Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка методу та засобів реалізації мобільної системи оцінки операцій з криптовалютою»

Виконав: студент IV курсу
групи 1ПІ-20б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Герасименко К. О.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Войтко В. В.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Колесник І. С.
(прізвище та ініціали)

Допущено до захисту
Зав. кафедри _____
«10» червня 2024 р.

ВНТУ – 2024

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
« 12 » березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Герасименку Кирилу Олеговичу

1. Тема роботи – «Розробка методу та засобів реалізації мобільної системи оцінки операцій з криптовалютою».

Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. №80.

2. Термін подання студентом роботи: 3 червня 2024 року.

3. Вихідні дані до роботи: середовище розробки – Visual Studio Code, система управління базами даних – Firebase; мова запитів – NoSQL; мови розробки – HTML, CSS, JavaScript/TypeScript; фреймворк React Native; операційна система – Windows 8/10/11, Android 11/12/13/14.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз розвитку мобільних систем для відстежування активів та постановка задач дослідження; розробка методу, моделі та алгоритмів роботи мобільної системи; розробка програмних модулів мобільної системи; тестування програми; висновки; список використаних джерел; додатки; ілюстративна частина.

5. Перелік ілюстративного матеріалу: титульний слайд – автор, науков керівник бакалаврської дипломної роботи, тема; мета, об'єкт та предм дослідження; задачі дослідження; наукова новизна; практична цінні отриманих результатів; блок-схема загального алгоритму роботи мобіль системи; блок-схема алгоритму керування сторінкою з котируванням; бл схема алгоритму користування портфелем; uml-діаграми; тестуван мобільної системи; апробація та публікації результатів роботи; впроваджен фінальний слайд.

6. Консультанти розділів бакалаврської дипломної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завданн прийня
1 – 4	Войтко В. В., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.2

7. Дата видачі завдання 13 березня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітк
1	Аналіз розвитку мобільних систем для відстежування активів та постановка задач дослідження	14.03.24 – 21.03.24	Виконо
2	Розробка методу, моделі та алгоритмів роботи мобільної системи	22.03.24 – 10.04.24	Виконан
3	Розробка програмних модулів мобільної системи	12.04.24 – 30.04.24	Виконан
4	Тестування програми	01.05.24 – 20.05.24	Виконан
5	Оформлення матеріалів до захисту БДР	22.05.24 – 30.05.24	Виконан

Студент _____ (підпис) **Герасименко К.** (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ (підпис) **Войтко В. В.** (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 59 сторінок формату А4, на яких є 45 рисунки, 3 таблиці, список використаних джерел містить 21 найменування.

У бакалаврській дипломній роботі було проведено аналіз сфери мобільних систем для відстежування активів. Проведено аналіз аналогів, визначено їх переваги і недоліки та доведено актуальність розробки власного мобільного сервісу.

Розроблена мобільна система дозволяє покращити моніторинг дій за криптоактивами шляхом розробки та використання мобільної системи для допомоги інвесторам, що дозволить зручно відстежувати усі операції та власні активи. Розроблено метод конвертації та додавання і віднімання активів з портфелю.

Мобільну систему було написано на мові програмування JavaScript/TypeScript, каскадних стилях CSS та мові гіпертекстової розмітки HTML. Середовищем розробки було обрано Visual Studio Code, а базою даних платформу Firebase.

У результаті виконання бакалаврської дипломної роботи отримано мобільну систему, що підвищить ефективність процесу моніторингу за криптоактивами.

Отримані в бакалаврській дипломній роботі результати можна використати для покращення процесу моніторингу криптовалюти.

Ключові слова: мобільна система, моніторинг криптоактивів

ABSTRACT

The bachelor’s thesis consists of 59 pages of A4 format, on which there are 45 figures, 3 tables, the list of used sources contains 21 names.

The bachelor thesis analyzed the field of mobile asset tracking systems. An analysis of analogues was carried out, their advantages and disadvantages were determined, and the relevance of developing one's own mobile service was proven.

The developed mobile system allows to improve the monitoring of crypto-asset activities by developing and using a mobile system to help investors, which will allow convenient monitoring of all transactions and own assets. A method of conversion and adding and subtracting assets from the portfolio has been developed.

The mobile system was written in JavaScript/TypeScript programming language, CSS cascading styles and HTML hypertext markup language. Visual Studio Code was chosen as the development environment, and the Firebase platform as the database.

As a result of completing the bachelor's thesis, a mobile system was obtained, which will increase the effectiveness of monitoring of crypto assets.

The results obtained in the bachelor thesis can be used to improve the cryptocurrency monitoring process.

Keywords: mobile system, monitoring of cryptoasset

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі криптовалютні технології набувають розвитку й поширення і стають невід'ємною складовою глобальної фінансової системи. Крім того, криптовалюти відкривають нові можливості для фінансової інклюзії, дозволяючи людям з доступом до Інтернету здійснювати фінансові операції без посередництва традиційних фінансових установ [1].

Розробка програмних модулів для реалізації мобільної системи оцінки операцій з криптовалютою, яке включає в себе кілька етапів розробки. Модулі повинні вміти ефективно управляти даними, обробляти актуальні котирування. Крім того, модулі повинні бути сумісними з конкретними вимогами мобільної системи та пристроєм користувача.

У цілому, криптовалютні технології продовжують змінювати ландшафт фінансової системи, прискорюючи децентралізацію та інновації. З часом їх роль може ще зрости, але також вони можуть зазнати викликів і змін у світлі вирішення технічних, регуляторних та екологічних проблем. Це зумовило потребу в програмних модулях, які можуть бути використані для реалізації таких систем. Ці програмні модулі можуть допомогти користувачам у створенні ефективних та безпечних транзакцій, сприяючи розвитку та упровадженню нових фінансових інструментів та сервісів, що відповідають потребам сучасного фінансового ринку.

Наявні аналогічні мобільні системи оцінки операцій з криптовалютою не надають достатньо досить потужного та корисного функціоналу. Відстежування криптоактивів одночасно з багатьох джерел є менш поширеними навіть серед вже існуючих рішень. До того ж, зазвичай існуючі рішення є платними.

Тому актуальною є розробка власної мобільної системи оцінки операцій з криптовалютою.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою дослідження є покращення моніторингу дій за своїми криптоактивами шляхом розробки та використання мобільної системи для допомоги інвесторам, що дозволить зручно відстежувати всі операції та власні активи.

Задачі дослідження:

- визначити найбільш якісне API для коректного відображення цін;
- розробити метод, модель, алгоритми роботи системи;
- розробити форму реєстрації, авторизації;
- реалізувати функцію створення власного портфелю;
- реалізувати функцію конвертації;
- розробити зручний та адаптивний графічний інтерфейс мобільної платформи;
- провести тестування мобільного застосунку згідно поставлених задач.

Об'єкт дослідження – процес розробки мобільної системи для оцінки операцій з криптовалютою.

Предмет дослідження – методи і засоби реалізації мобільної системи оцінки для операцій з криптовалютою.

Методи дослідження. У дослідженні було використано методи:

– методи порівняльного аналізу функціональних характеристик програм-аналогів для виявлення їх недоліків та постановки задач розробки;

– методи обробки та візуалізації даних для побудови графіків;

– методи захисту даних у Firebase для забезпечення захисту мобільної системи та даних користувачів;

– методи UI/UX для розробки та тестування інтерфейсу мобільної системи;

– методи тестування для перевірки працездатності роботи програми.

Новизна отриманих результатів:

1. Подальшого розвитку отримав метод формування графіку і конвертації криптовалюти, який, на відміну від існуючих, дозволяє при виборі криптовалюти паралельно візуалізувати обидва функціонали, що підвищує зручність і прозорість використання платформи та забезпечує можливість відстежувати еквівалентну

вартість обраної криптовалюти, легко перемикатися між різними криптовалютами та їх фіатними еквівалентами, здійснювати моніторинг і оцінку операцій.

2. Подальшого розвитку набула модель оцінки операцій з криптовалютою, яка, на відміну від існуючих, орієнтована на спрощення інтерфейсу та розширення функціоналу системи шляхом комплексного поєднання процесів моніторингу, аналізу, конвертації й оцінки операцій з криптовалютами, що підвищує зрозумілість моніторингових процесів і забезпечує зручність використання системи при відстеженні активів.

Практична цінність отриманих результатів. Практична цінність отриманих результатів полягає в тому, що розроблена у бакалаврській дипломній роботі мобільна система може використовуватись для покращення моніторингу дій за криптоактивами, стане в нагоді як досвідченим інвесторам, так і звичайним користувачам, що дозволить зручно відстежувати всі операції та власні активи.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У науковій праці [2], опублікованій у співавторстві, автору належать такі результати: таблиця порівняння функціональних характеристик аналогів, алгоритм роботи програми.

Апробація матеріалів бакалаврської дипломної роботи. Матеріали бакалаврської дипломної роботи доповідалися на конференції: «LIII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024)».

Публікації. Результати роботи були опубліковані в матеріалах конференції [2] – LIII Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2024).

Структура та обсяг роботи. Бакалаврська дипломна робота складається зі вступу, чотирьох розділів, висновків, списку використаних джерел, що містить 21 найменування, 4 додатків. Робота містить 16 ілюстрацій та 3 таблиці.

1 АНАЛІЗ РОЗВИТКУ МОБІЛЬНИХ СИСТЕМ ДЛЯ ВІДСТЕЖУВАННЯ АКТИВІВ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану мобільних систем для оцінки операцій з криптовалютою

Криптовалютні операції займають важливе місце в фінансовій сфері, надаючи користувачам можливість проводити безпечні та анонімні транзакції. Зростання популярності криптовалют викликає необхідність розвитку та вдосконалення інструментів для їх управління та операцій. Один із найбільш широко використовуваних інструментів – мобільні системи для операцій з криптовалютою. Цими системами виступають біржі, криптогаманці та defi. Ці системи пропонують зручний та мобільний спосіб керування криптовалютними активами.

Однією з основних переваг використання мобільних систем для операцій з криптовалютою є їх мобільність та доступність. Такі системи дозволяють користувачам здійснювати криптовалютні операції з будь-якого місця, де є доступ до мобільного зв'язку чи Інтернету [3]. Вони забезпечують зручний інтерфейс, який можна легко користувати на смартфонах та планшетах, що робить процес управління криптовалютними активами максимально зручним та ефективним для користувачів. Такий доступ забезпечує надійність та швидкість виконання транзакцій, а також забезпечує можливість оперативно реагувати на ринкові зміни та здійснювати управління фінансами в реальному часі.

Незважаючи на очевидні переваги мобільних систем для операцій з криптовалютою, варто враховувати і потенційні ризики, які можуть виникнути під час їх використання. Однією з таких потенційних проблем може бути недостатня прозорість та регулювання, що може призвести до виникнення сумнівів серед користувачів щодо надійності та безпеки використання мобільних систем для операцій з криптовалютою [4]. Крім того, існує ризик технічних помилок чи вразливостей, які можуть бути використані зловмисниками для кібератак або несанкціонованого доступу до особистої інформації користувачів. Тому важливою частиною розвитку мобільних систем для операцій з криптовалютою є

удосконалення їх технічної стійкості та захисту від потенційних загроз, що може забезпечити довіру користувачів та збільшити їх популярність у фінансовому середовищі.

Отже, розробка програмних модулів для впровадження мобільної системи оцінки операцій з криптовалютою є важливим кроком у вдосконаленні та розвитку фінансових інструментів. Враховуючи зростання популярності криптовалют та потребу користувачів у зручних та доступних способах управління криптовалютними активами, такі модулі можуть стати важливим інструментом для забезпечення ефективності та зручності у використанні. Проте, важливою частиною цього процесу є уважне урахування всіх можливих ризиків та впровадження відповідних заходів безпеки, щоб гарантувати захищеність та конфіденційність фінансових операцій для користувачів. Правильно розроблені та імплементовані програмні модулі можуть сприяти подальшому розвитку та поширенню криптовалют у фінансовому світі, забезпечуючи зручний та безпечний інструмент для управління цими активами.

1.2 Порівняльний аналіз аналогів

У цифровому світі існує ряд мобільних систем для операцій з криптовалютою, які надають користувачам можливість ефективно керувати своїми криптовалютними активами. Проте, перед вибором конкретної системи для використання, важливо провести детальний аналіз та порівняння різних аналогів. Такий підхід дозволить обрати найбільш оптимальне та відповідне рішення, враховуючи потреби та вимоги користувача. У цьому порівняльному аналізі будуть розглянуті ключові характеристики, переваги та недоліки різних мобільних систем для оцінки операцій з криптовалютою, щоб надати користувачам об'єктивну інформацію для прийняття інформованого рішення. До найбільш відомих рішень відносять:

- coinStats;
- cryptocompare;
- kubera.

Вважається найпопулярнішим застосунком серед аналогів. Застосунок CoinsStats для десктопних та мобільних пристроїв, що надає можливість користувачам відстежувати та аналізувати їх криптовалютні портфелі та ринкові дані [5] (рис. 1.1). Основні функції застосунка включають: відстеження портфеля, ринкові дані, сповіщення, портфельний аналіз.

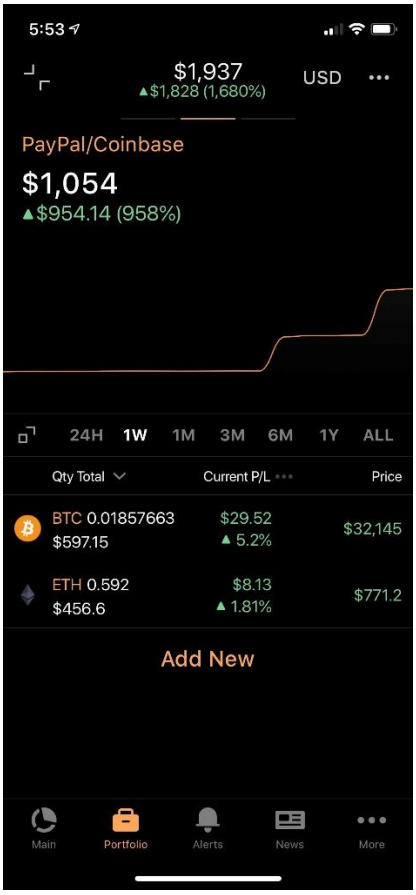

Рисунок 1.1 – Інтерфейс застосунку CoinStats

Інший приклад – застосунок Cryptocompare, вважається доволі не новим рішенням на сьогодні, має дві версії: десктопну та мобільну [6]. Виступає не тільки як трекер портфоліо, але і як повноцінна екосистема для криптовалют (рис. 1.2). Наприклад, користувачі мають можливість переглядати дані про ICO, такі як дати проведення, цінові пропозиції, обсяги продажів та інші важливі параметри, що допомагають їм приймати інформовані інвестиційні рішення.

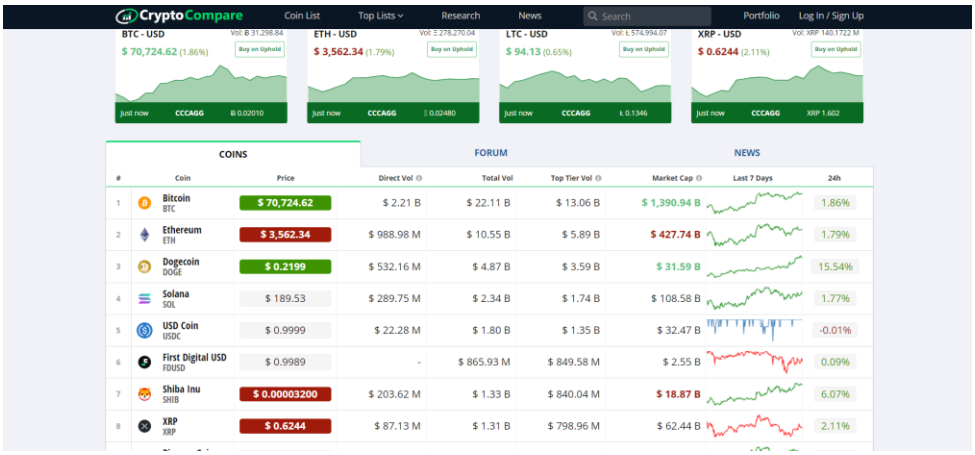

Рисунок 1.2 – Інтерфейс застосунку Cryptocompare

Наступний позиціонує себе як застосунок для професійних інвесторів, доступна не тільки десктопна версія, але і мобільна (рис. 1.3). Kubera пропонує детальне відстеження криптовалюти та DeFi майже будь-якого іншого типу активу чи облікового запису, який ви можете придумати — фіатні гроші, домени, біткойн ETF, предмети колекціонування, банківські рахунки, золото, NFT тощо [7]. Однією з найунікальніших функцій Kubera є керування бенефіціарами, яке дозволяє зберігати фінансові дані, планові документи та інші важливі файли в єдиному просторі. Потім ви можете назвати бенефіціара, який отримає доступ до всього вашого портфоліо, коли ви більше не зможете ним керувати.

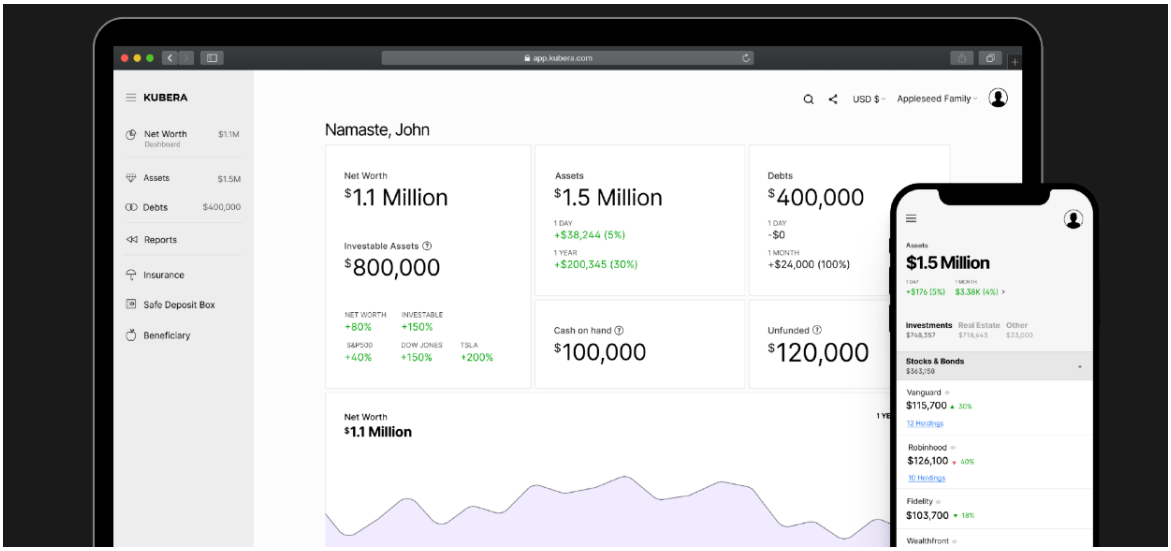

Рисунок 1.3 – Інтерфейс застосунку Kubera

Щоб наочно продемонструвати відмінності між цими додатками, була створена порівняльна таблиця (таблиця 1.1).

Таблиця 1.1 — Порівняльний аналіз аналогів

Критерії	CoinStats	CryptoCompare	Kubera	Власна розробка
Можливість додавання транзакцій вручну	-	+	-	+
Сучасний дизайн	+	-	+	+
Безкоштовна версія	+	+	-	+
Можливість створення облікового запису	+	+	+	+
Синхронізація даних між пристроями	+	+	+	+
Сумарний коефіцієнт	4	4	3	5

Відповідно до результатів таблиці порівняльних характеристик (табл. 1.1) розробка власної мобільної системи оцінки операцій з криптовалютою є доцільною. Отриманий продукт зможе покрити недоліки існуючих рішень та забезпечити новий рівень зручності та функціональності для користувачів.

Отже, в результаті проведеного аналізу аналогів було створено таблицю з порівнянням функціоналу (табл. 1.1), яка доводить актуальність власної розробки програмного додатку.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів реалізації мобільного застосунку для оцінки операцій з криптовалютою може включати ряд ключових кроків та модулів, які забезпечать функціональність та зручність використання.

У процесі створення програмного застосунку для оцінки операцій з криптовалютою, існує кілька можливих методів та інструментів, які можна використовувати. Один із можливих підходів – використання готових бібліотек або

фреймворків, таких як React Native [8], для розробки мобільного застосунку. Написання програмного застосунку на базі готового фреймворку, може забезпечити швидку розробку та високу кросплатформеність, оскільки додаток може бути запущений на різних мобільних платформах без значних змін у коді. Однак, варто врахувати, що використання готового фреймворку може призвести до обмежень у виборі функціональності та дизайну, що може не відповідати всім потребам та вимогам проекту.

Для забезпечення актуальних котирувань криптовалют у мобільному застосунку можна використовувати API [9] від популярних бірж. Це дозволить отримувати реальний часовий оновлення цін на криптовалюти, які потрібні для оцінки операцій. API може надавати доступ до різних типів даних, таких як ціни закриття, обсяги торгів, графіки цін, інформація про статистику криптовалют і багато іншого. При виборі API важливо враховувати його надійність, швидкість оновлення даних, обмеження по використанню та доступність необхідних даних. Деякі біржі надають безкоштовний доступ до своїх API з обмеженим функціоналом, в той час як інші можуть вимагати підписки або платну підтримку для доступу до розширених функцій та даних. Після вибору API важливо налаштувати правильну обробку та інтеграцію даних в мобільний застосунок. Це може включати регулярне оновлення даних, обробку помилок та відсутності з'єднання, а також відображення даних у зручному для користувача форматі, наприклад, у вигляді графіків або списків. Забезпечення актуальних котирувань через API дозволить користувачам отримувати найбільш об'єктивну та своєчасну інформацію про ціни на криптовалюти, що є важливим елементом для ефективної оцінки операцій та прийняття рішень.

Використовуючи API бірж та React Native, можна створити застосунок для створення свого криптопортфеля, який дозволить користувачам швидко та легко відстежувати свої активи. Цей застосунок може включати функції додавання та видалення різних криптовалютних активів, відстеження їхньої вартості в реальному часі, а також аналіз історії операцій. За допомогою React Native можна

створити привабливий та інтуїтивно зрозумілий інтерфейс користувача, який забезпечить легкий доступ до всіх функцій додатку.

1.4 Постановка задач дослідження для розробки мобільної системи оцінки операцій з криптовалютою

Проаналізувавши переваги та недоліки існуючих систем для оцінки операцій з криптовалютою було визначено наступні завдання, які необхідно виконати для розробки мобільної системи:

- визначити найбільш якісне API для коректного відображення цін;
- розробити метод, модель, алгоритми роботи системи;
- розробити форму реєстрації, авторизації;
- реалізувати функцію створення власного портфелю;
- реалізувати функцію конвертації;
- розробити зручний та адаптивний графічний інтерфейс мобільної платформи;
- провести тестування мобільного застосунку згідно поставлених задач.

1.5 Висновки

У першому розділі було розглянуто актуальний стан систем для оцінки операцій з криптовалютою. Було проведено порівняння відомих платформ для оцінки операцій з криптовалютою, таких як: CoinStats, CryptoCompare, Kubera.

У результаті порівняння було виявлено, що досліджувані платформи набули значної популярності серед учасників ринку криптовалют завдяки своїй ідеї та зручності. Кожна з них має свої переваги та недоліки, що вказує на те, що розробка власного модулю оцінки криптовалют є доцільною. На основі отриманої інформації було сформовано перелік задач, які необхідно виконати для розробки власної мобільної платформи.

2 РОЗРОБКА МЕТОДУ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ МОБІЛЬНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Для реалізації мобільної системи оцінки операцій з криптовалютою, буде використано React Native. React Native – це відкрита мобільна платформа для розробки крос-платформних мобільних додатків, яка розроблена компанією Facebook. Вона дозволяє розробникам використовувати JavaScript та React, щоб створювати мобільні додатки для iOS та Android з одним і тим же кодом. React Native використовує ті ж основні концепції, що і React, такі як компоненти та віртуальний DOM [10], але вона забезпечує доступ до API мобільних платформ, таких як камера, геолокація, повідомлення тощо. Це дозволяє розробникам створювати швидкі та ефективні мобільні додатки з використанням відомих інструментів і практик розробки мобільних застосунків.

Розробка програмних модулів для впровадження оцінки операцій з криптовалютою включає кілька ключових етапів, що вимагають уважного підходу та технічної компетентності. Першим кроком є створення модуля реєстрації та авторизації користувачів, яке забезпечить безпеку та конфіденційність їхніх особистих даних. Інформація про користувачів буде зберігатись у Firebase, забезпечуючи надійний доступ до неї. Після цього потрібно забезпечити синхронізацію даних. Це потрібно для відображення стану портфеля, який прив’язаний до облікового запису користувача. Для цього буде використовуватись API криптовалютних бірж для отримання актуальної інформації про транзакції та баланси. Наступним етапом є розробка алгоритмів для оцінки операцій з криптовалютою. Вони включають в себе аналіз цін криптовалют, обчислення вартості портфеля, розрахунок прибутків або збитків від операцій тощо.

Для реалізації цих модулів будуть використані різноманітні інструменти та бібліотеки, такі як Firebase, React Native, Android Studio та Visual Studio Code. Вони забезпечать надійну роботу системи, ефективне управління даними користувачів, швидку розробку та тестування, а також можливість створення інтерактивного та

зручного для використання інтерфейсу користувача. Використання таких інструментів дозволить зосередитися на головних аспектах розробки застосунку і забезпечить його успіх.

Отже, розробка програмних модулів реалізації мобільної системи для оцінки операцій з криптовалютою реалізованих за допомогою: Firebase та React Native є ключовим завданням, яке включає кілька етапів. Модулі повинні бути належно інтегровані з Firebase, щоб забезпечити ефективний обмін даними та забезпечити безпеку операцій. Також повинні бути відповідним чином реалізовані з React Native, щоб забезпечити оптимальну продуктивність та користувацький досвід на різних платформах.

2.2 Розробка моделі системи

Щоб краще проілюструвати, як працює застосунок, було вирішено створити його модель за допомогою UML-діаграми (рис. 2.1) [11]. Цей підхід дозволить детально показати всі компоненти системи та їх взаємодію, що сприятиме більш глибокому розумінню архітектури та функціонування застосунку.

Діаграма мобільної системи оцінки операцій з криптовалютою складається з двох частин, а саме з реєстрації та авторизації та основного функціоналу застосунку.

Спершу користувачеві пропонується зареєструватись, або якщо користувач вже зареєстрований, він може авторизуватись в системі, в іншому випадку система повідомить користувача про те що такий обліковий запис вже існує.

Після входу в систему у користувача на вибір є 3 сторінки:

1. Сторінка “Market” яка відповідає за моніторинг усіх криптоактивів і надає детальну інформацію щодо кожной криптовалюти зі списку, містить функції перегляду графіку та конвертації.

2. Сторінка “Portfolio” відповідає за управління та відстеження криптовалютного портфелю користувача. На цій сторінці користувач може додавати, видаляти та редагувати свої криптовалютні активи, а також переглядати їхню динаміку з часом.

3. Сторінка "Profile" відповідає за управління особистою інформацією користувача та налаштуваннями його облікового запису. На цій сторінці користувач може переглядати та редагувати свої особисті дані, такі як ім'я, електронна пошта, пароль тощо. Присутня технічна підтримка.

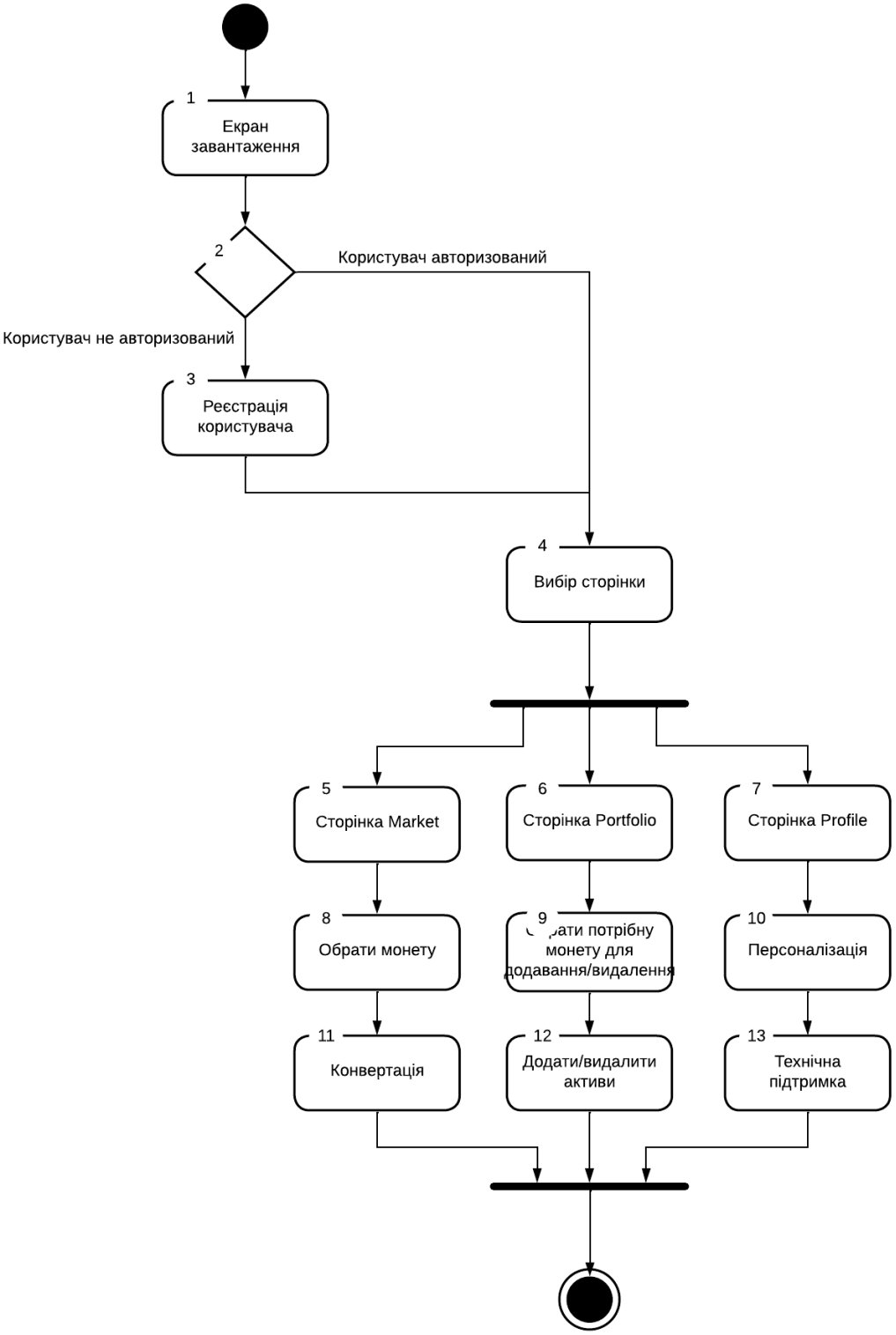

Рисунок 2.1 – Модель системи у вигляді діаграми діяльності

Ці три сторінки разом утворюють основний функціонал застосунку та надають користувачеві повний спектр можливостей для ефективного управління його криптовалютними активами та особистими налаштуваннями.

2.3 Розробка методу формування графіку і конвертації

Для зручного аналізу та моніторингу за криптоактивами користувачеві пропонується доволі широкий спектр послуг, як, наприклад, одночасний перегляд графіку та конвертації активів за поточним курсом, що дозволяє швидко приймати рішення щодо фінансів.

Графіки є невід'ємною частиною будь-якого аналізу та моніторингу криптоактивів, оскільки вони візуалізують динаміку цін і тенденції ринку. Графіки дозволяють користувачеві швидко оцінити рух цін за певний період часу, виявити ключові точки входу та виходу з позицій, а також спостерігати за патернами та формуванням трендів.

У свою чергу, доречність конвертації в цьому випадку полягає в можливості перегляду обміну криптоактивів за актуальним курсом, що дозволяє уникнути втрат від непередбачуваних коливань ринку та оцінки рентабельності активів на певний час. Така функціональність є особливо важливою для трейдерів, які прагнуть швидко реагувати на ринкові зміни та максимально використовувати можливості для отримання прибутку.

Комбінування перегляду графіків і можливості конвертації активів за поточним курсом робить процес аналізу та управління портфелем більш ефективним та простим для користувача. Такий підхід дозволяє оперативно реагувати на зміни ринку та приймати обгрунтовані рішення щодо оптимізації фінансових стратегій.

Метод формування графіків і конвертації дозволяє розмістити обидві функції при виборі певної криптовалюти, що демонструє більш детально і прозоро ситуацію щодо певного криптоактиву, сприяючи більш глибокому розумінню ринкових умов та збільшенню можливостей для успішних інвестиційних рішень.

Метод включає такий функціонал:

1. Авторизований користувач переходить на сторінку “Market”.

2. Користувач обирає певну криптовалюту в міру своєї зацікавленості.

3. При натисканні на певну криптовалюту користувач може спостерігати, як знизу екрану висувається біле спливаюче вікно, на якому, в свою чергу, розташовані графік і сама конвертація для певної криптовалюти.

4. Користувач може проаналізувавши, графік зручно оцінити кількість криптовалюти для подальшого формування портфелю.

5. Компонент BottomSheet використовується як плацдарм для розташування функцій графіку та конвертування на ньому. Він забезпечує зручний інтерфейс для користувача, дозволяючи легко доступатися до цих функцій у межах застосунку.

6. Метод handleConvert реалізує функцію конвертації, яка приймає вхідні дані у заданому форматі, обробляє їх відповідно до визначених правил та перетворює в необхідний вихідний формат. Для запуску цього методу користувачеві необхідно натиснути на кнопку, за яку відповідає метод CustomButton.

7. Метод LineChart використовується для побудови графіку монети яка відкрита користувачем на певний момент, data: об'єкт, який містить дані для побудови графіку.

8. Закриття вікна. Після використання всіх необхідних функцій користувач може закрити вікно, просто натиснувши на будь-яку зону поза межами самого вікна, що реалізується методом handlePressOutside.

Описаний метод включає кроки, які забезпечують зручність та ефективність взаємодії користувача з платформою, дозволяючи йому швидко та просто отримувати актуальну інформацію про ринок, аналізувати графіки та здійснювати конвертацію активів. Такий підхід сприяє не лише підвищенню рівня зручності використання платформи, але й покращує можливості користувача в управлінні своїм портфелем криптоактивів, допомагаючи зробити обґрунтовані та успішні інвестиційні рішення. Алгоритм роботи розробленого методу наведено на рис. 2.2.

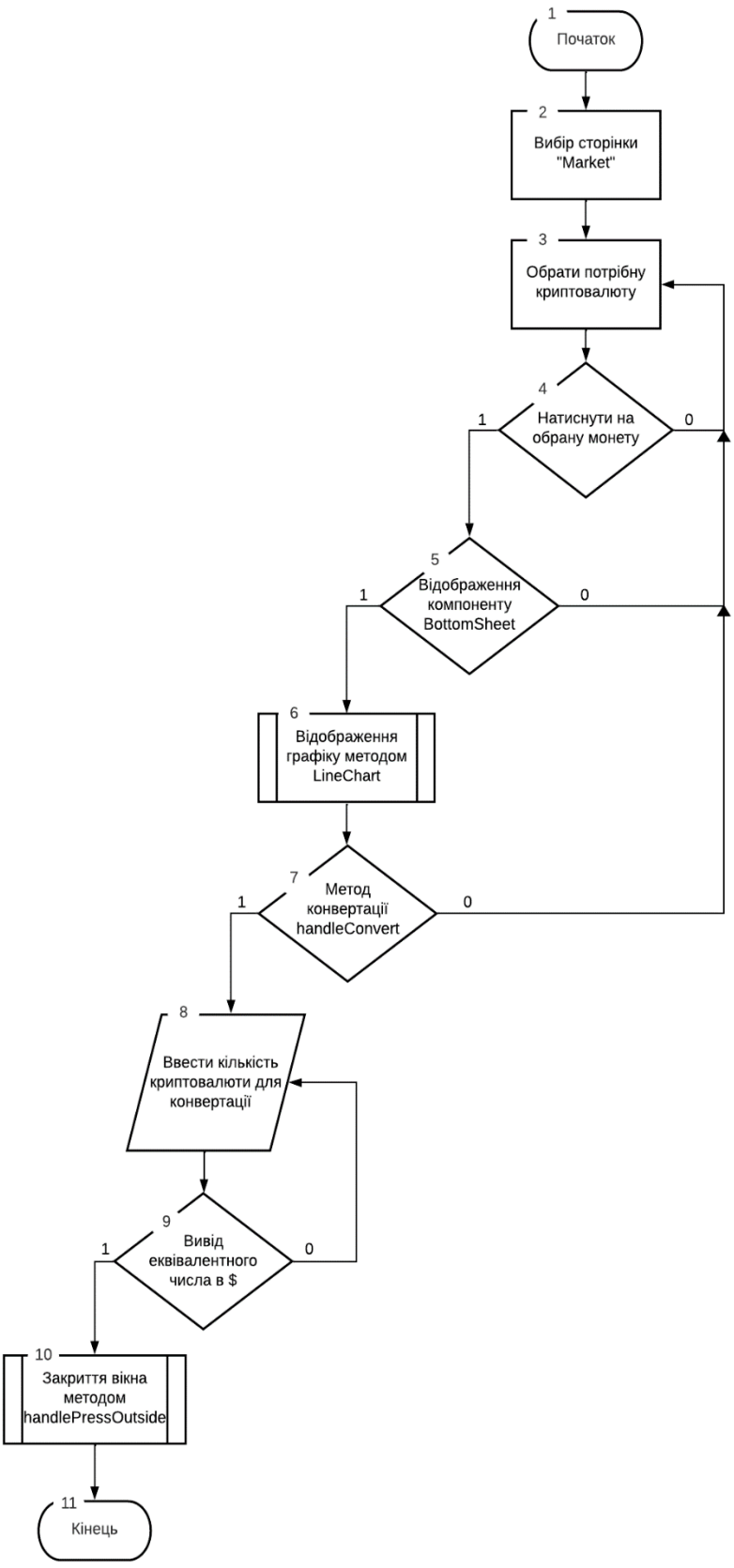

Рисунок 2.2 – Алгоритм роботи методу формування графіків і конвертації криптовалюти

2.4 Розробка алгоритмів роботи системи

Для успішної працездатності застосунку необхідно розробити алгоритми, за якими буде працювати система. Наведемо два розроблені алгоритми, за якими працюють скріни: Market, Portfolio.

Перший алгоритм описує дії стартової сторінки Market. Сенс сторінки полягає в тому, щоб відслідковувати актуальні котування популярних криптовалют з можливістю зручного конвертування та перегляду графіку при виборі певної монети (рис. 2.4).

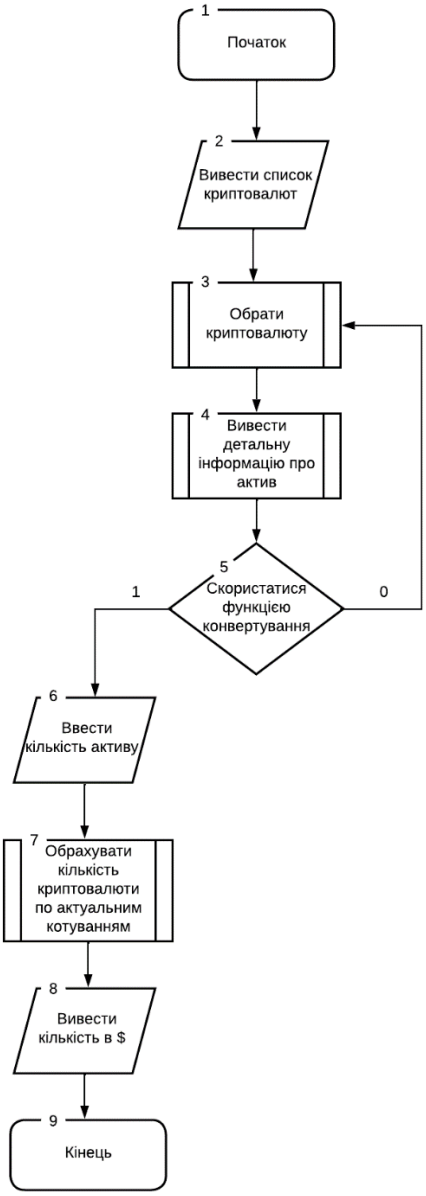

Рисунок 2.4 – Алгоритм дії сторінки Market

Другий алгоритм описує дії та функціонал для сторінки застосунку Portfolio, алгоритм демонструє дії та функціонал сторінки, а саме додавання або прибирання з портфеля своїх інвестицій, оновлення та відображення актуальної інформації щодо вартості портфеля (рис. 2.5).

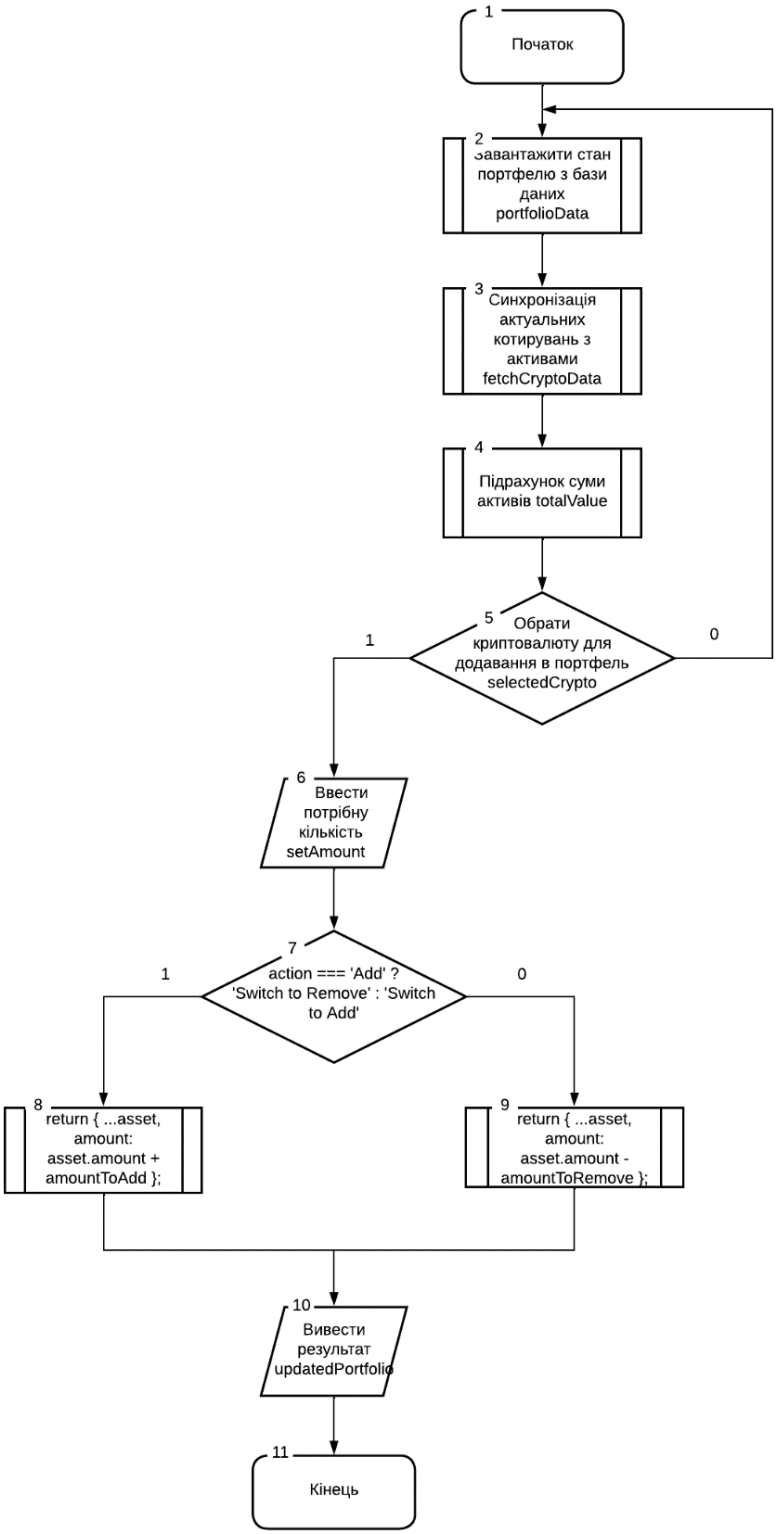

Рисунок 2.5 – Алгоритм роботи сторінки Portfolio

Запропоновані алгоритми визначають логіку роботи програми відповідно до її завдань і функціональності. Алгоритм Market відповідає за аналіз ринку та прийняття рішень щодо торгівельних операцій, тоді як алгоритм Portfolio відображає стратегії управління портфелем і розподіл активів. Кожен з цих алгоритмів має свої унікальні параметри та критерії, за якими система здійснює відповідні дії.

2.5 Висновки

У другому розділі було проведено аналіз вхідних та вихідних даних програмного продукту. Розглянута розробка методу формування графіків і конвертації та його функціональні можливості, створено блок-схему та графічну схему. Було розроблено два алгоритми для сторінок Market та Portfolio. Також було створено модель функціонування програмного продукту за допомогою UML діаграм.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ МОБІЛЬНОЇ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Один з важливих ключів успіху до побудови успішного та працездатного застосунку це вірно та влучно обрати мову програмування та середовища розробки, щоб максимально відповідати потребам проекту. Обираючи мову програмування, важливо враховувати характеристики проєкту, його масштаб, завдання, що потрібно вирішити, та відповідність мови цим вимогам.

JavaScript – це високорівнева скриптова мова програмування, яка широко використовується для створення інтерактивних застосунків і сайтів. JS додає анімацію і спливаючі повідомлення, валідує форми, оновлює контент і змушує інтерфейс реагувати на наші дії [12].

Firebase – це платформа для розробки мобільних та веб-додатків, яка надає набір інструментів та сервісів, спрощуючи роботу з серверною частиною додатка. Основна мета Firebase – забезпечити розробників зручними засобами для швидкого створення та масштабування додатків без необхідності великої інфраструктури [13].

React Native – це потужний інструмент для розробки мобільних додатків, який базується на React, одній з найпопулярніших бібліотек JavaScript для створення інтерфейсів користувачів. Використання React Native дозволяє розробникам створювати переносні мобільні додатки для платформ Android і iOS, використовуючи звичний для них синтаксис JavaScript.

Основні переваги використання фреймворку React Native:

- кросплатформенність;
- швидкість розробки;
- висока продуктивність;
- гнучкість та розширюваність.

Основні недоліки:

- обмежена підтримка нативного функціоналу;

- велика залежність від сторонніх бібліотек;

- нестабільність підтримки нових функцій.

Використання середовища Visual Studio Code при розробці продукту дозволяє розробникам зосередитися на написанні високоякісного коду завдяки його зручному інтерфейсу, багатофункціональним можливостям і широкому спектру розширень.

Visual Studio Code (VS Code) — це безкоштовний і відкритий редактор коду, розроблений компанією Microsoft [14]. Він доступний для операційних систем Windows, macOS і Linux і набув широкої популярності серед розробників завдяки своїм зручним інтерфейсом, потужним функціоналом та активній спільноті розробників, яка розробляє розширення для різноманітних мов програмування та технологій. VS Code підтримує вбудовану підсвітку синтаксису для багатьох мов програмування, автодоповнення, відлагодження коду, інтеграцію з системами контролю версій, вбудований термінал та багато іншого. Він став незамінним інструментом для розробників у різних галузях, від програмної розробки до штучного інтелекту (рис. 3.1).

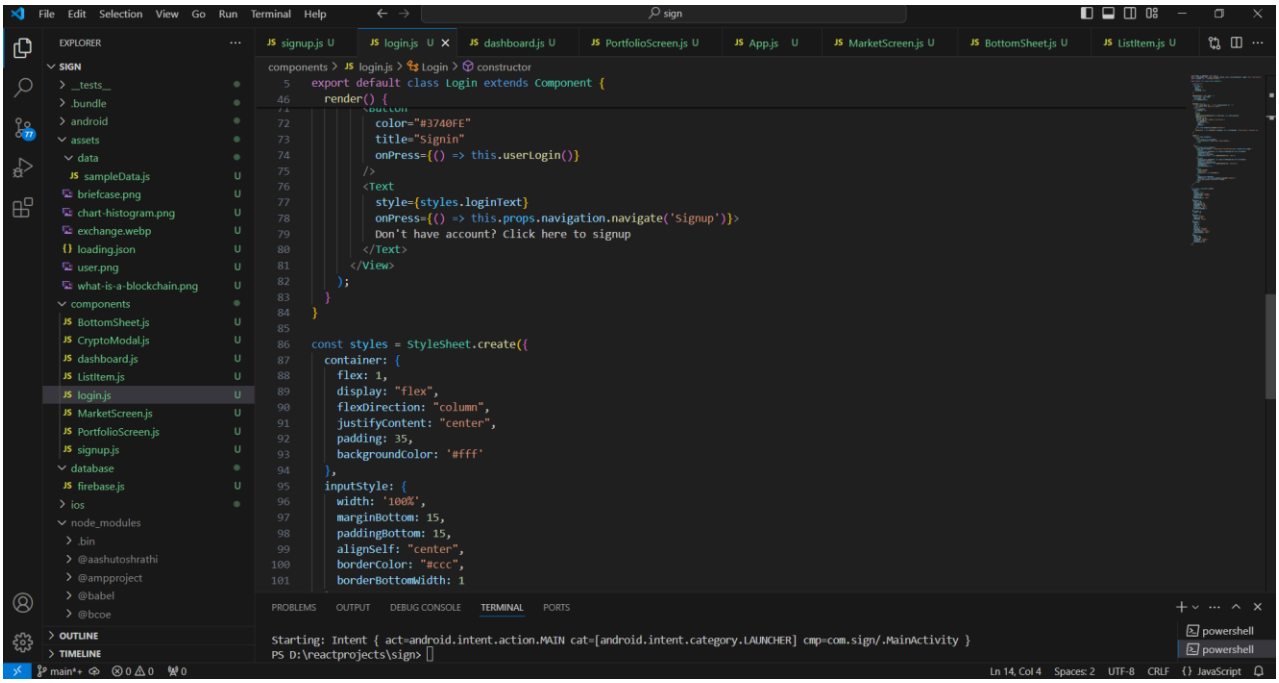

Рисунок 3.1 – Інтерфейс програмного середовища VS Code

Також при розробці мобільного застосунку варто використовувати емулятори.

Емулятори – це програмне забезпечення, яке дозволяє емулювати роботу мобільної операційної системи на комп'ютері. Це корисно для розробників програмного забезпечення, тестирування програм або просто для того, щоб мати можливість запускати мобільні додатки на більшому екрані. Один з таких є емулятор від Android Studio. Android Studio – це інтегроване середовище розробки (IDE) для платформи Android, розроблене компанією Google [15]. Це основний інструмент для розробки мобільних додатків для Android (рис. 3.2).

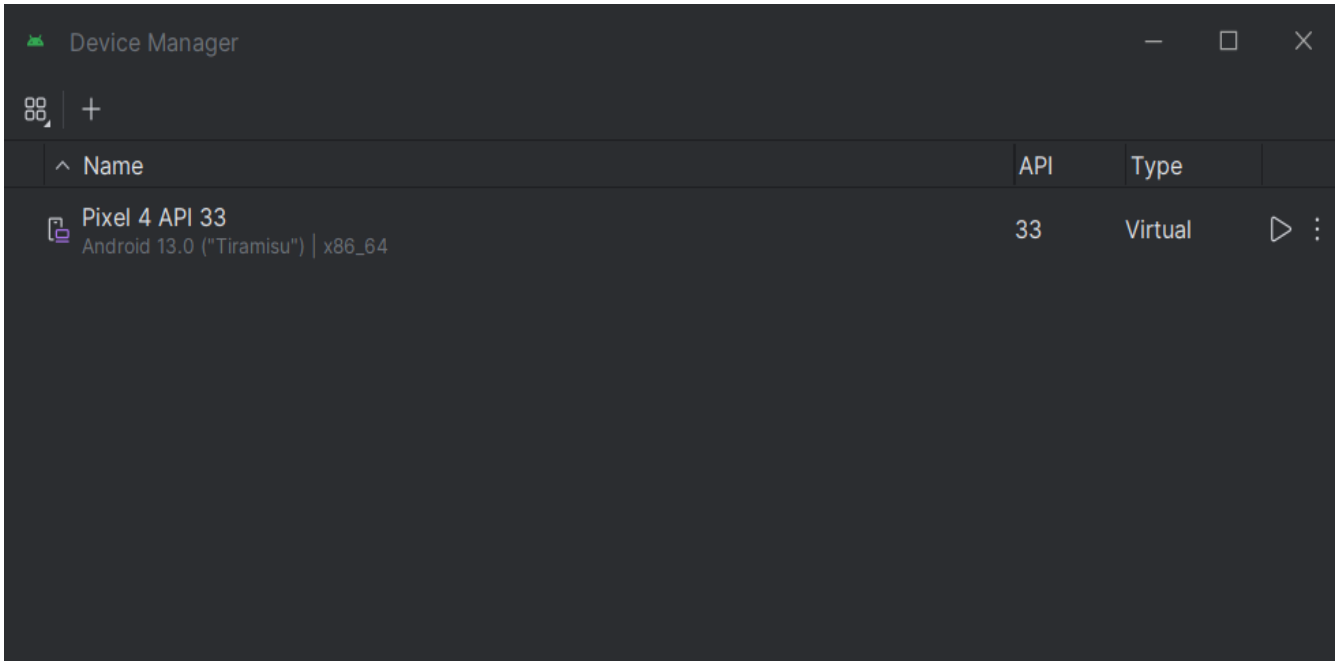

Рисунок 3.2 – Головне меню для запуску емулятора

Всі описані вище продукти дозволяють розробити мобільний застосунок, який забезпечує створення мобільної системи оцінки операцій з криптовалютою та задовольняють усі потреби для розробника.

Отже, обираючи інструменти для розробки мобільного застосунку, важливо враховувати ряд чинників, які включають в себе потреби проекту, відомості про програмування та специфіку платформи. React Native може бути прекрасним вибором для розробки кросплатформенних додатків, забезпечуючи швидкість розробки та високу продуктивність завдяки використанню JavaScript.

3.2 Розробка модулів для реєстрації та авторизації

Одним із головних модулів застосунку виступають модулі реєстрації та авторизації що забезпечують безпеку та персоналізацію взаємодії користувачів з системою. Модуль реєстрації дозволяє користувачам створювати облікові записи, заповнюючи необхідну інформацію та підтверджуючи її за допомогою електронної пошти або SMS-повідомлення (рис. 3.3). Після реєстрації користувач може авторизуватися в системі, використовуючи введені раніше дані або інші методи перевірки особи.

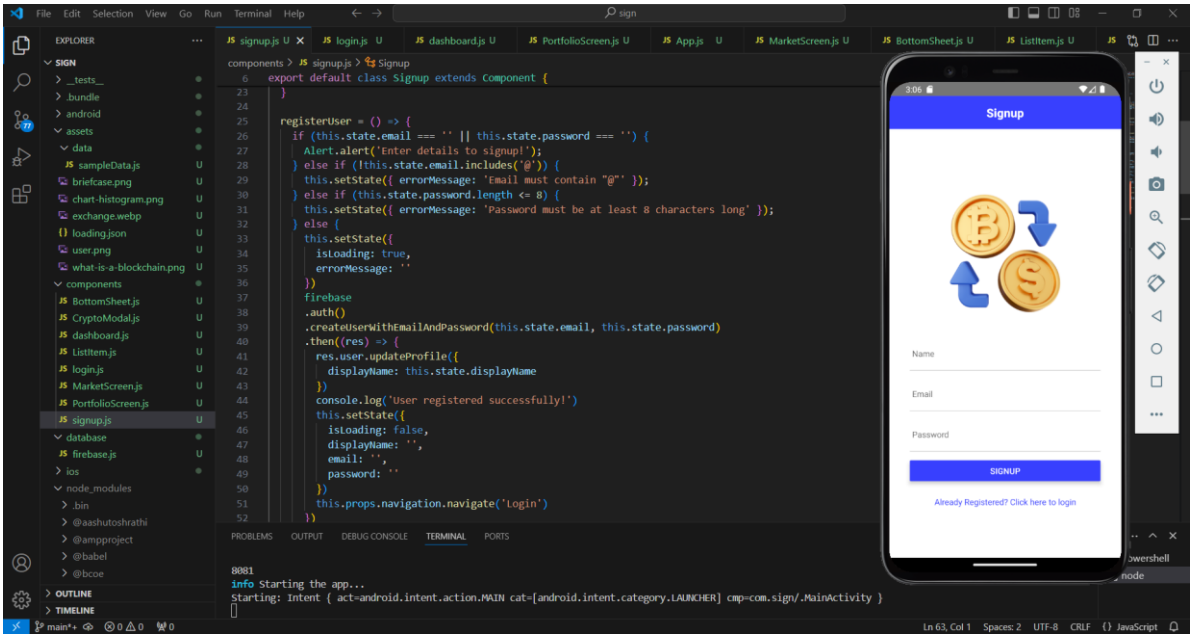

Рисунок 3.3 – Частина коду для виконання реєстрації

Основні цілі блоку реєстрації включають:

1. Забезпечення введення користувачем необхідних даних. Користувач вводить свої дані (наприклад, email та пароль) на екрані реєстрації.

2. Перевірка введених даних. Після натискання користувачем кнопки "Signup", введені дані проходять валідацію на наявність помилок чи відсутніх полів. Якщо будь-яке з полів порожнє, користувачу показується повідомлення: Alert.alert('Enter details to signup!').

3. Взаємодія з сервером для створення нового облікового запису. Після успішної валідації, дані передаються на сервер (в даному випадку, через Firebase) для створення нового облікового запису.

4. Обробка відповіді від сервера. В залежності від результату запиту, користувачу надається відповідний зворотний зв'язок. У разі успішної реєстрації Firebase повертає позитивну відповідь, після чого вхідні поля форми очищуються, і користувач перенаправляється на екран входу. Якщо реєстрація не вдалася, Firebase повертає помилку, і користувачу показується відповідне повідомлення про помилку (рис. 3.4).

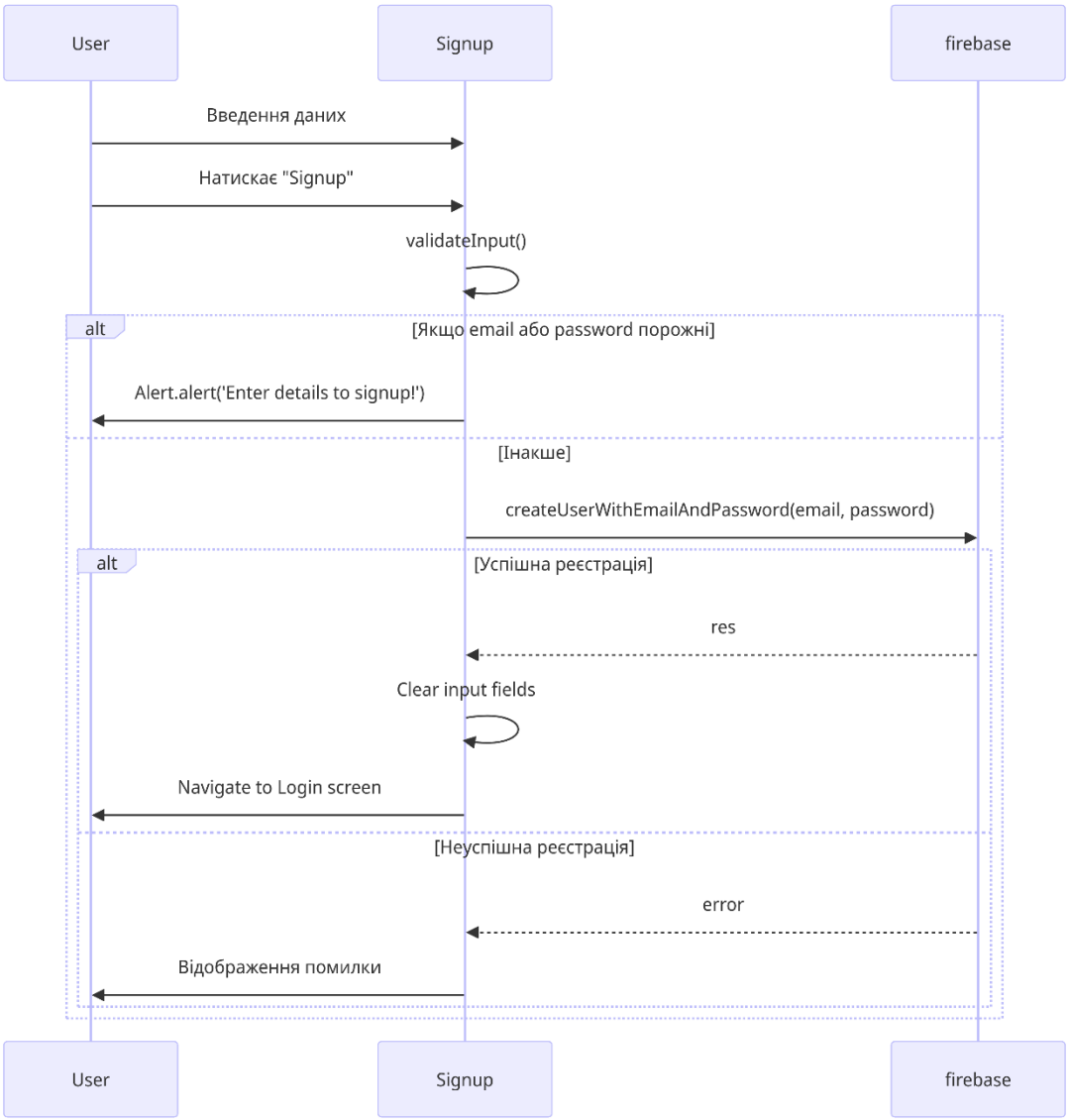

Рисунок 3.4 – Діаграма послідовності блоку реєстрації

Модуль авторизації є важливим компонентом інформаційної безпеки, що дозволяє контролювати доступ користувачів до ресурсів системи. Після успішної аутентифікації, коли особа користувача вже підтверджена, відбувається процес

авторизації, що визначає, до яких саме ресурсів та функцій системи цей користувач має доступ (рис. 3.5).

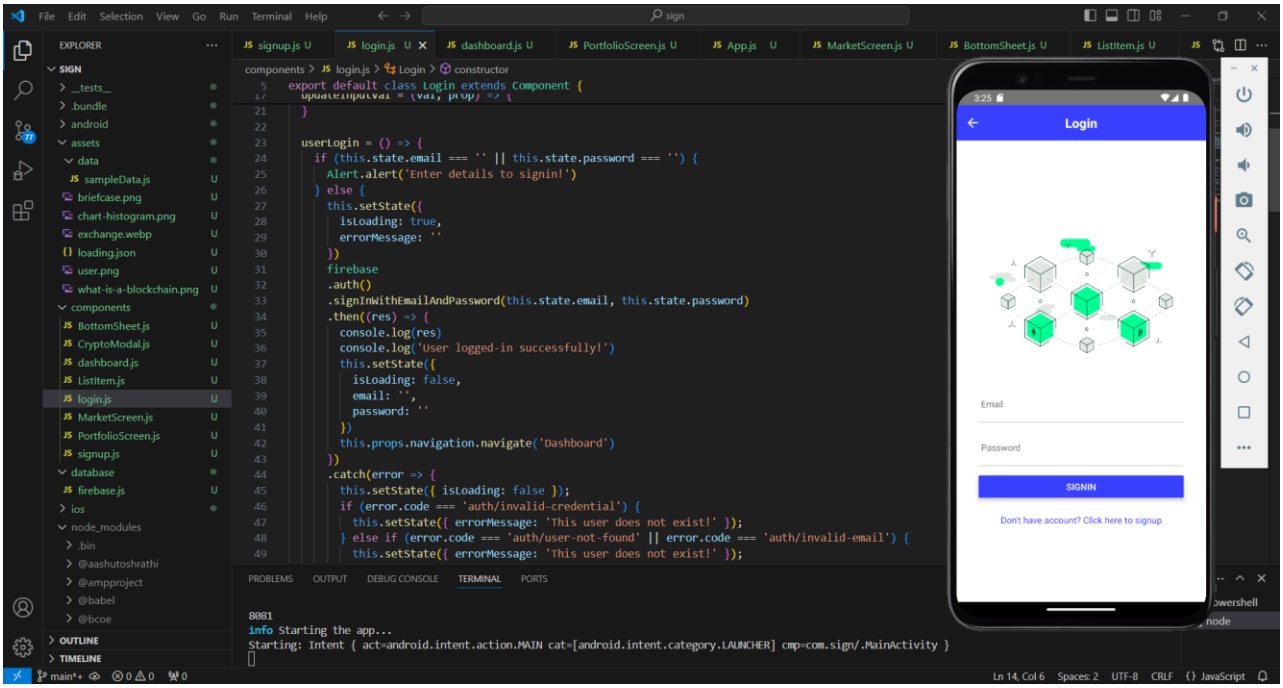

Рисунок 3.5 – Частина коду для виконання авторизації

Основні цілі блоку авторизації включають:

1. Перевірка введених даних користувача. Користувач вводить свої облікові дані, такі як електронна пошта та пароль.

2. Обробка введених даних. Після натискання кнопки "Signin" починається процес авторизації, викликаючи функцію userLogin().

3. Перевірка заповненості полів. Якщо будь-яке з полів електронної пошти або пароля порожнє, користувач отримує повідомлення про помилку з вимогою ввести всі необхідні дані.

4. Виконання запиту на авторизацію. Якщо всі поля заповнені, виконується запит до сервісу Firebase з використанням функції signInWithEmailAndPassword(email, password).

5. Обробка результатів авторизації:

5.1. Успішна авторизація. Якщо авторизація пройшла успішно, Firebase повертає відповідь (res), і користувач отримує повідомлення про успішну авторизацію.

5.2. Неуспішна авторизація. Якщо авторизація не вдалася, Firebase повертає помилку (error), і користувач отримує повідомлення про невдалу спробу входу (рис. 3.6).

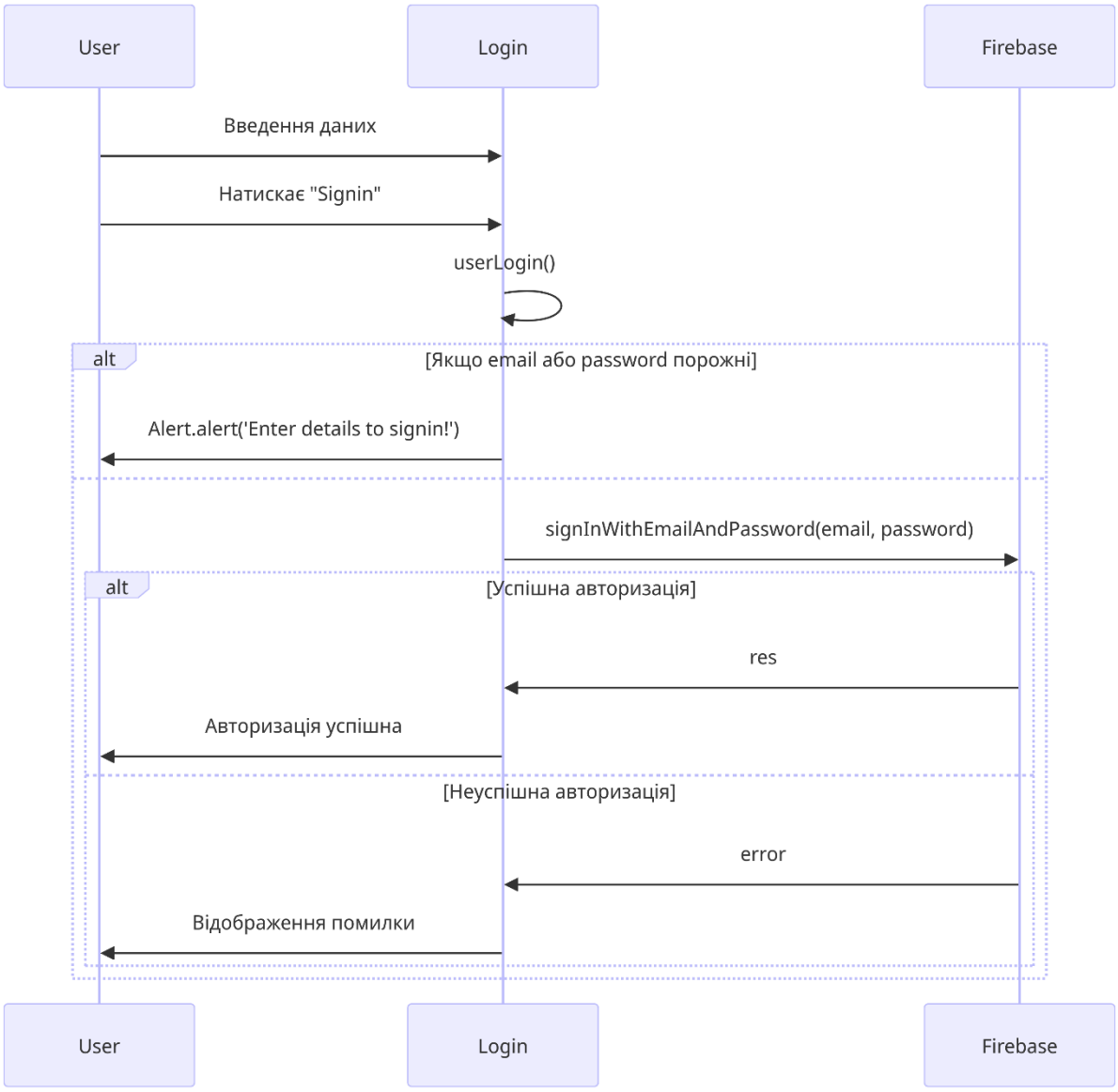

Рисунок 3.6 – Діаграма послідовності блоку авторизації

Методи реєстрації та авторизації здійснюється за допомогою FireBase, що генерує необхідні дані для створення власної бази даних та забезпечує безпечний доступ до них. Цей інструмент, який не лише спрощує процес реєстрації та авторизації, але й забезпечує надійне зберігання даних користувачів. Його функціонал дозволяє ефективно керувати даними, включаючи забезпечення конфіденційності та захист від несанкціонованого доступу (рис. 3.7).

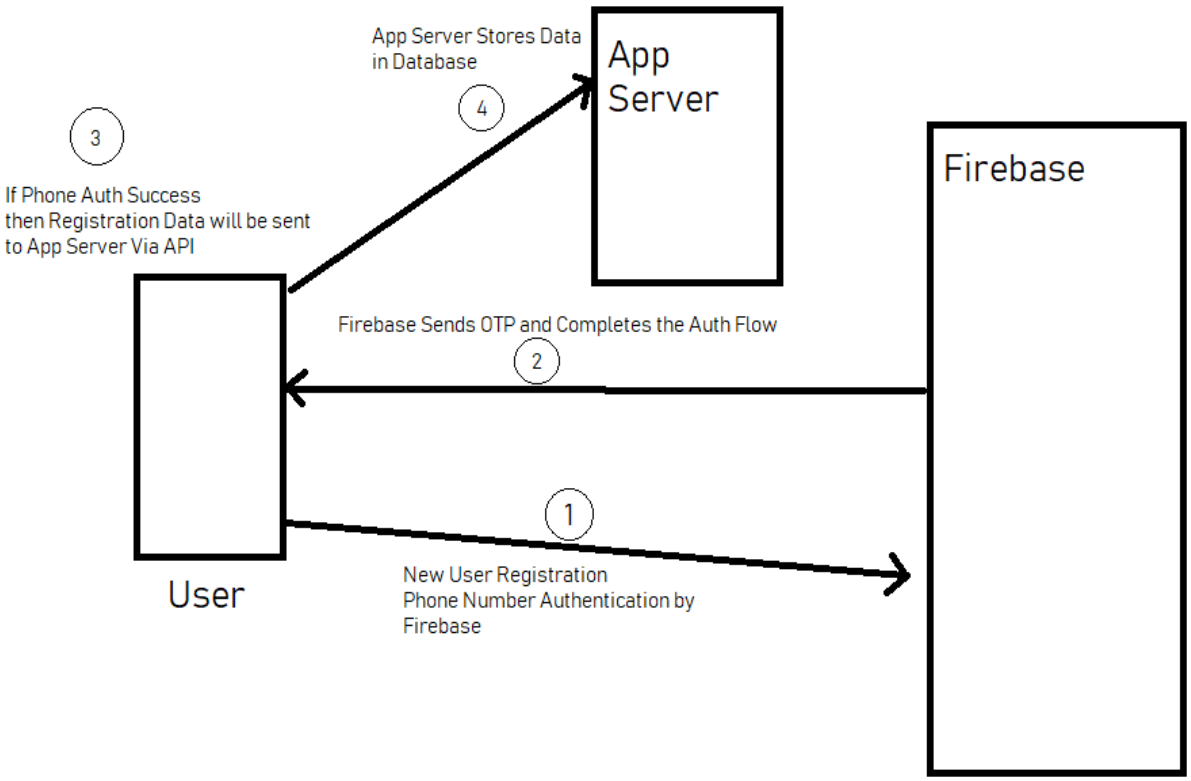

Рисунок 3.7 – Діаграма послідовності дій роботи Firebase

Конфігурація Firebase: Об'єкт firebaseConfig містить налаштування, необхідні для ініціалізації Firebase у мобільному застосунку. Ці налаштування включають ключ API, домен аутентифікації, ідентифікатор проекту, сховище, ідентифікатор відправника повідомлень та ідентифікатор додатку. Ці налаштування специфічні для проекту і їх можна знайти у консолі Firebase.

3.3 Розробка головних модулів застосунку

Найперший функціонал який був реалізований в мобільній системі – це правильне отримання даних про котирування, оскільки весь функціонал застосунку залежить від імпорту правильного API. Тому спершу було вирішено побудувати сторінку Market, де розташовані актуальні криптовалюти, їх котирування та графік змін курсів. Ця сторінка відображає основну інформацію про криптовалюти, яка є ключовою для будь-якого трейдера чи інвестора.

Функція fetchChartData виконує HTTP-запит до API для отримання даних графіка криптовалюти на основі її ідентифікатора. Отримані дані перетворюються у форматі дати та ціни, після чого повертаються (рис. 3.8).

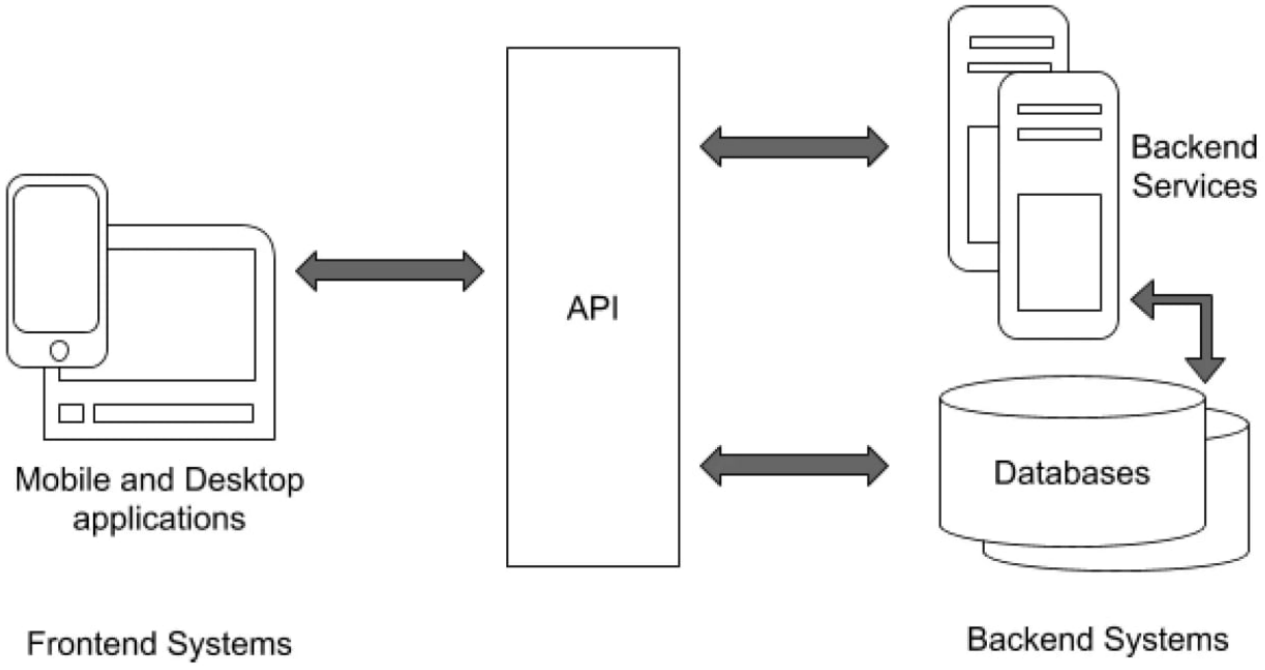

Рисунок 3.8 – Діаграма обробки даних

В свою чергу щоб переглянути детальну інформацію про монету, застосовується компонента BottomSheet яка відображає аркуш з детальною інформацією. У BottomSheet розташована конвертація, що дозволяє обрати любу криптовалюту та переглянути певну кількість активів для подальших дій з ними, за це відповідає handleConvert (рис. 3.9).

```
const handleConvert = () => {
  if (amount.trim() === '' || isNaN(parseFloat(amount))) {
    alert('Please enter a valid number.');
    return;
  }
  const convertedValue = parseFloat(amount) * parseFloat(currentPrice);
  setConvertedAmount(convertedValue.toFixed(2));
};
```

Рисунок 3.9 – handleConvert

Також присутній графік, який відіграє важливу роль в інтерфейсі користувача. Функція, яка відповідає за відображення графіка, розташована в компоненті BottomSheet.

Зокрема, графік створюється за допомогою компонента LineChart з бібліотеки react-native-chart-kit. Ця бібліотека забезпечує простий спосіб інтеграції різноманітних графіків та діаграм у застосунок на платформі React Native.

Діаграму прецедентів сторінки “Market” наведено на рисунку 3.10, спроєктований графічний інтерфейс на рисунку 3.11.

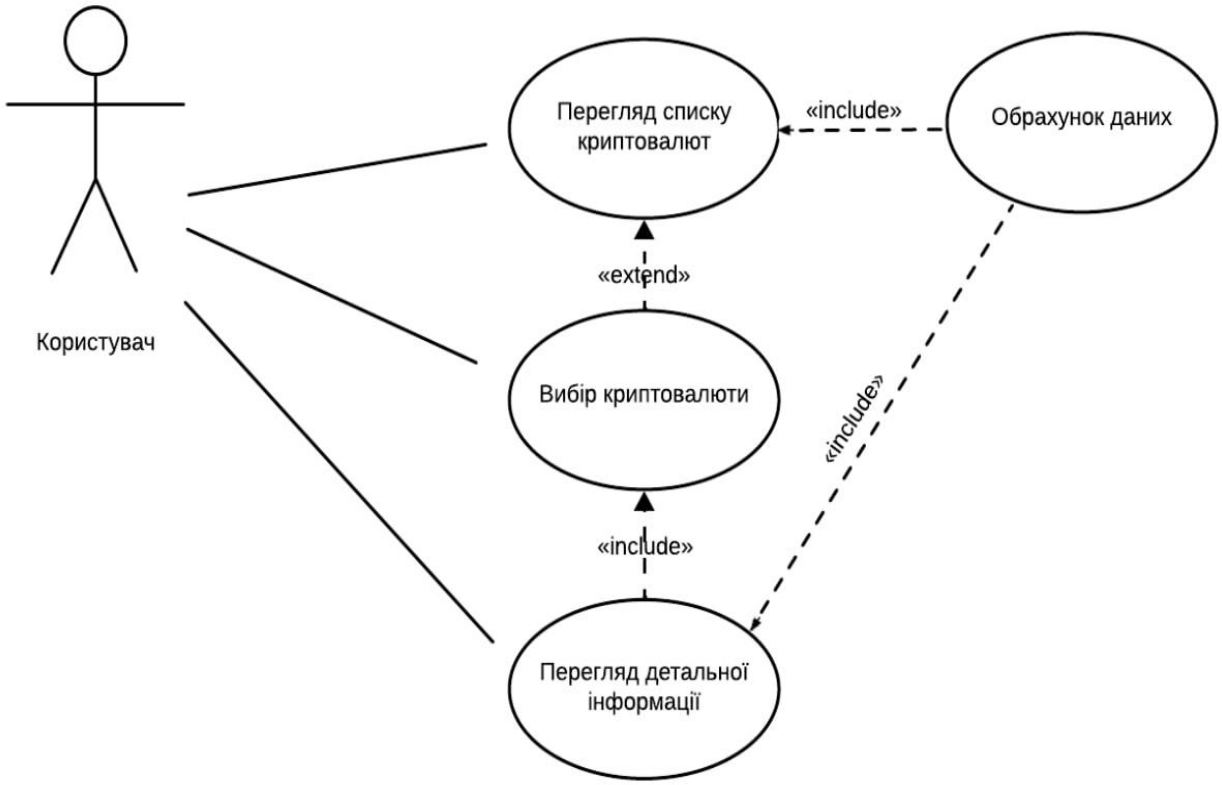

Рисунок 3.10 – Діаграма прецедентів сторінки “Market”

Попередньо для створення програмного графічного інтерфейсу необхідно спроєктувати відповідну графічну схему. Вона включає в себе визначення основних компонентів інтерфейсу, їх розташування на екрані та взаємодію між ними.

Рисунок 3.11 – Графічна схема інтерфейсу формування графіків і конвертації

Для реалізації наступної сторінки Portfolio у програмі використовується компонент під назвою PortfolioScreen. Основний функціонал сторінки включає в себе відображення графіка зміни вартості портфоліо в часі, списку активів портфоліо, введення кількості та вибору криптовалюти для додавання або видалення.

Основна функція handleAction оновлює портфель користувача. Вона створює новий масив updatedPortfolio, де змінюється кількість активу вибраної криптовалюти в залежності від обраної дії Add або Remove. Після цього оновлює стан portfolioData за допомогою setPortfolioData(updatedPortfolio). Діаграму прецедентів сторінки “Portfolio” наведено на рисунку 3.12.

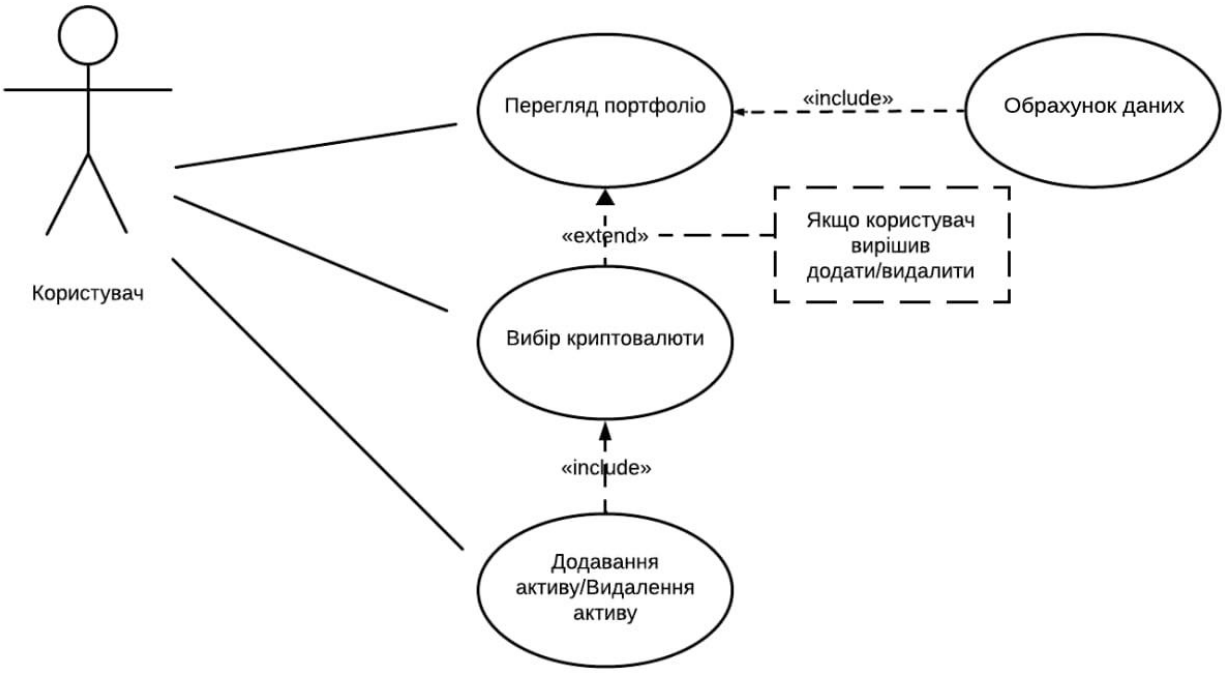

Рисунок 3.12 – Діаграма прецедентів сторінки “Portfolio”

Для побудови останньої сторінки Profile, потрібно створити компонент dashboard, який буде відображати інформацію про користувача. Цей компонент буде відповідальний за відображення особистих даних користувача, таких як ім'я, адреса електронної пошти тощо.

Метод componentDidMount здійснюється перевірка, чи існує поточний автентифікований користувач за допомогою Firebase Authentication (рис. 3.13). Якщо користувача знайдено, дані про його ім'я та унікальний ідентифікатор (displayName та uid) отримуються з об'єкта користувача і встановлюються у відповідні поля стану компонента за допомогою методу setState.

```
componentDidMount() {
  const user = firebase.auth().currentUser;
  if (user) {
    this.setState({
      displayName: user.displayName || '',
      uid: user.uid || ''
    });
  }
}
```

Рисунок 3.13 – ComponentDidMount

Зважаючи на це, система повинна безперебійно функціонувати навіть в умовах високого навантаження і забезпечувати надійну безпеку даних.

3.4 Розробка структури та інтерфейсу мобільного застосунку

Розробку самого інтерфейсу було вирішено зробити максимально простою, щоб зосередити увагу користувача на головному. Простий інтерфейс зменшує візуальний шум та полегшує навігацію, дозволяючи користувачеві швидко знаходити потрібну інформацію та виконувати необхідні дії.

Основними кольорами застосунку було обрано синій та білий кольори, оскільки ці кольори асоціюються з чистотою, спокоєм і довірою. Синій колір часто пов'язують із технологіями та професійністю. Білий колір, у свою чергу, відображає чистоту, простоту і мінімалізм. Використання цих кольорів у дизайні інтерфейсу допомагає забезпечити чіткість, легкість сприйняття та спокій для користувача, дозволяючи йому зосередитися на основних функціях та завданнях застосунку.

При першому запуску застосунку користувач бачить завантажувальний екран, на якому відображається анімація (рис. 3.14). Цей початковий етап є критично важливим, оскільки створює перше враження про програму та задає тон для подальшої роботи з нею. Тривалість завантаження може варіюватися в залежності від швидкості Інтернет-з'єднання та технічних характеристик пристрою. Під час завантаження на екрані зазвичай показується логотип або інші ідентифікуючі елементи застосунку, що допомагає користувачеві сформувати уявлення про його характер і основні функції. Якщо завантаження займає деякий час, важливо надавати користувачеві зворотний зв'язок щодо статусу цього процесу. Це можна зробити за допомогою анімації або текстових повідомлень, щоб уникнути ситуацій, коли користувач може подумати, що програма не працює або є недоступною.

Таким чином, ретельно продуманий дизайн інтерфейсу та правильне використання кольорів не тільки покращують естетичну привабливість застосунку, але й підвищують його функціональність, роблячи користування більш приємним і зручним.

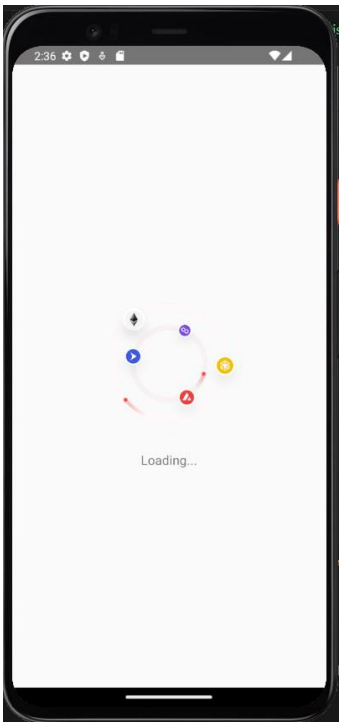

Рисунок 3.14 – Екран завантаження

Так як застосунок позиціонує себе для особистого оцінювання криптовалюти, для застосунку було передбачено реєстрацію та авторизацію. На цьому екрані користувач системи має ввести свої особисті дані, а саме ім’я, пошту та пароль для змоги користування застосунком (рис. 3.15).

Рисунок 3.15 – Екран реєстрації користувача

Коли профіль успішно зареєстровано, користувач перенаправляється на екран авторизації, де він повинен повторно ввести свої особисті дані для входу в систему (рис. 3.16). Цей етап забезпечує безпеку та захист інформації користувача, дозволяючи перевірити його ідентичність перед наданням доступу до особистого облікового запису.

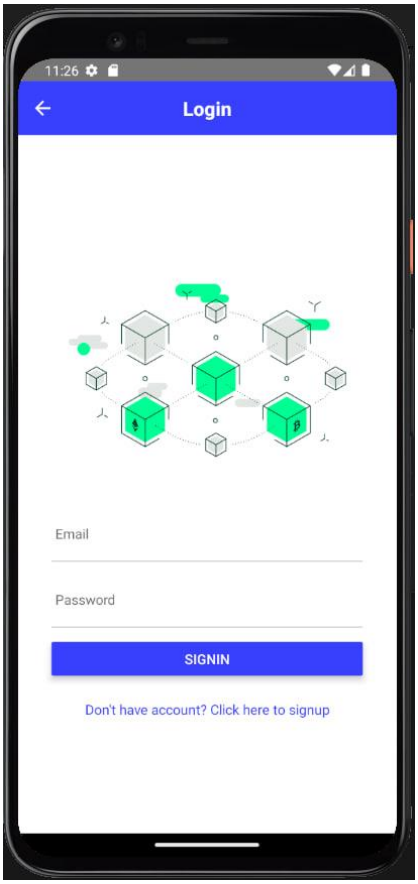

Рисунок 3.16 – Сторінка авторизації користувача

Після успішної авторизації, перше, що бачить користувач, це ринок криптовалют, де розташовані найпопулярніші криптовалюти разом з їх поточними цінами та графіками змін (рис. 3.17). Це дає користувачеві можливість швидко оцінити стан ринку та зробити відповідні рішення щодо купівлі, продажу або обміну криптовалют. Для актуальних котирувань було прийнято рішення про використання API від компанії CoinGecko [16], яка надає надійні дані про ціни та інші параметри для різних криптовалют. Це забезпечує користувачам точні та достовірні дані, необхідні для прийняття інформованих рішень на ринку криптовалют.

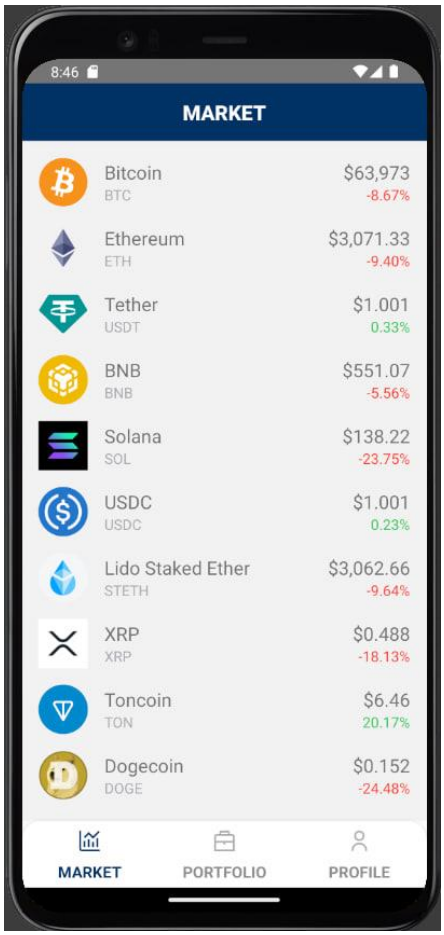

Рисунок 3.17 – Інтерфейс сторінки ринку криптовалют

Для більшого ознайомлення з криптовалютою можна натиснути на одну з перелічених та побачити ціну, графік в реальному часі (рис. 3.18). Це дозволяє краще розуміти динаміку ціни та ринкові тенденції. Багато платформ також надають можливість аналізувати технічні показники, що допомагає інвесторам приймати обгрунтовані рішення. Такі інструменти допомагають краще зрозуміти ринок криптовалют та зменшити ризики при торгівлі.

Також присутня функція конвертації при відкритті однієї з криптовалют, можна конвертувати будь-яку суму в долари, це допомогає користувачам легко здійснювати обмін криптовалют на фіатні кошти та забезпечує більшу гнучкість в управлінні їх портфелем. Ця функція дозволяє швидко реагувати на зміни на ринку, забезпечуючи можливість перетворити криптовалютні активи в стабільніші активи, такі як долари, коли це необхідно.

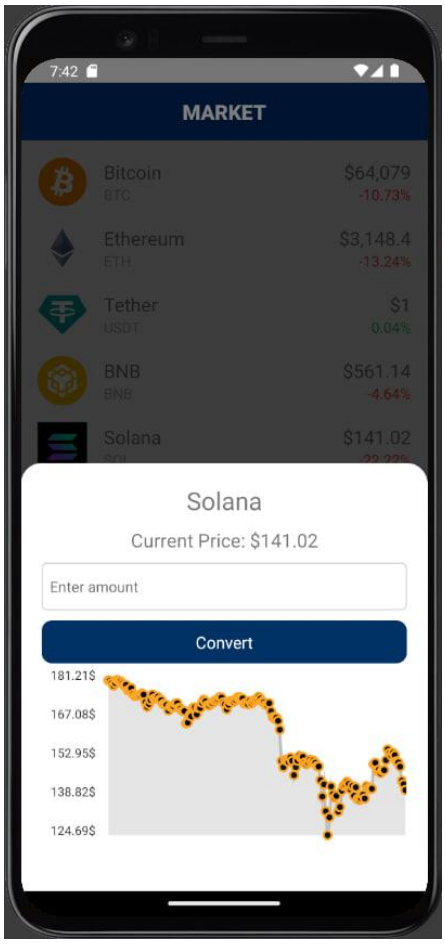

Рисунок 3.18 – Детальна інформація про монету

На скріні Portfolio можемо побачити активи, які знаходяться у портфоліо і керувати ними за допомогою кнопок “add” та “remove”. Ці кнопки дозволяють додавати нові активи до портфоліо або видаляти вже існуючі. Стан портфоліо оновлюється в реальному часі за допомогою API від CoinGecko, що дозволяє користувачеві завжди бачити актуальну вартість свого портфоліо. Крім того, у верхній частині екрана відображається “Total Value” – повна сума усіх активів у портфоліо, що допомагає зрозуміти загальну фінансову ситуацію (рис. 3.19).

Графік, розміщений на екрані, ілюструє загальну тенденцію приросту або занепаду портфоліо. Він відображає динаміку змін вартості портфоліо протягом певного періоду часу, допомагаючи користувачеві аналізувати його ефективність та приймати обгрунтовані рішення щодо інвестицій. Графік може бути корисним інструментом для виявлення тенденцій та прогнозування майбутнього розвитку портфоліо.

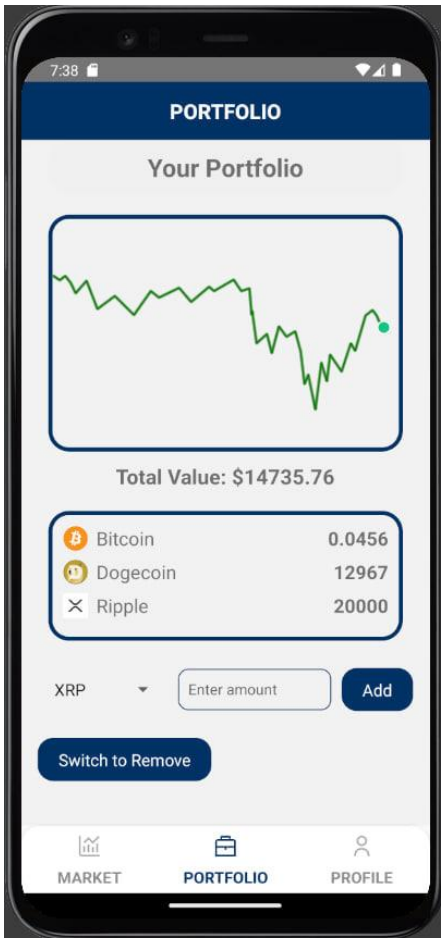

Рисунок 3.19 – Інтерфейс сторінки Portfolio

На скріні Profile можна знайти усю персональну інформацію для перегляду, а також подальшого редагування, якщо воно необхідне. Присутня можливість сповіщень, які необхідні для того, щоб інформувати користувача про різкий ріст або падіння на ринку. Ця функція дуже корисна, адже може врятувати чимало коштів та допомогти приймати обґрунтовані рішення щодо інвестування.

Також є технічна підтримка, доступна через вбудовану систему чату. Технічна підтримка завжди на зв'язку, готова відповісти на будь-які запитання та вирішити будь-які проблеми, що виникають у користувачів. Це забезпечує надійність та зручність у використанні платформи, дозволяючи користувачам швидко отримувати допомогу та поради.

На цій же сторінці можна зручно вийти з застосунку за потреби, просто натиснувши внизу на кнопку Logout (рис. 3.20).

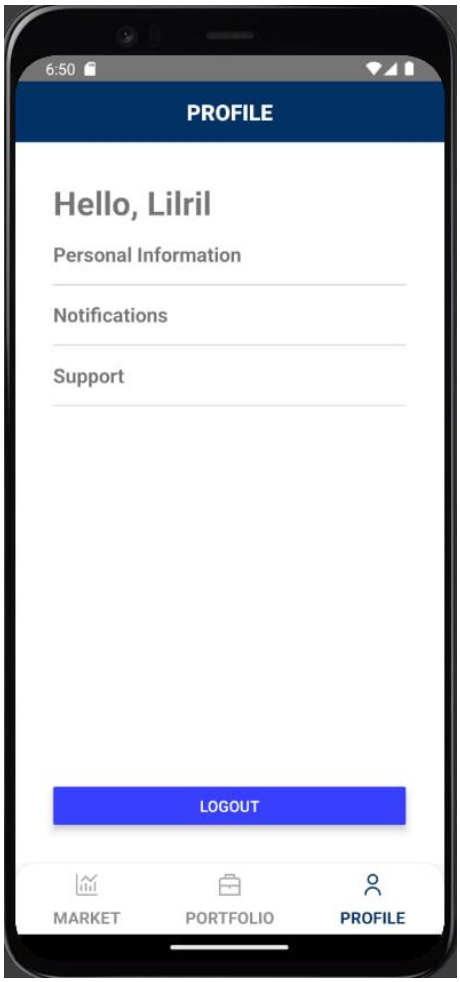

Рисунок 3.20 – Інтерфейс сторінки Profile

Розробка графічного інтерфейсу онлайн платформи була проведена у середовищі розробки Visual Studio Code з використанням технології React Native.

3.5 Висновки

У третьому розділі було обґрунтовано вибір фреймворку та технологій для розробки системи. Проаналізувавши потреби програмного продукту було обрано фреймворк React Native. Розроблено модулі системи. Для зручності розробки графічного інтерфейсу було обрано емулятор від Android Studio через найширший набір інструментів для тестування та налагодження. В якості сервера бази даних було обрано FireBase.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування мобільних застосунків на Android – це важлива складова процесу розробки програмного забезпечення для мобільних пристроїв. Цей процес включає в себе перевірку функціональності, продуктивності, стабільності та сумісності програмного забезпечення з різними версіями операційної системи Android та пристроями різних виробників, а також забезпечує високу якість та надійність програмного продукту перед його випуском на ринок. Основні етапи тестування мобільних застосунків на Android включають в себе ручне та автоматизоване тестування.

Ручне тестування передбачає пряму взаємодію з програмою на мобільному пристрої для перевірки її функціональності, а також виявлення можливих дефектів та аномалій у роботі програми. Цей процес може включати в себе тестування різних аспектів програми, таких як інтерфейс користувача, роботу з базою даних, мережеву взаємодію та інші [17].

Автоматизоване тестування використовує спеціальні програмні засоби для виконання тестових сценаріїв без прямого втручання людини. Це дозволяє швидше та ефективніше виявляти та виправляти помилки у програмному коді, забезпечуючи при цьому високу якість продукту. Автоматизоване тестування також дозволяє автоматизувати тестування на різних пристроях та версіях ОС Android, що значно збільшує його ефективність [18].

Одним з таких є Android Studio Profiler – це потужний інструмент, що дозволяє проводити детальний аналіз продуктивності та ресурсного споживання мобільних застосунків на платформі Android [19]. Він надає розширені можливості для вимірювання різних параметрів, таких як використання ЦП, пам'яті, мережі та інших ресурсів. Android Studio Profiler забезпечує інтерактивні графіки та діаграми, що дозволяють розробникам аналізувати продуктивність застосунку в реальному часі під час його виконання на пристрої. Крім того, він дозволяє виявляти потенційні проблеми з продуктивністю та оптимізувати роботу застосунку для

забезпечення оптимальної продуктивності та ефективності. Android Studio Profiler є незамінним інструментом для розробників мобільних застосунків, який допомагає забезпечити високу якість та надійність програмного продукту перед його випуском на ринок.

Результат тестування зображено на рисунку 4.1.

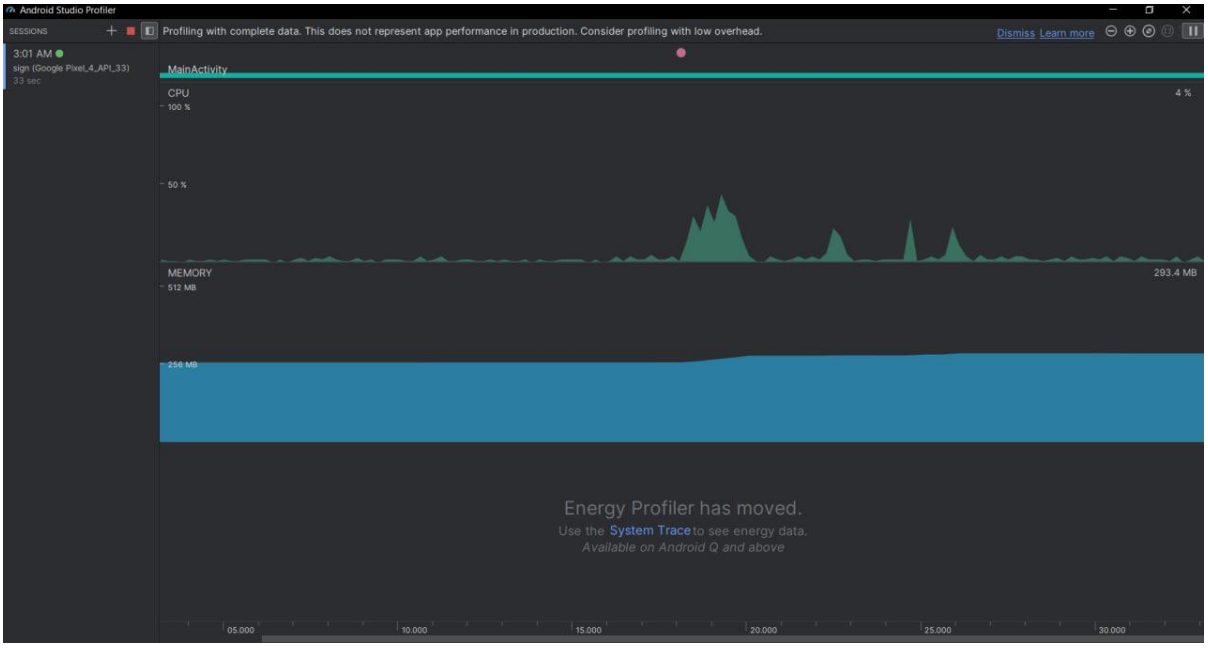

Рисунок 4.1 – Вікно тестування Android Studio Profiler

Інструмент перевіряє різні аспекти роботи програми на платформі Android. Він аналізує споживання ресурсів, таких як центральний процесор, оперативна пам'ять, мережа та батарея. Дозволяє відслідковувати виклики методів, звернення до пам'яті, передачу даних через мережу, витрату енергії та інші параметри, що впливають на продуктивність і коректну роботу програми.

Наступним кроком буде перевірка екрану реєстрації користувача шляхом спроби зареєструвати користувача під некоректними даними. Під час цієї перевірки система буде аналізувати введені дані, переконуючись, що користувач введе правильні дані у відповідних полях. В разі, якщо будуть виявлені помилки або недостовірні дані, система повинна надати зрозумілі повідомлення користувачу про те, які саме дані не відповідають вимогам і як виправити ситуацію. У результаті перевірки, користувач отримає подібне повідомлення (рис. 4.2).

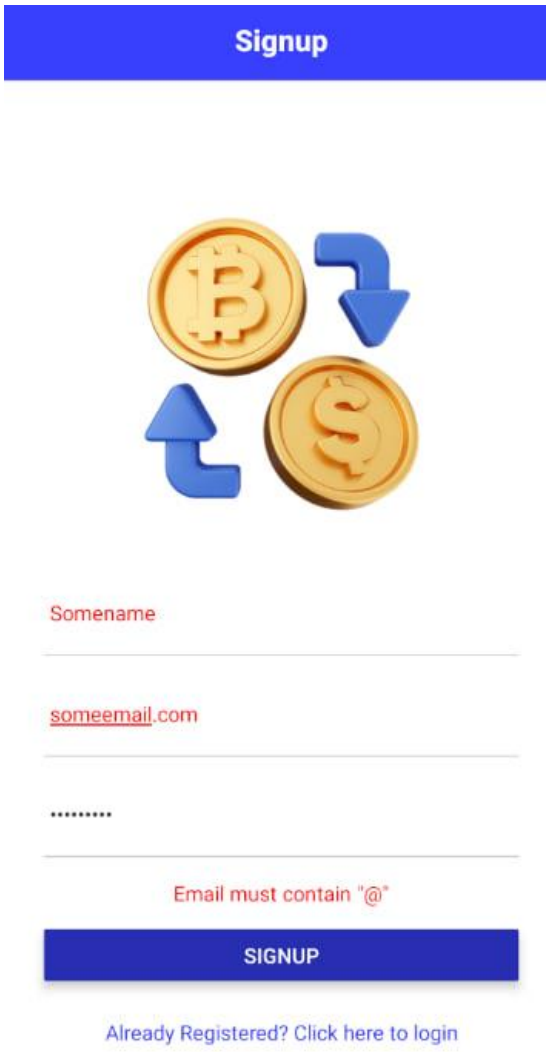

Рисунок 4.2 – Повідомлення про некоректно введені дані

Крім правильності введених даних також варто приділяти увагу на складність паролю. Надійний пароль має містити комбінацію великих і малих літер, цифр та спеціальних символів. Важливо рекомендувати користувачам використовувати унікальні паролі з певною довжиною символів для кожного облікового запису та періодично їх змінювати (рис. 4.3). Такий підхід допоможе підвищити безпеку користувачів та запобігти можливим випадкам несанкціонованого доступу до їх облікових записів.

Рисунок 4.3 – Перевірка на складність паролю

Важливу роль відіграють вже зареєстровані акаунти. Чому варто приділяти увагу оскільки це необхідно для запобігання створенню дублікатів облікових записів. Перед тим як дозволити реєстрацію нового користувача, система повинна перевірити базу даних на предмет наявності подібних облікових записів. Якщо буде виявлено, що введені дані вже існують у системі, користувачеві буде надана відповідна інформація. Це допоможе уникнути змішання даних та забезпечить коректну ідентифікацію користувачів у системі. В результаті тестування користувач буде отримувати такого роду повідомлення (рис. 4.4).

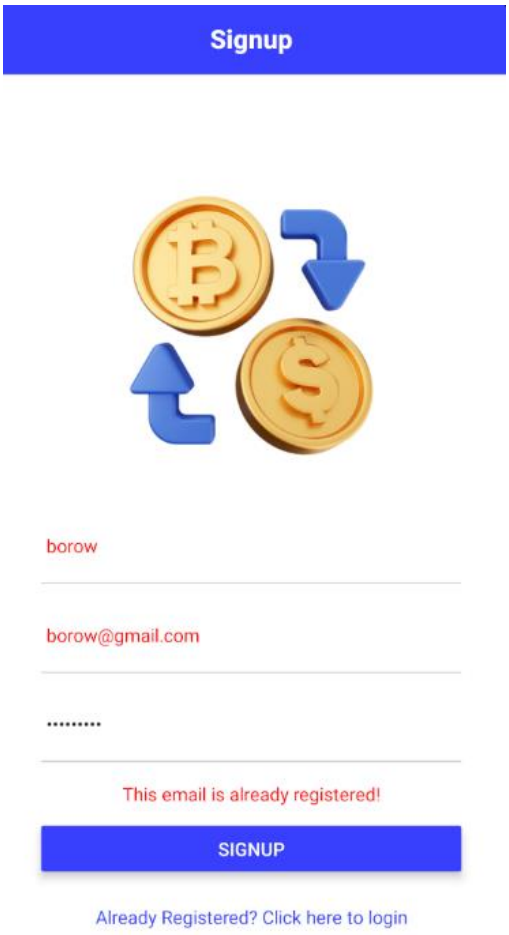

Рисунок 4.4 – Перевірка на вже зареєстрованого користувача

В результаті у FireBase можна запевнитись що даний користувач вже існує, це можна перевірити – просто відкривши панель керування базою (рис. 4.5).

Рисунок 4.5 – База даних FireBase

В свою чергу в консолі можна побачити як цей користувач може успішно авторизуватися, про результат свідчать успішні логи (рис. 4.6).

Рисунок 4.6 – Свідчення про успішну авторизацію

Також варто звернути увагу на спробу авторизуватися у неіснуючий аккаунт, таке повідомлення допоможе користувачеві який вже до цього ймовірно користувався сервісом, підібрати потрібні йому дані для входу, якщо він володіє не тільки одною поштою, у потрібний для нього акаунт. На рисунку 4.7 повідомлення про те що такого користувача не існує.

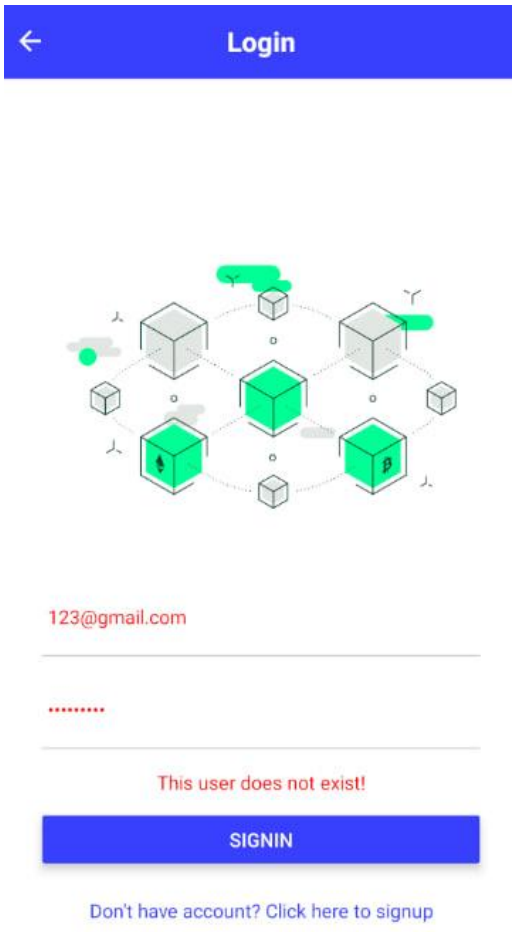

Рисунок 4.7 – Перевірка на існуючого користувача

Важливим аспектом мобільної платформи є валідація вхідних даних полів застосунку. Ця процедура необхідна для забезпечення коректності та безпеки даних, які вводять користувачі. Результат такої перевірки можна побачити на рисунку рисунку 4.8.

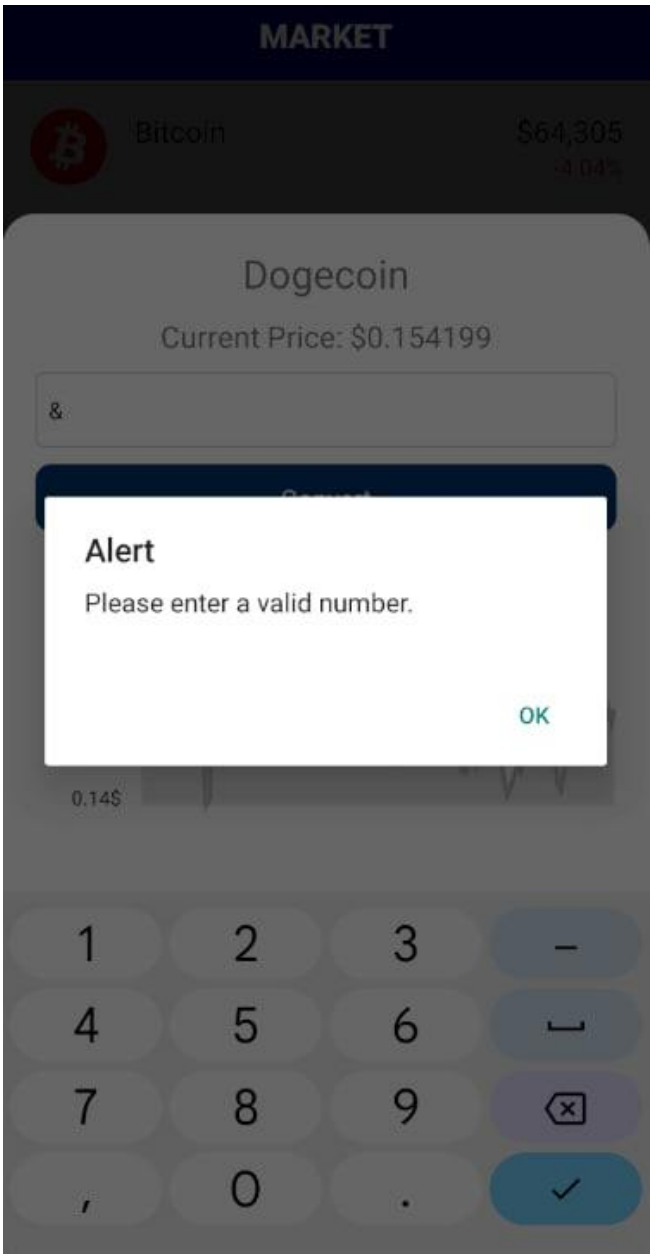

Рисунок 4.8 – Перевірка валідації даних сторінки Market

Відповідно така ж перевірка присутня на всіх сторінках застосунку в даному випадку сторінка Portfolio (рис. 4.9).

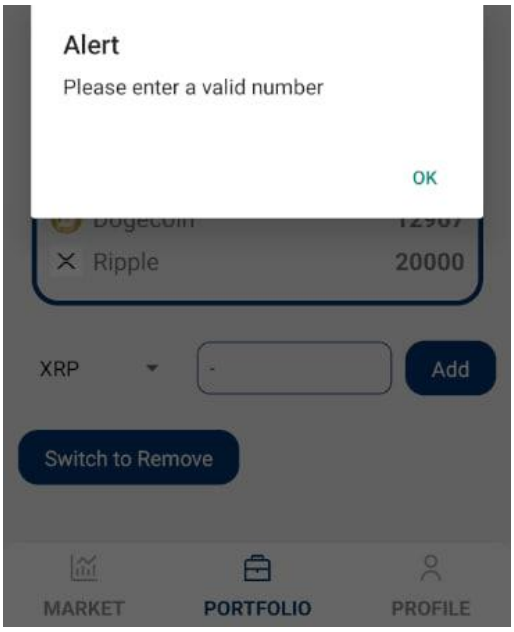

Рисунок 4.9 – Перевірка валідації даних сторінки Portfolio

Отже, в результаті тестування було перевірено та усунено усі випадки які могли б виникнути під час взаємодії з застосунком. У результаті чого була підтверджена працездатність та безпека взаємодії з системою, що гарантує надійну роботу платформи та захист особистої інформації користувачів.

4.2 Розробка інструкції користувача

Інструкція користувача є ключовим документом для користувача, який надає необхідну інформацію про мінімальну конфігурацію та як використовувати певний продукт [20]. Докладну інформацію щодо мінімальної та рекомендованої конфігурації можна знайти в таблицях 4.1 та 4.2.

Таблиця 4.1 – Мінімальна конфігурація

Процесор	ARM Cortex-A53 або еквівалентний процесор
Об’єм оперативної пам’яті	2 ГБ
Об’єм пам'яті на диску	16 ГБ вбудованої пам'яті
Операційна система	Android 7.0 (Nougat)

Таблиця 4.2 – Рекомендована конфігурація

Процесор	Qualcomm Snapdragon 660 або еквівалентний процесор
Об’єм оперативної пам’яті	4 ГБ
Об’єм пам'яті на диску	32 ГБ внутрішньої пам'яті
Операційна система	Android 9.0 (Pie)

Після завантаження усіх необхідних ресурсів застосунку, користувачеві пропонується зареєструватись (рис. 4.10).

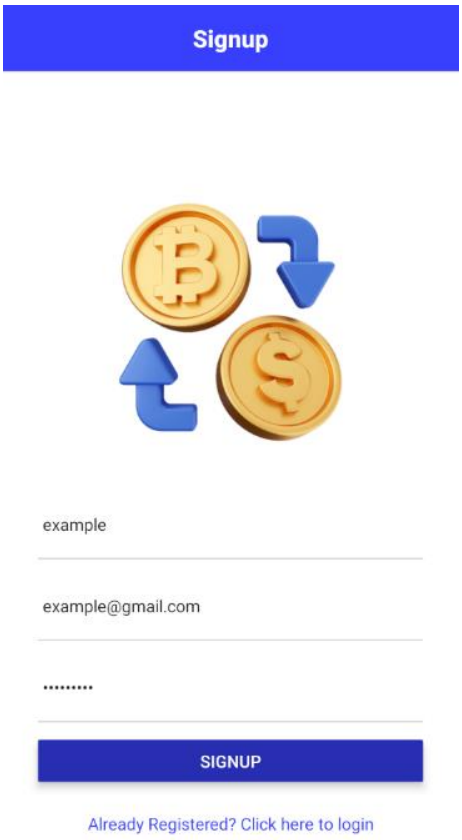

Рисунок 4.10 – Сторінка для реєстрації

Якщо користувач вже зареєстрований, він може натиснути на відповідний текст який перенаправить його на сторінку для авторизації (рис. 4.11).

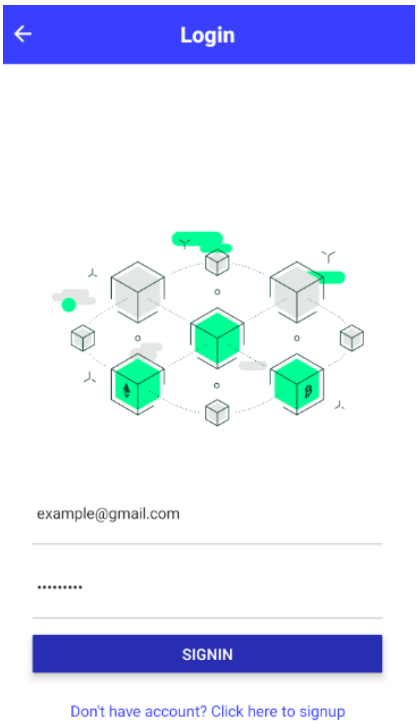

Рисунок 4.11 – Сторінка для авторизації

Після успішного входу у застосунок користувач попадає на першу сторінку Market, де може ознайомитися з ринком криптовалют та їх ситуацією на момент перегляду (рис. 4.12).

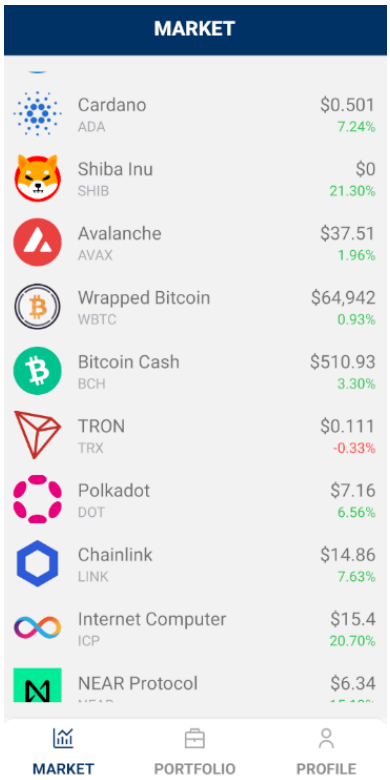

Рисунок 4.12 – Сторінка для перегляду криптовалюти

Якщо користувача зацікавила якась певна криптовалюта, він може обрати любу з переліку та подивитися детальну інформацію, а саме її графік та можливість скористатися функцією конвертування за поточним курсом (рис. 4.13). Крім того, він може оцінити її потенціал для інвестування або використання в торгівлі. Така доступність інформації дозволяє користувачам бути більш інформованими та обґрунтованими у своїх фінансових рішеннях, зменшуючи ризики та максимізуючи можливості отримання прибутку.

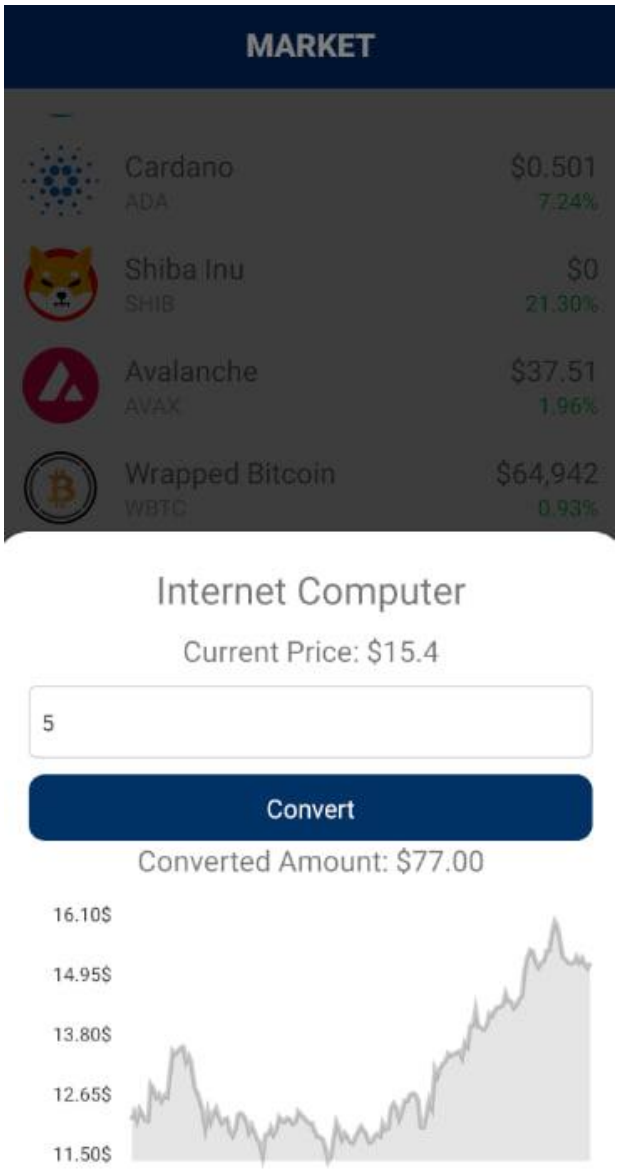

Рисунок 4.13 – Детальна інформація про монету

Після перегляду цін та ситуацій на ринку можна перейти до формування свого власного портфелю з криптовалютою (рис. 4.14). Це може бути важливим кроком для диверсифікації інвестиційного портфелю та зменшення ризику. Сформований портфель відображає стратегію управління активами та вибір конкретних криптовалютних активів, які є найбільш перспективними для досягнення своїх інвестиційних цілей.

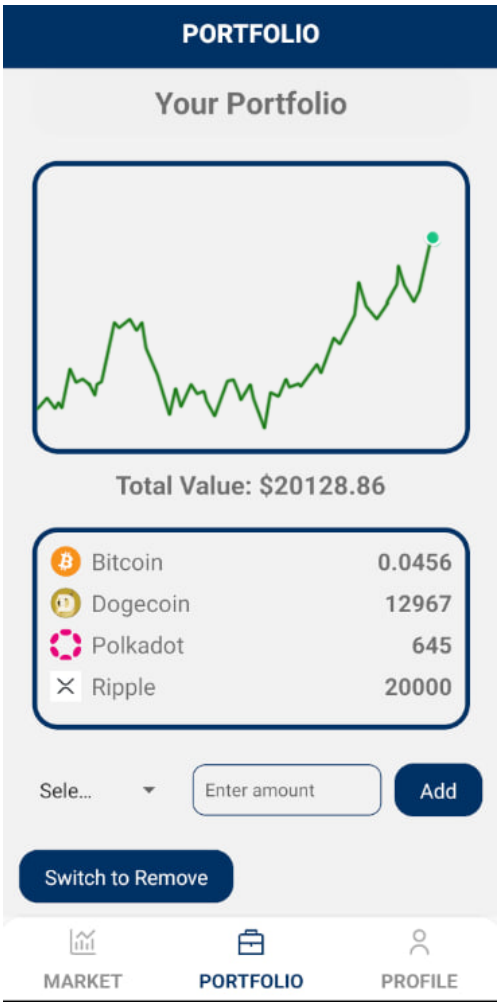

Рисунок 4.14 – Сторінка для моніторингу власних активів

Для того щоб додавати та прибирати криптовалюту з портфеля, треба обрати потрібну криптовалюту, щоб потім вже виконувати операції над нею, зробити це можна натиснувши на “Select Crypto”, після чого можна побачити дропдаун з переліком криптовалюти (рис. 4.15).

Рисунок 4.15 – Перелік криптовалюти для операцій

Після вибору необхідно вказати кількість бажаної криптовалюти та додати у портфель (рис. 4.16).

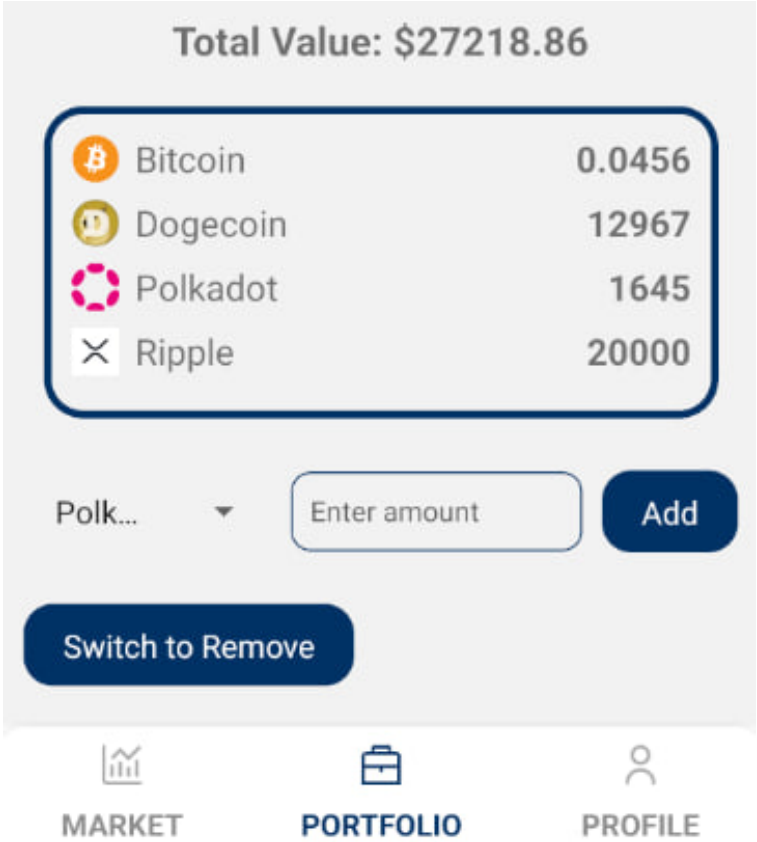

Рисунок 4.16 – Змінене портфоліо після додавання криптовалюти

Крім додавання можна також забрати криптовалюту з портфелю (рис. 4.17).

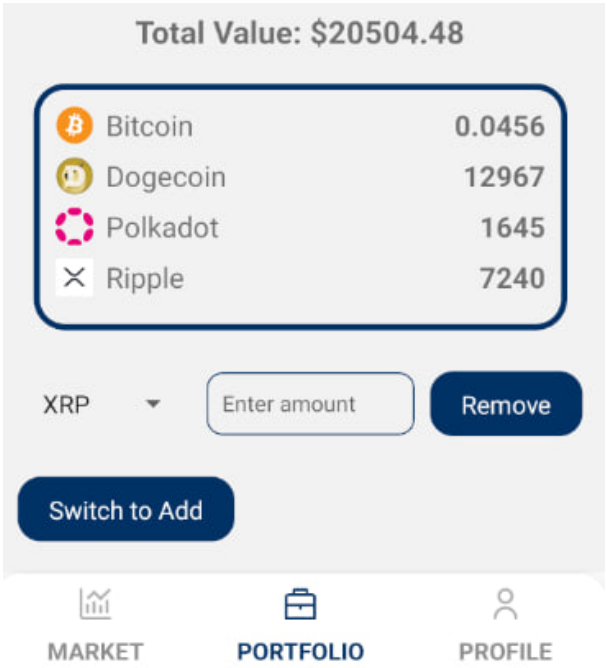

Рисунок 4.17 – Змінене портфоліо після зняття криптовалюти

Присутня сторінка профілю в якому користувач може налаштувати свою персоналізацію, сповіщення та поговорити з техпідтримкою (рис. 4.18).

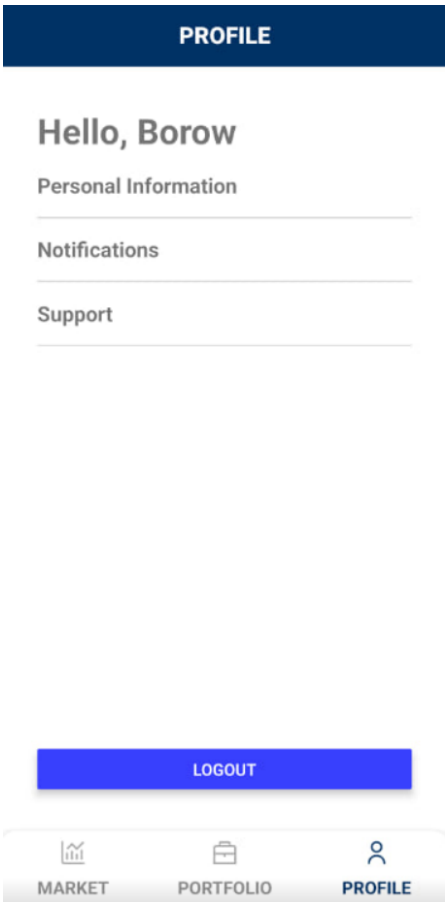

Рисунок 4.18 – Сторінка профілю

Подібний функціонал забезпечує безперебійну роботу мобільної системи оцінки операцій з криптовалютою.

4.3 Висновки

У четвертому розділі було здійснено тестування продуктивності та працездатності програмних модулів системи. А саме логін та реєстрацію користувача, зокрема протестованно і валідацію полів. Також розроблена інструкція користувача.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено програмні засоби реалізації мобільної системи оцінки операцій з криптовалютою.

Для розробки було використано середовище програмної розробки Microsoft Visual Studio Code та Android Studio. Бакалаврську дипломну роботу було оформлено згідно методичних вказівок [21].

Під час виконання бакалаврської дипломної роботи було проаналізовано стан проблеми на сьогоднішній день. Було розглянуто основні аналоги програмного продукту. Було визначено їхні недоліки та порівняно з власним програмним продуктом. Базуючись на результатах порівняння, було сформульовано основні задачі бакалаврської дипломної роботи.

Під час аналізу технологій розробки було обґрунтовано вибір мови програмування JavaScript та фреймворк React Native. Також була використана база даних FireBase для зберігання даних користувачів.

Відповідно до поставлених задач було:

- визначено найбільш якісне API для коректного відображення цін;
- розроблено метод, модель, алгоритми роботи системи;
- розроблено форму реєстрації, авторизації;
- реалізовано функцію створення власного портфелю;
- реалізовано функцію конвертації;
- розроблено зручний та адаптивний графічний інтерфейс мобільної платформи;
- проведено тестування мобільного застосунку згідно поставлених задач.

Було розроблено схеми загального алгоритму роботи мобільної системи та модуля довідки. Також було розроблено модель системи з використанням UML діаграм.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню. Також було розроблено інструкцію користувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Demystifying cryptocurrency and digital assets [Електронний ресурс] – Режим доступу до ресурсу: https://www.pwc.com/us/en/tech-effect/emerging-tech/understanding-cryptocurrency-digital-assets.html.

2. Герасименко К. О. Розробка мобільної системи оцінки операцій з криптовалютою / К. О. Герасименко, В. В. Войтко, А. В. Денисюк, О. В. Гаврилюк, Н. Є. Барчук // Матеріали LIII Науково-технічної конференції підрозділів Вінницького національного технічного університету, Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20410/16964.

3. Операції з криптовалютою: як створити криптогаманець та користуватися ним [Електронний ресурс] – Режим доступу до ресурсу: https://elle.ua/crypto-guida/operacii-z-kriptovalyutoyu.

4. Cryptocurrency Regulations Around the World [Електронний ресурс] – Режим доступу до ресурсу: https://www.investopedia.com/cryptocurrency-regulations-around-the-world-5202122.

5. CoinsStats [Електронний ресурс] – Режим доступу до ресурсу: https://coinstats.app.

6. Cryptocompare [Електронний ресурс] – Режим доступу до ресурсу: https://www.cryptocompare.com.

7. Kubera [Електронний ресурс] – Режим доступу до ресурсу: https://www.kubera.com.

8. React Native [Електронний ресурс] – Режим доступу до ресурсу: https://reactnative.dev.

9. What is an API (Application Programming Interface)? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/api.

10. What is the DOM? [Електронний ресурс] – Режим доступу до ресурсу: https://www.freecodecamp.org/news/what-is-the-dom-document-object-model-meaning-in-javascript.

11. UML - A Simple Guide for Beginners: A basic guide to systems analysis and design. Learn how to structure software projects using UML diagrams. Kindle Edition / Marco Aurelio M. C. Regis – 2024. – 66 с.

12. The Definitive Guide: Master the World's Most-Used Programming Language 7th Edition / David Flanagan – 2020. – 704 с.

13. FireBase [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com.

14. Visual Studio Code [Електронний ресурс] – Режим доступу до ресурсу: https://code.visualstudio.com.

15. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio.

16. CoinGecko [Електронний ресурс] – Режим доступу до ресурсу: https://www.coingecko.com/uk.

17. Ручне тестування [Електронний ресурс] – Режим доступу до ресурсу: https://www.zaptest.com/uk/ручне-тестування-що-це-таке-типи-проц.

18. Автоматизоване тестування [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya.

19. Android Studio Profiler [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio/profile/memory-profiler.

20. Інструкція користувача [Електронний ресурс] – Режим доступу до ресурсу: https://wiki.checkbox.ua/uk/app/mobile/instruction.

21. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

61

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«_12_» березня 2024 р.

Технічне завдання

на бакалаврську дипломну роботу

«Розробка методу та засобів реалізації мобільної системи оцінки операцій з криптовалютою»

за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

_____ к.т.н., доц. Войтко В. В.

«_12_» березня 2024 р.

Виконав:

_____ студент гр. Герасименко К. О.

«_12_» березня 2024 р.

Вінниця – 2024 рік

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка методу та засобів реалізації мобільної системи оцінки операцій з криптовалютою».

Галузь застосування – моніторинг активів.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від 11 березня 2024 року ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою дослідження є покращення моніторингу дій за своїми криптоактивами шляхом розробки та використання мобільної системи для допомоги інвесторам, що дозволить зручно відстежувати всі операції та власні активи.

Призначення роботи – розробка програмного продукту для моніторингу криптовалюти.

4. Вихідні дані для проведення НДР

1. Demystifying cryptocurrency and digital assets [Електронний ресурс] – Режим доступу до ресурсу: https://www.pwc.com/us/en/tech-effect/emerging-tech/understanding-cryptocurrency-digital-assets.html.

2. UML - A Simple Guide for Beginners: A basic guide to systems analysis and design. Learn how to structure software projects using UML diagrams. Kindle Edition / Marco Aurelio M. C. Regis – 2024. – 66 с.

3. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio.

4. FireBase [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com.

5. Технічні вимоги

Серверний комп’ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об’єм оперативної пам’яті 4 ГБ, розмір жорсткого диску 128 ГБ. Операційна система Windows 10.

З клієнтського боку: Android 10/11/12/13/14.

6. Конструктивні вимоги.

Мобільний застосунок має відповідати ергономічним та технічним вимогам. Текстова та графічна документація повинна відповідати стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

- пояснювальна записка до бакалаврської дипломної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітки
1	Аналіз розвитку мобільних систем для відстежування активів та постановка задач дослідження	14.03.24 – 21.03.24	
2	Розробка методу, моделі та алгоритмів роботи мобільної системи	22.03.24 – 10.04.24	
3	Розробка програмних модулів мобільної системи	12.04.24 – 30.04.24	
4	Тестування програми	01.05.24 – 20.05.24	
5	Оформлення матеріалів до захисту БДР	22.05.24 – 30.05.24	

10 Порядок контролю та прийняття.

Всі етапи бакалаврської дипломної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

Додаток Б – Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

65

Назва роботи: «Розробка методу та засобів реалізації мобільної системи оцінки операцій з криптовалютою»

Тип роботи: БДР

Підрозділ: кафедра програмного забезпечення, ФІТКІ

Науковий керівник: Войтко В. В.

Оригінальність	96,3%
Схожість	3,7%

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unichek

Автор роботи _____ Герасименко К. О.

Керівник роботи _____ Войтко В. В.

Додаток В – Лістинг програми

```
        //signup
        import React, { Component } from 'react';
        import { StyleSheet, Text, View, TextInput, Button, Alert, ActivityIndicator, Image
} from 'react-native';
        import firebase from '../database/firebase';

        export default class Signup extends Component {

          constructor() {
            super();
            this.state = {
              displayName: '',
              email: '',
              password: '',
              isLoading: false,
              errorMessage: ''
            }
          }

          updateInputVal = (val, prop) => {
            const state = this.state;
            state[prop] = val;
            this.setState(state);
          }

          registerUser = () => {
            if (this.state.email === '' || this.state.password === '') {
              Alert.alert('Enter details to signup!');
            } else if (!this.state.email.includes('@')) {
              this.setState({ errorMessage: 'Email must contain "@"' });
            } else if (this.state.password.length <= 8) {
              this.setState({ errorMessage: 'Password must be at least 8 characters long'
});
            } else {
              this.setState({
                isLoading: true,
                errorMessage: ''
              })
              firebase
```

```
    .auth()
    .createUserWithEmailAndPassword(this.state.email, this.state.password)
    .then((res) => {
      res.user.updateProfile({
        displayName: this.state.displayName
      })
      console.log('User registered successfully!')
      this.setState({
        isLoading: false,
        displayName: '',
        email: '',
        password: ''
      })
      this.props.navigation.navigate('Login')
    })
    .catch(error => {
      this.setState({ isLoading: false });
      if (error.code === 'auth/email-already-in-use') {
        this.setState({ errorMessage: 'This email is already registered!' });
      } else {
        this.setState({ errorMessage: error.message });
      }
    });
  }
}

render() {
  if (this.state.isLoading) {
    return(
      <View style={styles.preloader}>
        <ActivityIndicator size="large" color="#9E9E9E"/>
      </View>
    )
  }
  return (
    <View style={styles.container}>
      <Image source={require('../assets/exchange.webp')} style={styles.image} />
      <TextInput
        style={[styles.inputStyle, this.state.errorMessage && styles.errorInput]}
        placeholder="Name"
        value={this.state.displayName}
```

```
                onChangeText={(val) => this.updateInputVal(val, 'displayName')}
              />
              <TextInput
                style={[styles.inputStyle, this.state.errorMessage && styles.errorInput]}
                placeholder="Email"
                value={this.state.email}
                onChangeText={(val) => this.updateInputVal(val, 'email')}
              />
              <TextInput
                style={styles.inputStyle}
                placeholder="Password"
                value={this.state.password}
                onChangeText={(val) => this.updateInputVal(val, 'password')}
                maxLength={15}
                secureTextEntry={true}
              />
              {this.state.errorMessage                    ?                    <Text
style={styles.errorMessage}>{this.state.errorMessage}</Text> : null}
              <Button
                color="#3740FE"
                title="Signup"
                onPress={() => this.registerUser()}
              />
              <Text
                style={styles.loginText}
                onPress={() => this.props.navigation.navigate('Login')}>
                Already Registered? Click here to login
              </Text>
            </View>
          );
        }
      }

      const styles = StyleSheet.create({
        container: {
          flex: 1,
          display: "flex",
          flexDirection: "column",
          justifyContent: "center",
          padding: 35,
          backgroundColor: '#fff'
```

```
  },
  inputStyle: {
    width: '100%',
    marginBottom: 15,
    paddingBottom: 15,
    alignSelf: "center",
    borderColor: "#ccc",
    borderBottomWidth: 1
  },
  errorInput: {
    color: 'red'
  },
  errorMessage: {
    color: 'red',
    marginBottom: 15,
    textAlign: 'center'
  },
  loginText: {
    color: '#3740FE',
    marginTop: 25,
    textAlign: 'center'
  },
  preloader: {
    left: 0,
    right: 0,
    top: 0,
    bottom: 0,
    position: 'absolute',
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: '#fff'
  },
  image: {
    width: 250,
    height: 250,
    resizeMode: 'contain',
    alignSelf: 'center',
    marginBottom: 20,
  },
});
```

```
        //Login
        import React, { Component } from 'react';
        import { StyleSheet, Text, View, TextInput, Button, Alert, ActivityIndicator, Image
} from 'react-native';
        import firebase from '../database/firebase';

        export default class Login extends Component {

          constructor() {
            super();
            this.state = {
              email: '',
              password: '',
              isLoading: false,
              errorMessage: ''
            }
          }

          updateInputVal = (val, prop) => {
            const state = this.state;
            state[prop] = val;
            this.setState(state);
          }

          userLogin = () => {
            if (this.state.email === '' || this.state.password === '') {
              Alert.alert('Enter details to signin!')
            } else {
              this.setState({
                isLoading: true,
                errorMessage: ''
              })
              firebase
              .auth()
              .signInWithEmailAndPassword(this.state.email, this.state.password)
              .then((res) => {
                console.log(res)
                console.log('User logged-in successfully!')
                this.setState({
                  isLoading: false,
                  email: '',
```

```
                password: ''
              })
              this.props.navigation.navigate('Dashboard')
            })
            .catch(error => {
              this.setState({ isLoading: false });
              if (error.code === 'auth/invalid-credential') {
                this.setState({ errorMessage: 'This user does not exist!' });
              } else if (error.code === 'auth/user-not-found' || error.code === 'auth/invalid-email') {
                this.setState({ errorMessage: 'This user does not exist!' });
              } else if (error.code === 'auth/wrong-password') {
                this.setState({ errorMessage: 'Invalid email or password!' });
              } else {
                this.setState({ errorMessage: error.message });
              }
            });
        }
      }

      render() {
        if (this.state.isLoading) {
          return(
            <View style={styles.preloader}>
              <ActivityIndicator size="large" color="#9E9E9E"/>
            </View>
          )
        }
        return (
          <View style={styles.container}>
            <Image              source={require('../assets/what-is-a-blockchain.png')} style={styles.image} />
            <TextInput
              style={[styles.inputStyle, this.state.errorMessage && styles.errorInput]}
              placeholder="Email"
              value={this.state.email}
              onChangeText={(val) => this.updateInputVal(val, 'email')}
            />
            <TextInput
              style={[styles.inputStyle, this.state.errorMessage && styles.errorInput]}
              placeholder="Password"
```

```
                value={this.state.password}
                onChangeText={(val) => this.updateInputVal(val, 'password')}
                maxLength={15}
                secureTextEntry={true}
              />
              {this.state.errorMessage                    ?                    <Text
style={styles.errorMessage}>{this.state.errorMessage}</Text> : null}
              <Button
                color="#3740FE"
                title="Signin"
                onPress={() => this.userLogin()}
              />
              <Text
                style={styles.loginText}
                onPress={() => this.props.navigation.navigate('Signup')}>
                Don't have account? Click here to signup
              </Text>
            </View>
        );
      }
    }

    const styles = StyleSheet.create({
      container: {
        flex: 1,
        display: "flex",
        flexDirection: "column",
        justifyContent: "center",
        padding: 35,
        backgroundColor: '#fff'
      },
      inputStyle: {
        width: '100%',
        marginBottom: 15,
        paddingBottom: 15,
        alignSelf: "center",
        borderColor: "#ccc",
        borderBottomWidth: 1
      },
      errorInput: {
        color: 'red'
```

```
  },
  errorMessage: {
    color: 'red',
    marginBottom: 15,
    textAlign: 'center'
  },
  loginText: {
    color: '#3740FE',
    marginTop: 25,
    textAlign: 'center'
  },
  preloader: {
    left: 0,
    right: 0,
    top: 0,
    bottom: 0,
    position: 'absolute',
    alignItems: 'center',
    justifyContent: 'center',
    backgroundColor: '#fff'
  },
  image: {
    width: 350,
    height: 250,
    resizeMode: 'contain',
    alignSelf: 'center',
    marginBottom: 20,
  },
});

//MarketScreen
import React, { useState, useEffect } from 'react';
import { View, FlatList, StyleSheet, TouchableOpacity, Text } from 'react-native';
import axios from 'axios';
import ListItem from './ListItem';
import BottomSheet from './BottomSheet';

const MarketScreen = () => {
  const [cryptoData, setCryptoData] = useState([]);
  const [bottomSheetVisible, setBottomSheetVisible] = useState(false);
  const [selectedCrypto, setSelectedCrypto] = useState(null);
```

```
  const [chartData, setChartData] = useState(null);
  const [currentPrice, setCurrentPrice] = useState(null); // Add currentPrice state

  useEffect(() => {
    const fetchCryptoData = async () => {
      try {
        const response = await axios.get('https://api.coingecko.com/api/v3/coins/markets', {
          params: {
            vs_currency: 'usd',
            per_page: 20,
            page: 1,
            sparkline: false,
            price_change_percentage: '7d',
          },
        });
        setCryptoData(response.data);
      } catch (error) {
        console.error('Error fetching cryptocurrency data:', error);
      }
    };

    fetchCryptoData();
  }, []);

  const fetchChartData = async (cryptoId) => {
    try {
      const response = await axios.get(`https://api.coingecko.com/api/v3/coins/${cryptoId}/market_chart`, {
        params: {
          vs_currency: 'usd',
          days: 7,
        },
      });
      const chartData = response.data.prices.map(([timestamp, price]) => ({
        date: new Date(timestamp),
        price: price,
      }));
      return chartData;
    } catch (error) {
      console.error('Error fetching chart data:', error);
```

```
        return null;
      }
    };

    const handleCryptoPress = async (crypto) => {
      setSelectedCrypto(crypto);
      setBottomSheetVisible(true);
      const data = await fetchChartData(crypto.id);
      setChartData(data);
      setCurrentPrice(crypto.current_price); // Set currentPrice
      console.log("Current Price:", crypto.current_price); // Debugging output
    };

    return (
      <View style={styles.container}>
        <FlatList
          data={cryptoData}
          keyExtractor={(item) => item.id}
          renderItem={({ item }) => (
            <ListItem
              crypto={{
                name: item.name,
                symbol: item.symbol.toUpperCase(),
                currentPrice: item.current_price || 0,
                priceChangePercentage7d: item.price_change_percentage_7d_in_currency
!== null ? `${parseFloat(item.price_change_percentage_7d_in_currency).toFixed(2)}%` :
'N/A',
                image: item.image,
              }}
              onPress={() => handleCryptoPress(item)}
            />
          )}
        />
        {bottomSheetVisible && (
          <TouchableOpacity
            style={styles.overlay}
            onPress={() => setBottomSheetVisible(false)}
          >
            <BottomSheet
              visible={bottomSheetVisible}
              onClose={() => setBottomSheetVisible(false)}
```

```
                crypto={selectedCrypto}
                chartData={chartData}
                currentPrice={currentPrice} // Pass currentPrice to BottomSheet
              />
            </TouchableOpacity>
          )}
        </View>
      );
    };

    const styles = StyleSheet.create({
      container: {
        flex: 1,
        padding: 16,
      },
      overlay: {
        ...StyleSheet.absoluteFillObject,
        backgroundColor: 'rgba(0, 0, 0, 0.5)',
        justifyContent: 'flex-end',
      },
    });

    export default MarketScreen;

    //BottomSheet
    import React, { useState } from 'react';
    import { Modal, View, Text, TouchableOpacity, StyleSheet, Dimensions, TextInput,
Button } from 'react-native';
    import { LineChart } from 'react-native-chart-kit';

    const CustomButton = ({ title, onPress }) => (
      <TouchableOpacity style={styles.button} onPress={onPress}>
        <Text style={styles.buttonText}>{title}</Text>
      </TouchableOpacity>
    );

    const BottomSheet = ({ visible, onClose, crypto, chartData, currentPrice }) => {
      if (!visible || !crypto || !chartData) return null;

      const { name } = crypto;
      const [amount, setAmount] = useState('');
```

```
  const [convertedAmount, setConvertedAmount] = useState(null);

  const handleConvert = () => {
    if (amount.trim() === '' || isNaN(parseFloat(amount))) {
      alert('Please enter a valid number.');
      return;
    }
    const convertedValue = parseFloat(amount) * parseFloat(currentPrice);
    setConvertedAmount(convertedValue.toFixed(2));
  };

  const handlePressOutside = () => {
    onClose();
  };

  return (
    <Modal visible={visible} animationType="slide" transparent>
      <TouchableOpacity
        style={styles.overlay}
        onPress={handlePressOutside}
      >
        <View style={styles.bottomSheet}>
          <View style={styles.content}>
            <Text style={styles.title}>{name}</Text>
            <Text style={styles.subtitle}>Current Price: ${currentPrice}</Text>
            <TextInput
              style={styles.input}
              placeholder="Enter amount"
              keyboardType="numeric"
              value={amount}
              onChangeText={setAmount}
            />
            <CustomButton title="Convert" onPress={handleConvert} />
            {convertedAmount !== null && (
              <Text            style={styles.subtitle}>Converted            Amount:
${convertedAmount}</Text>
            )}
            <LineChart
              data={{
                datasets: [
                  {
```

```
                    data: chartData.map((dataPoint) => dataPoint.price),
                  },
                ],
              }}
              width={Dimensions.get('window').width - 40}
              height={200}
              yAxisSuffix="$"
              chartConfig={{
                backgroundColor: '#ffffff',
                backgroundGradientFrom: '#ffffff',
                backgroundGradientTo: '#ffffff',
                decimalPlaces: 2,
                color: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
                labelColor: (opacity = 1) => `rgba(0, 0, 0, ${opacity})`,
                style: {
                  borderRadius: 16
                },
                propsForDots: {
                  r: '4',
                  strokeWidth: '2',
                  stroke: '#ffa726'
                }
              }}
              withVerticalLines={false}
              withHorizontalLines={false}
              withDots={false}
            />
          </View>
        </View>
      </TouchableOpacity>
    </Modal>
  );
};

const styles = StyleSheet.create({
  overlay: {
    flex: 1,
    backgroundColor: 'rgba(0, 0, 0, 0.5)',
    justifyContent: 'flex-end',
  },
  bottomSheet: {
```

```
    backgroundColor: 'white',
    borderTopLeftRadius: 20,
    borderTopRightRadius: 20,
    padding: 20,
    elevation: 5,
  },
  content: {
    alignItems: 'center',
  },
  title: {
    fontSize: 24,
    marginBottom: 10,
  },
  subtitle: {
    fontSize: 18,
    marginBottom: 10,
  },
  input: {
    width: '100%',
    borderWidth: 1,
    borderColor: '#ccc',
    borderRadius: 5,
    padding: 8,
    marginBottom: 10,
  },
  button: {
    width: '100%',
    backgroundColor: '#003366',
    borderRadius: 10,
    padding: 10,
    alignItems: 'center',
  },
  buttonText: {
    color: 'white',
    fontSize: 16,
  },
});

export default BottomSheet;

//PortfolioScreen
```

```
        import React, { useState, useEffect } from 'react';
        import { View, Text, StyleSheet, ScrollView, Image, TextInput, TouchableOpacity }
from 'react-native';
        import axios from 'axios';
        import { LineChart } from 'react-native-svg-charts';
        import { Picker } from '@react-native-picker/picker';

        const PortfolioScreen = () => {
          const [portfolioData, setPortfolioData] = useState([
            { id: 'bitcoin', name: 'Bitcoin', amount: 0.0456 },
            { id: 'dogecoin', name: 'Dogecoin', amount: 12967 },
            { id: 'polkadot', name: 'Polkadot', amount: 645 },
            { id: 'ripple', name: 'Ripple', amount: 20000 },
          ]);
          const [cryptoData, setCryptoData] = useState([]);
          const [loading, setLoading] = useState(true);
          const [error, setError] = useState(null);
          const [totalValue, setTotalValue] = useState(0);
          const [selectedCrypto, setSelectedCrypto] = useState('');
          const [amount, setAmount] = useState('');
          const [action, setAction] = useState('Add');
          const [valueData, setValueData] = useState([]);

          useEffect(() => {
            const fetchCryptoData = async () => {
              try {
                const                    response                    =                    await
axios.get('https://api.coingecko.com/api/v3/coins/markets', {
                  params: {
                    vs_currency: 'usd',
                    ids: portfolioData.map(asset => asset.id).join(','),
                  },
                });
                setCryptoData(response.data);
                setLoading(false);
              } catch (error) {
                setError('Error fetching cryptocurrency data');
              }
            };

            fetchCryptoData();
```

```
        }, []);

        useEffect(() => {
          if (!loading && cryptoData.length > 0) {
            const totalValue = portfolioData.reduce((acc, asset) => {
              const cryptoInfo = cryptoData.find(crypto => crypto.id === asset.id);
              const assetValue = asset.amount * cryptoInfo.current_price;
              return acc + assetValue;
            }, 0);
            setTotalValue(totalValue);
            setValueData(prevData => [...prevData, totalValue]);
          }
        }, [loading, cryptoData, portfolioData]);

        const handleAction = () => {
          if (!selectedCrypto) {
            alert('Please select a cryptocurrency');
            return;
          }

          if (!amount || isNaN(amount)) {
            alert('Please enter a valid number');
            return;
          }

          const newAmount = parseFloat(amount);
          const updatedPortfolio = portfolioData.map(asset => {
            if (asset.id === selectedCrypto) {
              return { ...asset, amount: action === 'Add' ? asset.amount + newAmount :
asset.amount - newAmount };
            } else {
              return asset;
            }
          });
          setPortfolioData(updatedPortfolio); // Update portfolio state
          setAmount('');

          // Recalculate total value
          const totalValue = updatedPortfolio.reduce((acc, asset) => {
            const cryptoInfo = cryptoData.find(crypto => crypto.id === asset.id);
            const assetValue = asset.amount * cryptoInfo.current_price;
```

```
    return acc + assetValue;
  }, 0);
  setTotalValue(totalValue);
  setValueData(prevData => [...prevData, totalValue]);
};

const handlePickerChange = (cryptoId) => {
  setSelectedCrypto(cryptoId);
};

const handleActionChange = (actionType) => {
  setAction(actionType);
};

if (loading) {
  return <Text>Loading...</Text>;
}

if (error) {
  return <Text>{error}</Text>;
}

const data = valueData.map((value, index) => ({ value, key: index }));

const contentInset = { top: 20, bottom: 20 };

return (
  <ScrollView contentContainerStyle={styles.container}>
    <View style={styles.headerContainer}>
      <Text style={styles.headerText}>Your Portfolio</Text>
    </View>
    <View style={styles.chartWrapper}>
      <View style={styles.chartContainer}>
        <LineChart
          style={{ height: 200 }}
          data={data}
          contentInset={contentInset}
          svg={{ stroke: 'rgb(134, 65, 244)' }}
        />
      </View>
    </View>
```

```
            <View style={styles.totalValueContainer}>
              <Text              style={styles.totalValueText}>Total              Value:
${totalValue.toFixed(2)}</Text>
            </View>
            <View style={styles.assetListContainer}>
              <View style={styles.assetList}>
                {portfolioData.map((asset, index) => {
                  const cryptoInfo = cryptoData.find(crypto => crypto.id === asset.id);
                  return (
                    <View key={index} style={styles.assetContainer}>
                      <View style={styles.assetInfo}>
                        <Image       source={{       uri:       cryptoInfo.image       }}
style={styles.assetIcon} />
                        <Text style={styles.assetName}>{asset.name}</Text>
                      </View>
                      <Text style={styles.assetAmount}>{asset.amount}</Text>
                    </View>
                  );
                })}
              </View>
            </View>
            <View style={styles.actionContainer}>
              <Picker
                selectedValue={selectedCrypto}
                style={styles.picker}
                onValueChange={(itemValue, itemIndex) => handlePickerChange(itemValue)}
              >
                <Picker.Item label="Select a cryptocurrency" value="" />
                {cryptoData.map((crypto, index) => (
                  <Picker.Item key={index} label={crypto.name} value={crypto.id} />
                ))}
              </Picker>
              <TextInput
                placeholder="Enter amount"
                keyboardType="numeric"
                style={styles.input}
                value={amount}
                onChangeText={text => setAmount(text)}
              />
              <TouchableOpacity style={styles.actionButton} onPress={handleAction}>
                <Text style={styles.actionButtonLabel}>{action}</Text>
```

```
        </TouchableOpacity>
      </View>
      <View style={styles.actionContainer}>
        <TouchableOpacity
          style={styles.actionButton}
          onPress={() => handleActionChange(action === 'Add' ? 'Remove' : 'Add')}
        >
          <Text style={styles.actionButtonLabel}>
            {action === 'Add' ? 'Switch to Remove' : 'Switch to Add'}
          </Text>
        </TouchableOpacity>
      </View>
    </ScrollView>
  );
};

const styles = StyleSheet.create({
  container: {
    flexGrow: 1,
    paddingHorizontal: 16,
  },
  headerContainer: {
    alignItems: 'center',
    backgroundColor: '#f0f0f0',
    paddingVertical: 10,
    borderRadius: 20,
    marginHorizontal: 10,
    marginBottom: 20,
  },
  headerText: {
    fontSize: 24,
    fontWeight: 'bold',
  },
  chartWrapper: {
    borderWidth: 4,
    borderColor: '#003366',
    borderRadius: 20,
    marginHorizontal: 10,
    marginBottom: 10,
  },
  chartContainer: {
```

```
  alignItems: 'center',
  marginBottom: 20,
},
totalValueContainer: {
  alignItems: 'center',
  marginBottom: 20,
},
totalValueText: {
  fontSize: 20,
  fontWeight: 'bold',
},
assetListContainer: {
  borderRadius: 20,
  borderWidth: 4,
  borderColor: '#ccc',
  padding: 10,
  marginHorizontal: 10,
  marginBottom: 20,
  alignSelf: 'stretch',
  borderColor: '#003366',
},
assetList: {},
assetContainer: {
  flexDirection: 'row',
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: 8,
},
assetInfo: {
  flexDirection: 'row',
  alignItems: 'center',
},
assetIcon: {
  width: 24,
  height: 24,
  marginRight: 8,
},
assetName: {
  fontSize: 18,
},
assetAmount: {
```

```
    fontSize: 18,
    fontWeight: 'bold',
  },
  actionContainer: {
    flexDirection: 'row',
    alignItems: 'center',
    marginBottom: 20,
  },
  picker: {
    flex: 1,
    height: 40,
    marginRight: 10,
  },
  input: {
    flex: 1,
    height: 40,
    borderColor: '#003366',
    borderWidth: 1,
    paddingHorizontal: 10,
    marginRight: 10,
    borderRadius: 10,
  },
  actionButton: {
    backgroundColor: '#003366',
    paddingVertical: 10,
    paddingHorizontal: 20,
    borderRadius: 15,
  },
  actionButtonLabel: {
    color: '#FFFFFF',
    fontSize: 16,
    width: '100%'
  },
});

export default PortfolioScreen;

//Dashboard
import React, { Component } from 'react';
import { StyleSheet, View, Text, Button, TouchableOpacity, FlatList } from 'react-
native';
```

```
import firebase from '../database/firebase';

const menuItems = [
  { id: 1, title: 'Personal Information' },
  { id: 2, title: 'Notifications' },
  { id: 3, title: 'Support' },
  // Add more menu items as needed
];

export default class Dashboard extends Component {
  constructor() {
    super();
    this.state = {
      displayName: '',
      uid: ''
    };
  }

  componentDidMount() {
    const user = firebase.auth().currentUser;
    if (user) {
      this.setState({
        displayName: user.displayName || '',
        uid: user.uid || ''
      });
    }
  }

  signOut = () => {
    firebase
      .auth()
      .signOut()
      .then(() => {
        this.props.navigation.navigate('Login');
      })
      .catch(error => this.setState({ errorMessage: error.message }));
  };

  renderMenuItem = ({ item }) => {
    return (
```

```
            <TouchableOpacity        style={styles.menuItem}        onPress={()        =>
console.log(item.title)}>
              <Text style={styles.menuItemText}>{item.title}</Text>
            </TouchableOpacity>
          );
        };

        render() {
          const { displayName } = this.state;

          return (
            <View style={styles.container}>
              <Text style={styles.textStyle}>Hello, {displayName}</Text>
              <View style={styles.menu}>
                <FlatList
                  data={menuItems}
                  renderItem={this.renderMenuItem}
                  keyExtractor={item => item.id.toString()}
                />
              </View>
              <View style={styles.content}>
                {/* Existing content */}
              </View>
              <View style={styles.logoutButton}>
                <Button
                  color="#3740FE"
                  title="Logout"
                  onPress={() => this.signOut()}
                  style={styles.logoutButtonText} // Add this style to the button
                />
              </View>
            </View>
          );
        }
      }

      const styles = StyleSheet.create({
        container: {
          flex: 1,
          display: 'flex',
          justifyContent: 'space-between',
```

```
    alignItems: 'flex-start', // Align items to the left
    padding: 35,
    backgroundColor: '#fff'
  },
  menu: {
    width: '100%',
    marginBottom: 20,
  },
  menuItem: {
    paddingVertical: 15,
    paddingHorizontal: 0,
    borderBottomWidth: 1,
    borderBottomColor: '#ccc',
  },
  menuItemText: {
    fontSize: 18,
    fontWeight: 'bold',
  },
  content: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  logoutButton: {
    marginBottom: 0,
    width: '100%', // Set the width to 100%
  },
  logoutButtonText: {
    width: '100%', // Set the width to 100% for the button text
  },
  textStyle: {
    fontSize: 30, // Adjust font size
    fontWeight: 'bold', // Optionally, make the text bold
    marginTop: 0, // Add margin from top
    marginLeft: 0, // Add margin from left
  }
});

//App
import * as React from 'react';
import { View, Text, StyleSheet, Image } from 'react-native';
```

```
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import LottieView from 'lottie-react-native'; // Import LottieView
import Login from './components/login';
import Signup from './components/signup';
import Dashboard from './components/dashboard';
import MarketScreen from './components/MarketScreen';
import PortfolioScreen from './components/PortfolioScreen';
import firebase from './database/firebase'; // Import firebase

const Stack = createStackNavigator();
const Tab = createBottomTabNavigator();

function DashboardStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name="DashboardMain"
        component={Dashboard}
        options={{
          title: null,
          headerLeft: () => null, // This will remove the back arrow
          headerShown: false // This will hide the header
        }}
      />
    </Stack.Navigator>
  );
}


function MainTabs() {
  return (
    <Tab.Navigator
      screenOptions={{
        tabBarActiveTintColor: '#003366',
        tabBarLabelStyle: {
          fontWeight: 'bold',
          fontSize: 15,
          marginBottom: 5
        },
```

```
        tabBarStyle: {
          backgroundColor: '#fff',
          borderTopLeftRadius: 20, // Top left radius
          borderTopRightRadius: 20, // Tab border radius
          height: 65, // Increase the height of the tab bar
        },
      }}
    >
      <Tab.Screen
        name="Market"
        component={MarketScreen}
        options={{
          tabBarIcon: ({ focused }) => (
            <Image
              source={require('./assets/chart-histogram.png')}
              style={{
                width: 20,
                height: 20,
                tintColor: focused ? '#003366' : '#aaa', // Adjust color based on
tab focus

              }}
            />
          ),
          tabBarLabel: 'MARKET',
          headerTitle: 'MARKET',
          headerTitleAlign: 'center',
          headerStyle: {
            backgroundColor: '#003366', // navy blue color
          },
          headerTintColor: '#fff',
          headerTitleStyle: {
            fontWeight: 'bold',
          },
        }}
      />
      <Tab.Screen
        name="Portfolio"
        component={PortfolioScreen}
        options={{
          tabBarIcon: ({ focused }) => (
            <Image
```

```
                    source={require('./assets/briefcase.png')}
                    style={{
                      width: 20,
                      height: 20,
                      tintColor: focused ? '#003366' : '#aaa', // Adjust color based on
tab focus
                    }}
                  />
                ),
                tabBarLabel: 'PORTFOLIO',
                headerTitle: 'PORTFOLIO',
                headerTitleAlign: 'center',
                headerStyle: {
                  backgroundColor: '#003366', // navy blue color
                },
                headerTintColor: '#fff',
                headerTitleStyle: {
                  fontWeight: 'bold',
                },
              }}
            />
            <Tab.Screen
              name="Profile"
              component={DashboardStack}
              options={{
                tabBarIcon: ({ focused }) => (
                  <Image
                    source={require('./assets/user.png')}
                    style={{
                      width: 20,
                      height: 20,
                      tintColor: focused ? '#003366' : '#aaa', // Adjust color based on
tab focus
                    }}
                  />
                ),
                tabBarLabel: 'PROFILE',
                headerTitle: 'PROFILE',
                headerTitleAlign: 'center',
                headerStyle: {
                  backgroundColor: '#003366', // navy blue color
```

```
          },
          headerTintColor: '#fff',
          headerTitleStyle: {
            fontWeight: 'bold',
          },
        }}
      />
    </Tab.Navigator>
  );
}

function AuthStack() {
  return (
    <Stack.Navigator
      initialRouteName="Signup"
      screenOptions={{
        headerTitleAlign: 'center',
        headerStyle: {
          backgroundColor: '#3740FE',
        },
        headerTintColor: '#fff',
        headerTitleStyle: {
          fontWeight: 'bold',
        },
      }}>
      <Stack.Screen
        name="Signup"
        component={Signup}
        options={{ title: 'Signup' }}
      />
      <Stack.Screen
        name="Login"
        component={Login}
        options={{ title: 'Login' }}
      />
    </Stack.Navigator>
  );
}

export default function App() {
  const [initializing, setInitializing] = React.useState(true);
```

```
          const [user, setUser] = React.useState();
          const [isLoading, setIsLoading] = React.useState(true); // State for loading
          const [isLoggedIn, setIsLoggedIn] = React.useState(false); // State for logged in
status

          function onAuthStateChanged(user) {
            setUser(user);
            if (initializing) setInitializing(false);
          }

          React.useEffect(() => {
            const subscriber = firebase.auth().onAuthStateChanged(onAuthStateChanged);
            return subscriber; // unsubscribe on unmount
          }, []);

          // Simulate loading
          React.useEffect(() => {
            setTimeout(() => {
              setIsLoading(false);
            }, 3000);
          }, []);

          if (initializing || isLoading) { // Show loading animation if initializing or
loading
            return (
              <View style={styles.loadingContainer}>
                <LottieView
                  source={require('./assets/loading.json')}
                  autoPlay
                  loop
                  style={styles.animation}
                />
                <Text style={styles.loadingText}>Loading...</Text>
              </View>
            );
          }

          return (
            <NavigationContainer>
              {user ? <MainTabs /> : <AuthStack />}
            </NavigationContainer>
```

```
  );
}

const styles = StyleSheet.create({
  loadingContainer: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  animation: {
    width: 200,
    height: 200,
  },
  loadingText: {
    marginTop: 20,
    fontSize: 20,
  },
});
```

Додаток Г – Ілюстративна частина

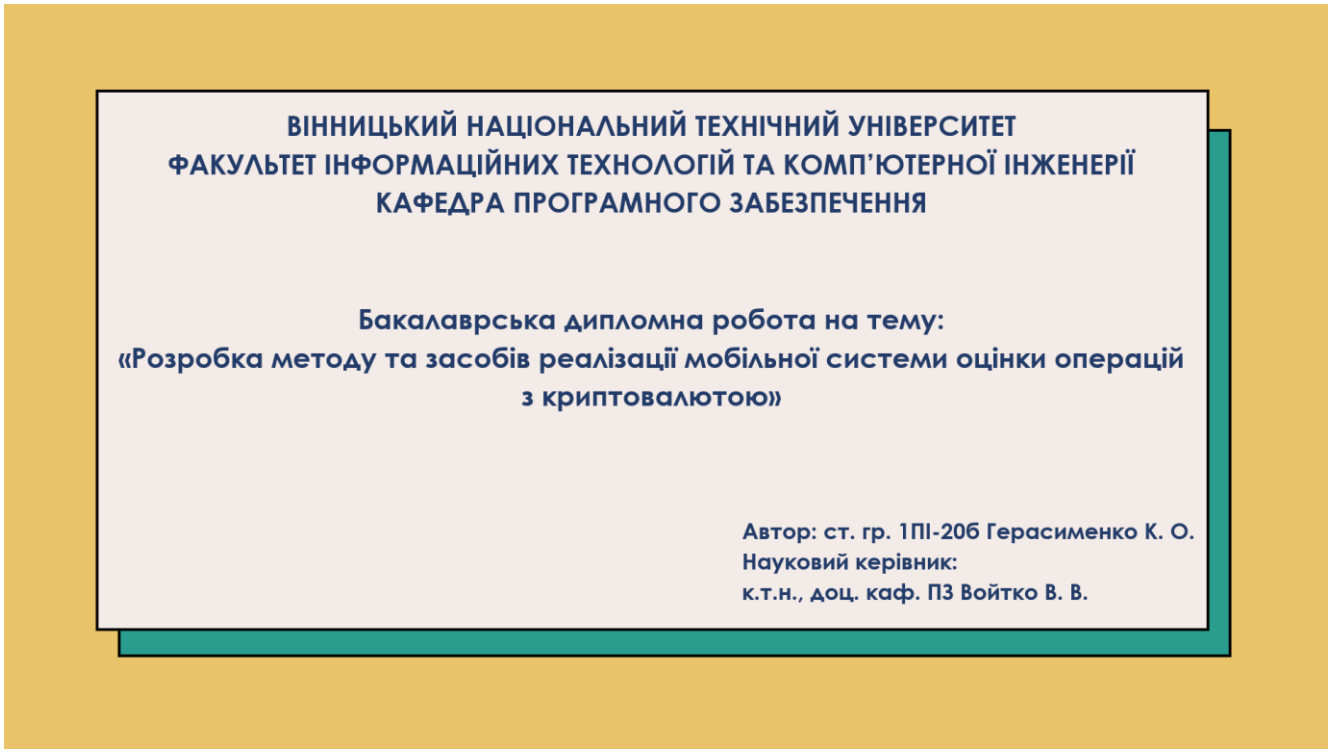

Рисунок Г.1 — Титульний слайд

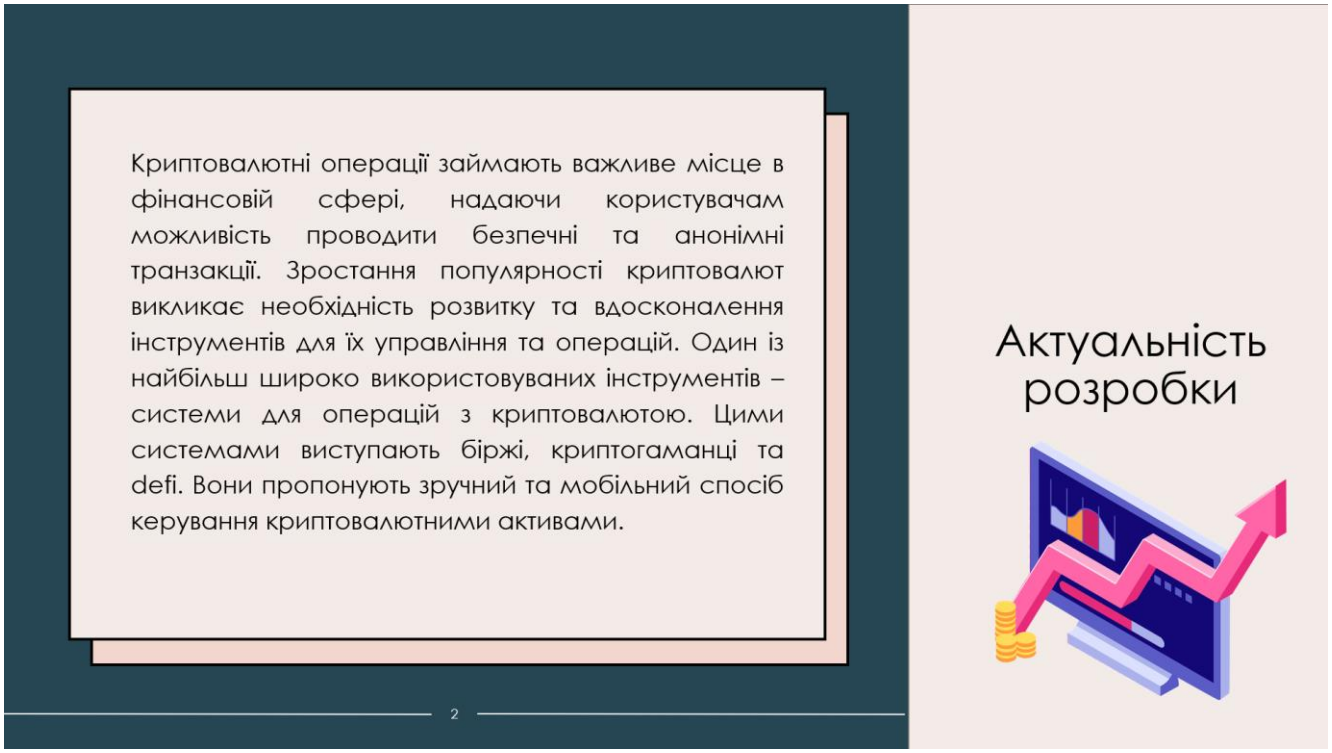

Рисунок Г.2 — Актуальність розробки

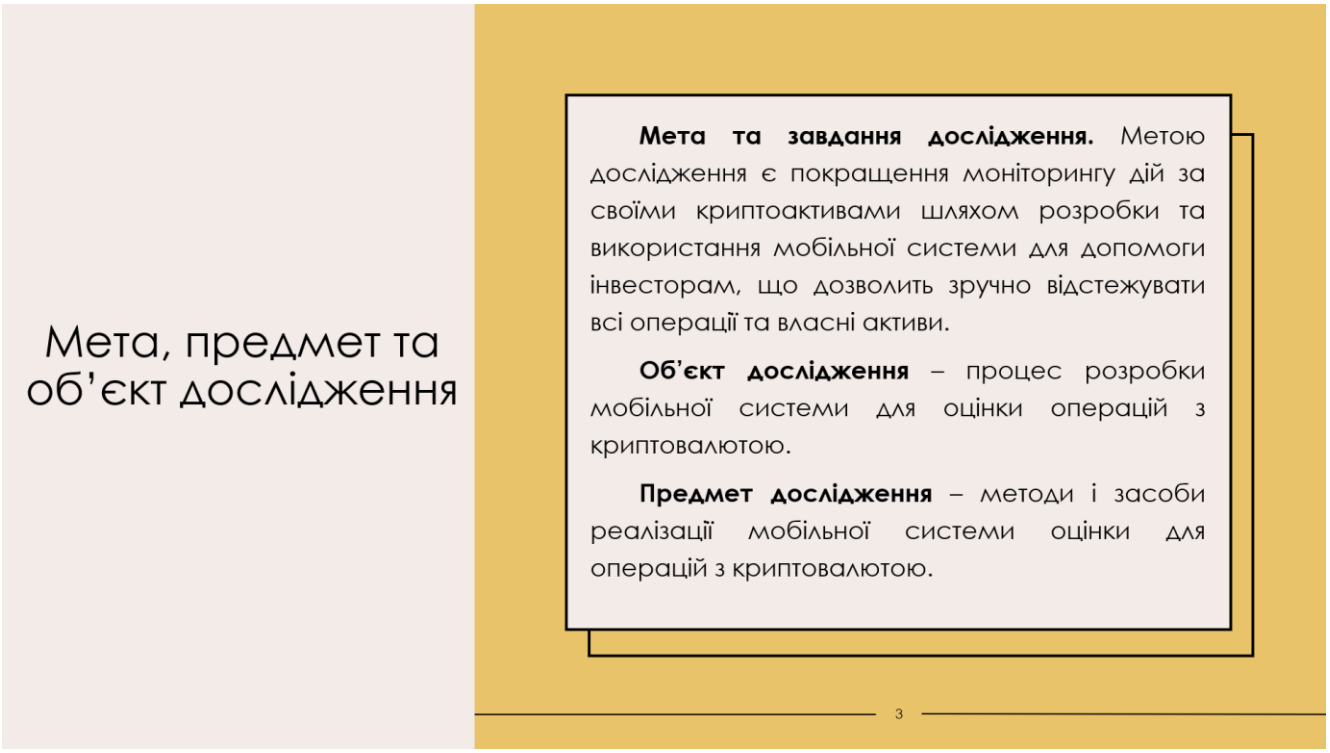

Рисунок Г.3 — Мета, предмет та об’єкт дослідження

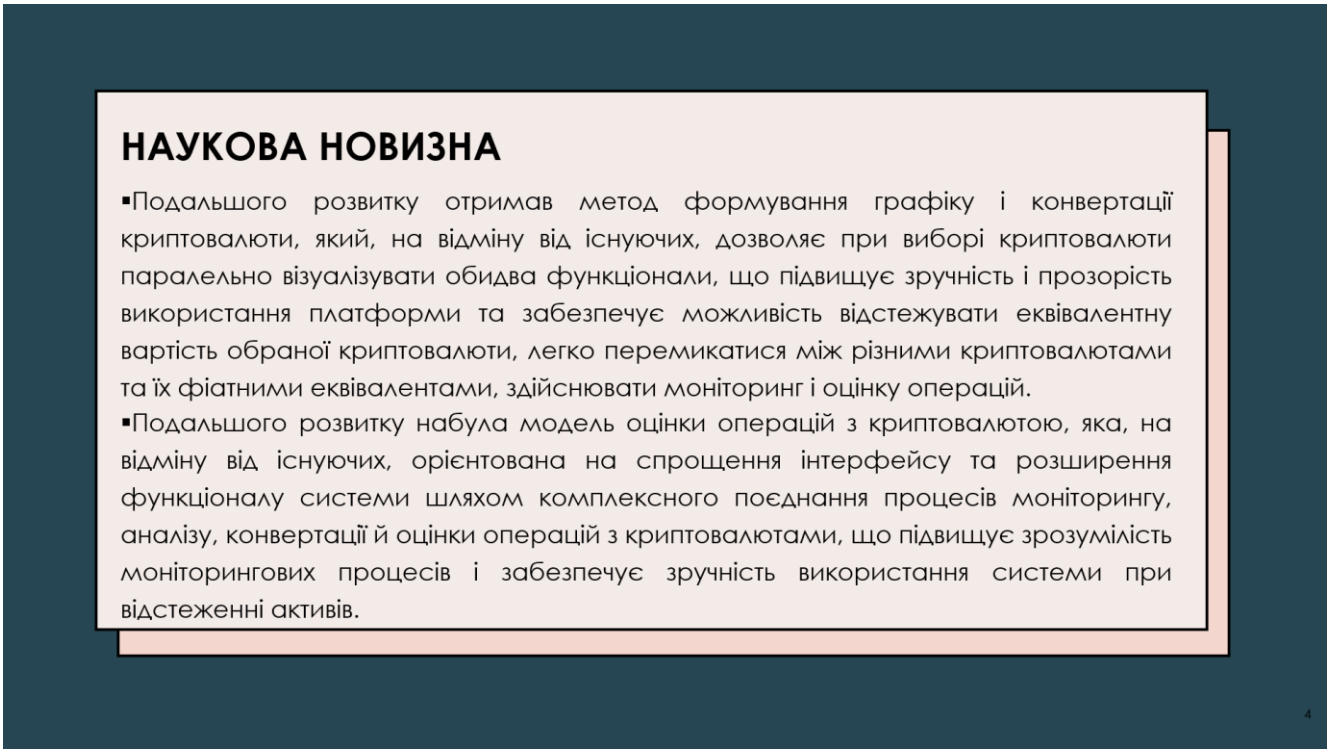

Рисунок Г.4 — Наукова новизна

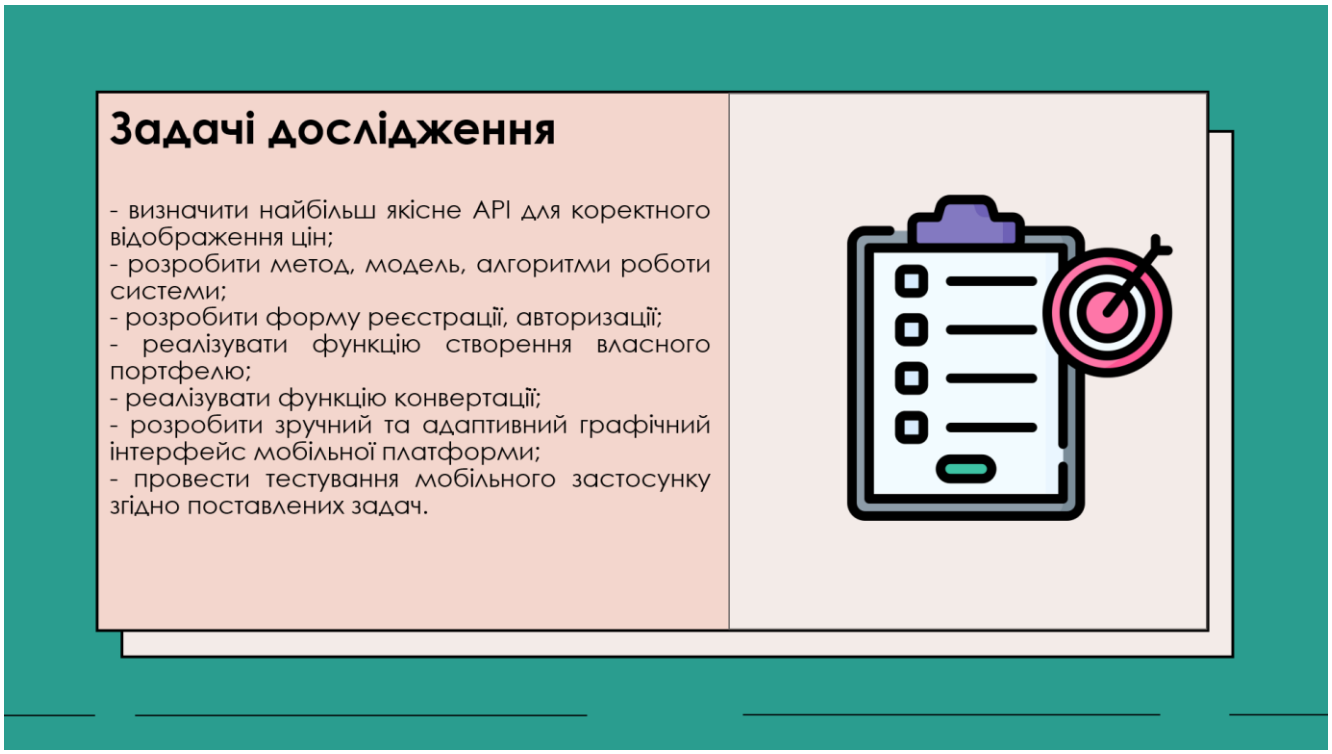

Рисунок Г.5 — Задачі дослідження

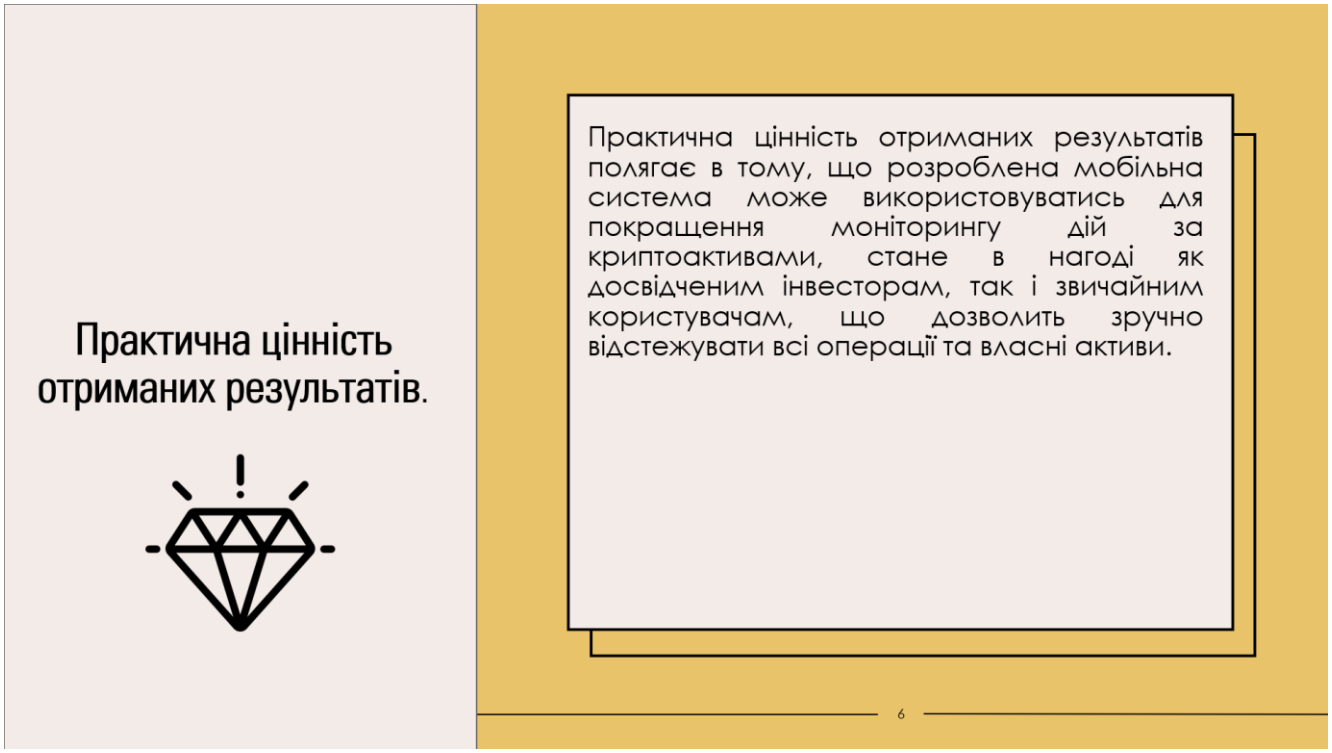

Рисунок Г.6 — Практична цінність отриманих результатів

Рисунок Г.7 — Порівняння аналогів

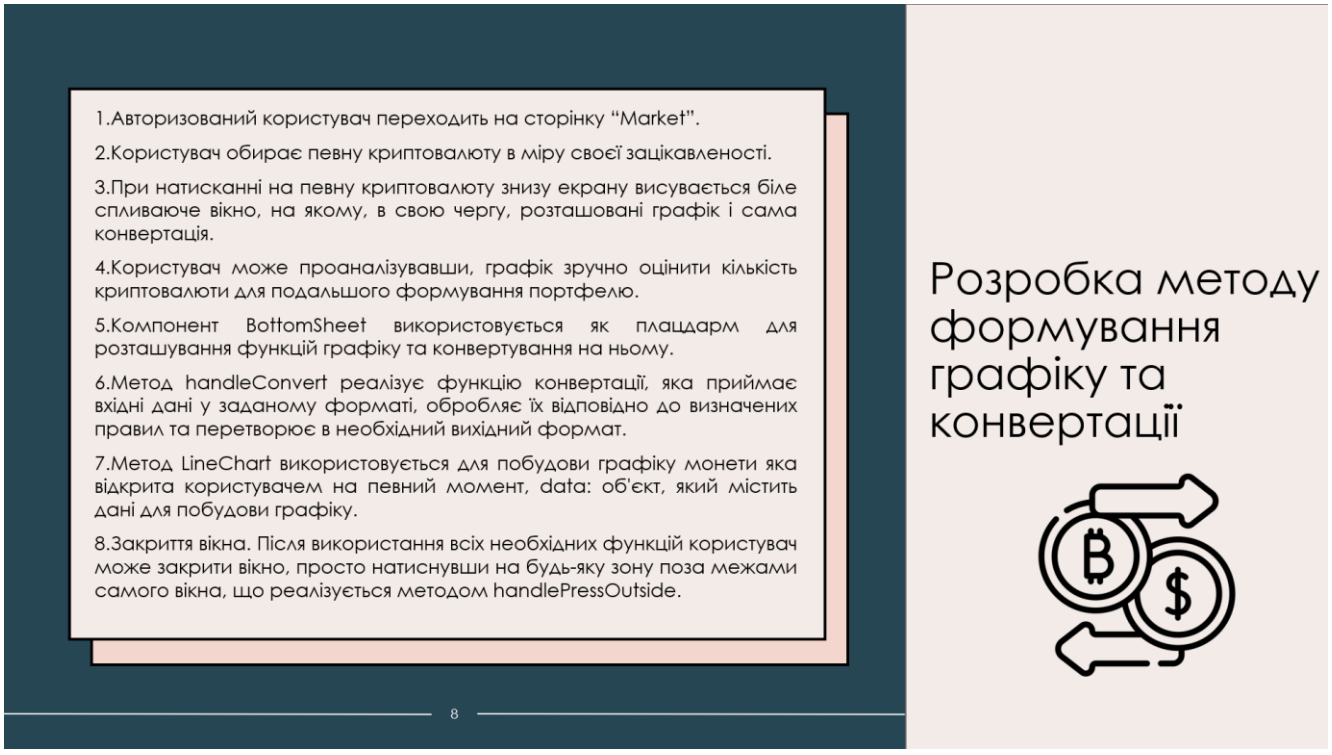

Рисунок Г.8 — Розробка методу формування графіку та конвертації

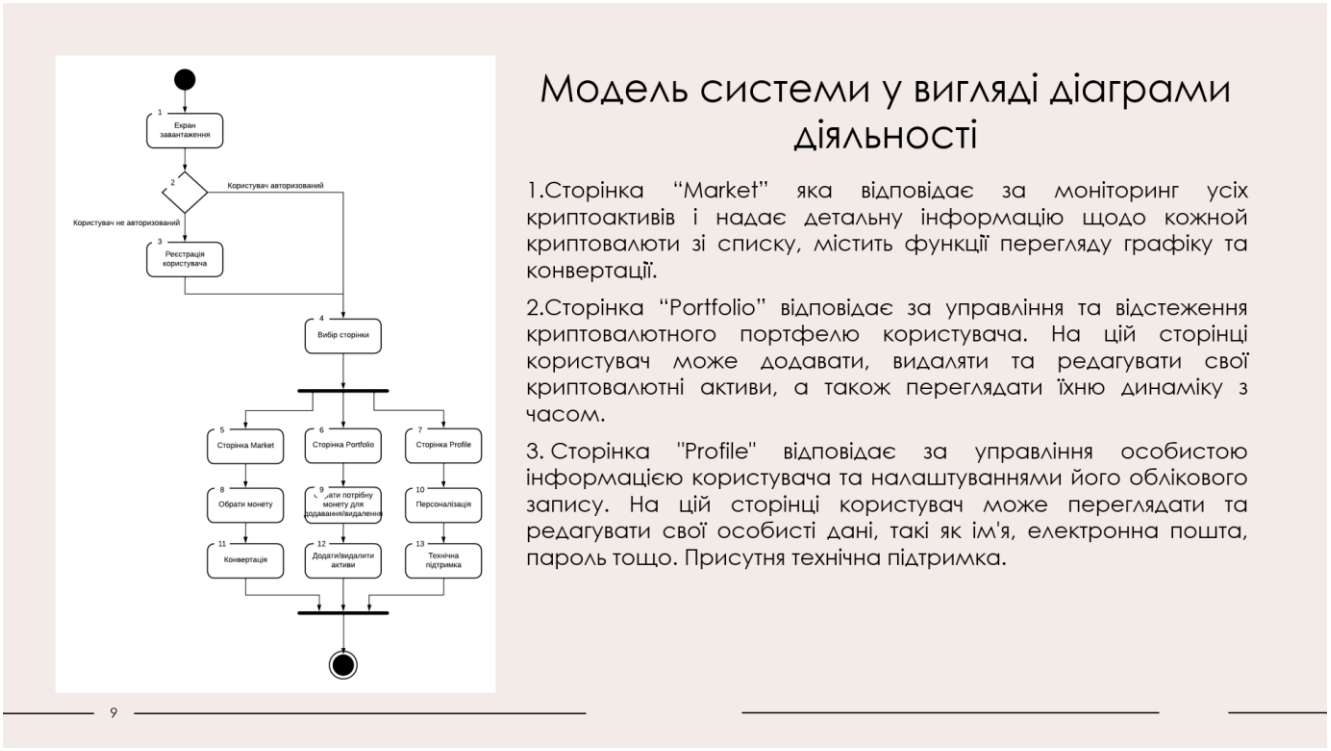

Рисунок Г.9 — Модель системи у вигляді діаграми діяльності

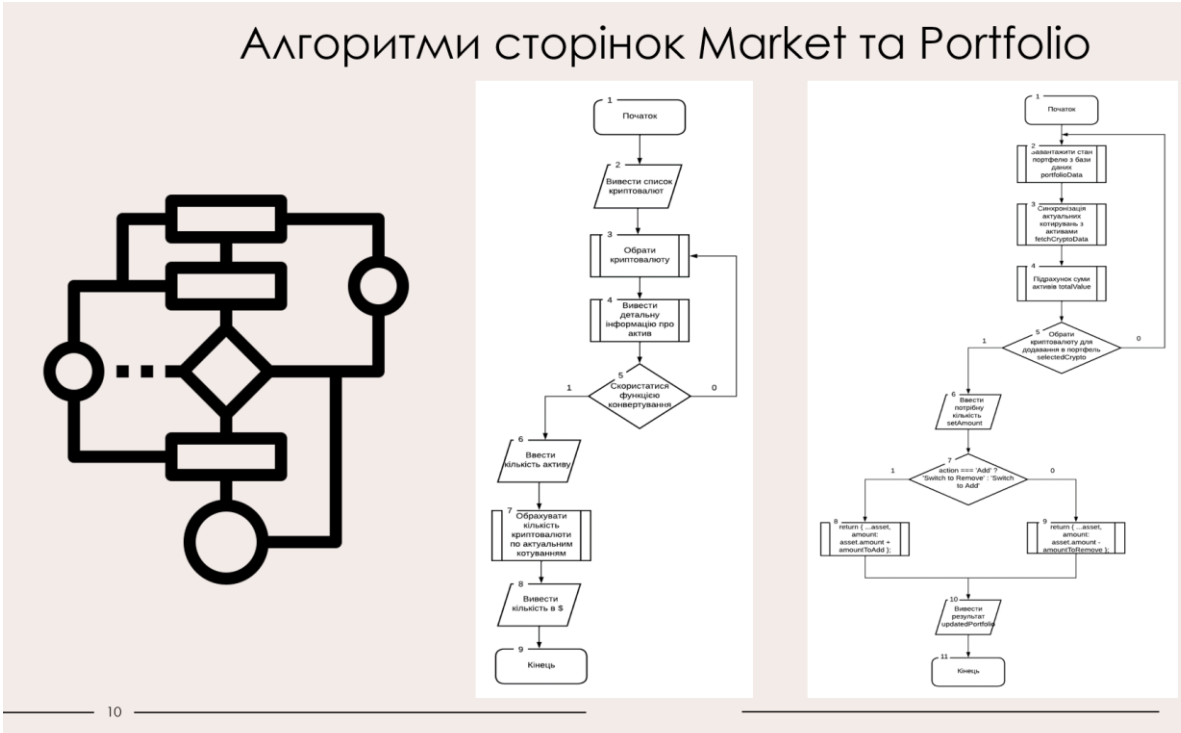

Рисунок Г.10 — Алгоритми сторінок Market та Portfolio

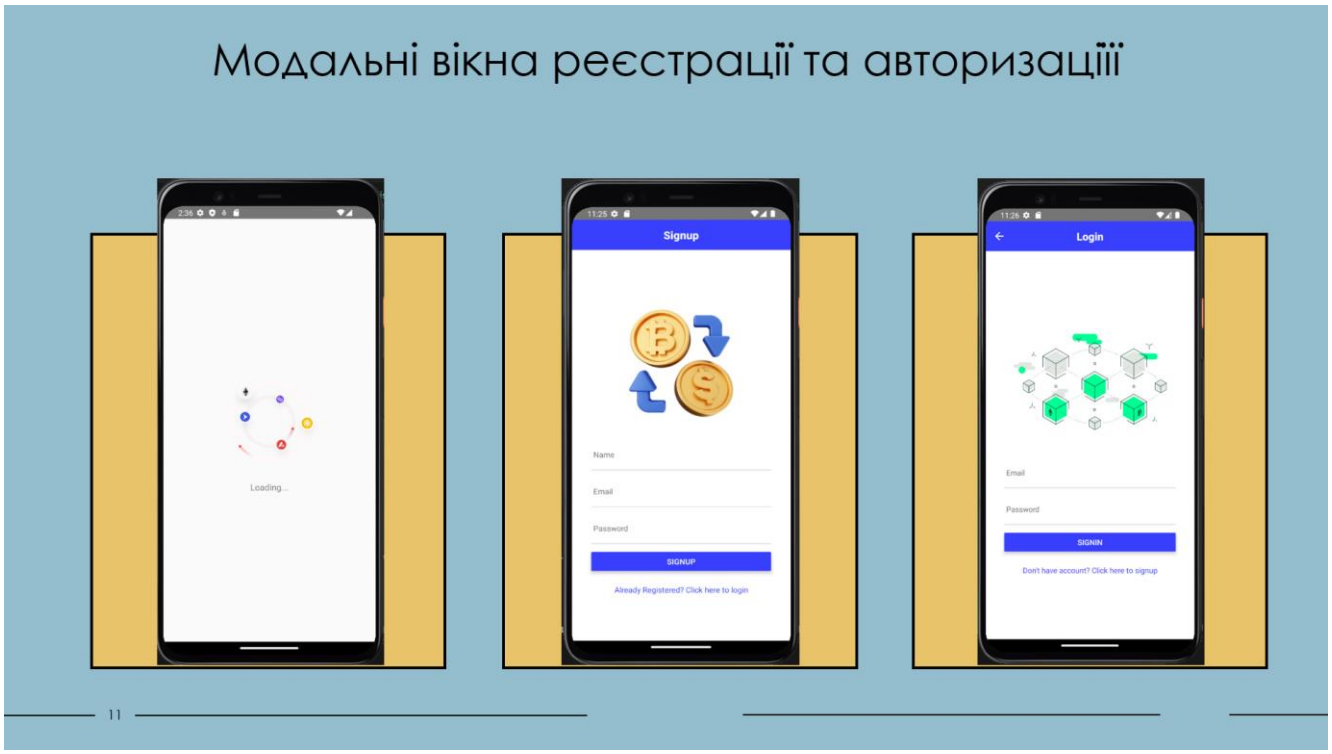

Рисунок Г.11 — Модальні вікна реєстрації та авторизації

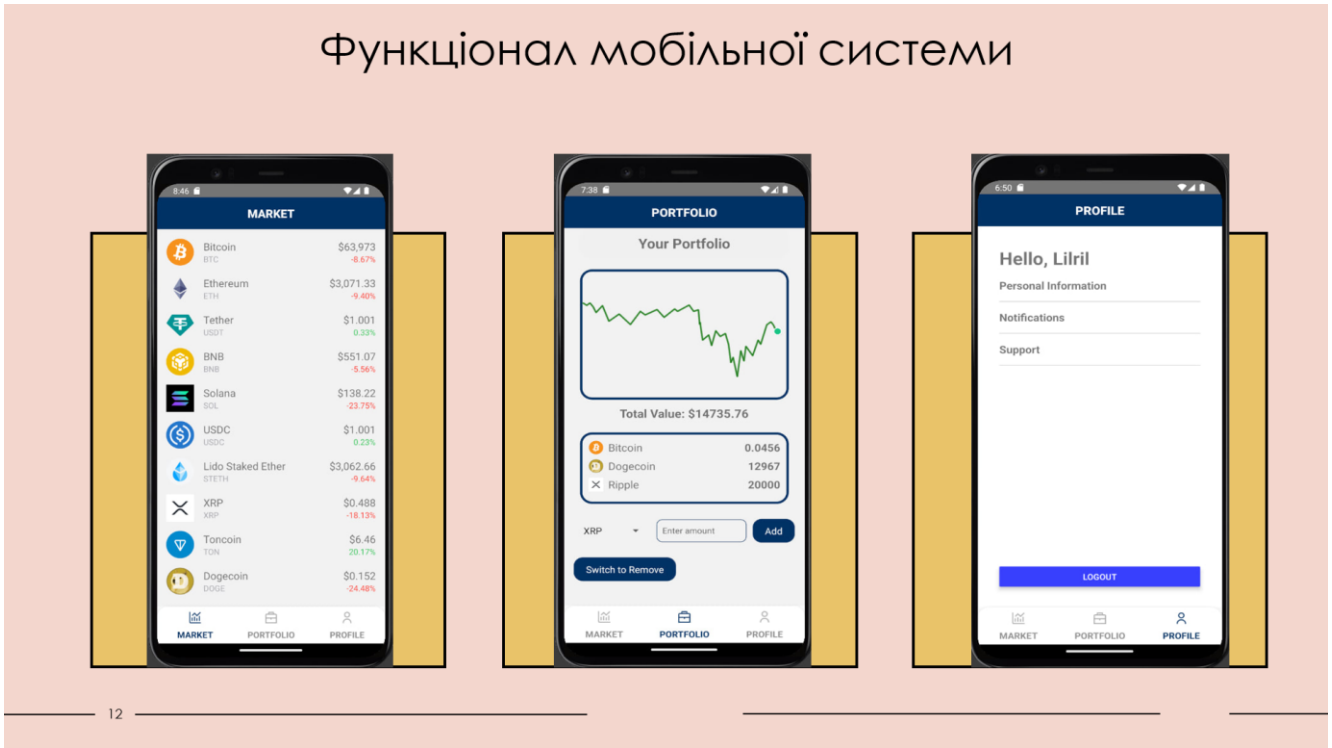

Рисунок Г.12 — Функціонал мобільної системи

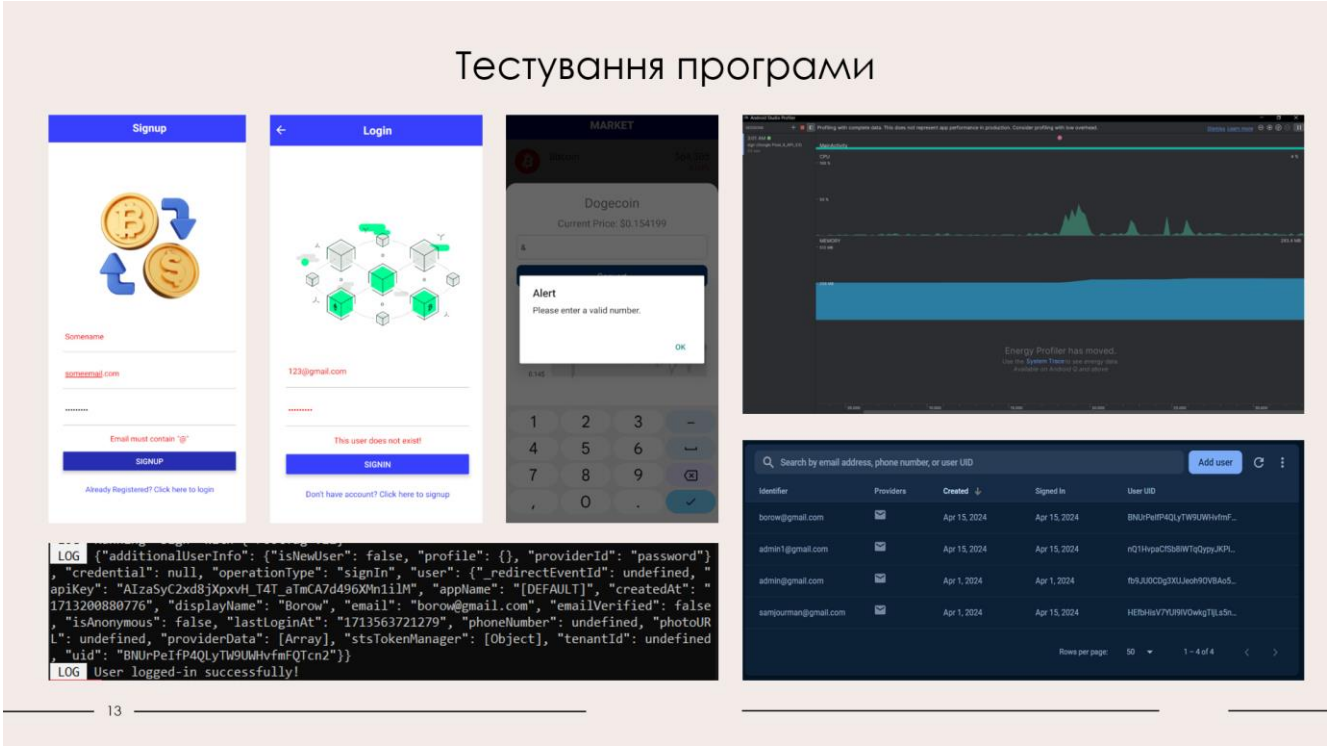

Рисунок Г.13 — Тестування програми

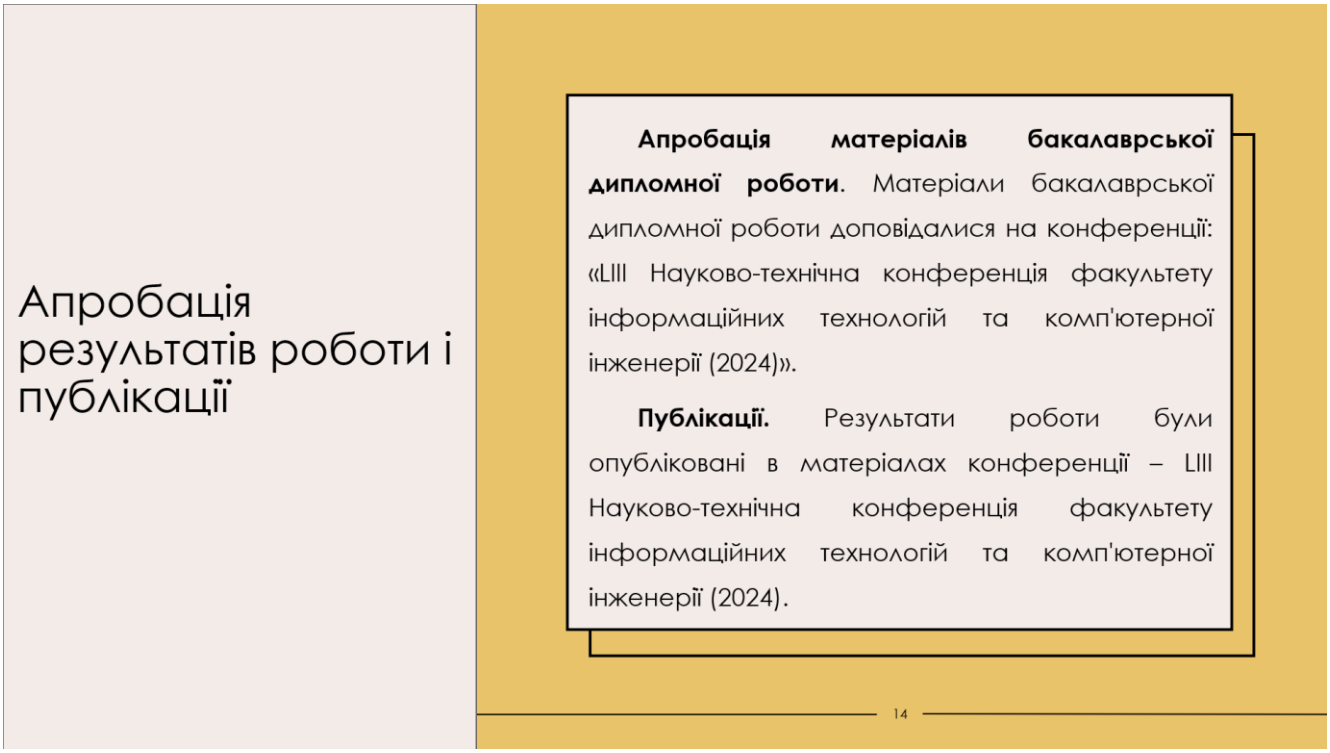

Рисунок Г.14 — Апробація результатів роботи і публікації

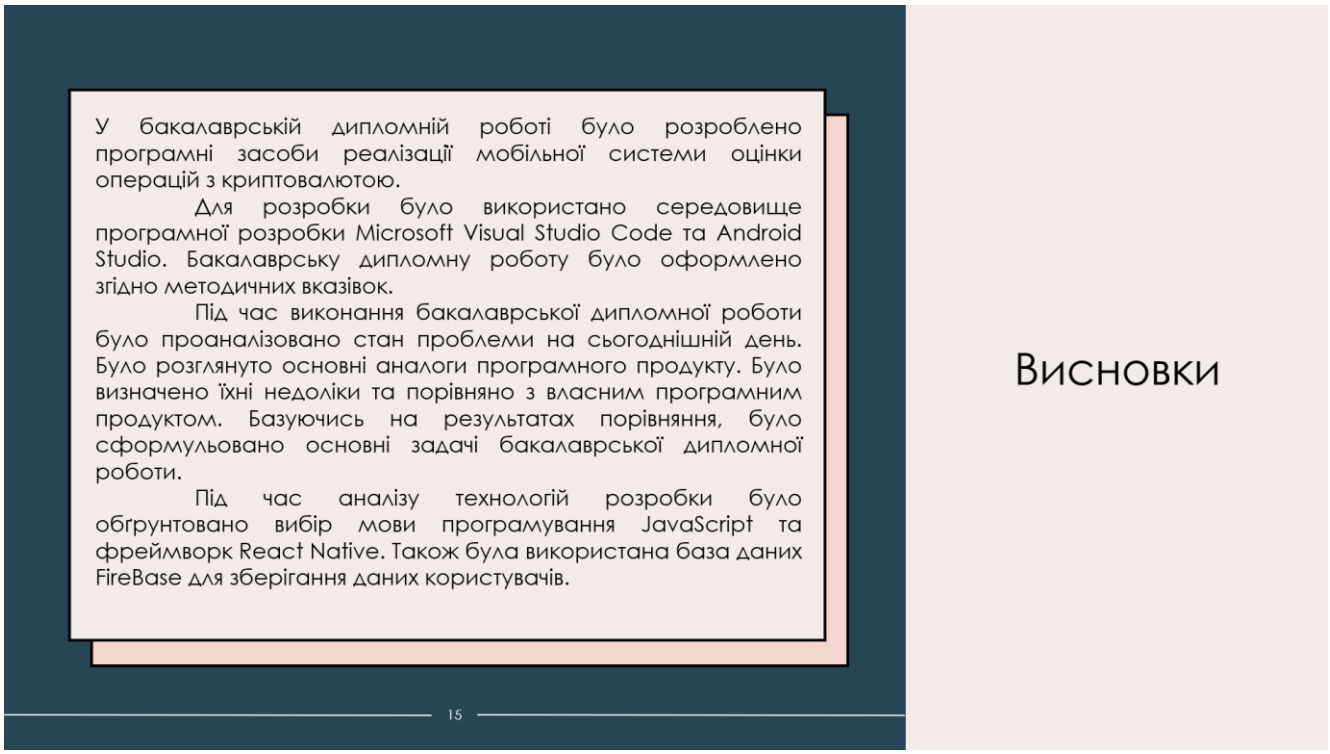

Рисунок Г.15 — Висновки (перша частина)

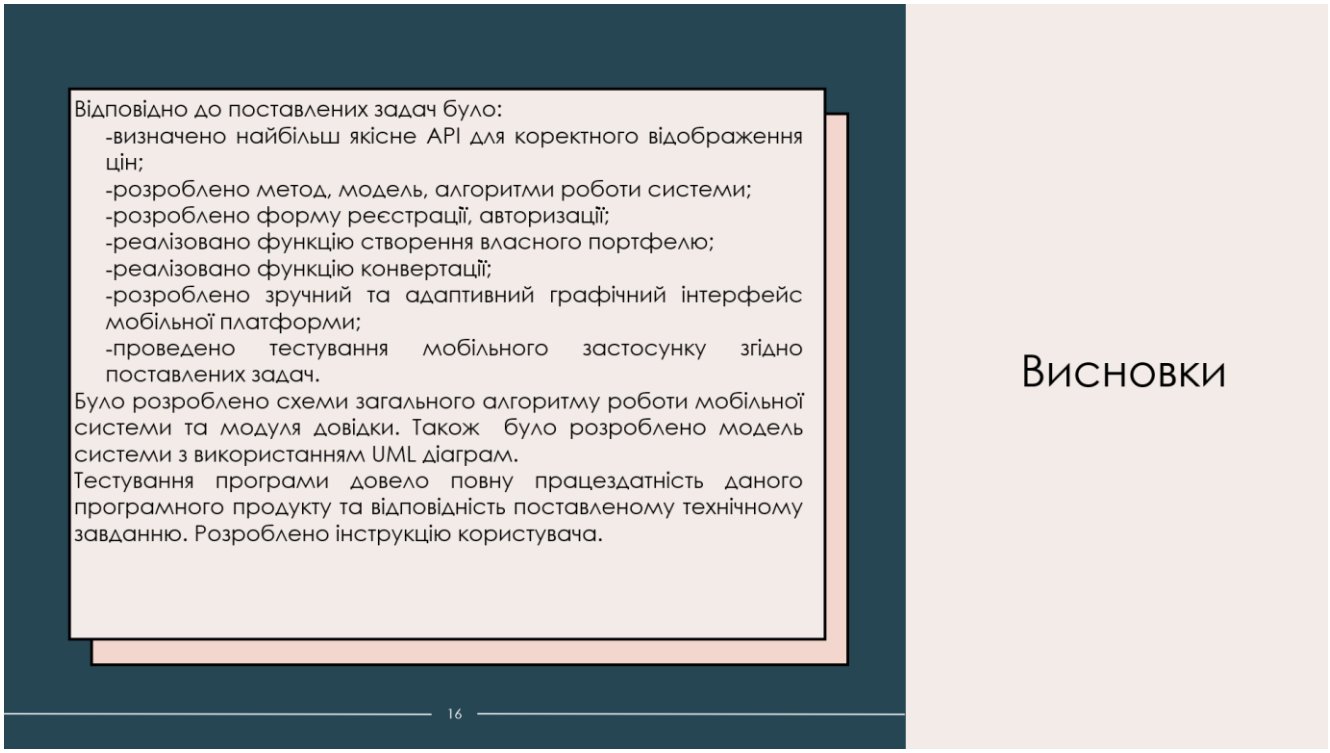

Рисунок Г.16 — Висновки (друга частина)