МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ВЕБДОДАТОК ДЛЯ МОНІТОРИНГУ ТА АНАЛІЗУ КРИПТОВАЛЮТНИХ ПОРТФЕЛІВ»**

Виконав: студент 4 курсу, групи 6.1211-1пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Б.Е. Бойчук
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бойчуку Богдану Едуардовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Вебдодаток для моніторингу та аналізу криптовалютних портфелів

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі та аналіз предметної області.
2. Теоретичні основи та вибір технологій.
3. Проєктування вебдодатку.
4. Програмна реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.01.2025	
2.	Збір вихідних даних.	15.01.2025	
3.	Обробка методичних та теоретичних	03.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	14.03.2025	
5.	Розробка третього та четвертого розділу.	12.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	16.06.2025	

Студент _____ Б.Е. Бойчук
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Вебдодаток для моніторингу та аналізу криптовалютних портфелів»: 72 с., 49 рис., 2 табл., 19 джерел, 3 додатки.

АУТЕНТИФІКАЦІЯ, БАЗА ДАНИХ, ВЕБІНТЕРФЕЙС, КРИПТОВАЛЮТА, КРИПТОПОРТФЕЛЬ, МОНІТОРИНГ, ОБРОБКА ДАНИХ, РИНОК ЦИФРОВИХ АКТИВІВ, ТЕСТУВАННЯ, ФІНАНСОВИЙ АНАЛІЗ, ХМАРНЕ ЗБЕРЕЖЕННЯ, NODE.JS, REACT, TAILWIND CSS, WEBSOCKET.

Об’єкт дослідження – розробка вебдодатоку для зручного моніторингу та аналізу криптовалютних активів користувача з можливістю візуалізації динаміки вартості та оцінки прибутковості портфеля.

Предмет дослідження – програмні засоби та технології розробки вебдодатку для збору, обробки та візуалізації даних криптовалютного портфеля в режимі реального часу.

Мета роботи: розробити вебдодаток для зручного моніторингу, зберігання, аналізу та візуалізації змін вартості криптовалютного портфеля користувача.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання та тестування програмного забезпечення.

У роботі проаналізовано підходи до моніторингу криптовалют, розглянуто архітектуру популярних платформ. Розроблено вебдодаток із використанням сучасних технологій (React.js, Node.js, Firebase тощо), який дає змогу створювати портфель, відстежувати його вартість, оцінювати прибутковість і візуалізувати дані. Проведено тестування та оцінено зручність використання. Результати можуть бути корисні приватним інвесторам для моніторингу портфеля в реальному часі.

SUMMARY

Bachelor’s qualifying paper “Web Application for Monitoring and Analyzing Cryptocurrency Portfolios”: 72 pages, 49 figures, 2 tables, 19 references, 3 supplements.

AUTHENTICATION, CLOUD STORAGE, CRYPTO PORTFOLIO, CRYPTOCURRENCY, DATA PROCESSING, DATABASE, DIGITAL ASSET MARKET, ENCRYPTION, FINANCIAL ANALYSIS, MONITORING, TESTING, UX/UI DESIGN, WEB INTERFACE, WEBSOCKET.

The object of the study is development of a web application for convenient monitoring and analysis of user’s cryptocurrency assets with the ability to visualize the value dynamics and assess the portfolio profitability.

The subject of the study is software tools and technologies for developing a web application for collecting, processing, and visualizing cryptocurrency portfolio data in real time.

The aim of the study is to develop a web application for convenient monitoring, storage, analysis, and visualization of changes in the value of a user’s cryptocurrency portfolio.

The methods of research are collecting and analyzing software requirements, methods of modeling, designing, constructing, and testing software.

The study analyzes modern approaches to monitoring cryptocurrencies and considers the architecture of popular platforms. A web application has been developed using modern technologies (React.js, Node.js, Firebase, etc.) that allows you to create a portfolio, track its value, estimate profitability, and visualize data. We have tested and evaluated its usability. The results can be useful for private investors to monitor their portfolio in real time.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Постановка задачі та аналіз предметної області 10
1.1 Актуальність теми 10
1.2 Вимоги до вебдодатку 10
1.3 Огляд існуючих рішень та їх аналіз 14
1.4 Постановка задачі 18
1.5 Очікувані результати 19
2 Теоретичні основи та вибір технологій 21
2.1 Основи веброзробки 21
2.1.1 Клієнтська частина 22
2.1.2 Серверна частина 22
2.1.3 База даних 24
2.1.4 Механізми взаємодії 25
2.1.5 Адаптивність та UX 27
2.2 Аналіз та обґрунтування вибору технологій 27
2.2.1 Firebase Firestore 28
2.2.2 Firebase Authentication 29
2.2.3 React.js 31
2.2.4 Tailwind CSS 33
2.2.5 Node.js та Express.js 35
2.2.6 WebSocket 36
2.2.7 TradingView 37
2.2.8 Binance API 39
2.2.9 CoinGecko API 40

3 Проєктування вебдодатку 42
3.1 Загальна архітектура системи 42
3.2 Схема бази даних 44
3.3 Компонентна структура клієнтської частини 45
3.4 Проєктування UI/UX 47
3.5 Логіка обробки даних 48
4 Програмна реалізація та тестування 51
4.1 Реалізація основних функцій вебдодатку 52
4.2 Забезпечення безпеки та авторизації користувачів 55
4.3 Тестування функціональності та продуктивності 57
Висновки 59
Перелік посилань 60
Додаток А Інтерфейс вебдодатку 62
Додаток Б Структура бази даних Firestore 68
Додаток В Тестування вебдодатку 71

ВСТУП

Сучасний світ характеризується стрімким розвитком інформаційних технологій, зокрема цифрових фінансів, серед яких особливе місце посідають криптовалюти. Вони стали не лише інструментом інвестування, а й самостійною сферою економічної діяльності. Зі зростанням популярності цифрових активів виникає потреба у зручних та ефективних засобах моніторингу та управління криптовалютними портфелями. Саме тому створення вебдодатку для аналізу криптовалютних портфелів є актуальним напрямом розробки в межах фінансових технологій.

Актуальність криптовалют зумовлена їхньою децентралізованістю, високою волатильністю та потенціалом отримання прибутку. Однак для ефективного управління цифровими активами необхідно оперативно відстежувати ринкову вартість, баланс портфеля, зміну курсу валют у реальному часі та інші важливі показники. Традиційні інструменти, як-от електронні таблиці або біржові сайти, не завжди забезпечують достатній рівень автоматизації та зручності. Вебдодатки ж надають інтерактивні засоби роботи з даними, дозволяючи в реальному часі аналізувати стан інвестицій, будувати графіки змін вартості та швидко реагувати на ринкові коливання [3].

Мета роботи: розробити вебдодаток для зручного моніторингу, зберігання, аналізу та візуалізації змін вартості криптовалютного портфеля користувача.

Досягнення поставленої мети передбачає реалізацію наступних завдань:

- проаналізувати існуючі рішення для моніторингу криптовалют;
- дослідити архітектуру та принципи роботи сучасних вебдодатків;
- розробити структуру додатку з урахуванням вимог до UI/UX;
- спроєктувати архітектуру клієнтської та серверної частини;
- реалізувати функціонал підключення до WebSocket API для отримання актуальних курсів криптовалют;
- протестувати додаток у різних сценаріях використання.

Об’єкт дослідження – розробка вебдодатоку для зручного моніторингу та аналізу криптовалютних активів користувача з можливістю візуалізації динаміки вартості та оцінки прибутковості портфеля.

Предмет дослідження – програмні засоби та технології розробки вебдодатку для збору, обробки та візуалізації даних криптовалютного портфеля в режимі реального часу.

Методи дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання, проєктування, конструювання та тестування програмного забезпечення.

Отримані результати мають практичну цінність, оскільки можуть бути використані як основа для реального комерційного продукту або інтегровані в існуючі фінансові платформи. Розроблений додаток дозволяє користувачам зручно відслідковувати стан свого криптовалютного портфеля, виявляти тенденції, приймати зважені рішення на основі аналітики в реальному часі.

Кваліфікаційна робота складається зі вступу, чотирьох розділів, висновків, переліку використаних джерел та додатків. Загальний обсяг роботи становить 72 сторінки. Додатки викладено на 62-72 сторінках. У переліку посилань наведено 19 джерел.

Перший розділ містить постановку задачі, обґрунтування актуальності теми та аналіз сучасних інструментів для моніторингу криптовалют.

Другий розділ присвячено теоретичним основам і обґрунтуванню вибору технологій для реалізації вебдодатку.

У третьому розділі описано структуру, архітектуру та інтерфейс користувача розроблюваного додатку.

Четвертий розділ подає реалізацію програмного забезпечення, тестування функціоналу та аналіз результатів.

1 ПОСТАНОВКА ЗАДАЧІ ТА АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність теми

У сучасному цифровому суспільстві фінансові інструменти, засновані на блокчейн-технологіях, відіграють дедалі важливішу роль. Криптовалюти вже давно перестали бути лише технічною новинкою й перетворилися на повноцінний актив, що використовується у торгівлі, інвестуванні та розрахунках. Разом із цим зростає кількість користувачів, які володіють портфелем із кількох видів криптовалют. Це вимагає наявності інструментів, які дозволяють ефективно відстежувати зміну курсу, структуру активів і загальну прибутковість.

Традиційні інструменти, зокрема електронні таблиці чи блокноти для запису вручну, мають суттєві обмеження: низький рівень автоматизації, відсутність візуалізації динаміки змін, потребу в ручному оновленні інформації. Зі зростанням обсягів активів і частоти змін курсу криптовалют стає очевидною необхідність створення інтерактивного рішення, яке дозволяє об’єднати облік, аналітику та візуалізацію криптопортфеля.

Крім того, нестабільність крипторинку вимагає швидкої реакції на зміни, що неможливо реалізувати за допомогою ручного або напівавтоматизованого контролю. Таким чином, виникає потреба в інтерактивному вебдодатку, який би поєднував доступ до актуальних ринкових даних, зручний інтерфейс і функції глибокої аналітики.

1.2 Вимоги до вебдодатку

Вебдодаток для моніторингу криптовалютного портфеля повинен відповідати сучасним вимогам до продуктивності, зручності, адаптивності та функціональності. Розглянемо що включають основні вимоги.

Можливість зберігання криптовалют у віртуальному гаманці: користувач має змогу створити віртуальний гаманець, додавати криптовалюти, контролювати баланс кожної монети, бачити історію змін вартості в контексті загального портфеля (див. рис. 1.1).

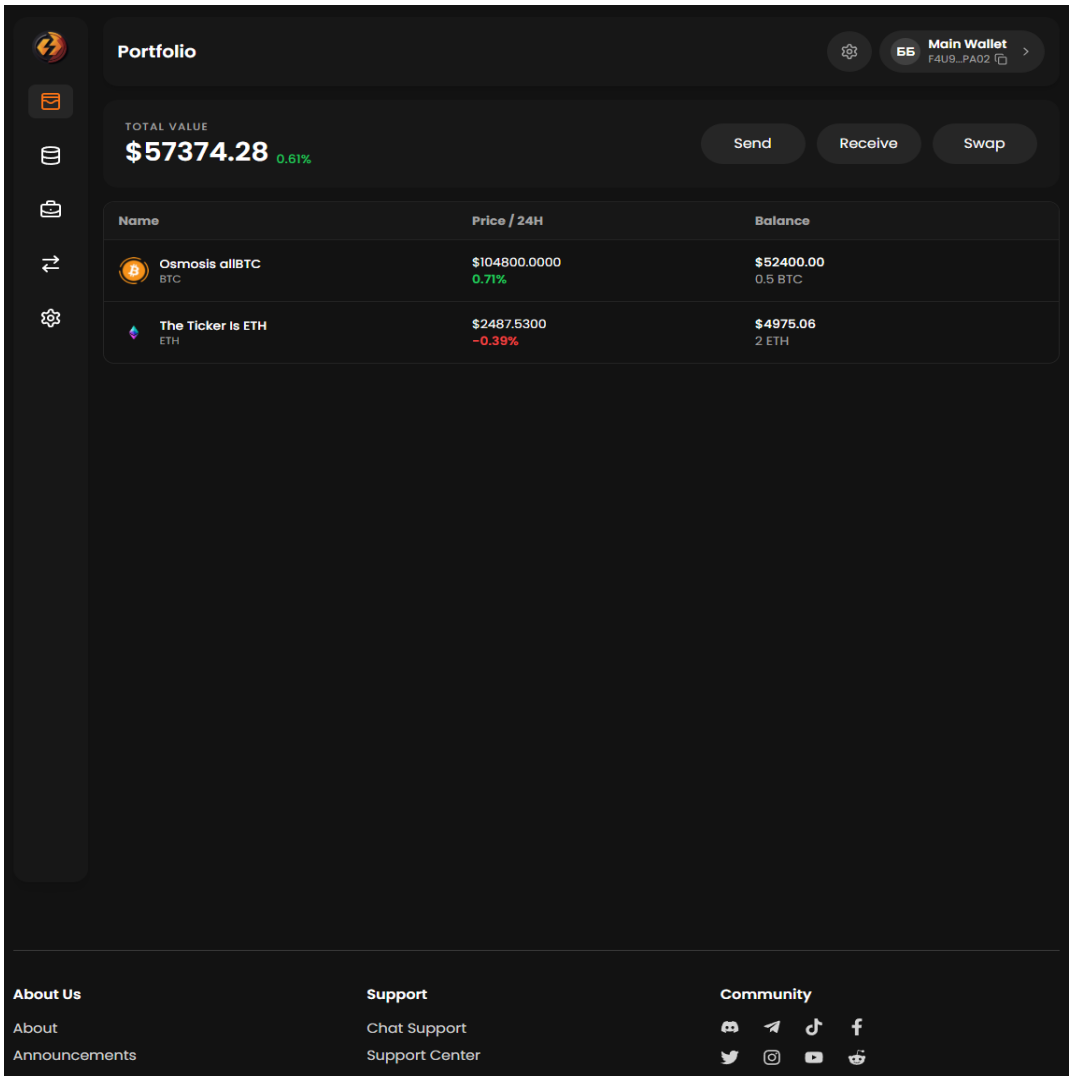


Рисунок 1.1 – Інтерфейс віртуального гаманця у вебдодатку

Актуальні ринкові дані – необхідне інтегрування з API бірж або моніторингових сервісів (CoinGecko, CoinMarketCap, Binance), що дозволить отримувати інформацію про курс у реальному часі (див. рис. 1.2).

Функціонал аналітики – обчислення загального прибутку/збитку, середньої ціни входу, зміни капіталізації.

Безпечний вхід і зберігання даних – реалізація механізмів реєстрації, авторизації та захисту даних користувача.

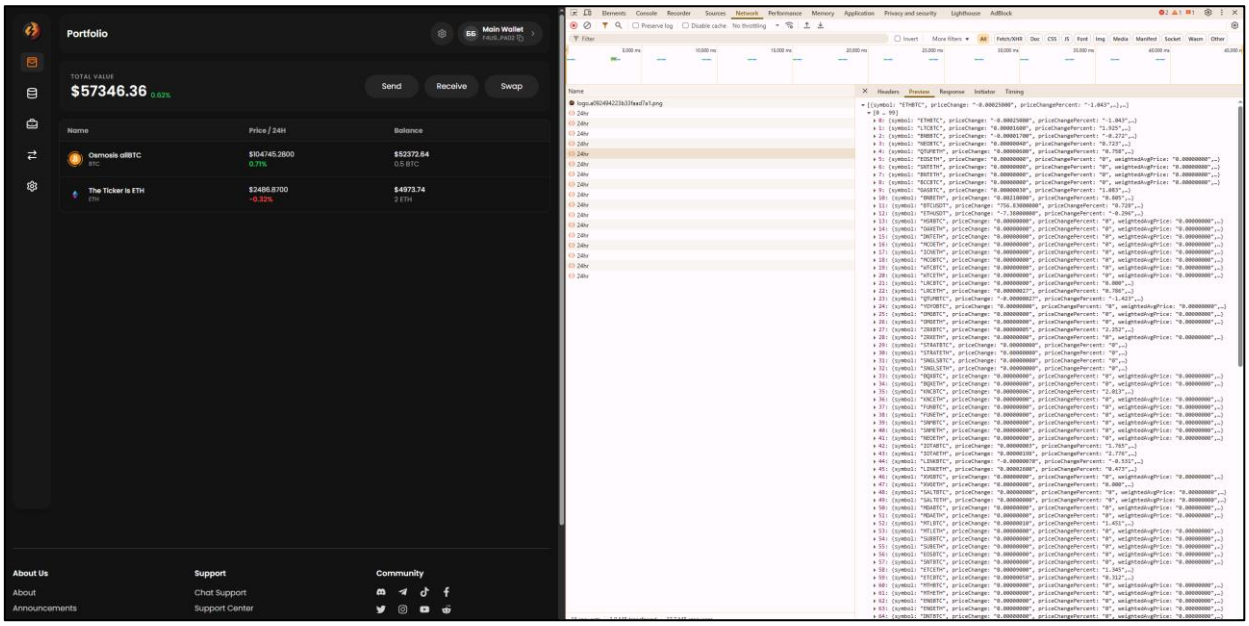


Рисунок 1.2 – JSON-відповіді API Binance із даними про криптовалюту

Адаптивний дизайн – коректне відображення інтерфейсу як на десктопах, так і на мобільних пристроях (див. рис. 1.3).

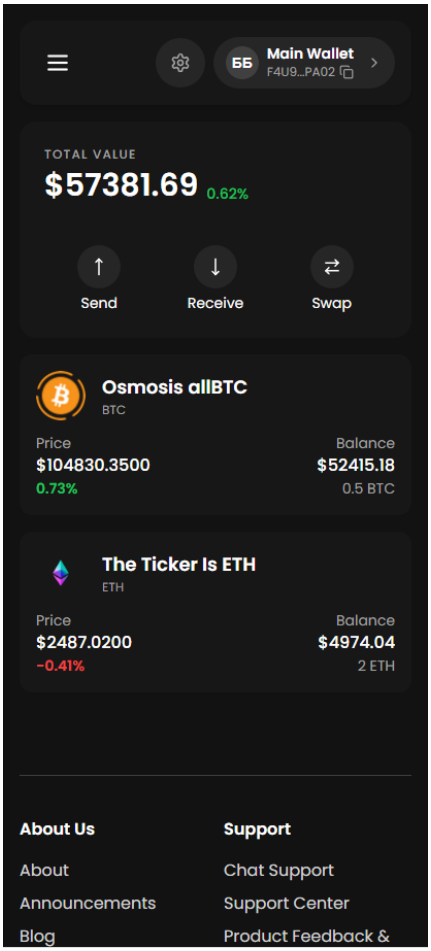


Рисунок 1.3 – Адаптивний макет інтерфейсу вебдодатку на різних пристроях

Побудова графіків і динаміки вартості – реалізація засобів для візуального представлення змін вартості окремих активів, прибутковості тощо (див. рис. 1.4).

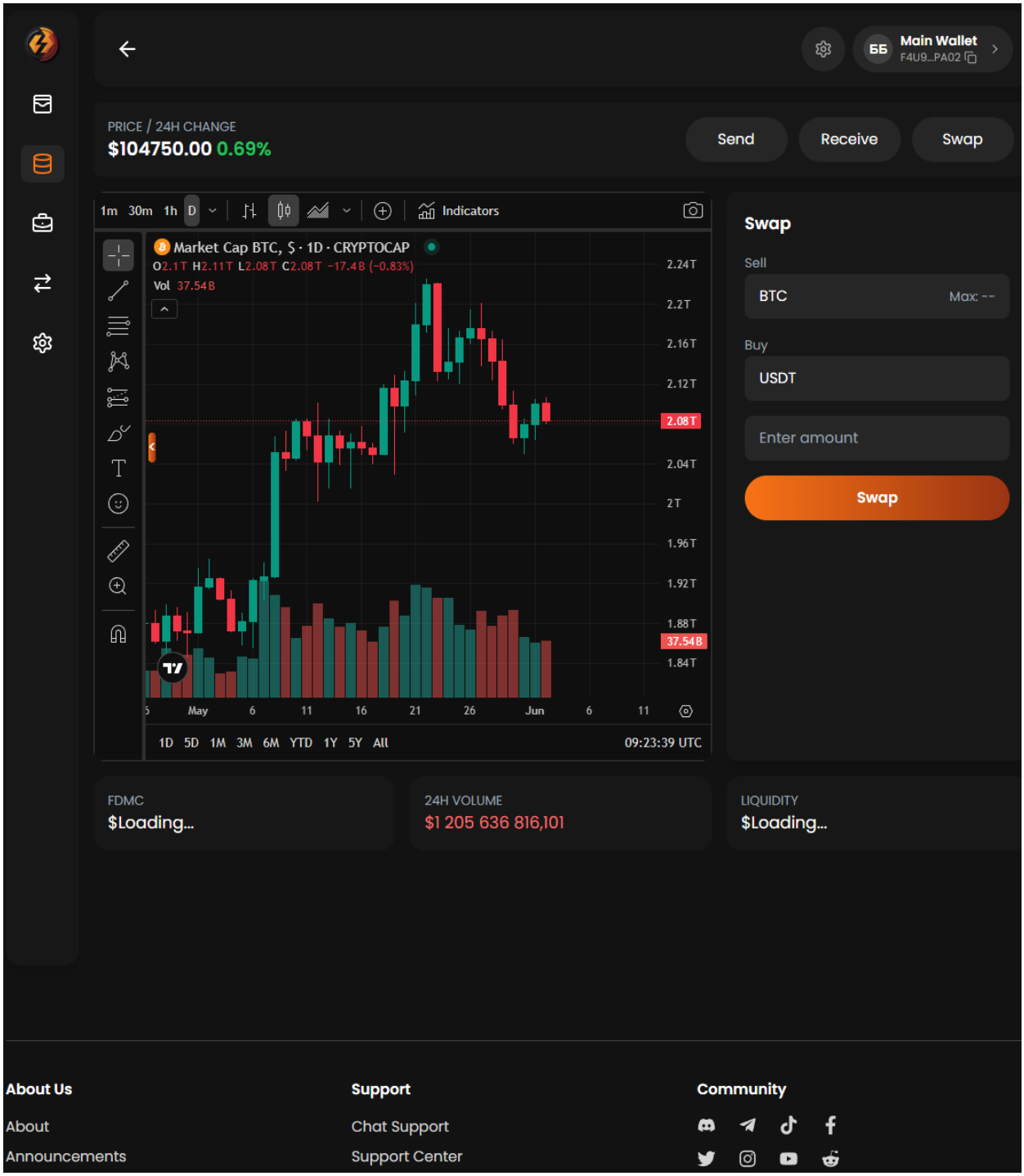


Рисунок 1.4 – Графік зміни вартості монети

Ці вимоги забезпечують зручність використання системи широким колом користувачів та відповідність сучасним стандартам веброзробки.

1.3 Огляд існуючих рішень та їх аналіз

Аналіз свідчить, що на ринку криптовалютних сервісів існує велика кількість платформ, які надають можливості моніторингу, обліку, аналізу та торгівлі цифровими активами.

Нині існує низка сервісів і додатків, що пропонують функції моніторингу криптоактивів. Серед найвідоміших – CoinMarketCap, Binance, Coinbase та CoinGecko (див. табл. 1.1).

Таблиця 1.1 – Порівняльна таблиця можливостей існуючих сервісів

Характеристика	CoinMarketCap	Binance	Coinbase	CoinGecko
Тип платформи	Агрегатор	Біржа + гаманець	Біржа + гаманець	Агрегатор
Створення портфеля	Обмежене	Повне	Повне	Обмежене
Дані в реальному часі	Так	Так	Так	Так
Графіки та аналітика	Частково	Розширено	Обмежено	Частково
Підтримка API	Так	Так	Так	Так
Мобільний застосунок	Так	Так	Так	Так
Безпека доступу	Стандартна	Висока (2FA)	Висока (2FA)	Стандартна

CoinMarketCap – один із найвідоміших агрегаторів криптовалютних ринкових даних.

Платформа дозволяє переглядати вартість монет у реальному часі, ринкову капіталізацію, обсяги торгів, динаміку цін, доступ до новин і аналітики. Проте функціональність створення власного портфеля є обмеженою, а головний акцент зроблено на інформуванні користувача (див. рис. 1.5).

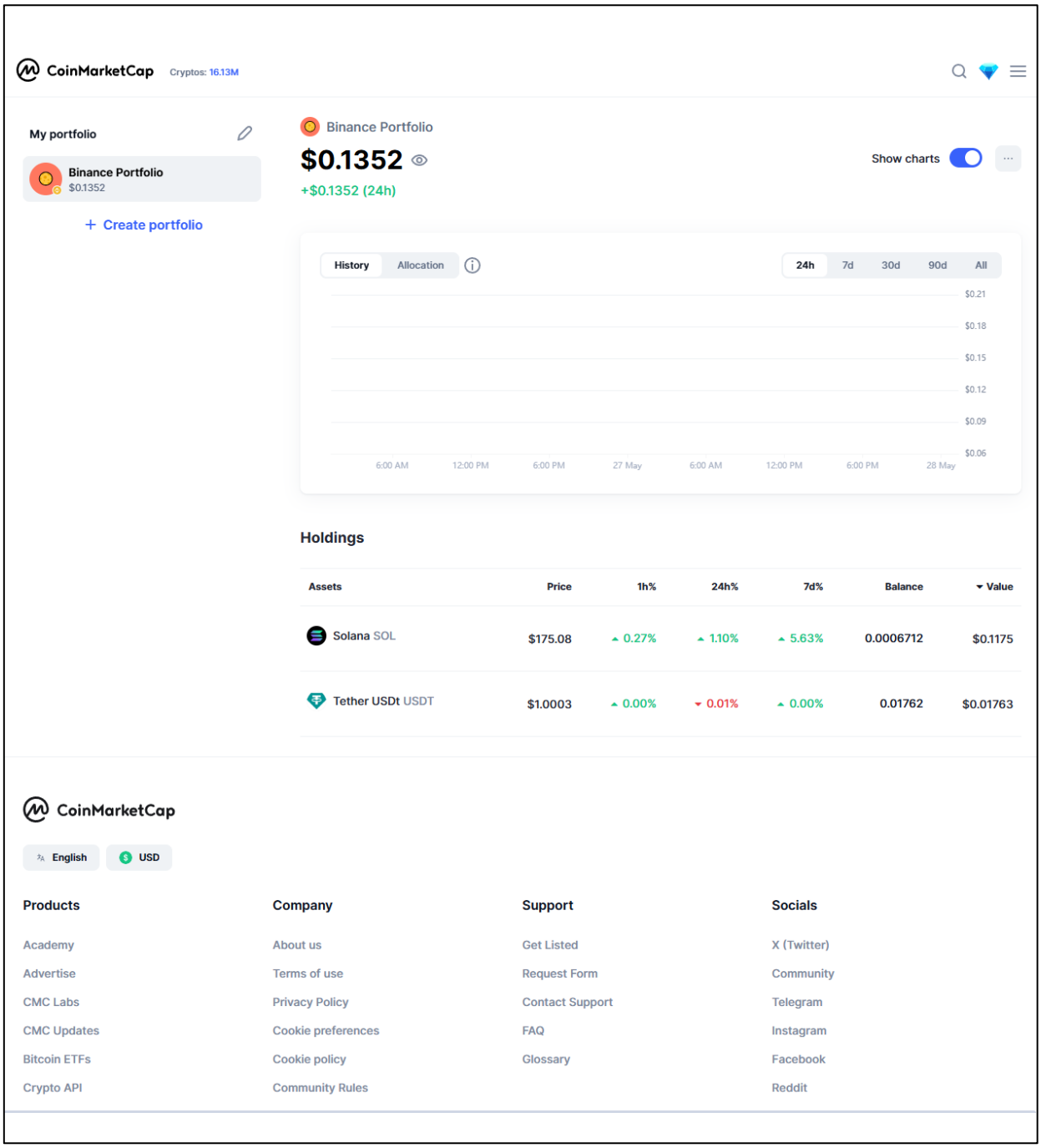


Рисунок 1.5 – Інтерфейс CoinMarketCap

Binance – це не лише криптобіржа, а й повноцінна екосистема для трейдингу, зберігання та моніторингу активів. Користувачі можуть створювати портфелі автоматично через торговий обліковий запис, аналізувати динаміку свого балансу, історію транзакцій. Вебінтерфейс і мобільний додаток надають розширену аналітику, однак повна функціональність доступна тільки зареєстрованим користувачам (див. рис. 1.6).

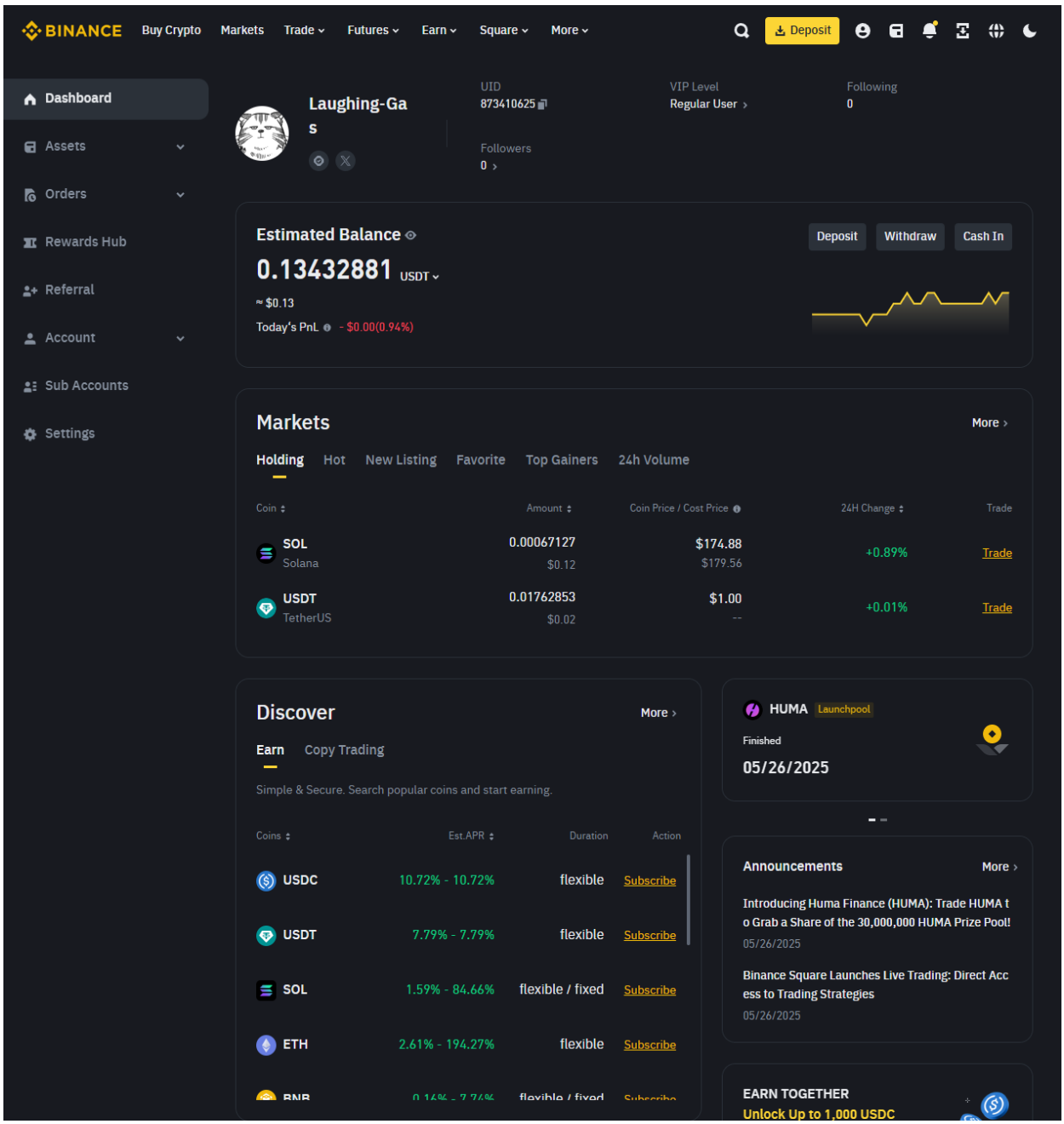


Рисунок 1.6 – Інтерфейс Binance

Coinbase – одна з найпопулярніших криптовалютних платформ у США. Платформа має простий інтерфейс і високий рівень безпеки. Coinbase надає користувачам можливість зберігати криптовалюти, відстежувати зміни курсу, аналізувати структуру портфеля. Проте можливості візуалізації аналітики обмежені в безкоштовному доступі (див. рис. 1.7).

CoinGecko – альтернатива CoinMarketCap з відкритим API та більшою прозорістю. CoinGecko також дозволяє переглядати статистику ринку, додавати монети до обраного списку, використовувати API для сторонніх

застосунків. Однак функції повноцінного портфеля для інвестора обмежені (див. рис. 1.8).

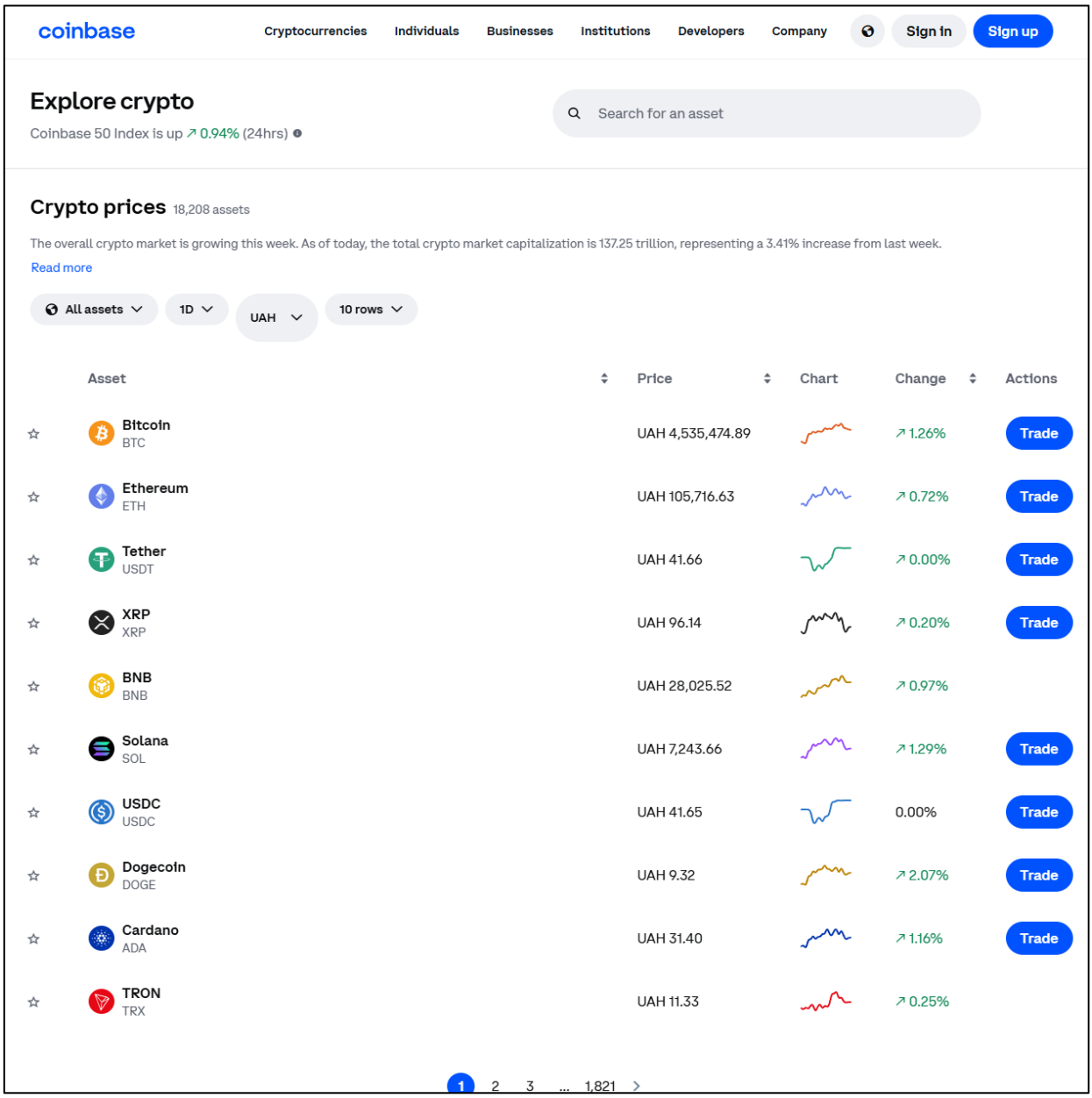


Рисунок 1.7 – Інтерфейс Coinbase

Аналіз показує, що попри наявність численних рішень, користувачі часто потребують персоналізованого, гнучкого та доступного рішення, яке можна адаптувати під свої інвестиційні потреби без обмежень чи залежності від сторонніх сервісів.

Отже, доцільним є створення власного вебдодатку, який поєднує в собі гнучкість, масштабованість, інтерактивність і простоту використання.

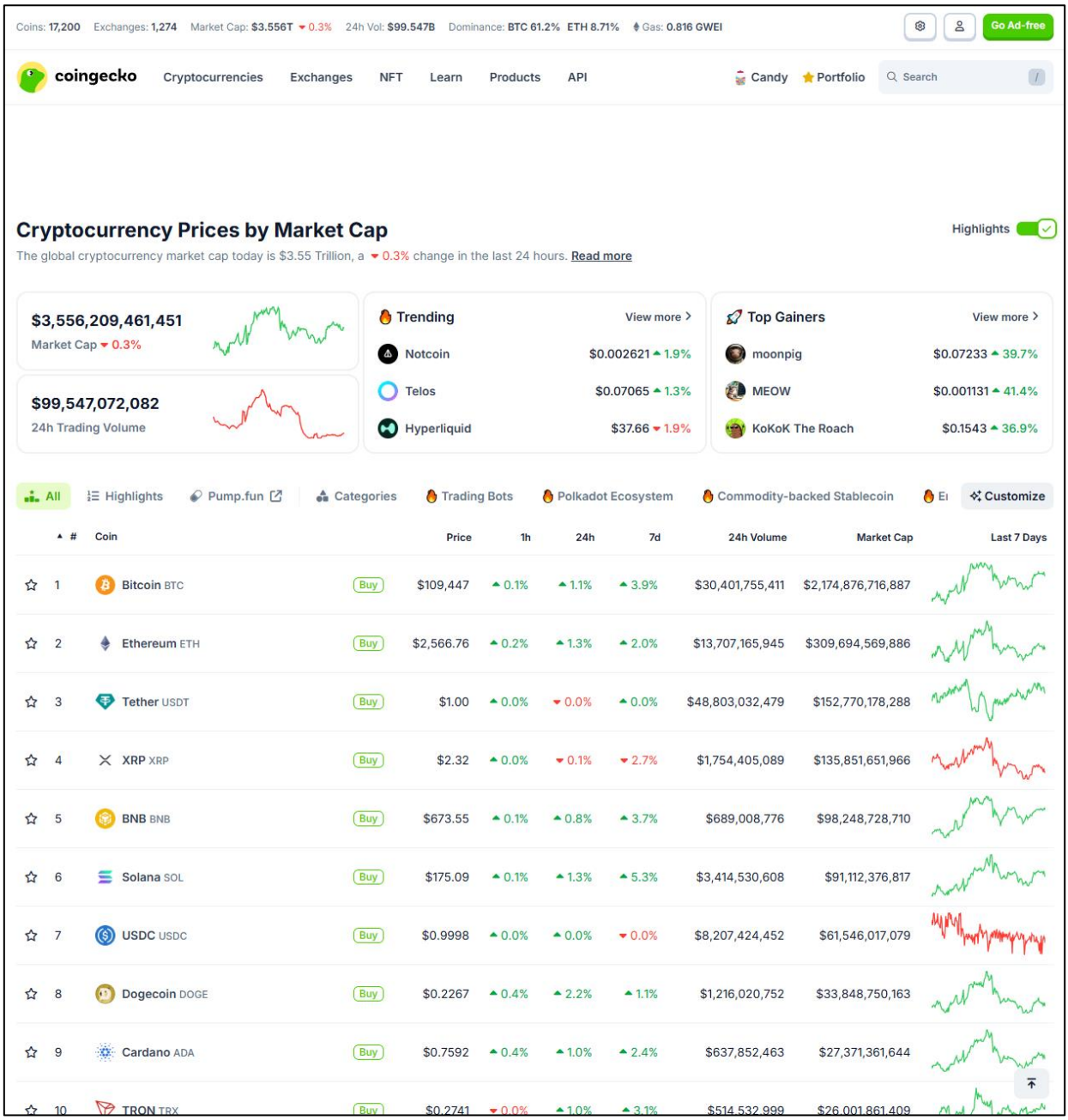


Рисунок 1.8 – Інтерфейс CoinGecko

1.4 Постановка задачі

Метою цієї роботи є розробка функціонального вебдодатку, який дозволить користувачу зберігати криптовалюти у віртуальному гаманці, переглядати їхню поточну вартість у режимі реального часу, аналізувати прибутковість, а також візуалізувати ринкові дані для прийняття обґрунтованих рішень.

Для досягнення цієї мети необхідно виконати такі основні завдання:

– здійснити аналіз предметної області та сформулювати вимоги до майбутнього програмного забезпечення;
– дослідити технології для створення клієнтської та серверної частини;
– розробити архітектуру вебдодатку та логіку зберігання і обробки даних;
– реалізувати модулі: віртуальний гаманець, оновлення курсів через API, графіки, аналітику;
– забезпечити авторизацію та безпечне збереження персональних даних користувача;
– провести тестування функціональності системи у різних сценаріях використання;
– оцінити перспективи подальшого розвитку додатку.

1.5 Очікувані результати

У результаті виконання кваліфікаційної роботи очікується створення повноцінного вебдодатку, який має такі функціональні можливості:

– створення та керування віртуальним криптогаманцем користувача;
– інтеграція з зовнішніми API для отримання актуальної ринкової інформації (див. рис. 1.9);
– візуалізація динаміки зміни курсу криптовалют за допомогою інтерактивних графіків;
– аналітичні модулі для підрахунку прибутковості, оцінки змін портфеля, розподілу активів;
– захищена реєстрація та автентифікація користувачів;
– адаптивний інтерфейс для різних пристроїв.

```
const COINGECKO_API = "https://api.coingecko.com/api/v3/coins/list";
const COINGECKO_MARKETS = "https://api.coingecko.com/api/v3/coins/markets";
const BINANCE_WS_URL = "wss://stream.binance.com:9443/ws/!ticker@arr";
```

Рисунок 1.9 – Інтеграція з зовнішнім API

Розроблений додаток може бути використаний як приватними інвесторами, так і в складі ширших фінансових платформ. Він закладає основу для масштабування проєкту в напрямку трейдингу, сповіщень про зміни курсу або підключення до біржових акаунтів.

2 ТЕОРЕТИЧНІ ОСНОВИ ТА ВИБІР ТЕХНОЛОГІЙ

На етапі розробки будь-якого програмного забезпечення надзвичайно важливим є правильний вибір інструментів, мов програмування, бібліотек та сервісів, які будуть використані для реалізації архітектури системи. Від цього залежить не лише продуктивність та масштабованість додатку, але й ефективність його підтримки, гнучкість при розширенні функціоналу та зручність для кінцевого користувача.

У рамках цієї роботи проаналізовано основи веброзробки, розглянуто типову структуру вебдодатків, а також обґрунтовано вибір відповідних технологій для реалізації функціональних компонентів системи моніторингу криптовалютного портфеля.

Зокрема, у розділі розглядаються: основи вебархітектури; роль клієнтської та серверної частин; обробка даних у реальному часі; безпечна автентифікація; технології баз даних, адаптивний інтерфейс та засоби візуалізації. Усі вибрані інструменти були підібрані з урахуванням вимог до швидкодії, інтерактивності, безпеки та зручності використання, які є критичними для фінансових вебрішень.

У цьому розділі буде розглянуто як загальнотеоретичні засади веброзробки, так і конкретні технічні рішення, що застосовуються у створенні системи: Firebase (Authentication та Firestore), React.js, Tailwind CSS, Node.js з Express, WebSocket, API від Binance і CoinGecko, а також бібліотеку TradingView для візуалізації графіків.

2.1 Основи веброзробки

Веброзробка – це процес створення веборієнтованих інформаційних систем, які функціонують за клієнт-серверною архітектурою та доступні через

інтернет. Основу веброзробки становлять три ключові компоненти: клієнтська частина (frontend), серверна частина (backend) і база даних. У сучасних вебдодатках також застосовуються зовнішні API, протоколи передачі даних у реальному часі (WebSocket), хмарні сервіси та бібліотеки для візуалізації інформації.

2.1.1 Клієнтська частина

Клієнтська частина відповідає за інтерфейс користувача – те, що бачить і з чим взаємодіє користувач у браузері. До основних технологій клієнтської частини належать:

- HTML (HyperText Markup Language) – розмітка структури сторінки;
- CSS (Cascading Style Sheets) – стилізація елементів;
- JavaScript – логіка, динамічна поведінка елементів, тощо;
- фреймворки й бібліотеки – React, Vue, Angular та інші, які спрощують побудову інтерфейсу.

У роботі для створення динамічного, адаптивного інтерфейсу використовується React.js у поєднанні з Tailwind CSS.

2.1.2 Серверна частина

Серверна частина керує бізнес-логікою, обробкою запитів, взаємодією з базою даних та сторонніми API. Вона також відповідає за автентифікацію, зберігання й обробку даних користувачів.

Для побудови серверної логіки часто використовують: Node.js (JavaScript-середовище виконання на сервері), Express.js (фреймворк для побудови REST API) і мови програмування (Python, Java, PHP, Ruby – залежно від проєкту і вимог до нього).

Node.js – це середовище виконання JavaScript на сервері, засноване на рушії V8 від Google Chrome. Його основною перевагою є подієво-орієнтована, неблокуюча модель обробки запитів, яка забезпечує високу продуктивність при роботі з великою кількістю одночасних підключень. Завдяки Node.js розробники можуть використовувати одну мову програмування – JavaScript – як для клієнтської, так і для серверної частини додатку, що значно спрощує розробку та підтримку проєкту [14].

Node.js використовується для реалізації:

– обробки HTTP-запитів;
– підключення до бази даних;
– реалізації логіки автентифікації;
– інтеграції з API сторонніх сервісів;
– обробки WebSocket-з’єднань у реальному часі.

Express.js – це мінімалістичний і гнучкий вебфреймворк для Node.js, який полегшує створення серверних додатків і REST API. Express дозволяє організовувати маршрутизацію запитів, налаштовувати middleware (проміжні обробники), керувати сесіями, а також легко масштабувати серверну частину додатку [7].

Сумісне використання Node.js і Express.js дає змогу швидко створювати ефективні серверні застосунки, що легко інтегруються з клієнтською частиною через API. Це особливо актуально для систем, які працюють з динамічними даними, як-от криптовалютні платформи, де важлива швидка обробка запитів, безпека та можливість масштабування.

Node.js – це середовище виконання JavaScript на сервері, що дозволяє будувати масштабовані додатки. У поєднанні з Express.js воно забезпечує гнучкий інструмент для побудови API, обробки запитів та реалізації серверної логіки. Перевагою є можливість використання єдиної мови програмування як на клієнті, так і на сервері.

У роботі обрана технологія Node.js для обробки запитів клієнта, підключення до бази даних та реалізації WebSocket-з’єднань.

2.1.3 База даних

Бази даних зберігають усю необхідну інформацію: акаунти користувачів, криптовалютні портфелі, історію змін, налаштування тощо. У веброзробці бази даних поділяються на два основні типи – реляційні (SQL) та нереляційні (NoSQL). Розуміння цих підходів є важливим для вибору технології зберігання даних у вебдодатках.

Реляційні бази даних зберігають інформацію у вигляді таблиць з чітко визначеною схемою. Вони базуються на математичній теорії реляцій і використовують мову SQL (Structured Query Language) для управління даними. Основними особливостями є:

– жорстка структура таблиць (колонки з визначеними типами);
– підтримка складних зв’язків між таблицями (foreign keys);
– транзакції та цілісність даних.

Найпопулярніші представники – MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

Нереляційні бази даних (NoSQL) розроблені для роботи з великими обсягами неструктурованих або слабо структурованих даних. Вони не мають фіксованої схеми та підтримують різні моделі зберігання: документоорієнтовану, графову, колонкову, ключ-значення. Характерні особливості:

– гнучка структура даних (JSON, BSON);
– висока масштабованість і швидкодія;
– відсутність суворих зв’язків між об’єктами;
– придатність для горизонтального масштабування.

Найпоширеніші системи – MongoDB, Firebase Firestore, CouchDB, Cassandra.

При реалізації додатку було обрано Firebase Firestore, яка дозволяє зберігати дані у вигляді документів у колекціях, підтримує синхронізацію в реальному часі й тісно інтегрується з іншими сервісами Firebase.

2.1.4 Механізми взаємодії

Ефективна взаємодія між клієнтською та серверною частинами є критично важливою складовою сучасного вебдодатку. Вона забезпечує обмін даними, синхронізацію інформації, ініціювання дій, авторизацію користувачів і багато іншого. У межах вебдодатку для моніторингу криптовалют використовуються такі ключові механізми як REST API, JSON і WebSocket.

REST (Representational State Transfer) – це архітектурний стиль, який дозволяє створювати гнучкі та масштабовані API. Кожен ресурс має унікальну адресу (URL), і до нього можна звертатися через стандартні HTTP-запити [1].

Переваги REST API:

– простота реалізації;
– незалежність між клієнтом і сервером;
– можливість кешування;
– легка масштабованість.

У вебдодатку REST API використовується для:

– створення, оновлення, видалення даних портфеля;
– реєстрації та входу користувача;
– взаємодії з базою даних (Firestore).

WebSocket – це протокол, що забезпечує постійне двостороннє з’єднання між клієнтом і сервером. На відміну від REST API, де клієнт сам ініціює кожен запит, WebSocket дозволяє серверу надсилати повідомлення клієнту автоматично у будь-який момент часу.

Переваги WebSocket:

– мінімальні затримки;
– зменшене навантаження на сервер;
– актуальні дані в режимі реального часу.

У вебдодатку WebSocket використовується для отримання оновлених курсів криптовалют безперервно, що дозволяє користувачу бачити зміни вартості активів миттєво.

JSON (JavaScript Object Notation) – формат представлення структурованих даних, який є надзвичайно легким для читання людиною і простим у обробці мовами програмування (особливо JavaScript). У вебдодатку JSON використовується для:

– передачі даних між клієнтом і сервером;
– збереження інформації у Firebase Firestore (див. рис. 2.1);
– обробки відповіді від Binance API (див. рис. 2.2).

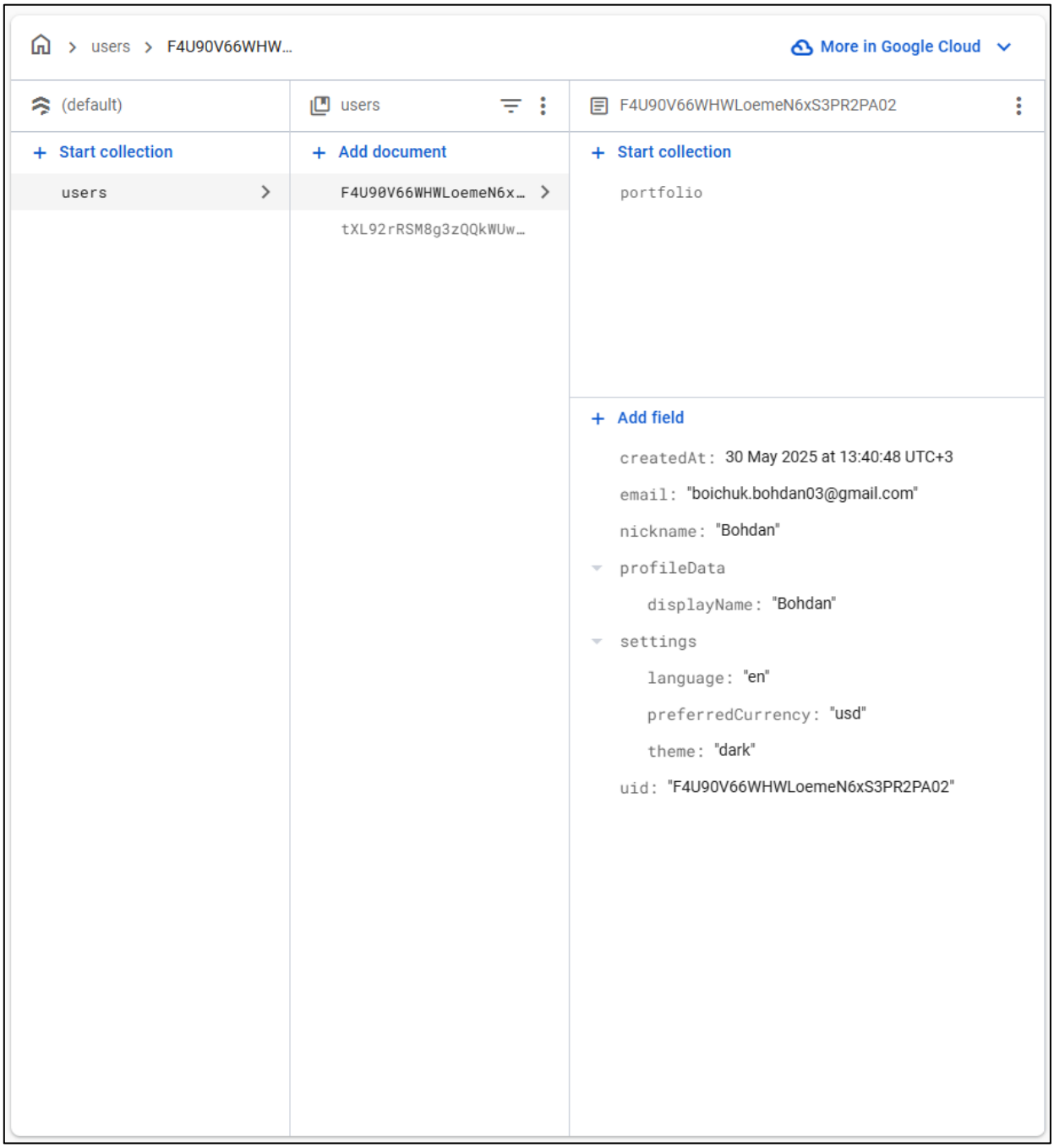


Рисунок 2.1 – Збережені документи у Firebase Firestore

```
useEffect(() => {
  const ws = new WebSocket(BINANCE_WS_URL);

  Windsurf: Refactor | Explain | Generate JSDoc | ×
  ws.onmessage = (event) => {
    try {
      const updates = JSON.parse(event.data);
      updates.forEach((u) => {
        if (u.s.endsWith("USDT")) {
          const symbol = u.s.replace("USDT", "");
          pricesRef.current[symbol] = {
            price: parseFloat(u.c),
            change: parseFloat(u.P),
          };
        }
      });
    } catch (err) {
      console.error("WebSocket error:", err);
    }
  };
```

Рисунок 2.2 – Обробка відповіді від Binance API

Використання вказаних механізмів – REST API, WebSocket і JSON – забезпечує ефективну, масштабовану та безпечну взаємодію між компонентами вебдодатку. Це дозволяє досягнути високої швидкодії, гнучкості, а також створити зручне середовище для роботи кінцевого користувача в умовах криптовалютного ринку.

2.1.5 Адаптивність та UX

З урахуванням різноманіття пристроїв вебдодаток повинен підтримувати адаптивну верстку (responsive design). Це забезпечується використанням фреймворків, таких як Tailwind CSS, що дозволяють швидко налаштувати інтерфейс для мобільних і десктопних пристроїв.

2.2 Аналіз та обґрунтування вибору технологій

Розробка вебдодатку для моніторингу криптовалютного портфеля потребує вибору інструментів, які забезпечують високу швидкодію, гнучкість, масштабованість та безпечність обробки даних.

2.2.1 Firebase Firestore

Firebase Firestore – це хмарна NoSQL-база даних, яка дозволяє зберігати документи у вигляді колекцій (див. рис. 2.3). У контексті вебдодатку вона використовується для зберігання інформації про користувачів, їхні криптовалютні портфелі, історію дій та налаштування [2, 9].

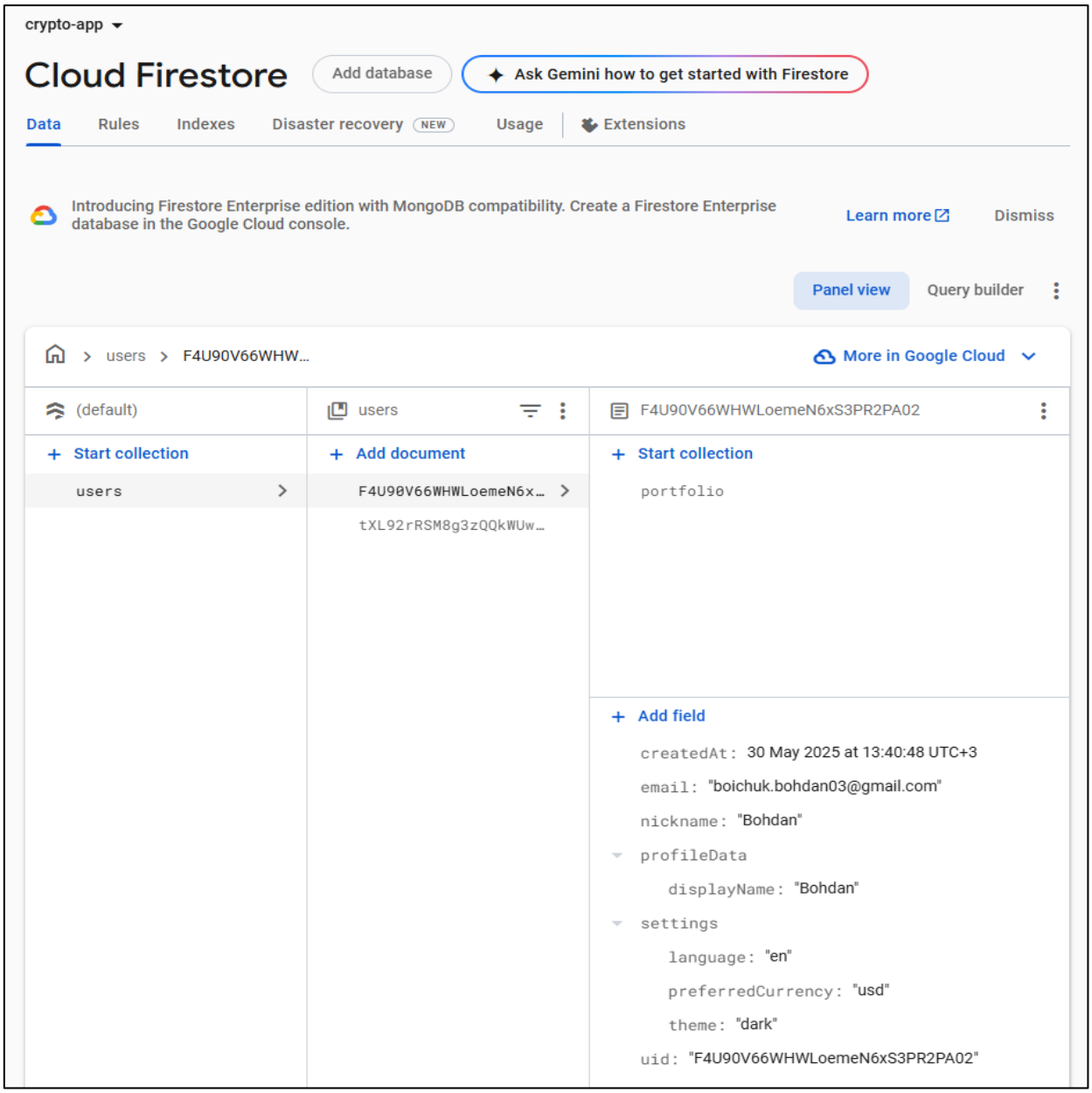


Рисунок 2.3 – Структура бази даних Firestore

Firestore підтримує гнучку структуру даних, яка дозволяє зберігати документи у вкладених колекціях. Це дозволяє легко організовувати складні

типи даних, наприклад: користувач – портфель – криптомонети. Firestore підтримує офлайн-доступ, автоматичне кешування та функціонал синхронізації в реальному часі між клієнтами, що особливо важливо у додатках, які вимагають миттєвого оновлення інтерфейсу.

Firestore також забезпечує контроль доступу до даних за допомогою правил безпеки (security rules), що дозволяє гнучко обмежити права на читання/запис документів в залежності від ролі або ID користувача.

Серед переваг вибору Firestore – легкість налаштування, швидке масштабування, інтеграція з Firebase Authentication та зручність обробки даних у форматі JSON. Для нашого додатку Firestore забезпечує стабільне зберігання персоналізованої інформації кожного користувача, підтримує реєстрацію змін у портфелі, дозволяє працювати як у реальному часі, так і в режимі офлайн.

Firestore підтримує структурований формат документів JSON, автоматичну синхронізацію в режимі реального часу, офлайн-доступ, масштабування та безпечний доступ через правила безпеки.

Технологія Firestore була обрана через зручність у роботі з React-додатками, інтеграцію з іншими сервісами Firebase, можливість зберігання даних з різною структурою та оновлення інтерфейсу в реальному часі не акцентуючи увагу на конструюванні баз даних.

2.2.2 Firebase Authentication

Firebase Authentication – це сервіс для реалізації автентифікації користувачів через email/пароль, Google, GitHub тощо (див. рис. 2.4, 2.5) [2, 8].

Firebase Authentication забезпечує надійну систему входу та керування користувачами. Підтримується кілька способів автентифікації: класична (email/пароль), OAuth-провайдери (Google, Facebook, GitHub), анонімна автентифікація. Сервіс автоматично обробляє керування сесіями, збереження токенів доступу, скидання пароля та перевірку електронної пошти.

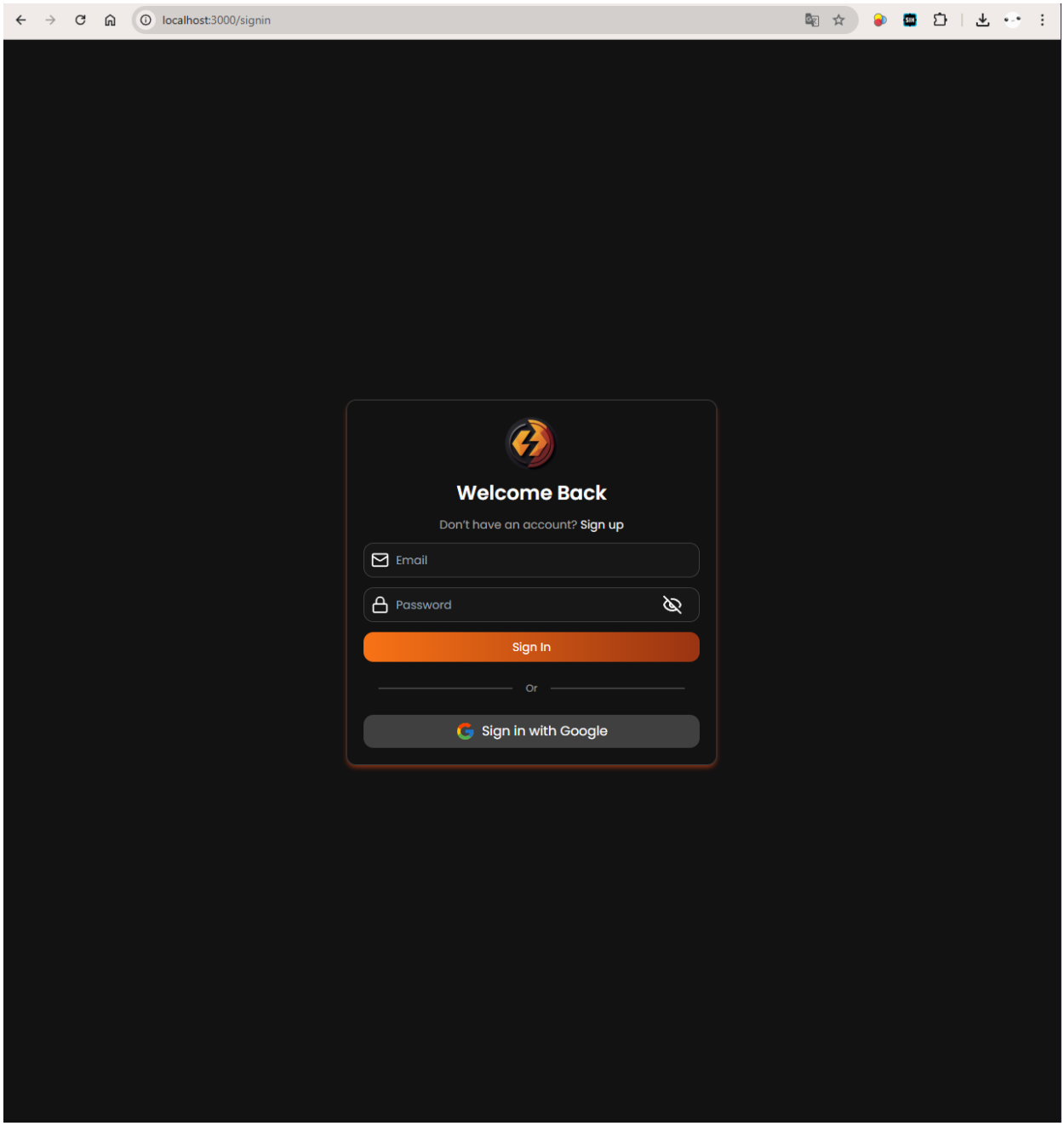


Рисунок 2.4 – Сторінка авторизації користувача у вебдодатку

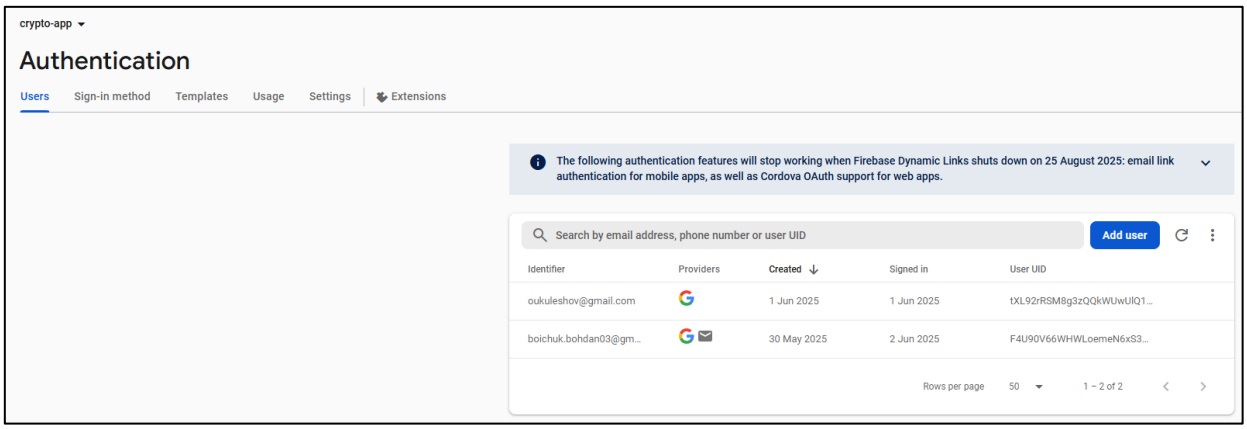


Рисунок 2.5 – Інтерфейс Firebase Authentication у Firebase Console

Всі ці функції доступні через консоль Firebase, що спрощує керування обліковими записами та інтеграцію в додаток.

Обрана технологія дозволяє значно зекономити час розробки, забезпечуючи безпечну авторизацію без потреби створювати власний бекенд. Простий API, доступний SDK для React, а також інтеграція з Firestore дозволяють швидко реалізувати персоналізований доступ користувача до своїх криптоактивів. Забезпечує керування сесіями, відновлення паролю, перевірку email, авторизацію через сторонні провайдери.

Вибір даної технології обґрунтовано простотою інтеграції, безпека, підтримка кількох способів входу без створення власної інфраструктури автентифікації.

2.2.3 React.js

React – це бібліотека JavaScript для створення інтерфейсів користувача (див. рис. 2.6, 2.7). Вона забезпечує побудову UI на основі компонентів із використанням Virtual DOM [13, 15, 16].

React дозволяє розробляти односторінкові додатки (SPA), у яких зміна контенту відбувається без перезавантаження сторінки.

Ключовими концепціями React є компоненти, props (властивості) та state (стан).

Бібліотека реалізує реактивний підхід до розробки – коли зміни в стані автоматично відображаються в DOM. Використовується JSX – розширення синтаксису JavaScript, яке дозволяє писати розмітку всередині коду.

React обрано для реалізації інтерфейсу криптовалютного додатку завдяки високій продуктивності, активній спільноті, можливості розділення логіки на окремі компоненти та великій кількості супутніх бібліотек. Інтерфейс користувача, побудований на React, є адаптивним, швидким та зручним як для настільних, так і для мобільних пристроїв.

Реалізує архітектуру компонентів, дозволяє створювати SPA (Single Page Applications), забезпечує повторне використання коду та швидке оновлення DOM.

Дана технологія використовується через продуктивність, підтримку великою спільнотою, гнучкість у розробці та зручну інтеграцію з іншими бібліотеками.



Рисунок 2.6 – Структура компонента React у редакторі коду

src > pages > Dashboard > Portfolio.jsx > ...

```jsx
import Footer from "../../components/common/Footer.jsx";
import Sidebar from "../../components/Dashboard/Sidebar.jsx";
import TopBar from "../../components/Dashboard/TopBar.jsx";
import { useAuth } from "../../contexts/AuthContext.jsx";
import PortfolioOverview from "../../components/Dashboard/PortfolioOverview.jsx";
import PortfolioBalance from "../../components/Dashboard/PortfolioBalance.jsx";
import { useState } from "react";

Windsurf: Refactor | Explain | Generate JSDoc | ×
const Portfolio = () => {
    const { currentUser } = useAuth();
    const [mobileMenuOpen, setMobileMenuOpen] = useState(false);
    Windsurf: Refactor | Explain | Generate JSDoc | ×
    const toggleMobileMenu = () => {
      setMobileMenuOpen((prev) => !prev);
    };

    return (
      <div className="max-w-6xl mx-auto p-4">
        <div className="flex gap-4">
          <Sidebar
            mobileOpen={mobileMenuOpen}
            onCloseMobile={() => setMobileMenuOpen(false)}
          />
          <div className="flex flex-col flex-1">
            <TopBar user={currentUser} onBurgerClick={toggleMobileMenu} />
              <div className="flex-1 overflow-y-auto pt-4">
                <PortfolioBalance />
                <PortfolioOverview />
              </div>
          </div>
        </div>
        <Footer />
      </div>
    );
};

export default Portfolio;
```

Рисунок 2.7 – Фрагмент інтерфейсу вебдодатку, реалізованого на React

2.2.4 Tailwind CSS

Tailwind CSS – це утилітарний CSS-фреймворк, який дозволяє швидко створювати адаптивні інтерфейси з допомогою готових класів (див. рис. 2.8) [17].

Tailwind реалізує підхід “utility-first”, який передбачає використання атомарних CSS-класів для кожної властивості стилю – кольору, відступу, розміру, шрифтів тощо. Це дозволяє стилізувати компоненти без потреби

писати власні CSS-файли або створювати глобальні стилі. Tailwind має вбудовану підтримку адаптивної верстки (медіазапити), темної теми, псевдокласів та кастомізації теми через конфігураційний файл tailwind.config.js.

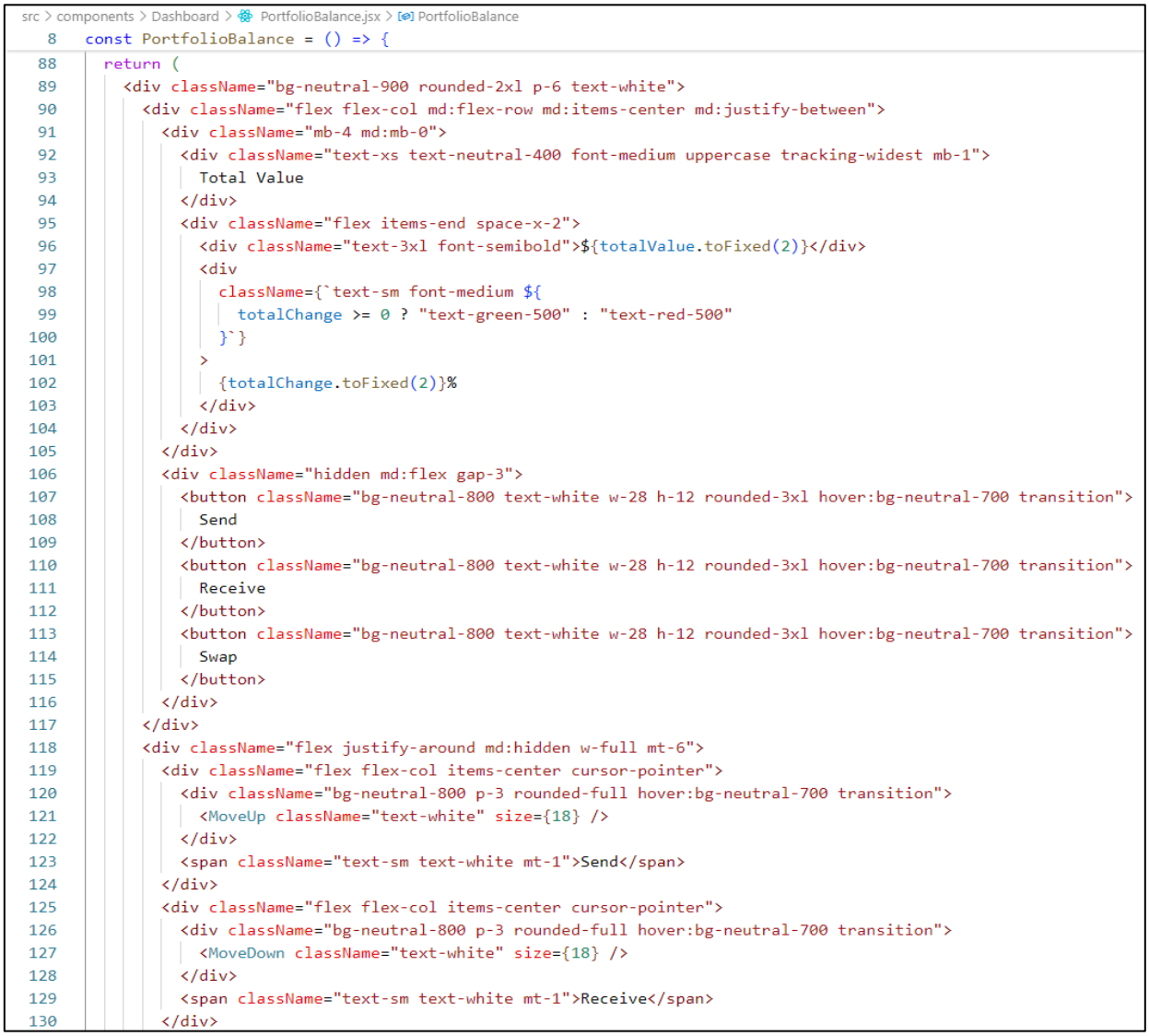


Рисунок 2.8 – Використання класів Tailwind CSS для побудови адаптивних компонентів JSX

Tailwind обрано через його швидкість розробки, гнучкість і сумісність із React-компонентами. Він значно спрощує побудову UI, зменшує кількість CSS-коду, підвищує читабельність і масштабованість інтерфейсу. Крім того, Tailwind має активну спільноту, велику кількість плагінів і офіційну документацію.

Підхід “utility-first” дозволяє стилізувати компоненти без створення кастомних CSS-файлів. Підтримує адаптивність і темну тему. Забезпечує швидку розробку, просту підтримку дизайну, гнучкість при кастомізації стилів.

2.2.5 Node.js та Express.js

Node.js – це середовище виконання JavaScript-коду на сервері, а Express.js – популярний фреймворк для створення серверних додатків на основі Node.js.

Node.js дозволяє виконувати JavaScript-код поза межами браузера, що забезпечує створення повноцінних серверів та API. Його неблокуюча модель введення/виведення (event-driven, non-blocking I/O) дозволяє обробляти велику кількість одночасних з’єднань. Express.js додає до Node.js зручні засоби маршрутизації, middleware, обробку помилок, шаблонізацію тощо.

Node.js активно підтримує бібліотеки та модулі через npm (Node Package Manager), що значно розширює його функціональні можливості. Express є «легким» фреймворком, що дозволяє зосередитися на логіці додатку, не перевантажуючи систему непотрібними залежностями.

У проєкті Node.js з Express використовується для побудови REST API, обробки запитів клієнта, підключення до бази даних Firestore, а також реалізації WebSocket-серверу для реального часу. Завдяки єдиній мові JavaScript на клієнті й сервері зменшується складність розробки. Node.js забезпечує масштабованість, продуктивність і гнучкість, необхідну для сучасного вебдодатку.

Node.js забезпечує асинхронне виконання коду, Express дозволяє реалізувати маршрутизацію, middleware, обробку запитів і REST API.

Обрано для реалізації серверної частини через спільну мову (JavaScript), високу продуктивність, простоту API і підтримку WebSocket.

2.2.6 WebSocket

WebSocket – це протокол для встановлення постійного двостороннього з’єднання між клієнтом і сервером (див. рис. 2.9, 2.10) [10-12].

```
useEffect(() => {
  const ws = new WebSocket(BINANCE_WS_URL);
  Windsurf: Refactor | Explain | Generate JSDoc | ×
  ws.onmessage = (event) => {
    const updates = JSON.parse(event.data);
    updates.forEach((u) => {
      if (u.s.endsWith("USDT")) {
        const base = u.s.replace("USDT", "");
        pricesRef.current[base + "USD"] = {
          price: parseFloat(u.c),
          change: parseFloat(u.P),
        };
      }
    });
  };
  return () => ws.close();
}, []);
```

Рисунок 2.9 – Реалізація WebSocket- з’єднання до API і передача повідомлень

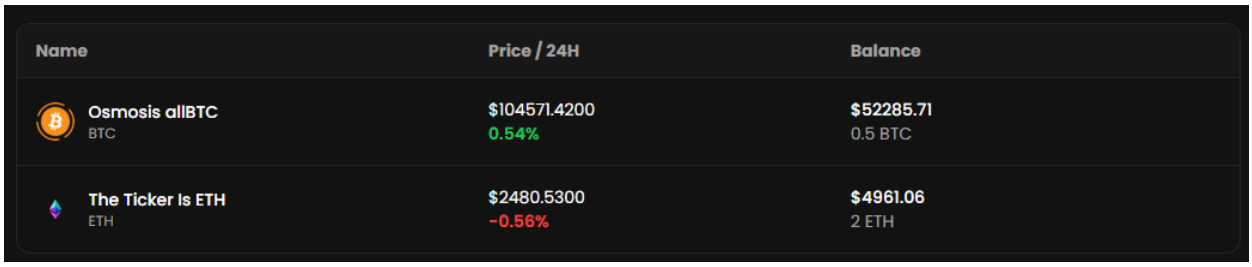


Рисунок 2.10 – Фрагмент інтерфейсу вебдодатку, де в режимі реального часу оновлюються ціни криптовалют на основі даних із WebSocket-з’єднання

WebSocket дозволяє клієнту та серверу підтримувати постійний відкритий канал зв’язку. Це усуває необхідність повторного встановлення з’єднання при кожному обміні повідомленнями, як це відбувається при REST-запитах. WebSocket працює поверх TCP, відкриваючи двосторонній канал, який дозволяє надсилати повідомлення як клієнтом, так і сервером у будь-який момент часу.

Використання WebSocket дає змогу виводити актуальні дані крипторинку в режимі реального часу без потреби оновлювати сторінку або повторно надсилати запити. Це критично важливо для користувачів, які слідкують за динамікою портфеля чи приймають інвестиційні рішення на основі змін курсу.

2.2.7 TradingView

TradingView – це сторонній віджет/бібліотека для інтерактивного відображення графіків фінансових активів (див. рис. 2.11, 2.12) [18, 19].

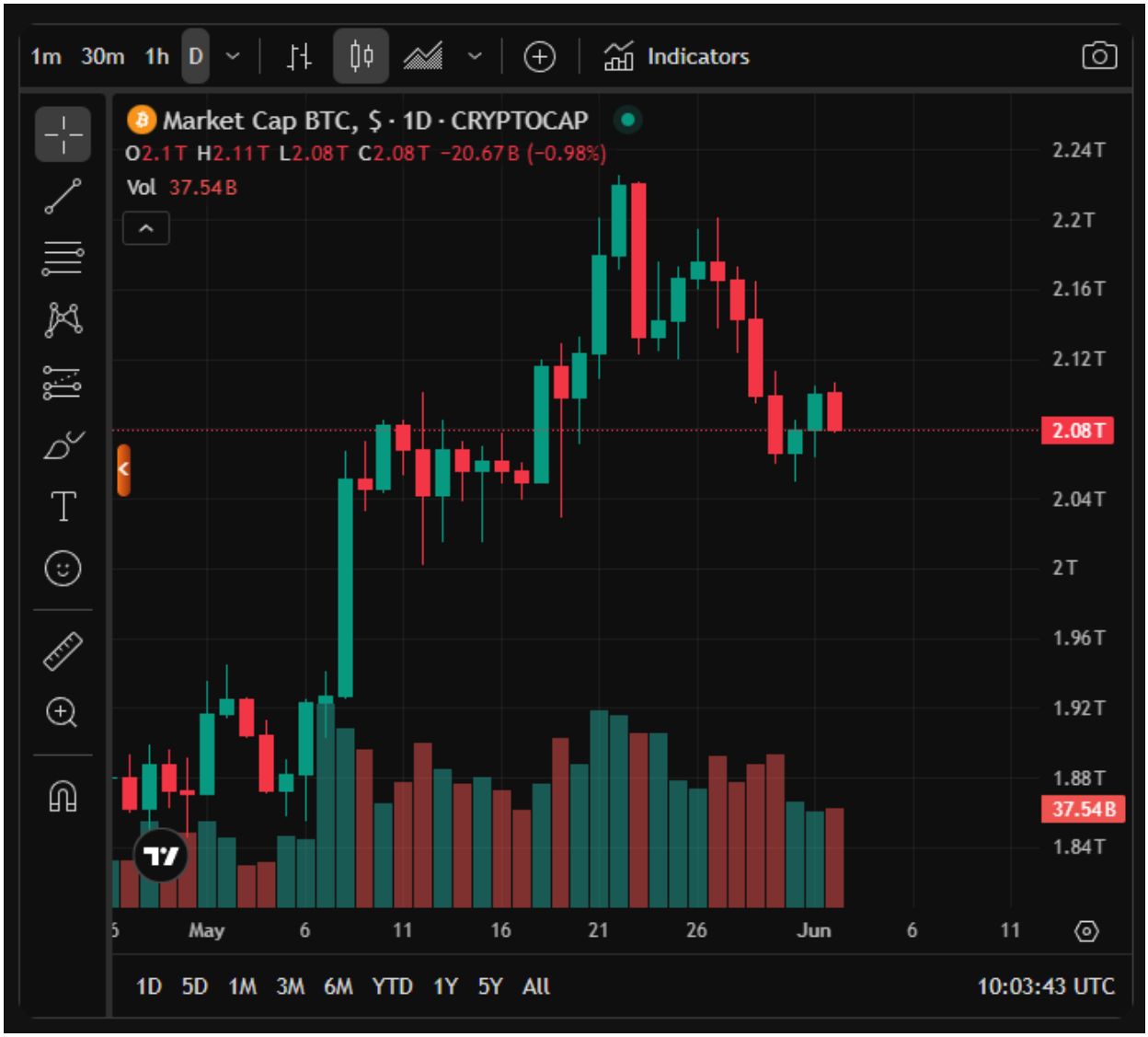


Рисунок 2.11 – Вбудований графік TradingView у вебінтерфейсі

```
useEffect(() => {
  if (!containerRef.current) return;
  containerRef.current.innerHTML = "";

  const script = document.createElement("script");
  script.src = "https://s3.tradingview.com/tv.js";
  script.async = true;
  Windsurf: Refactor | Explain | Generate JSDoc | ×
  script.onload = () => {
    new window.TradingView.widget({
      width: "100%",
      height: 610,
      symbol: `${symbol.toUpperCase()}`,
      interval: "D",
      timezone: "Etc/UTC",
      theme: "dark",
      style: 1,
      locale: "en",
      container_id: "tv_chart_container",
      withdateranges: true,
      hide_side_toolbar: hideToolbar,
      allow_symbol_change: false,
      save_image: true,
    });
  };
```

Рисунок 2.12 – Код інтеграції віджета TradingView у React-додатку

TradingView пропонує розширену систему графіків із підтримкою лінійних, свічкових, гістограмних типів візуалізації. Платформа також надає понад 100 технічних індикаторів і можливість створення власних індикаторів через мову Pine Script. Графіки можна інтегрувати в вебдодаток через офіційний бібліотечний віджет, який дозволяє передавати в нього дані через API або використовувати вже налаштовані потоки ринку. Доступне масштабування, перемикання таймфреймів, порівняння активів та відображення торгових обсягів.

TradingView є провідним інструментом візуалізації ринкових даних, що широко використовується трейдерами та аналітиками. Його готовий віджет легко вбудовується в React-додаток та забезпечує високу якість графіків із мінімальними зусиллями з боку розробника. У цьому проєкті віджет використовується для відображення зміни ціни конкретної криптовалюти на основі даних із Binance API.

2.2.8 Binance API

Binance API – офіційне API найбільшої криптовалютної біржі (див. рис. 2.13, 2.14) [4, 11].

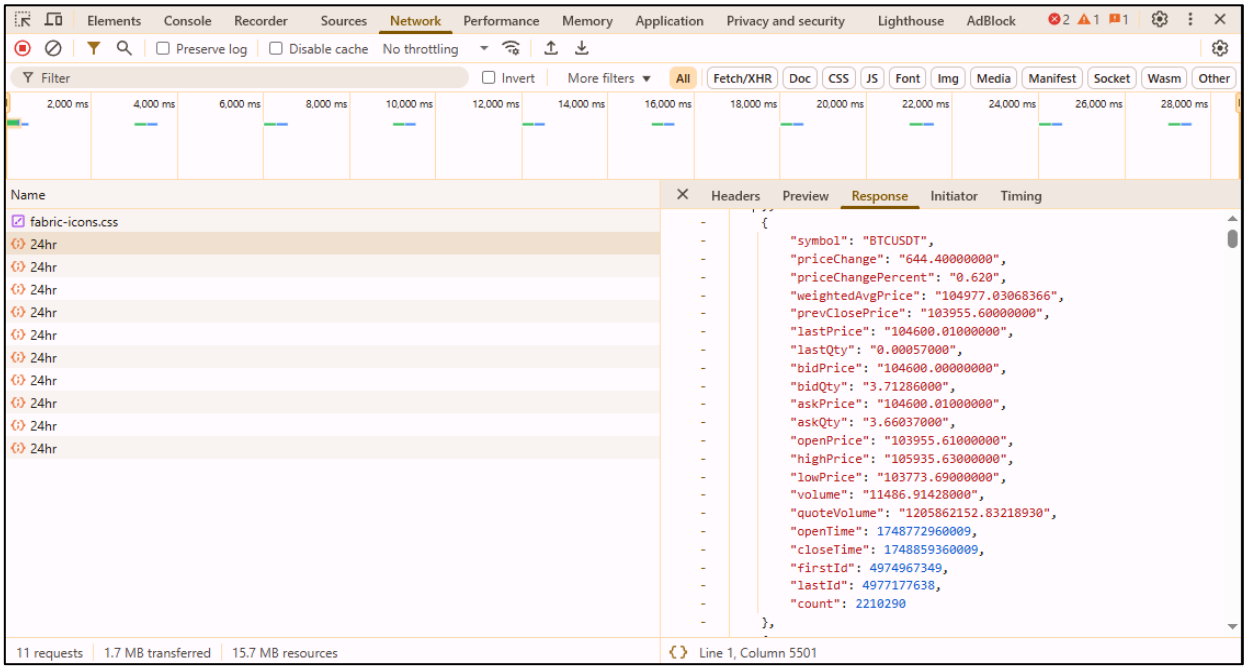


Рисунок 2.13 – Відповідь запиту до Binance API з інформацією про поточну ціну монети

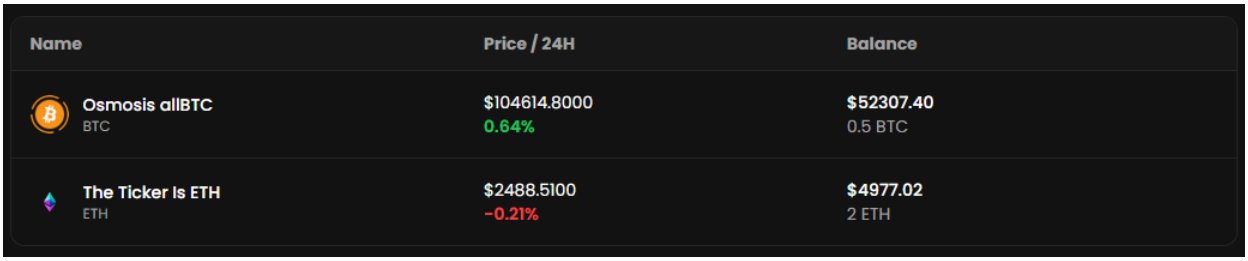


Рисунок 2.14 – Відображення отриманих даних на інтерфейсі вебдодатку

Binance API надає доступ до великого спектра даних про криптовалютні ринки. Він підтримує як REST-запити для отримання поточних цін, глибини ринку (order book), історичних угод і біржових пар, так і WebSocket-з'єднання для підписки на стрімінг реального часу про зміну ціни, обсяги торгів, події ринку. REST API дозволяє налаштовувати параметри запитів, включно з символами монет, інтервалами часу та глибиною відповіді. WebSocket надає

оновлення в реальному часі з мінімальною затримкою, що важливо для аналітичних інструментів. Має REST і WebSocket варіанти.

Binance API обрано як основне джерело цінових даних для вебдодатку через його високу надійність, швидкодію, точність та широке охоплення активів. За допомогою WebSocket оновлення курсів монет транслюються в інтерфейс користувача практично миттєво. REST-запити застосовуються для первинного завантаження даних або вибіркових запитів. API має чітку документацію, доступний навіть без ключів для базових запитів, та широко підтримується в екосистемі JavaScript.

2.2.9 CoinGecko API

CoinGecko API – безкоштовне публічне API для отримання криптовалютної інформації (див. рис. 2.15, 2.16) [5, 11]. CoinGecko API забезпечує доступ до повного набору інформації про криптовалютні активи: поточні ціни, історія змін, ринкова капіталізація, обсяги торгів, рейтинги та інші метрики.

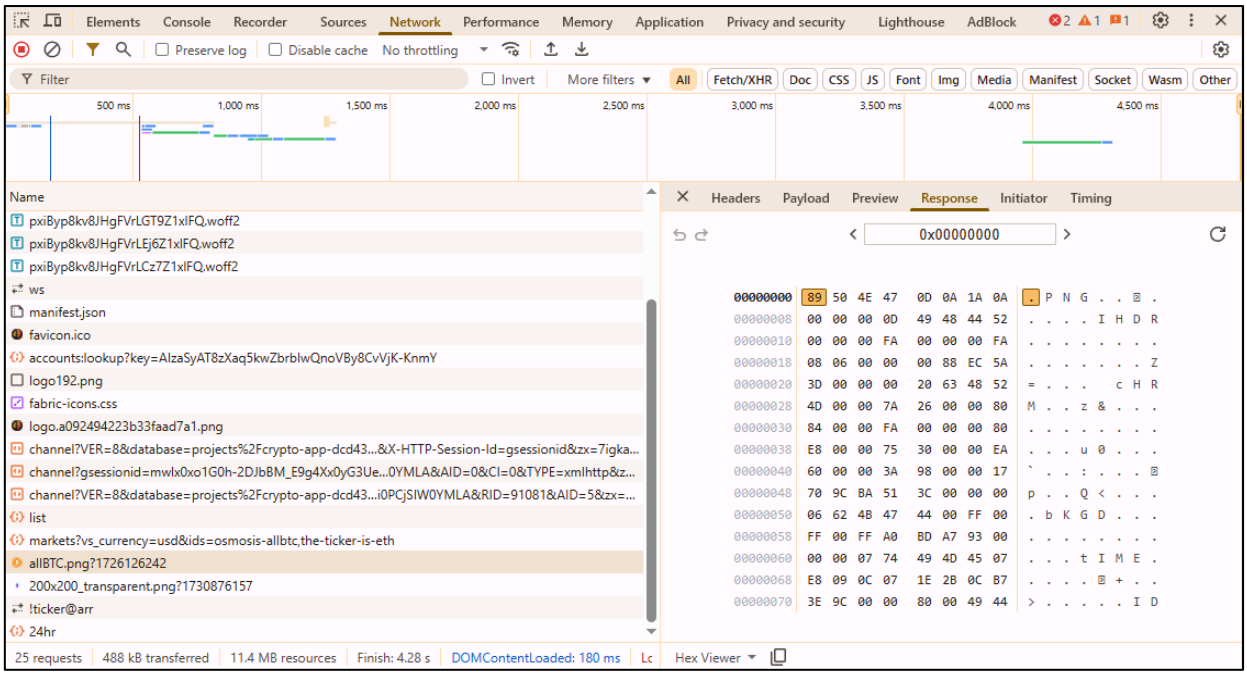


Рисунок 2.15 – Відповідь CoinGecko API

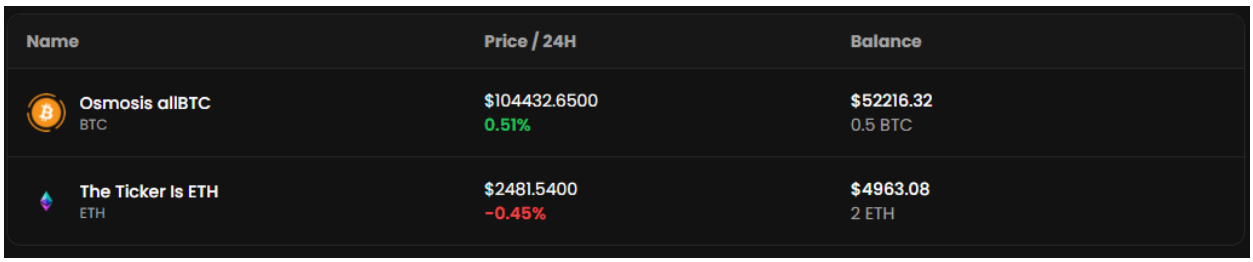


Рисунок 2.16 – Таблиця в інтерфейсі вебдодатку з логотипами криптовалют, завантаженими через CoinGecko API

Окремо підтримується зручний ендпоінт /coins/list і /coins/markets, які надають масив об’єктів з назвами, цінами, символами та URL-адресами іконок відповідних монет. Дані повертаються у форматі JSON, що спрощує обробку у фронтенді на React.

CoinGecko API обрано для виведення іконок криптовалют у додатку завдяки простоті доступу, відсутності авторизації для базових запитів, стабільності сервісу та актуальності візуальних матеріалів. Отримані URL-адреси іконок напряму підключаються до таблиці або списку монет, що забезпечує візуальну впізнаваність та кращу навігацію для користувача.

3 ПРОЄКТУВАННЯ ВЕБДОДАТКУ

Проєктування є одним із ключових етапів у розробці інформаційної системи. Воно включає побудову логічної та фізичної моделі програмного забезпечення, структуру бази даних, архітектуру сервісів, а також компонування інтерфейсу користувача з урахуванням вимог до зручності, швидкодії та функціональності. Саме на цьому етапі визначається, як окремі елементи системи взаємодіють між собою, зберігаються та обробляються дані, а також як користувач отримає доступ до необхідних функцій.

Проєктування вебдодатку для моніторингу та аналізу криптовалютних портфелів охоплює створення архітектури з клієнтською, серверною частинами та базою даних, моделювання структури даних, визначення технологій взаємодії між компонентами та побудову адаптивного, інтуїтивно зрозумілого інтерфейсу. Усі рішення приймаються з урахуванням вимог до безперервного оновлення курсів, безпеки персональних даних та масштабованості.

У цьому розділі розглянуто процес проєктування архітектури вебдодатку, що включає визначення основних компонентів системи, побудову взаємозв’язків між ними, структуру бази даних, інтерфейс користувача та особливості UI/UX дизайну. Проєктування базується на вимогах, сформованих у попередніх розділах, та враховує принципи масштабованості, модульності та зручності використання.

3.1 Загальна архітектура системи

Загальна архітектура вебдодатку базується на трирівневій моделі, яка передбачає поділ системи на клієнтську частину (frontend), серверну частину (backend) та базу даних (див. рис. 3.1). Такий підхід забезпечує

масштабованість, розподілення відповідальностей між компонентами, гнучкість під час розробки та підтримку у майбутньому.

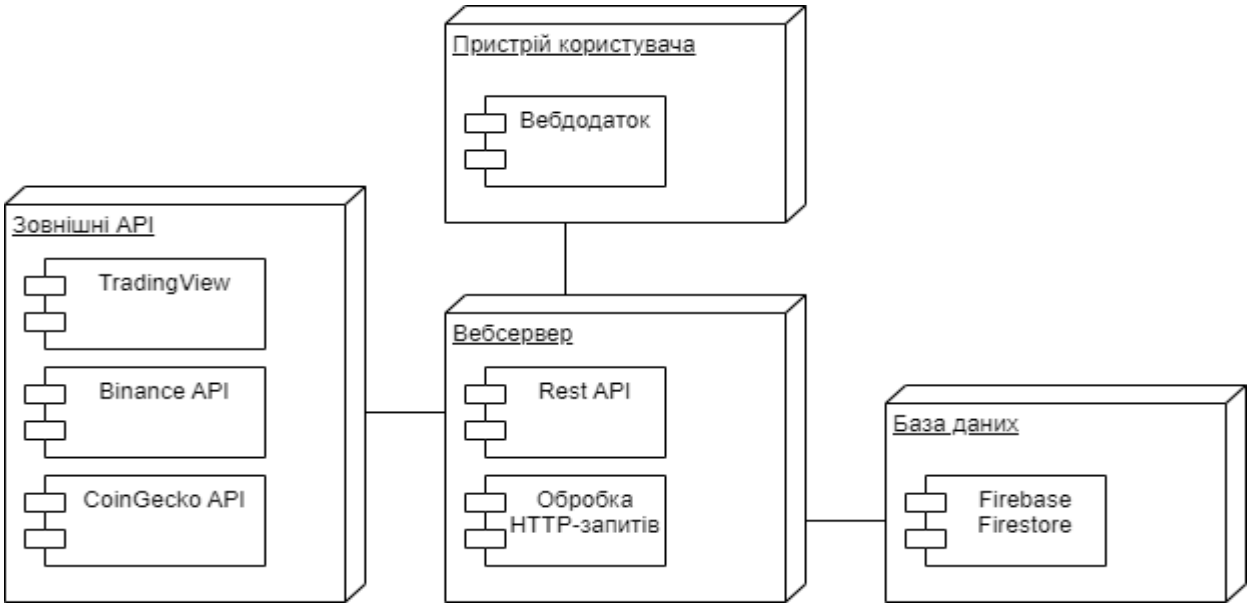


Рисунок 3.1 – Загальна архітектурна схема взаємодії frontend, backend та бази даних (діаграма розгортання)

Кожен рівень має свої функції та технологічне забезпечення.

Клієнтська частина (Frontend) – відповідає за візуалізацію та взаємодію з користувачем. Реалізована за допомогою React.js – бібліотеки, що дозволяє створювати SPA-додатки. Використання Tailwind CSS забезпечує гнучку та швидку стилізацію елементів інтерфейсу. Клієнтська частина ініціює запити до серверу, обробляє отримані дані, відображає портфель користувача, графіки цін і дозволяє змінювати налаштування.

Серверна частина (Backend) – побудована на основі Node.js і фреймворку Express.js. Вона приймає запити від клієнта, обробляє їх, виконує перевірку автентифікації через Firebase Authentication, з’єднується з Firestore для зчитування і запису даних, а також обробляє WebSocket-з’єднання для трансляції цін у реальному часі. Сервер також відповідає за валідацію даних та реалізацію REST API.

База даних (Firestore) – виступає як основне сховище користувацьких і транзакційних даних. Вона зберігає структуру портфелів, історію змін,

налаштування користувача та лог подій. Структура даних організована у вигляді колекцій і документів, що дозволяє гнучко масштабувати систему за кількістю користувачів та їх активністю.

Важливим є те, що всі частини працюють асинхронно та взаємодіють за допомогою HTTP/HTTPS-запитів, REST API та WebSocket, що забезпечує швидке реагування системи навіть за великої кількості одночасних користувачів.

3.2 Схема бази даних

Проєктування структури бази даних є критичним етапом для забезпечення ефективного зберігання та обробки інформації. У цьому проєкті використовується Firebase Firestore – масштабована, документоорієнтована NoSQL база даних, яка дозволяє організувати інформацію у вигляді колекцій і документів (див. рис. 3.2).

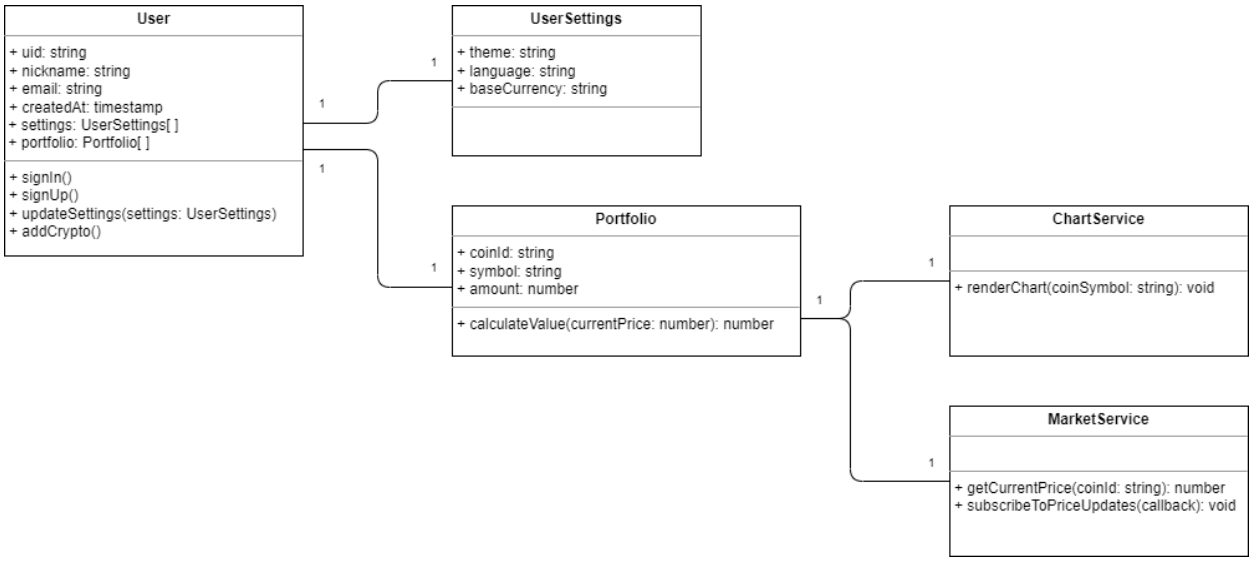


Рисунок 3.2 – Структура бази даних Firestore (діаграма класів)

Firestore реалізує структуру даних на основі вкладених колекцій, що дозволяє логічно пов’язати сутності між собою. Така модель є оптимальною для зберігання користувацьких даних, які оновлюються в реальному часі.

У системі передбачені такі основні колекції:

– колекція users;
– підколекція portfolios;
– колекція transactions;
– колекція settings.

Колекція users – містить документи, кожен із яких відповідає окремому користувачу. У документі зберігаються ідентифікатор (UID), email, дата реєстрації та інша основна інформація.

Підколекція portfolios – розташована всередині кожного документа користувача. Вона містить документи з інформацією про кожну криптовалюту в портфелі: назва монети, кількість, ціна покупки, час останнього оновлення, валютна пара (наприклад, BTC/USDT).

Колекція transactions – глобальна колекція для зберігання історії всіх транзакцій користувачів (купівля, продаж, оновлення). Вона пов’язана з користувачем через UID і містить дані про тип операції, валюту, кількість і часову мітку.

Колекція settings – зберігає налаштування для кожного користувача: вибрана мова інтерфейсу, тема оформлення, базова валюта, інші параметри персоналізації.

Така структура дозволяє легко фільтрувати дані, забезпечує швидкий доступ до портфеля конкретного користувача, оптимізує пошук і дає змогу реалізувати гнучку систему прав доступу завдяки правилам безпеки Firestore.

3.3 Компонентна структура клієнтської частини

Компонентна архітектура є одним із ключових підходів при розробці сучасних односторінкових додатків (SPA) за допомогою бібліотеки React (див. рис. 3.3). Кожен компонент відповідає за окрему частину функціональності інтерфейсу, має власний стан (state), обробляє події та може бути повторно

використаний. Такий підхід значно спрощує підтримку, тестування та масштабування коду.

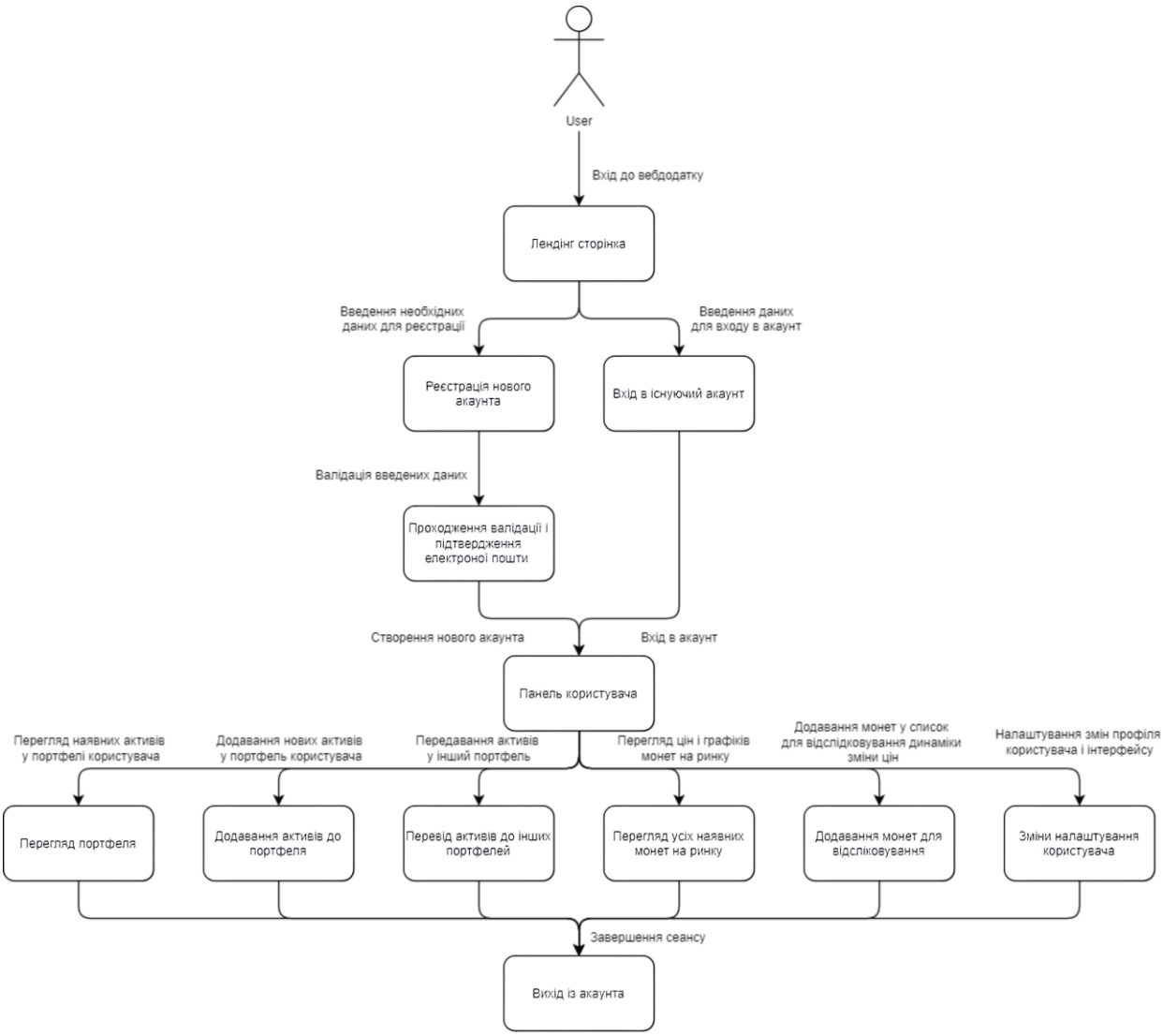


Рисунок 3.3 – Схема компонентів інтерфейсу користувача та їх взаємозв’язків (діаграма використання)

У рамках проєкту розроблено набір функціональних компонентів, згрупованих за логікою роботи:

– Navbar та Sidebar – забезпечують навігацію між сторінками, а також доступ до основних функцій, таких як профіль, налаштування, вихід із системи;
– SettingsPage – сторінка налаштувань профілю користувача (тема, валюта, мова);

- PortfolioView – головний компонент, який відображає список криптовалют у портфелі користувача, виводить їхні поточні курси, сумарну вартість та зміни за добу;
- AssetCard – окремий компонент для візуалізації інформації про кожну монету (містить назву, іконку, поточну ціну, кількість, загальну вартість, зміну у відсотках);
- ChartPanel – містить інтегрований віджет TradingView для побудови графіка зміни ціни вибраної криптовалюти (дозволяє обирати валютну пару, таймфрейм та відображати індикатори);
- AuthPage, RegisterPage – сторінки для входу/реєстрації, інтегровані з Firebase Authentication.

Для керування станом додатку використовуються React-хуки – useState (локальний стан), useEffect (життєвий цикл компонента), useContext (глобальний стан, наприклад, користувача). Дані передаються між компонентами через props, або ж контекст при потребі глобального доступу.

Компоненти стилізуються за допомогою Tailwind CSS, що дозволяє швидко формувати UI без створення окремих CSS-файлів. Усі компоненти адаптовані для різних розмірів екранів, що забезпечує якісний користувацький досвід на мобільних і настільних пристроях.

3.4 Проєктування UI/UX

UI/UX-дизайн є важливою складовою якісного програмного забезпечення, оскільки визначає, наскільки зручно, зрозуміло та привабливо користувачеві взаємодіяти із системою. У цьому проєкті проєктування інтерфейсу базувалося на принципах мінімалізму, інтуїтивності та функціональної простоти.

У фокусі – швидкий доступ до інформації, чітка ієрархія даних та взаємодія в один-два кліки.

Усі елементи дизайну побудовані за допомогою Tailwind CSS, що дозволяє швидко створювати адаптивні компоненти із суворим контролем візуальної консистентності.

Для користувачів важливо бачити стан свого портфеля в реальному часі. Саме тому елементи, пов’язані з оновленням ринкової інформації, виділено в інтерфейсі – значення цін динамічно змінюються без втручання користувача. Це реалізується через WebSocket, а графіки побудовано з використанням TradingView.

Основні вимоги до UI:

– логічне групування інформації;
– мінімалістичний стиль оформлення (Tailwind CSS);
– швидкий доступ до основних функцій (портфель, графік, налаштування);
– інтуїтивно зрозуміла навігація;
– урахування темної та світлої тем для покращення зорового сприйняття та персоналізації;
– використання чіткої типографіки, контрастних кольорів і зручного розміщення елементів;
– адаптація для мобільних пристроїв через mobile-first дизайн.

Особливу увагу приділено відображенню актуальної інформації в реальному часі: значення криптовалют постійно оновлюються через WebSocket, а графік TradingView візуалізує зміни курсу.

3.5 Логіка обробки даних

Обробка та передача даних у системі є важливою частиною функціонування додатку, оскільки саме завдяки налагодженим механізмам обміну інформацією між клієнтською частиною, сервером і базою даних забезпечується швидкість, актуальність та точність інформації, що відображається користувачеві (див. рис. 3.4).

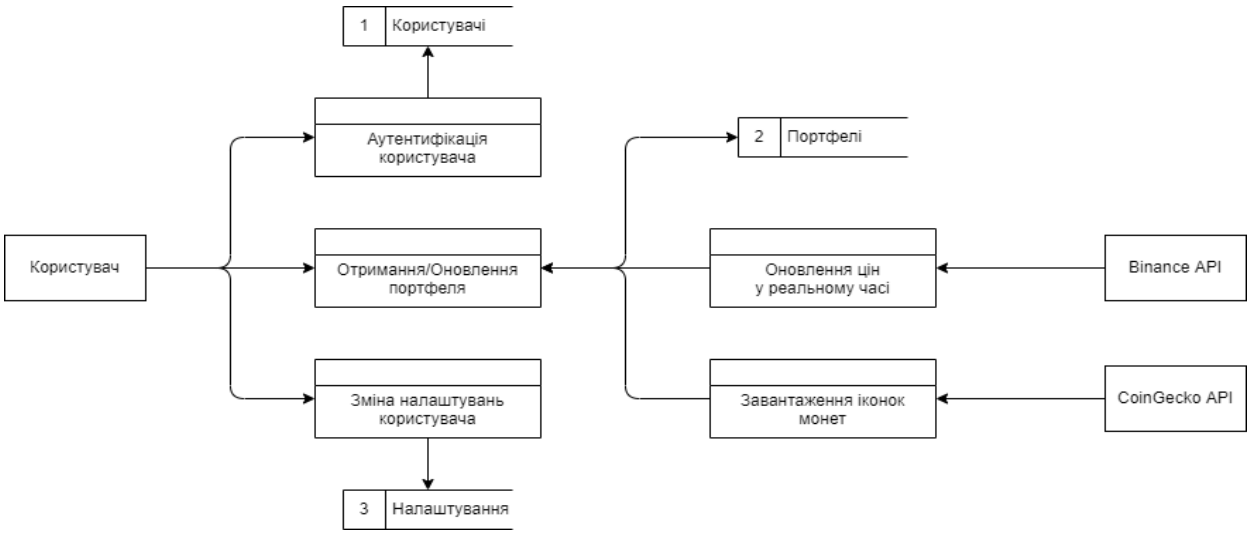


Рисунок 3.4 – Схема потоків обробки даних між клієнтом, сервером і базою даних (діаграма потоків даних)

Ключовими етапами логіки обробки є:

– авторизація користувача: після завантаження додатку виконується перевірка автентифікації за допомогою Firebase Authentication (у разі успішної авторизації клієнт отримує доступ до особистих даних користувача);
– завантаження портфеля: клієнтська частина звертається до Firestore і отримує інформацію про криптовалюти у портфелі користувача: назви монет, кількість, вартість покупки тощо (ці дані використовуються для первинного формування інтерфейсу);
– підключення до WebSocket Binance API: встановлюється з’єднання з Binance WebSocket для підписки на поточні курси монет у реальному часі (після отримання оновлень курси оновлюються у стані React-компонентів, що забезпечує миттєве оновлення інтерфейсу);
– розрахунок вартості та прибутковості: на основі отриманих цін та кількості кожної монети обчислюється сумарна вартість портфеля, а також прибуток або збиток у відсотках та у валюті (ці значення відображаються на головному екрані);
– редагування портфеля: при додаванні, редагуванні або видаленні монети оновлені дані зберігаються у Firestore, після чого

автоматично синхронізуються з інтерфейсом користувача через механізм реального часу;

- відображення графіків: користувач може переглянути зміну вартості обраної монети за допомогою графіку TradingView, який отримує ринкові дані відповідно до обраної пари та таймфрейму;
- налаштування: вибрані користувачем параметри (тема, мова, базова валюта) зберігаються у Firestore, впливають на візуальне оформлення та перерахунок сум, а також застосовуються автоматично при наступному вході.

Такий підхід забезпечує швидке реагування системи, синхронізацію даних між клієнтами, зниження навантаження на сервер і зручність взаємодії з додатком.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

Після завершення етапу проєктування, наступним кроком у створенні вебдодатку є його безпосередня програмна реалізація. На цьому етапі здійснюється написання коду для всіх частин системи: клієнтської (React.js), серверної (Node.js з Express.js) та зв’язку з базою даних (Firebase Firestore). Крім того, інтегруються зовнішні API-сервіси, зокрема Binance API для отримання ринкових даних у реальному часі та CoinGecko API для виведення іконок криптовалют.

Програмну реалізацію було здійснено з дотриманням принципів модульності, повторного використання коду, безпечної обробки даних та масштабованості.

Особлива увага приділялася синхронізації даних між клієнтом і сервером, обробці динамічного контенту та забезпеченню швидкого відгуку інтерфейсу.

Реалізація включала такі ключові етапи:

– створення структури проєкту та встановлення необхідних залежностей (React, Tailwind CSS, Express, Firebase SDK);
– розробка функціональних компонентів інтерфейсу, включаючи елементи навігації, таблицю активів, графіки, налаштування користувача;
– налаштування аутентифікації через Firebase Authentication;
– підключення до Firestore та організація схеми зберігання даних;
– реалізація REST API для обробки запитів до серверної частини;
– підключення до WebSocket для оновлення цін у реальному часі;
– інтеграція віджету TradingView для побудови графіків;
– тестування функціональності на різних пристроях і екранах.

4.1 Реалізація основних функцій вебдодатку

Програмна реалізація вебдодатку охоплює усі основні елементи, необхідні для зручного, безпечного та ефективного моніторингу криптовалютного портфеля в режимі реального часу (див. рис. 4.1). Зокрема, було створено модульну систему з підтримкою авторизації, динамічного інтерфейсу, графіків, WebSocket-з'єднання та гнучкого зберігання даних.

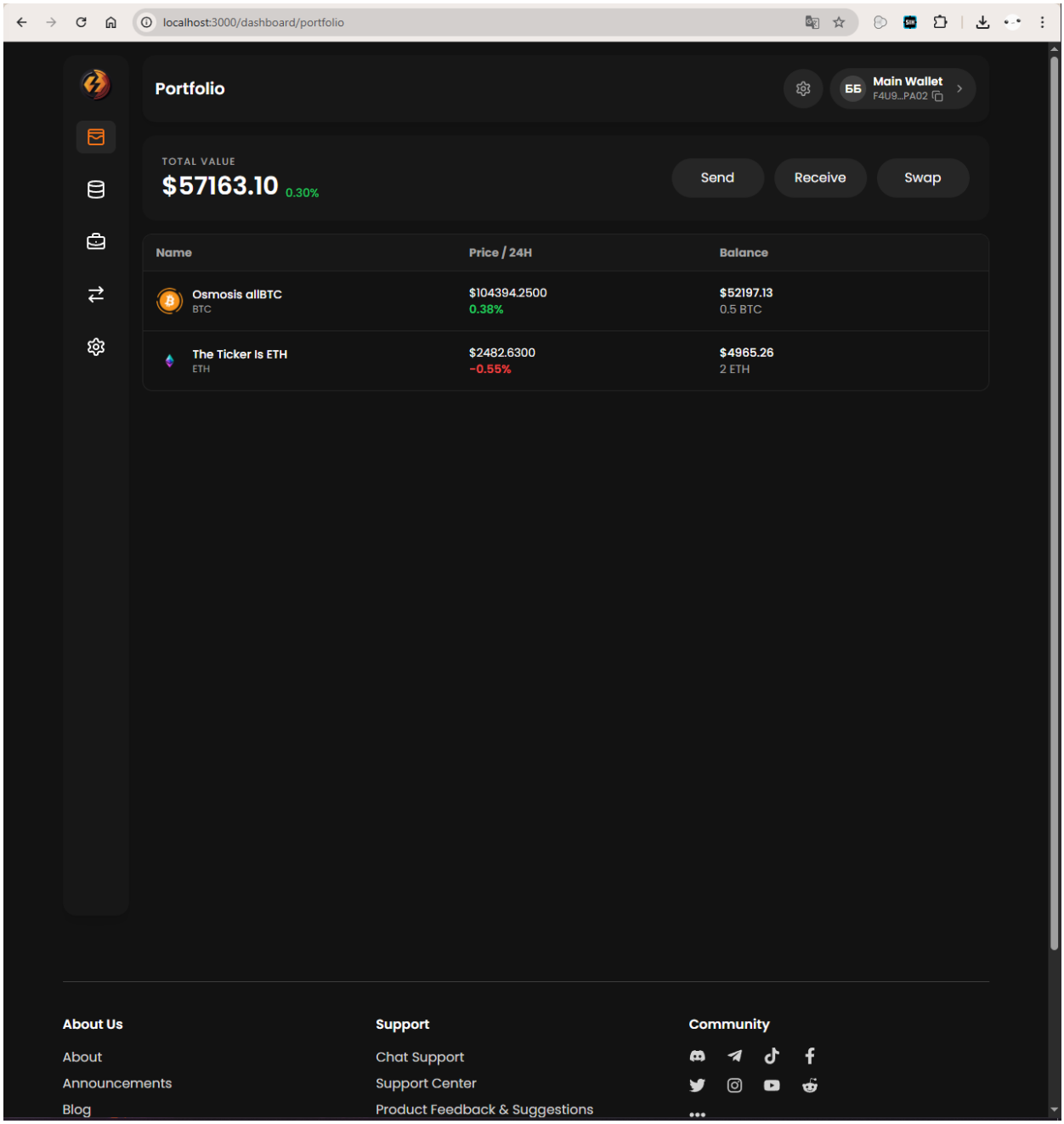


Рисунок 4.1 – Сторінка вебдодатку з таблицею криптовалют і загальною вартістю портфеля

У процесі реалізації було впроваджено такі основні функції:

– створення, перегляд та редагування криптовалютного портфеля користувача;
– автоматичне оновлення курсу кожної монети через WebSocket API від Binance;
– побудова графіків зміни вартості активів за допомогою TradingView;
– відображення динаміки прибутковості в режимі реального часу;
– візуалізація даних за допомогою таблиць, карток активів і графіків;
– адаптивне відображення компонентів на екранах різного розміру.

Усі функціональні блоки розроблено як окремі React-компоненти з підтримкою реактивного оновлення стану через хуки. Компоненти мають чітке розмежування відповідальностей: навігація, перегляд портфеля, відображення ринкової інформації, редагування даних, перегляд графіків тощо.

Особливу увагу приділено взаємодії між frontend і backend. Дані з Binance WebSocket API передаються у реальному часі на клієнтську частину, де відразу оновлюються відповідні стани і перераховуються аналітичні показники. Результати обчислень синхронно оновлюються в інтерфейсі користувача, не потребуючи ручного оновлення сторінки.

Інтерфейс було реалізовано за принципами mobile-first дизайну (див. рис. 4.2) з використанням Tailwind CSS. Це забезпечує швидке завантаження сторінок і зручну адаптацію для екранів різних розмірів. Навігація реалізована через React Router, а маршрути захищені перевіркою стану автентифікації користувача.

Завдяки Firebase Firestore дані користувачів зберігаються надійно, синхронізуються в реальному часі (див. рис. 4.3), а також доступні з будь-якого пристрою.

Усі зміни – додавання монет, оновлення кількості, редагування налаштувань – миттєво зберігаються та відображаються без потреби оновлення інтерфейсу вручну.

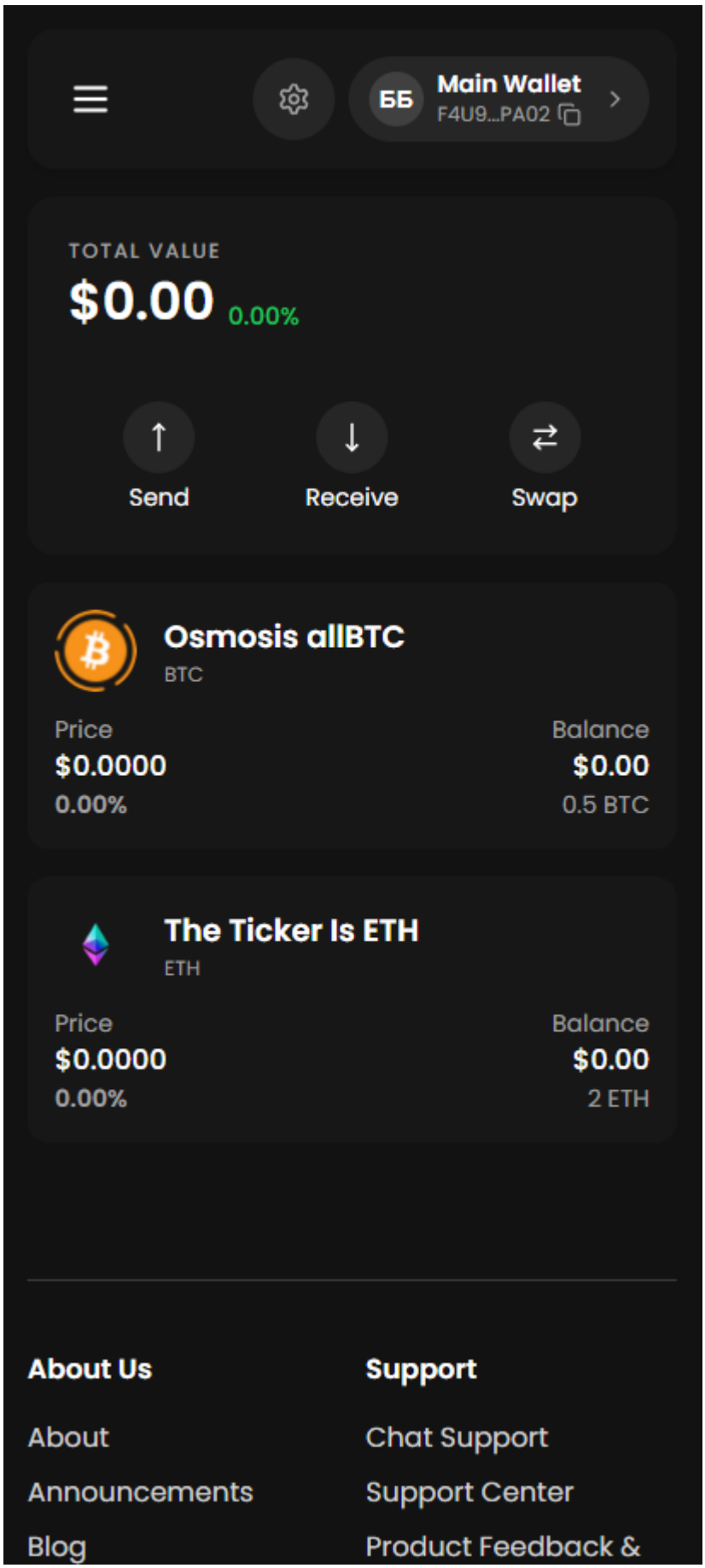


Рисунок 4.2 – Адаптивне відображення таблиці активів на мобільному екрані

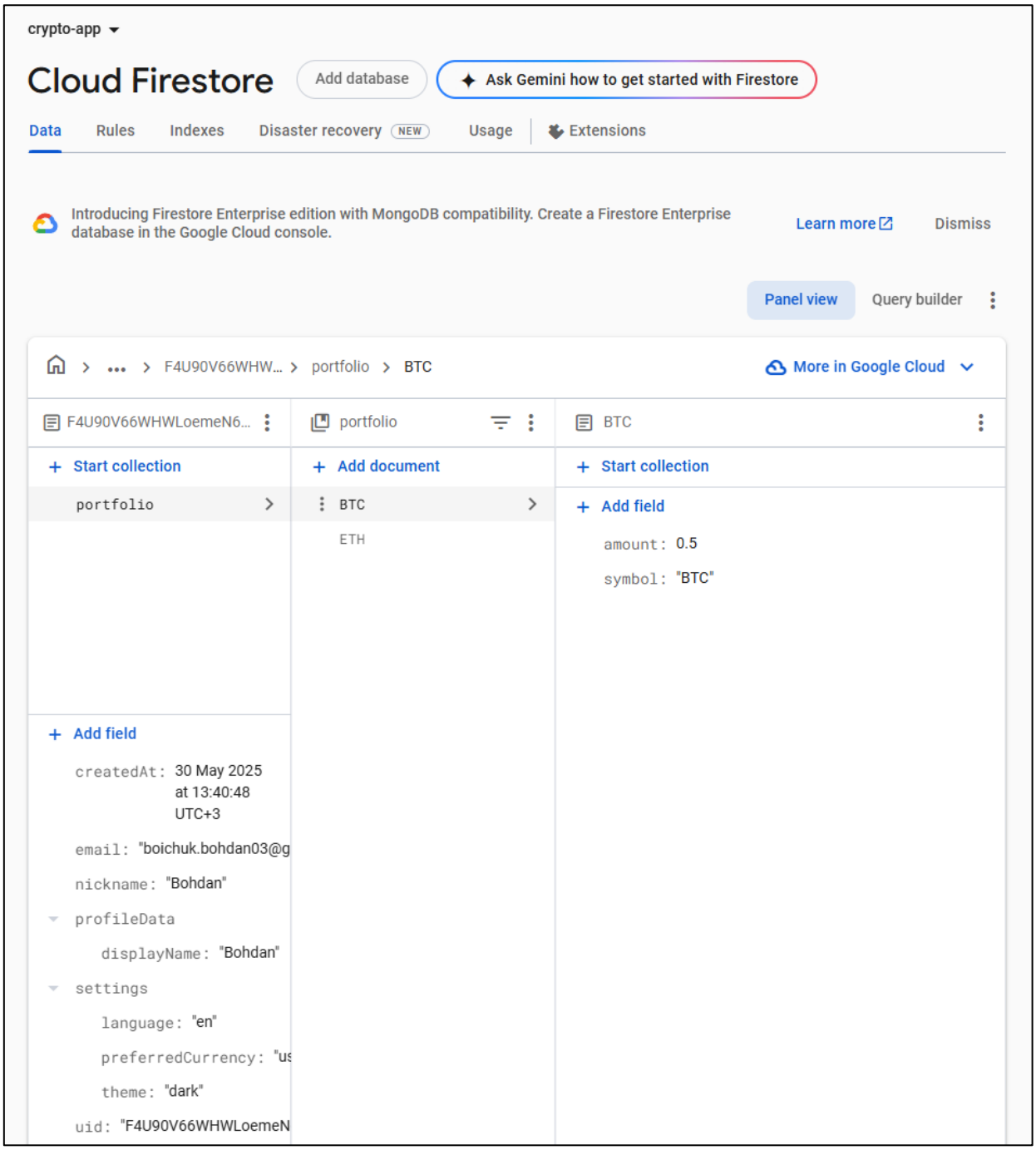


Рисунок 4.3 – Збереження портфеля користувача у Firestore через панель адміністратора Firebase Console

4.2 Забезпечення безпеки та авторизації користувачів

Для захисту особистих даних і обмеження доступу до персонального портфеля було реалізовано систему автентифікації за допомогою Firebase Authentication (див. рис. 4.4). Цей сервіс надає простий у реалізації, але

водночас надійний механізм входу до системи з підтримкою різних методів ідентифікації користувача. Основним способом автентифікації, реалізованим у додатку, є використання email та пароля.

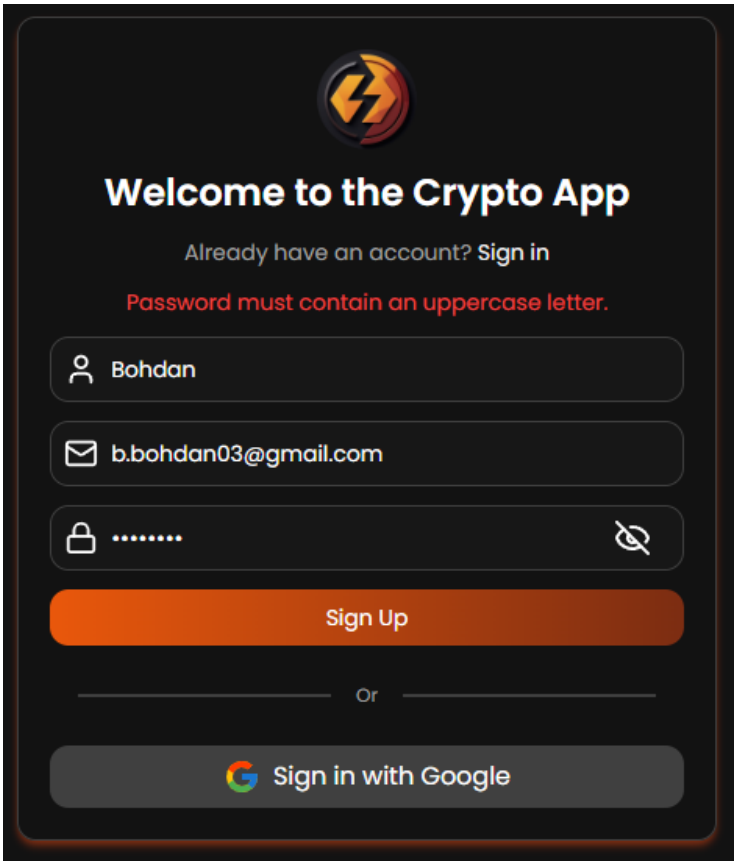


Рисунок 4.4 – Форма входу до системи з полями email/пароль та обробкою помилок валідації

Процес автентифікації інтегровано у фронтенд за допомогою Firebase SDK. Користувач має змогу зареєструватися або увійти до системи через відповідні React-компоненти з формами, які містять клієнтську валідацію введених даних. Після успішної автентифікації, Firebase створює унікальний ідентифікатор користувача (UID), що використовується для зв’язку з відповідними даними у базі Firestore.

Для підвищення безпеки додано перевірку токена автентифікації при кожному доступі до захищених сторінок. Також реалізовано збереження сесії користувача через localStorage, що дозволяє залишатися в системі після перезавантаження сторінки.

Крім цього, всі операції з Firestore обмежені правилами безпеки Firebase, які перевіряють UID користувача перед наданням доступу до даних. Завдяки цьому, кожен користувач бачить лише свій портфель і не має доступу до даних інших користувачів.

Підтримуються такі дії:

– реєстрація нового користувача з валідацією email і пароля;
– вхід до системи для авторизованих користувачів;
– вихід із облікового запису;
– перевірка автентифікованого стану користувача на кожній сторінці додатку;
– доступ до Firestore обмежено згідно з UID користувача через правила безпеки Firebase.

4.3 Тестування функціональності та продуктивності

Для перевірки коректності та ефективності роботи вебдодатку було проведено тестування як функціональної, так і продуктивної складових системи. Тестування охоплювало типові сценарії використання, обробку помилок, швидкодію, стабільність та адаптивність інтерфейсу.

Функціональне тестування включало перевірку наступного:

– реєстрація та авторизація користувача;
– перегляд поточного портфеля та оновлення цін у реальному часі;
– додавання, редагування та видалення монет із портфеля;
– збереження та завантаження налаштувань користувача;
– взаємодія з графіком TradingView;
– адаптивність інтерфейсу для мобільних пристроїв.

Було перевірено обробку помилок, таких як введення некоректних даних, втрату інтернет-з’єднання або помилки доступу до API. Усі виявлені помилки було усунуто в процесі розробки.

Тестування продуктивності включало:

– вимірювання швидкості завантаження головної сторінки при різній кількості активів (від 1 до 100 записів);
– перевірку затримки при оновленні цін через WebSocket;
– аналіз часу рендерингу таблиць та графіків;
– перевірку стабільності роботи при повільному інтернет-з’єднанні;
– моніторинг використання оперативної пам’яті та завантаженості процесора у браузері.

Результати засвідчили стабільну роботу додатку, миттєве оновлення цін (менше 1 с), швидкий рендеринг компонентів та відсутність помітних затримок або зависань навіть при підвищеному навантаженні (див. рис. 4.5).

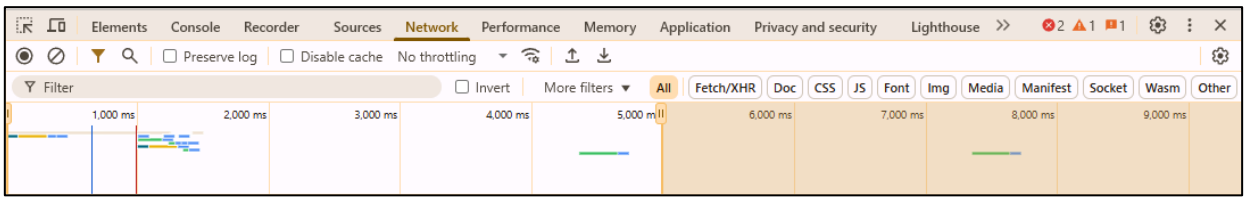


Рисунок 4.5 – Діаграма часу завантаження сторінки та рендерингу таблиць залежно від кількості активів

У результаті тестування підтверджено коректну роботу всіх компонентів вебдодатку, що забезпечує якісний досвід для кінцевого користувача.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було реалізовано повнофункціональний вебдодаток для моніторингу та аналізу криптовалютного портфеля. У процесі дослідження було проаналізовано актуальність криптовалютних технологій, вивчено сучасні підходи до побудови клієнт-серверних систем, обґрунтовано вибір технологічного стеку та розроблено архітектуру системи.

Вебдодаток забезпечує зручний інтерфейс для користувача, підтримує реєстрацію та автентифікацію, дозволяє створювати персональний портфель, отримувати актуальні ціни в режимі реального часу, відстежувати прибутковість та переглядати графіки зміни вартості активів.

Технічна реалізація охоплює такі ключові елементи, як React.js для побудови інтерфейсу, Tailwind CSS для адаптивної стилізації, Node.js і Express для серверної логіки, Firebase Firestore для зберігання даних, Firebase Authentication для безпечної авторизації, а також WebSocket і API Binance/CoinGecko для динамічного оновлення ринкової інформації.

Результати тестування підтвердили стабільну роботу системи, високу швидкодію, адаптивність і зручність інтерфейсу. Розроблений додаток може бути використаний як основа для реального продукту або розширений відповідно до потреб кінцевих користувачів.

Таким чином, поставлену мету роботи досягнуто, а всі заплановані функціональні завдання реалізовано.

ПЕРЕЛІК ПОСИЛАНЬ

1. Amazon Web Services (AWS). *RESTful API*. URL: https://aws.amazon.com/what-is/restful-api (дата звернення: 18.04.2025).

2. Avada Media. *Firebase*. URL: https://avada-media.ua/blog/firebase (дата звернення: 14.04.2025).

3. Що таке криптовалюта та як вона працює? *Binance Academy*. URL: https://academy.binance.com/uk-UA/articles/what-is-a-cryptocurrency (дата звернення: 01.02.2025).

4. Binance API Official Docs. *Binance*. URL: https://binance-docs.github.io/apidocs/spot/en (дата звернення: 18.04.2025).

5. CoinGecko API Docs. *CoinGecko API*. URL: https://www.coingecko.com/en/api/documentation (дата звернення: 18.04.2025).

6. Crypto App. URL: https://github.com/Laugh1ngGas/crypto-app (дата звернення: 29.03.2025).

7. Express.js Documentation. *Express*. URL: https://expressjs.com/en/starter/installing.html (дата звернення: 18.04.2025).

8. Firebase Documentation. *Firebase Authentication*. URL: https://firebase.google.com/docs/auth (дата звернення: 14.04.2025).

9. Firebase Documentation. *Firestore*. URL: https://firebase.google.com/docs/firestore (дата звернення: 15.04.2025).

10. JavaScript.Info. *WebSocket*. URL: https://uk.javascript.info/websocket (дата звернення: 18.04.2025).

11. MDN Web Docs. *Fetch API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 18.04.2025).

12. MDN Web Docs. *WebSocket Documentation*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket (дата звернення: 18.04.2025).

13. Minnick C. Beginning ReactJS Foundations: Building User Interfaces with ReactJS : An Approachable Guide. Hoboken, NJ : Wiley, 2022. 512 p.

URL: https://dl.ebooksworld.ir/books/Beginning.ReactJS.Foundations.Chris.Minnick.Wiley.9781119685548.EBooksWorld.ir.pdf (дата звернення: 14.03.2025).

14. Node.js Documentation. URL: https://nodejs.org/en/docs (дата звернення: 14.03.2025).

15. React Documentation. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 14.03.2025).

16. ReactJS Foundations. URL: https://www.reactjsfoundations.com (дата звернення: 14.03.2025).

17. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 14.03.2025).

18. TradingView Charts Documentation. URL: https://tradingview.github.io/lightweight-charts/docs (дата звернення: 25.04.2025).

19. TradingView Charts Library. URL: https://www.tradingview.com/lightweight-charts (дата звернення: 25.04.2025).

ДОДАТОК А

Інтерфейс вебдодатку

Основні сторінки розробленого вебдодатку для моніторингу та аналізу криптовалютного портфеля. Зображення демонструють ключові елементи інтерфейсу користувача, реалізовані функції, адаптивність до різних пристроїв, а також використання зовнішніх бібліотек для візуалізації фінансових даних. З розробленим вебдодатком можна ознайомитися за посиланням на репозиторій GitHub [6].

Лендінг сторінка додатку демонструє назву додатку, логототип, кнопки авторизації, а також інформатичні та анімаційні компоненти, що пояснюють мету сервісу (див. рис. А.1). Таблиця з актуальними курсами криптовалют і графіками представлена на рисунку А.2.

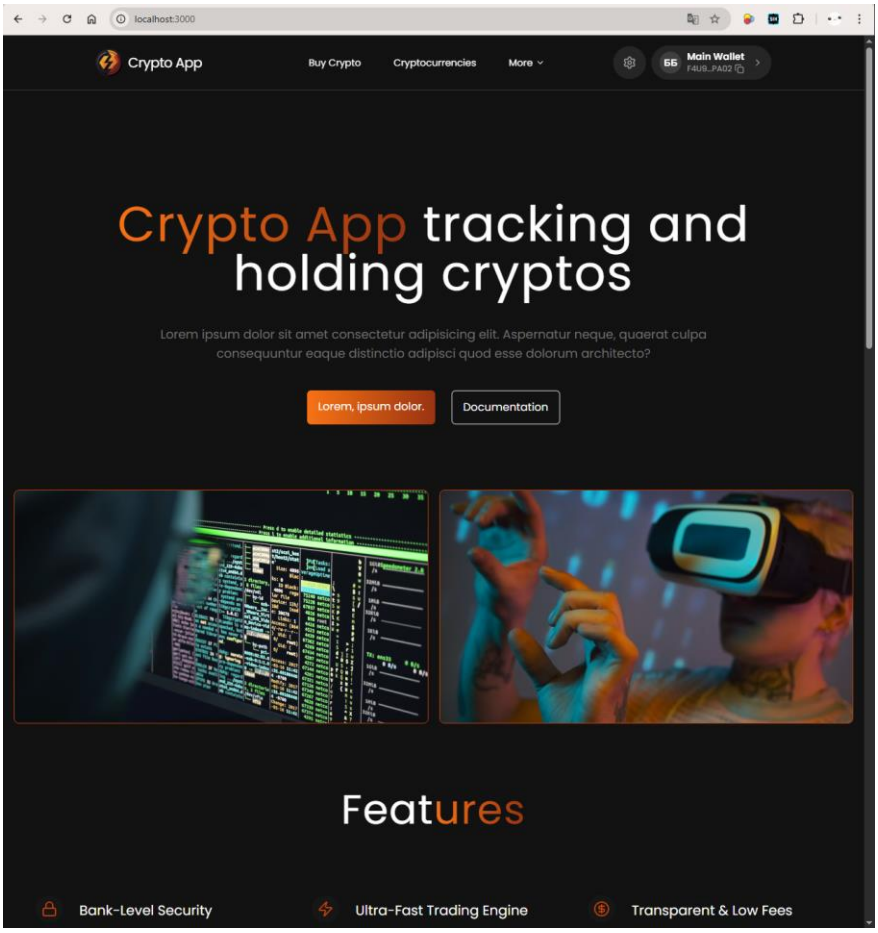


Рисунок А.1 – Лендінг сторінка додатку

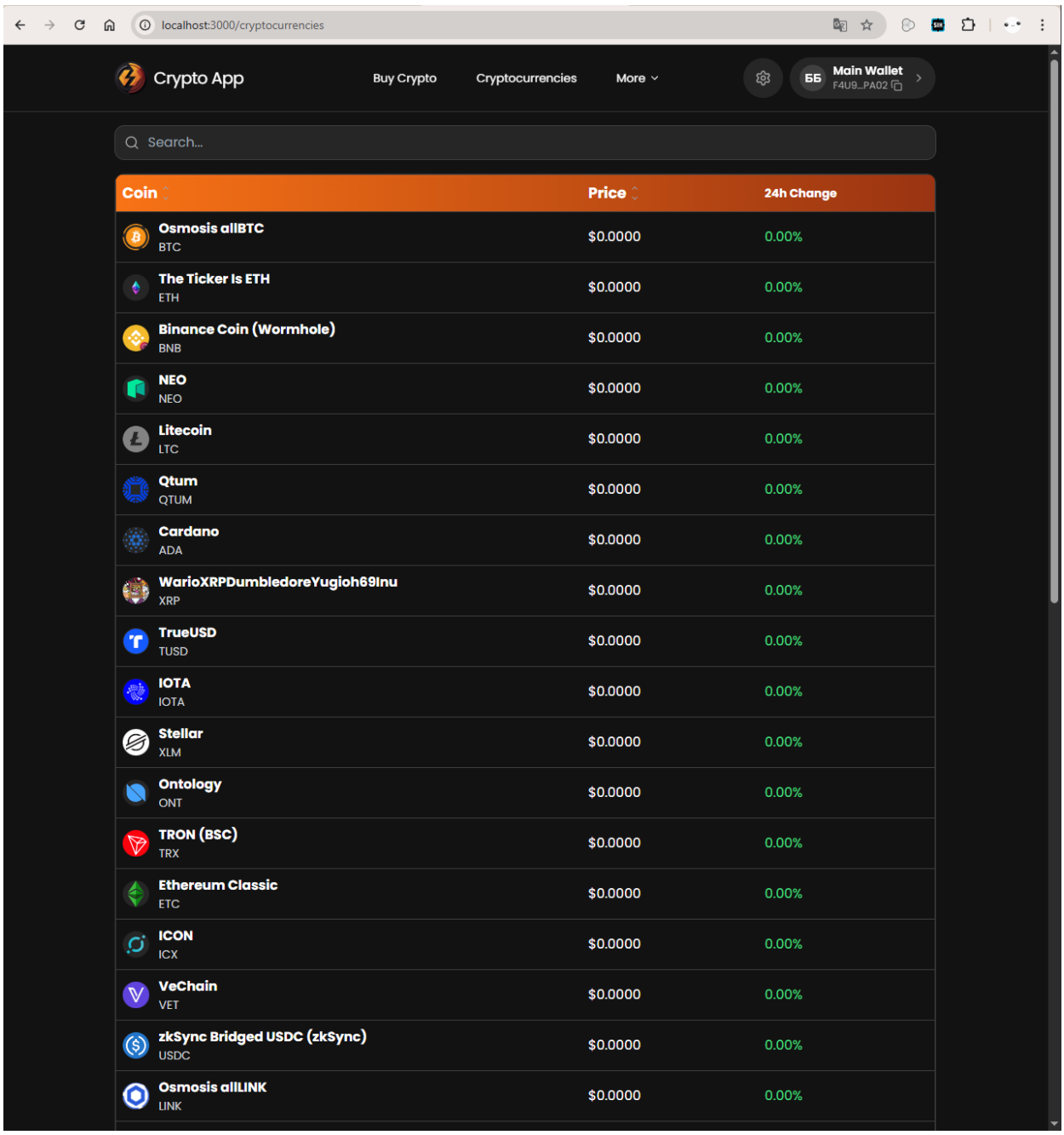


Рисунок А.2 – Таблиця курсів криптовалют

Інтерактивний графік ціни криптовалюти (див. рис. А.3) реалізовано за допомогою бібліотеки TradingView.

Форма авторизації користувача (див. рис. А.4) реалізовано вхід за email і паролем із валідацією або за допомогою акаунта Google, інтегровано з Firebase Authentication. У разі помилки виводиться відповідне повідомлення (див. рис. А.5).

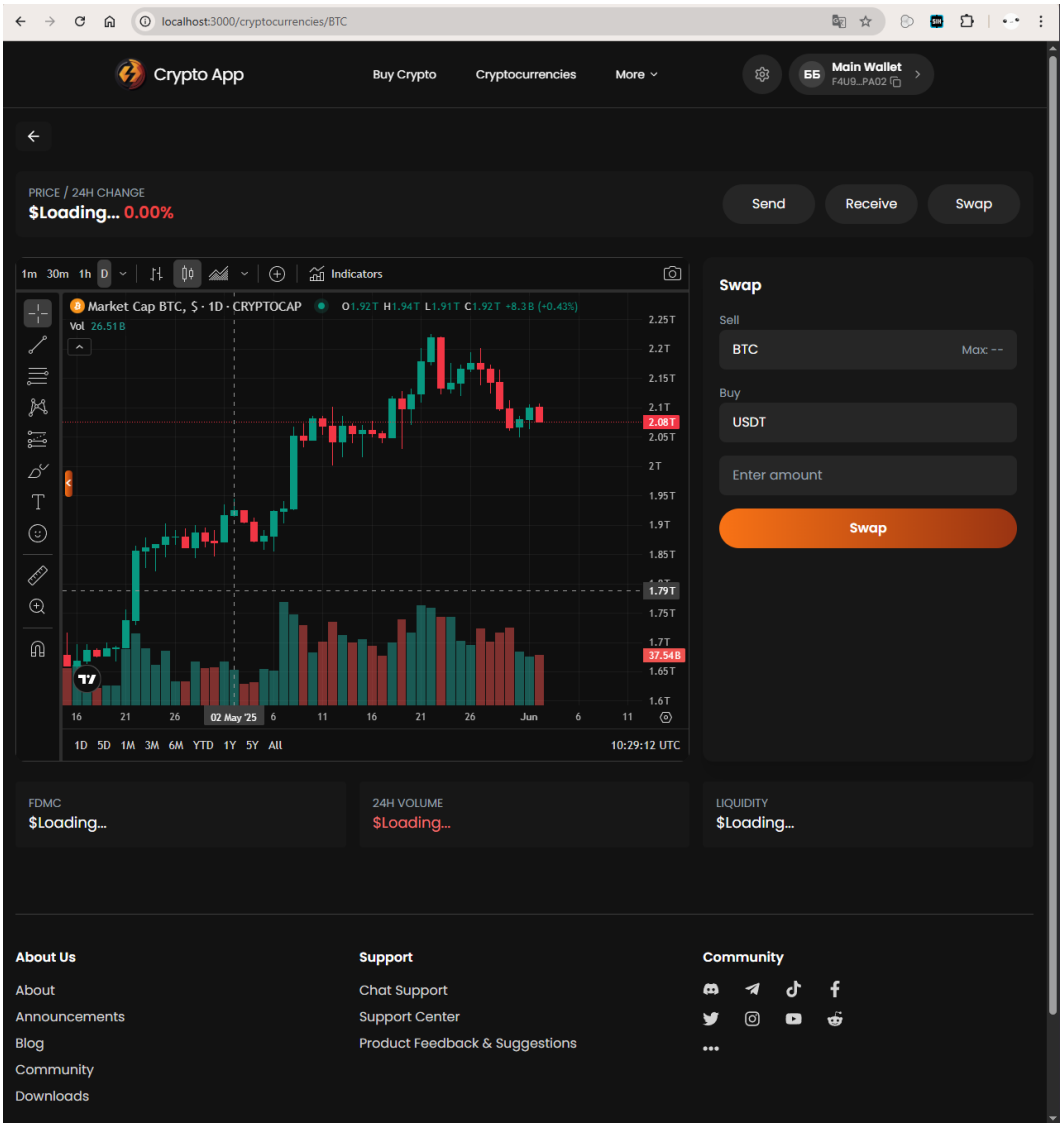


Рисунок А.3 – Графік зміни цін на криптовалюту

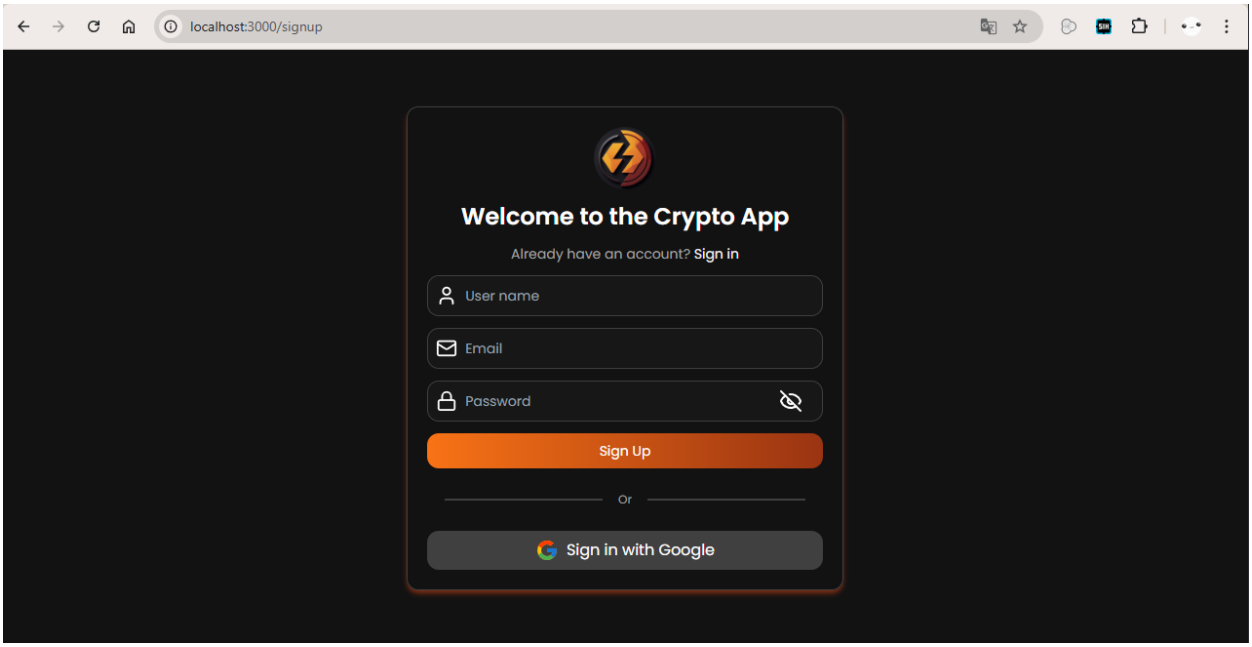


Рисунок А.4 – Форма авторизації користувача

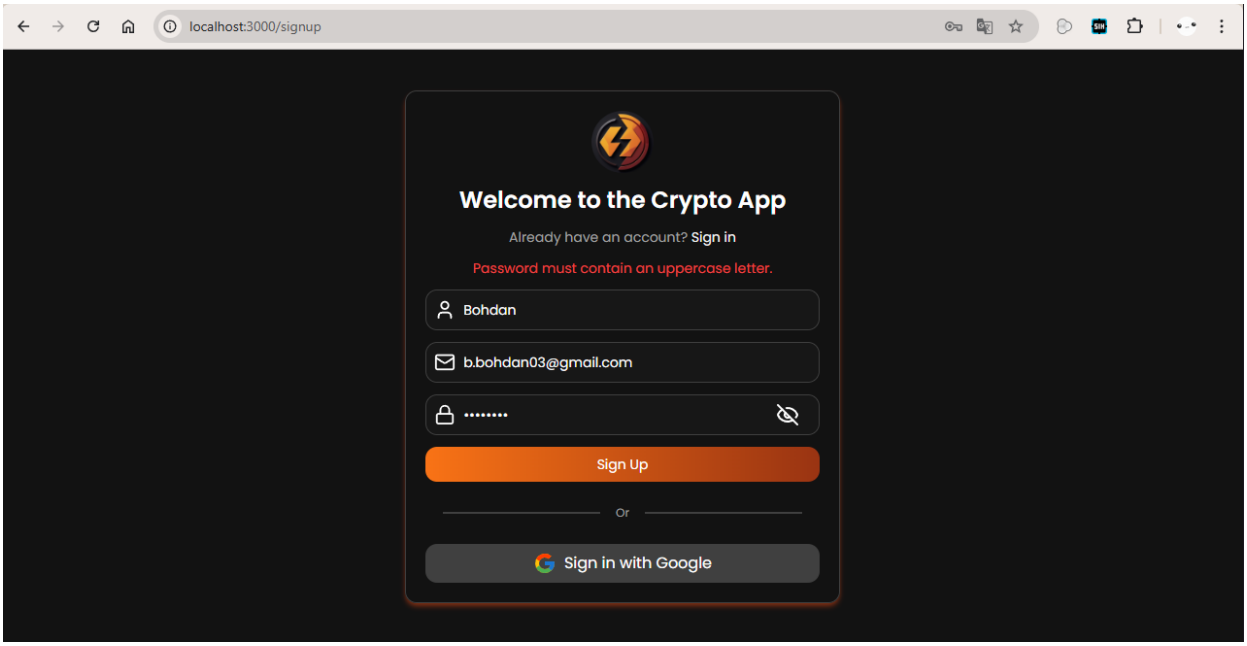


Рисунок А.5 – Помилка при авторизації

Мобільна адаптація компонентів сторінок (див. рис. А.6). Завдяки Tailwind CSS інтерфейс коректно масштабується під екрани смартфонів і зберігає повний функціонал.

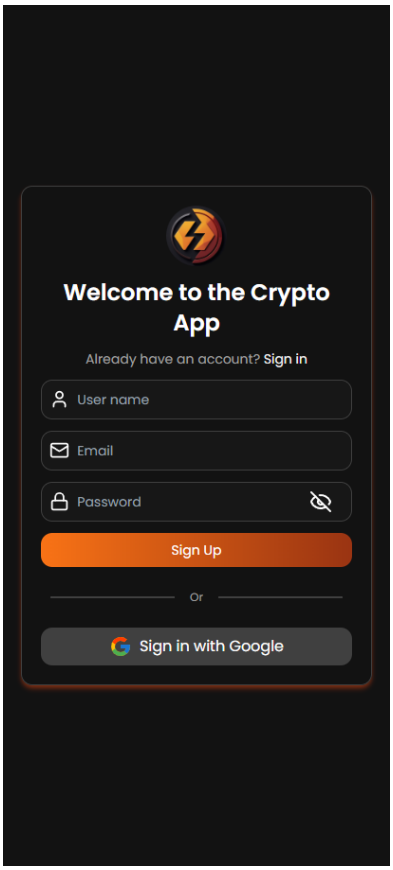


Рисунок А.6 – Адаптована сторінка під мобільні пристрої

Сторінка користувача після входу до системи (див. рис. А.7) має бокову панель, яка містить сторінки профілю (див. рис. А.8), портфеля (див. рис. А.9) і налаштування користувача.

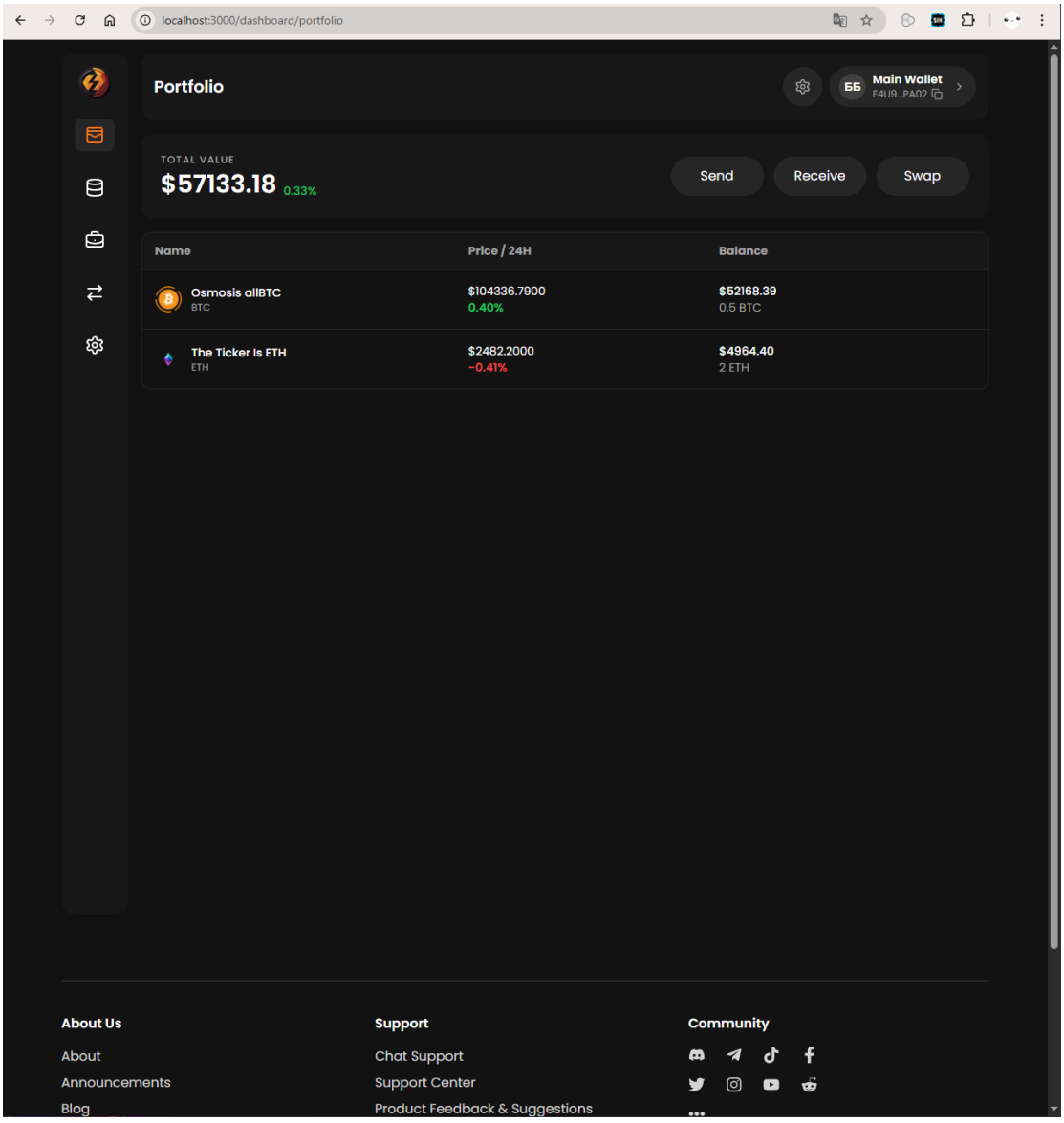


Рисунок А.7 – Авторизований користувач

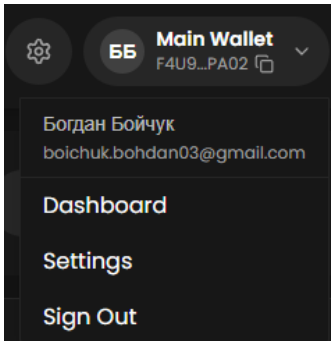


Рисунок А.8 – Профіль користувача

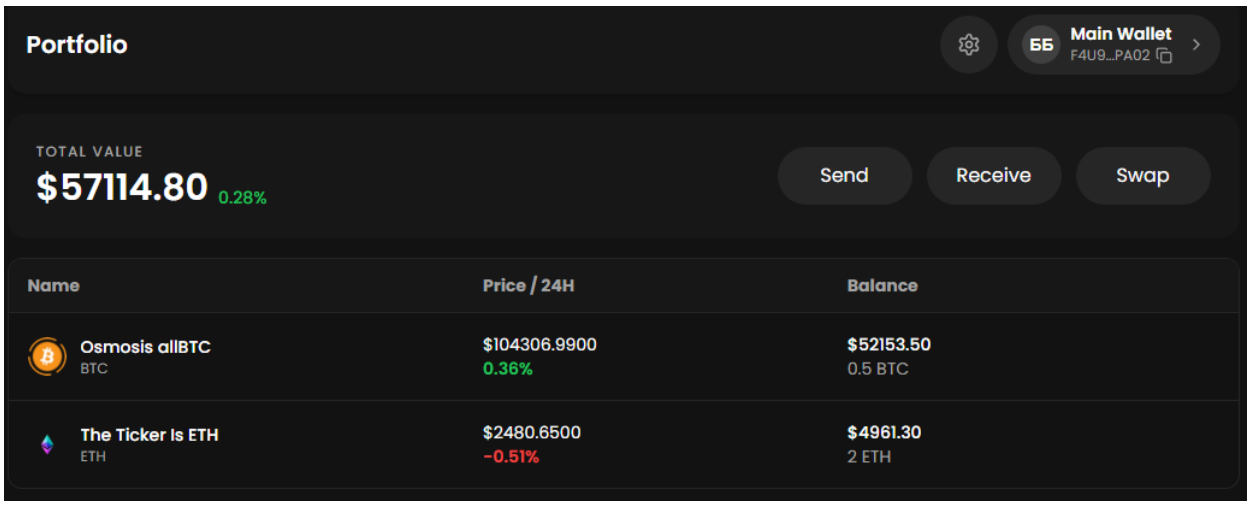


Рисунок А.9 – Портфель користувача

Активи користувача відображаються загальною вартістю портфеля, прибутком/збитком у грошовому вираженні та у відсотках (див. рис. А.10).

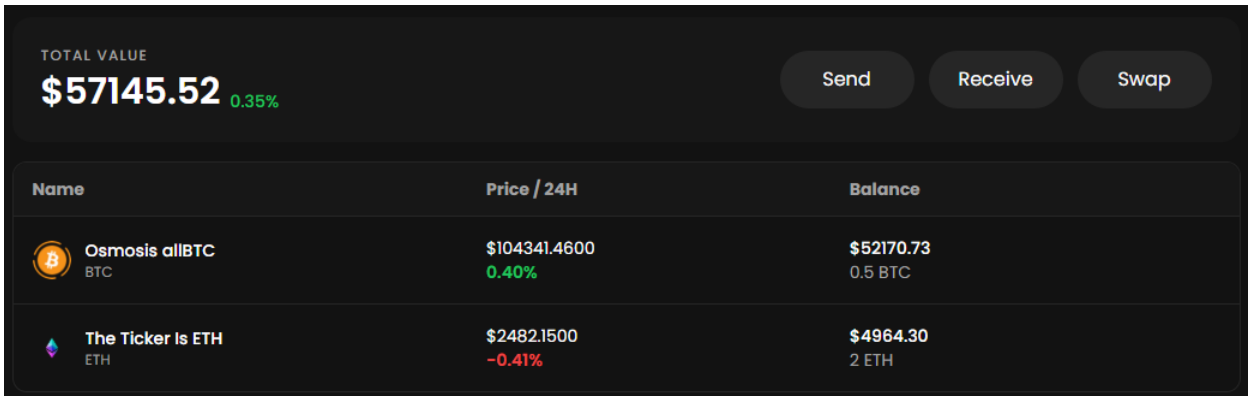


Рисунок А.10 – Портфель користувача

ДОДАТОК Б

Структура бази даних Firestore

Представлення структури даних, які використовуються у Firestore – хмарній NoSQL-базі даних від Firebase. Реалізована структура є документно-орієнтованою та дозволяє ефективно організовувати, зберігати й обробляти інформацію про користувачів і їхні криптовалютні портфелі.

Головна колекція users (див. рис. Б.1) містить документи користувачів, кожен з яких має вкладену підколекцію portfolios, а також доступ до колекцій settings.

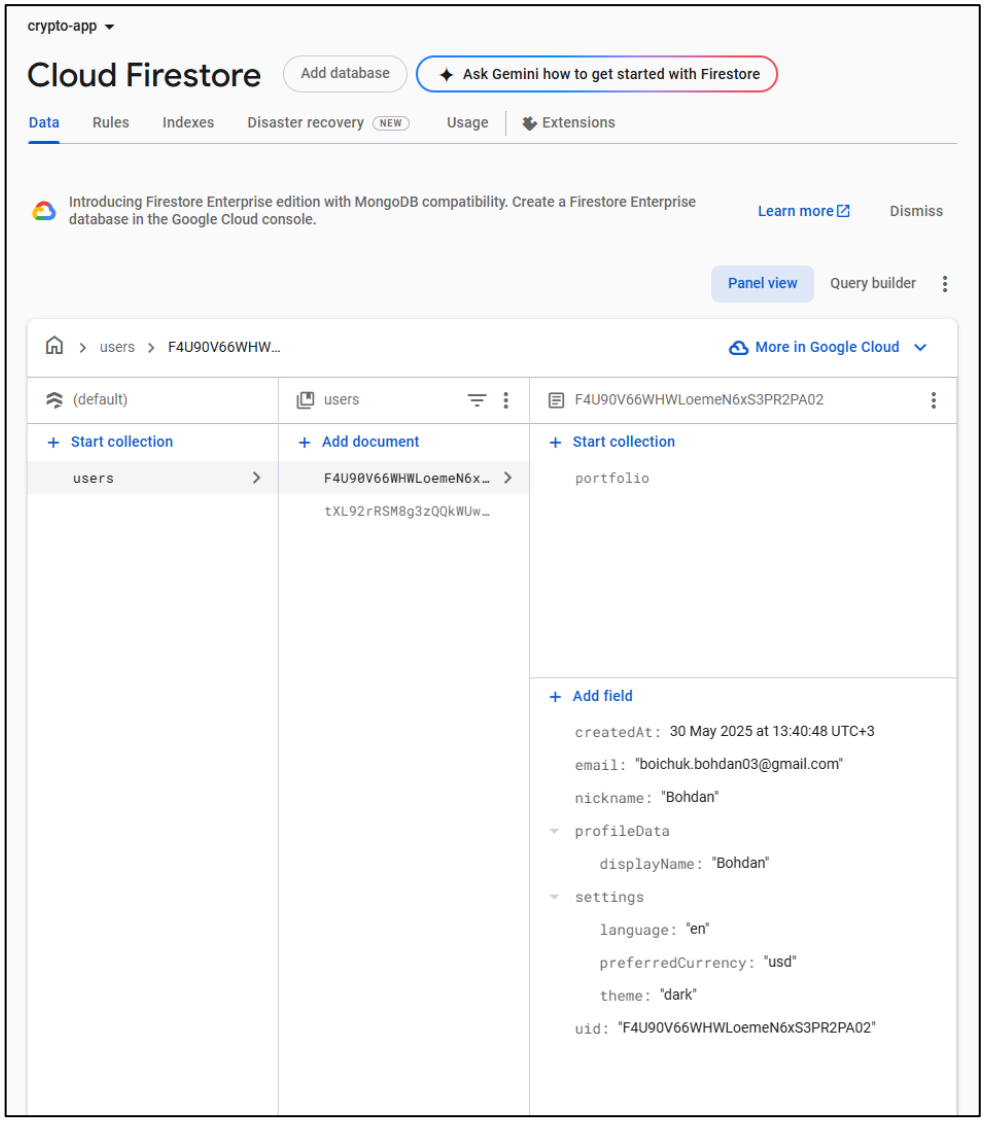


Рисунок Б.1 – Загальна схема структури Firestore

Документ користувача (див. рис. Б.2) містить UID, email, дата створення облікового запису, активний статус, посилання на підколекції.

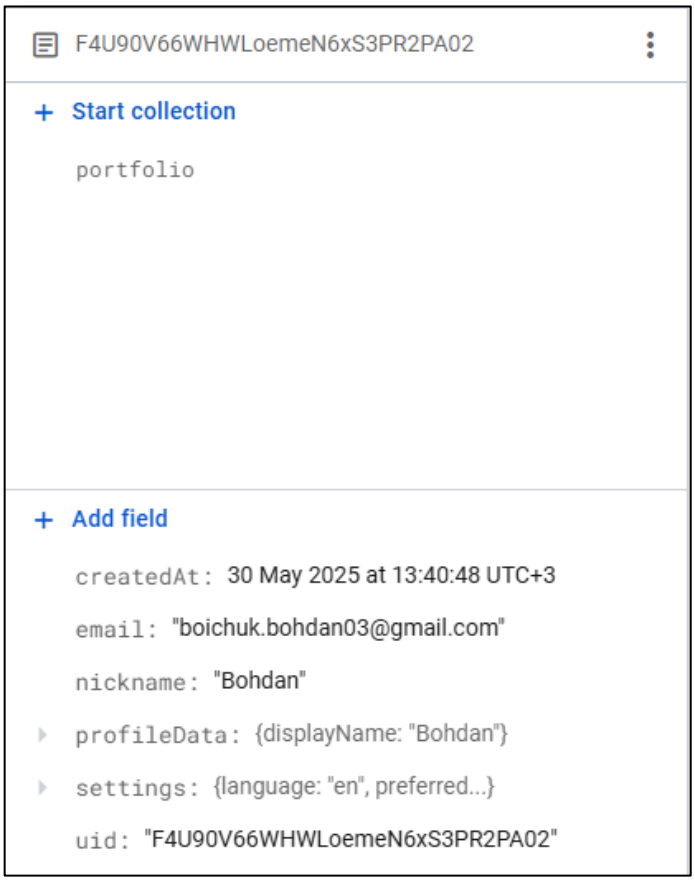


Рисунок Б.2 – Документ користувача

У колекції portfolios (див. рис. Б.3) кожен документ відповідає криптовалюті у портфелі, містить назву, символ, кількість.

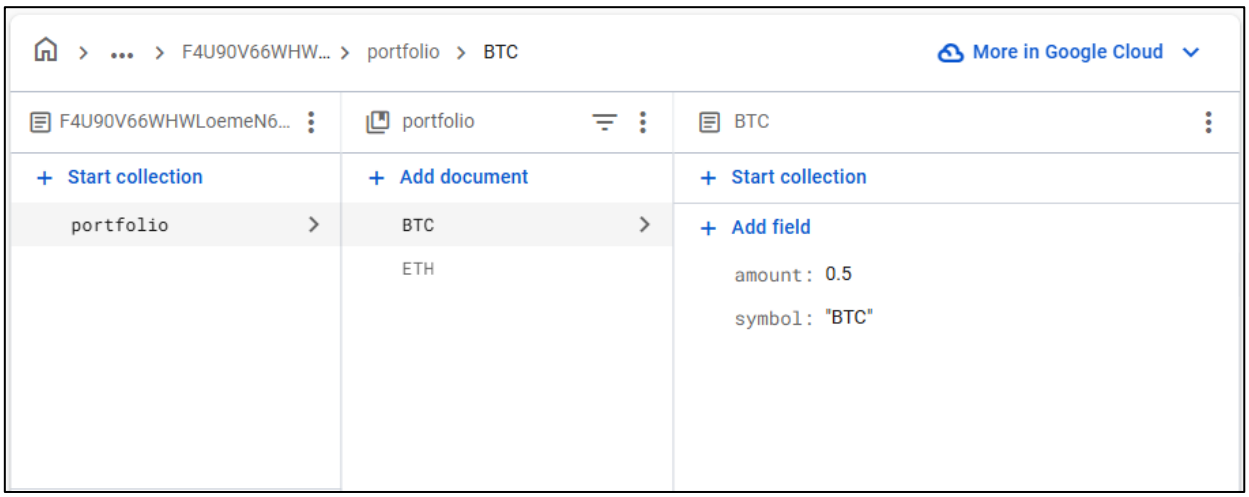


Рисунок Б.3 – Вміст підколекції portfolios

Колекція settings (див. рис. Б.4) зберігає індивідуальні параметри користувача – вибір мови, теми інтерфейсу, базової валюти, режиму відображення цін.



Рисунок Б.4 – Документ у колекції settings

Така структура дозволяє швидко отримувати, фільтрувати та оновлювати дані, а також гарантує масштабованість та ефективність при великій кількості користувачів.

ДОДАТОК В

Тестування вебдодатку

Результати ручного функціонального тестування основних компонентів вебдодатку, а також базову перевірку продуктивності та адаптивності інтерфейсу користувача.

Таблиця функціонального тестування (див. табл. В.1) охоплює сценарії реєстрації, авторизації, перегляду портфеля, редагування активів, збереження налаштувань, обробки помилок.

Таблиця В.1 – Таблиця функціонального тестування

№	Назва тесту	Вхідні дані	Очікуваний результат	Фактичний результат
1	Реєстрація нового користувача	Email, пароль	Користувача успішно зареєстровано	Користувача зареєстровано
2	Авторизація зареєстрованого користувача	Валідний email і пароль	Користувач потрапляє на головну сторінку	Авторизація успішна
3	Авторизація з неправильним паролем	Email + неправильний пароль	Повідомлення про помилку автентифікації	Повідомлення виведено
4	Перегляд криптопортфеля	Авторизований користувач	Виводиться таблиця з криптовалютами	Виводиться таблиця
5	Додавання нової монети до портфеля	Назва монети, кількість	Монету додано до таблиці портфеля	Монету додано
6	Редагування монети	Зміна кількості активу	Оновлено дані у таблиці портфеля	Дані оновлено
7	Видалення монети з портфеля	Натискання кнопки «Видалити»	Монету вилучено з таблиці	Монету вилучено
8	Збереження налаштувань	Тема: темна, валюта: USD	Налаштування збережено і застосовано	Тема змінилася, валюта USD
9	Побудова графіка ціни активу	Вибрана криптовалюта	Відображається інтерактивний графік (TradingView)	Графік відображено
10	Оновлення цін у реальному часі	Відкрита таблиця активів	Ціни змінюються автоматично без оновлення сторінки	Ціни оновлюються

Тест адаптивності в Chrome (див. рис. В.1) інтерфейс коректно масштабується та функціонує на смартфонах і планшетах, жодних візуальних дефектів не виявлено.

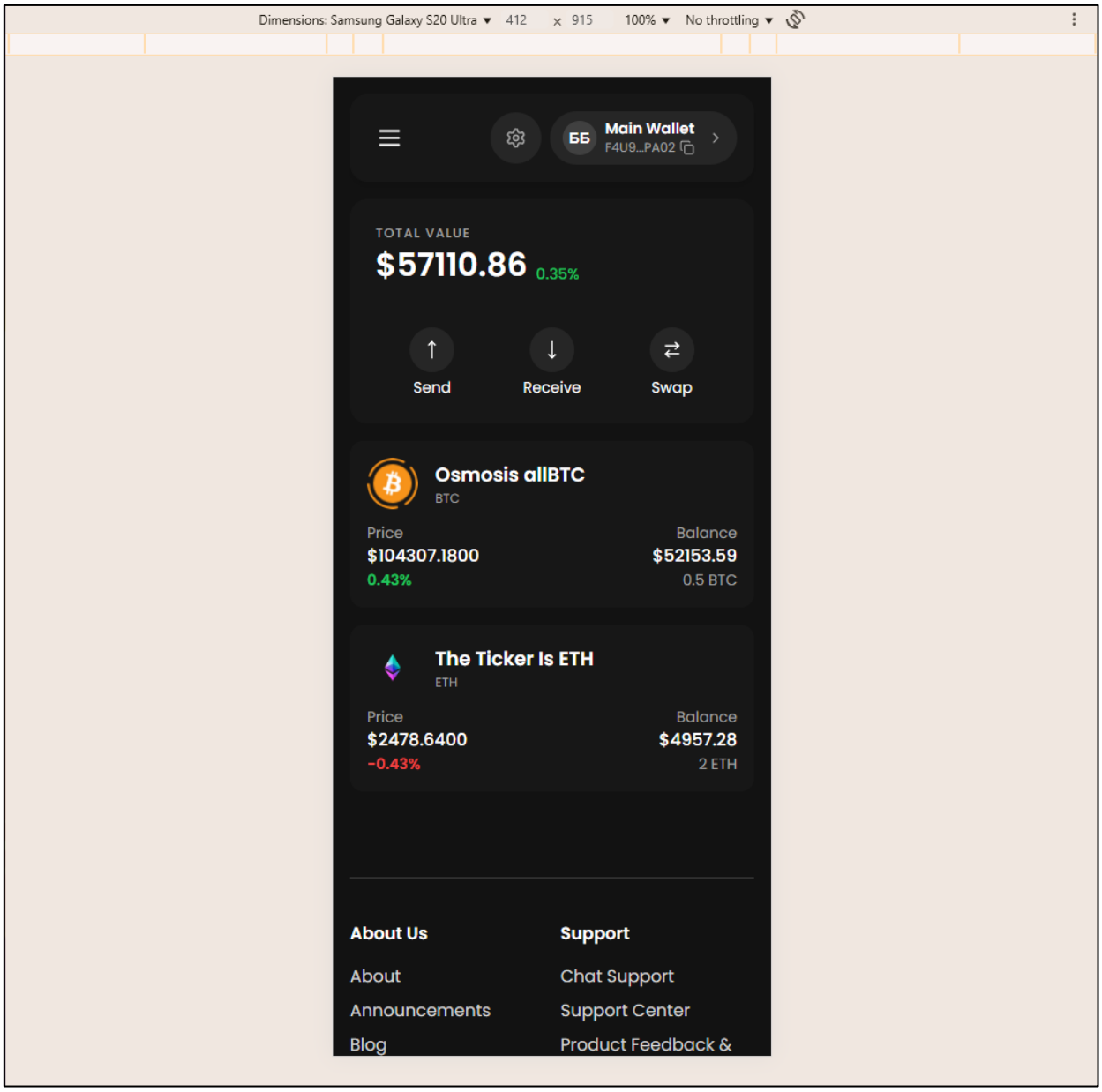


Рисунок В.1 – Тест адаптивності в Chrome

Отримані результати підтверджують стабільність та ефективність розробленого програмного забезпечення у типовому середовищі використання.