Міністерство освіти і науки
України Національний
технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(навчально-науковий інститут)

факультет інформаційних технологій
(факультет)

Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня __бакалавра__
(бакалавра, магістра)

Здобувача вищої освіти Бурдіна Андрія Дмитровича
(ПІБ)

академічної групи 126-21-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою
(за наявності)
Інформаційні системи та технології
(офіційна назва)

на тему Розробка веб-застосунку для моніторингу інвестиційного портфелю криптовалют зі зміною показників прибутковості у режимі реального часу
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	доц. Сергєєва К.Л			
розділів:				
Аналіз стану області рішення завдання	доц. Сергєєва К.Л			
Проєктні рішення	доц. Сергєєва К.Л			
Тестування та результати роботи	доц. Сергєєва К.Л			

Рецензент				

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
_____Інформаційних технологій та інженерії_____
(повна назва)

_____ _Гнатушенко В.В._
(підпис) (ініціали та прізвище)

«_____» _____2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня бакалавра

(бакалавра, магістра)

здобувача вищої освіти_____Бурдіна А.Д._____**академічної групи**_____126-21-1_____
(прізвище та ініціали) (шифр)

спеціальності_____Інформаційні системи та технології_____

спеціалізації за освітньою-професійною програмою
_____«Інформаційні системи та технології»_____
(за наявності)

на тему ___Розробка веб-застосунку для моніторингу інвестиційного портфелю криптовалют зі зміною показників прибутковості у режимі реального часу_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від _05.05.2025_ № _336-с_

Розділ	Зміст	Термін виконання
Розділ 1. Аналіз стану області рішення завдання	Дослідження ринку криптовалютних портфельних трекерів. Аналіз функціональних можливостей існуючих рішень (CoinStats, DropsTab).	15.05.2025
Розділ 2. Проектні рішення	Обґрунтування вибору технологій (Flask, SQLite). Огляд методів розрахунку середньої ціни та прибутковості. Реалізація продукту на Python.	08.06.2025
Розділ 3. Тестування та результати роботи	Перевірка коректності розрахунків. Тестування інтерфейсу. Аналіз продуктивності системи. Підготовка рекомендацій щодо вдосконалення.	13.06.2025

Завдання видано _____ _____Сергєєва К.Л._____
(підпис керівника) (ініціали та прізвище)

Дата видачі _____

Дата подання до екзаменаційної комісії_____

Прийнято до виконання _____ _____Бурдін А.Д._____
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: __ с., __ рис., __ додатки, __ джерел.

API, PYTHON, SQLITE, ІНВЕСТИЦІЙНІ СТРАТЕГІЇ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС, КРИПТОВАЛЮТА, МОНІТОРИНГ ПОРТФЕЛІВ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РИНКОВІ ДАНІ, СЕРЕДОВИЩЕ РОЗРОБКИ, ТРАНЗАКЦІЇ, ФІНАНСОВІ ТЕХНОЛОГІЇ

Об'єкт кваліфікаційної роботи: криптовалютний інвестиційний портфель.

Предмет кваліфікаційної роботи: процеси, алгоритми та інструменти моніторингу стану криптовалютного портфеля інвестора.

Мета роботи: розробка програмного додатку для моніторингу інвестиційного портфелю криптовалют з можливістю локального збереження транзакцій, отримання актуальних ринкових даних та відображення фінансової аналітики.

У вступі висвітлено актуальність проблеми ефективного моніторингу криптовалютного портфелю з позиції індивідуального інвестора. Проведено аналіз характеристик криптовалют як інвестиційного інструменту, типів портфелів, стратегій управління активами (HODL, трейдинг, ребалансування), а також значущості своєчасного отримання ринкових змін.

У першому розділі наведено огляд популярних сервісів моніторингу криптоактивів (CoinStats, CoinMarketCap Portfolio, DropsTab). Проаналізовано їх функціональність, переваги й недоліки, що дозволило обґрунтувати необхідність створення власного адаптованого програмного рішення.

У другому розділі описано процес проєктування та реалізації програмного додатку з використанням Python, Tkinter, SQLite та API для отримання ринкових даних. Представлено архітектуру, логіку збереження транзакцій, механізми обчислення середньої ціни, прибутковості та інтерфейс користувача. Подано результати тестування та демонстрації функціоналу.

У висновках узагальнено результати розробки. Сформульовано переваги застосування створеного інструменту для моніторингу інвестицій.

Розроблений додаток надає приватним інвесторам зручний інструмент для локального моніторингу криптопортфеля, що дозволяє уникнути залежності від сторонніх сервісів і підвищити ефективність управління активами. Також воно може бути впроваджене у повсякденне використання для контролю й аналітики інвест-портфелю.

ABSTRACT

Explanatory note: __ p., __ figures, __ appendices, __ sources.

API, PYTHON, SQLITE, INVESTMENT STRATEGIES, USER INTERFACE, CRYPTOCURRENCY, PORTFOLIO MONITORING, SOFTWARE, MARKET DATA, DEVELOPMENT ENVIRONMENT, TRANSACTIONS, FINANCIAL TECHNOLOGIES

Object of qualification work: cryptocurrency investment portfolio.

Subject of qualification work: processes, algorithms and tools for monitoring the state of the investor's cryptocurrency portfolio.

Purpose: to develop a software application for monitoring a cryptocurrency investment portfolio with the ability to locally save transactions, obtain up-to-date market data, and display financial analytics.

The introduction highlights the relevance of the problem of effective monitoring of a cryptocurrency portfolio from the perspective of an individual investor. The author analyzes the characteristics of cryptocurrencies as an investment instrument, types of portfolios, asset management strategies (HODL, trading, rebalancing), and the importance of timely market changes.

The first section provides an overview of popular crypto asset monitoring services (CoinStats, CoinMarketCap Portfolio, DropsTab). Their functionality, advantages, and disadvantages are analyzed, which allowed us to justify the need to create our own adapted software solution.

The second section describes the process of designing and implementing a software application using Python, Tkinter, SQLite, and APIs to obtain market data. The architecture, logic of saving transactions, mechanisms for calculating the average price, profitability, and user interface are presented. The results of testing and demonstration of the functionality are presented.

The conclusions summarize the development results. The advantages of using the created tool for monitoring investments are formulated.

The developed application provides private investors with a convenient tool for local monitoring of the crypto portfolio, which allows them to avoid dependence

on third-party services and increase the efficiency of asset management. It can also be implemented in everyday use to control and analyze the investment portfolio.

ЗМІСТ

ВСТУП

Розвиток фінансових технологій та зростання популярності криптовалют як інвестиційного інструменту створюють нові виклики для індивідуальних інвесторів. Зокрема, виникає потреба у створенні ефективних засобів моніторингу стану криптовалютного портфеля, які б дозволяли оперативно приймати обґрунтовані інвестиційні рішення. Задачі, пов’язані з моніторингом цифрових активів, безпосередньо стосуються фахівців напряму “Інформаційні системи та технології”, адже включають аналіз даних, розробку клієнтських додатків та взаємодію з зовнішніми API.

На сьогоднішній день існує низка популярних сервісів для відстеження криптоактивів, таких як CoinStats, CoinMarketCap Portfolio, DropsTab та інші. Водночас більшість із них мають низку обмежень: залежність від інтернет-з'єднання, закритий код, обмежена функціональність або орієнтація виключно на мобільні платформи. Також користувачі не завжди мають можливість зберігати дані локально або змінювати логіку обробки транзакцій.

Актуальність теми полягає у необхідності створення зручного, автономного програмного рішення для обліку криптовалютних інвестицій із можливістю гнучкої інтеграції зовнішніх джерел даних. Таке програмне забезпечення має сприяти підвищенню ефективності управління портфелем, особливо в умовах динамічного ринку.

Метою кваліфікаційної роботи є розробка програмного додатку для моніторингу інвестиційного портфеля криптовалют з використанням мови Python, графічного інтерфейсу на базі Tkinter та бази даних SQLite. Програма повинна забезпечувати локальне збереження транзакцій, інтеграцію з API для отримання ринкових даних і відображення ключових фінансових показників.

***Основні задачі, що вирішуються в роботі**:*

- аналіз вимог до програм для моніторингу криптоактивів;
- розробка архітектури застосунку;
- реалізація інтерфейсу користувача та логіки обробки транзакцій;

- підключення до джерел ринкових даних через API;
- збереження та оновлення інформації в локальній базі даних;
- тестування, налагодження та оцінка ефективності роботи додатку;

Реалізація зазначених задач дозволить створити готовий до використання програмний продукт, що задовольняє потреби інвестора у зручному та ефективному моніторингу криптовалютного портфелю.

РОЗДІЛ 1
АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАВДАННЯ

У цьому розділі подано огляд теоретичних і практичних аспектів, пов’язаних із криптовалютами, інвестиційними портфелями, а також системами для моніторингу активів. Проаналізовано сучасні підходи до управління цифровими активами, визначено потребу в моніторингу інвестиційного портфеля та здійснено огляд популярних програмних рішень у цій галузі.

1.1. Поняття криптовалют і їх роль в інвестиціях

Упродовж останніх років криптовалюти стали невід’ємною частиною сучасної фінансової системи. Технологія блокчейн, що лежить в основі криптовалют, забезпечує прозорість, безпечність та децентралізацію. Такі цифрові активи, як Bitcoin, Ethereum та інші, здобули визнання не лише як засіб обміну, але й як об’єкт інвестування. Зростаючий інтерес до криптовалют пояснюється можливістю отримання високих прибутків. Водночас, висока волатильність робить ці інвестиції ризикованими. Тому ефективне управління такими активами вимагає відповідних знань і інструментів.

Криптовалюта - це форма цифрової валюти, створення та облік якої здійснюється автоматизованою децентралізованою платіжною системою без участі зовнішніх чи внутрішніх адміністраторів, включаючи державні установи. Основною відмінною рисою криптовалют є збереження даних у блокчейні - розподіленому реєстрі, де для підтвердження прав доступу застосовується асиметричне шифрування, а також використовуються інші криптографічні механізми, зокрема докази виконаної роботи (Proof-of-Work, POW), або підтвердження частки володіння (Proof-of-Stake, POS) [1].

Станом на початок 2025 року ринок криптовалют демонструє значне зростання як за кількістю активів, так і за їхньою капіталізацією. Згідно з даними CoinMarketCap, на січень 2025 року існувало понад 13 217 активних

криптовалют, а загальна ринкова капіталізація перевищила $2,54 трлн . При цьому лише близько 40 криптовалют мали ринкову капіталізацію понад $1 млрд. Ці дані свідчать про стрімке зростання кількості криптовалют, що створює нові можливості для інвесторів, але також підкреслює необхідність ретельного аналізу та моніторингу ринку для прийняття обґрунтованих інвестиційних рішень.

Криптовалюти не мають центрального органу управління. Усі операції перевіряються мережею учасників, тобто іншими користувачами. Кожна операція комплектується з іншими для формування блоку, з яких складається безперервний ланцюг. Кожен блок має криптографічне посилання на попередника, що робить неможливим зміну інформації в одному блоці без необхідності внесення змін в усі наступні. Тому підробити або скасувати запис неможливо або дуже витратно. Нижче представленний графік інтересу до використання криптовалютою (підвищення капіталізації ринку) за останні 5 років (рисунок 1.1).

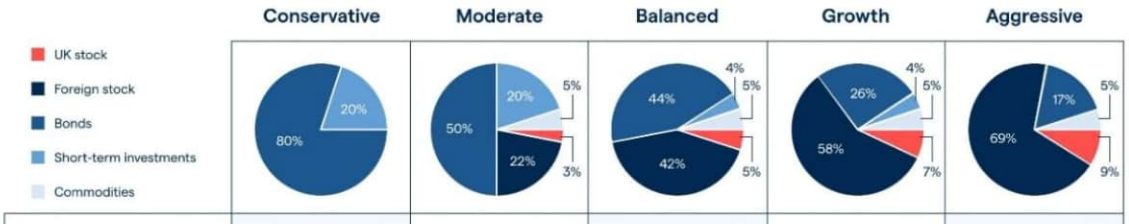
Рисунок 1.1 – Збільшення показника капіталізації[1]

Окрім основних монет, таких як Bitcoin або Ethereum, інвестори також розглядають альткоїни (альтернативні криптовалюти), стейблкоїни (валюти з

[1] Рисунок запозичений з сервісу coincodex.com. [2]

фіксованим курсом, як-от USDT або USDC), а також нові токени, які можуть швидко набрати популярності, але мають високі ризики. Інвестиції в криптовалюти здійснюються як через безпосереднє володіння активами (на гаманцях або біржах), так і опосередковано — через криптовалютні фонди, індекси, або токенізовані активи.

Особливу роль відіграє диверсифікація криптопортфеля, яка дозволяє зменшити ризики. Враховуючи волатильність ринку, інвестори активно використовують аналітичні інструменти для оцінки динаміки цін, об'ємів торгів, настроїв ринку та новинного фону. Саме тому ефективні інструменти моніторингу криптоактивів стають невід'ємною частиною стратегії управління портфелем. Нижче наведені різні варіації диверсифікації (аюо фінансових стратегій) та їх прибутковість на дистанції з різними частками, як приклад (рисунок 1.2).



	Conservative	Moderate	Balanced	Growth	Aggressive
Average annual return	1.4%	4.3%	5.3%	5.9%	6.1%
Annualised volatility	2.2%	5.4%	8.5%	11.0%	12.9%
Best 12-month return	4.6%	12.6%	17.7%	21.3%	24.6%
Worst 12-month return	-1.1%	-4.3%	-6.2%	-9.1%	-12.1%

Рисунок 1.2 – Порівняння різних варіантів долей при управлінні портфелем[2]

Загалом, криптовалюти сформували новий клас фінансових активів, який поєднує в собі інноваційність, потенціал високого доходу та високі ризики. Це вимагає від інвесторів глибоких знань, регулярного аналізу та

[2] Рисунок запозичений з ресурсу ig.com. [3]

використання спеціалізованих програмних засобів для обліку і контролю за своїм портфелем.

1.2. Типи криптопортфелів і стратегій управління

Ринок криптовалют характеризується високою динамікою, що потребує від інвестора не лише глибокого розуміння поточних ринкових умов, а й здатності адаптувати свою стратегію у реальному часі. В умовах нестабільності та високої волатильності, ефективне управління криптовалютним портфелем набуває ключового значення.

Умовно криптопортфелі можна класифікувати за різними ознаками:

1. За типом активів:

   - **Монопортфель** - містить один актив, найчастіше це Bitcoin або Ethereum. Основна перевага - простота в управлінні, однак високий ризик через відсутність диверсифікації.

   - **Диверсифікований портфель -** включає кілька криптовалют із різним рівнем капіталізації та ризику. Дає змогу збалансувати прибутковість і ризики.

2. За метою інвестування:

   - **Спекулятивний** - орієнтований на короткострокові коливання цін. Такий портфель потребує частого моніторингу та миттєвих рішень.

   - **Інвестиційний (довгостроковий)** - спрямований на збереження активів протягом місяців або років. Зазвичай включає проєкти з сильними фундаментальними показниками.

3. За рівнем ризику:

   - **Консервативний** - містить переважно стейблкоїни (наприклад, USDT, USDC(є найбільш використовуваним)), а також топ-10

монет. Призначений для збереження капіталу з мінімальним ризиком.

- ***Збалансований*** - поєднання відносно стабільних активів із часткою високо ризикових (наприклад, DeFi або мем-коїнів).
- ***Агресивний*** - побудований із високоволатильних активів з метою отримання великого прибутку, супроводжується значним ризиком втрат.

Стратегії управління криптопортфелем

Успішне управління криптовалютним портфелем передбачає застосування чіткої стратегії, що враховує як ринкові умови, так і особисті інвестиційні цілі. Основні стратегії включають:

- **HODL (Hold On for Dear Life)** - довгострокове зберігання активів незалежно від поточної ринкової ситуації. В основі – віра в зростання вартості криптовалют у майбутньому.
- **Ребалансування** - періодичне коригування співвідношення активів у портфелі для підтримання початкової стратегії. Наприклад, якщо один актив сильно виріс у ціні, його частка може бути зменшена для зниження ризиків.
- **Dollar-Cost Averaging (DCA)** - регулярне інвестування фіксованої суми незалежно від поточної ринкової ціни. Стратегія дозволяє уникнути спроб «зловити дно» і зменшує вплив волатильності.
- **Технічний аналіз** - застосування графіків, індикаторів і патернів для прийняття рішень про купівлю або продаж. Часто використовується трейдерами при спекулятивних підходах.
- **Фундаментальний аналіз** - оцінка перспективності активу на основі команди розробників, мети проєкту, обсягів використання, партнерств тощо. Дає змогу приймати обґрунтовані рішення на довгостроковому горизонті.

Роль моніторингу в управлінні портфелем

Моніторинг портфеля - це систематичний процес відстеження стану інвестицій, який дозволяє:

- відстежувати прибутковість активів;
- приймати обґрунтовані рішення щодо купівлі або продажу;
- оперативно реагувати на зміни ринку (наприклад, падіння ціни, новини, оновлення протоколів);
- знижувати вплив емоційних факторів на інвестора;

У сучасних умовах існує велика кількість інструментів для моніторингу: мобільні додатки, десктоп-програми, торгові платформи з вбудованим трекером портфеля тощо.

1.3. Існуючі рішення для моніторингу криптопортфеля

Станом на сьогодні криптовалютний ринок продовжує активно розвиватися, що створює потребу в ефективних інструментах моніторингу інвестицій. З метою забезпечення контролю за своїми активами, користувачі все частіше звертаються до спеціалізованих сервісів, які дозволяють у зручному інтерфейсі відстежувати зміну вартості активів, прибутковість, структуру портфеля, а також синхронізувати дані з біржами, гаманцями або вводити їх вручну.

До найпоширеніших рішень належать такі платформи:

1. CoinStats

CoinStats (рисунок 1.3) - один із найпопулярніших додатків для моніторингу криптовалютних портфелів, що підтримує понад 8000 криптовалют і синхронізацію з більш ніж 300 біржами та гаманцями. Основні функції включають:

- автоматичне оновлення балансу через API бірж;

- аналітика прибутків/збитків;
- підтримка DeFi-гаманців;
- повідомлення про зміни цін;

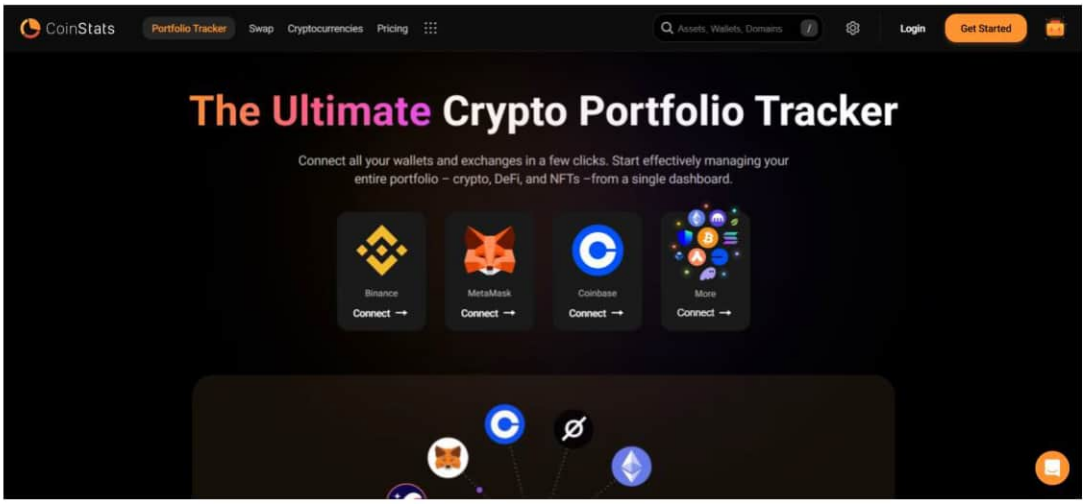


Рисунок 1.3 – Головная сторінка CoinStats [4]

Переваги:

- широкий вибір монет і платформ;
- зручний мобільний та веб-інтерфейс;
- наявність темної теми, графіків, історії змін;

Недоліки:

- обмежений функціонал у безкоштовній версії (наприклад, обмежена кількість підключень до акаунтів);
- потреба в інтернет-з’єднанні;
- прив’язка до централізованих акаунтів (може викликати занепокоєння щодо конфіденційності);

2. CoinMarketCap Portfolio

CoinMarketCap (рисунок 1.4) — це простий та зручний інструмент для базового відстеження криптоінвестицій, який інтегрований у екосистему одного з найпопулярніших агрегаторів ринкових даних. Він ідеально

підходить для початківців, оскільки дозволяє швидко додавати активи вручну (без API чи підключення гаманців) та відстежувати їхню вартість у контексті загального ринку. Однак, на відміну від спеціалізованих трекерів, функціонал обмежений відсутністю розширеної аналітики, автоматичного оновлення балансів чи інтеграції з DeFi-протоколами.

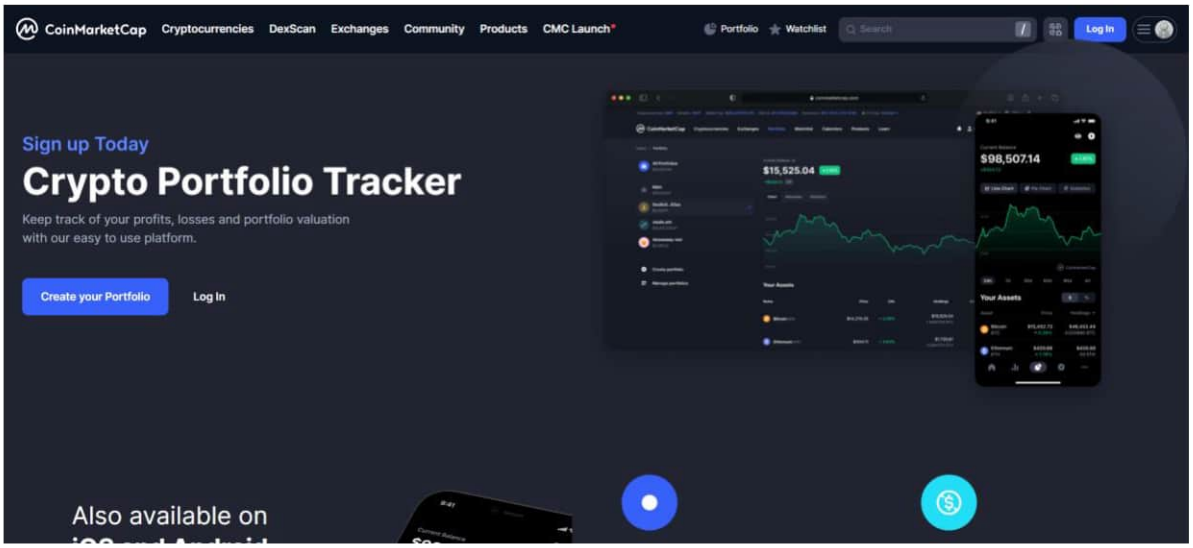


Рисунок 1.4 – Сторінка створення інвест-портфелю у CMC [5]

Переваги:

- повністю безкоштовний;
- проста інтеграція в екосистему CoinMarketCap;
- можливість швидко переглядати зміни вартості активів прямо на головній сторінці;

Недоліки:

- відсутність автоматичної синхронізації;
- немає розширеної аналітики;
- орієнтований більше на новачків;

3. DropsTab

DropsTab (рисунок 1.5) — це сучасний веб-сервіс для відстеження криптовалютних інвестицій, який спеціалізується на DeFi (децентралізованих фінансах), токенах та агрегованій аналітиці ринку. Він швидко набуває популярності серед інвесторів завдяки інтуїтивному інтерфейсу, потужним інструментам аналітики та підтримці Web3-гаманців. Має широку підтримку різних токенів та самих популярних гаманців, та гарно налаштовані інтерактивні графіки цін із зручною функцією порівняння активів.

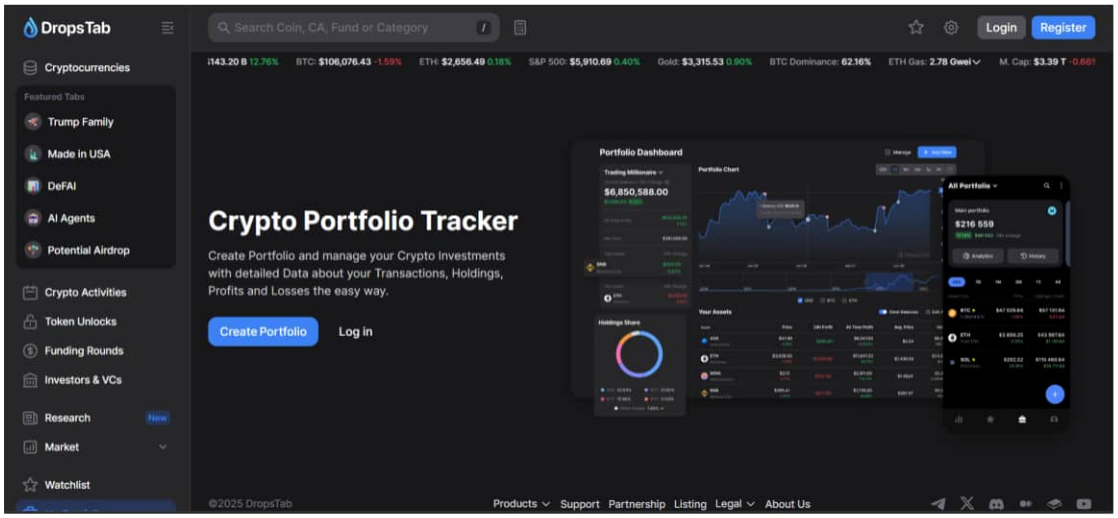


Рисунок 1.5 – Зовнішній вигляд DropsTab [6]

Переваги:

- підтримка Web3-гаманців;
- можливість відстеження нових токенів;
- розширена статистика по кожному активу (обсяги, капіталізація, зміна за добу, тиждень тощо);

Недоліки:

- частина функцій доступна лише через авторизацію;
- не підтримує офлайн-режим;

1.4. Вибір мови програмування та середовища

Для розробки як основу функціоналу було обрано мову программування Python (рисунок 1.6), так як вона є універсальною для багатьох задач, також активно використовується у сфері фінансів, обробки даних та створення десктоп-додатків. Серед її переваг: простий синтаксис, велика кількість готових бібліотек (наприклад, requests, tkinter, pybit для нашого випадку) та активна спільнота, що дозволяє швидко знаходити рішення під час розробки.

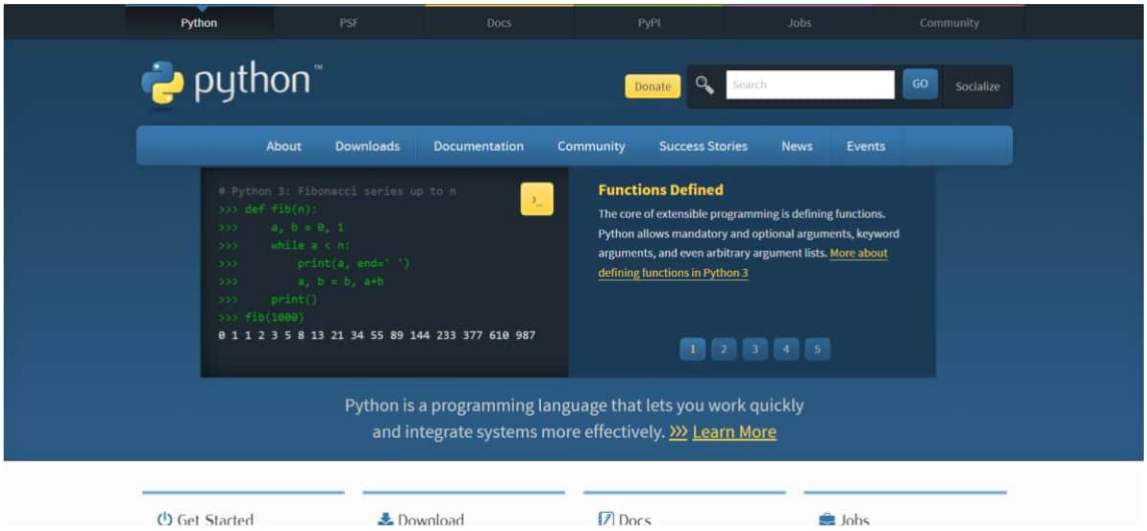


Рисунок 1.6 – Головна сторінка для скачування інтерпретатора Python [7]

Як середовище розробки у нашій роботі будемо використовувати Visual Studio Code (рисунок 1.7), так як воно досить легке (мінімальна версія великого родича VisualStudio (рисунок 1.8)), але це вже більш потужне середовище з відкритим кодом, яке підтримує досить велику кількість розширень для Python. Для нашого випадку - створення прототипу, перший варіант є більш зручним.

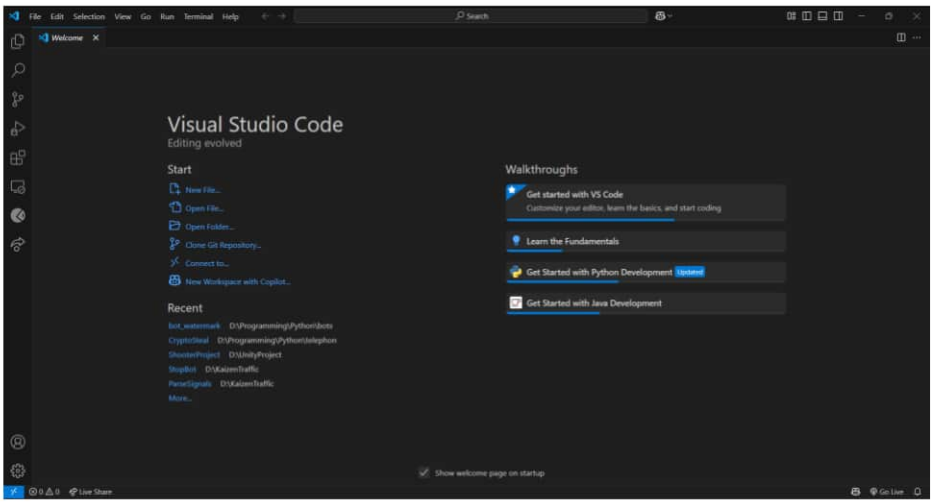


Рисунок 1.7 – Головний екран VSCode [8]

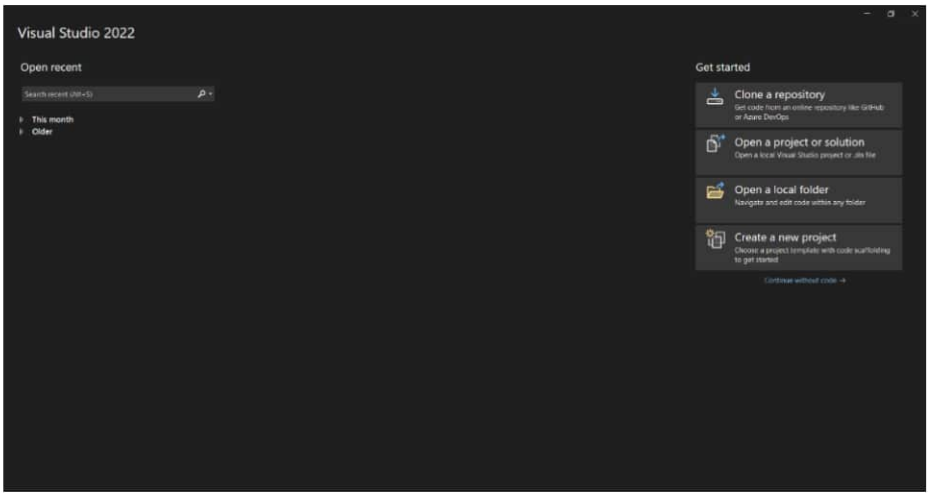


Рисунок 1.8 – Екран вибору або створення проектів у VisualStudio

Поєднання Python і Visual Studio Code створює комфортне середовище для розробника: швидке налаштування, гнучка структура коду і можливість миттєвого тестування змін дозволяють зосередитись безпосередньо на логіці додатку. Тож як підсумок для розробки програмного забезпечення було обрано мову Python та середовище Visual Studio Code.

1.5. Висновки

У результаті проведеного аналізу області вивчення та розробки рішення для завдання можна зробити висновок, що криптовалюти відіграють значну роль в сучасному інвестиційному середовищі. Управління криптоактивами

вимагає використання спеціалізованих програмних засобів, які здатні забезпечити ефективний моніторинг та аналіз портфеля. Огляд існуючих рішень показав, що хоча на ринку присутні функціональні інструменти, все ще існує потреба у створенні автономного, гнучкого та локального додатку з широкими можливостями персоналізації, яких не мають рішення на ринку.

РОЗДІЛ 2
ПРОЄКТНІ РІШЕННЯ

Під час створення інформаційної системи для моніторингу криптовалютного портфеля важливо прийняти низку технічних рішень, які визначатимуть ефективність, масштабованість та зручність у використанні майбутнього програмного забезпечення. У цьому розділі розглянуто ключові компоненти, що формують архітектуру системи: вибір бази даних, API для отримання ринкових даних, інструменти створення користувацького інтерфейсу, а також функціональні вимоги та принципи побудови структури додатку. Усі рішення обґрунтовано з урахуванням вимог до сучасних фінансово-аналітичних застосунків та специфіки роботи з криптовалютами.

2.1. Бази даних для зберігання транзакцій

SQLite

SQLite — це легковагова реляційна база даних, яка не вимагає налаштування сервера. Вона є вбудованою та зберігається у вигляді одного файлу на диску. Основними перевагами SQLite [9] є простота інтеграції, відсутність потреби в адмініструванні та висока швидкість роботи з невеликими обсягами даних.

Переваги SQLite:

- невеликих десктопних додатків;
- прототипів або MVP (мінімально життєздатних продуктів);
- програм з одним користувачем;
- локального зберігання даних без потреби в мережевій синхронізації;

Певні обмеження SQLite:

- не підтримує одночасну роботу великої кількості клієнтів;

- не призначена для обробки великого навантаження;
- обмежена підтримка складних транзакцій;

PostgreSQL

PostgreSQL — потужна об'єктно-реляційна система управління базами даних з відкритим кодом. Вона підтримує розширені можливості SQL, складні транзакції, масштабування та реплікацію. PostgreSQL [10] вважається стабільним, безпечним і придатним для роботи у багатокористувацьких і високонавантажених системах.

Переваги PostgreSQL:

- висока продуктивність при роботі з великими обсягами даних;
- підтримка одночасного доступу з багатьох клієнтів;
- потужна система індексації й оптимізації запитів;
- підтримка збережених процедур, тригерів, JSON-документів;

Недоліки PostgreSQL:

- складніше налаштування у порівнянні з SQLite;
- потреба в адмініструванні та супроводі бази даних;
- вищі вимоги до ресурсів системи;

Нижче представлені відносні порівняння вищепредставленних варінтів рішення для баз даних на основі яких можна було б обрати саме потрібну для нашої задачі (таблиця 2.1).

Таблиця 2.1 – Порівняння баз даних

Характеристика	**SQLite**	**PostgreSQL**
Тип	Вбудована БД	Серверна СУБД

Масштабованість	Низька	Висока
Паралельність	Обмежена	Підтримується повноцінно
Налаштування	Не потребує	Потребує
Придатність	Локальні, малі проекти	Веб-сервіси, складні системи
Продуктивність	Висока на малих об’ємах	Висока при великих об’ємах

Вибір для проєкту

У контексті інформаційної системи для моніторингу криптовалютного портфеля вибір бази даних залежить від цільової платформи й масштабу системи. Якщо планується створення десктопного застосунку з обробкою даних лише на рівні одного користувача — доцільно використати SQLite через простоту, мінімальні ресурси та достатню швидкість.

У разі вебзастосунку, особливо з підтримкою декількох користувачів, підключенням до API, регулярним оновленням даних та необхідністю аналітики — доцільніше обрати PostgreSQL. Така система дозволить масштабувати проєкт, забезпечити надійне збереження даних та обробку складних запитів та використання даних у майбутньому.

Інші (незвичайні) альтернативи баз даних

Окрім SQLite і PostgreSQL, існують також інші варіанти, які можна розглядати залежно від специфіки проєкту:

- MySQL/MariaDB - популярні СУБД з широкою підтримкою, аналогічні PostgreSQL за функціональністю;

- MongoDB - документоорієнтована NoSQL-база для зберігання неструктурованих даних, зручно для гнучких моделей;
- Firebase Realtime Database - хмарне рішення для реального часу, підходить для мобільних застосунків;

В кінці кінців маємо розуміння, що для розробки криптофінансового застосунку доцільніше розпочинати з SQLite для швидкого прототипування або локального зберігання. У майбутньому, при збільшенні навантаження або потребі в багатокористувацькій роботі, рекомендовано перейти на PostgreSQL або іншу серверну СУБД, що підтримує високий рівень надійності, безпеки та масштабування та зберігання даних.

2.2. API для отримання ринкових даних

Однією з ключових складових інформаційної системи для моніторингу криптовалют є отримання актуальних ринкових даних. Без цього неможливо в режимі реального часу оцінювати вартість активів, динаміку портфеля та прибуток чи збиток від інвестицій. У цьому контексті API (інтерфейси прикладного програмування) стають невід'ємною частиною функціональності системи.

API надають розробникам можливість автоматично отримувати інформацію з бірж або агрегаторів даних, таких як CoinGecko, CoinMarketCap, Bybit, Binance та багато інших. Ці сервіси повертають ринкові дані у форматі JSON і дозволяють виконувати запити до їхньої інфраструктури для отримання цін, історичних графіків, об'ємів торгів, ринкової капіталізації тощо.

Основні типи даних, які потрібні для системи:

- Поточна ціна криптовалюти;
- Історія цін (для побудови графіків);

- Об’єм торгів за останні 24 години;
- Ринкова капіталізація;
- Дані про конкретні біржі (спреди, глибина ринку);
- Інформація про токени (назва, тикер, адреса контракту тощо);

Огляд популярних API

CoinGecko API

CoinGecko - один із найпопулярніших безкоштовних API для отримання криптоданих. Він надає широкий набір інформації без необхідності авторизації за допомогою API-ключа [11].

Основні особливості:

- Безкоштовне використання до 100 запитів/хв;
- Понад 13 000 монет і токенів;
- Дані з понад 700 бірж;
- Підтримка ринкової капіталізації, історії цін, індикаторів;

Приклад застосування:

Запит на отримання поточної ціни Bitcoin:

- https://api.binance.com/api/v3/ticker/price?symbol=BTCUSDT;

Переваги:

- Відсутність API-ключа;
- Надійність та актуальність;
- Підтримка великої кількості валют;

Недоліки:

- Немає даних про облікові записи або приватні транзакції;
- Ліміти на кількість запитів у безкоштовній версії;

- Не завжди найвища точність (через агрегацію з багатьох бірж);

CoinMarketCap API

CoinMarketCap - один із найстаріших агрегаторів криптовалютної інформації. Його API вимагає реєстрації та використання API-ключа [12].

Можливості:

- Доступ до цін, об’ємів, ринків;
- Підтримка фіатних валют;
- Висока точність і частота оновлення;

Приклад запиту:

- https://pro-api.coinmarketcap.com/v1/cryptocurrency/listings/latest (вимагає заголовок з API-ключем);

Переваги:

- Широка підтримка токенів;
- Детальна інформація;
- Висока надійність;

Недоліки:

- Необхідний API-ключ;
- Обмеження у безкоштовному тарифі (333 запити/добу);
- Комерційна модель - деякі функції платні;

Bybit API

Bybit - криптобіржа, яка надає API як для публічних, так і приватних запитів [13]. Це особливо корисно, якщо користувачі мають акаунти на біржі і бажають інтегрувати дані про свої портфелі напряму.

Типи API:

- Public API — ціни, об'єми, книги ордерів;
- Private API — баланси, історія транзакцій, ордери;

Переваги:

- Дані в реальному часі;
- Підтримка WebSocket для миттєвих оновлень;
- Докладна документація;

Недоліки:

- Потрібна реєстрація та підписання запитів (для приватних методів);
- Обмеження на кількість запитів у хвилину;

Інші варіації API

- Binance API — для отримання ринкових даних та роботи з акаунтами на Binance [14];
- CryptoCompare — хороша візуалізація історичних даних;
- Messari API — інституційна аналітика та фундаментальні метрики;
- Pump.fun / Jupiter API — для роботи з токенами на Solana;

Щоб оцінити потрібний нам варіант для реалізаці отримання ринкових даних, складемо порівняльну таблицю з параметрами «Потрібен ключ», «Наявність історії цін», «Дані біржі», «Загальна підтримка токенів» та «Реальний час», а «Приватні дані» уточнюються в залежності від повноти розробки (таблиця 2.2).

Таблиця 2.2 – Порівняння агрегаторів даних (API)

Параметр	**CoinGecko**	**CoinMarketCap (CMC)**	**Bybit**	**Binance**
Потрібен API-ключ	Ні	Так	Так	Так
Історія цін	Так	Так	Частково	Так
Дані біржі	Так	Так	Так	Так
Підтримка токенів	13 000+	9 000+	Обмежено	Обмежено
Реальний час	Так	Так	Так (WebSocket)	Так (WebSocket)
Приватні дані	Ні	Ні	Так	Так

Вибір API для системи

При виборі API для розробки системи моніторингу криптоактивів слід враховувати такі фактори:

- Тип даних, які необхідно отримувати — для публічної інформації вистачить Binance або CoinMarketCap;
- Частота оновлення — для реального часу краще використовувати WebSocket API (Bybit, Binance);
- Простота інтеграції — CoinGecko має найнижчий поріг входу;
- Масштаб проєкту — для розширених можливостей краще підійдуть платні версії API з високими лімітами;

Для MVP (minimal version product) або десктопного додатку з базовим функціоналом буде цілком достатньо BinanceAPI, бо саме сам ми будемо мати найбільш правильне значення цін. У разі розширення системи й інтеграції з

обліковими записами користувачів — доцільно підключати API (CoinMarketCap, CoinGecko).

Тож інтеграція API є обов'язковою складовою сучасних фінансових систем. Для системи моніторингу криптопортфеля, залежно від функціональності та архітектури, можна обрати як простий агрегатор (BinanceAPI), для простого розширеного використання у інформаційній системі. Важливо мати баланс між простотою реалізації, надійністю даних та можливістю масштабування в майбутньому.

2.3. Фреймворки для створення інтерфейсу користувача

У процесі розробки програмного забезпечення значну увагу приділяють інтерфейсу користувача, оскільки саме він забезпечує взаємодію користувача із системою. Веб-інтерфейси мають ряд переваг: доступність із будь-якого пристрою через браузер, відсутність потреби в установці, гнучкість у дизайні та можливість масштабування. Для створення таких інтерфейсів використовуються веб-фреймворки, зокрема Flask, Django та React. Нижче розглянуто особливості кожного з них.

Flask

Flask — це мікрофреймворк на мові Python [15], орієнтований на простоту та розширюваність. Його основною перевагою є мінімалістичний підхід: замість готової архітектури розробник самостійно визначає структуру застосунку. Його мінімалістичний підхід дозволив би зосередитись на логіці додатку, а не на конфігурації фреймворку, що значно би прискорило розробку.

Переваги:

- Швидке створення прототипів;
- Проста інтеграція з HTML/CSS/JS;
- Гнучкість — немає жорстких обмежень;

- Велика кількість плагінів та бібліотек;
- Добре підходить для невеликих та середніх проєктів;

Недоліки:

- Відсутність вбудованої ORM (ORM потрібно підключати окремо, наприклад SQLAlchemy);
- Немає стандартної адміністративної панелі;
- Потреба реалізації багатьох функцій вручну (авторизація, валідація тощо);

Django

Django — це повноцінний фреймворк для розробки веб-додатків на Python, який працює за принципом «все включено» [16]. Він надає розробнику готові інструменти: ORM, систему маршрутизації, шаблони, панель адміністратора, механізми аутентифікації та ін.

Переваги:

- Висока швидкість розробки за рахунок великої кількості готових модулів;
- Вбудована ORM для зручної роботи з базами даних;
- Зручна адміністративна панель "із коробки";
- Високий рівень безпеки (захист від CSRF, SQL-ін’єкцій тощо);
- Добре підходить для складних систем з багатьма модулями;

Недоліки:

- Вищий поріг входу для новачків;
- Менша гнучкість у порівнянні з Flask;
- Може бути "зайвим" для невеликих проєктів;

React

React дзволяє створювати компонентні, динамічні, інтерактивні веб-інтерфейси [17]. Часто використовується у поєднанні з бекенд-фреймворками (як-от Flask чи Django), створюючи архітектуру типу "frontend-backend separation". Але так як він більш орієнтований на складні динамічні інтерфейси з великою кількістю інтерактивних елементів, тоді як наш додаток потребував максимально простої та легкої реалізації, використання React у такому випадку призвело б до зайвого ускладнення архітектури, необхідності додаткового налаштування та збільшення часу розробки без суттєвих переваг для функціоналу.

Переваги:

- Висока продуктивність завдяки використанню Virtual DOM;
- Повна інтерактивність інтерфейсу;
- Компонентна структура спрощує підтримку коду;
- Потужна екосистема (Redux, React Router, Hooks тощо);
- Підходить для створення SPA (Single Page Applications);

Недоліки:

- Потреба у знанні JavaScript, а також JSX;
- Складність конфігурації проєкту для початківців;
- Наявність додаткового шару логіки між фронтендом і бекендом;

Нижче представлена ще одна порівняльна таблиця функціональності фреймворків для більше структурованої оцінки вибору (таблиця 2.3).

Таблиця 2.3 – Порівняльна таблиця фреймворків

Характеристика	**Flask**	**Django**	**React**
Тип	Бекенд	Бекенд	Фронтенд

Характеристика	Flask	Django	React
Мова програмування	Python	Python	JavaScript
Швидкість розробки	Висока	Висока	Середня (залежить від складності)
Структура	Гнучка	Жорстка/структурована	Компонентна
Поріг входу	Низька	Середня	Середня/висока
Масштабованість	Середня	Висока	Висока
Підходить для	API, невеликі сайти	Складні системи	Динамічні інтерфейси

При виборі фреймворку для розробки веб-інтерфейсу слід враховувати складність застосунку, обсяг функціональності та вимоги до дизайну. Якщо мета — створити простий веб-додаток або REST API, доцільно використати Flask, який дозволяє швидко реалізувати базову логіку. Для побудови повноцінних веб-платформ, які мають складну бізнес-логіку, зручну роботу з БД, авторизацію та адміністративну панель, кращим вибором буде Django, так як маштабованість та деталізація розробки продуктів на ньому значно більша [18].

У випадку, якщо потрібен сучасний, динамічний інтерфейс з високою інтерактивністю (наприклад, при великій кількості користувацьких подій або даних, що постійно оновлюються), варто було б поєднувати бекенд (Flask/Django) із React на фронтенді. Такий підхід дозволяє ефективно розділити логіку клієнтської та серверної частин.

2.4. Вимоги до інформаційної системи

Інформаційна система для моніторингу криптовалютного портфеля має задовольняти сучасні потреби інвесторів, забезпечуючи точність даних, зручність використання та безпеку. Вона повинна надавати інструменти для:

- Ведення обліку активів і транзакцій;
- Автоматичного розрахунку прибутків/збитків;
- Отримання актуальних ринкових даних;
- Генерації звітів та аналітики;

Система розробляється з урахуванням функціональних та нефункціональних вимог, що гарантує її стабільність, продуктивність і масштабованість.

2.4.1 Функціональні вимоги:

Управління активами

- Додавання нових активів (криптовалют) до портфеля;
- Видалення активів з портфеля;
- Редагування існуючих активів (коригування кількості, ціни);
- Відображення списку всіх активів з детальною інформацією:
    - Поточна вартість;
    - Середня ціна придбання;
    - Кількість одиниць;
    - Відсоткова частка в портфелі;

Облік транзакцій

- Фіксація операцій купівлі/продажу;
- Ведення історії всіх транзакцій;
- Можливість видалення транзакцій;

- Автоматичний перерахунок середньої ціни при додаванні нових транзакцій;

Розрахунок прибутків/збитків

- Відображення:
  - Загального прибутку/збитку портфеля;
  - Прибутку/збитку по кожному активу;
  - Відсоткової рентабельності інвестицій;
- Розділення на:
  - Реалізований прибуток (від продажів);
  - Нереалізований прибуток (від зміни цін);

Отримання ринкових даних

- Автоматичне оновлення цін криптовалют;
- Відображення історичної динаміки цін;
- Інтеграція з публічними API (Binance, CoinGecko);

Звітність

- Генерація звітів у форматі CSV;
- Візуалізація даних у вигляді графіків та діаграм;
- Можливість експорту даних портфеля;

2.4.2. Нефункціональні вимоги

Продуктивність

- Час відгуку інтерфейсу: ≤ 1 секунди;
- Частота оновлення ринкових даних: кожні 3-5 хвилин;
- Обробка до 10 000 транзакцій без суттєвого падіння продуктивності;

Надійність

- Автоматичне збереження даних при закритті програми;
- Механізм відновлення після збоїв;
- Обробка помилок при роботі з API;

Безпека

- Локальне зберігання даних (без використання хмарних сховищ);
- Валідація вхідних даних;
- Захист від SQL-ін'єкцій;

Зручність використання

- Інтуїтивно зрозумілий інтерфейс;
- Адаптивний дизайн (підтримка мобільних пристроїв);
- Темна/світла тема інтерфейсу;
- Підказки та сповіщення;

Масштабованість

- Можливість розширення функціоналу (додавання нових типів активів);
- Підтримка додаткових API для отримання даних;
- Гнучкі налаштування інтерфейсу;

2.4.3. Інтерфейс користувача

Головне вікно

- Панель загальної статистики портфеля;
- Таблиця активів з ключовими показниками;
- Кнопки швидкого доступу до основних функцій;

Вікно транзакцій

- Форма для додавання нових транзакцій;

- Історія операцій з фільтрами;
- Можливість редагування/видалення транзакцій;

Вікно аналітики

- Графіки зміни вартості портфеля;
- Діаграми розподілу активів;
- Детальна статистика по кожному активу;

Налаштування

- Вибір валюти відображення;
- Налаштування частоти оновлення даних;
- Керування темою інтерфейсу;

Ці вимоги цілком відповідають потребам користувачів і забезпечують ефективну роботу системи моніторингу інвестиційного портфелю.

2.5. Архітектура додатку

Сучасний додаток для моніторингу криптовалютного портфеля потребує продуманої архітектури, яка забезпечує стабільність, масштабованість та зручність використання. У цьому розділі розглядається структура системи, що включає:

- Клієнт-серверну модель для ефективної взаємодії між користувачем та бекендом.
- Монолітну архітектуру з модульним підходом, що дозволяє легко розширювати функціонал.
- Механізми збереження даних (локальна база SQLite) для забезпечення автономності та безпеки.

Архітектура побудована з урахуванням сучасних стандартів розробки, що забезпечує швидку обробку запитів, точність розрахунків та зручний інтерфейс для користувача.

2.5.1. Загальна схема роботи (Клієнт-Сервер)

Система працює за класичною клієнт-серверною моделлю:

Клієнтська частина:

- Веб-інтерфейс на HTML/CSS/JS (рендериться у браузері);
- Відправляє запити до бекенду через REST API (наприклад, додавання транзакцій);

Серверна частина:

- Backend: Flask (Python), обробляє логіку:
  - Розрахунки середньої ціни, прибутків;
  - Інтеграція з Binance API для оновлення цін;
  - Робота з базою даних;
- База даних: SQLite (локальне сховище) (коротка схема бази (рисунок 2.1));

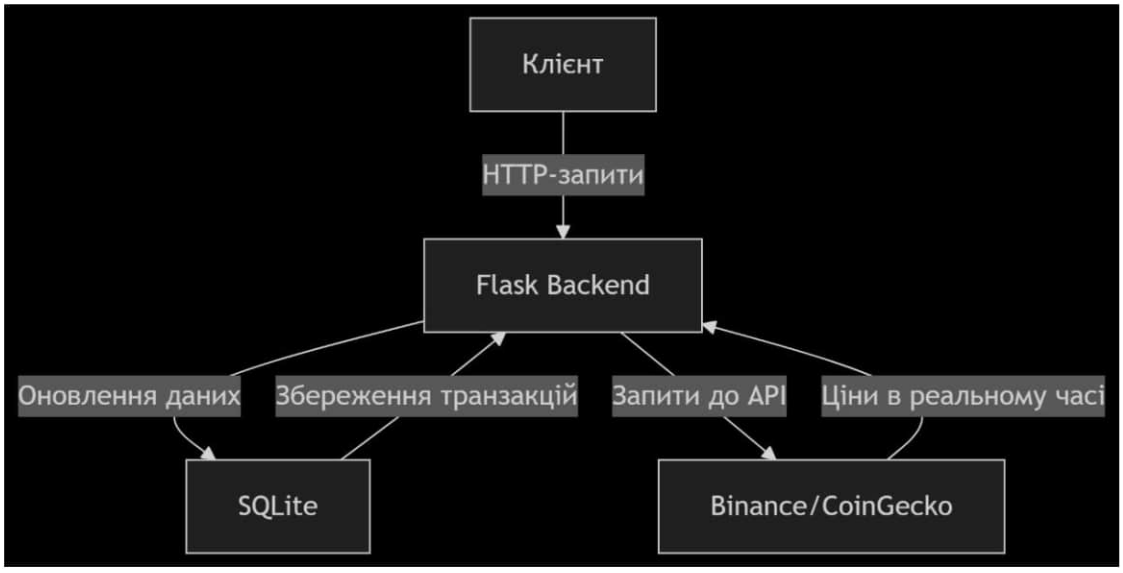


Рисунок 2.1 – Схема роботи Клієнт-Сервер

2.5.2. Тип архітектури (Моноліт з модулями)

Додаток використовує монолітну архітектуру з чітким розділенням на модулі:

Модуль транзакцій:

- Додавання/видалення операцій купівлі/продажу;
- Перерахунок середньої ціни (/add);

Модуль аналітики:

- Розрахунок прибутків (реалізованих/нереалізованих);

Модуль API-інтеграції:

- Оновлення цін криптовалют (Binance API);
- Кешування даних для зменшення запитів (@lru_cache);

Переваги:

- Простота розробки та розгортання;
- Легка підтримка через єдину кодобаз;

2.5.3. Збереження даних

Локальна база даних (SQLite):

- Таблиці: investments (активи), transactions (історія операцій);
- Як працює:
    - При купівлі: оновлюється investments (середня ціна перераховується);
    - При продажу: зменшується кількість, але середня ціна не змінюється;

Резервне копіювання:

- Експорт даних у CSV (через Flask-ендпоїнти);

2.5.4. Схема взаємодії компонентів

Нижче описана коротка схера зв’язків компонентів додатку (рисунок 2.2).

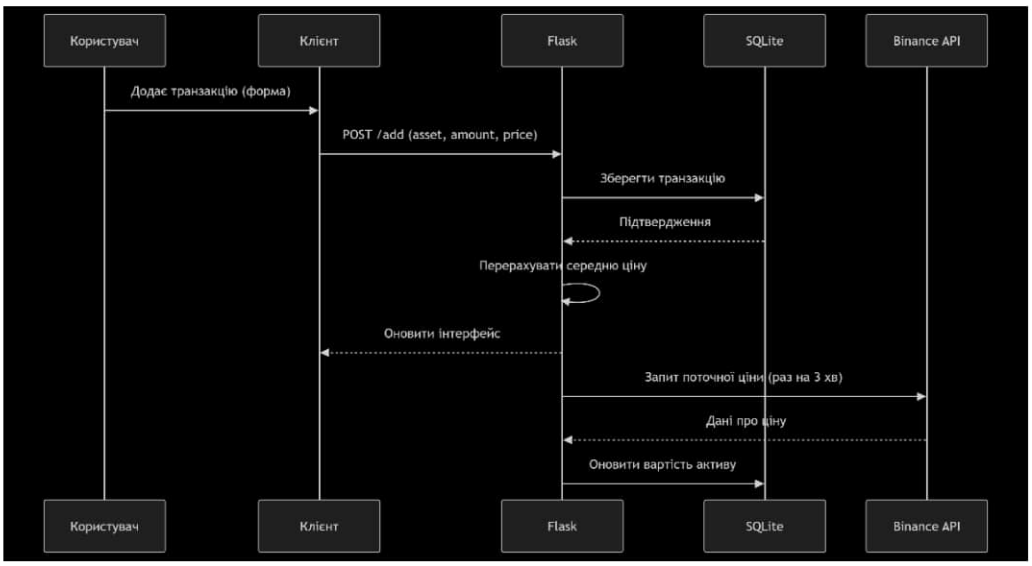


Рисунок 2.2 – Схема компонентів додатку

2.5.5. Висновок

- Архітектура: Клієнт-серверна, моноліт з модульною структурою.
- Ключові переваги:
    - Простота (немає складної інфраструктури);
    - Автономність (SQLite не вимагає окремого сервера);
    - Гнучкість (легко додати нові API або аналітичні функції);

- Для масштабування:
    - Можливий перехід на PostgreSQL;
    - Додаток легко перенести на сервер (наприклад, через Docker);

Ця архітектура ідеально підходить для локального використання з можливістю розширення.

2.6 Структура бази даних

У будь-якій інформаційній системі структура бази даних відіграє ключову роль у забезпеченні ефективного зберігання та обробки даних. ER-діаграма (Entity-Relationship diagram) візуалізує зв'язки між сутностями (таблицями) та їх атрибутами (полями), що дозволяє чітко зрозуміти архітектуру даних. У вашому додатку для моніторингу криптопортфеля використовуються дві основні таблиці: investments (активи) та transactions (транзакції). Далі наведено їх детальний опис та зв’язки (рисунок 2.3).

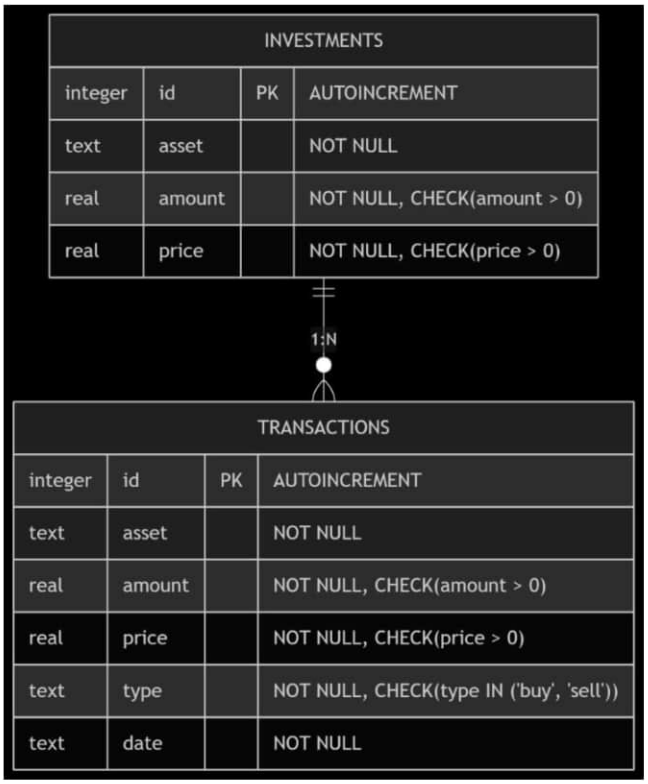


Рисунок 2.3 – Схема зв’язків таблиць

2.6.1 Опис ER-діаграми

Складемо короткий опис таблиць у базі з описами полів, типами даних та іншим (таблиця 2.4).

Таблиця 2.4 – Таблиці бази даних та їх поля

Таблиця	Поле	Тип даних	Обмеження	Опис
investments	id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор
	asset	TEXT	NOT NULL	Код криптовалюти (BTC)
	amount	REAL	NOT NULL, CHECK(amount > 0)	Кількість одиниць
	price	REAL	NOT NULL, CHECK(price > 0)	Середня ціна придбання
transactions	id	INTEGER	PRIMARY KEY, AUTOINCREMENT	Унікальний ідентифікатор
	asset	TEXT	NOT NULL	Код криптовалюти
	amount	REAL	NOT NULL, CHECK(amount > 0)	Обсяг операції
	price	REAL	NOT NULL, CHECK(price > 0)	Ціна за одиницю

Таблиця	Поле	Тип даних	Обмеження	Опис
	type	TEXT	NOT NULL, CHECK(type IN ('buy', 'sell'))	Тип операції
	date	TEXT	NOT NULL	Дата транзакції

2.6.2 Зв’язок між таблицями

- Тип зв’язку: Один-до-багатьох (1:N); Один запис у таблиці investments (наприклад, BTC) може мати багато пов’язаних транзакцій у transactions (купівля/продаж).
- Ключ зв’язку: Поле asset (назва криптовалюти);

2.6.3 Приклад роботи

Купівля BTC:

- Додається запис у transactions (type='buy');
- Оновлюється investments: збільшується amount, перераховується price;

Продаж BTC:

- Додається запис у transactions (type='sell');
- Оновлюється investments: зменшується amount (середня ціна не змінюється);

2.6.4 Висновок

ER-діаграма вашого додатку відображає просту, але ефективну структуру бази даних, оптимізовану для:

- Швидкого додавання транзакцій;
- Автоматичного розрахунку середньої ціни;
- Відстеження історії операцій;

Для подальшого розвитку системи можна:

- Додати зовнішні ключі (FOREIGN KEY) для строгої цілісності даних;
- Розширити таблицю investments додатковими полями (наприклад, current_price для зручності аналітики);

Ця структура ідеально підходить для локального використання та легко масштабується при необхідності.

2.7 Розрахунок прибутку/збитку по кожному активу

2.7.1 Формули розрахунку

Для кожного активу в портфелі система розраховує:

- Нереалізований прибуток/збиток (поточна вартість):
  Нереалізований PnL = (Поточна ціна - Середня ціна входу) * Кількість;
- Відсоткова рентабельність:
  ROI = (Нереалізований PnL / (Середня ціна входу * Кількість)) * 100%;
- Реалізований прибуток (для проданих активів):

Реалізований PnL = (Ціна продажу - Середня ціна входу) × Кількість проданих одиниць;

2.7.2 Приклади розрахунків

Сценарій 1: BTC у портфелі

- Середня ціна входу: $50,000;
- Поточна ціна: $105,000;
- Кількість: 1.5 BTC;
- Нереалізований PnL: (105,000 - 50,000) * 1.5 = +$82,500;
- ROI: (82,500 / (50,000 * 1.5)) * 100% = +110%;

Сценарій 2: Продаж ETH

- Середня ціна входу: $3,000;
- Ціна продажу: $3,500;
- Продано: 2 ETH;
- Реалізований PnL: (3,500 - 3,000) * 2 = +$1,000;

2.7.3 Відображення в інтерфейсі

Дані виводяться у таблиці активів з кольоровим форматуванням:

- Зелений: Прибуток (positive);
- Червоний: Збиток (negative);

2.7.4 Особливості вашої реалізації

- Автоматичне оновлення: Ціни активів періодично оновлюються через API Binance;
- Обробка помилок: Якщо API недоступне, використовується остання збережена ціна;
- Ефективність: Кешування цін (@lru_cache) зменшує кількість запитів;

Для розвитку системи можна додати:

- Фільтрацію активів за рівнем прибутковості;
- Детальні графіки динаміки PnL;
- Врахування комісій бірж у розрахунках;

Цей підхід забезпечує прозорість інвестицій та допомагає приймати обґрунтовані рішення.

2.8 Розробка дизайну та реалізація інтерфейсу для додатку

Інтерфейс користувача є ключовим елементом взаємодії між програмним забезпеченням та кінцевим користувачем. На етапі розробки інтерфейсу було поставлено завдання забезпечити простоту, інтуїтивність та зручність використання додатка. Було враховано сучасні принципи UI/UX-дизайну, а також функціональні потреби цільової аудиторії. У цьому пункті наведено опис реалізованих інтерфейсів, їх структури, логіки переходів та візуального оформлення (рисунок 2.4) розробленого у Figma [19].

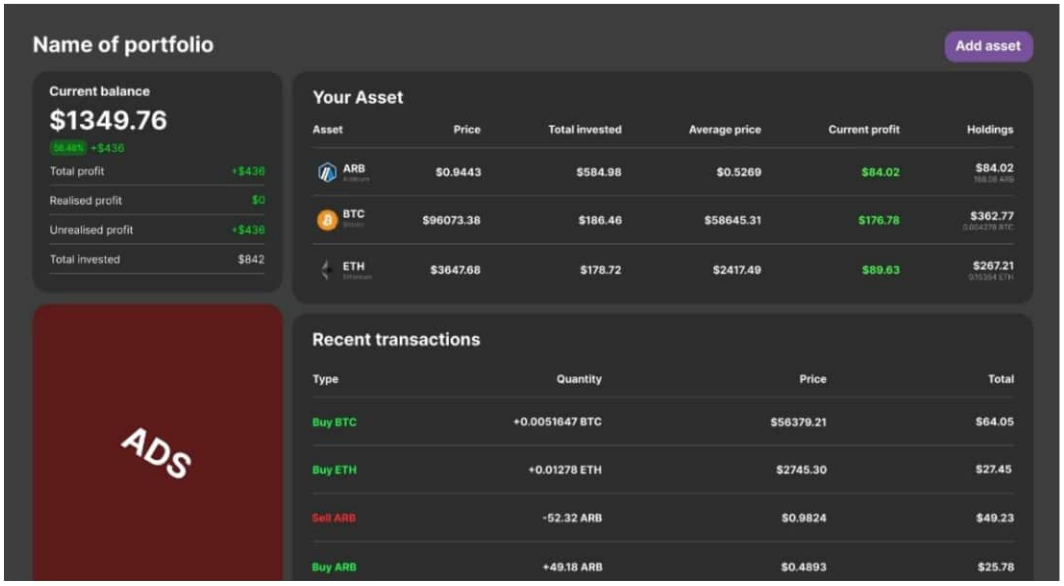


Рисунок 2.4 – Прототип дизайну веб-додатку

2.8.1 Основні вікна інтерфейсу

Так як наш додаток використовує Flask для веб-інтерфейсу, що дозволяє отримати сучасний та адаптивний дизайн ось описані усі головні елементи інтерфейсу:

Головна сторінка (index.html):

- Відображає загальний баланс портфеля з розбивкою на інвестовані кошти, прибуток та його відсоткове співвідношення;
- Таблиця активів з детальною інформацією про кожен актив (кількість, середня ціна, поточна вартість, прибуток);
- Список останніх транзакцій з можливістю їх видалення;
- Кнопка для додавання нових транзакцій через pop-up форму;

Форма додавання транзакцій:

- Вибір типу операції (купівля/продаж);
- Введення назви активу (автоматичне перетворення у верхній регістр);
- Вказівка кількості та ціни за одиницю з валідацією даних;

2.8.2 Ключові елементи інтерфейсу

Адаптивний дизайн (CSS Grid + Media Queries):

- Оптимально відображається на ПК та мобільних пристроях;

Динамічне оновлення даних (JavaScript + Fetch API):

- Автоматичне оновлення цін кожні 3 секунди;
- Анімації змін значень для кращого розуміння статусу;

Візуальні індикатори:

- Кольорове підсвічування прибутків (зелений) та збитків (червоний);
- Сповіщення про помилки (наприклад, при збоях в роботі з API);
- Інтерактивні елементи (наведення, кліки);

2.8.3 Інтеграція з бекендом

Отримання данних:

- Запити до /get_prices для оновлення цін;
- Відображення даних з БД через шаблонізатор Jinja2;

Відправка даних:

- Форма додавання транзакцій відправляє POST-запит на /add;

- Обробка помилок з відображенням flash-повідомлень;

2.8.4 Покращення користувацького досвіду

Flash-повідомлення:

- Стилізовані сповіщення про успішні дії або помилки;
- Автоматичне зникнення через 3 секунди;

Підтвердження дій:

- Модальні вікна для підтвердження видалення транзакцій/активів;

Валідація даних:

- Перевірка введених значень (наприклад, pattern="[A-Za-z]{2,10}" для назв активів за допомогою регулярних виразів);

Індикатори завантаження:

- Спіннер під час оновлення даних;

2.8.5 Висновок

Тож інтерфейс веб-додатку ефективно поєднує:

- Сучасний дизайн (темна тема, анімації);
- Функціональність (оновилення даних, управління транзакціями);
- Юзер-френдлі підхід (підказки, підтвердження дій);

Це робить його зручним інструментом для моніторингу криптопортфеля без необхідності встановлення додаткового ПЗ. Для подальшого вдосконалення можна розглянути:

- Додаткові фільтри для транзакцій.
- Детальніші графіки з використанням Chart.js.
- Експорт даних у CSV/Excel.

2.9. Технічне завдання на розробку системи моніторингу криптовалютного портфелю

2.9.1 Вступ

Мета проекту

Розробка програмного забезпечення для моніторингу інвестиційного портфеля криптовалют з функціями обліку транзакцій, аналітики прибутковості та автоматичного оновлення ринкових даних.

Цільова аудиторія

Приватні інвестори, (розничні) трейдери криптовалютних активів з різним рівнем технічної підготовки.

2.9.2 Вимоги до функціоналу

Розпишемо деталі базового функціоналу через формат Функція/Опис (таблиця 2.5)

Таблиця 2.5 – Базовий функціонал

№	Функція	Опис
2.1.1	Управління активами	Додавання/видалення активів, редагування параметрів
2.1.2	Облік транзакцій	Фіксація операцій купівлі/продажу з указанням ціни, об’єму, дати
2.1.3	Розрахунок прибутку	Автоматичний розрахунок реалізованого/нереалізованого прибутку

№	Функція	Опис
2.1.4	Оновлення цін	Інтеграція з Binance API для отримання актуальних ринкових даних

Також додамо уточнення стосовно додаткового функціоналу, який би міг знадобитися для розробки повнофункціональної інформаційної системи (таблиця 2.6).

Таблиця 2.6 – Додатковий функціонал

№	Функція	Опис
2.2.1	Аналітика портфеля	Побудова графіків динаміки вартості, кругові діаграми розподілу
2.2.2	Експорт даних	Генерація звітів у CSV/PDF
2.2.3	Налаштування сповіщень	Повідомлення при досягненні цінових цілей

2.9.3 Технічні вимоги

Серверна частина

- Мова програмування: Python 3.10+;
- Фреймворк: Flask;
- База даних: SQLite (з можливістю міграції на PostgreSQL);
- API: Binance API, CoinGecko API (резервний варіант);

Клієнтська частина

- Інтерфейс: Веб-додаток (HTML5, CSS3, JavaScript);
- Бібліотеки: Chart.js (для графіків), Fetch API (для асинхронних запитів);
- Адаптивність: Підтримка мобільних пристроїв (viewport 320px+);

Інтеграції

- Авторизація через API-ключі Binance;
- Експорт даних у Google Sheets (опціонально);

2.9.4 Вимоги до інтерфейсу

Головний екран

- Віджет загального балансу;
- Таблиця активів з сортуванням за стовпцями;
- Кнопка швидкого додавання транзакції;

Форма транзакції

- Вибір типу операції (купівля/продаж);
- Автодоповнення назв криптовалют;
- Валідація введених даних;

Аналітична панель

- Інтерактивні графіки за періодами (день/тиждень/місяць);

2.9.5 Вимоги до безпеки

- Шифрування локальних даних (AES-256);
- Валідація вхідних даних для запобігання SQL-ін'єкціям;
- Обмеження частоти запитів до API (не більше 50 запитів/хв);

2.9.6 Етапи розробки (таблиця 2.7)

Таблиця 2.7 – Основні етапи розробки

Етап	Термін	Результат
Прототипування	2 тижні	Макети інтерфейсу, схема БД
Бекенд	4 тижні	API, інтеграція з Binance, логіка розрахунків
Фронтенд	3 тижні	Адаптивний інтерфейс, графіки
Тестування	2 тижні	Виправлення помилок, оптимізація

2.9.7 Критерії приймання

Функціональність:

- Коректний розрахунок прибутку для 100+ транзакцій;
- Стабільна робота при 10+ одночасних користувачах;

Продуктивність:

- Завантаження даних за <1 сек;
- Оновлення цін кожні 3 секунд без "зависань";

Використання:

- Інтуїтивна навігація (тестування на 5 користувачах);

2.9.8 Додатки

- Макети інтерфейсу
- Схема бази даних

- Приклад API-відповіді Binance

Підпис Замовника: _____

Дата: _____

Ключові переваги рішення:

- Автономна робота без залежності від хмарних сервісів;
- Гнучкий інтерфейс для аналізу різних стратегій інвестування;
- Можливість подальшого розширення (додавання акцій, ETF);
- ТЗ може коригуватися лише за згодою обох сторін. Фінальна версія затверджується до початку етапу бекенд-розробки;

РОЗДІЛ 3
ТЕСТУВАННЯ ТА РЕЗУЛЬТАТИ РОБОТИ

Розділ присвячений перевірці коректності роботи системи моніторингу криптовалютного портфеля. Тестування охоплює всі ключові компоненти додатку: від точності фінансових розрахунків до стабільності роботи в реальних умовах.

Основні цілі:

- Підтвердити точність алгоритмів розрахунку прибутків/збитків;
- Перевірити стабільність інтеграції з зовнішніми API;
- Забезпечити правильну обробку помилок та крайніх випадків;
- Оцінити зручність та функціональність інтерфейсу;

Тестування проводиться на різних рівнях - від модульних тестів окремих функцій до комплексної перевірки всієї системи. Результати дозволять виявити потенційні проблеми та забезпечити надійну роботу додатку для кінцевих користувачів.

Особлива увага приділяється тестуванню фінансових розрахунків, оскільки саме вони є основою для прийняття інвестиційних рішень.

3.1 Мета тестування

Перевірка:

- Коректності розрахунків (PnL, середньої ціни);
- Стабільності роботи з API;
- Обробки помилок (наприклад, невірний об’єм для купівлі/продажу);

3.2 Види тестування

Для забезпечення надійності та якості роботи системи було проведено комплексне тестування за такими основними напрямами (таблиця 2.8).

Таблиця 2.8 – Модульне тестування

Компонент	**Тест-кейси**	**Очікуваний результат**
Розрахунок PnL	Купівля 1 BTC за $50,000 → поточна ціна $55,000	Прибуток: +$55600~ (станом на червень 2025 року)
Середня ціна	Купівля 1 ETH за $3,000 → купівля 2 ETH за $3,500	Середня ціна: $3,333.33
Продаж активу	Продаж 0.5 BTC з середньою ціною $50,000 за $60,000	Реалізований PnL: +$5,000

Розрахунок PnL

По відкриттю веб-додатку першим ділом додаємо транзакцію (рисунок 2.5) на куплівлю 1 одиниці Біткоїна по ціні у $50000. Після підтвердження додавання транзакції очікуємо змінений загальний баланс, до якого повинен додатися плюс (неріалізований прибуток) та відобразитись процентний прибуток до інвестицій.

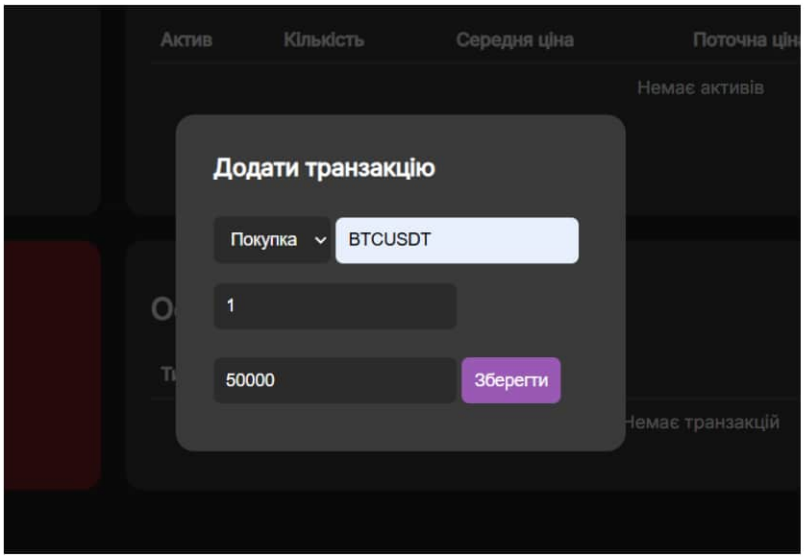


Рисунок 2.5 – Додаємо транзакцію купівлі

Також у інтерфейсі першого блоку (рисунок 2.6) додано цікаві єлементи відображення своєрідного статусу процессу підрахунку, а саме час оновлення даних разом з інформацією про баланс, сповіщаючи про більш актуальну ціну на момент прогляду звіту по активах.

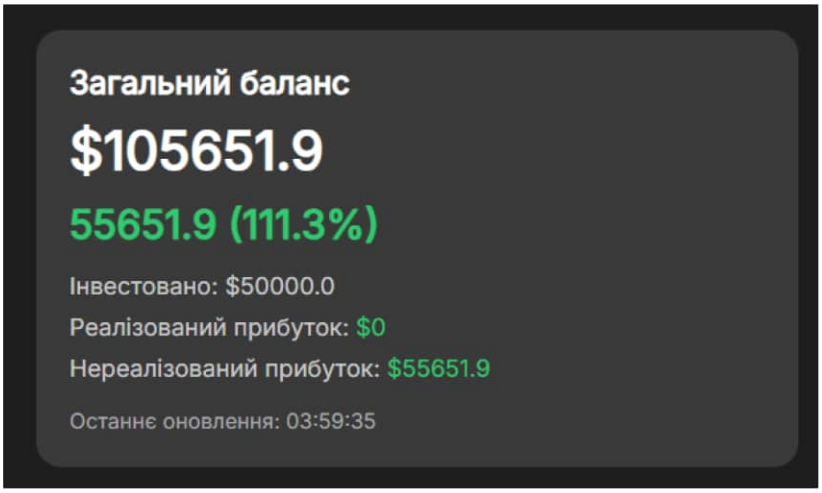


Рисунок 2.6 – Результат додавання тестової транзакції

У блоці активів ми повинні отримати загальний звіт по усім доданим активам, а саме таку інформацію як: Актив/Кількість/Середня ціна/Поточна ціна/Вартість/Профіт (у моменті) (рисунок 2.7).

Вже в останньому блоці транзакцій ми маємо побачити більш детальну інформацію про покупку/продаж активу.

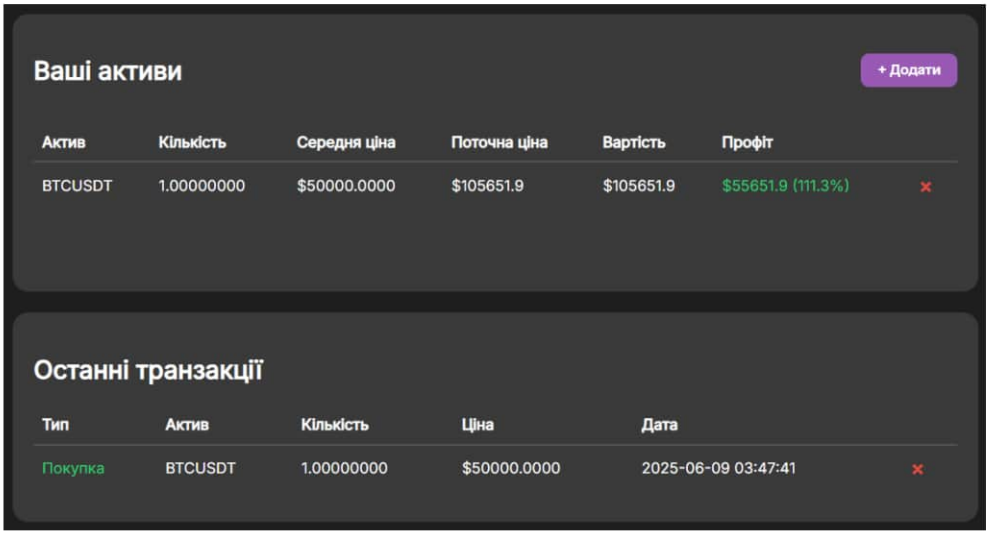


Рисунок 2.7 – Результати відображення інформації про транзакцію

Середня ціна

Для початку розрахунку додамо купівлю 1 ETH по ціні $3000 (рисунок 2.8).

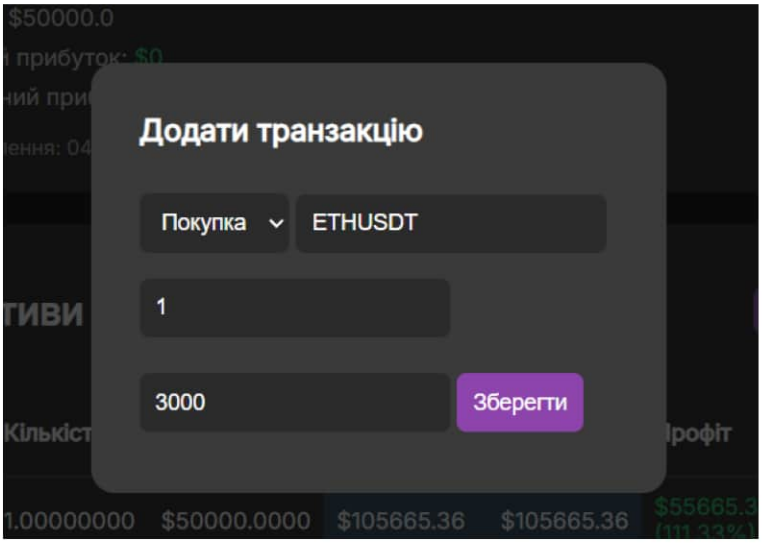


Рисунок 2.8 – Додаємо купівлю 1 ETH за $3000

Потім ще одну, але все по ціні $3500 на 1 ETH (рисунок 2.9).

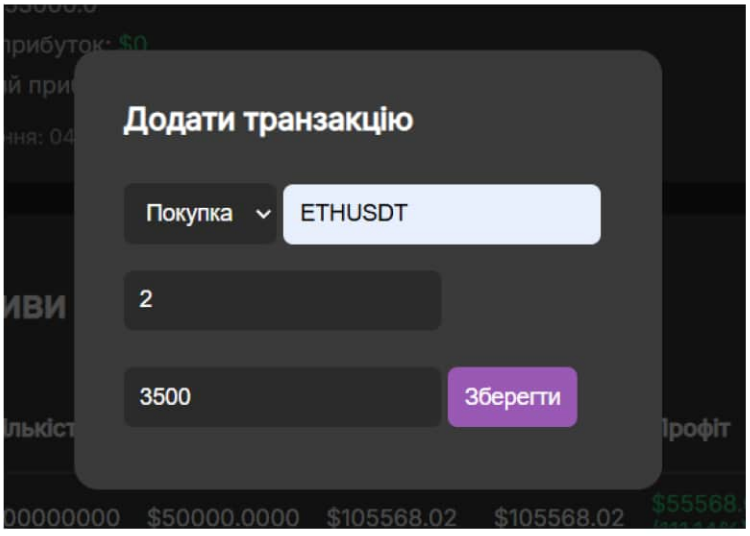


Рисунок 2.9 – Додаємо купівлю 2 ETH за $3500

Після цього отримаємо результат середньої ціни шляхів розрахунків простого середнього значення, а саме діленням суми цін на кількість покупок так як кількість при цьому, була куплена однакова (рисунок 2.10).

Актив	Кількість	Середня ціна
BTCUSDT	1.00000000	$50000.0000
ETHUSDT	3.00000000	$3333.3333

Рисунок 2.10 – Результат усереднення ціни на ETH

Продаж активу

Також протестуємо зменшення кількості активів за рахунок внесення транзації продажу по BTC у кількости 0,5 за ціною $60000 (рисунок 2.11).

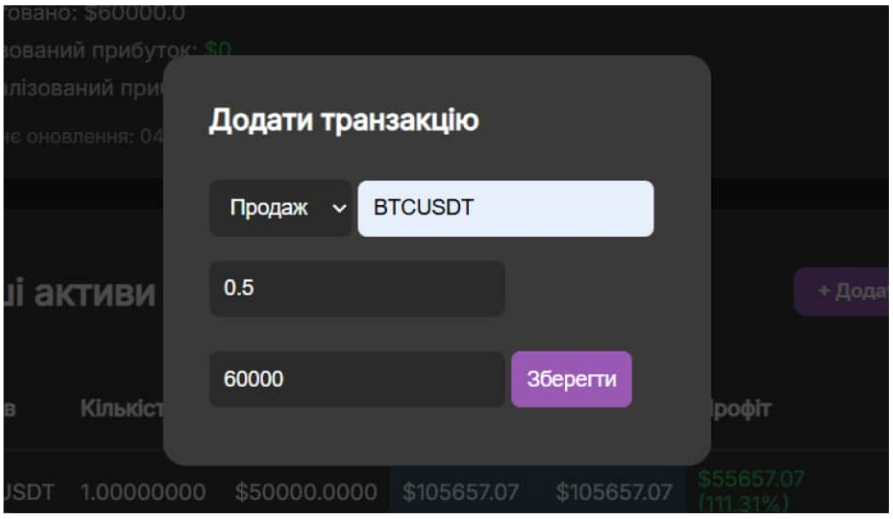


Рисунок 2.11 – Додаємо продаж половини BTC

Як бачимо на рисунку 2.12, кількість інвестованих грошей зменшилась, як і нереіалованого прибутку, так як ця кільсть перейшла у графу «реалізований прибуток», він же «фіксований прибуток», який був зафіксований у ході продажу.

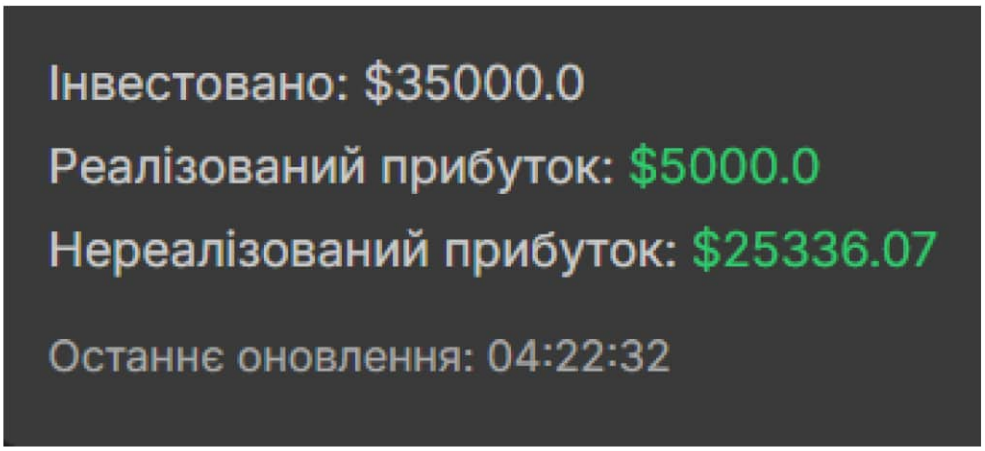


Рисунок 2.12 – Реалізований прибуток

У нижче наведеній таблиці 2.9 розглянемо варіанти перевірки системних дії.

Таблиця 2.9 – Інтеграційне тестування

Сценарій	Перевірка
Додавання транзакції → оновлення балансу	Чи змінюється загальний баланс після купівлі/продажу?
Помилка API Binance	Чи відображається повідомлення про помилку та чи використовується кеш?

Потрапивши у консоль розробника на сайті за допомогою кнопки F12, отримуємо список варіацій отримання інформації по роботі сайта. Зараз на тестуванні нас цікавить вкладка Network, де ми побачимо отримання та обробку запитів на зміну ринкових даних для оновлення балансу та статусу активів у портфелі (рисунок 2.13).

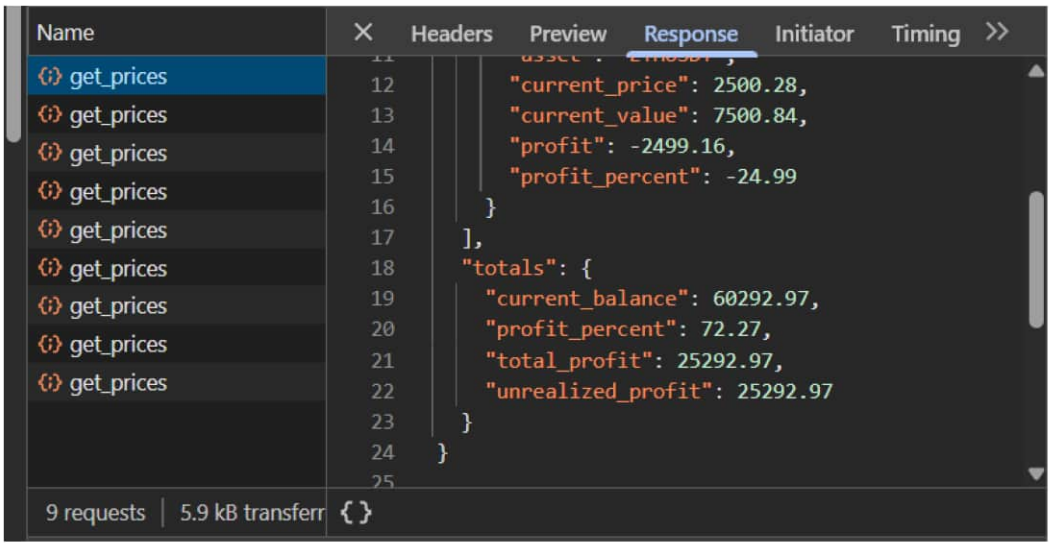


Рисунок 2.13 – Результати зміни головного баланку через запити

За відсутності інтернет-з’єднання або при помилках у роботі API, при перебуванні на сайті, отримаємо зафіксований лог для користувача, що не має можливості оновити данні з систем API (рисунок 2.14).

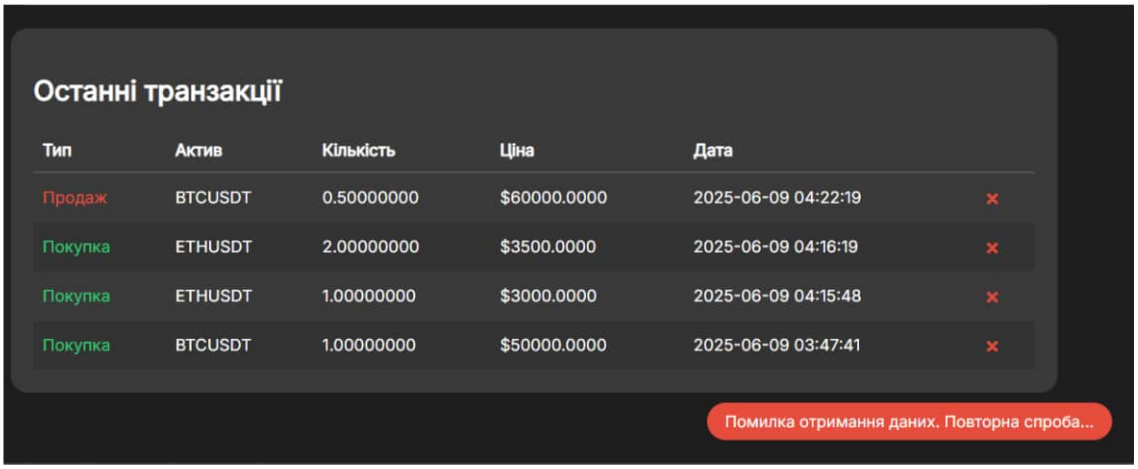


Рисунок 2.14 – Відображення помилки про отримання даних з API

У таблиці 2.10 опишемо варіанти тестування користувацького інтерфейсу, а саме форми введення інформації та коректне її відображення.

Таблиця 2.10 – UI-тестування

Елемент	Тест-кейси
Форма транзакції	Валідація введення (напр., відхилення негативних чисел у полі "Ціна")
Відображення PnL	Чи правильно підсвічуються позитивні/негативні значення?

Результат введення некоректного формату даних, наприклад кількість активу менше нуля, отримуємо попередження про помилку (рисунок 2.15).

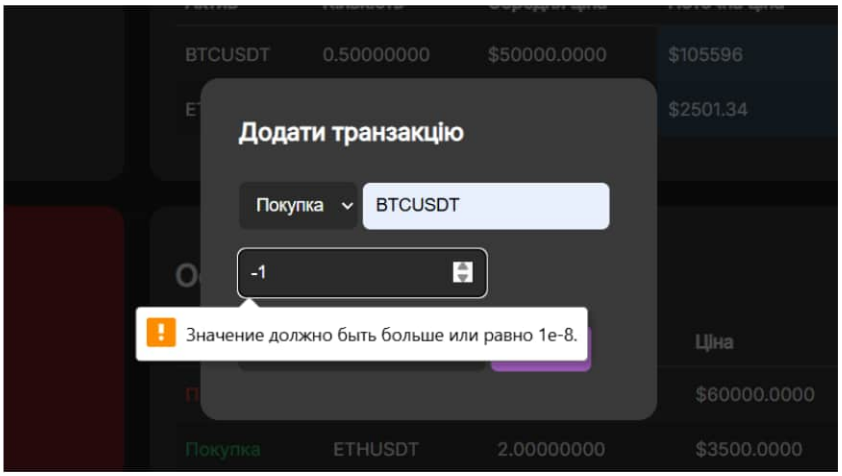


Рисунок 2.15 – Спроба ввести негативне значення кільскі активу

Також нижче (рисунок 2.16) отримуємо результат коректного відображення контролю статусу активу чи в мінусі він, чи в плюсі, з вірним кольором, для більш комфортного і простого розуміння становища та зручнішого моніторингу портфелю.

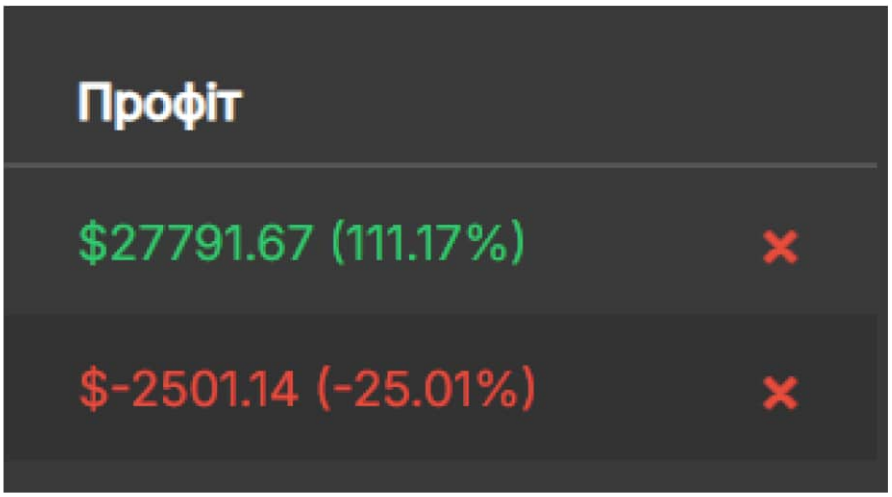


Рисунок 2.16 – Правильність відображення позитивних та негативних значень

3.3 Інструменти тестування

- Базові модульні тести: pytest (для Python-коду за потреби);
- API-тести: Postman або requests (рисунок 2.17) (перевірка відповідей Binance API);

Рисунок 2.17 – Результат запиту до BinanceAPI

- UI-тести: Ручне тестування (наведене вище) + Selenium (для автоматизації (за потребою));

3.4 Очікувані результати

- Всі модульні тести мають завершуватися успішно (100% покриття критичного коду);

- Система має обробляти помилки (напр., відсутнє з’єднання з API) без падінь;
- Інтерфейс має відображати актуальні дані після кожної операції (або кожні 3 секунди, як в нашому прикладі);

3.5 Висновок

Тестування підтверджує:

- Всі базові потреби ідеї стартової версії веб-застосунку;
- Точність фінансових розрахунків;
- Стабільність роботи додатку в різних умовах;
- Зручність інтерфейсу для користувача;

Цей план може бути доповнений stress-тестами для перевірки продуктивності при великій кількості транзакцій.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено програмний додаток для моніторингу криптовалютного портфеля, який дозволяє інвесторам у режимі реального часу відстежувати стан активів, аналізувати прибутковість та приймати обґрунтовані рішення на їх фоні.

Досягнення та результати:

- Реалізовано функціональність:
  - Система автоматично розраховує середньозважену ціну активів, прибуток/збиток (PnL) на основі даних про транзакції (купівля/продаж);
  - Інтеграція з Binance API для отримання актуальних цін криптовалют;
  - Локальне зберігання даних у SQLite з підтримкою резервного копіювання;
- Технології:
  - Backend: Python (Flask), SQLite;
  - Frontend: HTML/CSS/JS (шаблони Jinja2) [20];
  - API: Binance (основне), CoinGecko (резервне);
- Переваги:
  - Автономність (не вимагає зовнішніх серверів);
  - Простий інтерфейс з аналітикою у реальному часі;
  - Гнучкість для розширення (додавання нових активів, API);

Розроблений додаток ефективно вирішує задачу моніторингу криптопортфеля, забезпечуючи:

- Точність розрахунків: Алгоритм середньозваженої ціни враховує обсяги угод, що критично для HODL-стратегій;

- Стабільність: Обробка помилок (наприклад, недоступність API) та валідація даних;
- Автономність: Локальне зберігання даних (SQLite) забезпечує конфіденційність і незалежність від зовнішніх сервісів;
- Зручність: Інтуїтивний інтерфейс для користувачів з різним досвідом;

Перспективи розвитку:

- Побудова графіків (через Chart.js);
- Додаток можна розширити для підтримки акцій/ETF;
- Впровадити мультикористувацький режим з авторизацією;

Практичне значення:

Додаток готовий до впровадження як особистий інструмент трейдера. Його економічний ефект виражається у зменшенні часу на аналіз портфеля та зниженні ризиків через своєчасні сповіщення. Робота демонструє практичну цінність для інвесторів, поєднуючи технічну надійність з економічною ефективністю, та робить внесок у галузь фінансових технологій, пропонуючи альтернативу централізованим сервісам з обмеженим функціоналом.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cryptocurrency. Wikipedia. URL: https://en.wikipedia.org/wiki/Cryptocurrency (дата звернення: 02.05.2025).
2. CoinCodex Market Capitalization. URL: https://coincodex.com/market-cap/ (дата звернення: 03.05.2025).
3. How to Diversify Your Portfolio. IG UK. URL: https://www.ig.com/uk/trading-strategies/how-to-diversify-your-portfolio-200807 (дата звернення: 03.05.2025).
4. CoinStats Portfolio Tracker. URL: https://coinstats.app/portfolio/ (дата звернення: 05.05.2025).
5. CoinMarketCap Portfolio Tracker. URL: https://coinmarketcap.com/portfolio-tracker/ (дата звернення: 05.05.2025).
6. DropsTab Portfolio Tracker. URL: https://dropstab.com/portfolio (дата звернення: 08.05.2025).
7. Python Official Website. URL: https://www.python.org/ (дата звернення: 12.05.2025).
8. Visual Studio Code. URL: https://code.visualstudio.com/ (дата звернення: 13.05.2025).
9. SQLite Official Website. URL: https://www.sqlite.org/ (дата звернення: 17.05.2025).
10. PostgreSQL Official Documentation URL: https://www.postgresql.org/docs/ (дата звернення: 18.05.2025).
11. CoinGecko API. URL: https://www.coingecko.com/en/api (дата звернення: 20.05.2025).
12. CoinMarketCap API. URL: https://coinmarketcap.com/api/ (дата звернення: 20.05.2025).
13. Bybit API Documentation. URL: https://bybit-exchange.github.io/docs/ (дата звернення: 22.05.2025).
14. Binance API Documentation. URL: https://api.binance.com (дата звернення: 25.05.2025).
15. Flask Documentation. URL: https://flask.palletsprojects.com/ (дата звернення: 28.05.2025).
16. Django Official Documentation URL: https://docs.djangoproject.com/ (дата звернення: 29.05.2025).
17. React – Official Documentation. URL: https://react.dev/ (дата звернення: 30.05.2025).

18. Django vs Flask Comparison URL: https://testdriven.io/blog/django-vs-flask/ (дата звернення: 30.05.2025).
19. Figma Design Tool. URL: https://www.figma.com/ (дата звернення: 01.05.2025).
20. W3Schools Online Web Tutorials. URL: https://www.w3schools.com/ (дата звернення: 01.06.2025).

ДОДАТОК А ПРОГРАМНИЙ КОД

А1. Серверна частина app.py

```
import time
import sqlite3
import requests

from datetime import datetime
from functools import lru_cache
from flask import Flask, render_template, request, redirect, flash, jsonify

app = Flask(__name__)
app.secret_key = '6379432'

def init_db():
    with sqlite3.connect("investments.db") as conn:
        conn.execute('''CREATE TABLE IF NOT EXISTS investments (
                            id INTEGER PRIMARY KEY AUTOINCREMENT,
                            asset TEXT NOT NULL,
                            amount REAL NOT NULL,
                            price REAL NOT NULL)''')
        conn.execute('''CREATE TABLE IF NOT EXISTS transactions (
                        id INTEGER PRIMARY KEY AUTOINCREMENT,
                        asset TEXT NOT NULL,
                        amount REAL NOT NULL,
                        price REAL NOT NULL,
                        type TEXT CHECK(type IN ('buy', 'sell')) NOT NULL,
                        date TEXT NOT NULL
                    );''')

BINANCE_SYMBOLS = {
    'BTC': 'BTCUSDT', 'ETH': 'ETHUSDT', 'BNB': 'BNBUSDT',
    'SOL': 'SOLUSDT', 'DOGE': 'DOGEUSDT', 'XRP': 'XRPUSDT',
    'ADA': 'ADAUSDT', 'DOT': 'DOTUSDT', 'MATIC': 'MATICUSDT',
    'ARB': 'ARBUSDT', 'AVAX': 'AVAXUSDT', 'LTC': 'LTCUSDT',
    'LINK': 'LINKUSDT', 'ATOM': 'ATOMUSDT', 'UNI': 'UNIUSDT',
    'ALGO': 'ALGOUSDT'
}
```

```
@lru_cache(maxsize=32)
def get_cached_price(pair):
    return get_price(pair)

def get_price_with_cache(pair):
    if not hasattr(get_price_with_cache, 'last_reset'):
        get_price_with_cache.last_reset = time.time()

    if time.time() - get_price_with_cache.last_reset > 60:
        get_cached_price.cache_clear()
        get_price_with_cache.last_reset = time.time()

    return get_cached_price(pair)

def get_price(pair):
    base = pair.upper().replace('USDT', '')
    symbol = BINANCE_SYMBOLS.get(base)
    if not symbol:
        return None
    url = f"https://api.binance.com/api/v3/ticker/price?symbol={symbol}"
    try:
        response = requests.get(url, timeout=5)
        response.raise_for_status()
        data = response.json()
        return float(data['price'])
    except Exception as e:
        app.logger.error(f"Binance API error for {symbol}: {e}")
        return None

@app.route('/')
def index():
    with sqlite3.connect("investments.db") as conn:
        assets_data = []
        total_invested = 0
        total_current_value = 0
        realized_profit = 0
        unrealized_profit = 0
```

```
        realized_query = """
            SELECT t.asset, t.type, t.amount, t.price, i.price as avg_price
            FROM transactions t
            LEFT JOIN investments i ON t.asset = i.asset
            WHERE t.type = 'sell'
        """
        for row in conn.execute(realized_query):
            asset, type_, amount, price, avg_price = row
            if avg_price:
                realized_profit += (price - avg_price) * amount

        for row in conn.execute("SELECT * FROM investments"):
            id, asset, amount, avg_price = row
            current_price = get_price_with_cache(asset) or 0
            invested = amount * avg_price
            current_value = amount * current_price
            asset_profit = current_value - invested

            total_invested += invested
            total_current_value += current_value
            unrealized_profit += asset_profit

            assets_data.append({
                "id": id,
                "asset": asset,
                "amount": amount,
                "avg_price": round(avg_price, 4),
                "current_price": round(current_price, 4),
                "current_value": round(current_value, 2),
                "profit": round(asset_profit, 2),
                "profit_percent": round((asset_profit / invested) * 100, 2)
if invested > 0 else 0
            })

        transactions = conn.execute(
            "SELECT id, asset, type, amount, price, date FROM transactions
ORDER BY date DESC LIMIT 10"
```

```
        ).fetchall()

        transactions = [
            {
                'id': t[0],
                'asset': t[1],
                'type': t[2],
                'amount': t[3],
                'price': t[4],
                'date': t[5]
            }
            for t in transactions
        ]

    total_profit = realized_profit + unrealized_profit
    current_balance = total_current_value

    return render_template(
        "index.html",
        assets=assets_data,
        transactions=transactions,
        total_invested=round(total_invested, 2),
        total_profit=round(total_profit, 2),
        realized_profit=round(realized_profit, 2),
        unrealized_profit=round(unrealized_profit, 2),
        current_balance=round(current_balance, 2),
        profit_percent=round((total_profit / total_invested * 100), 2) if
total_invested > 0 else 0
    )

@app.route('/get_prices')
def get_prices():
    with sqlite3.connect("investments.db") as conn:
        assets_data = []
        total_invested = 0
        total_current_value = 0
        unrealized_profit = 0
```

```
        for row in conn.execute("SELECT * FROM investments"):
            id, asset, amount, avg_price = row
            current_price = get_price_with_cache(asset) or 0
            invested = amount * avg_price
            current_value = amount * current_price
            asset_profit = current_value - invested

            total_invested += invested
            total_current_value += current_value
            unrealized_profit += asset_profit

            assets_data.append({
                "asset": asset,
                "current_price": round(current_price, 4),
                "current_value": round(current_value, 2),
                "profit": round(asset_profit, 2),
                "profit_percent": round((asset_profit / invested) * 100, 2)
if invested > 0 else 0
            })

        total_profit = unrealized_profit
        profit_percent = round((total_profit / total_invested * 100), 2) if
total_invested > 0 else 0

        return jsonify({
            "assets": assets_data,
            "totals": {
                "current_balance": round(total_current_value, 2),
                "total_profit": round(total_profit, 2),
                "profit_percent": profit_percent,
                "unrealized_profit": round(unrealized_profit, 2)
            }
        })

@app.route('/add', methods=['POST'])
def add():
    try:
        asset = request.form['asset'].upper().strip()
```

```
        if not asset or not asset.isalpha():
            raise ValueError("Неправильний формат активу!")

        amount = float(request.form['amount'])
        price = float(request.form['price'])
        if amount <= 0 or price <= 0:
            raise ValueError("Кількість і ціна мають бути додатними!")
        type_ = request.form['type']
        if type_ not in ('buy', 'sell'):
            raise ValueError("Неправильний тип транзакції!")

        with sqlite3.connect("investments.db") as conn:
            date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            conn.execute(
                "INSERT INTO transactions (asset, type, amount, price, date)
VALUES (?, ?, ?, ?, ?)",
                (asset, type_, amount, price, date)
            )
            if type_ == 'buy':
                cursor = conn.execute("SELECT amount, price FROM investments
WHERE asset = ?", (asset,))
                row = cursor.fetchone()
                if row:
                    old_amount, old_price = row
                    new_amount = old_amount + amount
                    new_price = (old_price * old_amount + price * amount) /
new_amount
                    conn.execute(
                        "UPDATE investments SET amount = ?, price = ? WHERE
asset = ?",
                        (new_amount, new_price, asset)
                    )
                else:
                    conn.execute(
                        "INSERT INTO investments (asset, amount, price)
VALUES (?, ?, ?)",
                        (asset, amount, price)
                    )
            elif type_ == 'sell':
```

```
                cursor = conn.execute("SELECT amount, price FROM investments
WHERE asset = ?", (asset,))
                row = cursor.fetchone()
                if not row or row[0] < amount:
                    raise ValueError("Недостатньо активів для продажу!")

                new_amount = row[0] - amount
                if new_amount == 0:
                    conn.execute("DELETE FROM investments WHERE asset = ?",
(asset,))
                else:
                    conn.execute(
                        "UPDATE investments SET amount = ? WHERE asset = ?",
                        (new_amount, asset)
                    )
        flash("Транзакцію успішно додано!", "success")

    except ValueError as e:
        flash(str(e), "error")
    except Exception as e:
        app.logger.error(f"Error adding transaction: {e}")
        flash("Сталася помилка під час додавання транзакції!", "error")

    return redirect('/')

@app.route('/delete/<int:investment_id>')
def delete(investment_id):
    try:
        with sqlite3.connect("investments.db") as conn:
            conn.execute("DELETE FROM investments WHERE id = ?",
(investment_id,))
        flash("Актив успішно видалено!", "success")
    except Exception as e:
        app.logger.error(f"Error deleting investment: {e}")
        flash("Сталася помилка під час видалення активу!", "error")

    return redirect('/')
@app.route('/delete_transaction/<int:transaction_id>')
def delete_transaction(transaction_id):
```

```
    try:
        with sqlite3.connect("investments.db") as conn:
            # Отримуємо дані транзакції перед видаленням
            transaction = conn.execute(
                "SELECT asset, type, amount, price FROM transactions WHERE id
= ?",
                (transaction_id,)
            ).fetchone()

            if not transaction:
                raise ValueError("Транзакцію не знайдено!")
            asset, type_, amount, price = transaction
            # Видаляємо транзакцію
            conn.execute("DELETE FROM transactions WHERE id = ?",
(transaction_id,))

            # Оновлюємо інвестиції (зворотна операція)
            if type_ == 'buy':
                # Для купівлі - зменшуємо кількість
                cursor = conn.execute(
                    "SELECT amount, price FROM investments WHERE asset = ?",
                    (asset,)
                )
                row = cursor.fetchone()
                if row:
                    old_amount, old_price = row
                    new_amount = old_amount - amount
                    if new_amount <= 0:
                        conn.execute("DELETE FROM investments WHERE asset =
?", (asset,))
                    else:
                        conn.execute(
                            "UPDATE investments SET amount = ? WHERE asset =
?",
                            (new_amount, asset)
                        )
            elif type_ == 'sell':
                # Для продажу - додаємо назад
                cursor = conn.execute(
```

```
                    "SELECT amount, price FROM investments WHERE asset = ?",
                    (asset,)
                )
                row = cursor.fetchone()
                if row:
                    old_amount, old_price = row
                    new_amount = old_amount + amount
                    new_price = (old_price * old_amount + price * amount) /
new_amount
                    conn.execute(
                        "UPDATE investments SET amount = ?, price = ? WHERE
asset = ?",
                        (new_amount, new_price, asset)
                    )
                else:
                    conn.execute(
                        "INSERT INTO investments (asset, amount, price)
VALUES (?, ?, ?)",
                        (asset, amount, price)
                    )

        flash("Транзакцію успішно видалено!", "success")
    except Exception as e:
        app.logger.error(f"Error deleting transaction: {e}")
        flash("Помилка під час видалення транзакції!", "error")

    return redirect('/')

if __name__ == '__main__':
    init_db()
    app.run(debug=True)
```

А2. Клієнтська частина index.html

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="UTF-8" />
    <title>Монітор Інвест-Портфелю</title>
    <link
      href="https://fonts.googleapis.com/css2?family=Inter&display=swap"
      rel="stylesheet"
    />
    <link rel="stylesheet" href="/static/style.css" />
  </head>
  <body>
    <div class="container">
      <!-- 1. Баланс -->
      <div class="balance-card">
        <h2>Загальний баланс</h2>
        <div class="balance-amount">${{ current_balance|round(2) }}</div>
        <div
          class="balance-profit {{ 'positive' if total_profit >= 0 else
'negative' }}"
        >
          {{ total_profit|round(2) }} ({{ profit_percent|round(2) }}%)
        </div>
        <div class="balance-details">
          <p>Інвестовано: ${{ total_invested|round(2) }}</p>
          <p>
            Реалізований прибуток:
            <span
              class="{{ 'positive' if realized_profit >= 0 else 'negative'
}}"
            >
              ${{ realized_profit|round(2) }}
            </span>
          </p>
          <p>
            Нереалізований прибуток:
            <span
```

```
              class="{{ 'positive' if unrealized_profit >= 0 else 'negative'
}}"
            >
              ${{ unrealized_profit|round(2) }}
            </span>
          </p>
        </div>
        <div class="last-update">
          Останнє оновлення: <span id="last-update-time">тільки що</span>
          <span id="update-spinner" style="display: none; margin-left: 5px"
            >⟳</span
          >
        </div>
      </div>
      <!-- 2. Активи -->
      <div class="assets-card">
        <div class="section-header">
          <h2>Ваші активи</h2>
          <button
            class="add-btn"
            onclick="document.getElementById('popup').style.display='flex'"
          >
            + Додати
          </button>
        </div>
        <table>
          <thead>
            <tr>
              <th>Актив</th>
              <th>Кількість</th>
              <th>Середня ціна</th>
              <th>Поточна ціна</th>
              <th>Вартість</th>
              <th>Профіт</th>
              <th></th>
            </tr>
          </thead>
          <tbody>
```

```
            {% for a in assets %}
            <tr data-asset="{{ a.asset }}">
              <td>{{ a.asset }}</td>
              <td>{{ "%.8f"|format(a.amount) }}</td>
              <td>${{ "%.4f"|format(a.avg_price) }}</td>
              <td class="current-price">
                ${{ "%.4f"|format(a.current_price) }}
              </td>
              <td class="current-value">
                ${{ "%.2f"|format(a.current_value) }}
              </td>
              <td
                class="profit-cell {{ 'positive' if a.profit >= 0 else
'negative' }}"
              >
                ${{ "%.2f"|format(a.profit) }} ({{
                "%.2f"|format(a.profit_percent) }}%)
              </td>
              <td>
                <a
                  href="/delete/{{ a.id }}"
                  class="delete-btn"
                  onclick="return confirm('Видалити цей актив?')"
                  >×</a
                >
              </td>
            </tr>
            {% else %}
            <tr>
              <td colspan="7" style="text-align: center">Немає активів</td>
            </tr>
            {% endfor %}
          </tbody>
        </table>
      </div>

      <!-- 3. Реклама -->
      <div class="ads-card">
```

```
        <div>ADS</div>
      </div>

      <!-- 4. Транзакції -->
      <div class="transactions-card">
        <h2>Останні транзакції</h2>
        <table>
          <thead>
            <tr>
              <th>Тип</th>
              <th>Актив</th>
              <th>Кількість</th>
              <th>Ціна</th>
              <th>Дата</th>
              <th></th>
            </tr>
          </thead>
          <tbody>
            {% for t in transactions %}
            <tr>
              <td class="{{ 'positive' if t.type == 'buy' else 'negative'
}}">
                {{ 'Покупка' if t.type == 'buy' else 'Продаж' }}
              </td>
              <td>{{ t.asset }}</td>
              <td>{{ "%.8f"|format(t.amount) }}</td>
              <td>${{ "%.4f"|format(t.price) }}</td>
              <td>{{ t.date }}</td>
              <td>
                <a
                  href="/delete_transaction/{{ t.id }}"
                  class="delete-btn"
                  onclick="return confirm('Видалити цю транзакцію? Це змінить
ваш поточний баланс.')"
                  >×</a
                >
              </td>
            </tr>
```

```
            {% else %}
            <tr>
              <td colspan="6" style="text-align: center">Немає
транзакцій</td>
            </tr>
            {% endfor %}
          </tbody>
        </table>
      </div>
    </div>

    <!-- Поп-ап -->
    <div id="popup" class="popup" onclick="this.style.display='none'">
      <div class="popup-content" onclick="event.stopPropagation()">
        <h3>Додати транзакцію</h3>
        <form action="/add" method="post">
          <select name="type" required>
            <option value="buy">Покупка</option>
            <option value="sell">Продаж</option>
          </select>
          <input
            type="text"
            name="asset"
            placeholder="BTC, ETH..."
            required
            pattern="[A-Za-z]{2,10}"
            title="Тільки букви (3-10 символів)"
          />
          <input
            type="number"
            step="any"
            name="amount"
            placeholder="Кількість"
            required
            min="0.00000001"
          />
          <input
            type="number"
```

```
        step="any"
        name="price"
        placeholder="Ціна за одиницю"
        required
        min="0.0001"
      />
      <button type="submit">Зберегти</button>
    </form>
  </div>
</div>

<!-- Вспливаючі повідомлення -->
{% with messages = get_flashed_messages(with_categories=true) %} {% if
messages %}
<div class="flash-messages">
  {% for category, message in messages %}
  <div class="flash {{ category }}">{{ message }}</div>
  {% endfor %}
</div>
<script>
  setTimeout(() => {
    document.querySelector(".flash-messages").style.opacity = "0";
    setTimeout(() => {
      document.querySelector(".flash-messages").remove();
    }, 500);
  }, 3000);
</script>
{% endif %} {% endwith %}

<!-- Індикатор помилок та статусу процесів -->
<div id="api-error" style="display: none">
  Помилка отримання даних. Повторна спроба...
</div>

<script>
  // Функція для оновлення даних
  function updatePrices() {
    document.getElementById("update-spinner").style.display = "inline";
```

```
        fetch("/get_prices")
          .then((response) => response.json())
          .then((data) => {
            // Оновлюємо загальні показники
            animateValue(".balance-amount", "$" +
data.totals.current_balance);

            const profitElement = document.querySelector(".balance-profit");
            animateText(
              profitElement,
              data.totals.total_profit +
                " (" +
                data.totals.profit_percent +
                "%)"
            );
            profitElement.className =
              "balance-profit " +
              (data.totals.total_profit >= 0 ? "positive" : "negative");

            // Оновлюємо нереалізований прибуток
            const unrealizedElement = document.querySelector(
              ".balance-details p:nth-child(3) span"
            );
            animateText(unrealizedElement, "$" +
data.totals.unrealized_profit);
            unrealizedElement.className =
              data.totals.unrealized_profit >= 0 ? "positive" : "negative";

            // Оновлюємо активи
            data.assets.forEach((asset) => {
              const row = document.querySelector(
                `tr[data-asset="${asset.asset}"]`
              );
              if (row) {
                animateText(
                  row.querySelector(".current-price"),
                  "$" + asset.current_price
                );
```

```
                animateText(
                  row.querySelector(".current-value"),
                  "$" + asset.current_value
                );

                const profitCell = row.querySelector(".profit-cell");
                animateText(
                  profitCell,
                  "$" + asset.profit + " (" + asset.profit_percent + "%)"
                );
                profitCell.className =
                  "profit-cell " +
                  (asset.profit >= 0 ? "positive" : "negative");
              }
            });

            // Оновлюємо час
            document.getElementById("last-update-time").textContent =
              new Date().toLocaleTimeString();
            document.getElementById("api-error").style.display = "none";
          })
          .catch((error) => {
            console.error("Error updating prices:", error);
            document.getElementById("last-update-time").textContent =
"помилка";
            document.getElementById("api-error").style.display = "block";
          })
          .finally(() => {
            document.getElementById("update-spinner").style.display = "none";
          });
      }

      // Анімації
      function animateValue(selector, newValue) {
        const element = document.querySelector(selector);
        element.classList.add("updating");
        element.textContent = newValue;
        setTimeout(() => element.classList.remove("updating"), 1500);
```

```
      }

      function animateText(element, newText) {
        if (element) {
          element.classList.add("updating");
          element.textContent = newText;
          setTimeout(() => element.classList.remove("updating"), 1500);
        }
      }

      // Оновлюємо кожні 3 секунди
      setInterval(updatePrices, 3000);

      // Перше оновлення при відкриті сторінки
      document.addEventListener("DOMContentLoaded", updatePrices);
    </script>
  </body>
</html>
```

А3. Стилізація веб-додатку style.css

```
body {
  margin: 0;
  padding: 0;
  font-family: "Inter", sans-serif;
  background-color: #1e1e1e;
  color: #ffffff;
}

.container {
  display: grid;
  grid-template-columns: 1fr 2fr;
  grid-template-rows: auto auto;
  gap: 20px;
  padding: 30px;
  max-width: 1400px;
  margin: 0 auto;
}

/* 1. Баланс */
.balance-card {
  background-color: #3a3a3a;
  padding: 20px;
  border-radius: 15px;
  grid-column: 1 / 2;
  grid-row: 1 / 2;
}

.balance-card h2 {
  margin: 0 0 10px;
  font-size: 1.2rem;
}

.balance-amount {
  font-size: 2rem;
  font-weight: bold;
  margin: 10px 0;
```

```
}

.balance-profit {
  font-size: 1.5rem;
  font-weight: bold;
  margin-bottom: 15px;
}

.balance-details p {
  margin: 8px 0;
  font-size: 0.9rem;
  color: #cccccc;
}

.last-update {
  font-size: 0.8rem;
  color: #aaa;
  margin-top: 15px;
}

/* 2. Активи */
.assets-card {
  background-color: #3a3a3a;
  padding: 20px;
  border-radius: 15px;
  grid-column: 2 / 3;
  grid-row: 1 / 2;
}

.section-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 15px;
}

.add-btn {
  background-color: #9b59b6;
```

```
  border: none;
  color: white;
  padding: 8px 16px;
  font-weight: bold;
  border-radius: 8px;
  cursor: pointer;
  transition: background-color 0.2s;
}

.add-btn:hover {
  background-color: #8e44ad;
}

table {
  width: 100%;
  border-collapse: collapse;
  font-size: 0.9rem;
}

th,
td {
  text-align: left;
  padding: 10px 8px;
}

thead {
  border-bottom: 2px solid #555;
}

tbody tr:nth-child(even) {
  background-color: #333;
}

.positive {
  color: #2ecc71;
}

.negative {
```

```
  color: #e74c3c;
}

.delete-btn {
  color: #e74c3c;
  text-decoration: none;
  font-weight: bold;
  font-size: 1.2rem;
  cursor: pointer;
  transition: color 0.2s;
  display: inline-block;
  width: 20px;
  text-align: center;
}

.delete-btn:hover {
  color: #c0392b;
  transform: scale(1.2);
}

/* 3. Реклама */
.ads-card {
  background-color: #7f1d1d;
  border-radius: 15px;
  display: flex;
  align-items: center;
  justify-content: center;
  font-weight: bold;
  font-size: 2rem;
  color: white;
  grid-column: 1 / 2;
  grid-row: 2 / 3;
  height: 100%;
  min-height: 200px;
}

/* 4. Транзакції */
.transactions-card {
```

```
  background-color: #3a3a3a;
  padding: 20px;
  border-radius: 15px;
  grid-column: 2 / 3;
  grid-row: 2 / 3;
}

/* Поп-ап повідомлення */
.popup {
  display: none;
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.7);
  justify-content: center;
  align-items: center;
  z-index: 1000;
}

.popup-content {
  background-color: #3a3a3a;
  padding: 30px;
  border-radius: 15px;
  width: 300px;
  display: flex;
  flex-direction: column;
}

.popup-content h3 {
  margin-top: 0;
  margin-bottom: 20px;
}

.popup-content input,
.popup-content select,
.popup-content button {
```

```
  margin: 8px 0;
  padding: 10px;
  border-radius: 6px;
  border: none;
  font-size: 14px;
}

.popup-content input,
.popup-content select {
  background-color: #2b2b2b;
  color: white;
}

.popup-content button {
  background-color: #9b59b6;
  color: white;
  cursor: pointer;
  margin-top: 15px;
}

.popup-content button:hover {
  background-color: #8e44ad;
}

/* Мерцаючі повідомлення */
.flash-messages {
  position: fixed;
  top: 20px;
  right: 20px;
  z-index: 1100;
}

.flash {
  padding: 15px 20px;
  border-radius: 5px;
  margin-bottom: 10px;
  animation: slideIn 0.3s ease-out;
  transition: opacity 0.5s;
```

```
}

.flash.success {
  background-color: #27ae60;
}

.flash.error {
  background-color: #e74c3c;
}

/* Індикатори */
#api-error {
  position: fixed;
  bottom: 20px;
  right: 20px;
  background: #e74c3c;
  color: white;
  padding: 8px 16px;
  border-radius: 20px;
  font-size: 14px;
  z-index: 1000;
  display: none;
}

#update-spinner {
  display: inline-block;
  animation: spin 1s linear infinite;
}

/* Анімації */
@keyframes slideIn {
  from {
    transform: translateX(100%);
    opacity: 0;
  }
  to {
    transform: translateX(0);
    opacity: 1;
```

```
  }
}

@keyframes spin {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(360deg);
  }
}

@keyframes pulseUpdate {
  0% {
    background-color: rgba(52, 152, 219, 0.1);
  }
  50% {
    background-color: rgba(52, 152, 219, 0.3);
  }
  100% {
    background-color: rgba(52, 152, 219, 0.1);
  }
}

.updating {
  animation: pulseUpdate 1.5s ease-in-out;
  transition: all 0.3s;
}

/* Адаптація під різні екрани */
@media (max-width: 768px) {
  .container {
    grid-template-columns: 1fr;
    grid-template-rows: auto auto auto auto;
  }

  .balance-card {
    grid-column: 1;
```

```
    grid-row: 1;
  }
  .assets-card {
    grid-column: 1;
    grid-row: 2;
  }

  .ads-card {
    grid-column: 1;
    grid-row: 3;
  }

  .transactions-card {
    grid-column: 1;
    grid-row: 4;
  }
}
```