

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО IOS-ЗАСТОСУНКУ ДЛЯ
ФОРМУВАННЯ ВЛАСНОГО КРИПТОПОРТФЕЛЯ
DEVELOPMENT OF A MOBILE IOS APPLICATION TO
CREATE YOUR OWN CRYPTO PORTFOLIO

Виконав(ла): студент(ка) 4 курсу, групи КНТ-129
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ФАЛАЛЄЄВ О.О.

(ПРИЗВИЩЕ та ініціали)

Керівник

КОЛПАКОВА Т.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент

КОРОТКИЙ О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.

Сергій СУББОТІН

« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ФАЛАЛЄЄВА Олега Олексійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного iOS-застосунку для формування власного криптопортфеля. Development of a Mobile iOS Application to Create Your Own Crypto Portfolio.

керівник проєкту (роботи) к.т.н., доцент, КОЛПАКОВА Тетяна Олексіївна,

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 01 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Огляд та вибір технологій. 3. Розробка застосунку. 4. Експлуатація та тестування застосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КОЛПАКОВА Т.О., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання « 04 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Олег ФАЛАЛЄСВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Тетяна КОЛПАКОВА
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
131 с., 3 табл., 29 рис., 2 дод., 14 джерел.

КРИПТОПОРТФЕЛЬ, МОБІЛЬНИЙ iOS ЗАСТОСУНОК, SWIFT,
XCODE, КРИПТОВАЛЮТИ.

Об'єкт дослідження – процес управління станом криптовалютного портфелю.

Предмет дослідження – мобільний застосунок на платформі iOS для управління криптовалютним портфелем.

Мета роботи – створення надійного, зручного та ефективного мобільного застосунку на платформі iOS для формування та управління власним криптопортфелем на мові програмування Swift.

Матеріали, методи та технічні засоби: мова програмування Swift розробки мобільних застосунків на платформі iOS, актуальні дані про криптовалютні ринки та технічні аспекти роботи з криптовалютами.

Результати. Розроблено мобільний застосунок для керування криптовалютним портфелем на платформі iOS, виконано інтеграцію з популярним криптовалютним маркетом CoinGecko.

Висновки. У процесі розробки мобільного застосунку для керування криптовалютним портфелем на платформі iOS на мові програмування Swift було досліджено сучасні технології, що дозволили створити зручний для користувачів застосунок. Розроблений додаток надає можливість отримувати статистичні дані та сповіщення про зміни цін, що сприяє прийняттю обґрунтованих рішень.

Галузь використання – проєкт може бути використаний як мобільне рішення управління активами для інвесторів та трейдерів у фінансовій сфері.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 131 pages, 3 tables, 29 figures, 2 appendixes, 14 sources.

CRYPTO WALLET, iOS MOBILE APPLICATION, SWIFT, XCODE, CRYPTOCURRENCIES.

Object of study – the process of managing the state of a cryptocurrency portfolio.

Subject of study – a mobile application on the iOS platform for managing a cryptocurrency portfolio.

The purpose of the work – to create a reliable, convenient, and efficient mobile application on the iOS platform for forming and managing a personal crypto wallet using the Swift programming language.

Materials, methods, and tools: Swift programming language for developing mobile applications on the iOS platform, current data on cryptocurrency markets, and technical aspects of working with cryptocurrencies.

Results. Developed a mobile application for managing a cryptocurrency portfolio on the iOS platform, integrated with the popular cryptocurrency market CoinGecko.

Conclusions. In the process of developing a mobile application for managing a cryptocurrency portfolio on the iOS platform using the Swift programming language, modern technologies were explored, allowing for the creation of a user-friendly application. The developed application provides the ability to receive statistical data and notifications about price changes, which contributes to making informed decisions.

The field of use – the project can be used as a mobile asset management solution for investors and traders in the financial sector.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	8
Вступ	9
1 Аналіз предметної області	11
1.1 Поняття криптовалюти	11
1.2 Блокчейн-технологія	12
1.3 Основні криптовалюти та їх відмінності	14
1.4 Криптовалютні гаманці	16
1.5 Майнінг криптовалют	17
1.6 Транзакції та криптовалютні біржі	18
1.7 Висновки за розділом 1	20
2 Огляд та вибір технологій	21
2.1. Огляд технологій розробки для платформи iOS	21
2.2 Мови програмування: Swift та Objective-C	23
2.3 Розробка застосунків з використанням UIKit та SwiftUI	25
2.4 Використання CocoaPods та Swift Package Manager для управління залежностями	26
2.5 Бази даних	27
2.6 Висновки за розділом 2	29
3 Розробка застосунку	31
3.1 Розробка інтерфейсу застосунку	31
3.2 Реєстрація та авторизація користувача	36
3.3 База даних	37
3.4 Робота з API	40
3.5 Імплементування бібліотеки Charts для роботи з графіками	42
3.6 Висновки за розділом 3	44
4 Експлуатація та тестування застосунку	46
4.1 Призначення застосунку	46
4.2 Умови запуску застосунку	46
4.3 Тестування застосунку	46

4.4 Експлуатація застосунку	47
4.5 Висновки за розділом 4	56
Висновки	57
Перелік джерел посилань	58
Додаток А Текст програми	60
Додаток Б Слайди презентації	125

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AML – Anti-Money Laundering;
API – Application Programming Interface;
ASIC – інтегральна схема для специфічного застосування;
CEX – централізована біржа;
DAO – децентралізовані фінансові організації;
DApps – децентралізовані додатки;
DeFi – децентралізоване фінансування;
DEX – децентралізована біржа;
GPU – графічний процесор;
GUI – графічного інтерфейсу користувача;
IDE – інтегроване середовище розробки;
KYC – Know Your Customer;
PoS – Proof of Stake;
PoW – Proof of Work;
SPM – Swift Package Manager.

ВСТУП

Сучасний світ фінансів переживає епоху значних трансформацій, зокрема з розвитком криптовалютних ринків та блокчейн-технологій. Відповідно до зростання популярності цифрових активів, інвестори та трейдери постійно шукають ефективні інструменти для управління своїми криптопортфелями. Мобільні додатки, які дозволяють зручно та безпечно стежити за ринком, проводити операції та аналізувати інвестиційні стратегії, стають все більш важливими у цьому контексті.

Проте, незважаючи на наявність численних мобільних рішень для керування криптопортфелями, не всі з них забезпечують достатню простоту використання, швидкість реагування та безпеку даних користувачів. Тому, розробка нових мобільних додатків, що відповідають вимогам сучасних інвесторів та трейдерів, має значний потенціал для розвитку та успіху на ринку.

Дипломна кваліфікаційна робота бакалавра на тему "Розробка мобільного iOS-застосунку для формування власного криптопортфеля" спрямований на створення надійного, зручного та ефективного мобільного застосунку для управління криптовалютним портфелем на платформі iOS. Розроблюваний застосунок дозволить користувачам створювати та управляти своїми криптопортфелями, відстежувати свої інвестиції, проводити транзакції та оптимізувати свої стратегії на криптовалютних ринках з максимальною зручністю та ефективністю. Застосунок буде інтегрований з популярним маркетом та, а також надаватиме статистичні дані та сповіщення про зміни цін для допомоги користувачам при прийнятті інвестиційних рішень.

Дипломна кваліфікаційна робота бакалавра буде виконана з використанням сучасних методів аналізу, проєктування, розробки та тестування програмного забезпечення. Розроблений застосунок матиме потенціал для подальшого розвитку та розширення функціональності, включаючи підтримку інших платформ, співпрацю між користувачами та інші можливості, що можуть

полегшити управління криптовалютами активами та підвищити ефективність інвестиційних стратегій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття криптовалюти

Криптовалюта – це цифровий чи віртуальний актив, який використовує криптографічні принципи для забезпечення безпеки транзакцій та контролю за створенням нових одиниць. Криптовалюти функціонують на основі децентралізованих мереж, які базуються на технології блокчейн – відкритому реєстрі, який містить інформацію про всі транзакції, що відбулися в мережі.

Основні характеристики криптовалют включають:

- децентралізація – відсутність центрального органу, який контролює криптовалютну мережу, дозволяє забезпечити незалежність від політичних та економічних чинників;

- анонімність – криптовалюти дозволяють здійснювати транзакції без розкриття особистої інформації користувачів, хоча рівень анонімності може відрізнятися в залежності від конкретної криптовалюти;

- незмінність – після проведення транзакції в блокчейні, її неможливо змінити або видалити, що забезпечує високий рівень безпеки та надійності;

- відкритість – будь-яка особа може приєднатися до криптовалютної мережі та стати учасником процесу створення нових блоків та підтвердження транзакцій;

- майнінг – процес створення нових блоків у блокчейні та винагородження майнерів криптовалютою за затрачений час та ресурси.

Криптовалютні активи можуть використовуватися для здійснення платежів та переказів коштів, інвестицій, децентралізованого фінансування (DeFi), децентралізованих додатків (DApps) та інших фінансових операцій. Криптовалюти дозволяють здійснювати транзакції з більшою швидкістю та нижчими комісіями, порівняно з традиційними фінансовими системами [1].

Однак, криптовалютні ринки є волатильними, і їх вартість може значно коливатися протягом короткого часу. Тому, перед використанням криптовалют,

користувачам слід ретельно вивчити ризики, а також правила та законодавство, що регулюють їх використання в їх країні.

1.2 Блокчейн-технологія

Блокчейн – це технологія децентралізованого відкритого реєстру, яка забезпечує зберігання та передачу інформації про транзакції в мережі. Блокчейн складається з ланцюжка блоків, кожен з яких містить групу транзакцій. Кожен блок містить унікальний хеш, що посилається на попередній блок у ланцюжку, тим самим створюючи безперервну послідовність блоків (рис. 1.1) [2].

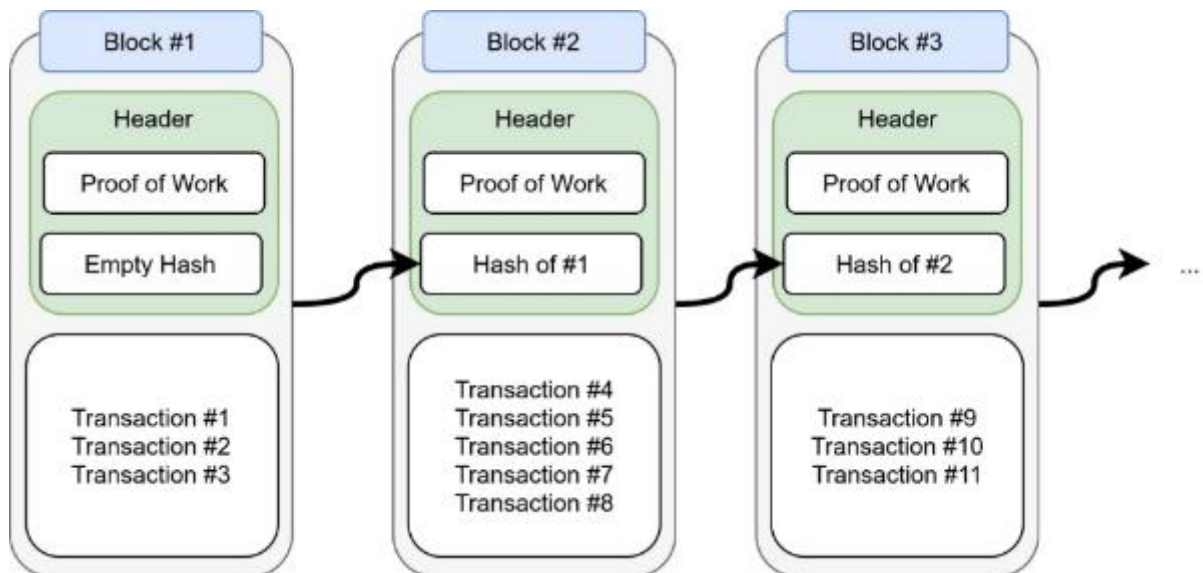


Рисунок 1.1 – Приклад блокчейну [2]

Основні принципи роботи блокчейну:

– консенсус – для додавання нового блоку до ланцюжка потрібно досягти консенсусу серед учасників мережі. Різні криптовалютні мережі використовують різні протоколи консенсусу, такі як Proof of Work (PoW), Proof of Stake (PoS) та інші;

– незмінність – одного разу доданий до ланцюжка блок стає незмінним. Це означає, що інформація про транзакції не може бути змінена чи видалена без зміни всіх наступних блоків, що є майже неможливим завданням;

– транспарентність – блокчейн є відкритим і публічним, що дозволяє будь-якому користувачеві мережі перевірити транзакції та переконатися в їх вірогідності;

– децентралізація – блокчейн не має центральної точки контролю, оскільки копії реєстру зберігаються на вузлах мережі, що забезпечує незалежність від єдиного провайдера та зменшує ризик збоїв.

Порівняння протоколів консенсусу наведено в таблиці (табл. 1.1).

Таблиця 1.1 – Порівняння протоколів консенсусу

Протоколи	Proof of Work (PoW)	Proof of Stake (PoS)
Опис	Доказ роботи	Доказ співпраці
Процес доказування	Майнінг	Холдінг
Споживання енергії	Дуже високе	Низьке
Складність налаштування	Висока	Низька
Швидкість підтвердження	Повільна	Швидка
Ризик централізації	Високий	Менший
Витрати на обладнання	Високі	Не потрібне
Безпека	Дуже висока	Висока
Ресурсозалежність	Сильна залежність від обладнання та енергії	Не залежить від обладнання та енергії
Приклад блокчейнів	Bitcoin, Ethereum Classic, Litecoin	Ethereum, Cardano, Algorand

Продовження таблиці 1.1

Протоколи	Proof of Work (Pow)	Proof of Stake (PoS)
Перспективність	Менш перспективний в контексті сталої екології	Більш перспективний в контексті сталої екології, дуже багато протоколів розглядають PoS як альтернативу PoW

Блокчейн технологія лежить в основі криптовалют та інших децентралізованих застосунків, таких як децентралізовані фінансові сервіси (DeFi), децентралізовані автономні організації (DAO), смарт-контракти та токенизовані активи. Використання блокчейну не обмежується фінансовою сферою і поширюється на такі галузі, як логістика, медицина, право, освіта, нерухомість та багато інших.

Деякі переваги використання блокчейну включають:

- зниження витрат на проведення транзакцій, оскільки відсутність посередників зменшує комісійні відрахування та інші витрати;
- збільшення швидкості та ефективності транзакцій, оскільки вони можуть бути проведені напряму між сторонами без затримок, пов'язаних з перевіркою та підтвердженням;
- забезпечення безпеки та надійності, оскільки блокчейн заснований на криптографічних принципах та незмінності записів, що зменшує ризик шахрайства та маніпуляцій.

1.3 Основні криптовалюти та їх відмінності

З моменту створення першої криптовалюти, Біткоїна, в 2009 році, кількість цифрових активів значно зросла. На сьогоднішній день існує більше 10,000 криптовалют, кожна з яких має свої відмінності та особливості. У цьому

підрозділі розглянемо деякі з найпопулярніших криптовалют та їх характеристики.

Біткоїн – перша криптовалюта, яка була створена у 2009 році під псевдонімом Сатоші Накамото. Біткоїн використовує технологію блокчейн на основі PoW для забезпечення децентралізації, безпеки та анонімності транзакцій. BTC часто вважається "цифровим золотом" та є найбільш популярною та цінною криптовалютою на ринку.

Ефіріум – криптовалюта, створена у 2015 році Віталіком Бутеріним, яка відрізняється від Біткоїна можливістю створення смарт-контрактів та DApps. Технологія блокчейн ефіріуму базується на алгоритмі Proof of Work, але має плани переходу на алгоритм PoS для збільшення ефективності та зменшення енергоспоживання.

Binance Coin – це криптовалюта, розроблена Binance, однією з найбільших криптовалютних бірж у світі. BNB використовується для оплати комісій на біржі та взаємодії з екосистемою Binance. Binance Coin базується на технології блокчейн Binance Smart Chain, яка підтримує смарт-контракти та сумісна з Ефіріум.

Cardano – криптовалюта, заснована на науковому підході до розробки блокчейн технології та створена співзасновником Ефіріума Чарльзом Хоскінсоном. Cardano використовує алгоритм консенсусу Proof of Stake з назвою Ouroboros, що забезпечує високий рівень ефективності та безпеки. Основні переваги Cardano включають смарт-контракти, шкальованість та стійкість до зловживань.

Polkadot – криптовалюта, створена для забезпечення міжблокчейнної взаємодії та сумісності між різними блокчейнами. Розроблена Гаваном Вудом, одним зі співзасновників Ефіріума, Polkadot допомагає передавати інформацію та активи між різними блокчейнами, тим самим сприяючи створенню єдиної децентралізованої інфраструктури.

1.4 Криптовалютні гаманці

Криптовалютні гаманці дозволяють користувачам зберігати та управляти своїми цифровими активами. Гаманці є невід'ємною складовою криптовалютної екосистеми, оскільки вони забезпечують безпечне зберігання, відправлення та отримання криптовалют. У цьому підрозділі ми розглянемо основні типи криптовалютних гаманців та їх відмінності.

Гарячі гаманці – це криптовалютні гаманці, які працюють онлайн та постійно з'єднані з інтернетом. Цей тип гаманців забезпечує швидкий та зручний доступ до коштів, але має певний рівень ризику, оскільки з'єднання з інтернетом може збільшити ймовірність крадіжки або злому. Гарячі гаманці включають вебгаманці, мобільні гаманці та десктопні гаманці.

Холодні гаманці – це криптовалютні гаманці, які зберігають приватні ключі офлайн, тим самим забезпечуючи високий рівень безпеки для коштів користувача. Ці гаманці підходять для довгострокового зберігання криптовалют і мінімізації ризиків злому або крадіжки. Холодні гаманці можуть бути апаратними гаманцями або паперовими гаманцями.

Мультисигнатурні гаманці – використовують спеціальні механізми авторизації, які вимагають підпису декількох осіб для здійснення транзакцій. Цей тип гаманців покликаний забезпечити додатковий рівень безпеки та контролю над коштами, оскільки він унеможливорює одноосібне управління активами. Мультисигнатурні гаманці зазвичай використовуються в організаціях, де необхідно забезпечити контроль та відповідальність різних осіб.

Кастодіальні гаманці – це гаманці, які зберігають приватні ключі користувача на стороні сервісу. Це означає, що користувачі передають контроль над своїми активами сервісу, який надає гаманець. Ці гаманці можуть забезпечувати зручність та підтримку, але також можуть мати певні ризики, пов'язані з безпекою сервісу та зломом.

Некастодіальні гаманці – це гаманці, які дозволяють користувачам повністю контролювати свої приватні ключі та кошти. Користувачі мають повну відповідальність за зберігання та управління своїми активами. Ці гаманці надають більшу безпеку та приватність, але вимагають від користувачів більшої уваги до безпеки та резервного копіювання своїх ключів.

1.5 Майнінг криптовалют

Майнінг криптовалют – це процес добування нових криптовалютних одиниць шляхом вирішення математичних задач та підтвердження транзакцій у мережі. Майнери забезпечують безпеку мережі та децентралізацію шляхом підтримки консенсусу між учасниками. У цьому підрозділі ми розглянемо основні аспекти майнінгу криптовалют та його вплив на екосистему [3].

Алгоритми консенсусу – це механізми, які дозволяють учасникам криптовалютної мережі домовлятися про стан реєстру транзакцій. Найпоширенішими алгоритмами консенсусу є PoW та PoS. PoW вимагає від майнерів виконання обчислювально складних задач, що забезпечує безпеку мережі, тоді як PoS базується на кількості криптовалют, які учасники заморожують як ставки, щоб взяти участь у процесі майнінгу.

Обладнання для майнінгу криптовалют залежить від алгоритму консенсусу та типу криптовалюти. Для PoW-алгоритмів необхідне потужне обчислювальне обладнання, таке як спеціалізовані інтегральні схеми для майнінгу (ASIC) або графічні процесори (GPU). PoS-алгоритми не вимагають такого потужного обладнання, але замість цього вимагають від користувачів заморожувати певну кількість криптовалют, щоб брати участь у майнінгу.

Майнінг криптовалют, зокрема на основі PoW-алгоритмів, споживає значну кількість електроенергії та може мати негативний вплив на довкілля через викиди вуглецевого сліду. Екологічні проблеми стали причиною пошуку альтернативних алгоритмів консенсусу, таких як PoS, які споживають менше енергії та мають менший вплив на довкілля.

Майнери отримують нагороди за свою роботу у вигляді нових криптовалютних одиниць та комісій за транзакції, які вони підтверджують. Нагороди мотивують майнерів виконувати свою роботу та підтримувати мережу, але також впливають на економічні аспекти криптовалют, такі як інфляція та кількість криптовалют, яка знаходиться в обігу.

1.6 Транзакції та криптовалютні біржі

Транзакції та криптовалютні біржі є основою екосистеми криптовалют, оскільки вони дозволяють користувачам обмінюватися активами, здійснювати покупки та інвестувати. У цьому підрозділі ми розглянемо основні аспекти транзакцій криптовалют та специфіку роботи криптовалютних бірж.

Криптовалютні транзакції мають декілька основних компонентів, таких як адреса відправника, адреса одержувача, сума переказу та цифровий підпис відправника, який забезпечує автентифікацію та цілісність транзакції. Крім того, транзакції можуть включати комісію майнерам за підтвердження транзакції та додавання її до блоку.

Криптовалютні біржі – це платформи, на яких користувачі можуть купувати, продавати та обмінювати криптовалюту. Біржі можуть пропонувати різні пари торгів, такі як крипто-крипто або крипто-фіат, та встановлювати свої правила та комісії. Є два основних типи криптовалютних бірж: централізовані та децентралізовані біржі.

Централізовані біржі (Centralized exchanges, CEX) контролюються компаніями або організаціями, які забезпечують платформу для торгівлі криптовалютами. Вони можуть пропонувати ряд послуг, таких як маржинальна торгівля та стейкінг, але також вимагають від користувачів певного рівня довіри, оскільки вони зберігають кошти користувачів на платформі. Централізовані біржі також здебільшого вимагають проходження процедури KYC (Know Your Customer) та AML (Anti-Money Laundering) перш ніж дозволити користувачам торгувати на платформі.

Децентралізовані біржі (Decentralized exchanges, DEX) використовують блокчейн-технології для створення бездозвільних торгових платформ, які дозволяють користувачам здійснювати обмін криптовалютами безпосередньо між собою, уникаючи посередників та централізованих контролюючих органів. DEX зазвичай пропонують вищий рівень безпеки та приватності, але можуть мати меншу ліквідність та менш користувацький інтерфейс порівняно з CEX.

Хоча криптовалюти все ще вважаються відносно новим видом активів, що ще розвивається, вони все більше використовуються в реальному світі для здійснення покупок, інвестицій та інших операцій. Деякі компанії та продавці вже приймають криптовалюти як форму оплати, а ряд криптовалютних бірж пропонують фіатні шлюзи, які дозволяють користувачам легко обмінювати криптовалюту на традиційні валюти.

У підрозділі 1.6 ми розглянули основні аспекти криптовалютних транзакцій та криптовалютних бірж, а також їх використання в реальному світі. Ця інформація допоможе зрозуміти, як криптовалютні активи можуть бути використані для різних фінансових операцій та як вони взаємодіють з традиційною економікою.

У зв'язку зі зростанням популярності криптовалют, уряди та регулятори по всьому світу звертають увагу на потребу в регуляції та контролі криптовалютних ринків. Регуляція може сягати від запровадження правил KYC/AML до встановлення обмежень на торгівлю та інвестиції в криптовалюту. Різні країни мають різні підходи до регуляції криптовалют, і це може впливати на розвиток та використання криптовалют у різних регіонах.

Криптовалютні інвестиції несуть високий рівень ризику через волатильність цін та невизначеність регуляції. Проте, вони також можуть пропонувати високі прибутки та унікальні можливості для диверсифікації портфеля інвестицій. Перед інвестуванням в криптовалюту необхідно вивчити ринок, розуміти ризики та знати свої інвестиційні цілі.

1.7 Висновки за розділом 1

У першому розділі дипломної кваліфікаційної роботи бакалавра було досліджено основні поняття та технології, пов'язані з криптовалютами. Розглянуто суть криптовалют, їх визначення та класифікацію. Встановлено, що криптовалюта є видом цифрової валюти, який використовує криптографічні принципи для забезпечення безпеки транзакцій та контролю над створенням нових одиниць.

Досліджено технологію блокчейн, яка лежить в основі більшості криптовалют. Розкрито принципи роботи блокчейну, його переваги. Визначено, що блокчейн є децентралізованою базою даних, яка складається з послідовності блоків, між якими утворюються криптографічні зв'язки.

Описано основні криптовалюти, зокрема Біткоїн, Ефіріум та інші. З'ясовано, що відмінність між криптовалютами полягає у їх архітектурі, способі консенсусу та призначенні.

Розглянуто криптовалютні гаманці, їх види та особливості роботи. Зрозуміло, що криптовалютні гаманці є програмними або апаратними засобами, які дозволяють зберігати, отримувати та відправляти криптовалюту.

2 ОГЛЯД ТА ВИБІР ТЕХНОЛОГІЙ

2.1. Огляд технологій розробки для платформи iOS

iOS – це операційна система, розроблена Apple для їхніх мобільних пристроїв, таких як iPhone та iPad. Розробка додатків під iOS вимагає використання спеціалізованих технологій, інструментів та мов програмування, які підтримуються Apple. У цьому підрозділі ми розглянемо ключові технології, які використовуються для розробки iOS-додатків.

Swift – це мова програмування, розроблена Apple спеціально для розробки додатків під iOS, macOS, watchOS та tvOS. Вона забезпечує швидкість, безпеку та ефективність роботи, а також сучасний синтаксис, що полегшує навчання та розробку. Swift є основною мовою програмування для розробки iOS-додатків.

Xcode – це інтегроване середовище розробки (IDE) від Apple, яке надає розробникам комплексний набір інструментів для створення, тестування та відлагодження додатків під iOS (рис. 2.1). Xcode включає в себе редактор коду, інтерфейсний конструктор, систему контролю версій, інструменти для автоматизації тестування та багато іншого [4].

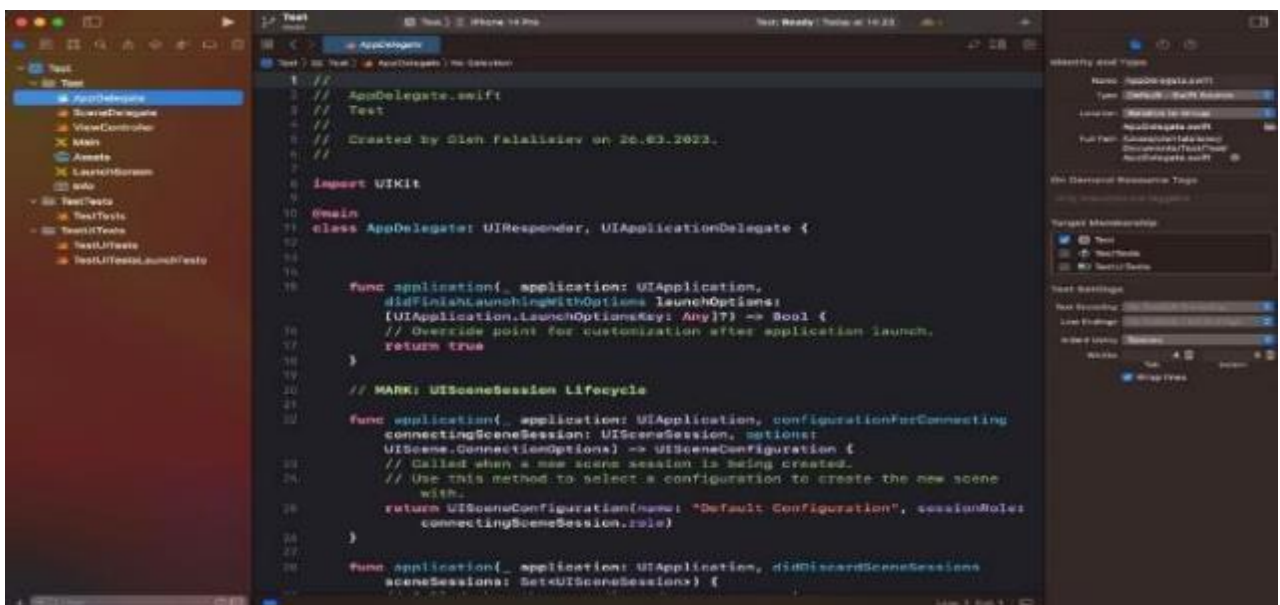


Рисунок 2.1 – Інтерфейс програми Xcode

Interface Builder – це візуальний редактор, який інтегрований в Xcode, і дозволяє розробникам створювати графічний інтерфейс додатків за допомогою перетягування елементів на екран (рис. 2.2). Розробники можуть використовувати Interface Builder для створення інтерфейсів, які працюють на різних пристроях та розмірах екранів.

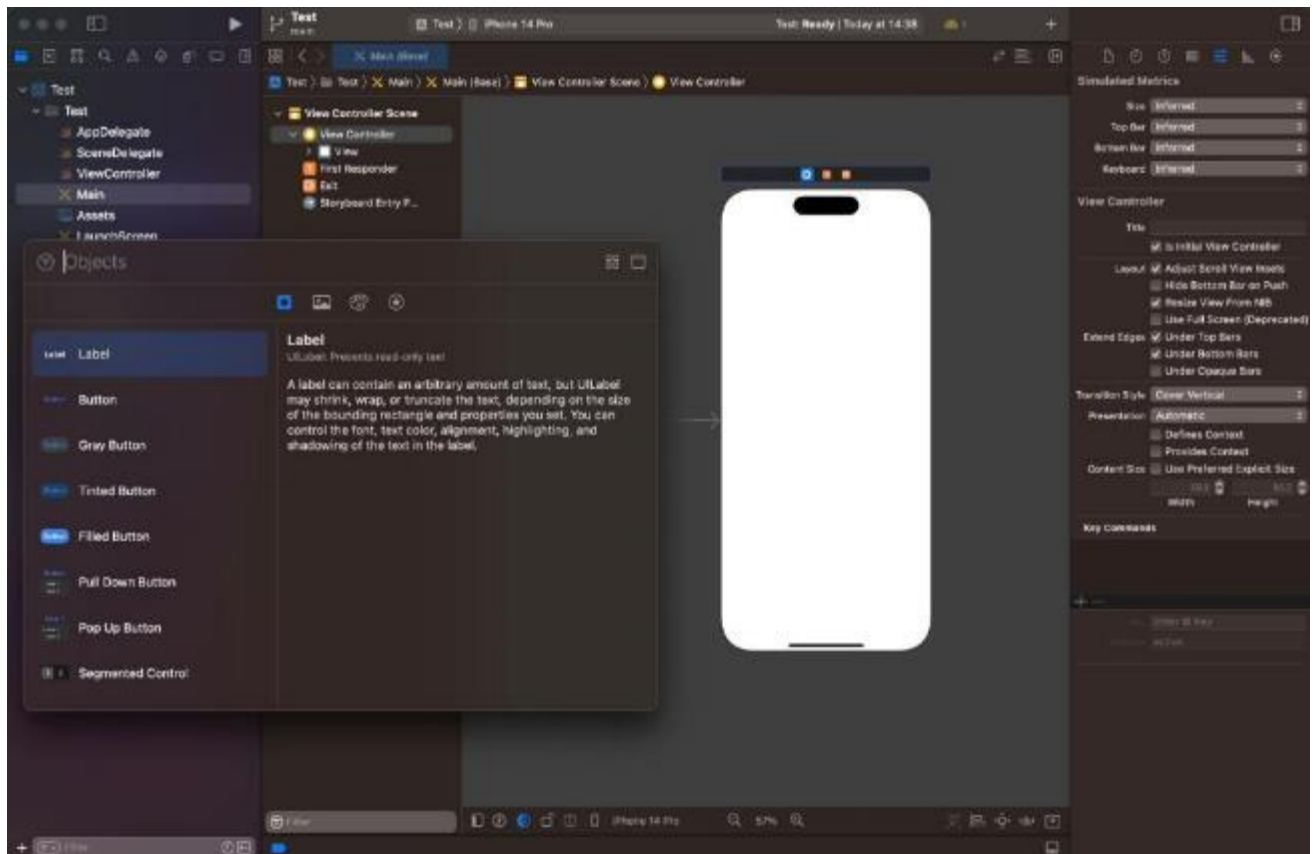


Рисунок 2.2 – Інтерфейс редактору Interface Builder

UIKit – це фреймворк, який надає набір графічних елементів, а також класи та функції для розробки графічного інтерфейсу iOS-додатків. UIKit містить стандартні віджети, такі як кнопки, перемикачі, текстові поля, таблиці та навігаційні елементи, а також інструменти для обробки подій, анімації та взаємодії з користувачем. Використання UIKit дозволяє розробникам створювати додатки зі зручним та знайомим для користувачів інтерфейсом [5].

Cocoa Touch – це набір фреймворків для розробки iOS-додатків, який включає в себе UIKit, Foundation, Core Data, Core Graphics та інші. Cocoa Touch

надає розробникам доступ до низькорівневих функцій операційної системи, таких як мережеві послуги, сенсорні події та інтеграція з сервісами Apple, такими як iCloud та Apple Pay.

2.2 Мови програмування: Swift та Objective-C

При розробці додатків для iOS розробники зазвичай використовують одну з двох основних мов програмування: Swift або Objective-C. У цьому підрозділі ми розглянемо відмінності та спільні риси цих мов.

Swift – це мова програмування, як було зазначено вище, розроблена Apple спеціально для створення додатків для iOS, macOS, watchOS та tvOS. Swift має декілька ключових переваг:

- сучасний синтаксис – Swift використовує лаконічний та зрозумілий синтаксис, що полегшує навчання та розробку;
- безпека – Swift розроблено з акцентом на безпеку, що допомагає запобігти поширеним помилкам програмування;
- швидкість – Swift оптимізований для швидкості виконання коду, що дозволяє створювати високопродуктивні додатки.

Сумісність з Objective-C: Swift може використовуватися разом з Objective-C, що дозволяє інтегрувати новий код Swift у існуючі проекти.

Objective-C – це старіша мова програмування, яка була основною мовою розробки для Apple пристроїв до введення Swift. Objective-C має наступні особливості:

- динамічна типізація: Objective-C дозволяє більшу гнучкість у використанні типів даних, але це може призвести до проблем з безпекою та непередбачуваним поведінкою коду;
- сумісність з C: Objective-C є надбудовою над мовою програмування C, що дозволяє використовувати більшість коду C безпосередньо в проектах Objective-C;

– відмінний синтаксис: У порівнянні з Swift, Objective-C використовує менш інтуїтивний синтаксис, що може ускладнити навчання та розробку для нових розробників [6]-[7].

Порівняльна таблиця мов програмування Swift та Objective-C наведена в таблиці (табл. 2.1).

Таблиця 2.1 – Порівняння мов програмування

Мова	Swift	Objective-C
Впровадження	Новіша мова з більш сучасними можливостями	Старіша мова, що потребує поновлення
Спадщина	Наслідує C та Objective-C, але з більш сучасним підходом	Наслідує тільки C та Smalltalk, менш сучасна спадщина
Читабельність коду	Дуже зрозумілий та легкий для читання	Менш зрозумілий та складний для читання
Розширюваність	Протоколи та розширення дозволяють легко додавати функціональність	Категорії та делегування менш гнучкі
Швидкодія	До 2,6 разів швидший за Objective-C	Повільніший за Swift
Підтримка бібліотек	Є	Є
Розробка UI	Interface Builder забезпечує легкість розробки інтерфейсу	Інтерфейс розробляється через взаємодію з кодом

На сьогоднішній день більшість нових проєктів для iOS розробляються на Swift через його сучасний синтаксис, безпеку та швидкість. Проте, Objective-C може бути використаний у випадках, коли розробники працюють над старішими проєктами або мають значний досвід роботи з Objective-C.

Завдяки сумісності Swift та Objective-C розробники можуть поєднувати код обох мов у одному проєкті. Це корисно при розробці нових функцій для існуючих проєктів на Objective-C або при використанні старіших бібліотек та фреймворків. Для інтеграції Swift та Objective-C коду в одному проєкті розробники повинні створити спеціальні містки (bridging headers) та врахувати відмінності між двома мовами програмування.

2.3 Розробка застосунків з використанням UIKit та SwiftUI

UIKit та SwiftUI – це два основні фреймворки для створення графічного інтерфейсу користувача (GUI) в iOS-додатках. Обидва фреймворки мають свої переваги та недоліки, тому розробники повинні обирати той, що найкраще підходить для конкретного проєкту. У цьому підрозділі ми розглянемо особливості кожного з цих фреймворків та їх використання в розробці застосунків.

UIKit – це встановлений та перевірений фреймворк, який використовується для створення графічного інтерфейсу користувача в iOS-додатках. Він пропонує широкий спектр готових до використання компонентів інтерфейсу, таких як кнопки, текстові поля, списки, графіки та інше [5].

Особливості UIKit:

- відносно високий поріг входження для нових розробників через його складний синтаксис та багаторівневу архітектуру;
- використання коду та/або Interface Builder для створення інтерфейсів користувача;
- широкий набір компонентів інтерфейсу та API, які забезпечують гнучкість та можливість створення складних інтерфейсів.
- широка підтримка різних версій iOS.

SwiftUI – це новий декларативний фреймворк для створення інтерфейсів користувача, який був представлений Apple в 2019 році. Він дозволяє

розробникам створювати інтерфейси з меншим обсягом коду та з використанням сучасного синтаксису Swift.

Нижче перелічені особливості SwiftUI:

- низький поріг входження для нових розробників завдяки декларативному синтаксису та спрощеній архітектурі;
- використання декларативного синтаксису Swift для створення інтерфейсів користувача, що забезпечує більшу читабельність коду та легкість супроводу;
- гаряче перезавантаження (Hot Reload) дозволяє розробникам побачити зміни в інтерфейсі миттєво після зміни коду, що сприяє швидкому ітеруванню та поліпшенню продуктивності;
- широка підтримка кросплатформеності, що дозволяє використовувати один і той же код для створення інтерфейсів користувача на iOS, macOS, watchOS та tvOS;
- обмежена підтримка старих версій iOS (SwiftUI доступний починаючи з iOS 13).

2.4 Використання CocoaPods та Swift Package Manager для управління залежностями

У процесі розробки додатків для iOS часто виникає необхідність використовувати зовнішні бібліотеки та фреймворки, щоб не винаходити велосипед заново та забезпечити швидку реалізацію функціональності. Управління залежностями є важливою частиною розробки, і CocoaPods та Swift Package Manager – два популярні інструменти для цієї задачі. У цьому підрозділі ми розглянемо їх особливості та використання.

CocoaPods – це менеджер залежностей для проєктів Objective-C та Swift, який дозволяє розробникам включати зовнішні бібліотеки та фреймворки у свої проєкти простим способом.

Особливості CocoaPods:

- велика кількість доступних бібліотек і фреймворків;
- інтеграція з Xcode за допомогою створення робочого простору (workspace) та управління схемами;
- використання файлу Podfile для опису залежностей проєкту;
- автоматичне завантаження та інтеграція залежностей в проєкт.

Swift Package Manager (SPM) – це менеджер залежностей, розроблений Apple спеціально для мови Swift. Він вбудований в Swift та Xcode і дозволяє розробникам включати зовнішні бібліотеки без необхідності встановлення додаткових інструментів.

Особливості Swift Package Manager:

- інтеграція з Xcode та Swift, що спрощує процес розробки та використання залежностей;
- використання файлу Package.swift для опису залежностей проєкту;
- підтримка кросплатформеності (iOS, macOS, watchOS, tvOS, Linux);
- можливість створювати свої власні пакети та використовувати їх у різних проєктах;
- більша сумісність з майбутніми версіями Swift та Xcode через вбудований характер SPM.

2.5 Бази даних

Для розробки застосунків для iOS існує багато баз даних, які можуть бути використані для зберігання та обробки даних. У цьому розділі ми порівняємо три з них: CoreData, Realm та Firestore, та обґрунтуємо вибір Firestore для розробки застосунку.

CoreData – це фреймворк для зберігання та обробки даних в iOS. CoreData зберігає дані в SQLite-підсистемі, що дозволяє зберігати великі обсяги даних та забезпечує швидкий доступ до них. CoreData також надає можливість

працювати з даними в офлайн-режимі, що є важливим для додатків, які мають обмежені можливості зв'язку з мережею. Однак, використання CoreData може бути складним для розробників, які не мають достатнього досвіду з цією технологією.

Realm – це база даних для мобільних платформ, яка дозволяє зберігати та обробляти дані в реальному часі. Realm надає швидкий доступ до даних, що є важливим для додатків, які потребують великої кількості операцій з даними. Realm також підтримує офлайн-режим та синхронізацію даних між пристроями. Однак, Realm має обмежену підтримку запитів та відсутність віддаленого зберігання даних, що може бути недостатньо для додатків, які працюють з великим обсягом даних.

Firestore – це база даних, яка розроблена для зберігання та обробки даних в режимі реального часу. Firestore надає швидкий доступ до даних та підтримує режим офлайн-роботи, що дозволяє додаткам працювати в безмережевому середовищі. Firestore також підтримує синхронізацію даних між різними пристроями та забезпечує високу масштабованість для додатків з великою кількістю користувачів та даних. Firestore також надає можливість створювати складні запити та фільтрувати дані за різними параметрами, що дозволяє ефективно працювати з даними [8]-[9].

Одним з головних переваг Firestore є його інтеграція з іншими сервісами Google, такими як Firebase Analytics, Firebase Authentication та інші. Це забезпечує готову інфраструктуру для створення додатків з різними функціональними можливостями та може значно спростити розробку та підтримку застосунку.

Також важливою перевагою Firestore є його масштабованість. Firestore може масштабуватись до великої кількості користувачів та даних, що дозволяє додаткам зростати та розвиватись з часом без необхідності переносу на інші бази даних.

Узагальнюючи, для розробки застосунків iOS, які потребують зберігання та обробки даних, Firestore є хорошим вибором через свою швидкість,

масштабованість, підтримку реального часу та готову інфраструктуру Firebase. Хоча CoreData та Realm також можуть бути відмінними виборами для деяких додатків, Firestore забезпечує більше функціональних можливостей та підходить для більш широкого спектру використання.

Порівняння оглянутих вище баз даних узагальнено в таблиці (табл. 2.2).

Таблиця 2.2 – Порівняння баз даних

База даних	CoreData	Realm	Firestore
Тип бази даних	Локальна	Локальна та мережева	Хмарна
Хмарна синхронізація	-	-	+
Підтримка мережевих з'єднань	-	+	+
Підтримка спільного доступу до даних	-	+	+
Масштабованість	Середня	Висока	Висока
Швидкість доступу до даних	Висока	Висока	Висока

2.6 Висновки за розділом 2

Другий розділ дипломної кваліфікаційної роботи бакалавра було присвячено огляду технологій, що використовуються для розробки платформи iOS, які включають мови програмування, фреймворки та бази даних.

Досліджено мови програмування Swift та Objective-C. Встановлено, що Swift є більш сучасною мовою програмування, яка має вищу продуктивність, забезпечує більшу безпеку програмного коду та має багатий функціонал.

Objective-C є старішою мовою, проте все ще використовується в деяких проектах, але зазвичай як підтримувана мова.

Розглянуто фреймворки розробки, зокрема UIKit та SwiftUI. Встановлено, що UIKit є класичним фреймворком, який має великий досвід та забезпечує більше можливостей для налаштування елементів інтерфейсу користувача. SwiftUI є новітнім фреймворком, який забезпечує більш простий та елегантний спосіб створення інтерфейсу користувача та працює на основі декларативного програмування [10].

Досліджено систему управління залежностями, такі як CocoaPods та Swift Package Manager. Встановлено, що CocoaPods є класичною та надійною системою управління залежностями, яка має велику спільноту та підтримується на різних платформах. Swift Package Manager є новітнім та стандартним засобом управління залежностями, який надає зручний інтерфейс та вбудовану підтримку для пакетів.

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Розробка інтерфейсу застосунку

Після створення нового проекту в Xcode, ми бачимо декілька файлів, один з яких – Main. Переходимо в нього та, за допомогою навігаційного меню, додаємо нове вікно. Перед нами з'явилося пусте вікно (рис. 3.1).

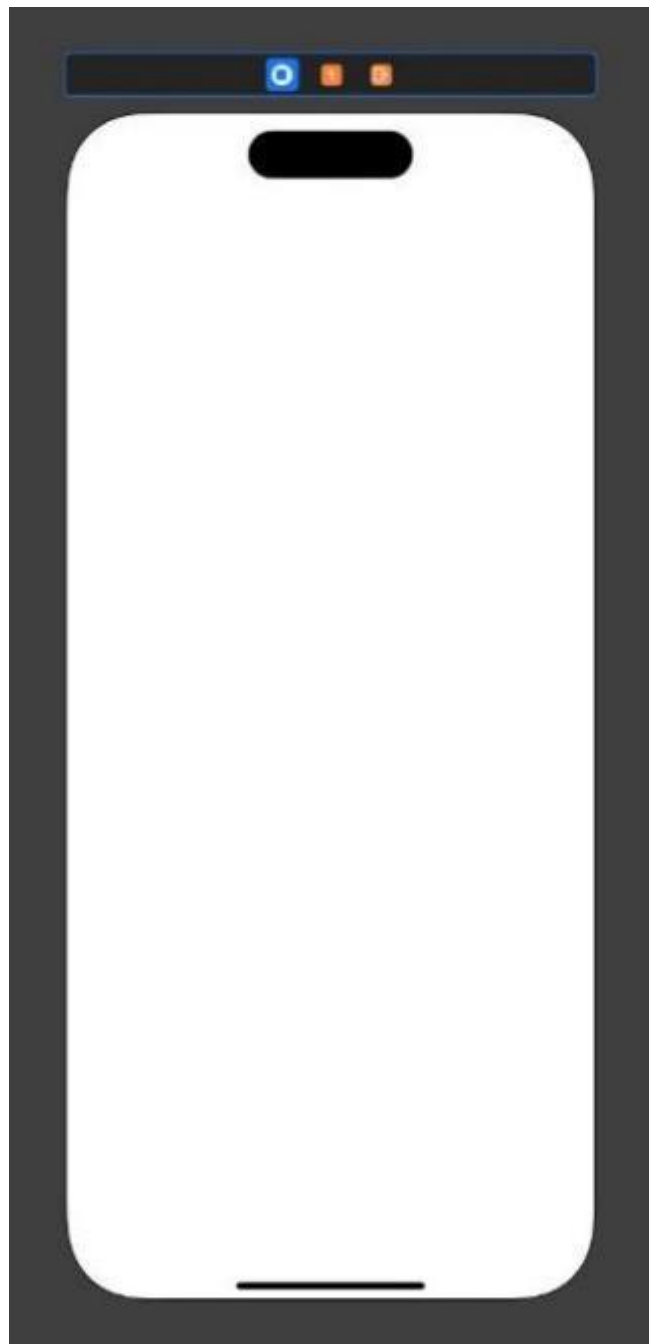


Рисунок 3.1 – Пусте вікно

За допомогою навігаційного меню ми можемо додати різноманітні елементи інтерфейсу, такі як текст, рисунки, таблиці, кнопки та інші (рис. 3.2). При цьому необхідно відкрити навігаційне меню, вибрати потрібний елемент та перетягти його у вікно.

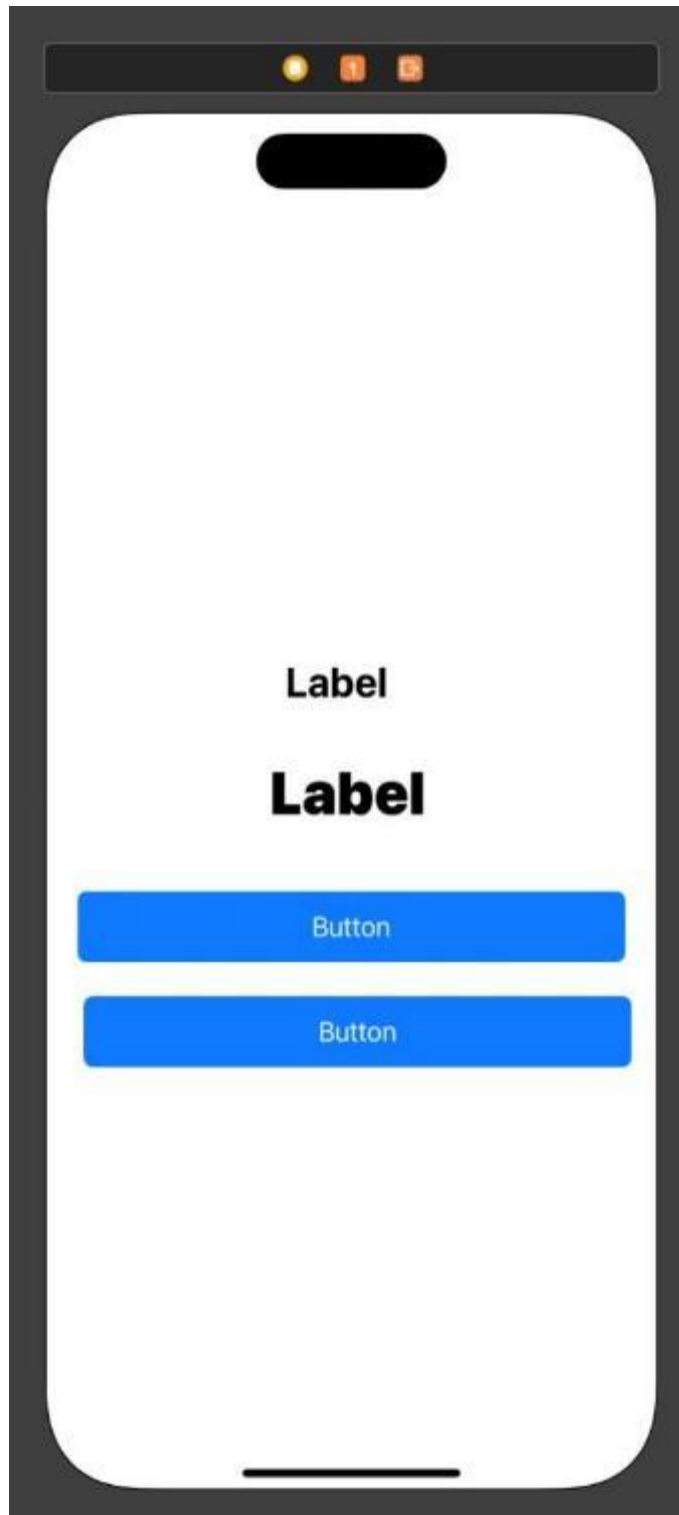


Рисунок 3.2 – Вікно з елементами інтерфейсу

Тепер ми повинні встановити позиції для елементів вікна (рис. 3.3).



Рисунок 3.3 – Вікно з встановленими позиціями елементів

Основні способи встановлення позицій елементів описані нижче:

– auto layout дозволяє встановлювати позиції елементів інтерфейсу на основі обмежень (constraints), що дозволяє забезпечити правильне розміщення елементів незалежно від розміру екрану. Щоб встановити обмеження,

необхідно вибрати елемент інтерфейсу та використовувати меню Editor – Resolve Auto Layout Issues – Add Missing Constraints;

– size inspector дозволяє встановлювати розмір та положення елементів інтерфейсу. Щоб відкрити Size Inspector, необхідно вибрати елемент інтерфейсу та вибрати меню View – Utilities – Show Size Inspector. У Size Inspector можна змінювати розмір, положення, відступи та інші параметри елемента;

– identity inspector дозволяє встановлювати ідентифікатори для елементів інтерфейсу, які можна використовувати у програмному коді. Щоб відкрити Identity Inspector, необхідно вибрати елемент інтерфейсу та вибрати меню View – Utilities – Show Identity Inspector. У Identity Inspector можна встановити ідентифікатор елемента та інші параметри;

– додатковим способом встановлення позицій елементів інтерфейсу є використання програматичного коду. За допомогою програмування можна встановлювати положення елементів, задавати їх розміри, відступи та інші параметри безпосередньо у коді програми. Цей підхід надає більшу гнучкість і можливості для динамічного розташування елементів на основі змінних умов, а також для виконання складніших маніпуляцій з інтерфейсом. Наприклад, використовуючи програматичний код, можна адаптувати розміщення елементів до різних орієнтацій екрану, обробляти різні розміри екрану та автоматично розташовувати елементи залежно від поточних умов.

Далі за допомогою Attribute Inspector ми можемо змінити розмір, колір фону, заголовок та інші властивості елементів. Щоб змінити властивості елемента, необхідно відкрити Attribute Inspector і вибрати елемент, властивості якого потрібно змінити. Потім можна змінити розмір, колір фону, шрифт, текст та інші параметри цього елемента [11]-[12].

Таким чином буде сформовано зовнішній вигляд вікна застосунку (рис. 3.4).



Рисунок 3.4 – Готове вікно

Використовуючи описані вище методи, створимо всі інші вікна застосунку.

3.2 Реєстрація та авторизація користувача

Реєстрація та авторизація користувача реалізована за допомогою імплементування фреймворка firebase authentication. Він дозволяє створювати нові облікові записи користувачів за допомогою електронної пошти та пароля та підтримує вхід користувачів за допомогою раніше створених облікових записів, що допомагає забезпечити безпеку застосунку.

Функція реєстрації має наступний вигляд:

```
func createAccount() {
    if let email = emailTextField.text, let password =
passwordTextField.text {
        Auth.auth().createUser(withEmail: email, password:
password) { authResult, error in
            guard let user = authResult?.user, error == nil
else {
                print("Error creating user:
(error!.localizedDescription)")
                return
            }
        }
    }
}
```

Такий вигляд має функція авторизації:

```
@IBAction func signInButton(_ sender: UIButton) {
    if let email = emailTextField.text, let password =
passwordTextField.text {
        Auth.auth().signIn(withEmail: email, password:
password) { authResult, error in
            if let e = error {
```

```

        print(e.localizedDescription)
    }
}
}
}

```

3.3 База даних

Робота з базою даних застосунку виконується за допомогою бібліотеки Firebase Firestore. Для того, щоб додати цю бібліотеку, нам потрібно відкрити проєкт, вибрати "File" > "Add Packages" в верхньому меню, або ж натиснути на кнопку "+" в розділі "Swift Packages" у бічній панелі нашого проєкту. У вікні, що з'явиться, ввести URL репозиторію Firebase: <https://github.com/firebase/firebase-ios-sdk>. Натиснути "Add Package" та вибрати потрібні продукти для проєкту. Для Firestore нам потрібно вибрати продукт "FirebaseFirestore".

Firestore – це гнучка та масштабована база даних в реальному часі від Firebase, яка дозволяє легко створювати, читати, оновлювати та видаляти дані. Firestore використовує модель даних, засновану на колекціях, документах та полях. Розглянемо кожен з цих елементів.

Колекції – це групи документів, які можуть містити будь-яку кількість документів. Вони слугують контейнерами для документів та допомагають організувати дані. Колекції можуть бути вкладеними, що дозволяє створювати ієрархічну структуру даних.

Документи – це об'єкти, які містять пари ключ-значення, відомі як поля. Вони представляють собою основні одиниці даних в Firestore. Документи зберігаються в колекціях і можуть мати унікальний ідентифікатор, що дозволяє легко отримувати доступ до них. Документи мають обмеження у 1 мегабайт розміру даних.

Поля – це пари ключ-значення, які містяться в документах. Кожне поле має унікальний ключ (ім'я поля) та значення, яке може бути різним типом даних, таких як рядки, числа, булеві значення, масиви, об'єкти, дати та географічні точки. Значення полів можуть бути простими типами даних або складними структурами, такими як вкладені об'єкти або масиви.

Організація даних в Firestore передбачає створення колекцій для групування документів та документів, що містять поля з відповідними даними. Це дає можливість створювати гнучкі та ієрархічні структури даних, які можуть легко масштабуватися та адаптуватися до змін вимог до застосунку.

Ось деякі приклади роботи з колекціями, документами та полями у Firestore.

Щоб створити нову колекцію з документом, можна використати наступний код:

```
let db = Firestore.firestore()
db.collection("users").document("user_id").setData([
  "name": "John Doe",
  "age": 25,
  "email": "john.doe@example.com"
]) { error in
  if let error = error {
    print("Error adding document: \(error)")
  } else {
    print("Document added with ID: user_id")
  }
}
```

Щоб отримати доступ до документа та його полів, можна використати наступний код:

```
let docRef = db.collection("users").document("user_id")
```

```

docRef.getDocument { (document, error) in
    if let document = document, document.exists {
        let dataDescription =
document.data().map(String.init(describing:)) ?? "nil"
        print("Document data: \(dataDescription)")
    } else {
        print("Document does not exist")
    }
}

```

Щоб оновити поле в документі, можна використати наступний код:

```

db.collection("users").document("user_id").updateData([
    "age": 26
]) { error in
    if let error = error {
        print("Error updating document: \(error)")
    } else {
        print("Document successfully updated")
    }
}

```

Щоб видалити документ, можна використати наступний код:

```

db.collection("users").document("user_id").delete() { error in
    if let error = error {
        print("Error removing document: \(error)")
    } else {
        print("Document successfully removed!")
    }
}

```

Робота з Firestore забезпечує можливість створення складних структур даних, які легко масштабуються та адаптуються до потреб вашого застосунку. Крім того, Firestore надає ряд можливостей для фільтрації, сортування та слухання змін даних в реальному часі.

Модель бази даних застосунку зображена на рисунку (рис. 3.5).

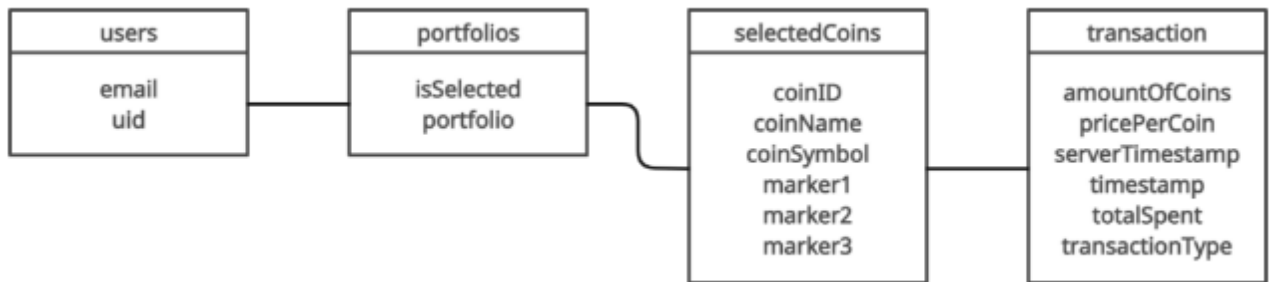


Рисунок 3.5 – Модель бази даних застосунку

3.4 Робота з API

Для відслідковування цін та формування графіків у застосунку використовується стороннє API (Application Programming Interface) CoinGecko.

Робота з API в Swift може здійснюватися за допомогою різних методів та бібліотек. Один з найпопулярніших способів роботи з API – використання URLSession, що є частиною Foundation фреймворка. URLSession надає можливість виконувати HTTP-запити для взаємодії з вебсервісами [13]-[14].

Ось приклад використання URLSession для виконання GET-запиту до REST API:

```

import Foundation

let url = URL(string: "https://api.example.com/data")!
let task = URLSession.shared.dataTask(with: url) { (data,
response, error) in

```

```

guard let data = data, error == nil else {
    print("Error: \(error?.localizedDescription ?? "Unknown
error")")
    return
}

do {
    let json = try JSONSerialization.jsonObject(with: data,
options: [])
    print("JSON: \(json)")
} catch {
    print("Error parsing JSON: \(error.localizedDescription)")
}

task.resume()

```

Цей код створює GET-запит до URL-адреси "https://api.example.com/data" та обробляє отримані дані у форматі JSON.

Щоб розпарсити JSON-дані, отримані від API, можна використовувати JSONDecoder з Codable протоколом.

Припустимо, ми отримали наступний JSON-об'єкт від API:

```

{
    "name": "John",
    "age": 30
}

```

Спочатку створюємо структуру, яка відповідає формату JSON-даних і реалізує протокол Codable:

```

struct Person: Codable {

```

```

let name: String
let age: Int
}

```

Тепер використовуємо `JSONDecoder` для декодування JSON-даних в екземпляр структури `Person`:

```

do {
    let decoder = JSONDecoder()
    let person = try decoder.decode(Person.self, from: data)
    print("Name: \(person.name), Age: \(person.age)")
} catch {
    print("Error decoding JSON: \(error.localizedDescription)")
}

```

Використання `JSONDecoder` з `Codable` протоколом є більш сучасним та безпечним підходом до роботи з JSON-даними у Swift, оскільки він обробляє типи даних статично і дозволяє нам працювати з більш структурованими моделями даних.

3.5 Імплементування бібліотеки `Charts` для роботи з графіками

Для створення графіків (рис. 3.6) у застосунку використовуємо популярну бібліотеку `Charts`.



Рисунок 3.6 – Лінійний графік

Для використання бібліотеки Charts спочатку потрібно встановити її в проєкт. Це можна зробити за допомогою Swift Package Manager.

Після встановлення бібліотеки Charts, імпортуємо її в файлі, де ми хочемо створити графік:

```
import Charts
```

Тепер ми можемо створювати різні типи графіків, налаштовувати їх візуальні характеристики, додавати значення, мітки осей, легенди, взаємодію з користувачем та інші функції.

Ось приклад створення простого лінійного графіка:

```
let lineChartView = LineChartView(frame: CGRect(x: 0, y: 0, width:
300, height: 200))
let entries = [ChartDataEntry(x: 0, y: 1), ChartDataEntry(x: 1, y:
2), ChartDataEntry(x: 2, y: 4), ChartDataEntry(x: 3, y: 6)]
let dataSet = LineChartDataSet(entries: entries, label: "Дані")
dataSet.colors = [UIColor.red]
dataSet.circleColors = [UIColor.red]
let lineChartData = LineChartData(dataSet: dataSet)
lineChartView.data = lineChartData
lineChartView.xAxis.labelPosition = .bottom
lineChartView.animate(xAxisDuration: 2.0, yAxisDuration: 2.0)
view.addSubview(lineChartView)
```

Цей приклад створює простий лінійний графік з чотирма точками. Ми можемо налаштувати його зовнішній вигляд, анімацію та інші характеристики за допомогою властивостей `LineChartView` та `LineChartDataSet`.

Крім того, бібліотека Charts надає можливість взаємодії з графіком, такі як масштабування, прокрутка, виділення точок та отримання подій взаємодії.

Для активації цих функцій, налаштуємо властивості `'LineChartView'`, наприклад:

```
lineChartView.scaleYEnabled = true  
lineChartView.dragEnabled = true  
lineChartView.delegate = self
```

Бібліотека Charts надає потужний інструментарій для візуалізації даних у застосунках iOS, tvOS та macOS додатках. Вивченням документації та прикладів коду, ви зможете створювати складні та гнучкі графіки, що відповідають потребам вашого застосунку.

3.6 Висновки за розділом 3

У третьому розділі було проведено розробку застосунку для платформи iOS, що включає створення інтерфейсу, реєстрацію та авторизацію користувача, роботу з базою даних та API, а також імплементацію бібліотеки Charts для роботи з графіками.

Досліджено створення інтерфейсу застосунку за допомогою Interface Builder Xcode, що дозволяє зручно та швидко створювати та налаштовувати елементи інтерфейсу користувача.

Реєстрація та авторизація користувача були реалізовані за допомогою Firebase Auth, що забезпечує швидку та надійну авторизацію користувачів за допомогою email та пароля.

Для роботи з базою даних було використано Firebase Firestore, що дозволяє зручно та швидко зберігати, оновлювати та видаляти дані з бази даних.

Для роботи з API було використано URLSession, який надає можливість здійснювати запити до серверів та отримувати відповіді у форматі JSON.

Для імплементації графіків була використана бібліотека Charts, що надає можливість швидко та зручно створювати та налаштовувати різноманітні графіки для відображення даних у застосунку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Призначення застосунку

Застосунок призначений для управління криптовалюtnим портфелем користувача. Він надає користувачам зручний інтерфейс для стеження за своїми активами, а також дозволяє виконувати ряд операцій з криптовалютами. Основні функції та можливості застосунку включають:

- відстеження криптовалюtnого портфеля;
- підтримка різних криптовалют;
- транзакції;
- відстеження графіків.

4.2 Умови запуску застосунку

Для запуску застосунку, пристрій користувача має бути сумісний з версією операційної системи iOS 13 та вище. Список пристроїв, що підтримують iOS 13, включає iPhone 6s та новіші моделі iPhone.

У разі, якщо пристрій користувача не підтримує iOS 13, він повинен оновити його.

4.3 Тестування застосунку

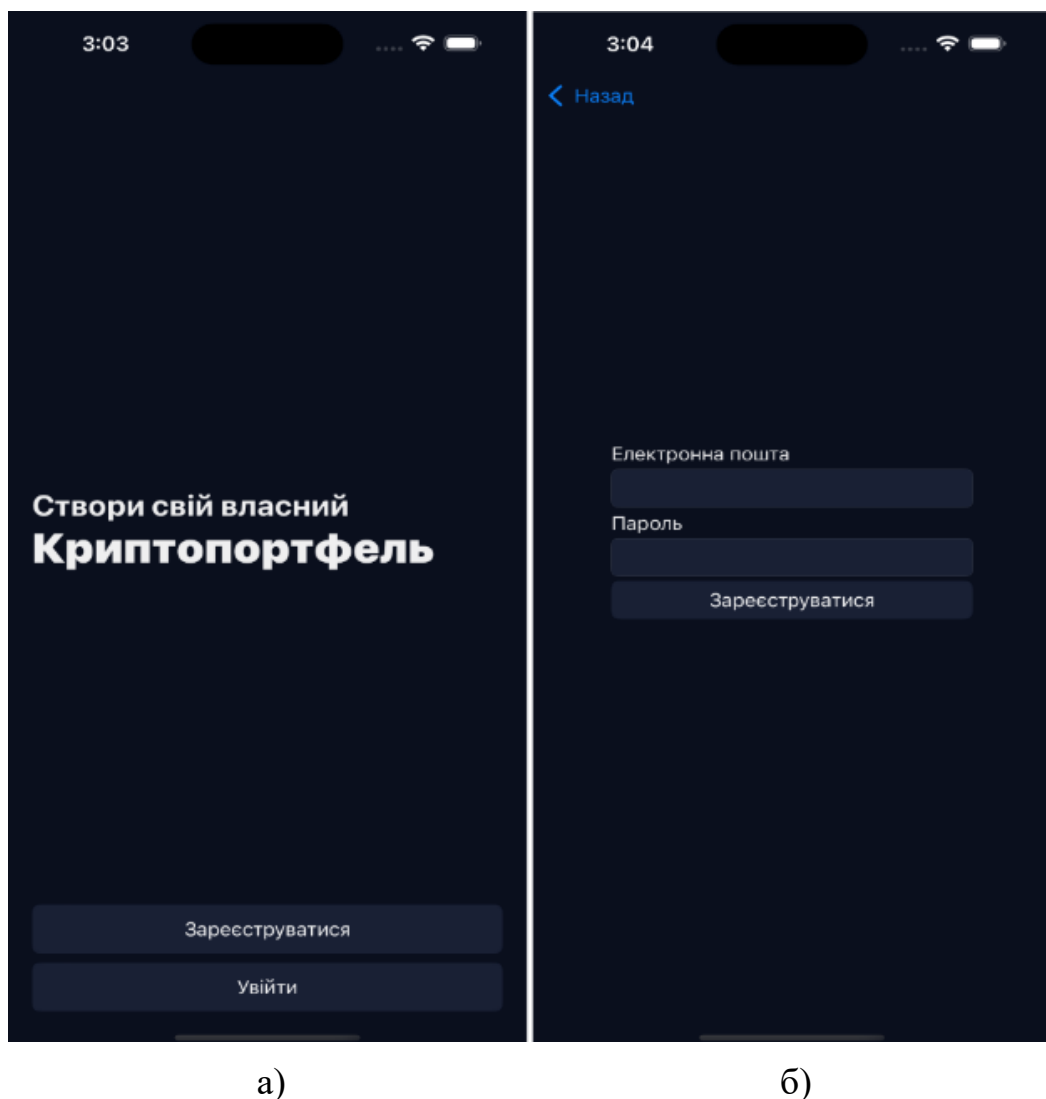
В процесі мануального тестування були виконали наступні дії:

- перевірка відповідності інтерфейсу користувача макетам та дизайну;
- тестування навігації та взаємодії з елементами інтерфейсу;
- перевірка валідації введених даних користувачем;
- тестування реєстрації та авторизації користувача;
- перевірка роботи з базою даних та збереження даних користувача;
- перевірка роботи на різних пристроях та розмірах екранів

У ході тестування були виявлені та усунені всі недоліки застосунку. Після усунення всіх недоліків та проходження повторного тестування, застосунок працює добре.

4.4 Експлуатація застосунку

Під час першого запуску застосунку, користувач потрапляє у привітальне вікно, де може авторизуватися чи пройти реєстрацію (рис. 4.1).



а – привітальне вікно; б – вікно реєстрації.

Рисунок 4.1 – Привітальне вікно

Після успішної реєстрації, користувач потрапляє в основне вікно застосунку (рис. 4.2), де він може бачити всі свої додані криптовалюти, їхні баланси та ціну.

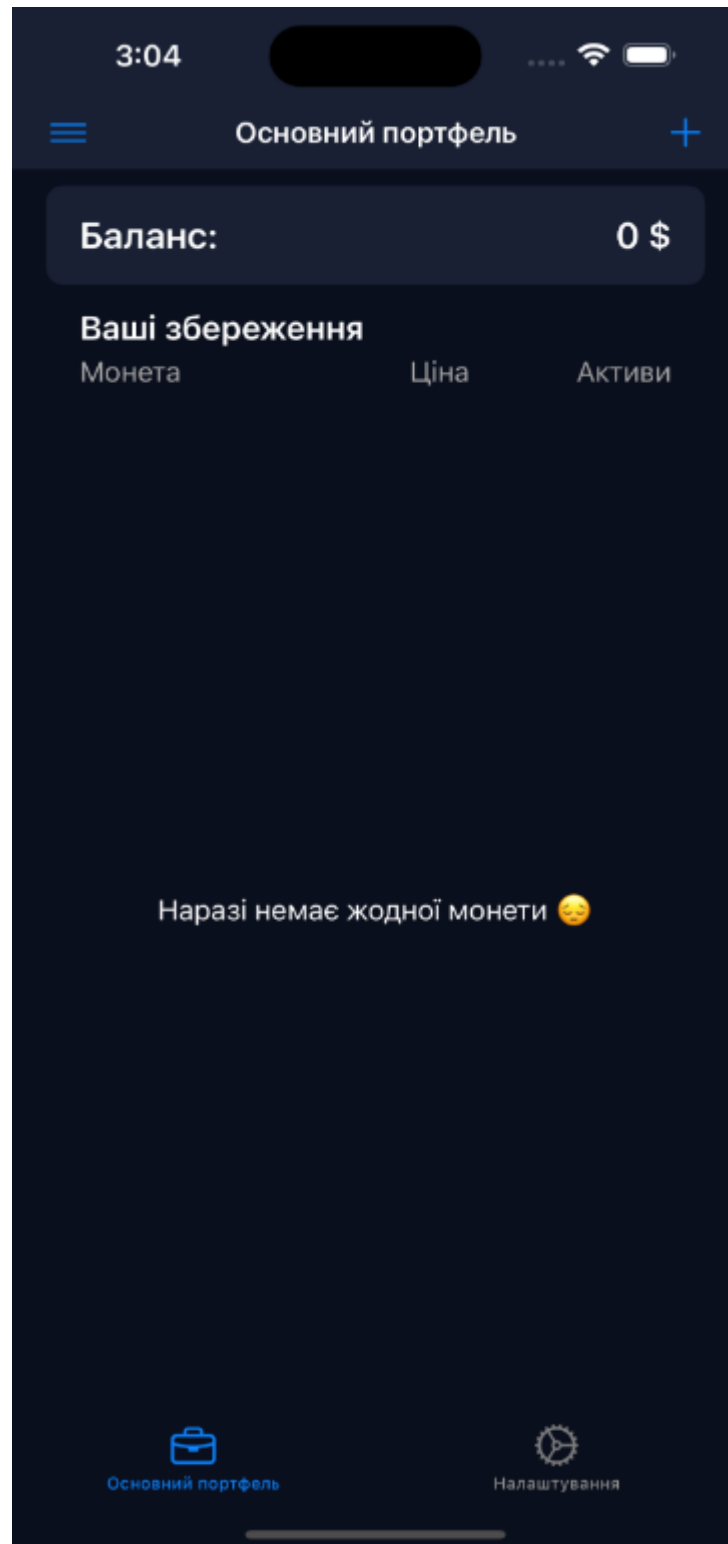
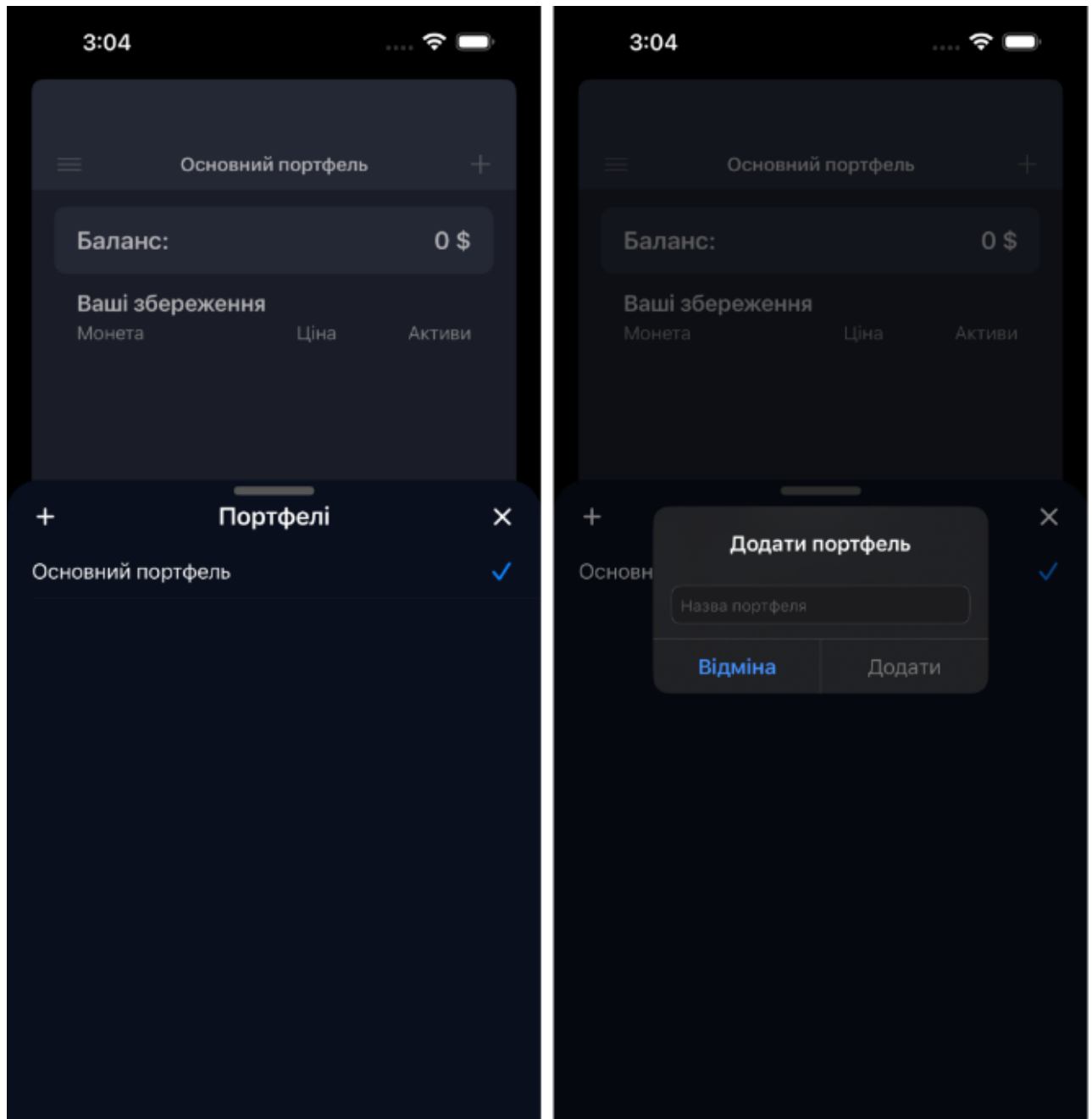


Рисунок 4.2 – Основне вікно застосунку

Натиснувши кнопку у верхньому лівому кутку екрану, користувач потрапляє на екран редагування своїх портфелів (рис. 4.3). Тут він може створювати нові портфелі або ж видаляти старі.



а)

б)

а – вікно портфелів; б – інтерфейс додавання нового портфеля.

Рисунок 4.3 – Вікно редагування портфелів

Якщо в основному вікні натиснути кнопку, яка знаходиться у правому верхньому кутку, то користувач потрапляє на екран, у якому він може додавати нові криптовалюти до свого портфеля (рис. 4.4). Зверху екрана додана зручна кнопка для прискорення пошуку.

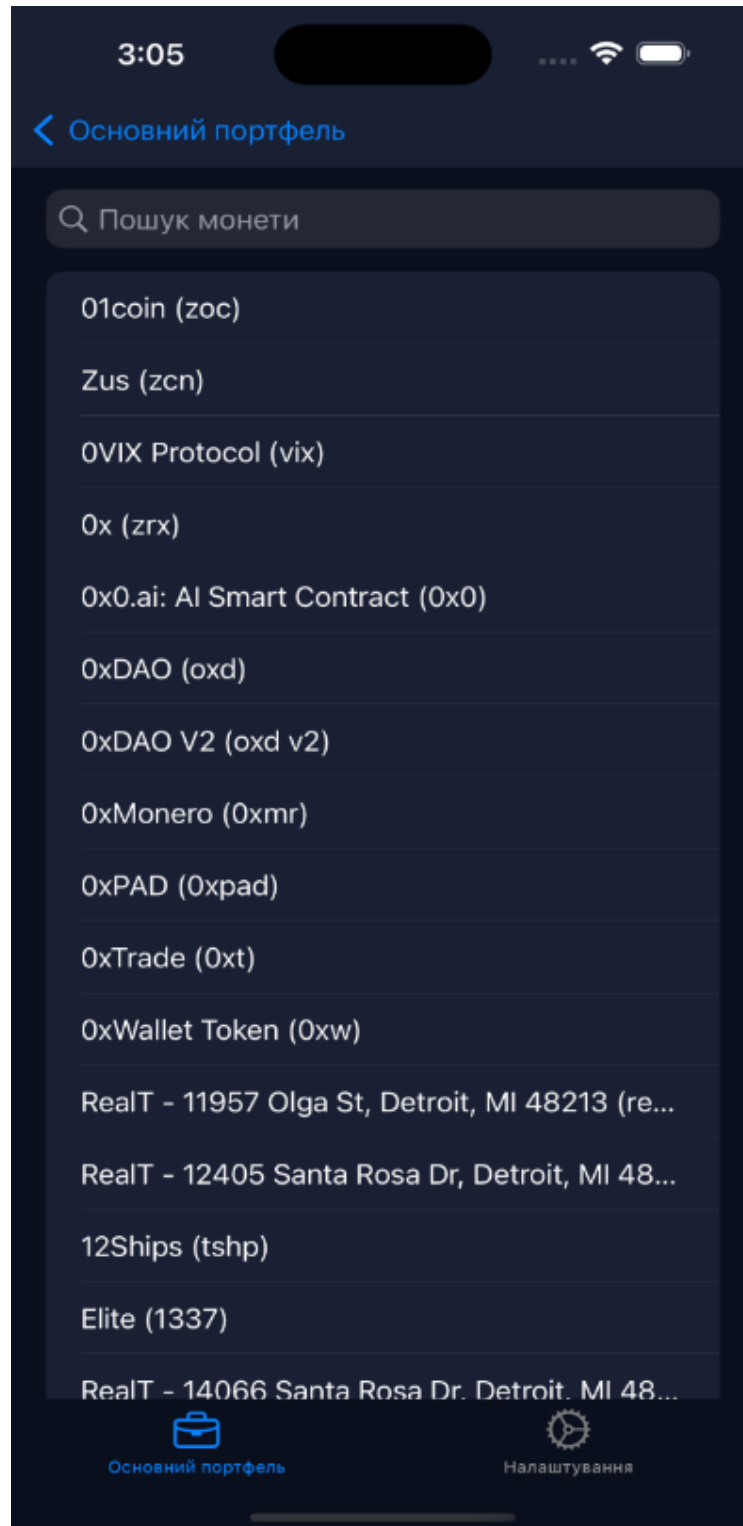


Рисунок 4.4 – Вікно пошуку криптовалют

Після додавання монети, вона з'явиться в основному екрані (рис. 4.5), та користувач зможе дивитися свій баланс і ціни на додані валюти.

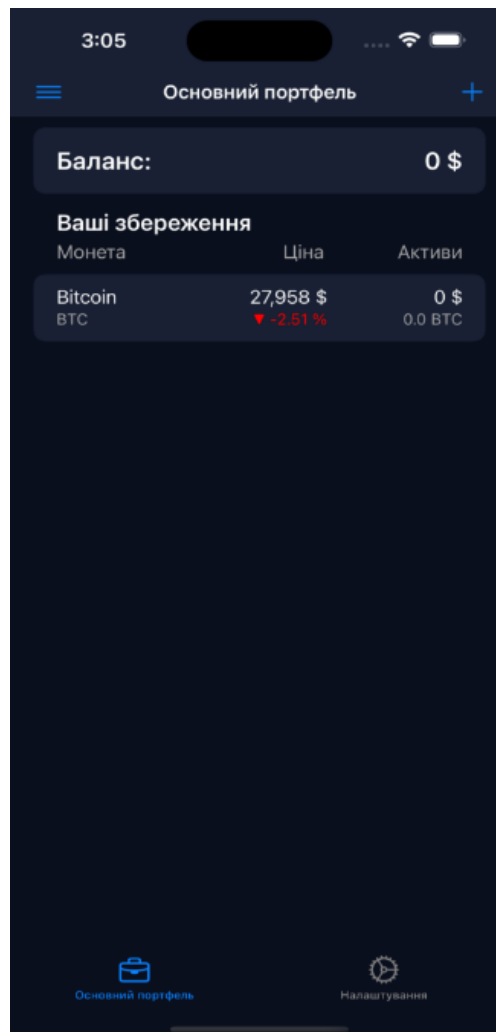


Рисунок 4.5 – Основне вікно з доданими криптовалютами

Натиснувши на додану криптовалюту, користувач потрапляє на екран транзакцій, де може переглядати історію операцій з монетою та додавати маркери ціни, натиснувши на три крапки у верхньому правому кутку (рис. 4.6).

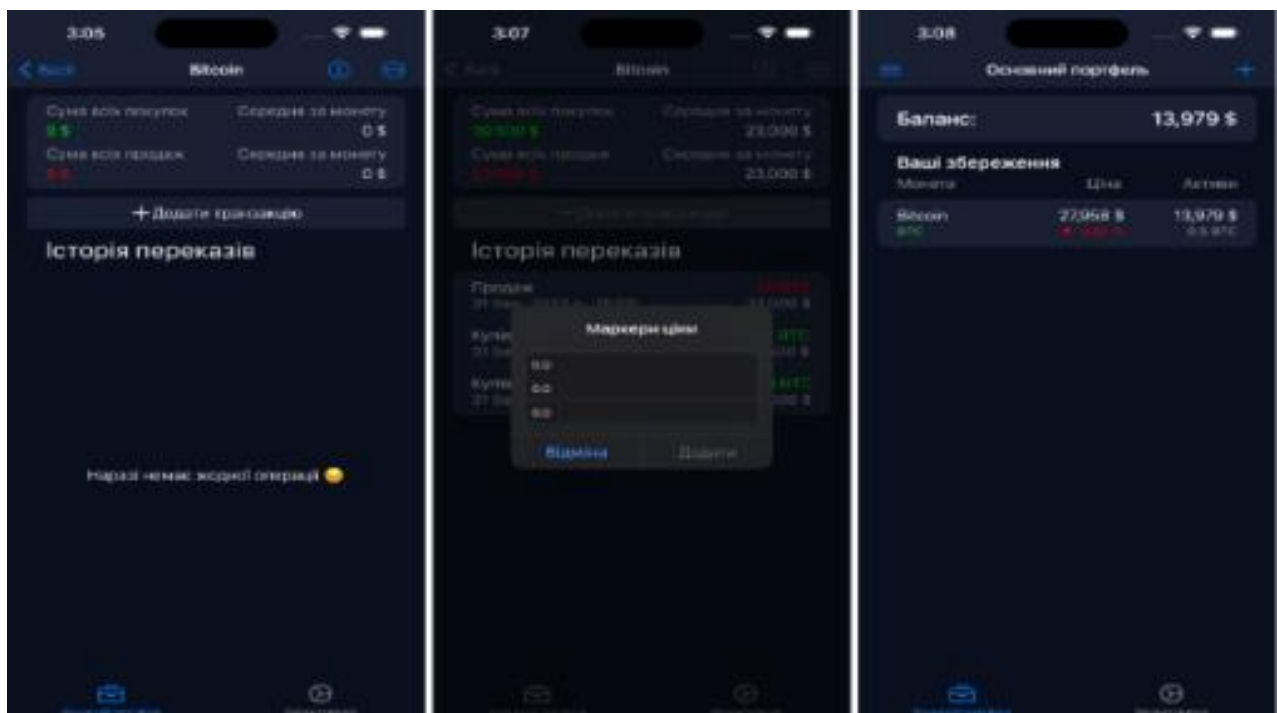
Маркери ціни вказують на рівні цін, при яких користувач хоче отримати сповіщення про зміну ціни монети в своєму криптопортфелі. Наприклад, користувач встановив три рівні цін: 22000\$, 20000\$ і 18000\$.

Якщо ціна монети впаде нижче першого маркера (22000\$), то монета буде підсвічуватися зеленим кольором. Це означає, що ціна зменшилася, але все ще перебуває в межах комфортного діапазону.

Якщо ціна монети впаде нижче другого маркера (20000\$), то монета буде підсвічуватися жовтим кольором. Це означає, що ціна знизилася настільки, що ви маєте можливість розглянути варіанти продажу монети або докупівлі на поточному рівні.

Якщо ціна монети впаде нижче третього маркера (18000\$), то монета буде підсвічуватися червоним кольором. Це означає, що ціна зменшилася настільки, що ви маєте серйозні проблеми зі своїм інвестиційним портфелем.

Для звичайного користувача, маркери ціни допомагають встановити відповідні рівні цін для змінення стану свого портфеля і забезпечити вчасне сповіщення про зміни цін монет. Це допомагає приймати обґрунтовані інвестиційні рішення та уникати несподіваних збитків.



а)

б)

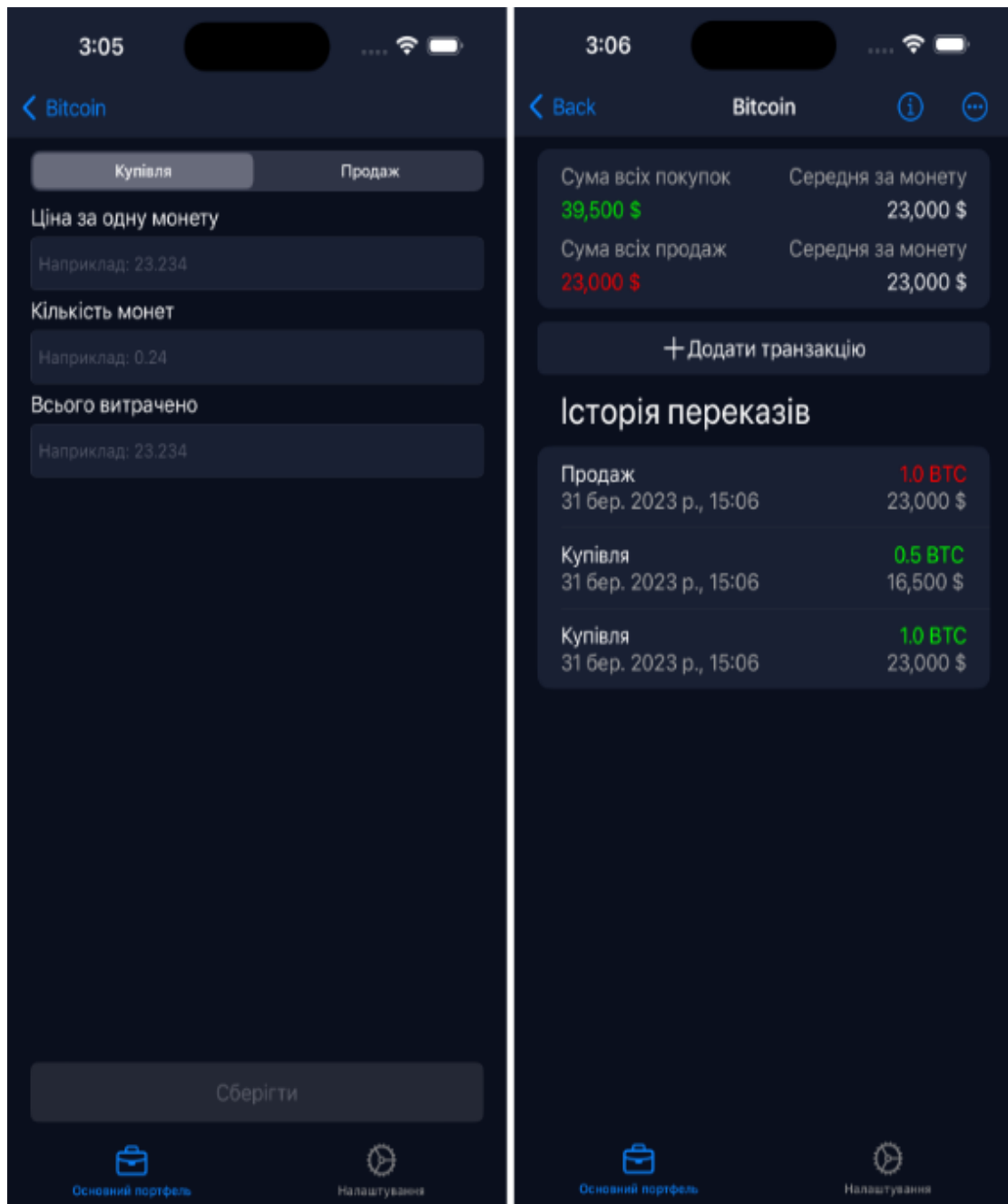
в)

а – вікно історії переказів; б – інтерфейс додавання маркерів;

в – основне вікно.

Рисунок 4.6 – Вікно транзакцій та додавання маркерів

При натисканні кнопки "Додати транзакцію" користувач потрапляє у вікно, де може додати операцію продажу або купівлі вибраної валюти (рис. 4.7).



а)

б)

а – вікно додавання транзакції; б – вікно історії переказів.

Рисунок 4.7 – Вікно додавання транзакцій

Якщо натиснути на кнопку зі знаком оклику у вікні транзакцій, то користувач потрапляє на екран, у якому бачить графік ціни на монети за певний період часу та її історичні дані (рис. 4.8).

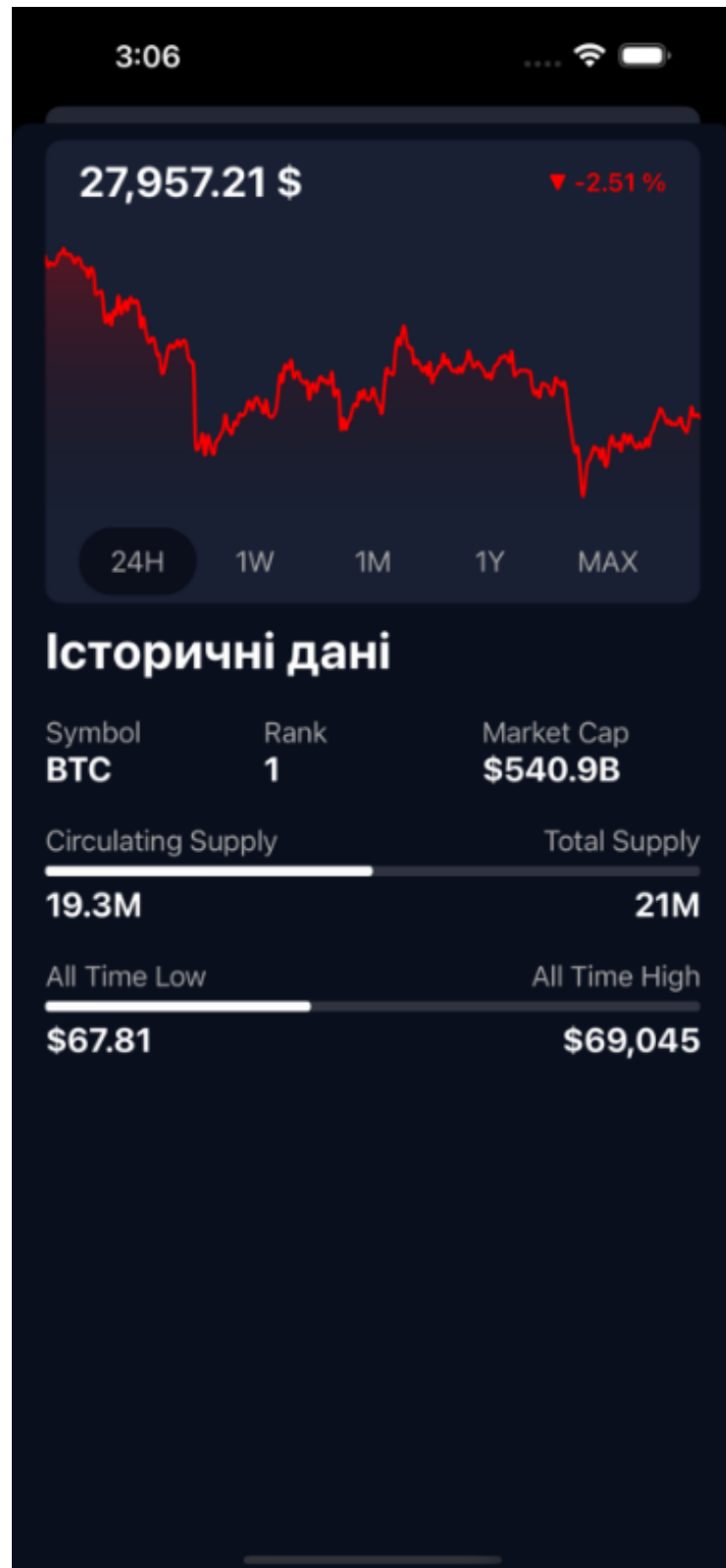
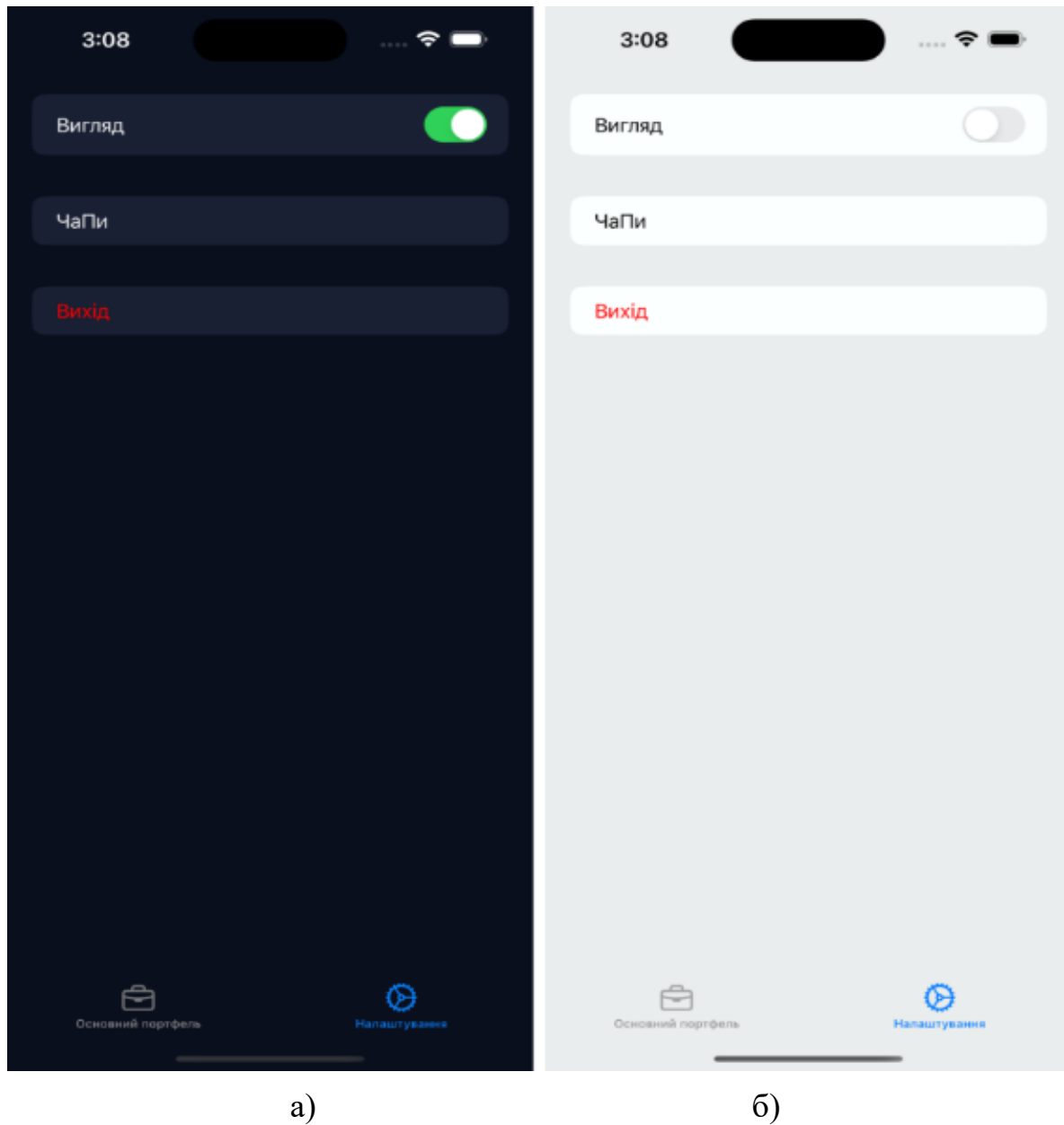


Рисунок 4.8 – Вікно історичних даних

Якщо натиснути кнопку налаштування знизу екрану то користувач потрапляє на екран у якому може вибрати вигляд застосунку прочитати часті питання або ж вийти із застосунку (рис. 4.9).



а – темний вигляд застосунку; б – світлий вигляд застосунку.

Рисунок 4.9 – Вікно налаштувань

4.5 Висновки за розділом 4

У четвертому розділі було проведено тестування розробленого застосунку, що включало визначення його призначення, умов запуску, основних дій, які користувач може виконувати в застосунку.

Були визначені умови запуску застосунку, які включають наявність пристроїв, що підтримують iOS, а також наявність доступу до мережі Інтернет.

Було проведено тестування застосунку, яке включало тестування на функціональність, тестування на стабільність та тестування на безпеку. Було виявлено та виправлено деякі помилки та недоліки у функціонуванні застосунку.

Для експлуатації застосунку було надано інструкції, які містять опис його функцій та можливостей.

ВИСНОВКИ

У результаті виконання дипломної кваліфікаційної роботи бакалавра на тему "Розробка мобільного iOS-застосунку для формування власного криптопортфеля" було створено зручний, надійний та ефективний мобільний додаток для управління криптовалютним портфелем на платформі iOS.

Застосунок інтегрується з популярними криптовалютним маркетом що дозволяє користувачам легко відстежувати свої інвестиції, проводити транзакції та оптимізувати свої стратегії на криптовалютних ринках.

Використання мови програмування Swift та Interface Builder для розробки користувацького інтерфейсу забезпечило високу адаптивність та зручність використання застосунку на різних пристроях iOS.

Розроблений додаток надає користувачам доступ до статистичних даних та сповіщень про зміни цін, що сприяє прийняттю обґрунтованих інвестиційних рішень.

Розробка проєкту передбачала використання сучасних методів аналізу, проєктування, розробки та тестування програмного забезпечення, що сприяло досягненню високої якості кінцевого продукту.

У майбутньому планується розширення функціоналу застосунку, включаючи підтримку інших платформ, інтеграцію з додатковими криптовалютними сервісами, автоматизовану торгівлю та співпрацю між користувачами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Tapscott, D. Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World / D. Tapscott, A. Tapscott. – Penguin Random House, 2016. – 384 p.
2. Thoma, M. The blockchain [Electronic resource]. – Access mode: <https://medium.com/coinmonks/the-blockchain-473aac352e5/>.
3. Narayanan, A. Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction / A. Narayanan, J. Bonneau, E. Felten, A. Miller, S. Goldfeder. – Princeton University Press, 2016. – 336 p.
4. Steinberg, D. Xcode: the definitive guide / D. Steinberg. – Apress, 2018. – 532 p.
5. About App Development with UIKit [Electronic resource]. – Access mode: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit.
6. Conway, J. iOS programming: the Big Nerd Ranch guide / J. Conway, A. Hillegass. – Big Nerd Ranch Guides, 2018. – 416 p.
7. Swift vs Objective C – Differences Explained in Detail [Electronic resource]. – Access mode: <https://www.knowledgehut.com/blog/programming/swift-vs-objective-c>.
8. Firebase. Firebase Authentication [Electronic resource]. – Access mode: <https://firebase.google.com/docs/auth?hl=en>.
9. Firebase. Add Firebase to your Apple project [Electronic resource]. – Access mode: <https://firebase.google.com/docs/ios/setup?hl=en>.
10. Neuburg, M. iOS 13 programming fundamentals with Swift / M. Neuburg. – O'Reilly Media, 2019. – 650 p.
11. Vandal, N. Swift programming: the big nerd ranch guide / N. Vandal, M. Mathias. – Big Nerd Ranch Guides, 2021. – 480 p.
12. Wenderlich, R. iOS Apprentice: Beginning iOS Development with Swift. - Razeware LLC, 2021. - 900 p.

13. Freeman, E. Head First Design Patterns: A Brain-Friendly Guide / E. Freeman, K. Sierra. – O'Reilly Media, 2004. – 694 p.
14. Klimov, A. Swift in Depth / A. Klimov. – Manning Publications, 2019. – 376 p.

ДОДАТОК А
Текст програми

A.1 Текст файла WelcomeScreenController.swift

```
import UIKit
import FirebaseAuth

class WelcomeScreenController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        navigationItem.backBarButtonItem = UIBarButtonItem(title:
"Назад", style: .plain, target: nil, action: nil)

        checkForAuthorization()
    }

    func checkForAuthorization() {
        if let user = Auth.auth().currentUser {
            let storyboard = UIStoryboard(name: "Main", bundle: nil)
            let vc = storyboard.instantiateViewController(withIdentifier:
"Main")

            self.navigationController?.isNavigationBarHidden = true
            self.navigationController?.pushViewController(vc, animated:
false)
        } else {
            // User is not logged in.
        }
    }

    @IBAction func signUpButton( sender: UIButton) {
        performSegue(withIdentifier: "signUpSegue", sender: self)
    }

    @IBAction func signInButton(_ sender: UIButton) {
```

```

        performSegue(withIdentifier: "signInSegue", sender: self)
    }

    @IBAction func prepareForUnwind(segue: UIStoryboardSegue) {}

}

```

A.2 Текст файла TransactionHistoryController.swift

```

import UIKit
import FirebaseAuth
import FirebaseFirestore

class TransactionHistoryController: UIViewController {

    //MARK: - Outlets

    @IBOutlet var avarageSellPriceLabel: UILabel!
    @IBOutlet var avarageBuyPriceLabel: UILabel!
    @IBOutlet var sumOfSellLabel: UILabel!
    @IBOutlet var sumOfBuyLabel: UILabel!
    @IBOutlet var tableView: UITableView!

    //MARK: - Properties

    var currentCoin: Coin?
    var currentPortfolio = String()
    var transactions: [TransactionData] = []

    //MARK: - ViewLifecycle

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}

```

```

        tableView.dataSource = self
        tableView.delegate = self

    }

    override func viewDidLoad(_ animated: Bool) {
        transactions = []
        loadTransactions()
    }

    //MARK: - Alert setup

    func createAlertWithThreeTextFields() -> UIAlertController {
        let alertController = UIAlertController(title: "Мапкери
ціни", message: nil, preferredStyle: .alert)

        alertController.addTextField { (textField) in
            textField.keyboardType = .decimalPad
            textField.placeholder = "Мапкер 1"
            textField.text = "\(self.currentCoin?.markerOne ?? 0)"
            textField.addTarget(self, action:
#selector(self.textFieldDidChange), for: .editingChanged)
        }

        alertController.addTextField { (textField) in
            textField.keyboardType = .decimalPad
            textField.placeholder = "Мапкер 2"
            textField.text = "\(self.currentCoin?.markerTwo ?? 0)"
            textField.addTarget(self, action:
#selector(self.textFieldDidChange), for: .editingChanged)
        }

        alertController.addTextField { (textField) in

```

```

        textField.keyboardType = .decimalPad
        textField.placeholder = "Маркеп 3"
        textField.text = "\(self.currentCoin?.markerThree ??
0)"

        textField.addTarget(self, action:
#selector(self.textFieldDidChange), for: .editingChanged)
    }

    let addAction = UIAlertAction(title: "Додати", style:
.default, handler: { (action) in
        if let textField1 = alertController.textFields?[0],
            let textField2 = alertController.textFields?[1],
            let textField3 = alertController.textFields?[2] {
            let text1 =
textField1.text?.trimmingCharacters(in: .whitespacesAndNewlines) ?? ""
            let text2 =
textField2.text?.trimmingCharacters(in: .whitespacesAndNewlines) ?? ""
            let text3 =
textField3.text?.trimmingCharacters(in: .whitespacesAndNewlines) ?? ""

            if !text1.isEmpty && !text2.isEmpty &&
!text3.isEmpty {
                self.updatePriceMarkers(markerOne:
text1.replacingOccurrences(of: ",", with: ".").doubleValue, markerTwo:
text2.replacingOccurrences(of: ",", with: ".").doubleValue, markerThree:
text3.replacingOccurrences(of: ",", with: ".").doubleValue)
            } else {
                print("One or more text fields are empty")
            }
        }
    })

    let cancelAction = UIAlertAction(title: "Відміна", style:
.cancel)

```

```

        cancelAction.isEnabled = true
        addAction.isEnabled = false

        alertController.addAction(addAction)
        alertController.addAction(cancelAction)

        return alertController
    }

    @objc func textFieldDidChange(_ textField: UITextField) {
        guard let alertController = presentedViewController as?
UIAlertController else { return }
        alertController.actions.first(where: { $0.title ==
"Додати" })?.isEnabled = alertController.textFields?.allSatisfy {
$0.text?.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty == false
} ?? false
    }

    //MARK: - DB

    func deleteTransaction(index: Int) {
        guard let uid = Auth.auth().currentUser?.uid else { return
    }

        Firestore.firestore()
            .collection("users")
            .document(uid)
            .collection("portfolios")
            .document(currentPortfolio)
            .collection("selectedCoins")
            .document(currentCoin!.id)
            .collection("transaction")
            .limit(to: 1)
    }

```

```

        .whereField("serverTimestamp", isEqualTo:
transactions.sorted { $0.serverTimestamp > $1.serverTimestamp
}[index].serverTimestamp)
        .getDocuments { (querySnapshot, error) in
            if let error = error {
                print("Error getting documents: \(error)")
            } else {
                for document in querySnapshot!.documents {
                    document.reference.delete()
                }
            }
            self.transactions = []
            self.loadTransactions()
        }
    }

func loadTransactions() {
    guard let uid = Auth.auth().currentUser?.uid else { return
}

```

Firestore

```

        .firestore()
        .collection("users")
        .document(uid)
        .collection("portfolios")
        .document(currentPortfolio)
        .collection("selectedCoins")
        .document(currentCoin!.id)
        .collection("transaction")
        .getDocuments { (snapshot, error) in
            guard let snapshot = snapshot, error == nil else {
                print("error")
                return
            }
        }
    }

```

```

        snapshot.documents.forEach({ (documentSnapshot) in
            let documentData = documentSnapshot.data()

            let timestamp = documentData["timestamp"] as?
String
            let amountOfCoins =
documentData["amountOfCoins"] as? Double
            let totalSpent = documentData["totalSpent"]
as? Double
            let pricePerCoins =
documentData["pricePerCoin"] as? Double
            let transactionType =
documentData["transactionType"] as? Int
            let serverTimestamp =
documentData["serverTimestamp"] as? Double

            let data = TransactionData(timestamp:
timestamp!, amountOfCoins: amountOfCoins!, pricePerCoin: pricePerCoins!,
totalPrice: totalSpent!, transactionType: transactionType!,
serverTimestamp: serverTimestamp!)

            self.transactions.append(data)

        })
        self.tableViewIsEmpty()
        self.tableView.reloadData()
        self.updateInfoView()
    }
}

func updatePriceMarkers(markerOne: Double, markerTwo: Double,
markerThree: Double) {
    guard let uid = Auth.auth().currentUser?.uid else { return
}

```

```

        Firestore.firestore()
            .collection("users")
            .document(uid)
            .collection("portfolios")
            .document(currentPortfolio)
            .collection("selectedCoins")
            .document(currentCoin!.id)
            .updateData(["marker1": markerOne,
                        "marker2": markerTwo,
                        "marker3": markerThree])

        currentCoin?.markerOne = markerOne
        currentCoin?.markerTwo = markerTwo
        currentCoin?.markerThree = markerThree
    }

    @IBAction func addTransactionPressed(_ sender: UIButton) {
        performSegue(withIdentifier: "addTransaction", sender:
self)
    }

    @IBAction func addMarkersButton(_ sender: UIBarButtonItem) {
        let alertController = createAlertWithThreeTextFields()
        present(alertController, animated: true, completion: nil)
    }

    override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
        if segue.identifier == "addTransaction" {
            if let destiantionVC = segue.destination as?
AddTransactionController {
                destiantionVC.currentCoin = currentCoin
            }
        }
    }

```

```

        destiantionVC.currentPortfolio = currentPortfolio
    }
}

if segue.identifier == "toHistoricalData" {
    if let destinationVC = segue.destination as?
CoinHistoricalInfoController {
        destinationVC.currentCoin = currentCoin!.id
        destinationVC.currentCoinSymbol =
currentCoin!.symbol
    }
}

//MARK: - View Update

func updateInfoView() {
    // Filter transactions by type
    let type0Transactions = transactions.filter {
$0.transactionType == 0 }
    let type1Transactions = transactions.filter {
$0.transactionType == 1 }

    // Calculate total amount of totalPrice for each type
    let type0TotalPrice = type0Transactions.reduce(0) { $0 +
$1.totalPrice }
    let type1TotalPrice = type1Transactions.reduce(0) { $0 +
$1.totalPrice }

    // Calculate average price of totalPrice for each type
    let type0AveragePrice = type0Transactions.reduce(0) { $0 +
$1.pricePerCoin } / Double(type0Transactions.count)
    let type1AveragePrice = type1Transactions.reduce(0) { $0 +
$1.pricePerCoin } / Double(type1Transactions.count)

```

```

        if type0Transactions.count == 0 {
            avarageBuyPriceLabel.text = "0 $"
            sumOfBuyLabel.text = "0 $"
        } else {
            avarageBuyPriceLabel.text =
"\(type0AveragePrice.roundToDecimal(2).formatted()) $"
            sumOfBuyLabel.text =
"\(type0TotalPrice.roundToDecimal(2).formatted()) $"
        }

        if type1Transactions.count == 0 {
            avarageSellPriceLabel.text = "0 $"
            sumOfSellLabel.text = "0 $"
        } else {
            avarageSellPriceLabel.text =
"\(type1AveragePrice.roundToDecimal(2).formatted()) $"
            sumOfSellLabel.text =
"\(type1TotalPrice.roundToDecimal(2).formatted()) $"
        }
    }
}

```

//MARK: - TableView

```

extension TransactionHistoryController: UITableViewDelegate,
UITableViewDataSource {

    func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"transactionCell", for: indexPath) as! TransactionCell

```

```

        cell.configure(transactionData: transactions.sorted {
$0.serverTimestamp > $1.serverTimestamp }, coin: currentCoin!, index:
indexPath.row)
        return cell
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSectionSection
section: Int) -> Int {
        return transactions.count
    }

    func tableView(_ tableView: UITableView,
heightForHeaderInSection section: Int) -> CGFloat {
        return CGFloat.leastNormalMagnitude
    }

    func tableView(_ tableView: UITableView,
trailingSwipeActionsConfigurationForRowAt indexPath: IndexPath) ->
UISwipeActionsConfiguration? {
        let deleteAction = UIContextualAction(style: .destructive,
title: "Delete") { contextualAction, view, boolValue in
            self.deleteTransaction(index: indexPath.row)
        }
        let swipeActions = UISwipeActionsConfiguration(actions:
[deleteAction])
        return swipeActions
    }

    func tableViewIsEmpty() {
        if transactions.isEmpty {
            let label = UILabel(frame: CGRect(x: 0, y: 0, width:
tableView.bounds.size.width, height: tableView.bounds.size.height))
            label.text = "Наразі немає жодної операції ☹️"
            label.textAlignment = .center

```

```

        tableView.backgroundView = label
        tableView.separatorStyle = .none
    } else {
        tableView.backgroundView = nil
        tableView.separatorStyle = .singleLine
    }
}
}

```

A.3 Текст файла CoinSearchController.swift

```

import UIKit
import FirebaseAuth
import FirebaseFirestore

class CoinSearchController: UIViewController {

    //MARK: - Properties

    var coins: [Coin] = []
    var currentPortfolio = String()
    var filteredCoins: [Coin] = []

    //MARK: - Outlets

    @IBOutlet var searchBar: UISearchBar!
    @IBOutlet var tableView: UITableView!

    //MARK: - ViewLifecycle

    override func viewDidLoad() {

```

```

        super.viewDidLoad()
        self.hideKeyboardWhenTappedAround()
        searchBar.searchTextField.textColor = UIColor(named:
"textColorSet")
        tableView.delegate = self
        tableView.dataSource = self
        searchBar.delegate = self

        if let url = Bundle.main.url(forResource:
"SupportedCoins", withExtension: "json") {
            do {
                let data = try Data(contentsOf: url)
                let decoder = JSONDecoder()
                let jsonData = try decoder.decode([Coin].self,
from: data)

                coins = jsonData
                filteredCoins = coins
            } catch {
                print("error:(error)")
            }
        }
    }

    //MARK: - TableView

    extension CoinSearchController: UITableViewDelegate,
UITableViewDataSource {
        func tableView( tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
            return filteredCoins.count
        }
    }

```

```

func tableView( tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier:
"cell", for: indexPath)
    cell.textLabel?.text =
"(filteredCoins[indexPath.row].name)
((filteredCoins[indexPath.row].symbol))"
    cell.textLabel?.textColor = UIColor(named: "textColorSet")
    return cell
}

```

```

func tableView( tableView: UITableView,
heightForHeaderInSection section: Int) -> CGFloat {
    return CGFloat.leastNormalMagnitude
}

```

```

func tableView( tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
    let coinSymbol = filteredCoins[indexPath.row].symbol
    let coinName = filteredCoins[indexPath.row].name
    let coinID = filteredCoins[indexPath.row].id

    let data = ["coinID": coinID,
                "coinName": coinName,
                "coinSymbol": coinSymbol,
                "marker1": 0,
                "marker2": 0,
                "marker3": 0] as [String : Any]

    let uid = Auth.auth().currentUser!.uid

    Firestore.firestore()
        .collection("users")
        .document(uid)

```

```

        .collection("portfolios")
        .document(currentPortfolio)
        .collection("selectedCoins")
        .document(coinID)
        .setData(data)

        self.navigationController?.popViewController(animated:
true)
    }
}

//MARK: - Searchbar

extension CoinSearchController: UISearchBarDelegate {
    func searchBar(_ searchBar: UISearchBar, textDidChange
searchText: String) {
        filteredCoins = coins

        if searchText.isEmpty == false {
            filteredCoins = coins.filter {
$0.name.localizedCaseInsensitiveContains(searchText) }
        }

        tableView.reloadData()
    }
}

```

A.4 Текст файла MainViewController.swift

```

import UIKit
import FirebaseAuth
import FirebaseFirestore

```

```

class MainViewController: UIViewController {

    //MARK: - Outlets
    @IBOutlet var totalBalance: UILabel!
    @IBOutlet var tableView: UITableView!

    //MARK: - Properties

    var selectedCoinIndex = 0
    var userCoins = [Coin]()
    var coinsBalance = [Double]()
    var coinPrice = [CoinPrice]()
    var currentPortfolio = String()
    var tD = [TransactionData]()

    //MARK: - ViewLifecycle

    override func viewDidLoad() {
        super.viewDidLoad()
        tableView.delegate = self
        tableView.dataSource = self

        configureRefreshControl()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        FirestoreUtilities().fetchCurrentUserPortfolio {
portfolioTitle in
            self.currentPortfolio = portfolioTitle
            self.title = portfolioTitle
            self.loadDataAndUpdateTableView()
        }
    }
}

```

```

        override func viewDidAppear(_ animated: Bool) {
            if let title = self.navigationItem.title {
                currentPortfolio = title
            }
        }

        override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {

            if segue.identifier == "transactionTab" {
                if let destinationVC = segue.destination as?
TransactionHistoryController {
                    destinationVC.title =
userCoins[selectedCoinIndex].name
                    destinationVC.currentCoin =
userCoins[selectedCoinIndex]
                    destinationVC.currentPortfolio = currentPortfolio
                }
            }

            if segue.identifier == "toUserPortfolios" {
                if let destinationVC = segue.destination as?
UserPortfoliosController {
                    destinationVC.delegate = self
                }
            }

            if segue.identifier == "searchForCoin" {
                if let destinationVC = segue.destination as?
CoinSearchController {
                    destinationVC.currentPortfolio = currentPortfolio
                }
            }
        }
    }

```

```

//MARK: - RefreshControl

func configureRefreshControl () {
    tableView.refreshControl = UIRefreshControl()
    tableView.refreshControl?.backgroundColor = .clear
    tableView.refreshControl?.tintColor = .white
    tableView.refreshControl?.addTarget(self, action:

#selector(handleRefreshControl),

                                for: .valueChanged)
}

@objc func handleRefreshControl() {
    loadDataAndUpdateTableView()

    DispatchQueue.main.async {
        self.tableView.refreshControl?.endRefreshing()
    }
}

@IBAction func addCoinButton(_ sender: UIBarButtonItem) {
    performSegue(withIdentifier: "searchForCoin", sender:
self)
}

//MARK: - DB

func deleteCoin(index: Int) {
    guard let uid = Auth.auth().currentUser?.uid else { return
}

    let db = Firestore.firestore()

```

```

        let selectedCoinRef =
db.collection("users").document(uid).collection("portfolios").document(cu
rrentPortfolio).collection("selectedCoins").document(userCoins[index].id)
        selectedCoinRef.collection("transaction").getDocuments {
snapshot, error in
            if let error = error {
                print(error.localizedDescription)
            } else {
                guard let snapshot = snapshot else { return }
                for transactionDoc in snapshot.documents {
                    transactionDoc.reference.delete()
                }
                selectedCoinRef.delete()
                self.loadDataAndUpdateTableView()
            }
        }
    }
}

```

```

func searchForSelectedCoins(completion: @escaping ([Coin]?,
Error?) -> Void) {
    guard let userID = Auth.auth().currentUser?.uid else {
        completion(nil, NSError(domain: "com.example.app",
code: 401, userInfo: [NSLocalizedStringKey: "User not
authenticated"])))
        return
    }
}

```

```

Firestore.firestore()
    .collection("users")
    .document(userID)
    .collection("portfolios")
    .document(currentPortfolio)
    .collection("selectedCoins")
    .getDocuments { snapshot, error in

```

```

        guard let snapshot = snapshot, error == nil else {
            completion(nil, error)
            return
        }

        let userCoins = snapshot.documents.compactMap {
documentSnapshot -> Coin? in
            let documentData = documentSnapshot.data()
            guard let coinID = documentData["coinID"] as?
String,
                let coinName = documentData["coinName"]
as? String,
                let coinSymbol =
documentData["coinSymbol"] as? String,
                let marker1 = documentData["marker1"]
as? Double,
                let marker2 = documentData["marker2"]
as? Double,
                let marker3 = documentData["marker3"]
as? Double else {
                    return nil
                }
            return Coin(id: coinID, symbol: coinSymbol,
name: coinName, markerOne: marker1, markerTwo: marker2, markerThree:
marker3 )
        }

        completion(userCoins, nil)
    }
}

func loadCoinBalance(completion: @escaping ([Double]) -> Void)
{

```

```

guard let uid = Auth.auth().currentUser?.uid else { return
}

var coinsBalanceDict = [String: Double]()
var coinsBalance = [Double]()
var requestsCount = userCoins.count

for coin in userCoins {
    Firestore.firestore()
        .collection("users")
        .document(uid)
        .collection("portfolios")
        .document(currentPortfolio)
        .collection("selectedCoins")
        .document(coin.id)
        .collection("transaction")
        .getDocuments() { snapshot, error in
            if let error = error {
                print(error.localizedDescription)
            } else {
                var sumTokens: [Double] = []
                if let snapshot = snapshot,
!snapshot.isEmpty {
                    for document in snapshot.documents {
                        let transactionType =
document.get("transactionType") as? Int
                        let amountOfCoins =
document.get("amountOfCoins") as? Double
                        if transactionType == 0 {

sumTokens.append(amountOfCoins!)

                        } else if transactionType == 1 {
                            sumTokens.append(-
amountOfCoins!)
                        }
                    }
                }
            }
        }
    }
}

```

```

        }
    }
    coinsBalanceDict[coin.id] =
sumTokens.sum()
    }

    requestsCount -= 1
    if requestsCount == 0 {
        for coin in self.userCoins {
            if let balance =
coinsBalanceDict[coin.id] {
                coinsBalance.append(balance)
            } else {
                coinsBalance.append(0)
            }
        }
        completion(coinsBalance)
    }
}

}

}

//MARK: - API

func getCurrentCoinPrice(completion: @escaping ([CoinPrice]) -
> Void) {
    let coinIDs = userCoins.map { $0.id }.joined(separator:
",")

    let urlString =
"https://api.coingecko.com/api/v3/simple/price?ids=\(coinIDs)&vs_currenci
es=usd&include_24hr_change=true"
    print(urlString)

    guard let url = URL(string: urlString) else {

```

```

        print("Error: Invalid URL")
        return
    }

    let task = URLSession.shared.dataTask(with: url) { data,
response, error in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }

        guard let data = data else {
            print("Error: No data received")
            return
        }

        do {
            let coinDictionary = try
JSONDecoder().decode([String: [String: Double?]].self, from: data)
            var coins: [CoinPrice] = coinDictionary.compactMap
{ coinID, values in
                let usd24hChange = values["usd_24h_change"] ??
0

                let currentUSDPrice = values["usd"] ?? 0

                return CoinPrice(id: coinID, usd24hChange:
usd24hChange, currentUSDPrice: currentUSDPrice)
            }
            coins.sort { coin1, coin2 in
                guard let index1 =
self.userCoins.firstIndex(where: { $0.id == coin1.id }),
                    let index2 =
self.userCoins.firstIndex(where: { $0.id == coin2.id }) else {
                    return false

```

```

        }
        return index1 < index2
    }
    completion(coins)
} catch {
    print("Error: \(error.localizedDescription)")
}
}
task.resume()
}

```

```

func calculateTotalPriceOfAssets() -> Double{
    var resultsArray: [Double] = []

    for i in 0..

```

//MARK: - Tableview

```

extension MainViewController: UITableViewDelegate,
UITableViewDataSource {
    func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"coinListCell", for: indexPath) as! MainViewCoinCell

```

```

        cell.configure(coin: userCoins, currentPrice: coinPrice,
userAmountOfCoins: coinsBalance, index: indexPath.row)
        return cell
    }

    func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
        return userCoins.count
    }

    func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
        selectedCoinIndex = indexPath.row
        performSegue(withIdentifier: "transactionTab", sender:
self)
    }

    func tableView(_ tableView: UITableView,
heightForHeaderInSection section: Int) -> CGFloat {
        return CGFloat.leastNormalMagnitude
    }

    func tableView(_ tableView: UITableView,
trailingSwipeActionsConfigurationForRowAt indexPath: IndexPath) ->
UISwipeActionsConfiguration? {
        let deleteAction = UIContextualAction(style: .destructive,
title: "Delete") { contextualAction, view, boolValue in
            self.deleteCoin(index: indexPath.row)
        }
        let swipeActions = UISwipeActionsConfiguration(actions:
[deleteAction])
        return swipeActions
    }

```

```

func tableViewIsEmpty() {
    if userCoins.isEmpty {
        let label = UILabel(frame: CGRect(x: 0, y: 0, width:
tableView.bounds.size.width, height: tableView.bounds.size.height))
        label.text = "Наразі немає жодної монети 😞"
        label.textAlignment = .center
        tableView.backgroundView = label
        tableView.separatorStyle = .none
    } else {
        tableView.backgroundView = nil
        tableView.separatorStyle = .singleLine
    }
}

func loadDataAndUpdateTableView() {

    searchForSelectedCoins { userCoin, error in

        guard let userCoin = userCoin, error == nil else {
            print("Error: \(error?.localizedDescription ??
"Unknown error")")
            return
        }
        self.userCoins = userCoin

        if self.userCoins.count == 0 {
            DispatchQueue.main.async {
                self.totalBalance.text = "0 $"
                self.tableViewIsEmpty()
                self.tableView.reloadData()
            }
        }
    }
}

```

```

self.loadCoinBalance { coinBalances in

    self.getCurrentCoinPrice { coinPrices in

        self.coinPrice = coinPrices
        self.coinsBalance = coinBalances
        DispatchQueue.main.async {
            self.totalBalance.text =
"\(self.calculateTotalPriceOfAssets().roundToDecimal(2).formatted()) $"
            self.tableViewIsEmpty()
            self.tableView.reloadData()
        }
    }
}

}

}

}

}

//MARK: - Popover

extension MainViewController: PopoverDelegate {
    func didDismissPopover() {
        FirestoreUtilities().fetchCurrentUserPortfolio {
portfolioTitle in

            self.currentPortfolio = portfolioTitle
            self.title = portfolioTitle
            self.loadDataAndUpdateTableView()
        }
    }
}

```

A.5 Текст файла AddTransactionController.swift

```
import UIKit
import FirebaseAuth
import FirebaseFirestore

class AddTransactionController: UIViewController {
    //MARK: - Outlets

    @IBOutlet var pricePerCoinTextfield: UITextField!
    @IBOutlet var totalSpentTextfield: UITextField!
    @IBOutlet var amountOfCoinsTextfield: UITextField!
    @IBOutlet var textLabel: UILabel!
    @IBOutlet var saveButton: UIButton!

    //MARK: - Properties

    var currentPortfolio = String()
    var currentCoin: Coin?
    var type: Int = 0

    //MARK: - ViewLifecycle

    override func viewDidLoad() {
        super.viewDidLoad()
        self.hideKeyboardWhenTappedAround()
        saveButton.isEnabled = false
        setupTextFields()
    }

    //MARK: - DB

    @IBAction func transactionTypeChanged(_ sender:
UISegmentedControl) {
```

```

type = sender.selectedSegmentIndex

if type == 0 {
    textLabel.text = "Всього витрачено"
} else {
    textLabel.text = "Всього отримано"
}
}

@IBAction func saveButtonPressed(_ sender: UIButton) {
    guard let uid = Auth.auth().currentUser?.uid else { return
}

    if let amountOfCoins = amountOfCoinsTextfield.text,
        let pricePerCoin = pricePerCoinTextfield.text,
        let totalSpent = totalSpentTextfield.text {

        let data = TransactionData(timestamp:
getCurrentTime(), amountOfCoins: amountOfCoins.replacingOccurrences(of:
",", with: ".").doubleValue, pricePerCoin:
pricePerCoin.replacingOccurrences(of: ",", with: ".").doubleValue,
totalPrice: totalSpent.replacingOccurrences(of: ",", with:
".").doubleValue, transactionType: type, serverTimestamp:
NSDate().timeIntervalSince1970)

        let data1 = ["pricePerCoin": data.pricePerCoin,
            "timestamp": data.timestamp,
            "totalSpent": data.totalPrice,
            "amountOfCoins": data.amountOfCoins,
            "transactionType": data.transactionType,
            "serverTimestamp": data.serverTimestamp]
as [String : Any]

        if let currentCoinID = currentCoin?.id {

```

```

        Firestore.firestore()
            .collection("users")
            .document(uid)
            .collection("portfolios")
            .document(currentPortfolio)
            .collection("selectedCoins")
            .document(currentCoinID)
            .collection("transaction")
            .document()
            .setData(data1)

        navigationController?.popViewController(animated:
true)
    }
}

//MARK: - TextfieldSetup

@objc func textFieldDidChange(_ textField: UITextField) {
    updateButtonEnabled()
}

func setupTextFields() {
    pricePerCoinTextfield.addTarget(self, action:
#selector(textFieldDidChange(_:)), for: .editingChanged)
    totalSpentTextfield.addTarget(self, action:
#selector(textFieldDidChange(_:)), for: .editingChanged)
    amountOfCoinsTextfield.addTarget(self, action:
#selector(textFieldDidChange(_:)), for: .editingChanged)
}

func updateButtonEnabled() {

```

```

        if let text1 = pricePerCoinTextfield.text, let text2 =
totalSpentTextfield.text, let text3 = amountOfCoinsTextfield.text {
            saveButton.isEnabled = !text1.trimmingCharacters(in:
.whitespacesAndNewlines).isEmpty && !text2.trimmingCharacters(in:
.whitespacesAndNewlines).isEmpty && !text3.trimmingCharacters(in:
.whitespacesAndNewlines).isEmpty
        } else {
            saveButton.isEnabled = false
        }
    }

func getCurrentTime() -> String {
    let currentDate = Date()
    let dateFormatter = DateFormatter()
    dateFormatter.locale = Locale(identifier: "uk-UA")
    dateFormatter.setLocalizedDateFormatFromTemplate("MMM d
yyyy HH:mm ")
    let dateString = dateFormatter.string(from: currentDate)
    return dateString
}
}

```

A.6 Текст файлу SignUpController.swift

```

import UIKit
import Firebase
import FirebaseAuth
import FirebaseFirestore

class SignUpController: UIViewController {

    @IBOutlet var emailTextfield: UITextField!
    @IBOutlet var passwordTextfield: UITextField!

```

[illegible]

```

userRef.collection("portfolios").document("Основной
портфель").setData(portfolioData) { error in
    if let error = error {
        print(error.localizedDescription)
    } else {
        let storyboard = UIStoryboard(name:
"Main", bundle: nil)

        let vc =
storyboard.instantiateViewController(withIdentifier: "Main")

self.navigationController?.isNavigationBarHidden = true

self.navigationController?.pushViewController(vc, animated: false)
    }
}
}
}
}
}
}
}
}
}
}

```

A.7 Текст файла SettingsController.swift

```

import UIKit
import FirebaseAuth

class SettingsController: UITableViewController {

    @IBOutlet var switchDarkModeStatus: UISwitch!

    override func viewDidLoad() {
        super.viewDidLoad()

        // Do any additional setup after loading the view.
    }
}

```

```

    }

    override func viewWillAppear( animated: Bool) {
        super.viewWillAppear(animated)

        if UserDefaultsModel.shared.theme == .dark {
            switchDarkModeStatus.isOn = true
        } else {
            switchDarkModeStatus.isOn = false
        }
    }

    @IBAction func darkModePressed( sender: UISwitch) {
        var darkMode = 0
        if sender.isOn {
            darkMode = 0
        } else {
            darkMode = 1
        }
        UserDefaultsModel.shared.theme = Theme(rawValue: darkMode)
    }

    ?? .light

        self.view.window?.overrideUserInterfaceStyle =
UserDefaultsModel.shared.theme.getUserInterfaceStyle()
    }

    override func tableView(_ tableView: UITableView,
didSelectRowAt indexPath: IndexPath) {
        if indexPath.section == 2 && indexPath.row == 0 {
            try? Auth.auth().signOut()

            self.performSegue(withIdentifier:
"unwindToWelcomeScreen", sender: self)
        }
    }
}

```

A.8 Текст файла SignInController.swift

```
import UIKit
import FirebaseAuth
import Firebase

class SignInController: UIViewController {

    @IBOutlet var emailTextField: UITextField!
    @IBOutlet var passwordTextField: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()
        self.hideKeyboardWhenTappedAround()
    }

    @IBAction func signInButton(_ sender: UIButton) {
        if let email = emailTextField.text, let password =
passwordTextField.text {
            Auth.auth().signIn(withEmail: email, password:
password) { authResult, error in
                if let e = error {
                    print(e.localizedDescription)
                } else {
                }
            }
        }
    }
}
```

A.9 Текст файла MainViewCoinCell.swift

```
import UIKit

class MainViewCoinCell: UITableViewCell {

    @IBOutlet var coinName: UILabel!
    @IBOutlet var coinCurrentPrice: UILabel!
    @IBOutlet var priceOfUserAsset: UILabel!
    @IBOutlet var coinTicker: UILabel!
    @IBOutlet var price24hChange: UILabel!
    @IBOutlet var amountOfUserCoin: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
    }

    func configure(coin: [Coin], currentPrice: [CoinPrice],
userAmountOfCoins: [Double], index: Int) {
        coinName.text = coin[index].name
        coinTicker.text = coin[index].symbol.uppercased()
        amountOfUserCoin.text =
"(userAmountOfCoins[index].roundToDecimal(5))
(coin[index].symbol.uppercased())"

        let value = userAmountOfCoins[index] *
(currentPrice[index].currentUSDPrice ?? 0)
        priceOfUserAsset.text =
"(value.roundToDecimal(2).formatted()) $"
    }
}
```

```

        if currentPrice[index].currentUSDPrice ?? 0 < 1 {
            coinCurrentPrice.text =
"(currentPrice[index].currentUSDPrice?.roundToDecimal(10).formatted() ??
"0") $"

        } else {
            coinCurrentPrice.text =
"(currentPrice[index].currentUSDPrice?.roundToDecimal(2).formatted() ??
"0") $"

        }

        if currentPrice[index].usd24hChange ?? 0 < 0 {
            price24hChange.textColor = .red
            price24hChange.text = "▼
(currentPrice[index].usd24hChange?.roundToDecimal(2) ?? 0) %"
        } else {
            price24hChange.textColor = .green
            price24hChange.text = "▲
(currentPrice[index].usd24hChange?.roundToDecimal(2) ?? 0) %"
        }

        if currentPrice[index].currentUSDPrice ?? 0 <
coin[index].markerOne! {
            coinTicker.textColor = .systemGreen
            if currentPrice[index].currentUSDPrice ?? 0 <
coin[index].markerTwo! {
                coinTicker.textColor = .systemYellow
                if currentPrice[index].currentUSDPrice ?? 0 <
coin[index].markerThree! {
                    coinTicker.textColor = .systemRed
                }
            }
        } else {
            coinTicker.textColor = .lightGray
        }

```

```

    }
}

```

A.10 Текст файла UserPortfolios.swift

```

import UIKit
import FirebaseAuth
import FirebaseFirestore

class UserPortfoliosController: UIViewController {
    //MARK: - Outlets

    @IBOutlet var tableView: UITableView!

    //MARK: - Properties

    var userPortfolios = [PortfolioData]()
    var alertTextField: UITextField?
    var addAction: UIAlertAction?
    var delegate: PopoverDelegate?

    //MARK: - ViewLifecycle

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
        tableView.delegate = self
        tableView.dataSource = self

        loadUserPortfolios()
    }

    override func viewDidDisappear(_ animated: Bool) {

```

```

        delegate?.didDismissPopover()
    }

//MARK: - DB

func deletePortfolio(index: Int) {
    guard let uid = Auth.auth().currentUser?.uid else { return
}

    let portfolioRef = Firestore.firestore()
        .collection("users")
        .document(uid)
        .collection("portfolios")
        .document(userPortfolios[index].portfolioTitle)

    portfolioRef.delete()

    portfolioRef.collection("selectedCoins").getDocuments {
snapshot, error in
        if let error = error {
            print(error.localizedDescription)
            return
        } else {
            guard let snapshot = snapshot else { return }

            for doc in snapshot.documents {

                let transactionRef =
doc.reference.collection("transaction")

                transactionRef.getDocuments { snapshot, error
in
                    if let error = error {

```

```

        print(error.localizedDescription)
    } else {
        guard let snapshot = snapshot else {
return }

        for transaction in snapshot.documents
{
            transaction.reference.delete()
        }
    }
    doc.reference.delete()
}
}
}
}
}

```

```

@IBAction func addPortfolioButtonPressed(_ sender: UIButton) {
    let alert = UIAlertController(title: "Додати портфель",
message: "", preferredStyle: .alert)
    alert.addTextField { textField in
        textField.placeholder = "Назва портфеля"
        self.alertTextField = textField
        self.addAction?.isEnabled = false
        textField.autocapitalizationType = .words
        textField.addTarget(self, action:
#selector(self.textFieldDidChange(_)), for: .editingChanged)
    }

    addAction = UIAlertAction(title: "Додати", style:
.default) { action in
        if let portfolioTitle = self.alertTextField?.text {
            guard let userID = Auth.auth().currentUser?.uid
else { return }

```

```

        Firestore.firestore()
            .collection("users")
            .document(userID)
            .collection("portfolios")
            .document(portfolioTitle)
            .setData(["portfolio": portfolioTitle,
                    "isSelected": false])

        self.loadUserPortfolios()
    }
}

addAction?.isEnabled = false

let cancel = UIAlertAction(title: "Відміна", style:
.cancel, handler: nil)

alert.addAction(addAction!)
alert.addAction(cancel)

present(alert, animated: true, completion: nil)
}

@objc func textFieldDidChange(_ textField: UITextField) {
    addAction?.isEnabled =
!textField.text!.trimmingCharacters(in: .whitespacesAndNewlines).isEmpty
}

@IBAction func dismissPopoverButton(_ sender: UIButton) {
    self.dismiss(animated: true)
}

func loadUserPortfolios() {
    print("pampam")
}

```

```

        userPortfolios = []

        guard let userID = Auth.auth().currentUser?.uid else {
            return }

        Firestore.firestore()
            .collection("users")
            .document(userID)
            .collection("portfolios")
            .getDocuments { snapshot, error in
                if let e = error {
                    print(e.localizedDescription)
                    return
                } else {
                    if let snapshot = snapshot {
                        for document in snapshot.documents {
                            let isSelected =
                                document.get("isSelected") as? Bool
                            let portfolioTitle =
                                document.get("portfolio") as? String

                            self.userPortfolios.append(PortfolioData(portfolioTitle: portfolioTitle!,
                                isSelected: isSelected!))
                        }
                    }
                }
                self.userPortfolios.sort(by: { $0.isSelected &&
                    !$1.isSelected })
                self.tableView.reloadData()
            }
        }
    }
}

```

```
//MARK: - Tableview
```

```
extension UserPortfoliosController: UITableViewDelegate,
UITableViewDataSource {
    func tableView(_ tableView: UITableView, numberOfRowsInSectionSection: Int) -> Int {
        return userPortfolios.count
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "portfolioCell", for: indexPath)

        cell.textLabel?.text =
userPortfolios[indexPath.row].portfolioTitle
        cell.textLabel?.textColor = UIColor(named: "textColorSet")

        if userPortfolios[indexPath.row].isSelected == true {
            cell.accessoryType = .checkmark
        } else {
            cell.accessoryType = .none
        }

        return cell
    }

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        guard let userID = Auth.auth().currentUser?.uid else {
return }

        for portfolio in userPortfolios {
            if portfolio.isSelected == true {
```

```

        Firestore.firestore()
            .collection("users")
            .document(userID)
            .collection("portfolios")
            .document(portfolio.portfolioTitle)
            .updateData(["isSelected": false])
    }
}

```

```

Firestore.firestore()
    .collection("users")
    .document(userID)
    .collection("portfolios")

```

```

.document(userPortfolios[indexPath.row].portfolioTitle)
    .updateData(["isSelected": true])

```

```

loadUserPortfolios()
}

```

```

func tableView(_ tableView: UITableView,
trailingSwipeActionsConfigurationForRowAt indexPath: IndexPath) ->
UISwipeActionsConfiguration? {

    let deleteAction = UIContextualAction(style: .destructive,
title: nil) { contextualAction, view, boolValue in
        if self.userPortfolios[indexPath.row].isSelected ==
true {

            let alertController = UIAlertController(title:
"Увага", message: "Неможливо видалити вибраний портфель.",
preferredStyle: .alert)

            let okAction = UIAlertAction(title: "Добре",
style: .default) { _ in }

            alertController.addAction(okAction)

```

```

        self.present(alertController, animated: true,
completion: nil)
    } else {
        self.deletePortfolio(index: indexPath.row)
        self.loadUserPortfolios()
    }
}

deleteAction.image = UIImage(systemName: "trash.fill")

let swipeActions = UISwipeActionsConfiguration(actions:
[deleteAction])

return swipeActions
}

}

```

A.11 Текст файла TransactionCell.swift

```

import UIKit

class TransactionCell: UITableViewCell {

    @IBOutlet var dateLabel: UILabel!
    @IBOutlet var amountOfCoinsLabel: UILabel!
    @IBOutlet var totalSpentLabel: UILabel!
    @IBOutlet var transactionTypeLabel: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }
}

```

```

        override func setSelected(_ selected: Bool, animated: Bool) {
            super.setSelected(selected, animated: animated)

            // Configure the view for the selected state
        }

        func configure(transactionData: [TransactionData], coin: Coin,
            index: Int) {

            if transactionData[index].transactionType == 0 {
                dateLabel.text = transactionData[index].timestamp
                totalSpentLabel.text =
                    "(transactionData[index].totalPrice.roundToDecimal(2).formatted()) $"
                amountOfCoinsLabel.text =
                    "(transactionData[index].amountOfCoins) (coin.symbol.uppercased())"
                amountOfCoinsLabel.textColor = .green
                transactionTypeLabel.text = "Купівля"
            } else {
                dateLabel.text = transactionData[index].timestamp
                totalSpentLabel.text =
                    "(transactionData[index].totalPrice.roundToDecimal(2).formatted()) $"
                amountOfCoinsLabel.text =
                    "(transactionData[index].amountOfCoins) (coin.symbol.uppercased())"
                amountOfCoinsLabel.textColor = .red
                transactionTypeLabel.text = "Продаж"
            }

        }

    }
}

```

A.12 Текст файла CoinHistoricalInfoController.swift

```
import UIKit
import Charts

class CoinHistoricalInfoController: UIViewController {

    //MARK: - ViewLifecycle

    override func viewDidLoad() {
        super.viewDidLoad()

        fetchHistoricalCoinData(coinID: currentCoin) { data in
            DispatchQueue.main.async {
                self.symbolLabel.text =
self.currentCoinSymbol.uppercased()
                self.rankLabel.text =
"\(data.marketData.marketCapRank)"
                self.marketCapLabel.text =
"$\(data.marketData.marketCap.usd.shortenedString())"
                self.circulationSupplyLabel.text =
data.marketData.circulatingSupply.shortenedString()
                self.totalSupplyLabel.text =
data.marketData.totalSupply?.shortenedString()
                self.atlLabel.text =
"$\(data.marketData.atl?.usd.roundToDecimal(2).formatted() ?? "0")"
                self.athLabel.text =
"$\(data.marketData.ath?.usd.roundToDecimal(2).formatted() ?? "0")"

                let percentage = (data.marketData.currentPrice.usd
– (data.marketData.atl?.usd ?? 0.0)) / ((data.marketData.ath?.usd ?? 0.0)
– (data.marketData.atl?.usd ?? 0.0))
                self.atlAthProgressView.progress =
Float(percentage)
```

```

    }
}

lineChartView.delegate = self

fetchDataForChart(days: "1", coinSymbol: currentCoin) {
data in
    self.chartData = data.map { ChartDataEntry(x:
Double($0.timestamp), y: $0.value) }
    DispatchQueue.main.async {
        self.setupLineChart()
        _ = self.percentageDifference()
        self.priceLabel.text =
"\(self.chartData.last?.y.roundToDecimal(2).formatted() ?? "0") $"
    }
}

h24Button.layer.cornerRadius = 20
h24Button.backgroundColor = UIColor(
    red: CGFloat(0x0B) / 255.0,
    green: CGFloat(0x0F) / 255.0,
    blue: CGFloat(0x1C) / 255.0,
    alpha: 1.0)

stackView.layer.cornerRadius = 10
stackView.clipsToBounds = true
lineChartView.minOffset = 0
}

//MARK: - Properties

var chartData = [ChartDataEntry]()
var currentCoin = String()

```

```

        var currentCoinSymbol = String()
        let historicalData =
HisctoricalCoinData.MarketData(currentPrice:
HisctoricalCoinData.MarketData.CurrentPriceData(usd: 0.0), marketCapRank:
0, marketCap: HisctoricalCoinData.MarketData.CurrentMarketCap(usd: 0.0),
circulatingSupply: 0.0, totalSupply: nil, ath: nil, atl: nil)

//MARK: - Outlets

@IBOutlet var supplyProgressView: UIProgressView!
@IBOutlet var atlAthProgressView: UIProgressView!
@IBOutlet var athLabel: UILabel!
@IBOutlet var atlLabel: UILabel!
@IBOutlet var totalSupplyLabel: UILabel!
@IBOutlet var circulationSupplyLabel: UILabel!
@IBOutlet var marketCapLabel: UILabel!
@IBOutlet var rankLabel: UILabel!
@IBOutlet var symbolLabel: UILabel!
@IBOutlet var stackView: UIStackView!
@IBOutlet var priceLabel: UILabel!
@IBOutlet var dateLabel: UILabel!
@IBOutlet var lineChartView: LineChartView!
@IBOutlet var priceView: UIView!
@IBOutlet var timeView: UIView!
@IBOutlet var h24Button: UIButton!
@IBOutlet var w1Button: UIButton!
@IBOutlet var m1Button: UIButton!
@IBOutlet var y1Button: UIButton!
@IBOutlet var maxButton: UIButton!

@IBAction func timeRangeButton(_ sender: UIButton) {
    let buttonArray = [h24Button, w1Button, m1Button,
y1Button, maxButton]

```

```

    for button in buttonArray {
        if button == sender {
            button?.layer.cornerRadius = 20
            button?.backgroundColor = UIColor(
                red: CGFloat(0x0B) / 255.0,
                green: CGFloat(0x0F) / 255.0,
                blue: CGFloat(0x1C) / 255.0,
                alpha: 1.0)
        } else {
            button?.backgroundColor = UIColor.clear
        }
    }

    switch sender.titleLabel!.text {
    case "24H":
        fetchDataForChart(days: "1", coinSymbol: currentCoin)
    { data in
        self.chartData = data.map { ChartDataEntry(x:
Double($0.timestamp), y: $0.value) }
        DispatchQueue.main.async {
            self.setupLineChart()
            self.priceLabel.text =
"\(self.chartData.last?.y.roundToDecimal(2).formatted() ?? "0") $"
        }
    }
    case "1W":
        fetchDataForChart(days: "7", coinSymbol: currentCoin)
    { data in
        self.chartData = data.map { ChartDataEntry(x:
Double($0.timestamp), y: $0.value) }
        DispatchQueue.main.async {
            self.setupLineChart()
            self.priceLabel.text =
"\(self.chartData.last?.y.roundToDecimal(2).formatted() ?? "0") $"

```

```

        }
    }
    case "1M":
        fetchDataForChart(days: "30", coinSymbol: currentCoin)
    { data in
        self.chartData = data.map { ChartDataEntry(x:
Double($0.timestamp), y: $0.value) }
        DispatchQueue.main.async {
            self.setupLineChart()
            self.priceLabel.text =
"\(self.chartData.last?.y.roundToDecimal(2).formatted() ?? "0") $"
        }
    }
    case "1Y":
        fetchDataForChart(days: "365", coinSymbol:
currentCoin) { data in
            self.chartData = data.map { ChartDataEntry(x:
Double($0.timestamp), y: $0.value) }
            DispatchQueue.main.async {
                self.setupLineChart()
                self.priceLabel.text =
"\(self.chartData.last?.y.roundToDecimal(2).formatted() ?? "0") $"
            }
        }
    case "MAX":
        fetchDataForChart(days: "max", coinSymbol:
currentCoin) { data in
            self.chartData = data.map { ChartDataEntry(x:
Double($0.timestamp), y: $0.value) }
            DispatchQueue.main.async {
                self.setupLineChart()
                self.priceLabel.text =
"\(self.chartData.last?.y.roundToDecimal(2).formatted() ?? "0") $"
            }
        }
    }
}

```

```

    }
    default:
        break
    }
}

```

```
//MARK: - API
```

```

func fetchDataForChart(days: String, coinSymbol: String,
completion: @escaping ([ChartData]) -> Void) {
    let urlString =
"https://api.coingecko.com/api/v3/coins/\(coinSymbol)/market_chart?vs_currency=usd&days=\(days)"

    guard let url = URL(string: urlString) else {
        print("Error: Invalid URL")
        return
    }

    let task = URLSession.shared.dataTask(with: url) { data,
response, error in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }

        guard let data = data else {
            print("Error: No data received")
            return
        }

        do {
            let decoder = JSONDecoder()

```

```

        let decodedData = try
decoder.decode(ChartDataResponse.self, from: data)
        let chartData = decodedData.chartData
        completion(chartData)
    } catch {
        print("Error: \(error.localizedDescription)")
    }
}

task.resume()
}

func fetchHistoricalCoinData(coinID: String, completion:
@escaping (HisctoricalCoinData) -> Void) {
    let urlString =
"https://api.coingecko.com/api/v3/coins/\(coinID)"

    guard let url = URL(string: urlString) else {
        print("Error: Invalid URL")
        return
    }

    let task = URLSession.shared.dataTask(with: url) { data,
response, error in
        if let error = error {
            print("Error: \(error.localizedDescription)")
            return
        }

        guard let data = data else {
            print("Error: No data received")
            return
        }

```

```

        do {
            let decoder = JSONDecoder()
            let decodedData = try
decoder.decode(HisctoricalCoinData.self, from: data)
            let historicalCoinData = decodedData
            completion(historicalCoinData)
        } catch {
            print("Error: \(error.localizedDescription)")
        }
    }

    task.resume()
}
}

//MARK: - Chart

```

```

extension CoinHistoricalInfoController: ChartViewDelegate {

    func chartValueSelected(_ chartView: ChartViewBase, entry:
ChartDataEntry, highlight: Highlight) {
        let timestamp = entry.x / 1000.0
        let date = Date(timeIntervalSince1970:
TimeInterval(timestamp))
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "d/M/yyyy HH:mm"
        let dateString = dateFormatter.string(from: date)

        dateLabel.textColor = UIColor(named: "textColorSet")
        dateLabel.text = dateString
        priceLabel.text =
"\(entry.y.roundToDecimal(2).formatted()) $"
    }

    func chartViewDidEndPanning(_ chartView: ChartViewBase) {

```

```

        chartView.highlightValue(nil)

        priceLabel.text =
"\(chartData.last?.y.roundToDecimal(2).formatted() ?? "0") $"
        _ = percentageDifference()
    }

    func percentageDifference() -> Double {
        let difference = (chartData.last?.y ?? 0.0) -
(chartData.first?.y ?? 0.0)
        let percentage = difference / (chartData.first?.y ?? 0.0)
* 100

        if percentage > 0 {
            dateLabel.textColor = .green
            dateLabel.text = "▲ \(percentage.roundToDecimal(2)) %"
        } else {
            dateLabel.textColor = .red
            dateLabel.text = "▼ \(percentage.roundToDecimal(2)) %"
        }
        return percentage
    }

    func setupLineChart() {

        lineChartView.backgroundColor = UIColor(named:
"tableViewColorSet")
        lineChartView.legend.enabled = false
        lineChartView.rightAxis.enabled = false

        let xAxis = lineChartView.xAxis
        xAxis.drawLabelsEnabled = false
        xAxis.drawAxisLineEnabled = false

```

```

xAxis.drawGridLinesEnabled = false

let leftAxis = lineChartView.leftAxis
leftAxis.drawLabelsEnabled = false
leftAxis.drawAxisLineEnabled = false
leftAxis.drawGridLinesEnabled = false

// Create line chart data set
let lineChartDataSet = LineChartDataSet(entries:
chartData, label: "Coin Price")
lineChartDataSet.lineWidth = 2
lineChartDataSet.mode = .cubicBezier
lineChartDataSet.drawCirclesEnabled = false
lineChartDataSet.drawValuesEnabled = false
lineChartDataSet.drawHorizontalHighlightIndicatorEnabled =
false

lineChartDataSet.highlightColor = .lightGray

//Create gradient

if percentageDifference() > 0 {
    lineChartDataSet.colors = [.green]
    let gradientColors = [UIColor.green.cgColor,
UIColor.clear.cgColor]
    let colorLocations:[CGFloat] = [1.0, 0.0]
    let gradient = CGGradient(colorsSpace:
CGColorSpaceCreateDeviceRGB(), colors: gradientColors as CFArray,
locations: colorLocations)
    lineChartDataSet.fill = LinearGradientFill(gradient:
gradient!, angle: 90)
    lineChartDataSet.drawFilledEnabled = true
} else {
    lineChartDataSet.colors = [.red]

```

```

        let gradientColors = [UIColor.red.cgColor,
UIColor.clear.cgColor]
        let colorLocations:[CGFloat] = [1.0, 0.0]
        let gradient = CGGradient(colorsSpace:
CGColorSpaceCreateDeviceRGB(), colors: gradientColors as CFArray,
locations: colorLocations)
        lineChartDataSet.fill = LinearGradientFill(gradient:
gradient!, angle: 90)
        lineChartDataSet.drawFilledEnabled = true
    }

    // Create line chart data
    let lineChartData = LineChartData(dataSet:
lineChartDataSet)
    lineChartView.data = lineChartData
}
}

```

A.13 Текст файла CoinData.swift

```

import Foundation
import FirebaseFirestore

struct Coin: Codable {
    let id: String
    let symbol: String
    let name: String
    var markerOne: Double?
    var markerTwo: Double?
    var markerThree: Double?
}

```

A.14 Текст файла TransactionData.swift

```
import Foundation

struct TransactionData: Codable {
    let timestamp: String
    let amountOfCoins: Double
    let pricePerCoin: Double
    let totalPrice: Double
    let transactionType: Int
    let serverTimestamp: Double
}
```

A.15 Текст файла PriceData.swift

```
import Foundation

struct PriceData: Codable {
    let coinID: String
    let currentUSDPrice: Double
    let usd24hChange: Double
}

struct CoinPrice: Codable {
    let id: String
    let usd24hChange: Double?
    let currentUSDPrice: Double?

    enum CodingKeys: String, CodingKey {
        case id
        case usd24hChange = "usd_24h_change"
        case currentUSDPrice = "usd"
    }
}
```

```
}
```

A.16 Текст з файла PortfolioData.swift

```
import Foundation
```

```
struct PortfolioData: Codable {
    let portfolioTitle: String
    let isSelected: Bool
}
```

A.17 Текст файлу DataForChart.swift

```
import Foundation
```

```
struct ChartData: Codable {
    let timestamp: Int
    let value: Double
}
```

```
struct ChartDataResponse: Codable {
    let prices: [[Double]]

    var chartData: [ChartData] {
        return prices.map { ChartData(timestamp: Int($0[0]),
value: $0[1]) }

    }

}
```

A.18 Текст файла HistoricalCoinData.swift

```
import Foundation

struct HisctoricalCoinData: Codable {
    let marketData: MarketData

    enum CodingKeys: String, CodingKey {
        case marketData = "market_data"
    }

    struct MarketData: Codable {
        let currentPrice: CurrentPriceData
        let marketCapRank: Int
        let marketCap: CurrentMarketCap
        let circulatingSupply: Double
        let totalSupply: Double?
        let ath: AthData?
        let atl: AtlData?

        enum CodingKeys: String, CodingKey {
            case currentPrice = "current_price"
            case marketCapRank = "market_cap_rank"
            case marketCap = "market_cap"
            case circulatingSupply = "circulating_supply"
            case totalSupply = "total_supply"
            case ath, atl
        }
    }

    struct CurrentMarketCap: Codable {
        let usd: Double
    }
}
```

```

    struct CurrentPriceData: Codable {
        let usd: Double
    }

    struct AthData: Codable {
        let usd: Double
    }

    struct AtlData: Codable {
        let usd: Double
    }
}
}

```

A.19 Текст файла Extensions.swift

```

import Foundation
import UIKit

extension Sequence where Element: AdditiveArithmetic {
    func sum() -> Element { reduce(.zero, +) }
}

extension Double {
    func roundToDecimal(_ fractionDigits: Int) -> Double {
        let multiplier = pow(10, Double(fractionDigits))
        return Darwin.round(self * multiplier) / multiplier
    }
}

extension String {
    var doubleValue: Double {
        return Double(self) ?? 0
    }
}

```

```

    }
}

extension String {
    func isBlankOrEmpty() -> Bool {

        // Check empty string
        if self.isEmpty {
            return true
        }
        // Trim and check empty string
        return (self.trimmingCharacters(in: .whitespaces) == "")
    }
}

extension UIView {
    func roundCorners(corners: UIRectCorner, radius: CGFloat) {
        let maskPath = UIBezierPath(roundedRect: bounds,
                                     byRoundingCorners: corners,
                                     cornerRadii: CGSize(width:
radius, height: radius))
        let maskLayer = CAShapeLayer()
        maskLayer.path = maskPath.cgPath
        layer.mask = maskLayer
    }

    func roundBottomCorners(radius: CGFloat) {
        roundCorners(corners: [.bottomLeft, .bottomRight], radius:
radius)
    }

    func roundTopCorners(radius: CGFloat) {
        roundCorners(corners: [.topLeft, .topRight], radius:
radius)
    }
}

```

```

    }
}

extension UIViewController {
    func hideKeyboardWhenTappedAround() {
        let tap = UITapGestureRecognizer(target: self, action:
#selector(UIViewController.dismissKeyboard))
        tap.cancelsTouchesInView = false
        view.addGestureRecognizer(tap)
    }

    @objc func dismissKeyboard() {
        view.endEditing(true)
    }
}

extension Double {
    func shortenedString() -> String {
        let number = abs(self)
        let suffixes = ["", "K", "M", "B", "T"]
        var suffixIndex = 0
        var shortened = number
        while shortened >= 1000 && suffixIndex < suffixes.count -
1 {
            shortened /= 1000
            suffixIndex += 1
        }
        let formatter = NumberFormatter()
        formatter.numberStyle = .decimal
        formatter.maximumFractionDigits = 1
        let formattedNumber = formatter.string(from:
NSNumber(value: shortened)) ?? "(shortened)"
        return "(self < 0 ? "-" :
"")(formattedNumber)(suffixes[suffixIndex])"
    }
}

```

```
}
```

A.20 Текст файла UserDefaults.swift

```
import Foundation

struct UserDefaultsModel {
    static var shared = UserDefaultsModel()

    var theme: Theme {
        get {
            return Theme(rawValue:
UserDefaults.standard.integer(forKey: "selectedTheme")) ?? .light
        }
        set {
            UserDefaults.standard.set(newValue.rawValue, forKey:
"selectedTheme")
        }
    }
}
```

ДОДАТОК Б
Слайди презентації

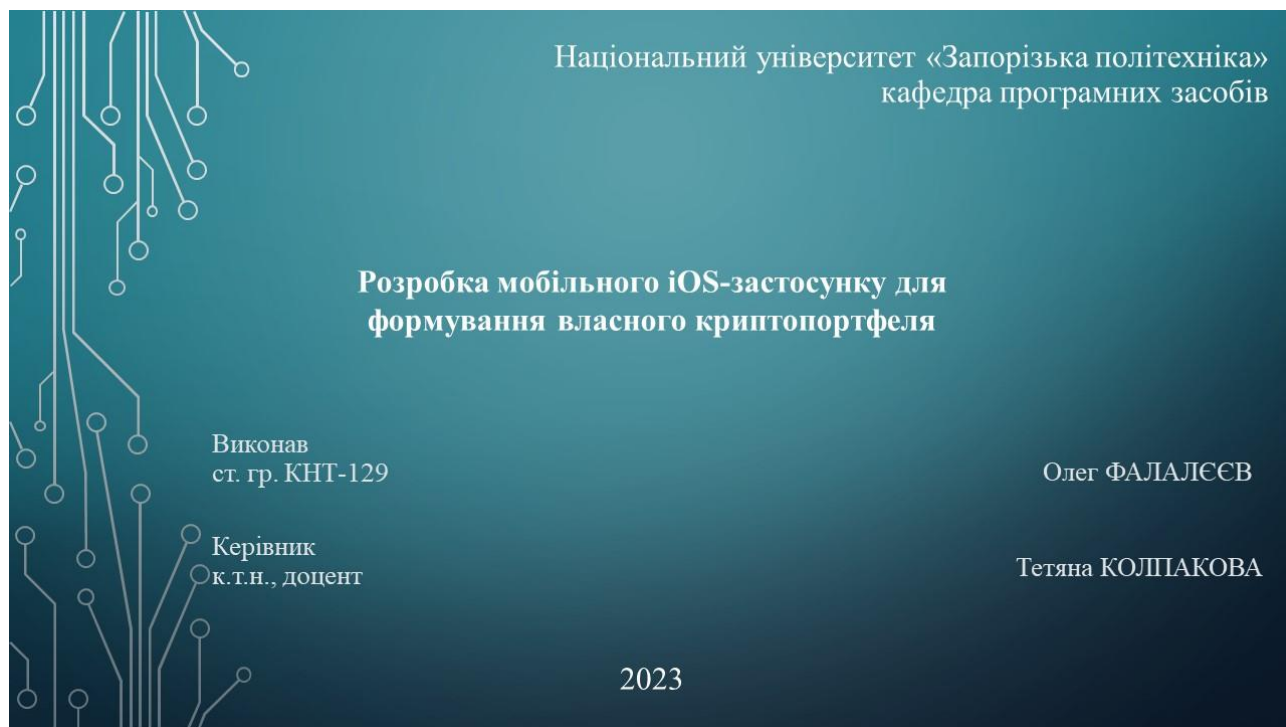


Рисунок В.1 – Слайд № 1

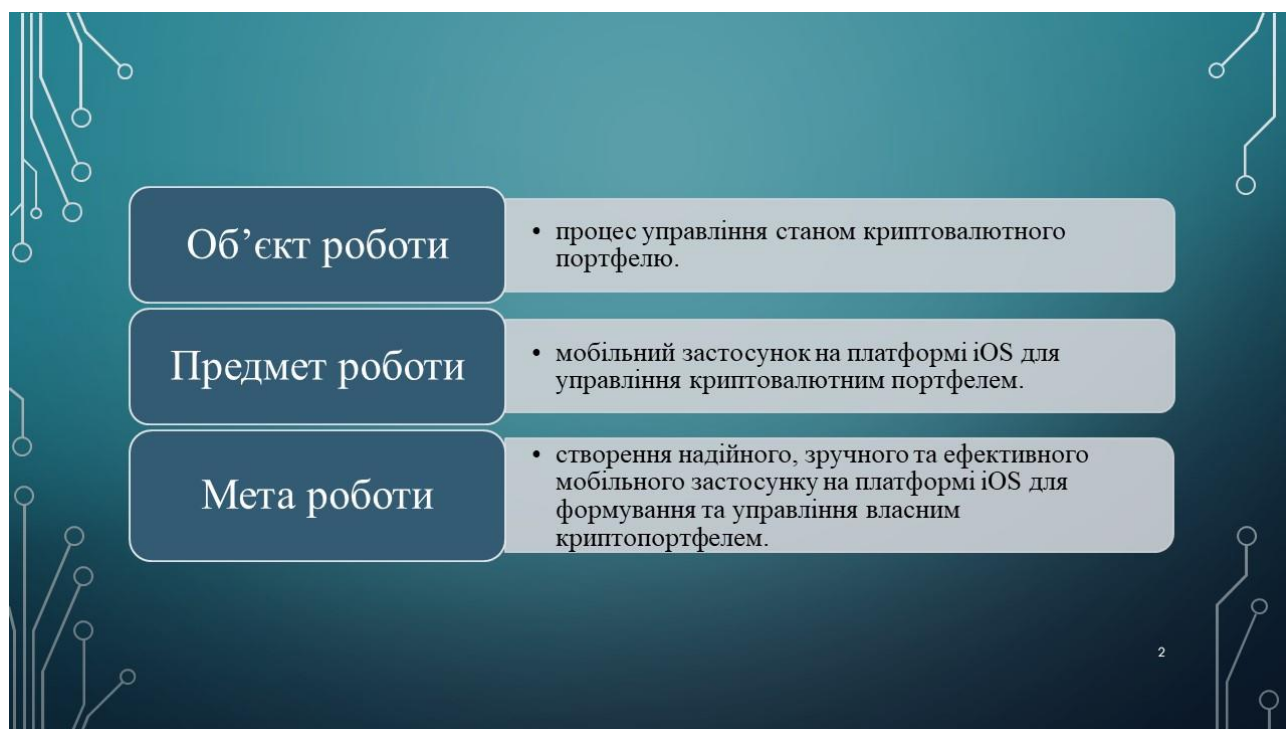


Рисунок В.2 – Слайд № 2

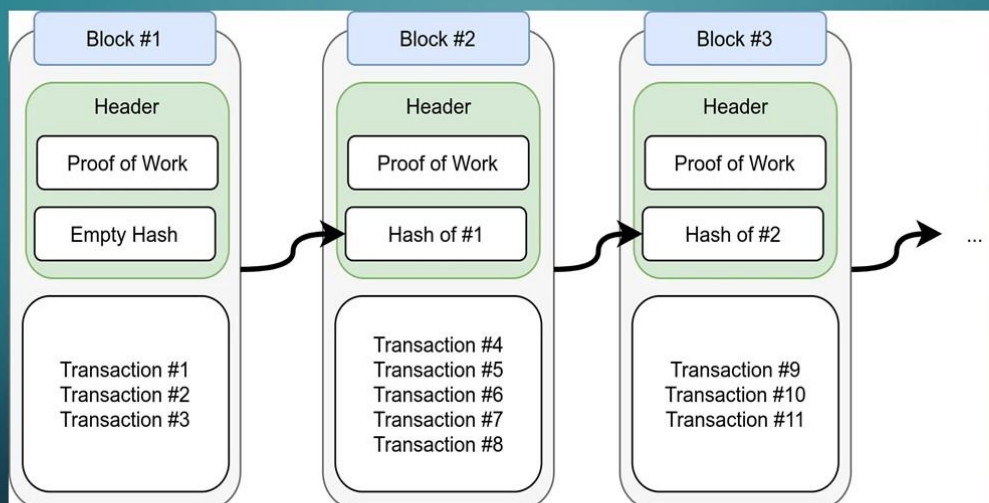
Завдання роботи

- аналіз предметної області;
- проектування програмного забезпечення криптовалютного портфеля;
- програмна реалізація криптовалютного портфеля;
- тестування розробленого програмного забезпечення.

3

Рисунок В.3 – Слайд № 3

Блокчейн технологія



4

Рисунок В.4 – Слайд № 4

Протоколи консенсусу

	Proof of Work (PoW)	Proof of Stake (PoS)
Опис	Доказ роботи	Доказ співпраці
Процес доказування	Майнінг	Холдінг
Споживання енергії	Дуже високе	Низьке
Складність налаштування	Висока	Низька
Швидкість підтвердження	Повільна	Швидка
Ризик централізації	Високий	Менший ризик централізації
Витрати на обладнання	Високі	Не потрібне
Безпека	Дуже висока	Висока
Ресурсозалежність	Сильна залежність від обладнання та енергії	Не залежить від обладнання та енергії
Приклад блокчейнів	Bitcoin, Ethereum Classic, Litecoin	Ethereum, Cardano, Algorand
Перспективність	Менш перспективний в контексті сталої екології	Більш перспективний в контексті сталої екології, дуже багато протоколів розглядають PoS як альтернативу PoW

Рисунок В.5 – Слайд № 5

Мови програмування iOS

	Swift	Objective-C
Впровадження	Новіша мова з більш сучасними можливостями	Старіша мова, що потребує поновлення
Спадщина	Наслідуює C та Objective-C, але з більш сучасним підходом	Наслідуює тільки C та Smalltalk, менш сучасна спадщина
Читабельність коду	Дуже зрозумілий та легкий для читання	Менш зрозумілий та складний для читання
Розширюваність	Протоколи та розширення дозволяють легко додавати функціональність	Категорії та делегування менш гнучкі
Швидкодія	До 2,6 разів швидший за Objective-C	Повільніший за Swift
Підтримка бібліотек	Є	Є
Розробка UI	Interface Builder забезпечує легкість розробки інтерфейсу	Інтерфейс розробляється через взаємодію з кодом

Рисунок В.6 – Слайд № 6

Вибір бази даних

	CoreData	Realm	Firestore
Тип бази даних	Локальна	Локальна та мережева	Хмарна
Хмарна синхронізація	-	-	+
Підтримка мережових з'єднань	-	+	+
Підтримка спільного доступу до даних	-	+	+
Масштабованість	Середня	Висока	Висока
Швидкість доступу до даних	Висока	Висока	Висока

Рисунок В.7 – Слайд № 7

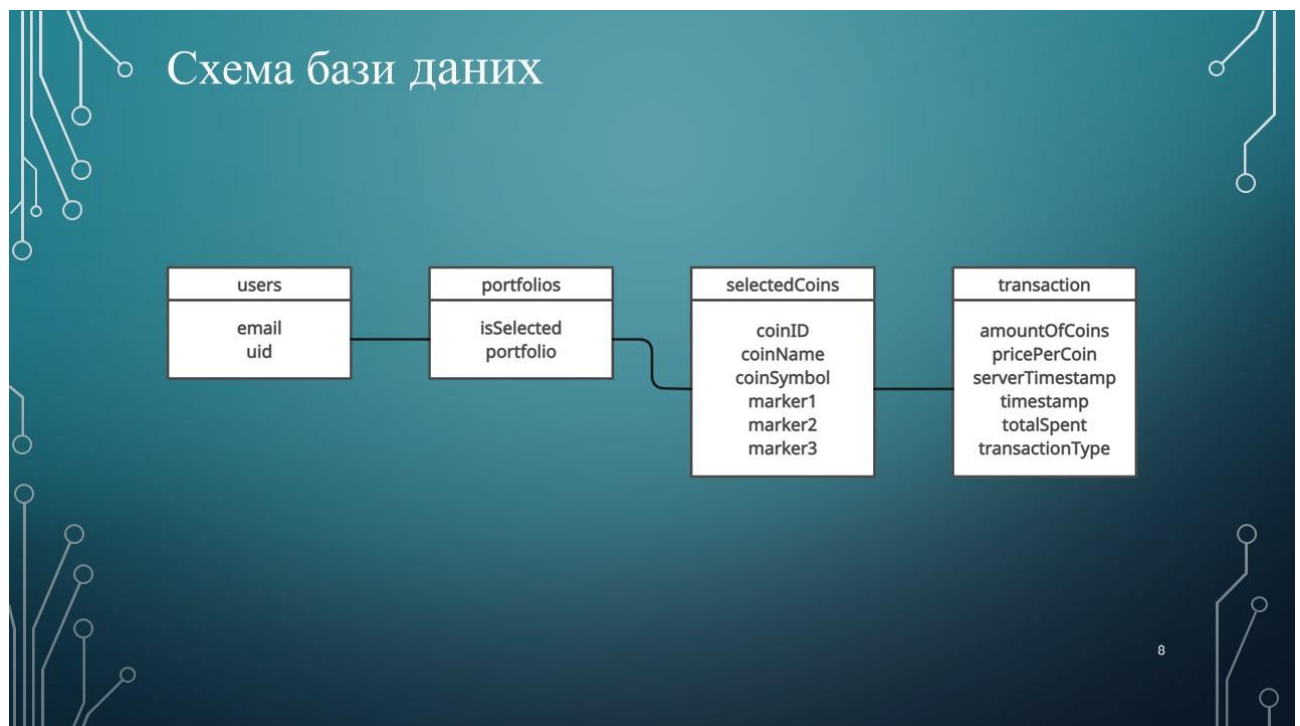


Рисунок В.8 – Слайд № 8

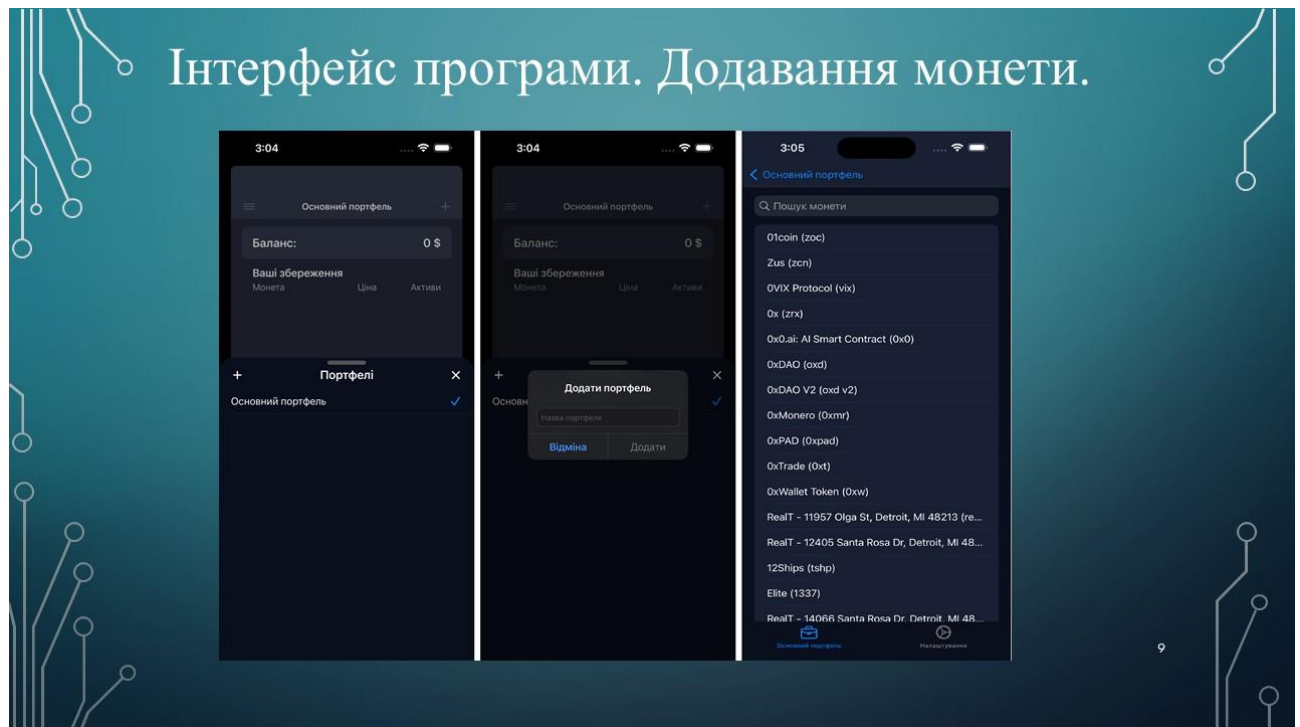


Рисунок В.9 – Слайд № 9

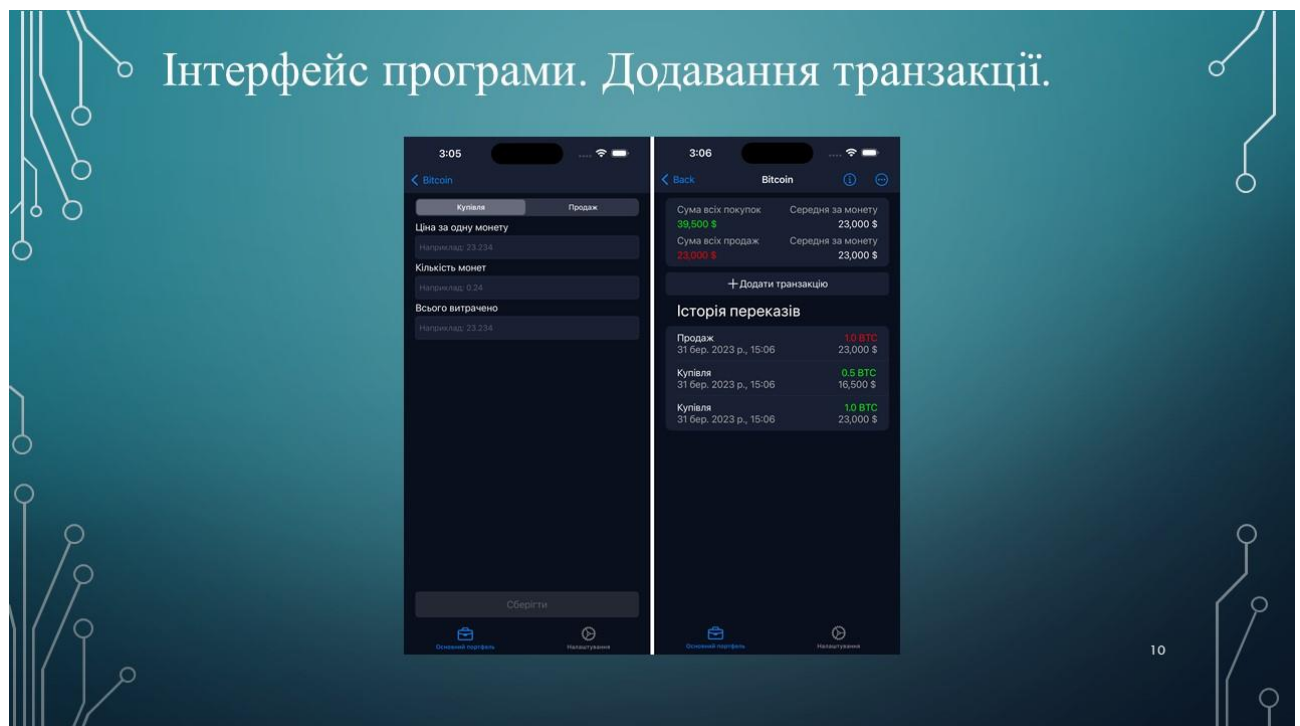


Рисунок В.10 – Слайд № 10

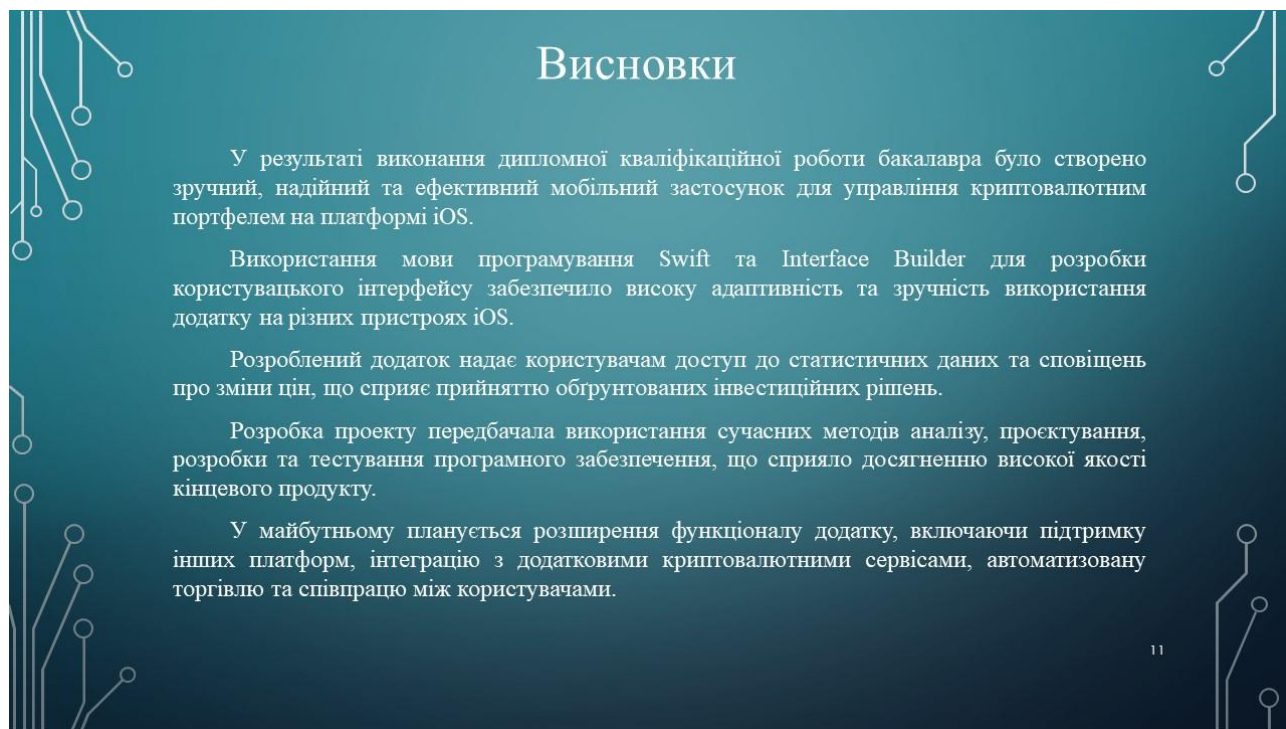


Рисунок В.11 – Слайд № 11