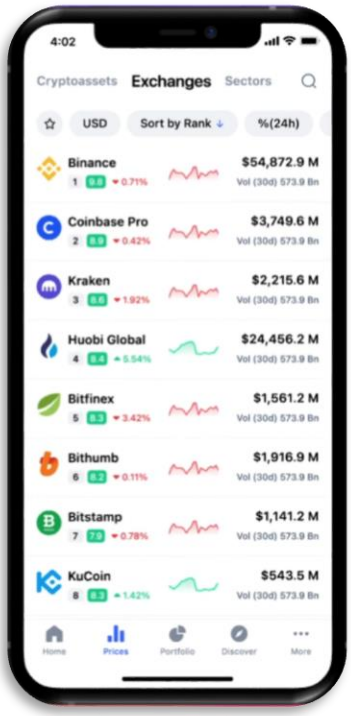

Рисунок 1.3 - Інтерфейс додатка CoinMarketCap

CoinMarketCap має понад 10 мільйонів завантажень та 275 тисяч відгуків у Google Play. Цей додаток був створений як аналог вебсайту, а у 2020 році найбільша криптобіржа Binance придбала його.

Додаток надає аналіз будь-якої з багатьох тисяч різних монет та діаграми до них, що є надзвичайно корисним для максимальної зрозумілості інтерфейсу. Ви

можете створити будь-яку кількість окремих портфелів і отримувати дані в реальному часі для кожного з них. Також він пропонує можливість створення кількох списків спостереження з можливістю ділитися ними з друзями, а також стежити за списками від інших інвесторів.

Найголовнішою перевагою цієї програми є доступність великого обсягу даних по ринку, якого немає у інших конкурентів, і свій профіль можна вести повністю безкоштовно. Інтерфейс екрану портфеля зображений на рисунку 1.4.

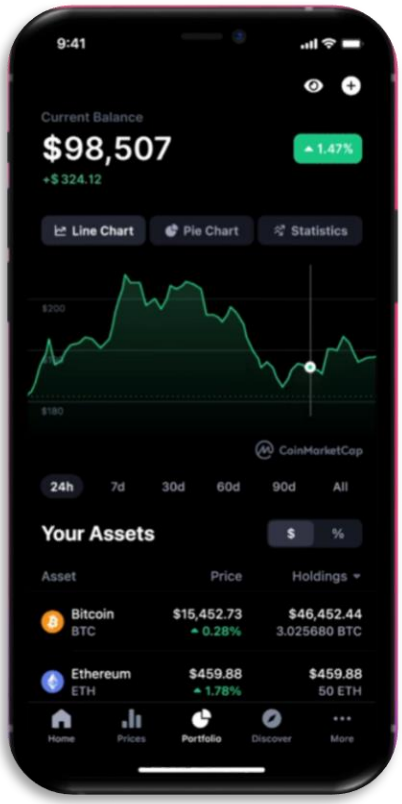

Рисунок 1.4 - Екран портфеля користувача в CoinMarketCap

Серед недоліків можна віднести те, що при першому знайомстві додаток може бути досить складним для розуміння для новачків.

Іншим прикладом чудового додатку для відстеження криптовалюти є CoinGecko. Подібно до CoinMarketCap, цей додаток також надає вичерпну інформацію про ринок. Він також пропонує користувачам різноманітні функції, такі як огляд портфоліо, новини, сповіщення та інше.

Також додаток пропонує можливість щодня збирати свої бонуси, які ви зможете обмінювати на нагороди, такі як NFT, знижки та інші.

CoinGecko дозволяє користуватися більшістю функціоналу повністю безкоштовно, але з підпискою ви матимете можливість отримати ексклюзивний NFT, вимкнути рекламу в додатку, отримати ранній доступ до нововведень сайту і доступ до закритої групи, де можна спілкуватися з аналітиками CoinGecko.

Додати криптовалюту до свого портфеля дуже просто – досить знайти дану монету у пошуку та натиснути на зірочку поряд з нею. У вашому портфелі ви зможете вказати, скільки та за якою ціною була придбана або продана дана монета, і після цього ви побачите повну аналітику у вашому портфелі. Приклад екрану портфеля можна побачити на рисунку 1.5.

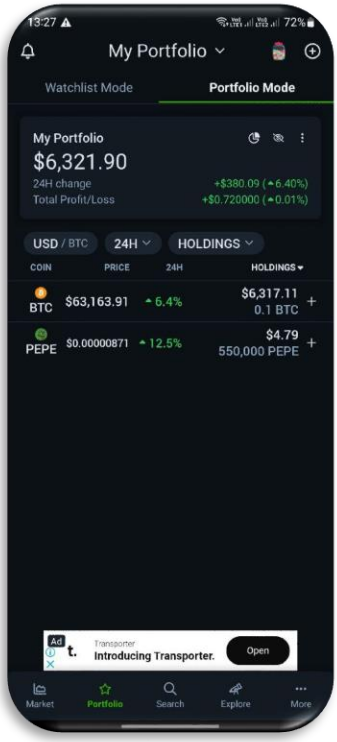

Рисунок 1.5 - Екран портфеля користувача в CoinGecko

У додатку також присутній список спотових, ф’ючерсних та децентралізованих бірж та основна інформація про них, така як торгові обороти та

відвідуваність користувачами. CoinGecko має власну систему оцінки кожної біржі від 1 до 10.

Також присутня окрема вкладка для новин та корисної інформації. Це дуже важливий функціонал, оскільки звідти можна черпати інформацію, базуючись на основі якої можна приймати свої інвестиційні рішення.

Якщо говорити про підрозділи CoinGecko то крім додатку туди також входить магазин CoinGecko та їх власне API. Саме цим API ми скористаємося при створенні нашого додатку.

Якщо говорити про недоліки даного додатку, то це відсутність українського перекладу, а також велика кількість негативних відгуків. Даний ресурс часто критикують за просування шахрайських проектів, а також CoinGecko часто називають неточним у порівнянні з іншими додатками. Потрібно розуміти, що в сфері криптовалюти важко на початкових етапах визначити, чи проект є шахрайським, тому це питання завжди залишатиметься спірним.

Менш популярним, але дуже зручним додатком є Delta. Особливістю цього трекера є те, що в ньому присутні такі активи як акції, індекси та фонди, а не тільки криптовалюта. Цей додаток буде дуже корисним для тих, хто цікавиться та інвестує кошти в різні активи.

У безкоштовному плані додатку є весь необхідний функціонал для більшості людей, а інтерфейс додатку є надзвичайно зручним і містить дуже багато налаштувань.

Портфель цього додатку є найбільш зручним з усіх попередніх, оскільки дозволяє легко прив’язуватися до різних видів гаманців або брокерського рахунку і підтягувати транзакції в автоматичному режимі. Також є можливість додавати активи вручну. Розділи портфелю розроблені так, щоб ви легко могли перемикатися між переглядом різних видів активів. Інтерфейс портфеля в додатку зображений на рисунку 1.6.

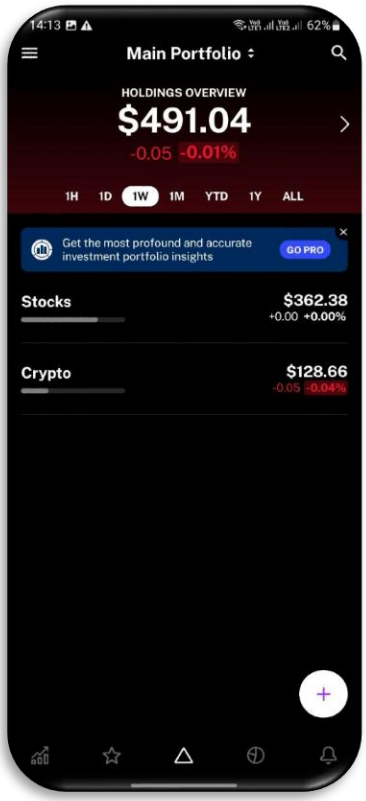

Рисунок 1.6 - Екран портфеля користувача в додатку Delta

За роки своєї роботи Delta заслужив репутацію серед користувачів і є найбільш універсальним додатком для інвестора. Функції додатку прості у використанні, проте варто зрозуміти, що якщо користувач передбачає можливість купувати активи з додатку криптотрекера, то Delta не підійде для таких операцій.

1.2 Проектування програмної системи

1.2.1 Вибір процесу розробки

Процес розробки програмного забезпечення — це структурований підхід до розробки програмного забезпечення для системи чи проекту, який іноді називають життєвим циклом розробки програмного забезпечення [4].

Загалом існує кілька моделей процесу, які відрізняються одна від одної:

a. Водоспадна (каскадна, послідовна)
b. Спіральна
c. Ітераційна

Водоспадна модель є традиційною і базується на послідовному виконанні всіх етапів розробки. На етапі формування вимог, кожна з них документується в ТЗ та повинна бути виконана протягом усього періоду розробки проекту. Приклад водоспадної моделі зображений на рисунку 1.7.

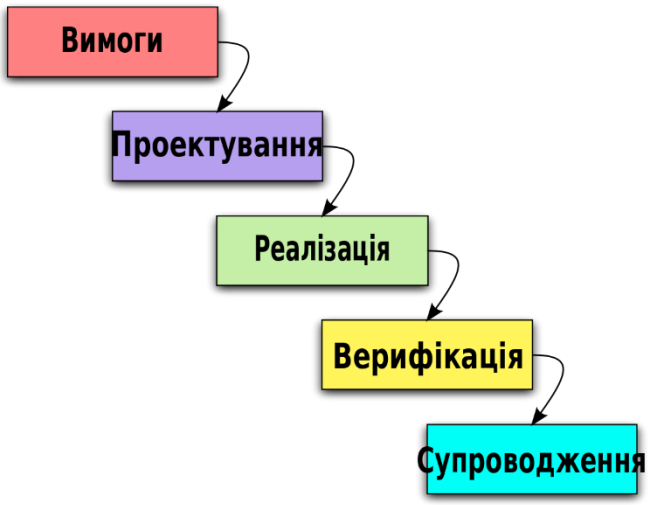

Рисунок 1.7 – Водоспадна модель процесу

Дана модель є досить зручною та нескладною у виконанні, проте потрібне чітке розуміння того, яким має бути кінцевий продукт. Нажаль, досить часто це неможливо зробити, оскільки правки можуть вноситися протягом реалізації проєкту. Це часто спричинено зміною бюджету компанії або зміною результатів після аналізу і тестування ніші.

Також необхідно пам’ятати, що часто при проектуванні ПЗ у команди недостатньо ресурсів для якісного введення документації, а це є надзвичайно важливо для даної моделі. В кінцевому результаті може статись така ситуація, що при зміні команди нові спеціалісти витрачатимуть ще більше часу через її брак і проект може закритись.

Якщо розглядати спіральну модель розробки, то можна з впевненістю сказати, що вона є інноваційною і досить гнучкою. Вона комбінує елементи як ітераційних, так і інкрементальних підходів.

Основною ідеєю цієї моделі є те, щоб кожна ітерація проекту пройшла через чотири основні фази: визначення цілей, аналіз та оцінка ризиків, розробка та тестування, і після цього планування наступних кроків.

Розробка програмної системи в даному випадку відбувається поетапно, і з кожним новим циклом додаються нові функції. Дана модель дозволяє поступово створювати кінцевий продукт і, на відміну від водоспадної моделі, дає можливість вносити зміни в проект під час розробки. Приклад спіральної моделі зображений на рисунку 1.8.

Рисунок 1.8 – Спіральна модель процесу

Така структура є дуже гнучкою і адаптивною до змін, і тому в цілому ця модель є ефективним інструментом для розробки складних проектів.

Серед недоліків цієї моделі є те, що вона вимагає багато часу для планування кожної ітерації, і тому потребує досвідченого керівника проекту.

Ітераційна модель розробки програмного забезпечення є дуже гнучкою, оскільки в її основі лежить послідовне повторення коротких циклів розробки і тестування продукту.

У даній моделі, на відміну від спіральної, акцент зроблений саме на ітеративному підході, де продукт розбивається на ітерації і постійно

вдосконалюється. У спіральній ж моделі основний акцент припадає на управління ризиками, і кожне обертання дозволяє вносити якісь корективи. Приклад ітераційної моделі зображений на рисунку 1.9.

Рисунок 1.9 – Ітераційна модель процесу

Для вибору моделі процесу розробки потрібно також визначитись з методологією розробки програмного забезпечення. Основні види цього - Scrum та Kanban.

Scrum - методологія, в якій кожен проект поділяється на ітерації. Кожна ітерація називається спринтом, який зазвичай триває від одного до кількох тижнів.

Цей метод додає гнучкості у плануванні за допомогою короткочасних спринтів. У кожного є своя чітко визначена роль у команді. За допомогою Product Owner пріоритети задач визначаються в беклозі для виконання development командою під час спринта.

Перед кожним спринтом проводяться спринт-зустрічі, де обговорюються його цілі, а щодня проводяться так звані дейлі зустрічі. В кінці кожного спринту відбувається ретроперспектива, де команда аналізує свої досягнення і проблеми. Графічне відображення Scrum методології зображено на рисунку 1.10.

Рисунок 1.10 – Методологія Scrum

Kanban - це менш структурований підхід, ніж Scrum. На дошці стани відображаються у стовпчиках, через які кожна задача проходить зліва направо [5].

Для кожного стовпчика задається межа Work in Progress. Цей ліміт показує, скільки задач може перебувати в певному стані одночасно. Якщо стовпчик заповнено, то ніхто більше не може переводити туди нові задачі, проте кожен з команди повинен допомогти закрити вже перетягнуті завдання. Графічне відображення прикладу Kanban методології зображено на рисунку 1.11.

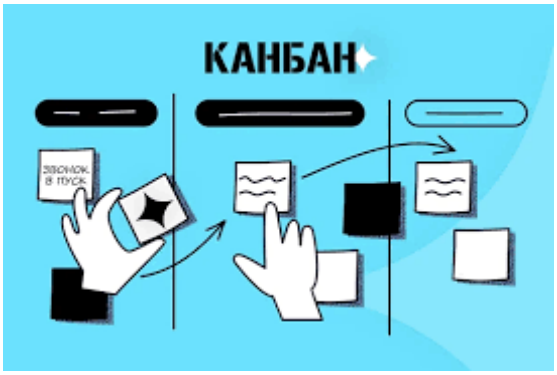

Рисунок 1.11 – Методологія Kanban

Серед основних переваг Scrum порівняно з Kanban є більша гнучкість і можливість вносити зміни через короткі спринти. Також цей метод передбачає жорстке планування ітерацій і обмеження на виконання завдань у межах спринту.

Таким чином, ми можемо зробити висновок, що методологія Scrum підійде для нас краще, оскільки ми маємо обмеження в часі та потребуємо цієї гнучкості.

Оскільки програмна система є невеликою, то для Scrum ітераційна модель розробки стане найкращим рішенням. Це сприятиме швидкому і гнучкому розвитку проекту, де ми зможемо отримувати більш швидкі результати.

1.2.2 Побудова схеми бази даних

Для збереження даних користувача ми будемо використовувати базу даних Cloud Firestore.

Cloud Firestore - це гнучка, масштабована, розміщена в хмарі NoSQL база даних, створена для розробки мобільних, веб-сайтів і серверів від Firebase і Google Cloud [6].

Ця база даних синхронізує ваші дані між клієнтськими додатками для прослуховування в реальному часі.

Якщо говорити про можливості Cloud Firestore, то варто сказати, що ця модель даних підтримує гнучкі ієрархічні структури даних. Ви можете зберігати свої дані у вигляді документів, які організовані в колекціях.

Ви можете створювати такі запити до бази даних, які дозволять отримувати конкретні документи або набір з документів, що відповідають певним фільтрам.

Серед переваг є те, що оновлення відбуваються в реальному часі, а дані, які активно використовуються додатком, кешуються, і це дає можливість користуватися додатком навіть в режимі офлайн.

Захист до читання та запису даних в БД здійснюється шляхом аутентифікації при вході в додаток. Графічний приклад зберігання даних в Firestore зображено на рисунку 1.12.

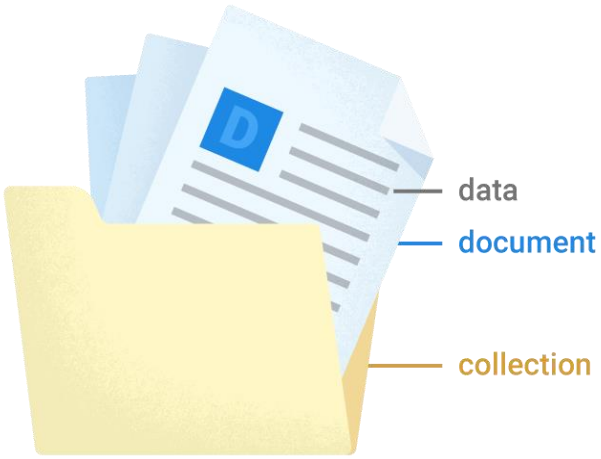

Рисунок 1.12 – Вигляд структури бази даних Firebase

При використанні моделі даних NoSQL Cloud Firestore ви зберігаєте дані у документах, які містять поля з власними значеннями. Кожен документ зберігається у певній колекції. Це допомагає створювати запити для отримання необхідних нам документів.

У документах ми можемо використовувати різноманітні типи даних, від простих чисел до складних об’єктів. Створюючи підколекції, ви будуєте ієрархічну структуру даних, яка легко масштабується.

В нашій БД ми будемо зберігати дані у двох колекціях. Перша колекція матиме назву “Users” і всередині ми зберігатимемо базову інформацію користувача при вході в додаток.

Враховуючи те, що в додатку буде реалізовано два види входу: через обліковий запис Google і за допомогою електроної пошти та пароля, наші поля даних можуть відрізнятись.

При реєстрації в додаток ми створюватимемо новий документ у колекції “Users” а ідентифікатор цього документу буде рівним Uid нашого користувача при вході.

Перший варіант входу, де користувач використовував свій обліковий запис Google, створює наступний документ, який зображений на рисунку 1.13.

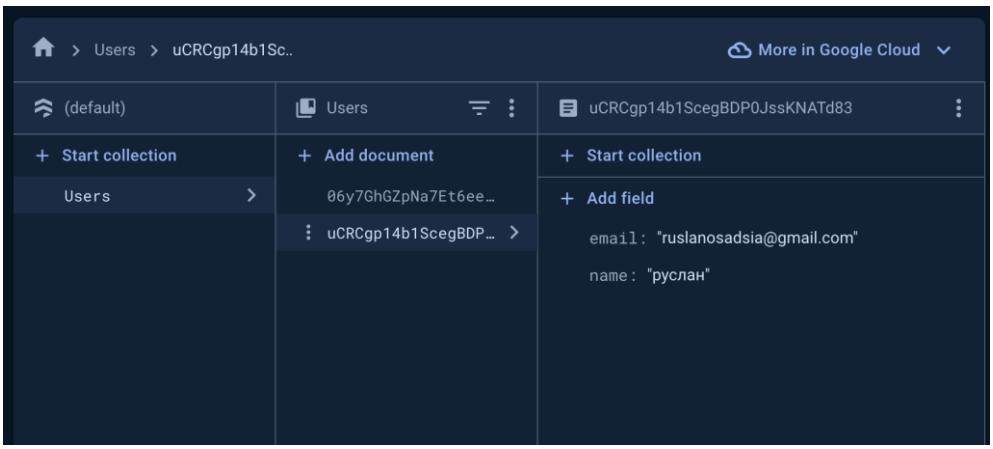

Рисунок 1.13 – Структура документу при першому варіанті аутентифікації

В документі зберігаються поля “email” з електроною поштою користувача та “name” з ім’ям, яке ми витягуватимемо з інформації Google.

При другому варіанті, коли користувач використовує електрону пошту і пароль для входу, буде створений наступний документ, який зображений на рисунку 1.14.

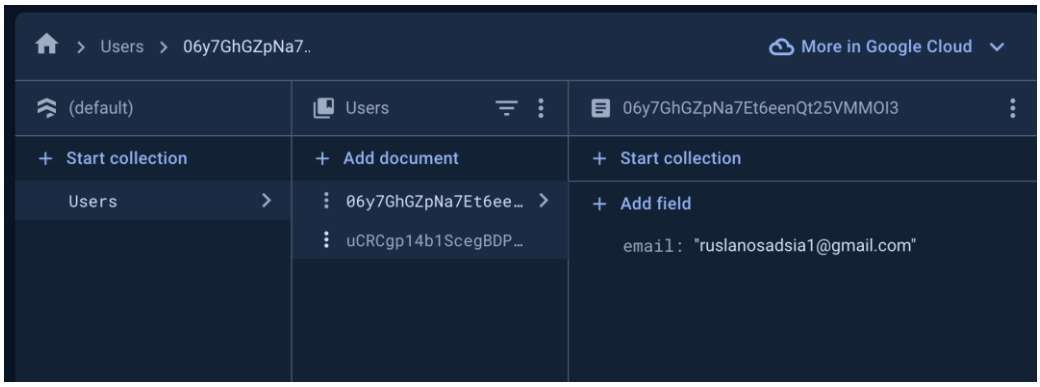

Рисунок 1.14 - Структура документу при другому варіанті аутентифікації

У даному випадку ми зберігатимемо тільки електрону пошту користувача в полі “email”. Ця колекція буде корисною при майбутньому розширенні функціоналу додатку.

Інша колекція, яка буде основною у додатку, виконуватиме роль для запису транзакцій користувача у його портфелі. Колекція буде створюватися динамічно при першій транзакції у портфелі, і її назва буде рівною Uid користувача. Приклад документу з транзакцією користувача зображено на рисунку 1.15.

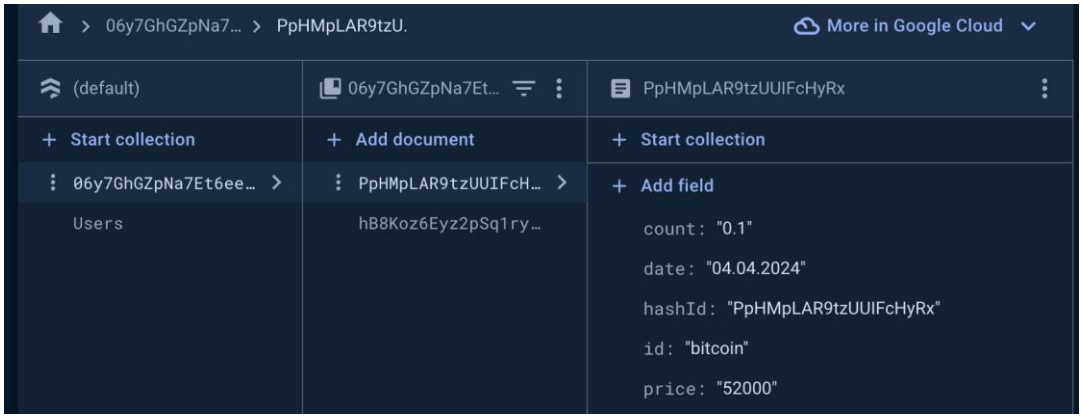

Рисунок 1.15 – Документ з даними про транзакцію в портфель

У колекції портфелю нашого користувача ми бачимо дві транзакції. У документі міститимуться такі поля, як: count, date, hashId, id, price.

Поле “count” показує те, який об’єм криптовалюти був куплений користувачем. У даному випадку це 0.1. Це значення буде корисним для розрахунків прибутків або збитків інвестора, а також для загальної суми портфеля.

Значення поля “date” відображатиме дату покупки. Ця інформація має чисто інформаційний характер для історії транзакцій.

Поле “hashId” відображає ідентифікатор документу транзакції, а поле “id” нам пригодиться для синхронізації з API для отримання даних про криптоактиви.

Останнє поле служить ціною, яка буде корисна для підрахунків в аналітиці портфеля інвестора.

Дана схема бази даних повністю задовольнятиме наші потреби. За допомогою Uid користувача ми зможемо створити запит для отримання списку зі всіх монет користувача, а при потребі в фільтруванні зможемо створювати більш складні запити до бази даних.

У розділі “Rules” потрібно налаштувати контроль доступу до наших колекцій. Приклад наших записів зображено на рисунку 1.16.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
    match /{uid} {
      allow get, write: if request.auth != null && request.auth.uid == uid;
    }
  }
}
```

Рисунок 1.16 – Правила запису та читання з бази даних

Рядок “match /databases/{database}/documents” визначає шлях до документів у базі Firestore. Наступна важлива частина це рядок “allow read, write: if request.auth != null;”, який встановлює правило за замовчуванням, про можливість читання та запис тільки для аутентифікованих користувачів.

Для колекцій з портфоліо користувачів застосовуємо правило, яке дозволяє читати або писати тільки у колекції, де Id рівне Uid користувача.

Правила Firestore Security Rules є дуже важливими для забезпечення безпеки та ефективності нашого додатку.

1.2.3 Побудова UML діаграми класів

UML (Unified Modeling Language) є стандартною мовою моделювання та використовується для візуалізації, побудови, а також документування програмних систем [7].

Основними типами діаграм є структурні та поведінкові. Структурні можуть містити діаграми класів, компонентів та об’єктів. Поведінкові ж діаграми відображають випадки використання, активностей або послідовностей.

Діаграма класів дозволяє представити класи, атрибути, методи та взаємозв’язки між ними, і кожна з них складається зі своїх компонентів.

Ім’я класу повинно бути унікальним у межах одного пакету і вказується воно в самій верхній секції прямокутника. При позначенні абстрактного класу використовується курсив, а для простого класу - напівжирний шрифт.

Атрибутом класу називається характеристика, яка описує значення, які можуть приймати об’єкти цього класу. Ім’я атрибута є єдиним обов’язковим елементом позначення та починається з малої літери.

Методи класу записуються в третій секції. Їх ім’я повинно бути унікальним у межах даного класу. Приклад оформлення класу на діаграмі зображено на рисунку 1.17.

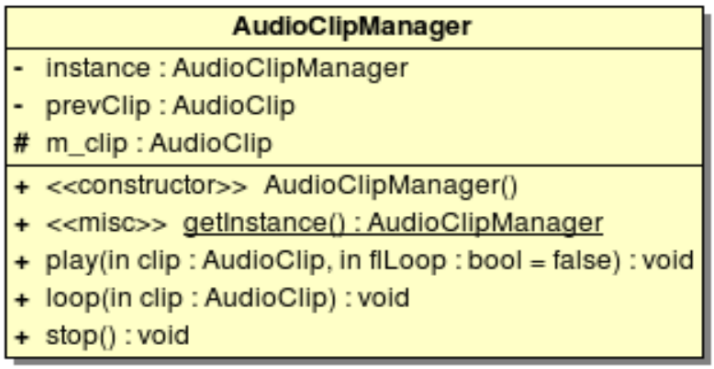

Рисунок 1.17 – Оформлення класу на діаграмі класів

Часто немає потреби детально показувати класи, тому зображення класу може складатися лише з двох частин або навіть однієї - імені класу. Приклад спрощеного зображення класу показано на рисунку 1.18.

Рисунок 1.18 – Спрощене оформлення класу на діаграмі класів

Серед основних класів є три класи Activity, які стануть основою нашого додатку. Діаграма цих класів зображена на рисунку 1.19.

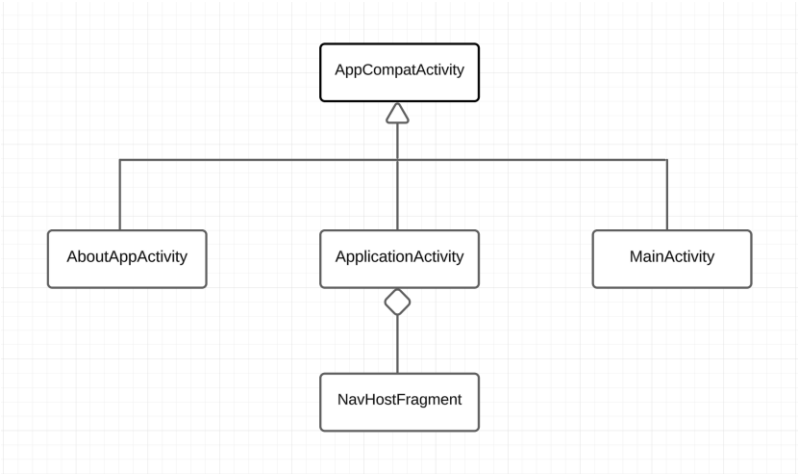

Рисунок 1.19 – Діаграма класів Activity

Клас MainActivity використовується як основа для фрагментів при реєстрації та вході користувача, а у випадку успішного входу ми переходимо на Application-Activity, який має відношення агрегації з класом навігації для наших екранів NavHostFragment.

Активність AboutAppActivity використовуватиметься для відображення інформації про додаток та контактів.

Також основними класами у нашому додатку будуть ті, які відповідають за роботу з API та аутентифікацію користувача. Діаграма цих класів зображена на рисунку 1.20.

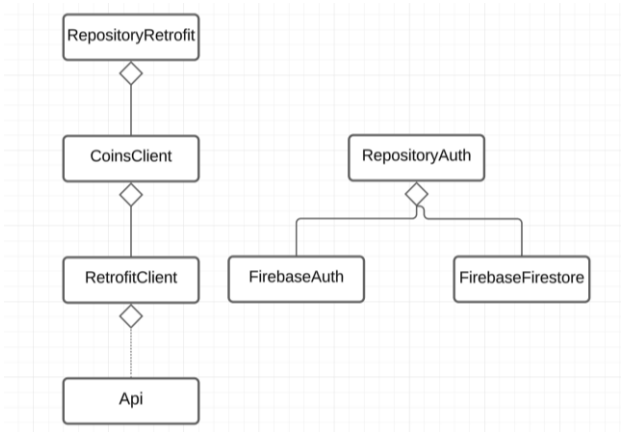

Рисунок 1.20 – Діаграма класів для роботи з API та базою даних

Клас RepositoryAuth у даному прикладі відповідає за вхід та реєстрацію в додатку, а також за запис та отримання наших транзакцій з бази даних Firebase Firestore. Класи FirebaseAuth та FirebaseFirestore є публічними бібліотеки Firebase та надають методи для роботи з нею.

Інтерфейс Api містить наші GET методи для роботи з API, а клас RetrofitClient є, так би мовити, прослойкою, яка за допомогою Retrofit білдера ініціалізує наш інтерфейс Api.

Клас CoinsClient описує методи для роботи з API через клас RetrofitClient, а RepositoryRetrofit вже використовує ці методи.

Також варто відмітити класи моделі, які будуть використовуватись для об’єктів наших криптовалют. Діаграма зображена на рисунку 1.21.

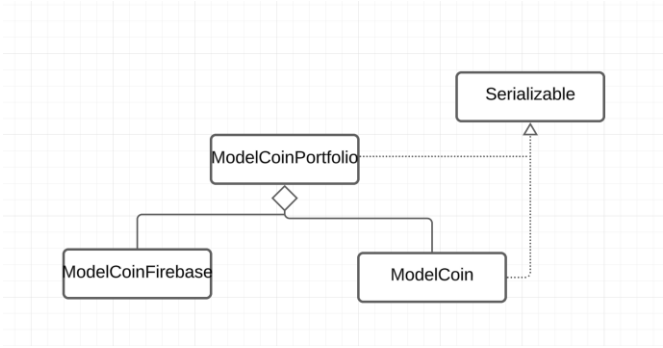

Рисунок 1.21 – Діаграма моделей криптовалют

Класи ModelCoin та ModelCoinPortfolio наслідують інтерфейс маркет Serializable для можливості серіалізації для коректного збереження.

Клас CoinModelPortfolio містить дані, які були отримані з API а також і з бази даних користувача, для відображення в портфелі.

1.2.4 Моделювання архітектури системи

Архітектура системи - це план його організації та структури, який показує, як різні компоненти додатка будуть взаємодіяти один з одним.

Додаток складається з різних компонентів, таких як інтерфейс, модель даних і бізнес-логіка.

Останніми роками в Android розробці досить популярними є три архітектурні патерни: MVC, MVVM та MVP. Далі розглянемо кожен з них.

MVC (Model-View-Controller) - патерн, який розділяє додаток на три компоненти. Взаємодія між компонентами виглядає так: коли користувач взаємодіє з View то View передає ці зміни у наш Controller і там наші дії обробляються та взаємодіють з шаром моделі (Model). Модель передає оновлення в наше View, де користувач бачить зміни. Графічний приклад даного патерну зображений на рисунку 1.22.

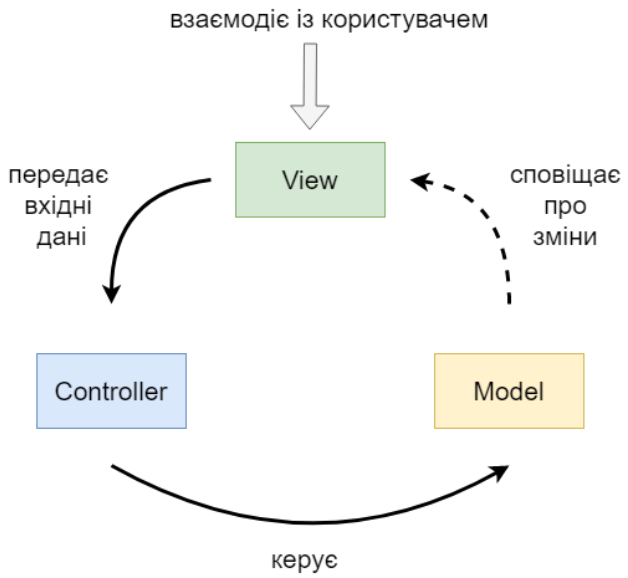

Рисунок 1.22 – Архітектурний патерн MVC

Плюсом даного патерна є те, що він є досить простим та дозволяє швидко розробляти додатки невеликого розміру. Проте, якщо говорити про його недоліки, то він є досить складним у тестуванні та при збільшенні розміру додатку важко масштабується.

MVVM (Model-View-ViewModel) - засновується на розділенні інтерфейсу користувача та бізнес-логіки з використанням прослойки у вигляді ViewModel. Саме вона замінює Controller, який є в MVC. Приклад взаємодії компонентів у патерні зображено на рисунку 1.23.

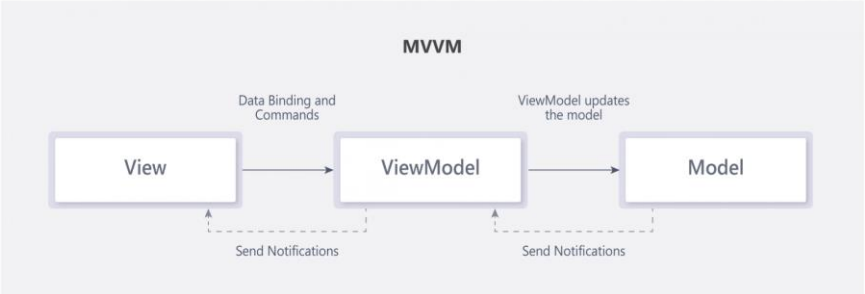

Рисунок 1.23 – Архітектурний патерн MVVM

Якщо контролер просто передає запити, які сформовані поданням до бази, то модель зображення постійно “питає”, чи не змінилося щось у View, що потрібно повідомити моделі і назад.

Даний патерн добре тестується та чітко розділяє обов’язки різних компонентів, проте може бути нелегким у розумінні новичками.

MVP (Model-View-Presenter) - дуже схожий на MVC, проте з активнішою роллю презентера для керування взаємодією між моделлю та представленням. Приклад взаємодії компонентів у патерні зображено на рисунку 1.24.

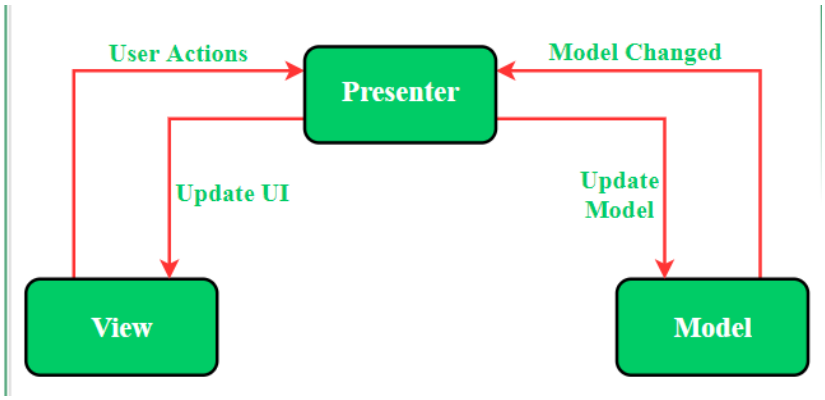

Рисунок 1.24 – Архітектурний патерн MVP

При використанні цього патерну значно збільшується кількість класів, що може ускладнити розробку та мсаштабування проекту, але незважаючи на це, даний патерн залишається одним з найпопулярніших до цього часу.

Для невеликих проектів на Android , патерн MVVM буде найкращим вибором. Ми можемо знайти безліч прикладів коду та реалізацій різних функціоналів саме за допомогою MVVM патерну.

У нашому випадку будуть створені спеціальні класи ViewModel, які будуть повідомляти інтерфейс про зміни і в той час взаємодіяти з класами репозиторіями (Model).

Всі класи будуть групуватися в пакети, такі як activities, adapters, fragments, repository, viewmodel та інші, для забезпечення зручності в навігації проектом.

Враховуючи невеликий розмір додатку, всі пакети будуть знаходитися в одному модулі, оскільки розбивка на різні модулі вимагає багато часу на початку і не рекомендується для маленьких проектів. Також розбивка на модулі призводить до складності в правильному налаштуванні залежностей в проекті. Найбільш поширеним рішенням є Clean Architecture, проте і цей підхід не варто розглядати з огляду на розмір програми.

1.3 Конструювання програмної системи

1.3.1 Вибір мови та середовища розробки

Вибір мови програмування для розробки є дуже важливим рішенням, оскільки від цього залежатиме як швидкість розробки системи, так і її безпека.

Якщо говорити про можливі підходи до створення Android додатків на даний момент, то це може бути як кросплатформена, так і нативна розробка.

Існує кілька способів, які дозволяють створювати кросплатформені додатки. Серед найпопулярніших є Flutter, React Native і Xamarin. Ці фреймворки мають як і свої переваги перед нативною розробкою, так і недоліки.

Серед переваг бібліотеки Flutter можна виділити відносно швидку розробку порівняно з іншими кросплатформеними рішеннями, а також однаковий вигляд та

поведінку, що досягається наявністю власного рендерингу віджетів. З недоліків можна виділити великий розмір кінцевої версії додатку.

Технологія React Native, з свого боку, славиться своєю шорокою спільнотою та підтримкою. Майже будь-який функціонал, який можна написати, вже був реалізований за допомогою цього фреймворку. Однак, серед недоліків можна відзначити можливе зниження продуктивності через обробку нативних функцій за допомогою JavaScript.

Серед усіх інших платформ, Xamarin не є настільки популярним для розробки. Додаток у даному випадку будується на основі середовища .Net. Перевагою є доступ до нативних API та функцій, а також велика підтримка від Microsoft. Недоліком є обмежена підтримка сторонніх бібліотек.

Основними перевагами кросплатформної розробки є її швидкість, оскільки використовується спільний код, а також незалежність від платформи. Очевидними недоліками є те, що часто те, що не працює на одній платформі, на іншій прекрасно працює, а тому потрібно витратити багато часу на тестування та виправлення помилок, пов’язаних з особливостями тої чи іншої платформи. Тому кросплатформний підхід створення додатку нам не підійде.

Нативні додатки - це додатки, розроблені для конкретної платформи (IOS або Android), з урахуванням специфіки даної платформи та доступом до її ресурсів [8].

При створенні нативного Android додатку ми отримуємо вищу швидкість роботи, на відміну від гібридних програм. Також інтерфейс виглядатиме так, як звикли його бачити користувачі на тій чи іншій платформі. Приклад такої різниці зображено на рисунку 1.25.

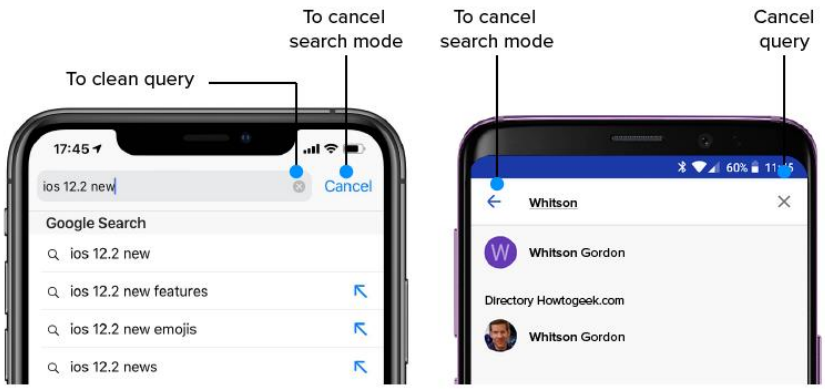

Рисунок 1.25 – Різниця в дизайні між платформами IOS та Android

Також, у нативній розробці ми маємо повний доступ до всіх API та функцій платформи, що дозволяє нам створювати додатки з високим рівнем функціональності.

При такому підході перед нами постає вибір, яку нам мову обрати - Java чи Kotlin. Ці дві мови є надзвичайно схожими за своїм синтаксисом, а з 2017 року Google рекомендує Kotlin як основну мову для створення Android додатків. Проте, не все так однозначно щодо цього вибору.

Java — об'єктно-орієнтована мова програмування, яка була випущена в 1995 році компанією Sun Microsystems як основний компонент платформи Java. Починаючи з 2009 року мову підтримує компанія «Oracle», яка придбала «Sun Microsystems» [9].

Kotlin — це статично типізована мова програмування, яка працює поверх JVM і розробляється компанією JetBrains. Розробники створили лаконічнішу та типо-безпечнішу мову, ніж Java. Мова розробляється з 2010 року, а публічно представлена в липні 2011 [10].

Між собою ці дві мови дуже подібні, проте в Kotlin реалізовано багато функціоналу, який дозволяє писати код швидше. Приклад синтаксису двох мов зображено на рисунку 1.26.

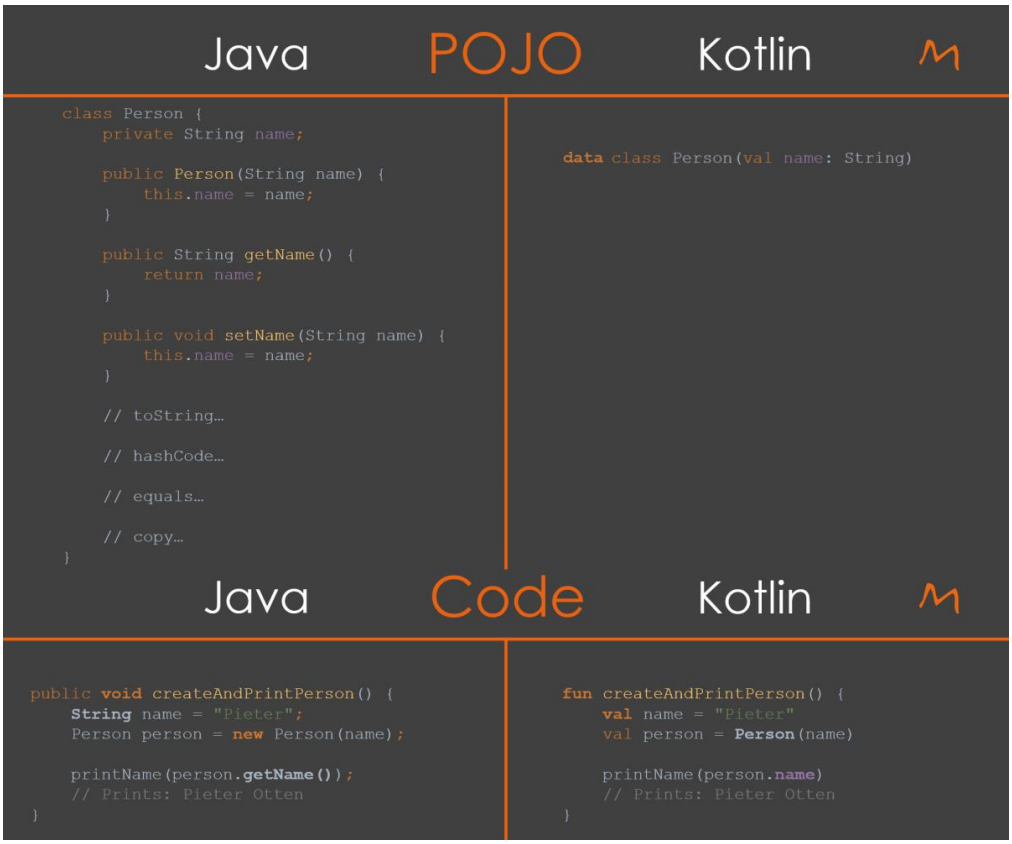

Рисунок 1.26 – Синтаксис Java та Kotlin

Для початківців Java є більш зрозумілою і подібною до таких мов, як C та C++, оскільки має багато запозичень від них. Також є багато ресурсів та документації, доступних для Java, що дозволяє швидко знаходити відповіді на свої запитання. Крім того, дана мова є більш надійною та сумісною з великою кількістю бібліотек, тому стане найкращим рішенням для нашого додатку.

Найкращим і майже не єдиним варіантом при виборі середовища розробки є Android Studio.

Android Studio – офіційне середовище розробки для Android-додатків, що надає безліч інструментів і функцій. Це включає в себе редактор коду з автодоповненням і підсвічуванням синтаксису, візуальний редактор інтерфейсів, налагоджувач, інструменти профілювання та емулятор Android. Вона також інтегрована з Gradle для управління залежностями, дає змогу експортувати та підписувати додатки у форматі APK [11].

Дане середовище дає можливість створювати додатки не тільки для смартфонів, але й і для годиників, окулярів та телевізорів на Android. З його

допомогою ми можемо розробляти та тестувати програми будь-якої складності та розміру. Інтерфейс середовища зображено на рисунку 1.27.

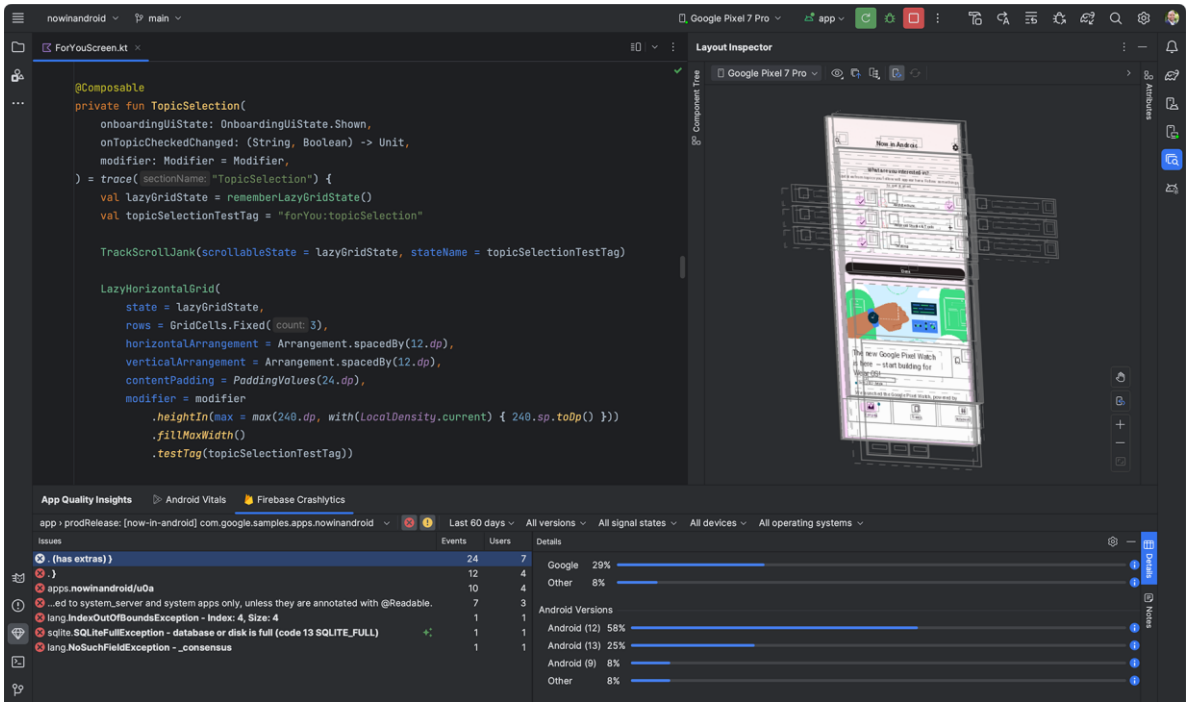

Рисунок 1.27 – Інтерфейс Android Studio

Серед основних функцій цього інтегрованого середовища є: візуальний редактор інтерфейсу користувача, емулятори Android пристроїв, підтримка як Java, так і Kotlin, плагіни та розширення, а також підтримка систем контролю версій. Всі ці можливості є важливими для зручного створення додатків будь-якої складності.

1.3.2 Опис програмної реалізації

Розпочнемо конструювання нашого додатку з розбиття модуля на пакети для комфорту при розробці, а також для слідування архітектурному патерну MVVM, який ми вибрали раніше. Вигляд структури нашого додатку зображений на рисунку 1.28.

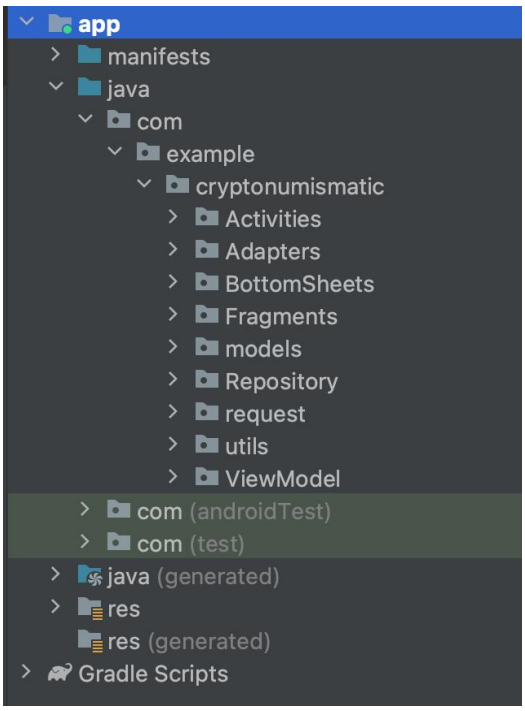

Рисунок 1.28 – Структура модуля нашої програми

Після створення структури нашого проекту потрібно додати імплементації бібліотек, які ми використовуватимемо при розробці. До цього списку входять такі інструменти, як Glide, який допомагатиме нам у роботі з зображеннями, Retrofit - для роботи з нашим API та інші.

Першим етапом розробки додатку стане додавання логіки входу та реєстрації. Всього ця частина функціоналу складатиметься з трьох екранів.
Перший екран буде стартовим і зустрічатиме користувача при вході в додаток. На ньому можна буде вибрати, чи користувач хоче зареєструватись, чи увійти, якщо у нього вже є обліковий запис у нашому додатку. Також буде можливість увійти за допомогою облікового запису Google.

Розпочнемо створення даної логіки з побудови інтерфейсу та створення класу Fragment у нашому коді. Дуже важливо створити максимально інтуїтивний дизайн для максимальної зручності користувача. Основою нашого xml файлу з інтерфейсом стане елемент ConstraintLayout, який дозволить компонувати наші елементи так, як ми забажаємо та є найбільш оптимізованим. Інтерфейс стартового екрану зображено на рисунку 1.29.

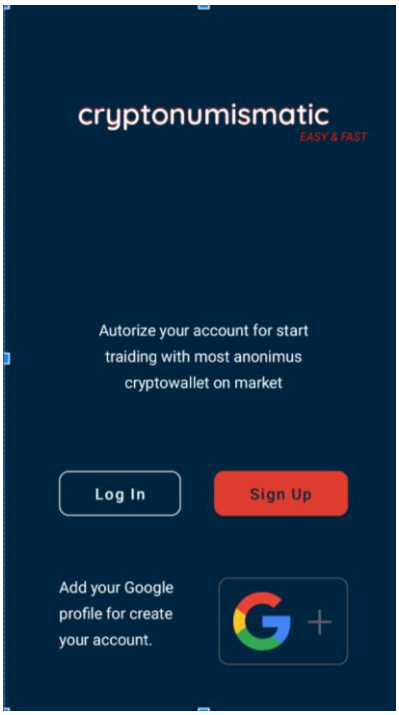

Рисунок 1.29 – Інтерфейс стартового екрану додатку

Також було додано клас ViewModelStartApp. У нашому фрагменті, використовуючи патерн спостерігач, ми підписуємося на зміну MutableLiveData, яка повідомляє, чи був здійснений вхід в додаток.

При натисканні на вхід за допомогою облікового запису Google ми запускаємо лаунчер з параметром GoogleSignInClient, який був створений на основі нашого Web client id. GoogleSignInClient - це клієнт для взаємодії з Google Sign In API [12].

Після цього ми побачимо спеціальний діалог з вибором облікового запису, приклад якого зображено на рисунку 1.30.

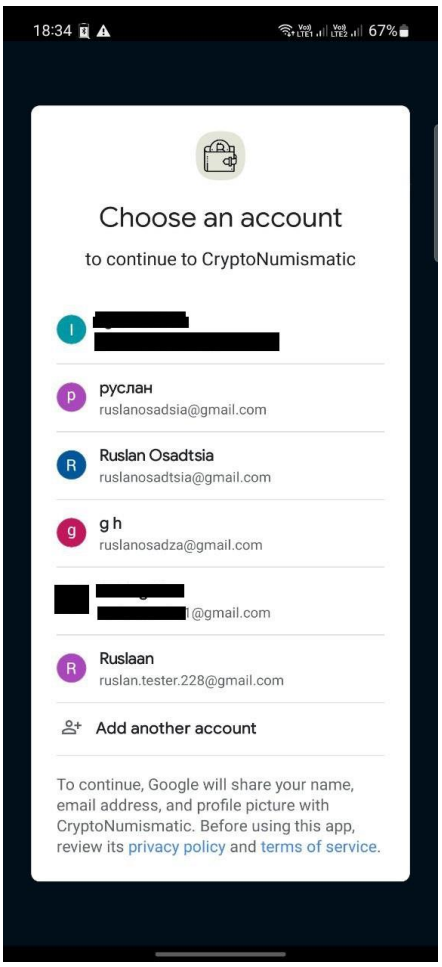

Рисунок 1.30 – Діалог вибору облікового запису Google

Коли користувач вибирає обліковий запис, ми отримуємо токен, а потім виконуємо вхід за допомогою Firebase методу signInWithCredential.

Взаємодія з методами Firebase відбувається не напряму у наших класах ViewModel, а через спеціальний клас репозиторій RepositoryAuth. Таким чином, нам не потрібно повторювати той самий код у класах ViewModel, а ми матимемо спільний репозиторій і дотримаємося ще одного принципу, під назвою DRY.

Don't repeat yourself - це принцип розробки програмного забезпечення, який вчить уникати дублювання коду, абстрагуючись від загальних речей і розміщуючи їх в одному місці [13].

При натисканні користувачем кнопки “Login” ми відкриваємо наступний екран для можливості входу за допомогою логіну та паролю, а при виборі “Sign up” переходимо на екран реєстрації.

Наступним кроком переходимо до екрану реєстрації користувача. При створенні інтерфейсу дотримуємося всіх попередньо згаданих принципів. Вигляд екрану реєстрації зображено на рисунку 1.31.

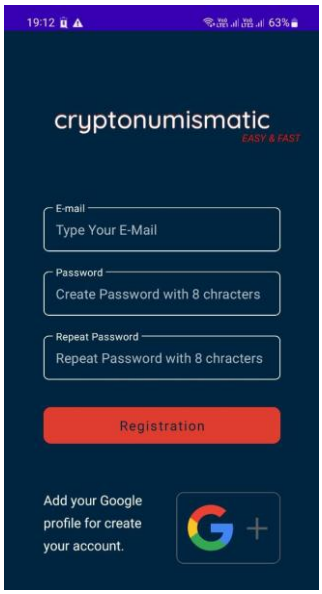

Рисунок 1.31 – Екран реєстрації користувача

На кожному полі додаємо валідацію при введенні. Перше поле повинно мати формат електронної пошти, а пароль повинен бути довшим ніж вісім символів. При невалідному введенні поля інтерфейс надає нам підказку, приклад якої зображено на рисунку 1.32.

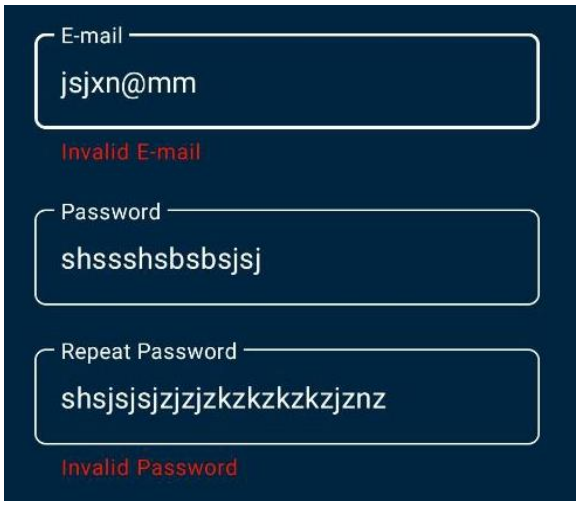

Рисунок 1.32 – Вигляд підказок при валідації

При натисканні на кнопку реєстрації ми викликаємо метод Firebase createUserWithEmailAndPassword, який створює новий обліковий запис.

Останнім екраном у навігації входу у додаток є можливість увійти за допомогою логіну та паролю. Інтерфейс зображено на рисунку 1.33.

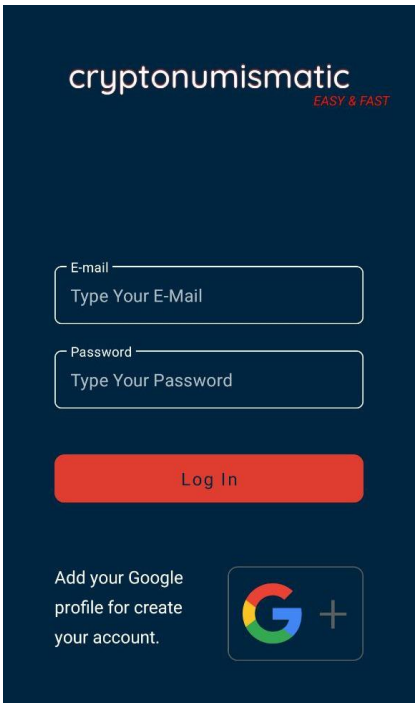

Рисунок 1.33 – Інтерфейс входу за допомогою логіну та паролю

Після успішного виконання методу для входу або реєстрації на будь-якому з попередніх екранів, наша змінна MutableLiveData в класі репозиторію отримує нові дані користувача. Оскільки ми підписалися на цю змінну в класі ViewModel, то дані передаються по ципочці до класу фрагмента. У фрагменті ми запускаємо ApplicationActivity, яке містить основну навігацію нашого додатку.

Основний функціонал нашого додатка складається з трьох екранів та двох діалогів. Першим екраном, який зустрічатиме користувача буде список криптовалют.

Цей екран складатиметься зі списку NFT токенів, які зараз у тренді, та зі списку криптовалют, який ми активуємо при друці в полі пошуку.

Для того, щоб додаток завжди показував найактуальнішу інформацію ми повинні оновлювати списки з певною періодичністю. Для списку NFT токенів це

оновлення буде відбуватись кожні 7.5 секунди, оскільки цей запит не є ресурсомістким. Для списку зі всіма монетами цей період становитиме 25 секунд.

Для запитів до нашого API створимо клас CoinsClient, який матиме такі методи, як: запит для отримання топ NFT, запит для отримання всіх монет та запит для отримання списку монет по ідентифікатору. Останній пригодиться нам для логіки нашого портфеля. Вигляд головного екрану при вході в додаток зображено на рисунку 1.34.

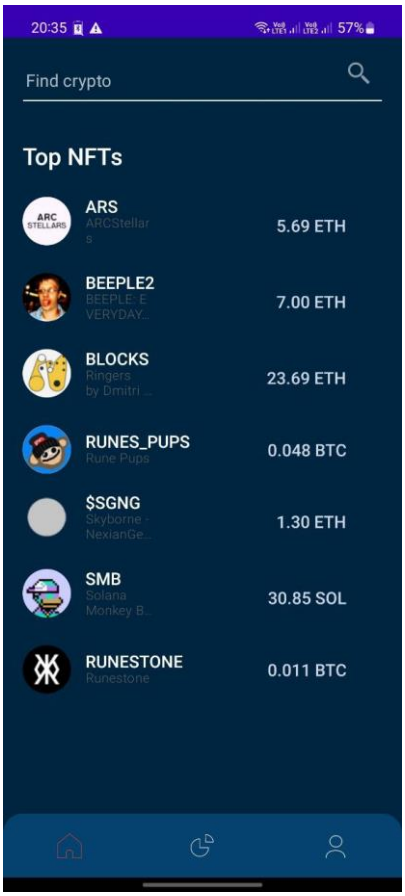

Рисунок 1.34 – Стартовий стан головного екрану

Фільтрація списку всіх монет відбувається локально на пристрої користувача без відпрвки додаткових запитів до API. Перевагою даного методу є швидкість оновлення нашого списку.

При введені назви криптовалюти ми бачимо список всіх тих, які містять в своєму імені таку саму комбінацію символів. Приклад такого стану додатку зображено на рисунку 1.35.

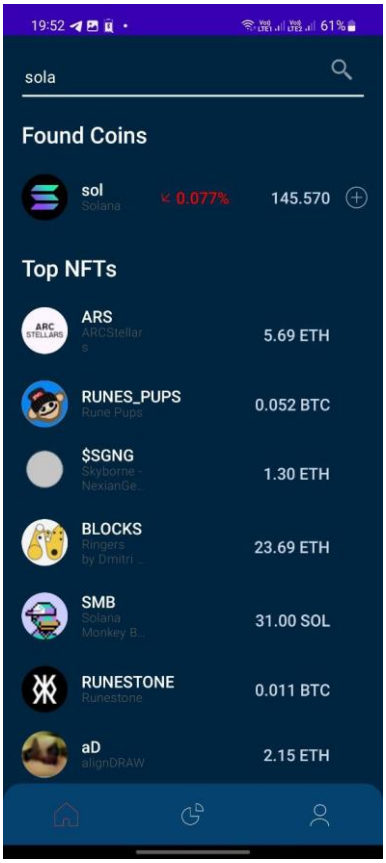

Рисунок 1.35 – Інтерфейс головного екрану при пошуку токенів

На кожному елементі криптовалюти ми бачимо такі дані, як фото, назва, ідентифікатор, ціна в даний момент, та зміна в ціні за останні 24 години. Це буде корисно користувачам, щоб швидко аналізувати загальний тренд ринку.

Ціни у списку криптовалют відображаються в такому стейблкоіні як Tether. Tether — запущені в 2014 році токени, які стали піонерами моделі стейблкоїнів і є найбільш популярними. Токени Tether пропонують стабільність і простоту фіатних валют у поєднанні з інноваційною природою технології блокчейн, представляючи ідеальне поєднання обох світів. Монета відрізняється високою стабільністю та низькою волатильністю, її курс завжди залишається в межах співвідношення 1:1 з доларом США [14].

Ціну NFT прийнято показувати відносно інших криптовалют, таких як BTC, ETH або SOL.

Для додавання вибраної криптовалюти в портфель наш користувач повинен натиснути на плюсик. Інтерфейс діалогу з додаванням токену у нашому додатку зображено на рисунку 1.36.

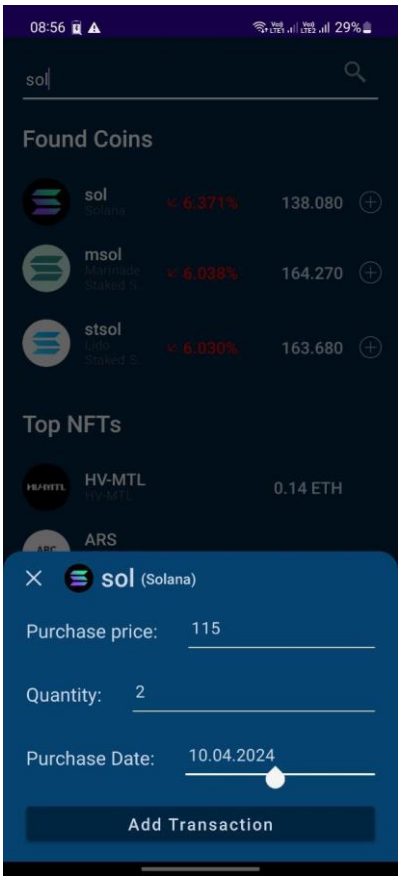

Рисунок 1.36 – Діалог додавання токену в портфель

У даному діалоговому вікні ми дозволяємо користувачу ввести такі дані, як ціну купівлі, кількість та дату проведення транзакції. Також тут присутня валідація всіх полів щодо обов’язковості заповнення.

Після натискання кнопки для додавання транзакції, всі дані про криптовалюту та введені значення користувачем передаються у наш репозиторій, де відбувається створення нового документу в Firebase Firestore із нашою транзакцією.

Для перегляду всіх транзакцій потрібно додати екран нашого портфелю. На ньому мають відображатись всі транзакції та детальна інформація по кожній із них, а також загальна інформація про портфель.

Спочатку на даному екрані ми робимо запит для отримання всіх документів транзакцій з нашої бази даних. Після цього зі списком ідентифікаторів всіх монет ми робимо запит до нашого API для отримання повної та актуальної інформації про токен. Далі ми поєднуємо ці дві моделі і відображаємо їх у списку. Інтерфейс екрану портфелю зображений на рисунку 1.37.

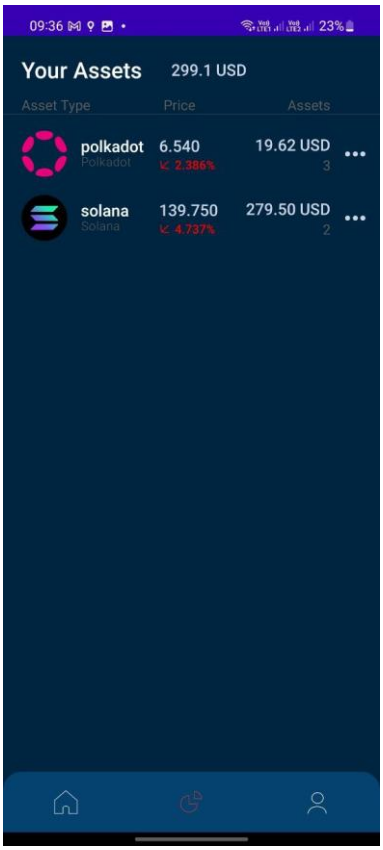

Рисунок 1.37 – Портфель користувача

У верхній частині ми відображаємо загальну вартість портфелю, а у кожному елементі списку - інформацію, що стосується даної монети. Серед цієї інформації - кількість монет, актуальна вартість в портфелі та зміни в ціні за останні 24 години.

Для того, щоб користувач мав змогу переглянути детальну інформацію про ту чи іншу транзакцію у своєму портфелі, додамо спеціальне діалогове вікно, яке можна відкрити натиснувши на значок трикрапки. Інтерфейс даного діалогу зображений на рисунку 1.38.

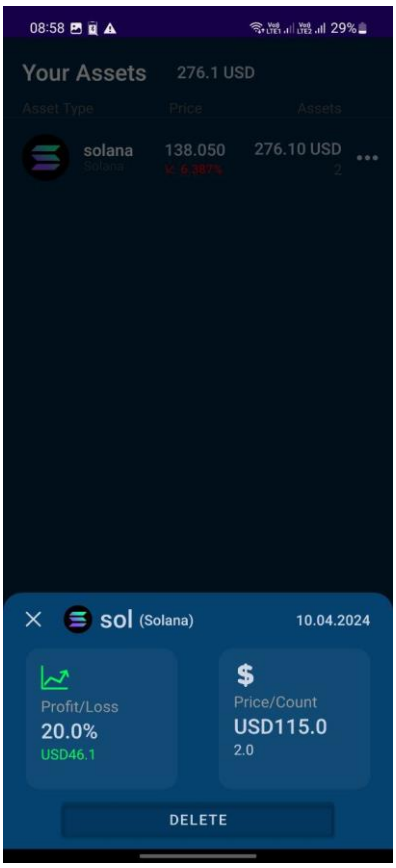

Рисунок 1.38 – Діалог з деталями транзакції

На цьому екрані відображаємо ціну, за яку був куплений даний актив, кількість, дату транзакції та загальний збиток чи прибуток у доларах та відсотках. Ці дані є надзвичайно важливими і можуть принести значну користь нашому потенційному користувачу.

Дуже важливо дати можливість видалення транзакції, оскільки вона може бути більше неактуальною або інвестор допустив помилку при її створенні. При видаленні транзакції знаходимо її у нашій базі даних по ідентифікатору та очищаємо.

Функціонал, який був доданий на даний екран, цілком достатній для середньостатистичного інвестора-початківця і зможе значно полегшити йому життя при ведені бухгалтерії своїх активів.

Останнім ераном у нас буде профіль користувача. У ньому буде можливість здійснити вихід з облікового запису або відкрити екран з контактами та інформацією про додаток. Інтерфейс даного екрану виглядає так, як показано на рисунку 1.39.

Рисунок 1.39 – Інтерфейс екрану профілю користувача

При натисканні на кнопку “Sign Out” відбувається виклик Firebase методу для виходу з облікового запису, і програма переводить користувача на екран для входу. Ця логіка дає можливість для створення декількох портфелей, які будуть пов’язані з різними обліковими записами. При натисканні на кнопку “About Application” відкриватиметься окремий екран, інтерфейс якого зображено на рисунку 1.40.

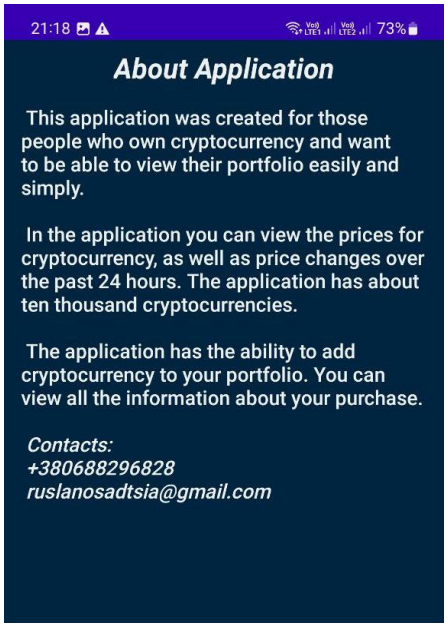

Рисунок 1.40 – Екран “Про додаток”

На цьому екрані користувач може прочитати інформацію про даний додаток, а також отримати контакти, за якими він може звернутись у разі виникнення проблем чи будь-яких питань.

2 ТЕСТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

2.1 План тестування

Тестування програмної системи — це тип тестування програмного забезпечення, який оцінює загальну функціональність і продуктивність повного та повністю інтегрованого програмного рішення. Він перевіряє, чи відповідає система зазначеним вимогам і чи придатна вона для доставки кінцевим користувачам [15].

Основною метою тестування програмної системи є перевірка її функціональності. Потрібно переконатись, що всі функції працюють відповідно до вимог.

Не менш важливим є перевірка додатку на надійність та безпеку, оскільки для користувачів дуже важливо знати, що третя сторона не зможе отримати доступ до їх даних.

Тестування проводиться для всього функціоналу додатку, оскільки тільки так можна зробити правильні висновки.

Вибираючи підхід і вид тестування, звернемо увагу саме на функціональне та тестування безпеки.

Функціональне тестування — це тип тестування, метою якого є встановлення того, чи кожна функція програми працює відповідно до вимог програмного забезпечення. Кожна функція порівнюється з відповідною вимогою, щоб переконатися, що її результат відповідає очікуванням кінцевого користувача [16].

Тестування безпеки ж відноситься до нефункціонального тестування, а його мета - зробити програмну систему зручною та безпечною для користувача.

Важливим кроком є продумування тестових сценаріїв, оскільки це допоможе максимально якісно і систематично протестувати кожен рівень програмної системи.

Список основних тестових сценаріїв:

- Вхід за допомогою облікового запису Google:

Опис: При спробі увійти в додаток з використанням облікового запису Google користувач потрапляє на головний екран додатку під його обліковим записом.

Передумови: Програма встановлена та запущена на телефоні.

Кроки виконання: На будь-якому з екранів для входу в додаток натиснути на логотип Google. Після цього на діалозі вибору облікового запису натиснути на той, який буде використовуватись.

Очікуваний результат: Додаток переходить з інтерфейсу для реєстрації або входу на головний екран.

- Вхід за допомогою логіну та паролю:

  Опис: При спробі увійти в додаток з використанням логіну та паролю користувач потрапляє на головний екран додатку під своїм обліковим записом.

  Передумови: Програма встановлена та запущена на телефоні. Користувач створив обліковий запис у додатку.

  Кроки виконання: На екрані для входу в додаток за допомогою логіну та паролю ввести дані для для входу та натиснути кнопку для входу.

  Очікуваний результат: Додаток переходить з інтерфейсу для входу на головний екран.

- Реєстрація облікового запису:

  Опис: При спробі реєстрації за допомогою логіну та паролю користувач потрапляє на головний екран додатку під новим обліковим записом.

  Передумови: Програма встановлена та запущена на телефоні.

  Кроки виконання: На екрані для реєстрації в додаток за допомогою логіну та паролю ввести дані та натиснути кнопку.

  Очікуваний результат: Додаток переходить з інтерфейсу для реєстрації на головний екран.

- Пошук токенів за назвою:

  Опис: Коли користувач вводить символи у поле пошуку, він отримує список всіх монет, ім’я яких містить такий самий набір символів.

  Передумови: Програма встановлена та запущена на телефоні. Користувач увійшов у свій обліковий запис.

  Кроки виконання: На головному екрані у полі для пошуку вводимо набір символів, який міститься в назві популярних криптовалют.

  Очікуваний результат: Отримуємо актуальний список токенів, відфільтрованих згідно з нашим алгоритмом пошуку.

- Додавання нового токену до портфеля:

  Опис: Коли користувач додає новий токен до свого портфеля, він відображається як нова транзакція разом з усіма даними у списку.

  Передумови: Програма встановлена та запущена на телефоні. Користувач увійшов до свого облікового запису та в поле для пошуку ввів потрібну назву монети.

  Кроки виконання: Натискаємо на плюсик біля токену і у відкритому діалоговому вікні вводимо всі необхідні дані, після чого натискаємо кнопку “Зберегти”.

  Очікуваний результат: У списку токенів нашого портфеля з’являється нова транзакція з даними, які користувач ввів при створенні.

- Видалення транзакції з портфелю користувача:

  Опис: Після видалення транзакції у діалоговому вікні токену список у портфелі повинен оновитись і більше не містити її.

  Передумови: Програма встановлена та запущена на телефоні. Користувач увійшов у свій обліковий запис та додав транзакцію до свого портфеля.

  Кроки виконання: Натискаємо трикрапку на транзакції у портфелі та у діалоговому вікні натискаємо кнопку для видалення.

Очікуваний результат: Всі дані про дану транзакцію у базі даних повинні очиститись і більше не відображатись у інтерфейсі портфеля.

При тестуванні безпеки ж потрібно звернути увагу на обробку помилок та винятків у коді, захист даних користувача та авторизацію. Це є дуже важливою частиною, оскільки навіть найбільш функціональний додаток не буде користуватись попитом, якщо користувачі не довірятимуть даному продукту.

Пристрої з операційною системою Windows та Mac мають чітку операційну систему, яка працює однаково на всіх пристроях. Мобільні пристрої відрізняються, оскільки часто різні виробники створюють модифікації операційних систем для власних лінійок смартфонів.

Тестування Android додатку слід виконувати на різних версіях операційної системи та пристроях. Тільки так можна максимально об’єктивно проаналізувати результати.

Для тестування програмної системи на різних пристроях найкраще підходить емулятор, який доступний в Android Studio. Він надає можливість створити екземпляр будь-якого пристрою від Google з вибраними версіями операційної системи. Інтерфейс створення емулятора зображено на рисунку 2.1.

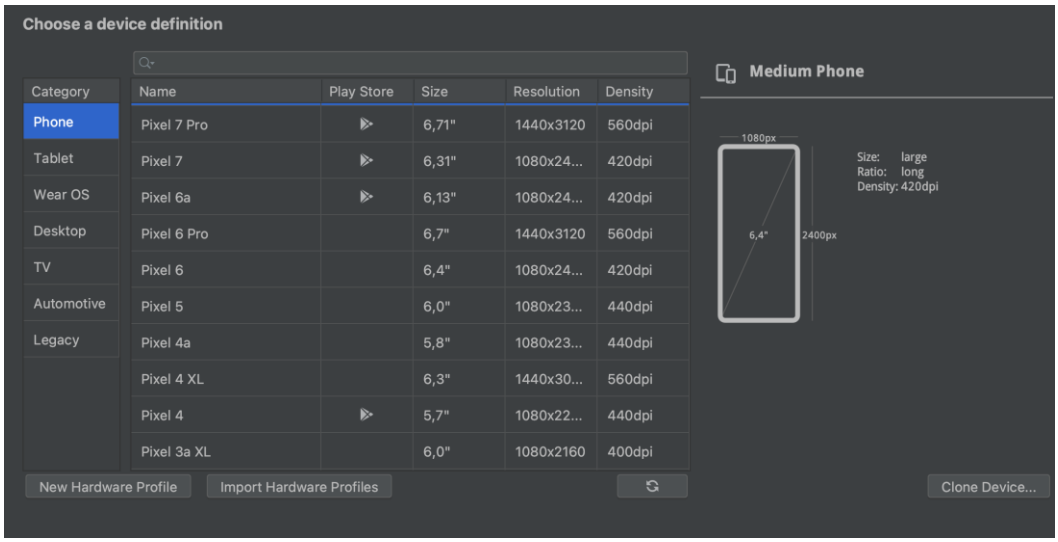

Рисунок 2.1 – Інтерфейс вікна створення емулятора в Android Studio

Не менш важливою є сама версія операційної системи. Якщо подивитися на графік використання різних версій Android у світі, який зображий на рисунку 2.2

можна зробити висновок, що найпопулярнішими версіями на даний момент є 11, 12, 13 та 14, яка стрімко набирає популярності.

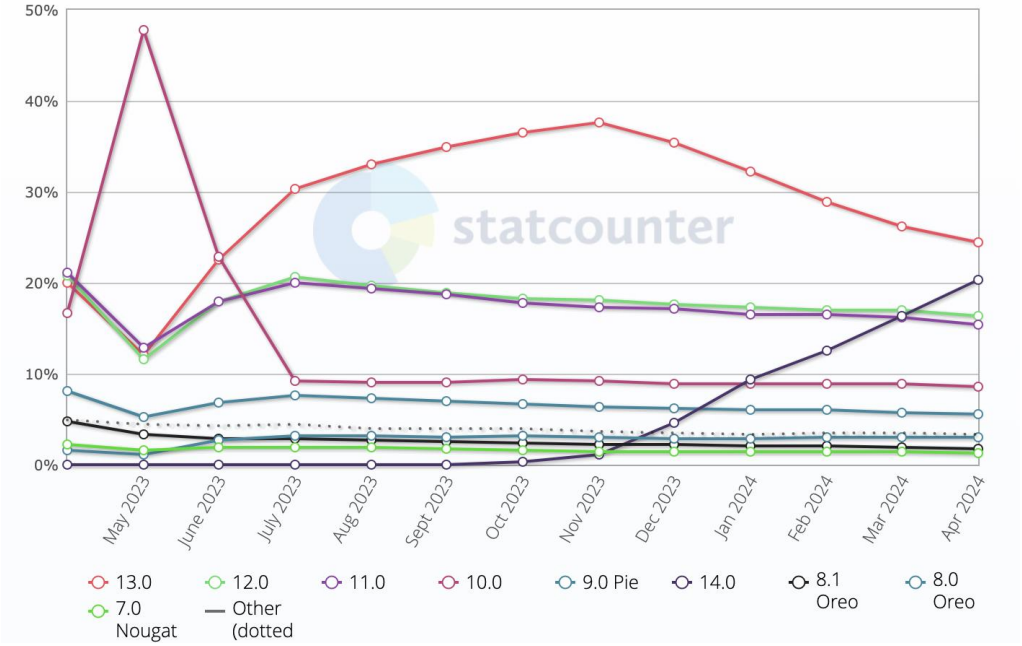

Рисунок 2.2 – Графік кількості користувачів кожної версії Android

Основуючись на цих даних, можна зробити висновок, що варто проводити тестування саме на найпопулярніших версіях, а також на одній старій версії, для того щоб розуміти, що програмна система працює належним чином. Оскільки мінімальна версія для нашої програмної системи є 8.0, ми будемо тестувати на ній.

2.2 Аналіз результатів

Використовуватимемо ручне тестування для більш якісного оцінювання справності того чи іншого функціоналу. На цьому етапі процесу ми самостійно заходимо у додаток і перевіряємо, чи відповідає дана програмна система всім стандартам.

У функціоналі входу за допомогою облікового запису Google було виявлено баг. Коли ми завантажуємо обліковий запис і після його вибору у діалоговому вікні згортаємо додаток, то при повторному відкритті додатку потрібно ще раз

натиснути на кнопку входу, щоб ввійти остаточно. Ця помилка не впливає на користувацький досвід і не загрожує безпеці програмної системи.

Також було знайдено кілька помилок, пов’язаних з інтерфейсом додатку. Наприклад, при натисканні на номер телефону або електронну пошту на екрані “Про додаток” нічого не відбувається, а тому користувач повинен вводити дані вручну, якщо він хоче зв’язатися з розробниками. Якщо ж видалити всі транзакції користувача і не оновити екран портфоліо, то ми не побачимо напису, який показує, що портфель порожній. Щоб його активувати, нам потрібно знову увійти на цей екран. Приклад такого напису зображено на рисунку 2.3.

Рисунок 2.3 - Вигляд екрану з пустим портфелем

Всі інші функціональності програмної системи працювали чудово під час тестування на різних пристроях та версіях операційної системи.

Найбільшу небезпеку для нашого додатку становить відсутність обфускації коду.

Обфускація означає зробити щось складним для розуміння. Програмний код маскується, щоб захистити інтелектуальну власність або комерційну таємницю, а також щоб запобігти зловмиснику зворотній інжиніринг програми [17].

Дана властивість є важливою, якщо додаток публікується на таких платформах як Google Play або Play Store. Суть її роботи досить проста: вона замінює велику частину назв та найменувань у коді та робить його нерозбірливим. Приклад обфускації показано на рисунку 2.4.

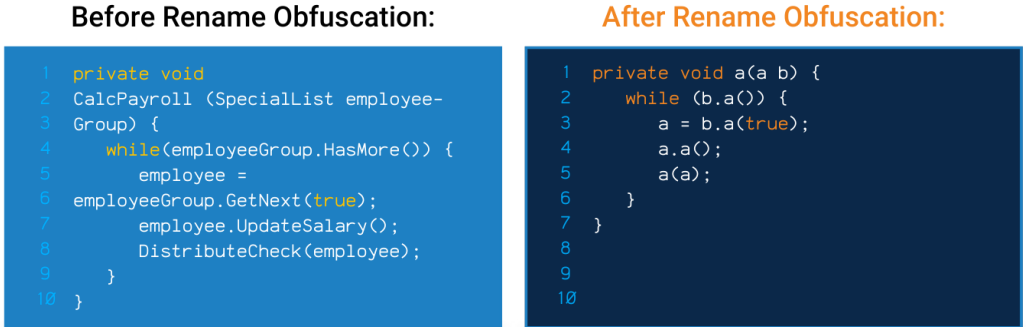

Рисунок 2.4 – Приклад роботи обфускації

Обфускація зберігає наш код в безпеці та ускладнює завдання хакерам для отримання ключів та іншої важливої інформації з програмної системи. Основним недоліком цього інструменту є сповільнення розробки додатку, оскільки процес компіляції тепер займатиме у десять разів більше часу. Це призводить до великого навантаження на комп'ютер, а тому не варто використовувати цю функціональність у некомерційних додатках.

Дані, які користувач надає додатку, а саме для входу, а також дані портфеля, повністю захищені Firebase та не можуть будь-яким чином бути отримані третіми особами.

Ще один важливий аспект у нашому додатку, який був протестований є User Experience. Користувачеві повинно бути легко освоїтися у нашому додатку та швидко зрозуміти, те, як працює основний функціонал. Перші декілька хвилин, коли інвестор відкриє наш додаток, є найважливішими, оскільки потрібно небагато часу, щоб оцінити, чи підійде він йому.

Зрозумілий та легкий інтерфейс зустрічає користувача ще з самого початку, а мінімалістичний стиль не дає йому заплутатися.

Якщо говорити про продуктивність, то тут вона досить висока. Інтеграція з Firebase дозволяє виконувати вхід у додаток без будь-яких затримок, так само як і додавати нову транзакцію до нашої бази даних.

Обмеженням для нас є обсяг пам'яті в базі даних для безкоштовного плану, а тому при збільшенні кількості користувачів нам доведеться придбати преміум

план. Кількість пам’яті, яка входить в безоплатний тариф зображено на рисунку 2.5.

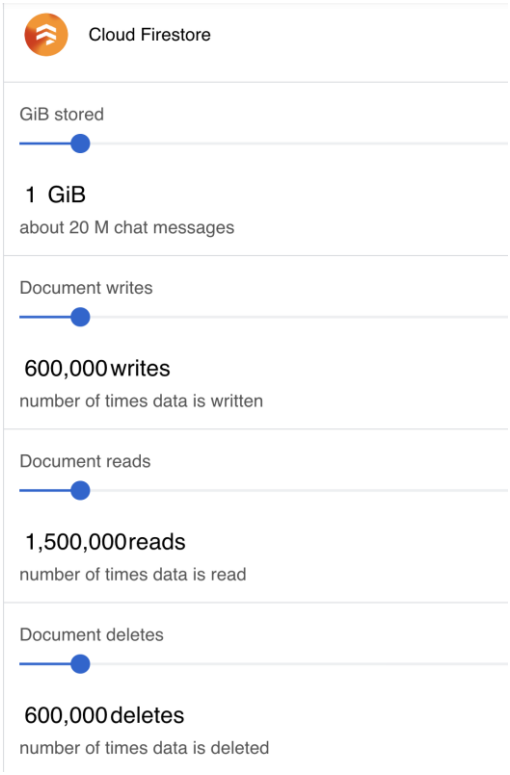

Рисунок 2.5 – Можливості безкоштовного плану Firebase для бази даних Firestore

Однак безкоштовний план надає достатній обсяг пам’яті та кількість створених документів в базі даних, тому не варто купувати підписку до перший декількох тисяч користувачів. Перевагою платних планів у Firebase є те, що ми оплачуємо тільки ту суму, на яку використали пам’ять, а отже не повинні переплачувати.

3 ЕКОНОМІКА

Поточні тренди на криптовалютному ринку є важливими для розвитку ринку в цілому. Аспекти, які пов’язані з криптоактивами у країнах першого світу, встановлюють правила для всіх. Важливим фактором є регуляторна чіткість та якість, оскільки заборона криптовалюти у будь-якій країні негативно вплине на весь ринок.

На даний момент багато країн приймають закони, які позитивно впливають на ринок, а недавнє затвердження біткоїн-ETF стане одним з головних факторів для росту всього ринку.

Біткоїн ETF – це особливий тип ETF, який дозволяє інвесторам торгівлю ціною біткоїна на звичайних фінансових ринках. Цей тип ETF дозволяє окремим особам й установам отримувати доступ до біткоїна без ризиків, пов’язаних із безпосередньою купівлею і зберіганням криптовалюти [18].

На 2023 рік кількість людей, які володіють криптовалютою, перевищила 580 мільйонів, і ця цифра продовжує зростати. Графік збільшення кількості користувачів зображений на рисунку 3.1.

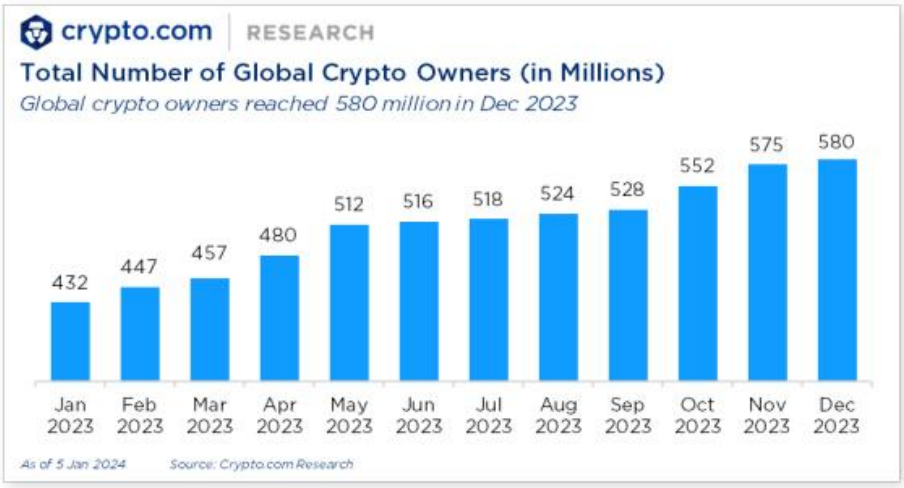

Рисунок 3.1 – Графік кількості власників криптовалют

Таким чином, кількість людей у нашій цільовій аудиторії є надзвичайно великою, оскільки ця програмна система буде корисною для кожного власника криптовалюти.

Останні роки інвестори вибирають саме мобільні додатки, оскільки це надзвичайно зручно у будь-який момент мати можливість переглядати зміни у своєму крипто-портфелі.

Вартість розробки цієї програмної системи буде залежати від багатьох факторів, таких як кількість людей у команді, країна, де буде відбуватись розробка, та інші. При конструюванні програмної системи з таким набором функціоналу можна обмежитись одним дизайнером та одним Android розробником. На розробку дизайну було виділено приблизно 35 годин а програмування зайняло приблизно 80 годин.

Якщо проаналізувати ринок зарплат, то можна зробити висновок, що спеціаліст рівня Middle в Україні може отримувати приблизно 10$ за годину, а тому розробка цієї програмної системи обійдеться приблизно в 1150$.

Варто врахувати, що дана програмна система може вважатися MVP і не містить всього функціоналу, необхідного для конкуренції на ринку мобільних додатків. Для розробки повного набору функціоналу додатку для трекінгу криптовалюти потрібно приблизно 1000 годин. Таким чином, можна порахувати, що додаток на платформу Android обійдеться більше ніж у 10000$.

У ціну підтримки програмної системи входить оплата Firebase та інших платних підписок, а також зарплата програміста та дизайнера для виправлення багів та створення нового функціоналу. Це буде коштувати більше ніж дві тисячі доларів на місяць. Статистика зарплати спеціаліста з досвідом від 1 до 2 років зображена на рисунку 3.2.

Рисунок 3.2 – Статистика зарплат спеціалістів рівня Middle

Моделі монетизації Android додатку бувають різних видів, але серед найпопулярніших є:

a. Платні програми
b. Реклама
c. Абонентська плата

Платні програми - це найпростіша модель, де користувач повинен заплатити за мобільну програму. Ця модель є вже досить застарілою, але все ще працює для деяких категорій цифрових продуктів. Одним з найважливіших секретів, щоб зробити цю модель вигідною, є включення пробного періоду, щоб користувач міг спробувати додаток.

При рекламній моделі потрібно розміщувати рекламу у своєму продукті. Показ реклами може відбуватись при переході між якимось екранами або за допомогою банеру у нижній частині екрану.

Модель на основі підписки є найпопулярнішою зараз. У цій моделі з користувачів стягується регулярна підписка за доступ до всього функціоналу додатку.

Враховуючи те, що дана програмна система буде публікуватись на платформі Google Play у майбутньому, то перша модель нам не підійде, оскільки користувачі смартфонів з операційною системою Android неохоче купують платні застосунки. Головною причиною є цільова аудиторія, яка переважно проживає у країнах

другого та третього світу, оскільки на даний момент є надзвичайно багато доступних і водночас потужних пристроїв з операційною системою Android.

Рекламна модель є дуже прибутковою у тих додатках та іграх, де користувач проводить багато часу протягом дня. В такому випадку під час сесії програмної системи ми маємо змогу показати багато реклам. Така модель не підійде для нашого додатку, оскільки інвестори будуть відкривати його нечасто і не надовго.

Модель на основі підписки стане найкращим рішенням для цієї програмної системи. Вона забезпечує постійне джерело доходу, яке дозволить підтримувати розробку та обслуговування. Важливо виконати A/B тестування для того, щоб підібрати найоптимальнішу ціну та період для максимізації прибутків.

Серед основних економічних ризиків для цієї програмної системи є волатильність ринку, а також регуляторні ризики. Волатильність сильно впливає на настрій серед інвесторів, і при сильному падінні цін на токени все менше людей проявляє будь-який інтерес до цього ринку. Таким чином, можна зробити висновок, що найсприятливішим часом для запуску подібного продукту стане період бичачого тренду.

Ринки, на яких спостерігається стійке та/або значне зростання, називаються бичачими ринками [19]. Приклад зростаючого тренду зображено на рисунку 3.3.

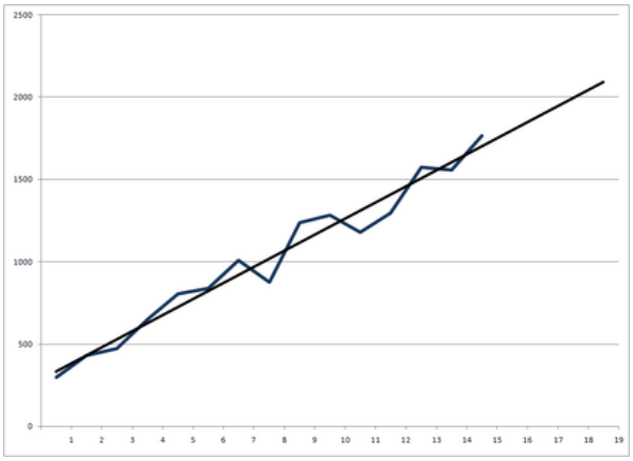

Рисунок 3.3 – Вигляд зростаючого тренду

Регуляторні ризики також впливають на цей ринок в цілому. Останніми роками все більше країн легалізують криптовалюту, але не всі врегулювали її відповідно до законодавства. На даний момент часу близько 52% країн підготували всеосяжне законодавство з цього питання. Це число стрімко зростає, порівняно з 2018 роком кількість держав, які забезпечили законодавче врегулювання криптовалют, зросла на 53%. Карта з цими країнами зображена на рисунку 3.4.

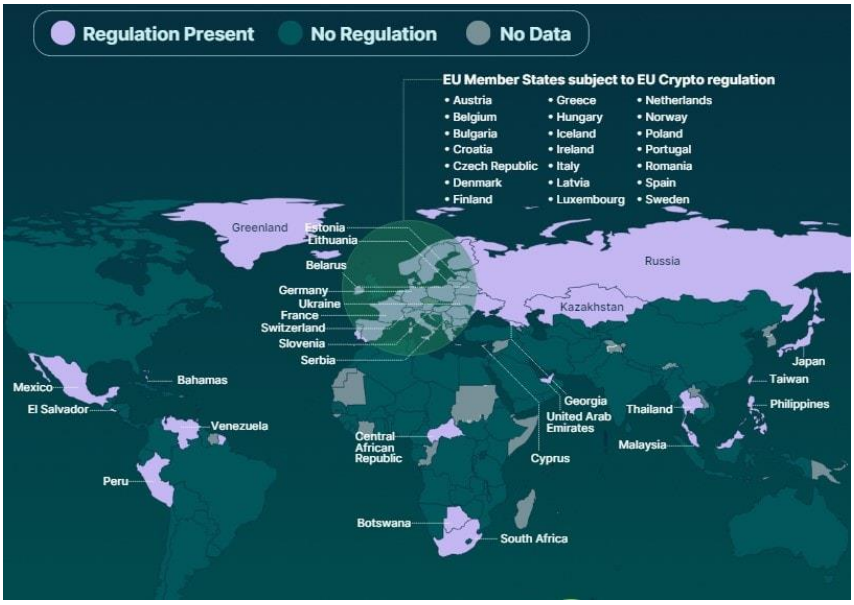

Рисунок 3.4 – Карта країн з врегульованою криптовалютою

Таким чином, варто зазначити, що зараз мінімальні ризики для запуску такого продукту, оскільки всі основні фактори позитивно на це впливають. Через постійне зростання кількості людей у цільовій аудиторії є велика перспектива та вигода від реалізації цього додатку.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ ТА ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Долікарська допомога при контузіях

Контузією головного мозку ми називаємо порушення цілісності мозкової речовини на обмеженій ділянці, характеризується наявністю локалізованого макроскопічного ушкодження речовини мозку [20].

Частіше за все причиною контузії є ударна хвиля від вибуху або удар людини об землю чи вводу. Характер і ступінь впливу ударної хвилі залежать від потужності вибуху, відстані, метеорологічних умов, місцезнаходження (у будинку, на відкритій місцевості) і положення тіла (лежачи, сидячи, стоячи) людини. Вони характеризуються легкими, середніми, важкими і дуже важкими травмами.

Надлишковий тиск у фронті ударної хвилі величиною 10 кПа і менше вважається безпечним для людей і тварин, розташованих поза укриттями.

Легкі ураження настають за дії надлишкового тиску 20...40. Вони виражаються в швидкоминучих порушеннях функцій організму (дзенькіт у вухах, запаморочення, головний біль). Можливі вивихи, забиті місця.

Ураження середньої важкості виникають за умови надлишкового тиску 40…60 кПа. Така величина надлишкового тиску може призводити до вивихів кінцівок, контузії головного мозку, ушкодження органів слуху, викликати кровотечу з носа й вух.

Важкі контузії і травми можливі за надлишкового тиску від 60 до 100 кПа. Вони характеризуються сильною контузією всього організму, втратою свідомості, переломами кісток, кровотечею з носа й вух. У цьому разі можливе ушкодження внутрішніх органів і внутрішня кровотеча.

Дуже важкі контузії і травми в людей виникають за умови дії надлишкового тиску більше, ніж 100 кПа. Вони проявляються в розриві внутрішніх органів, переломах кісток, внутрішній кровотечі, струсу мозку, тривалій втраті свідомості. Розриви спостерігаються в органах, які містять велику кількість крові (печінка, селезінка, нирки), наповнених газом (легені, кишечник) чи порожнини, наповнені

рідиною (головний мозок, сечовий і жовчний міхур). Ці травми можуть призвести до смертельного результату [21].

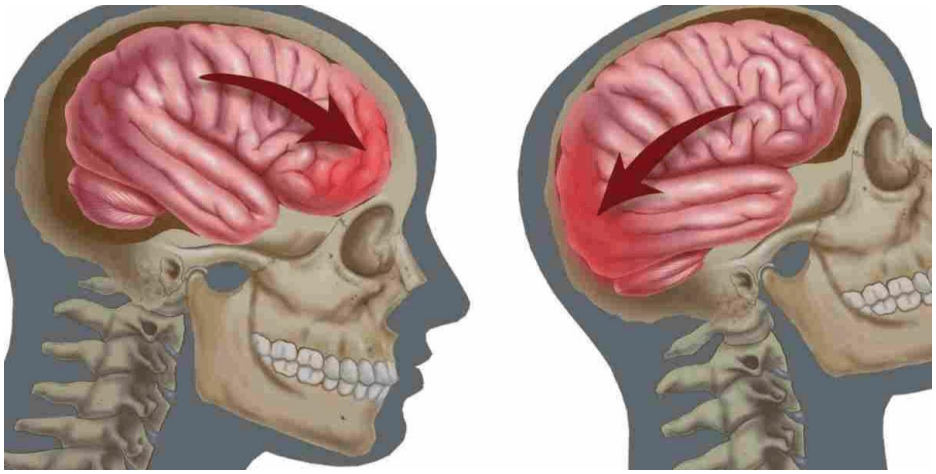

Рисунок 4.1 – Контузія головного мозку

В очікуванні медичної допомоги потрібно надати людині, яка постраждала, долікарську допомогу:

1. Якщо постраждалий втратив свідомість – необхідно забезпечити прохідність дихальних шляхів методом виведення нижньої щелепи. Варто обмежити рухи у шийному відділі хребта. У людини також може бути блювота. Вкрай необхідно забезпечити вільне дихання, оскільки є ризик аспірації блювотними масами.
2. Очистити порожнину рота від сторонніх предметів, які могли потрапити туди під час вибуху й заважають прохідності дихальних шляхів.
3. Звільнити груди і шию від одягу, якщо він стискає та заважає постраждалому вільно дихати.
4. Якщо постраждалий не дихає, варто розпочати базові реанімаційні заходи до приїзду медиків: натискання на грудну клітку і, за можливості, штучну вентиляцію у співвідношенні 30:2 [22].

Контузія у більшості випадків не є смертельною і пацієнт відновлюється після неї досить швидко. Найбільшу небезпеку вона несе у довгостроковій

перспективі, оскільки часто є причиною затяжних психічних розладів, головних болей та частих змін настрою.

Отже, варто серйозно ставитися до такого ураження як контузія особливо в наші дні, коли більшість бійців і навіть деякі цивільні проходить через це. Потрібно дотримуватися всіх рекомендацій лікарів і тоді пацієнт має великі шанси на повне одужання.

4.2 Електробезпека на будівельному майданчику

Сучасне будівництво неможливо уявити без електропостачання, при цьому з кожним роком спостерігається зростання ступеня електрифікації багатьох технологічних процесів і операцій. Споживачами електричної енергії на будівельному майданчику є будівельні машини і механізми (крани та інші підйомні механізми, ручний електричний інструмент, штукатурні і малярні станції, зварювальні апарати тощо), освітлювальні установки, а також технологічні процеси (електричне прогрівання чи пропарювання бетону, відігрівання ґрунту, відкачування води тощо). Тому електропостачання будівельного майданчика є першочерговим завданням підготовчого періоду [23].

Під час організації електропостачання будівельного майданчика в обов’язковому порядку необхідно дотримуватися заходів з електробезпеки [23].

Улаштування і технічне обслуговування тимчасових і постійних електричних мереж на виробничій території повинен здійснювати персонал, що має відповідну кваліфікаційну групу з електробезпеки. Група допуску з електробезпеки працівника визначає, перш за все, рівень його знань безпечних методів роботи з електроустановками. Існує всього п’ять груп:

1. Першу групу повинні мати працівники, які обслуговують електроустановки але не мають спеціальних електротехнічних знань (будівельники, муляри, штукатури тощо).

2. Другу групу повинні мати кранівники, електрослюсарі, зварювальники тощо.
3. Третю групу повинні мати працівники оперативно-ремонтного електротехнічного персоналу.
4. Четверту групу повинні мати інженери з техніки безпеки, енергетики будівельних управлінь та інші ІТП з числа електротехнічного персоналу.
5. П’яту групу повинні мати відповідальні особи та електротехнічний персонал, що обслуговують електроустановки напругою вище 1000 В [23].

Переважна частина електричних мереж на будівельному майданчику мають тимчасовий характер. Незважаючи на специфічні особливості, вони повинні відповідати тим же правилам і нормам, що й постійні. Улаштування та експлуатація електроустановок повинні здійснюватися відповідно до Правил технічної експлуатації електроустановок споживачів, Правил улаштування електроустановок, , НПАОП 40.1-1.01, НПАОП 40.1-1.21, НПАОП 0.00-1.29, НПАОП 40.1-1.32 [23].

Як правило, електропостачання будівельного майданчика забезпечується від найближчих стаціонарних джерел, до яких приєднується трансформаторна підстанції (ТП) об’єкта будівництва. Трансформаторна підстанція містить трифазні трансформатори з двома обмотками, а також апарати комутації і захисту, пристрої управління, контролю і обліку електроенергії. За конструкцією бувають відкриті, закриті і пересувні (мобільні) підстанції [23].

Відкриті підстанції повинні мати огорожу і зовнішнє освітлення. Закриті трансформаторні підстанції розташовують в приміщеннях, які споруджуються з конструкцій, що швидко збираються. Розміщувати їх на будівельному майданчику слід ближче до центу підключення всіх споживачів, але поза зоною дії підйомних механізмів [23].

Більшість споживачів електричної енергії на об’єктах будівництва розраховані на напругу 380/220 В. В умовах будівництва споруджують, як правило, чотирипровідну глухозаземлену трифазну мережу змінного струму частотою 50 Гц. В таких мережах, відповідно до ПУЕ, обов’язково заземлюють нейтраль (нульову

точку) трансформаторної підстанції (ТП). Для цього споруджують заземлюючий контур, до якого приєднують вивід нульової точки трансформатора, а отже, і нульовий провід мережі. Опір заземлюючого пристрою ТП, відповідно до ПУЕ, повинен бути не більше 4 Ом [23].

Для живлення споживачів електричною енергією на будівельному майданчику споруджують, як правило, тимчасові електричні мережі, переважно повітряні лінії на опорах, як більш дешеві і легко споруджувані. Тимчасові електричні мережі також можуть бути розведені по території будівельного майданчика під землею, в траншеях, каналах, колекторах тощо [23].

Розведення тимчасових електромереж напругою до 1000 В, що використовуються для електрозабезпечення об'єктів будівництва, необхідно виконати ізольованими проводами чи кабелями на опорах або конструкціях, розрахованих на відповідну механічну міцність під час прокладання по них проводів і кабелів на висоті над рівнем землі та настилу не менше ніж, м: 2,5 – над робочими місцями; 3,5 – над проходами; 6,0 – над проїздами [23].

Освітлення підключають до окремої мережі, не пов'язаної з живленням електрообладнання великої потужності. Освітлювальну мережу допускається прокладати на опорах спільно з мережею 380 В. Траса тимчасової повітряної лінії повинна бути якомога прямою [23].

Усі електропускові пристрої слід розміщувати так, щоб унеможливлювався пуск машин, механізмів і устаткування сторонніми особами. Забороняється вмикання декількох струмоприймачів одним пусковим пристроєм. Розподільні щити і рубильники необхідно закривати на замок [23].

Під час улаштування електричних мереж і електроустановок необхідно виконувати маркування, написи, цифрові і буквені позначення, що вказують на призначення розподільного щита, пускового пристрою, кабелю, проводу тощо. Монтаж і експлуатація електричної мережі і електроустановок повинні виключати можливість теплових проявів електричного струму, які можуть привести до займання ізоляції або горючих матеріалів, що знаходяться поблизу [23].

Для захисту від ураження електричним струмом і запобігання небезпечним витокам струму, електричні мережі обладнують пристроями захисного відключення. Штепсельні розетки на номінальні струми до 20 А, призначені для живлення переносного електроустаткування і ручного електроінструменту, що застосовуються поза приміщеннями, повинні бути обладнані пристроями захисного відключення (ПЗВ) зі струмом спрацьовування не більше ніж 30 мА або кожна розетка повинна живитися від індивідуального розподільного трансформатора з напругою не більше ніж 25 В [23].

Металеві будівельні риштування, металеві огорожі місць, де виконуються роботи, полиці та лотки для прокладання кабелів і проводів, рейкові колії вантажопідіймальних кранів і транспортних засобів з електричним приводом, корпуси устаткування, машин і механізмів з електроприводом необхідно заземлювати одразу після їх встановлення на місце до початку виконання будь-яких робіт. Для заземлення електроустаткування можуть бути застосовані штучні або природні заземлювачі [23].

Отже, електробезпека на будівельному майданчику є критично важливою для того, щоб забезпечити працівників безпечними умовами, а також для уникнення нещасних випадків. Навіть недотримання одного правила електробезпеки може призвести до фатальних наслідків на будівельному майданчику.

ВИСНОВКИ

В результаті виконаної кваліфікаційної роботи бакалавра було проведено аналіз предметної області та аналогічних проектів, а після цього спроектовано та сконструйовано програмну систему для виконання операцій з криптовалютою.

Після проведення аналізу предметної області та проектів потенційних конкурентів, стало зрозуміло, який саме функціонал потребує цільова аудиторія даної тематики. Після отримання результатів аналізу було пріоритезовано всі задачі проекту.

Під час написання програмної системи було використано найновіші та найактуальніші підходи в розробці для Android, а також інструменти, які дозволяють створювати функціонал будь-якої складності. Серед таких інструментів можна виділити Firebase, IDE Android Studio та API для отримання всіх необхідних даних про токени.

Однією з найважливіших задач було забезпечення максимальної безпеки та захищеності додатку, оскільки жоден з користувачів не хоче, щоб хтось отримав доступ до даних його персонального криптопортфелю.

В підсумку було розроблено мобільний Android додаток, з наступним функціоналом:

- Реєстрація та вхід за допомогою логіну та паролю або облікового запису Google;
- Пошук криптоактивів за назвою;
- Додавання токену у вигляді транзакції у свій портфель, а також можливість його видалення згодом;
- Прорахунок загального прибутку або втрати по портфелі, а також для кожної транзакції окремо.

Дана програмна система містить весь необхідний функціонал для інвестора початківця, та є легкою у підтримці і розширенні в майбутньому.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ладуба М. Що таке криптовалюта та де її можна використовувати [Електронний ресурс] / Микола Ладуба – Режим доступу до ресурсу: https://mc.today/uk/kriptovalyuta/.

2. Що таке блокчейн? [Електронний ресурс] – Режим доступу до ресурсу: https://blog.whitebit.com/uk/what-is-blockchain-technology/.

3. Що таке NFT? [Електронний ресурс] – Режим доступу до ресурсу: https://www.kraken.com/uk-ua/learn/what-are-non-fungible-tokens-nft.

4. Software Development Process [Електронний ресурс] – Режим доступу до ресурсу: https://acqnotes.com/acqnote/careerfields/software-development-process.

5. Канбан/скрам/аджайл — що обрати для вашого проєкту? [Електронний ресурс] – Режим доступу до ресурсу: https://brainrain.com.ua/uk/%D1%81%D0%BA%D1%80%D0%B0%D0%BC-2/.

6. Cloud Firestore [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs/firestore.

7. Harmon P. Understanding UML / P. Harmon, M. Watson., 1998. – 367 с. – (Elsevier Science).

8. Типи мобільних додатків [Електронний ресурс] – Режим доступу до ресурсу: https://smile-ukraine.com/ua/mobile-apps/mobile-apps-types.

9. Norton W. Java Programming for Beginners / Will Norton., 2021. – 146 с. – (Giovanna de Rosa).

10. Stephen S. Learn Kotlin Programming / S. Stephen, S. Bocutiu., 2019. – 514 с. – (Packt Publishing).

11. Що має знати Android розробник [Електронний ресурс] – Режим доступу до ресурсу: https://foxminded.ua/shcho-potribno-znaty-android-rozrobnyku/.

12. GoogleSignInClient [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/android/reference/com/google/android/gms/auth/api/signin/GoogleSignInClient.

13. McLaughlin B. Head First Object-Oriented Analysis and Design / B. McLaughlin, G. Pollice, D. West., 2007. – 600 с. – (O\'Reilly Media, Incorporated).

14. Why use Tether? [Електронний ресурс] – Режим доступу до ресурсу: https://tether.to/en/why-tether/.

15. System Testing – Software Engineering [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/system-testing/.

16. What is Functional Testing? [Електронний ресурс] – Режим доступу до ресурсу: https://www.opentext.com/what-is/functional-testing.

17. Nissenbaum H. Obfuscation A User\'s Guide for Privacy and Protest / H. Nissenbaum, F. Brunton.. – (MIT Press).

18. Що таке Bitcoin ETF? [Електронний ресурс] – Режим доступу до ресурсу: https://academy.binance.com/uk/articles/bitcoin-etfs-explained.

19. What is a bull or bear market? [Електронний ресурс] – Режим доступу до ресурсу: https://www.coinbase.com/learn/crypto-basics/what-is-a-bull-or-bear-market.

20. Гринзовський А. М. Види ушкоджень в бойових і не бойових умовах / А. М. Гринзовський, П. Б. Волянський // Домедична допомога в екстремальних ситуаціях та медичний захист населення в надзвичайних ситуаціях / А. М. Гринзовський, П. Б. Волянський. – Київ: Скіф, 2018. – С. 69.

21. Сєріков Я. О. Безпека життєдіяльності та охорона праці / Я. О. Сєріков, Л. Ф. Коженевські, М. В. Хворост. – Харків, 2021. – 255 с. – (Харківський національний університет міського господарства імені О. М. Бекетова).

22. Контузія від дії вибухової хвилі: симптоми та домедична допомога [Електронний ресурс]. – 2022. – Режим доступу до ресурсу: https://moz.gov.ua/article/health/kontuzija-vid-dii-vibuhovoi-hvili-simptomi-ta-domedichna-dopomoga-.

23. Шаповалов В. А. Забезпечення електробезпеки на будівельному майданчику / В. А. Шаповалов. – Кривий Ріг: КНУ, 2017. – 162 с.

ДОДАТКИ

ДОДАТОК А

Код клієнту для роботи з API в класі CoinsClient

```
public class CoinsClient {
  private MutableLiveData<List<NftModel>>
mutableLiveDataTopNfts;
  private MutableLiveData<List<ModelCoin>>
mutableLiveDataAllCoins;
  private MutableLiveData<List<ModelCoin>>
mutableLiveDataIdsCoins;
  private static CoinsClient instance;

  public static CoinsClient getInstance() {
    if (instance == null) {
      instance = new CoinsClient();
    }
    return instance;
  }

  private CoinsClient() {
    mutableLiveDataTopNfts = new MutableLiveData<>();
    mutableLiveDataAllCoins = new MutableLiveData<>();
    mutableLiveDataIdsCoins = new MutableLiveData<>();
  }

  //отримати список топ nft
  public MutableLiveData<List<NftModel>>
getMutableLiveDataTopNfts() {
    return mutableLiveDataTopNfts;
 }

  //запит топ nft
  public void updateMutableLiveDataTopNfts() {
    Call<ModelTopNfts> call =
        RetrofitClient.getInstance().getApi().getTopNfts();

    call.enqueue(new Callback<ModelTopNfts>() {
      @Override
      public void onResponse(
          Call<ModelTopNfts> call, Response<ModelTopNfts>
response) {
```

```
        if (response.isSuccessful()) {
          mutableLiveDataTopNfts.postValue(response.body().
getNfts());
          Call l = call;
        } else {
        }
      }

      @Override
      public void onFailure(Call<ModelTopNfts> call,
Throwable t) {}
    });
  }

  //отримання списку всіх монет
  public MutableLiveData<List<ModelCoin>>
getMutableLiveDataAllCoins() {
    return mutableLiveDataAllCoins;
  }

  //отримання всіх монет
  public void updateMutableLiveDataAllCoins() {
    Call<List<ModelCoin>> call =
        RetrofitClient.getInstance().getApi().getAllCoins()
;
    call.enqueue(new Callback<List<ModelCoin>>() {
      @Override
      public void onResponse(
          Call<List<ModelCoin>> call,
Response<List<ModelCoin>> response) {
        if (response.isSuccessful()) {
          mutableLiveDataAllCoins.postValue(response.body()
);
        } else {
        }
      }

      @Override
      public void onFailure(Call<List<ModelCoin>> call,
Throwable t) {}
    });
  }

  //отримання списку по id
```

```
  public MutableLiveData<List<ModelCoin>>
getMutableLiveDataIdsCoins() {
    return mutableLiveDataIdsCoins;
  }

  //оновлення списку по id
  public void updateMutableLiveDataIdsCoins(String ids) {
    Call<List<ModelCoin>> call =
        RetrofitClient.getInstance().getApi().getIdsCoins(i
ds);
    call.enqueue(new Callback<List<ModelCoin>>() {
      @Override
      public void onResponse(
          Call<List<ModelCoin>> call,
Response<List<ModelCoin>> response) {
        if (response.isSuccessful()) {
          mutableLiveDataIdsCoins.postValue(response.body()
);
        } else {
        }
      }

      @Override
      public void onFailure(Call<List<ModelCoin>> call,
Throwable t) {}
    });
  }
}
```

ДОДАТОК Б

Код класу репозиторію

```
public class RepositoryAuth {
  private MutableLiveData<Boolean> mutableLiveDataIsLogIn;
  private MutableLiveData<FirebaseUser>
mutableLiveDataFirebaseUser;
  private MutableLiveData<List<ModelCoinFirebase>>
mutableLiveDataModelFirebase;

  private FirebaseAuth firebaseAuth;
  private FirebaseFirestore firebaseFirestore;
  private Context context;

  public RepositoryAuth(Application application) {
    this.context = application;
    firebaseAuth = FirebaseAuth.getInstance();
    firebaseFirestore = FirebaseFirestore.getInstance();

    mutableLiveDataIsLogIn = new MutableLiveData<>(false);
    mutableLiveDataFirebaseUser = new MutableLiveData<>();
    mutableLiveDataModelFirebase = new
MutableLiveData<>();
    if (firebaseAuth.getCurrentUser() != null) {
      mutableLiveDataIsLogIn.postValue(true);
      mutableLiveDataFirebaseUser.postValue(firebaseAuth.ge
tCurrentUser());
    }
  }
  public MutableLiveData<Boolean>
getMutableLiveDataIsLogIn() {
    return mutableLiveDataIsLogIn;
  }

  public MutableLiveData<FirebaseUser>
getMutableLiveDataFirebaseUser() {
    return mutableLiveDataFirebaseUser;
  }
  public MutableLiveData<List<ModelCoinFirebase>>
  getMutableLiveDataModelFirebase() {
    return mutableLiveDataModelFirebase;
```

```
  }
  //Вхід в аккаунт firebase
  public void logIn(String email, String password) {
    firebaseAuth.signInWithEmailAndPassword(email,
password)
        .addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
          @Override
          public void onComplete(@NonNull Task<AuthResult>
task) {
            if (task.isSuccessful()) {
              mutableLiveDataFirebaseUser.postValue(
                  firebaseAuth.getCurrentUser());
            } else {
              Toast.makeText(context, "User not found",
Toast.LENGTH_LONG)
                  .show();
            }
          }
        });
  }

  //Реєстрація аккаунту в firebase
  public void signUp(String email, String password) {
    firebaseAuth.createUserWithEmailAndPassword(email,
password)
        .addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
          @Override
          public void onComplete(@NonNull Task<AuthResult>
task) {
            if (task.isSuccessful()) {
              Toast
                  .makeText(context, "Registration
complete", Toast.LENGTH_LONG)
                  .show();
              DocumentReference documentReference =
                  firebaseFirestore.collection("Users").doc
ument(
                      firebaseAuth.getCurrentUser().getUid(
));
              Map<String, Object> map = new HashMap<>();
              map.put("email", email);
              documentReference.set(map);
              mutableLiveDataFirebaseUser.postValue(
```

```
                    firebaseAuth.getCurrentUser());
            } else
              Toast.makeText(context, "Registration
failed", Toast.LENGTH_LONG)
                    .show();
          }
        });
  }

  //Отримання клієнта нашого гугл аккаунта
  public GoogleSignInClient getClient() {
    GoogleSignInOptions gso =
        new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIG
N_IN)
            .requestIdToken(context.getString(R.string.defa
ult_web_client_id))
            .requestEmail()
            .build();

    return GoogleSignIn.getClient(context, gso);
  }

  //Вхід за допомогою гугл і нашого ключа
  public void logInWithGoogle(String token) {
    AuthCredential credential =
GoogleAuthProvider.getCredential(token, null);
    firebaseAuth.signInWithCredential(credential)
        .addOnCompleteListener(new
OnCompleteListener<AuthResult>() {
          @Override
          public void onComplete(@NonNull Task<AuthResult>
task) {
            if (task.isSuccessful()) {
              Toast.makeText(context, "Success Log In",
Toast.LENGTH_LONG)
                    .show();
              DocumentReference documentReference =
                  firebaseFirestore.collection("Users").doc
ument(
                      firebaseAuth.getCurrentUser().getUid(
));
              Map<String, Object> map = new HashMap<>();
              map.put("email",
task.getResult().getUser().getEmail());
```

```
                map.put("name",
task.getResult().getUser().getDisplayName());
                documentReference.set(map);
                mutableLiveDataFirebaseUser.postValue(
                    firebaseAuth.getCurrentUser());
              }
            }
          });
      /*FirebaseAuth.getInstance().signOut();
      getClient().signOut();*/
    }

    //Додавання нової монети в портфоліо
    public void addElementToFirebase(
        String id, String price, String count, String date)
{
      DocumentReference documentReference =
          firebaseFirestore.collection(firebaseAuth.getCurren
tUser().getUid())
              .document();
      Map<String, String> map = new HashMap<>();
      map.put("id", id);
      map.put("price", price);
      map.put("count", count);
      map.put("date", date);
      map.put("hashId", documentReference.getId());
      documentReference.set(map);
    }
    //Отримання всіх монет з портфоліо
    public void getAllElementsFromFirebase() {
      List<ModelCoinFirebase> listMyCoins = new
ArrayList<>();
      Task<QuerySnapshot> documentReference =
          firebaseFirestore.collection(firebaseAuth.getCurren
tUser().getUid())
              .get()
              .addOnCompleteListener(new
OnCompleteListener<QuerySnapshot>() {
                @Override
                public void onComplete(@NonNull
Task<QuerySnapshot> task) {
                  if (task.isSuccessful()) {
                    for (QueryDocumentSnapshot document :
task.getResult()) {
                      listMyCoins.add(new ModelCoinFirebase(
```

```
                        document.getData().get("id").toStri
ng(),
                        document.getData().get("price").toS
tring(),
                        document.getData().get("count").toS
tring(),
                        document.getData().get("date").toSt
ring(),
                        document.getData().get("hashId").to
String()));
                    }
                } else {
                }
                mutableLiveDataModelFirebase.postValue(list
MyCoins);
            }
        });
  }
  //Видалення елементу з портфоліо
  public void deleteElementFromFirebase(String name) {
    DocumentReference documentReference =
        firebaseFirestore.collection(firebaseAuth.getCurren
tUser().getUid())
            .document(name);
    documentReference.delete();
  }
}
```

ДОДАТОК В

Код екрану портфоліо PortfolioFragment

```
public class PortfolioFragment
    extends Fragment implements
CoinsAdapterPortfolio.ClickMoreInfoListener {
  private ViewModelBottomSheet viewModelBottomSheet;
  private ViewModelNetwork viewModelNetwork;
  private List<ModelCoinFirebase> listMyCoins = new
ArrayList<>();
  private List<ModelCoinPortfolio> listPortfolio;
  private RecyclerView recyclerView;
  private Disposable disposable, disposableRetrofit;
  private String ids = "";
  private TextView textAllMoney, textEmpty;
  private ProgressBar progressBarPortfolio;
  private CoinsAdapterPortfolio coinsAdapterPortfolio;
  private PortfolioBottomSheet portfolioBottomSheet;

  @Override
  public void onCreate(@Nullable Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    RxJavaPlugins.setErrorHandler(th -> {});
  }

  @Override
  public View onCreateView(
      LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    viewModelBottomSheet = new
ViewModelProvider(requireActivity())
                                .get(ViewModelBottomSheet.cl
ass);
    //Наблюдач за списком всіх своїх монет в Firebase
    viewModelBottomSheet.getMutableLiveDataModelFirebase().
observe(
        getViewLifecycleOwner(), new
Observer<List<ModelCoinFirebase>>() {
          @Override
```

```
          public void onChanged(List<ModelCoinFirebase>
modelCoinFirebases) {
            if (disposable != null)
              disposable.dispose();

            listMyCoins = modelCoinFirebases;
            for (ModelCoinFirebase el : listMyCoins) {
              ids += el.getIdName() + ",";
            }
            if (ids.trim().length() > 0) {
              disposable =
                  Observable.interval(1000, 7000,
TimeUnit.MILLISECONDS)
                      .observeOn(AndroidSchedulers.mainThre
ad())
                      .subscribe(
                          el -> {
viewModelNetwork.updateids(ids); }, e -> {});
              textEmpty.setVisibility(View.GONE);
            } else {
              progressBarPortfolio.setVisibility(View.GONE)
;
              textEmpty.setVisibility(View.VISIBLE);
            }
          }
        });
    viewModelNetwork =
        new
ViewModelProvider(requireActivity()).get(ViewModelNetwork.c
lass);
    //Наблюдач за зміною списку з Retrofit
    viewModelNetwork.getMutableLiveDataIdsCoins().observe(
        getViewLifecycleOwner(), new
Observer<List<ModelCoin>>() {
          @Override
          public void onChanged(List<ModelCoin> modelCoins)
{
            //Отримання списку з Firebase, і після
отримання зписку з Retrofit
            //конвертувати в список з кастомною Model
            disposableRetrofit =
                Observable.fromIterable(listMyCoins)
                    .map(o -> {
                      for (ModelCoin el : modelCoins) {
```

```
                        if
(el.getFull_id().equals(o.getIdName())) {
                          return new ModelCoinPortfolio(o,
el);
                        }
                      }
                      return 0;
                    })
                    .toList()
                    .subscribe(objects -> {
                      if (!objects.contains(0)) {
                        coinsAdapterPortfolio.updateList(ob
jects);
                        countAllMoneyAndGrowth(objects);
                      }
                    }, (throwable) -> {});

            progressBarPortfolio.setVisibility(View.GONE);
          }
        });
    return inflater.inflate(R.layout.fragment_portfolio,
container, false);
  }

  //Підрахунок загальної кількості грошей
  public void countAllMoneyAndGrowth(List<Serializable>
objects) {
    double allMoney = 0.0;

    listPortfolio = new ArrayList<>();
    for (Object el : objects) {
      listPortfolio.add((ModelCoinPortfolio) el);
    }

    for (ModelCoinPortfolio el : listPortfolio) {
      allMoney +=
Double.valueOf(el.getModelCoinFirebase().getCount())
          * Double.valueOf(el.getModelCoin().getPrice());
    }
    textAllMoney.setText(
        String.valueOf(new
DecimalFormat("0.0").format(allMoney)) + " USD");
  }

  @Override
```

```
  public void onViewCreated(
      @NonNull View view, @Nullable Bundle
savedInstanceState) {
    listPortfolio = new ArrayList<>();
    portfolioBottomSheet = new PortfolioBottomSheet();
    progressBarPortfolio =
view.findViewById(R.id.progressBarPortfolio);
    progressBarPortfolio.setVisibility(View.VISIBLE);
    textAllMoney = view.findViewById(R.id.textAllMoney);
    textEmpty =
view.findViewById(R.id.textEmptyPortfolio);
    recyclerView =
view.findViewById(R.id.recyclerPortfolio);

    coinsAdapterPortfolio =
        new CoinsAdapterPortfolio(new ArrayList(),
getActivity(), this);
    recyclerView.setAdapter(coinsAdapterPortfolio);

    viewModelBottomSheet.getAllElementsFromFirebase();
  }

  //Очищення disposable від інтервалів
  @Override
  public void onDestroyView() {
    if (disposable != null)
      disposable.dispose();
    super.onDestroyView();
  }

  //Натиск на кнопку додаткової інформації
  @Override
  public void onClickMoreInfoPortfolio(int position) {
    Bundle bundle = new Bundle();
    bundle.putSerializable("coin",
listPortfolio.get(position));
    portfolioBottomSheet.setArguments(bundle);
    portfolioBottomSheet.show(
        getActivity().getSupportFragmentManager(),
"Portfolio");
  }
}
```

ДОДАТОК Г

Диск з розробленою програмою