Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

«Вебзастосунок для моніторингу і обліку криптовалютного портфеля»
Назва теми

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.220194.01.01.ПЗ

Виконав студент IV курсу, група ІПЗ-22-1 _____ Роман БЕРЕЗНИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник канд. техн. наук, доцент _____ Оксана ЯШИНА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

Нормоконтролер канд. пед. наук, доцент _____ Наталія ПРАВОРСЬКА
Науковий ступінь, вчене звання Підпис Ім'я, ПРІЗВИЩЕ

До захисту допускаю:
Завідувач кафедри інженерії
програмного забезпечення _____ Леонід БЕДРАТЮК
Підпис Ім'я, ПРІЗВИЩЕ

1 червня 2026 р.

Хмельницький 2026

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Інженерії програмного забезпечення

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри ІПЗ

_____ Л. П. Бедратюк

02.01.2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Березному Роману Юрійовичу

Прізвище, ім'я, по батькові студента

1. Тема роботи Вебзастосунок для моніторингу і обліку криптовалютного портфеля

Керівник роботи Яшина Оксана Миколаївна, канд. техн. наук, доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 20.01.2026 р. № 7

2. Строк подання студентом роботи на кафедру 01.06.2026 р.

3. Вихідні дані до роботи Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження предметної області та постановка задач; проєктування вебзастосунку; програмна реалізація та тестування вебзастосунку.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Три креслення:

1. UML-діаграма варіантів використання.
2. Діаграма зв'язків модулів.
3. UML-діаграма класів серверної частини

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Праворська Н. І., доцент	04.05.26	06.05.26
Антиплагіат	Форкун Ю. В., доцент	05.05.26	06.05.26

7. Дата видачі завдання «02» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1 Ознайомлення з тематикою дипломного проектування, визначення та узгодження індивідуальних тем кваліфікаційних робіт (КвР)	01.12– 31.12.2025	
2 Збір матеріалу за темою КвР; дослідження предметної області, в якій планується використання програмного забезпечення (ПЗ), визначення задач та вимог, розробка технічного завдання	01.01 – 20.02.2026	
3 Проектування програмного забезпечення	21.02 – 20.03 2026	
4 Програмна реалізація з використанням відповідних засобів розробки. Тестування ПЗ	21.03 – 30.04.2026	
5 Написання вступу, загальних висновків, оформлення переліку джерел посилання та додатків. Оформлення пояснювальної записки КвР згідно вимог	01.05 – 25.05.2026	
6 Попередній захист КвР	травень 2026	Згідно графіка
7 Перевірка КвР на плагіат, нормоконтроль, отримання відгуків, рецензій та інших супровідних документів. Брошурування (зшиття) пояснювальної записки.	26.05 – 30.06.2026	
8 Здача КвР на кафедру; підготовка КвР для розміщення у репозитарії ХНУ; підготовка до захисту та захист КвР	з 01.06.2026	

Студент _____ Роман БЕРЕЗНИЙ
Підпис Ім'я, ПРІЗВИЩЕ

Керівник роботи _____ Оксана ЯШИНА
Підпис Ім'я, ПРІЗВИЩЕ

АНОТАЦІЯ

Тема кваліфікаційної роботи «Вебзастосунок для моніторингу і обліку криптовалютного портфеля».

Автор роботи: Березний Роман Юрійович.

Керівник роботи: Яшина Оксана Миколаївна.

Пояснювальна записка: 79 с., 31 рис., 12 табл., 2 дод., 49 джерел.

Графічна частина: 3 креслення ф. А3.

КРИПТОВАЛЮТНИЙ ПОРТФЕЛЬ, ВЕБЗАСТОСУНОК, БАЗА ДАНИХ, POSTGRESQL, PYTHON, REACT, FASTAPI, OPENAI, WEBAUTHN, API.

Мета кваліфікаційної роботи: проєктування та розроблення вебзастосунку для моніторингу й обліку криптовалютного портфеля з підтримкою багатогаманцевого обліку, ринкових курсів із відкритих API, аналітичних інструментів і вбудованого AI-помічника.

У кваліфікаційній роботі проведено аналіз предметної області моніторингу та обліку криптоактивів, розглянуто наявні програмні рішення, визначено функціональні та нефункціональні вимоги до програмної системи, спроєктовано клієнт-серверну архітектуру (SPA + REST/SSE), розроблено модель реляційної бази даних, реалізовано модулі автентифікації (пароль + WebAuthn-passkeys), обліку гаманців і операцій, отримання та кешування ринкових даних, аналітики, а також AI-агента з потоковими відповідями та підтримкою вебпошуку.

Для реалізації програмного продукту використано систему керування базами даних PostgreSQL, мову програмування Python (FastAPI, SQLAlchemy, Alembic) для серверної частини, TypeScript із бібліотекою React і Tailwind CSS – для клієнтської.

Інтерфейс реалізовано з підтримкою трьох мов (українська, англійська, польська) та світлої й темної тем. За допомогою цих засобів розроблено вебзастосунок для ведення гаманців і активів, обліку операцій, отримання ринкових курсів із відкритих API (Binance, CoinGecko), відображення аналітики

портфеля та надання користувачу інтелектуальних пояснень через AI-агента, що працює з узагальненим знімком портфеля

Практичне значення роботи полягає у створенні працездатного вебзастосунку, який може використовуватися для персонального обліку криптоактивів, а також слугувати основою для подальшого розширення (підтримка мереж, контрактів токенів, розширена аналітика, автоматизовані імпорт та експорт даних, покращені механізми автентифікації тощо).

26 травня 2026р.
Дата

Підпис

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
ПЗ	–	програмне забезпечення
СКБД	–	система керування базами даних
API	–	Application Programming Interface
SPA	–	Single-Page Application
REST	–	Representational State Transfer
SSE	–	Server-Sent Events
UI	–	User Interface
UML	–	Unified Modeling Language
ER	–	Entity-Relationship
OWASP	–	Open Worldwide Application Security Project
ASVS	–	Application Security Verification Standard
WEBAUTHN	–	Web Authentication
FIDO2	–	Fast Identity Online 2
JWT	–	JSON Web Token
ORM	–	Object-Relational Mapping
HTTP	–	HyperText Transfer Protocol
E2E	–	End-to-End
AI	–	Artificial Intelligence

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.220194.01.01.ПЗ	Пояснювальна записка	79		
2	А4		Завдання на кваліфіка-	1		
			ційну роботу			
3	А4		Анотація	2		
			Графічні документи			
4	А3	КвРІПЗ.220194.01.01.Е8	UML-діаграма варіантів	1		
			використання			
5	А3	КвРІПЗ.220194.01.01.Е8	Діаграма зв'язків	1		
			модулів			
6	А3	КвРІПЗ.220194.01.01.Е8	UML-діаграма класів	1		
			серверної частини			

Змн.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ.220194.01.01.ПЗ		
Виконав		Березний Р.Ю.		26.05	Вебзастосунок для моніторингу і обліку криптовалютного портфеля	Літ.	Арк. 6 / Акрушів 79
Керівник		Яшина О. М.		26.05			
Н. Контр.		Праворська Н. І.		26.05		ХНУ, ІПЗ-22-1	
Зав. каф.		Бедратюк Л. П.		26.05			

ЗМІСТ

ВСТУП 9

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 12

1.1 Змістовий аналіз предметної області 12

1.2 Аналіз наявного програмно-технічного забезпечення предметної області 15

1.3 Визначення вимог до вебзастосунку 20

1.4 Висновки. Постановка задачі 26

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ 29

2.1 Аналіз архітектурних підходів 29

2.2 Обґрунтування архітектури вебзастосунку 31

2.3 Декомпозиція системи на модулі 33

2.4 Проєктування моделі даних 35

2.5 Аналіз і вибір технологій реалізації 39

2.6 Проєктування інтерфейсу користувача 43

2.7 Проєктування безпеки та сесій 51

2.8 Висновки щодо проєктування вебзастосунку 54

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ 57

3.1 Особливості програмної реалізації 57

3.2 Програмна реалізація функціональних модулів 59

3.3 Реалізація інтерфейсу користувача 65

3.4 Вимоги до технічних та програмних засобів 69

3.5 Тестування вебзастосунку 71

3.5.1 Аналіз методів тестування 71

3.5.2 Аналіз результатів тестування 73

3.6 Висновки до третього розділу 77

ВИСНОВКИ 78

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 80
ДОДАТОК А 85
ДОДАТОК Б 107
ГРАФІЧНА ЧАСТИНА 115

ВСТУП

Сфера цифрових активів та криптовалют за останні роки перетворилася з нішевого явища на повноцінний сегмент фінансових технологій. Разом із розвитком інфраструктури (біржі, некастодіальні гаманці, різні блокчейн-мережі, токени та стейблкоїни) зростає й кількість користувачів, які зберігають активи одночасно на кількох платформах і в різних валютах. У таких умовах актуальною стає задача не лише зберігання коштів, а й їх системного обліку: користувачу важливо швидко оцінити загальну вартість портфеля, структуру активів, історію операцій та зміну прибутковості в часі.

Особливість криптовалютного ринку полягає у високій динаміці цін, наявності значної кількості активів та мереж, а також у розпорошеності даних. На практиці облік портфеля часто ведеться вручну (таблиці, нотатки або прості додатки), що призводить до помилок, втрати історії та ускладнює аналіз. Окремою проблемою є поєднання двох типів інформації: з одного боку – власних даних користувача (гаманці, залишки, транзакції), з іншого – ринкових даних (курси, зміна вартості, історичні графіки), які потрібно отримувати з відкритих джерел та регулярно оновлювати.

З огляду на це, розроблення вебзастосунку для моніторингу й обліку криптовалютного портфеля є актуальним завданням. Такий застосунок має забезпечити зручне ведення переліку власних гаманців і активів, фіксацію операцій (поповнення, виведення, перекази, конвертації), автоматичне підвантаження актуальних цін з відкритих API, а також інструменти візуалізації – зведені показники та графіки. На відміну від сервісів, що здійснюють торгівлю або пряме підключення до гаманців, у межах цієї роботи застосунок орієнтований на облік і аналіз: користувач вводить дані самостійно, а система виконує збереження, агрегацію та аналітичні розрахунки. Це дозволяє зосередитися на якості програмної реалізації, моделі даних, надійності обчислень, зручності інтерфейсу та безпеці облікового запису.

Складовою сучасних рішень є інтелектуальні функції для пояснення стану портфеля та надання довідкової інформації. У межах роботи передбачено реалізацію модуля AI-агента, який отримує знеособлений знімок портфеля користувача (структура активів, частки, історія змін, прибуток і збиток) і допомагає інтерпретувати ринкові зміни на основі відкритих джерел і вебпошуку. Архітектурно агент відокремлений від основного облікового модуля, не виконує дій із коштами користувача та має лише допоміжну, пояснювальну роль. Метою кваліфікаційної роботи є проєктування та розроблення вебзастосунку для моніторингу й обліку криптовалютного портфеля з підтримкою багатогаманцевого обліку, ринкових курсів із відкритих API, аналітичних інструментів і вбудованого AI-помічника.

Для досягнення поставленої мети необхідно розв’язати такі завдання:

- виконати аналіз архітектурних підходів до побудови вебзастосунків та обґрунтувати вибір архітектури розроблюваної системи (розподіл на клієнтську і серверну частини, принципи взаємодії, підходи до безпеки);
- розробити загальну структуру проєкту та декомпозицію на модулі, визначити їх призначення та взаємодію (модуль обліку гаманців, модуль операцій, модуль розрахунків, модуль ринкових даних, модуль аналітики, модуль профілю);
- спроєктувати модель даних для зберігання користувачів, гаманців, активів, журналу операцій, налаштувань профілю та ринкових котирувань;
- розробити макети (прототипи) користувацького інтерфейсу та визначити основні сценарії взаємодії з системою;
- виконати аналіз і вибір технологій реалізації (інструменти клієнтської та серверної частини, база даних, бібліотеки та підходи для візуалізації графіків, інтеграція з API ринкових даних);
- реалізувати модуль керування гаманцями та активами в гаманцях, забезпечивши коректну роботу CRUD-операцій;

– реалізувати модуль операцій (поповнення, виведення, переказ, конвертація) та автоматичне оновлення балансів активів із фіксацією історії;

– реалізувати модуль отримання ринкових даних (котирувань) із зовнішнього API, кешування, оновлення та використання цих даних у розрахунках;

– реалізувати модуль безпарольної автентифікації (WebAuthn / FIDO2): реєстрацію паскеїв, вхід через паскей, перегляд і керування переліком зареєстрованих автентифікаторів;

– реалізувати журнал входів користувача із шифруванням чутливих метаданих та визначенням приблизного географічного контексту;

– реалізувати багатомовний інтерфейс (українська, англійська, польська) і підтримку світлої й темної тем оформлення з реактивним перемиканням без перезавантаження сторінки.

– реалізувати модуль аналітики: відображення графіків вартості активів і/або загальної оцінки портфеля;

– реалізувати модуль AI-агента: ведення чатів зі збереженням історії, потокову подачу відповідей, формування знеособленого знімка портфеля як контексту, інтеграцію з відкритим API OpenAI з підтримкою вибору моделі та інструмента вебпошуку, типізовану обробку помилок зовнішнього API

– визначити підхід до тестування (функціональне, модульне, інтеграційне) та виконати тестування основних сценаріїв;

– проаналізувати результати реалізації та тестування, сформулювати висновки і перспективи подальшого розвитку системи.

Практичне значення роботи полягає у створенні працездатного вебзастосунку, який може використовуватися для персонального обліку криптоактивів, а також слугувати основою для подальшого розширення (підтримка мереж, контрактів токенів, розширена аналітика, автоматизовані імпорт та експорт даних, покращені механізми автентифікації тощо).

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовий аналіз предметної області

Сегмент криптоактивів став помітною частиною фінтех-сфери: користувачі працюють із біржами, некастодіальними гаманцями, різними мережами та великою кількістю активів. Через це портфель однієї людини часто розподілений між кількома джерелами, а отже потребує єдиного інструмента для оцінки загальної вартості, структури активів, історії операцій і зміни показників у часі.

Поширення криптоінфраструктури підтверджується аналітичними дослідженнями. Звіт Cambridge Centre for Alternative Finance показує зростання кількості користувачів і акаунтів у криптосервісах, причому ці значення є агрегованими оцінками провайдерів сервісів [1].

Рисунок 1.1 показує зростання кількості користувачів і акаунтів, що підтверджує типову ситуацію з кількома джерелами зберігання в одного користувача. Для обліку це означає потребу в агрегуванні даних портфеля

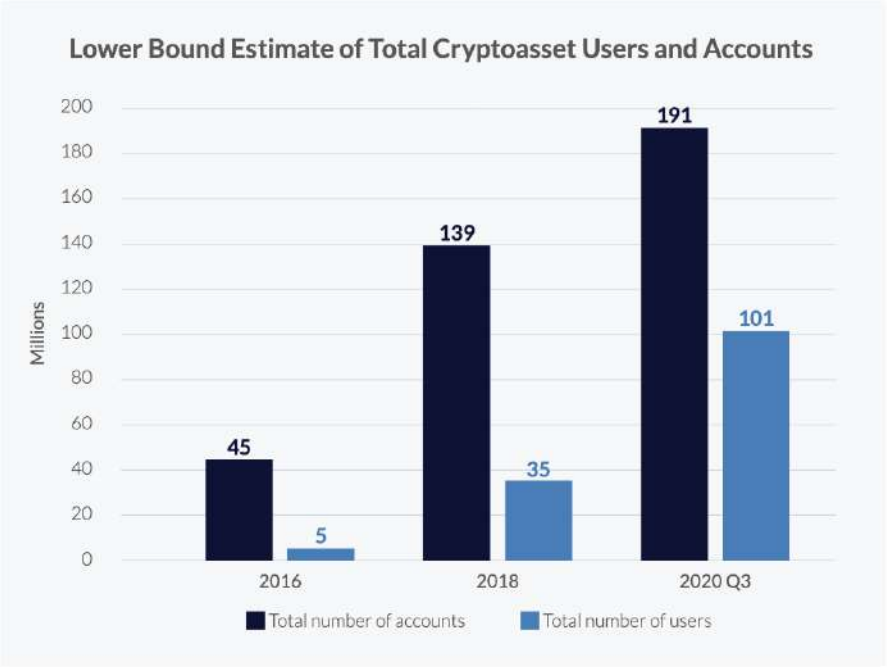


Рисунок 1.1 – Динаміка оцінок кількості користувачів і акаунтів криптосервісів за CCAF

Активність на крипторинку є нерівномірною: аналітика Chainalysis демонструє періоди зростання та спадів [2]. Такі коливання впливають на частоту операцій, інтерес до окремих активів і потребу в історичній аналітиці.

Рисунок 1.2 підкреслює, що користувачу потрібні не лише поточні залишки, а й історія подій, яка пояснює зміну оцінки портфеля в різні періоди.

Рисунок 1.2 – Приклад зміни глобальної активності та залучення до криптоактивів у часі за Chainalysis

Важливим компонентом криптоекосистеми є стейблкоїни – активи, вартість яких орієнтована на стабільність відносно фіатної валюти або кошика активів [4]. Вони часто використовуються як проміжна валюта для операцій, а ЄЦБ відзначає помітне зростання цього сегмента [3].

Рисунок 1.3 демонструє зростання ринку стейблкоїнів і домінування найбільших долар-номінованих інструментів. Тому система обліку повинна коректно враховувати активи, що використовуються як базові для оцінки та операцій.

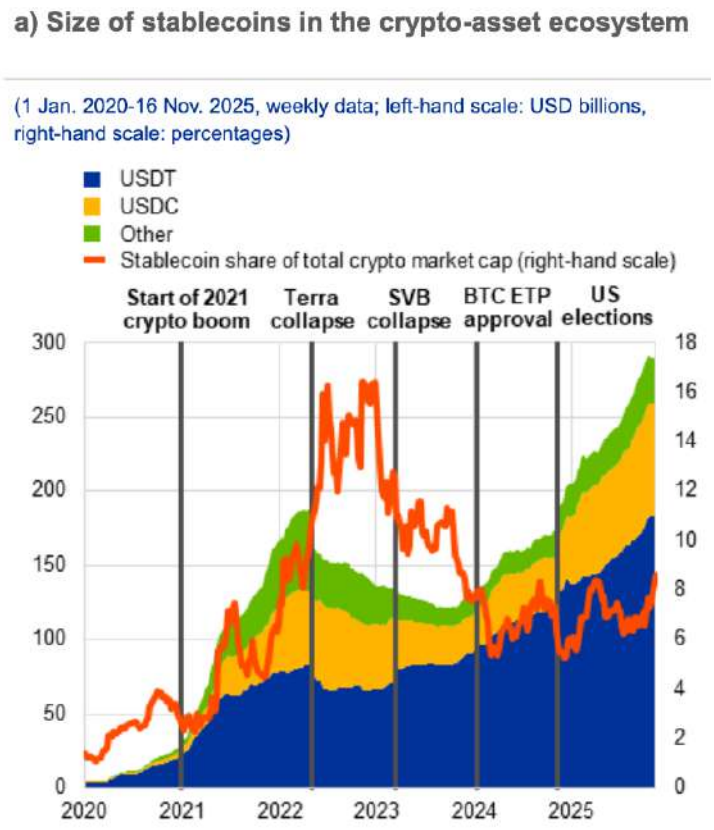


Рисунок 1.3 – Зростання ринку стейблкоїнів і структура часток найбільших за ECB FSR

З точки зору змісту предметної області портфельний облік у вебзастосунку можна описати як поєднання двох груп даних:

– дані користувача (перелік гаманців і джерел зберігання, залишки активів на кожному з них, а також події (операції), що змінюють ці залишки);

– ринкові дані (курси активів (і за потреби історія курсів), необхідні для перерахунку портфеля в базову валюту та побудови графіків).

У термінологічному сенсі віртуальні активи розглядаються як цифрове представлення вартості, що може передаватися або обмінюватися в цифровій формі [5]. У портфельному обліку це відповідає об’єктам: гаманець, актив, залишок, транзакція та оцінка у вибраній базовій валюті. Типові операції: поповнення, виведення, переказ, конвертація та інші події, що змінюють стан портфеля.

Ключова специфіка саме криптовалютного портфельного обліку полягає у відокремленні двох причин змін загальної вартості портфеля:

- власні дії користувача (операції, що змінюють кількість активів);
- зміни ринкової ціни активів у часі, які впливають на оцінку портфеля навіть за незмінних залишків [2], [3].

Таким чином, предметна область визначає потребу в вебзастосунку, який:

– забезпечує структуроване збереження даних про гаманці та операції;

– використовує зовнішні джерела котирувань для оцінки портфеля;

– надає користувачу зрозуміле зведення і візуалізацію змін у часі, спираючись на історію транзакцій та ринкові коливання.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Для формування вимог проаналізовано сервіси портфельного відстеження та обліку, а не торгові платформи. Це дозволяє визначити типові функції, способи подання даних і обмеження готових продуктів.

Під час аналізу використовуються такі критерії:

- спосіб формування портфеля (ручне введення, імпорт, підключення акаунтів);
- структура обліку (наявність журналу операцій, категорій подій, історії змін);
- представлення даних (зведення, списки, графіки, динаміка);
- зручність інтерфейсу для щоденного використання;
- придатність для сценарію персонального обліку без здійснення платежів і торгівлі.

CoinStats є прикладом портфельного трекера з акцентом на підключення бірж, гаманців і DeFi-протоколів [6], [13]. Для роботи він важливий як підтвердження потреби в єдиному представленні активів незалежно від місця їх зберігання.

Рисунок 1.4 демонструє інтерфейс CoinStats. Його сильна сторона – швидке зведення даних і широка підтримка джерел, але для цього проєкту обрано контрольований ручний облік гаманців і операцій без прямого підключення до коштів користувача.

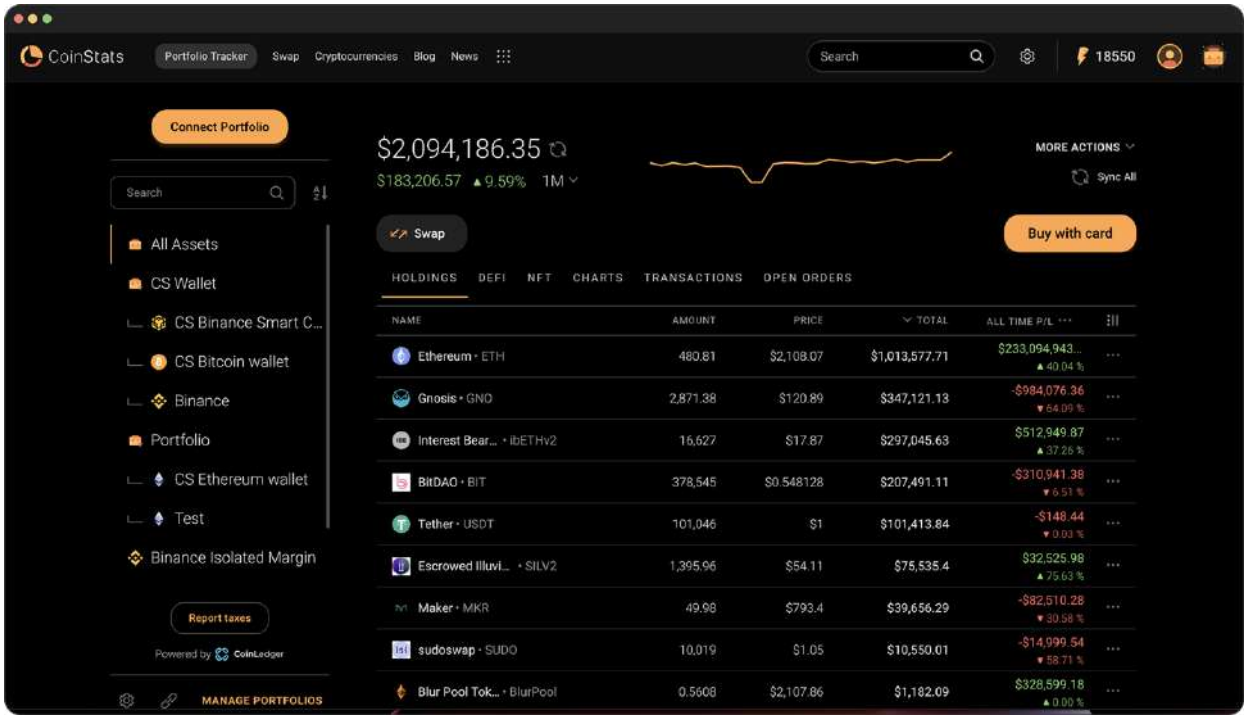


Рисунок 1.4 – Інтерфейс сервісу CoinStats

Delta by eToro є універсальним трекером інвестицій, що підтримує кілька класів активів і відображення результатів у реальному часі [7]. Для роботи він корисний як приклад поєднання загального балансу, структури портфеля, історії змін і динаміки вартості активів. Такий підхід дає змогу користувачу швидко оцінювати поточний стан інвестицій, відстежувати зміну вартості окремих позицій і виявляти основні зміни у портфелі.

Рисунок 1.5 показує інтерфейс Delta. Його інвестиційна спрямованість корисна як UX-орієнтир, зокрема щодо подання зведеної інформації, графіків і переліку активів. Водночас розроблюваний застосунок фокусується саме на криптовалютному обліку: гаманцях, залишках, операціях і конвертаціях.

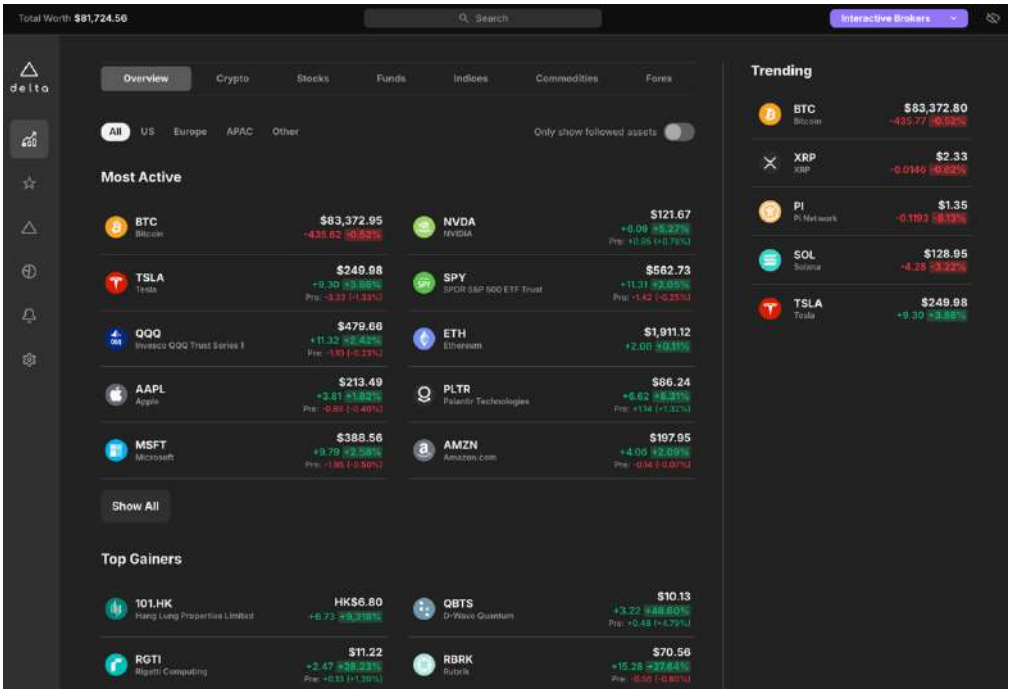


Рисунок 1.5 – Інтерфейс Delta by eToro

CoinMarketCap Portfolio поєднує ринкові дані з базовим портфельним функціоналом: користувач може відстежувати баланс, прибуток і збиток [8].

Рисунок 1.6 демонструє інтерфейс наведеного вебзастосунку. Перевагою є поєднання котирувань і власного портфеля, однак такі сервіси більше орієнтовані на моніторинг оцінки, ніж на детальний журнал внутрішніх операцій.

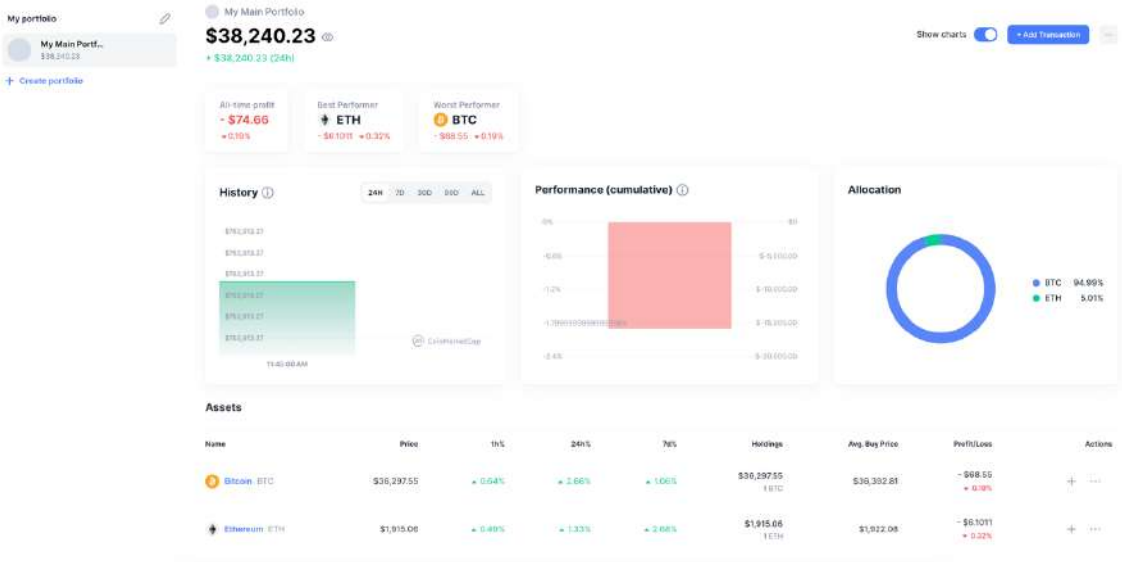


Рисунок 1.6 – Інтерфейс CoinMarketCap Portfolio

CoinGecko Portfolio поєднує ринкову платформу та простий портфельний інтерфейс, доступний у веб- і мобільному форматі [9].

Рисунок 1.7 демонструє інтерфейс досліджуваного вебзастосунку. Для проєкту важливі дві його риси: просте подання портфеля та наявність офіційного API для поточних і історичних ринкових даних [9], [11].

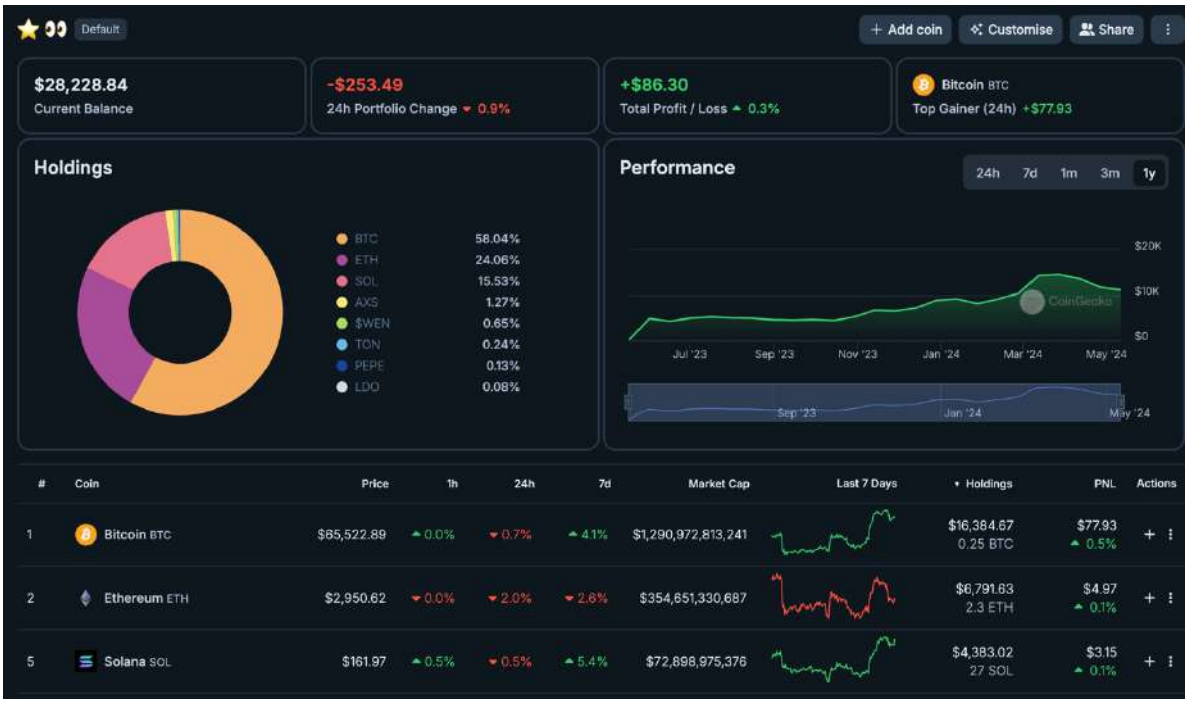


Рисунок 1.7 – Інтерфейс CoinGecko Portfolio

CoinTracking сильніше орієнтований на облік транзакцій, імпорт даних і звітність [10]. Це приклад системи, де центральним елементом є не лише поточний баланс, а й детальна історія операцій, джерела надходження активів і зміни вартості. Такий підхід демонструє важливість збереження повної історії дій користувача для аналізу руху коштів і стану портфеля.

Рисунок 1.8 демонструє користувацький інтерфейс сервісу CoinTracking. Для КвР корисною є ідея деталізованого журналу подій, фільтрації операцій і перегляду історії змін за окремими активами. Водночас у розроблюваному застосунку ці можливості доцільно реалізовувати без надлишкових податкових і звітних функцій, які виходять за межі поставленої задачі.

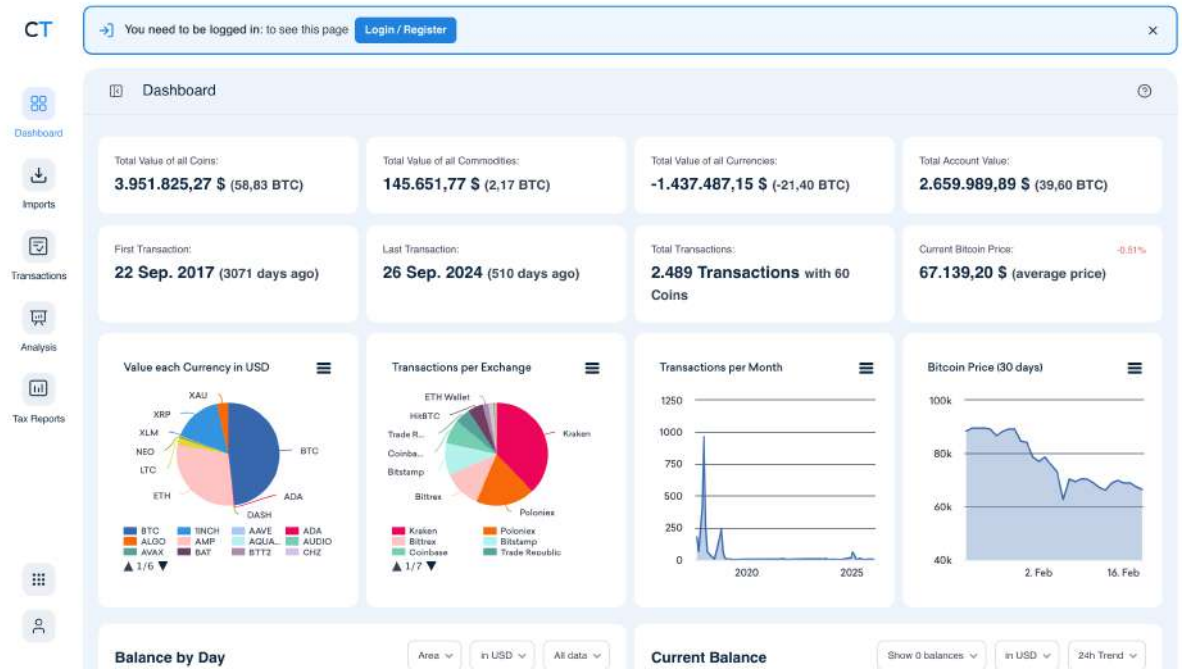


Рисунок 1.8 – Інтерфейс CoinTracking

Спільним для портфельних сервісів є використання зовнішніх API для ринкових даних. У власному застосунку доцільно спиратися на CoinGecko API та Binance Spot API, які надають дані для оцінки портфеля й побудови графіків [11], [12].

Для коректного опису предметної області використано термінологію FATF щодо virtual assets і VASP [5], оскільки робота оперує поняттями бірж, гаманців і сервісів криптоінфраструктури.

За результатами аналізу розглянутих рішень можна виділити кілька спільних характеристик сучасного програмного забезпечення цієї предметної області.

Аналіз аналогів показав три ключові вимоги до розроблюваного застосунку: зведене представлення портфеля, наявність графіків і підтримку журналу операцій [6]-[12]. Такі можливості забезпечують швидке розуміння стану активів, оцінювання динаміки вартості та контроль дій. При цьому для КвР обрано ручний контрольований облік із автоматичним отриманням ринкових курсів, що поєднує простоту використання з достатньою точністю даних.

Отже, існуючі продукти добре вирішують задачі моніторингу, але не завжди однаково повно покривають прозорий персональний облік операцій. Тому розроблюваний застосунок має поєднати:

- простий і зрозумілий інтерфейс портфеля;
- окремий журнал операцій із фіксацією контексту кожної події;
- автоматичне отримання ринкових курсів із відкритих API;
- базову аналітику та графіки для оцінки динаміки вартості активів.

1.3 Визначення вимог до вебзастосунку

Проведений аналіз дає підстави сформулювати вимоги до вебзастосунку як до системи персонального обліку: користувач самостійно веде гаманці, активи й операції, а система зберігає дані, агрегує портфель і формує аналітичні подання.

Оскільки застосунок працює з фінансовими записами користувача, важливими є автентифікація, керування сесіями та захист паролів відповідно до практик OWASP [14], [15]. Ринкові оцінки та графіки формуються через зовнішні API [11], [12], а AI-модуль має працювати лише з необхідним знеособленим контекстом.

До функціональних вимог, сформованих на основі типових сценаріїв користувача та логіки портфельного обліку, належать:

а) автентифікація та робота з профілем:

1) реєстрація облікового запису користувача (email + надійний пароль із індикатором сили);

2) вхід у систему за паролем;

3) додавання та використання паскеїв (WebAuthn / FIDO2) як другого або альтернативного фактора автентифікації;

4) керування активними паскеями (перегляд переліку, видалення);

5) перегляд журналу входів (час, IP, браузер, ОС, місто/країна) із шифруванням метаданих у базі;

6) вихід із системи з відкликанням refresh-токенів;

7) керування профілем: зміна пароля, вибір базової валюти портфеля (наприклад USD, USDT, UAH) та її зміна, керування набором валют для перемикання, збереження особистих налаштувань;

8) налаштування персонального ключа OpenAI (зберігається у вигляді шифротексту Fernet) та вибір моделі для AI-агента;

9) перемикання теми оформлення (світла / темна);

10) перемикання мови інтерфейсу (українська, англійська, польська);

б) керування гаманцями:

1) створення гаманця (введення назви);

2) редагування назви гаманця;

3) видалення гаманця (з урахуванням пов’язаних активів та операцій);

в) керування активами в гаманці:

1) додавання активу до гаманця та введення кількості;

2) редагування кількості активу;

3) видалення активу з гаманця;

4) перегляд переліку активів у конкретному гаманці;

г) операції та журнал:

1) додавання операції з обов’язковою фіксацією типу операції, активу та кількості (суми), дати, гаманця-джерела, коментаря або назви операції (за потреби);

2) підтримка типів операцій (поповнення, виведення, переказ між власними гаманцями, конвертація (з фіксацією курсу, умов обміну));

3) автоматичне оновлення балансів активів після внесення операції;

4) перегляд журналу операцій у вигляді списку (гаманець, назва, тип, сума, дата) з можливістю фільтрації за гаманцем або періодом (за потреби);

д) оцінка портфеля та ринкові дані:

1) перегляд портфеля (відображення гаманців і активів (плитки, список), відображення загальної вартості портфеля у базовій валюті);

2) отримання ринкових котирувань (цін) з зовнішнього API [11], [12];

3) перерахунок оцінки портфеля за актуальними котируваннями;

4) оновлення ринкових даних (за запитом користувача та/або періодично (в межах лімітів API));

е) аналітика:

1) перегляд аналітики портфеля з вибором періоду (1Д, 1Т, 1М, 1Р, увесь час);

2) зведене представлення з кільцевою діаграмою розподілу транзакцій за типами (поповнення, виведення, переказ, конвертація, інше) та поточної вартості портфеля;

3) деталізація на рівні гаманця: загальна вартість, прибуток і збиток, розбивка операцій за типами;

4) деталізація на рівні типу операцій: перелік активів, що брали участь, із сумами в нативних одиницях і в базовій валюті;

5) перегляд графіків вартості обраних активів (інтервали 1Г, 4Г, 1Д, 1Т, 1М, 1Р, увесь час);

6) додавання та видалення графіків зі списку відображення;

7) автоматичне визначення джерела цін (Binance – для активів, що торгуються на біржі; CoinGecko – для решти) та явне маркування активів, для яких використовуються спрощені дані;

ж) інтелектуальний помічник:

1) ведення кількох незалежних чатів з історією повідомлень, які зберігаються між сесіями;

2) потокова (SSE) подача відповіді з посимвольною анімацією і станом «агент друкує»;

3) автоматичне формування короткого заголовка чату на основі першого повідомлення;

4) перейменування та видалення чатів;

5) формування знеособленого знімка портфеля (псевдоніми гаманців замість реальних назв) і передача його як контексту в системне повідомлення моделі;

6) ручне оновлення «знімка» користувачем за потреби;

7) використання інструмента web_search_preview для отримання актуальних ринкових даних, якщо обрана модель це підтримує;

8) вибір доступної моделі OpenAI із каталогу сервісу та збереження вибору в профілі;

9) типізована обробка помилок з прикладного API OpenAI (недійсний ключ, вичерпана квота, відсутня модель, перевищений ліміт запитів) із поданням повідомлення кирилицею у вікні чату;

10) системний промпт, що забороняє конкретні торгові сигнали, гарантовані обіцянки прибутку та надмірно ризиковані рекомендації; У відповідях на критичні питання модель повинна дотримуватися послідовної структури: надавати необхідний контекст, аналізувати можливі сценарії та ризики, описувати типові підходи або практики реагування, а також додавати застереження про те, що користувач самостійно ухвалює рішення і несе відповідальність за його наслідки

11) модуль не виконує торгових дій та не керує коштами користувача.

Нефункціональні вимоги визначають якість роботи системи, її надійність та зручність, і включають

а) безпека:

1) паролі зберігаються у вигляді захищених хешів (bcrypt);

2) персональні чутливі поля (OpenAI-ключ, метадані входів – IP, User-Agent) зашифровані симетричним алгоритмом Fernet (AES-128-CBC + HMAC-SHA256);

3) сесії побудовано на короткоживучому access-токені та httpOnly refresh-cookie з ротацією;

4) реалізовано обмеження частоти запитів (rate limiting) для критичних маршрутів (вхід, реєстрація, виклики агента, оновлення ключа);

5) налаштовано заголовки HTTP-безпеки (Content-Security-Policy, X-Frame-Options DENY, X-Content-Type-Options nosniff, Referrer-Policy, Permissions-Policy);

6) користувач має доступ лише до власних даних, перевірка авторизації виконується на серверній стороні в кожному маршруті [14], [15];

б) надійність:

1) дані гаманців, активів, операцій, чатів і налаштувань зберігаються в реляційній СКБД PostgreSQL і не втрачаються при перезапуску;

2) еволюція схеми бази здійснюється через міграції Alembic;

3) критичні операції (наприклад, переказ між гаманцями) виконуються в рамках одної транзакції;

в) коректність:

1) перевірка введених даних на стороні клієнта (форми) і сервера (Pydantic-схеми);

2) узгоджене оновлення балансів після операцій;

3) контроль граничних випадків (від'ємні суми, неіснуючі активи/гаманці, конвертації самі в себе);

г) продуктивність:

1) типові API-відповіді – менше 300 мс для зведень портфеля та переліків транзакцій;

2) перший корисний рендер сторінки – менше 1,5 с на типовому каналі;

3) кешування ринкових цін на 5 хвилин для зменшення навантаження на зовнішні API;

д) зручність використання:

1) ключові дії (додати гаманець, додати операцію, перегляд портфеля) виконуються в межах 2–3 кроків;

2) інтерфейс відповідає вимогам WCAG 2.1 рівня AA щодо контрастності тексту й елементів керування у світлій та темній темах;

е) сумісність:

1) коректна робота в актуальних версіях браузерів Chrome, Firefox, Safari, Edge;

2) адаптивна верстка з трьома брейкпоінтами – настільний (≥ 1280 px), планшетний (768–1279 px), мобільний (≤ 767 px);

3) підтримка safe-area для пристроїв з вирізом і екранної клавіатури на iOS/Android;

ж) підтримуваність і відтворюваність:

1) окремі контейнери для серверної частини та СКБД (Docker); скрипт ініціалізації БД, що ідемпотентно завантажує каталог активів (~13 000) із Binance і CoinGecko; фонове завантаження логотипів активів без блокування старту контейнера.

Основним актором при взаємодії з вебзастосунком є користувач. Додатковими зовнішніми учасниками виступають: Binance Spot API і CoinGecko API – для отримання поточних і історичних ринкових курсів; OpenAI API – для роботи AI-агента (за наявності користувацького ключа); WebAuthn-сумісний автентифікатор пристрою (Touch ID, Face ID, Windows Hello, апаратний ключ) – для безпарольної автентифікації.

Основні варіанти використання системи користувачем охоплюють:

а) автентифікація та профіль:

1) реєстрація;

2) вхід і вихід;

3) керування профілем;

б) облік портфеля:

1) керування гаманцями;

2) керування активами в гаманці;

3) перегляд портфеля;

в) операції:

1) додавання операції (поповнення, виведення, переказ та конвертація);
2) автоматичне оновлення балансів;
3) перегляд журналу операцій;

г) аналітика:

1) перегляд аналітики і графіків;
2) оновлення ринкових даних;

д) AI-агент:

1) оновити дані для агента;
2) чат з агентом.

UML-діаграма варіантів використання наведена в графічній частині на кресленні «UML-діаграма варіантів використання».

1.4 Висновки. Постановка задачі

У розділі досліджено предметну область обліку криптовалютного портфеля та встановлено, що для користувача важливим є не лише відображення поточних залишків активів. Повноцінна система обліку має також забезпечувати збереження історії операцій, оцінювання портфеля в базовій валюті, перегляд структури активів і подання аналітичної інформації у зрозумілому вигляді. Це дає змогу користувачу контролювати зміни стану портфеля, аналізувати результати власних дій і швидше приймати рішення щодо подальшого управління активами.

Огляд наявних аналогів показав, що готові сервіси часто орієнтуються на автоматичні інтеграції з біржами, розширені торгові можливості, податкову звітність або складні інструменти професійної аналітики. Такі функції можуть бути корисними для досвідчених інвесторів, однак вони ускладнюють інтерфейс і не завжди відповідають задачі персонального контрольованого обліку. Тому в роботі доцільно зосередитися на безпечному веденні даних користувачем,

ручному внесенні операцій, автоматичному отриманні ринкових курсів і наочному відображенні стану криптовалютного портфеля.

За результатами аналізу сформовано функціональні та нефункціональні вимоги до розроблюваного вебзастосунку, а також визначено основні варіанти використання системи. Вони стали основою для подальшого проєктування архітектури, моделі даних, логіки взаємодії між модулями та програмної реалізації. Окрему увагу приділено вимогам до зручності інтерфейсу, захисту даних користувача, коректного обліку операцій і відображення актуальної ринкової інформації. Це дозволяє забезпечити цілісність майбутнього рішення та узгодити його функції з поставленою метою роботи.

На підставі сформованих вимог визначено перелік задач, які мають бути реалізовані в подальших розділах кваліфікаційної роботи. Їх виконання дасть змогу перейти від аналізу предметної області до проєктування структури системи, реалізації основних модулів і перевірки працездатності розробленого вебзастосунку.

Перелік поставлених задач:

- виконати аналіз архітектурних підходів до побудови вебзастосунків та обґрунтувати вибір архітектури розроблюваної системи (розподіл на клієнтську і серверну частини, принципи взаємодії, підходи до безпеки);
- розробити загальну структуру проєкту та декомпозицію на модулі, визначити їх призначення та взаємодію (модуль обліку гаманців, модуль операцій, модуль розрахунків, модуль ринкових даних, модуль аналітики, модуль профілю);
- спроєктувати модель даних для зберігання користувачів, гаманців, активів, журналу операцій, налаштувань профілю та ринкових котирувань;
- розробити макети (прототипи) користувацького інтерфейсу та визначити основні сценарії взаємодії з системою;

– виконати аналіз і вибір технологій реалізації (інструменти клієнтської та серверної частини, база даних, бібліотеки та підходи для візуалізації графіків, інтеграція з API ринкових даних);

– реалізувати модуль керування гаманцями та активами в гаманцях, забезпечивши коректну роботу CRUD-операцій;

– реалізувати модуль операцій (поповнення, виведення, переказ, конвертація) та автоматичне оновлення балансів активів із фіксацією історії;

– реалізувати модуль отримання ринкових даних (котирувань) із зовнішнього API, кешування, оновлення та використання цих даних у розрахунках;

– реалізувати модуль безпарольної автентифікації (WebAuthn / FIDO2): реєстрацію паскеїв, вхід через паскей, перегляд і керування переліком зареєстрованих автентифікаторів;

– реалізувати журнал входів користувача із шифруванням чутливих метаданих та визначенням приблизного географічного контексту;

– реалізувати багатомовний інтерфейс (українська, англійська, польська) і підтримку світлої й темної тем оформлення з реактивним перемиканням без перезавантаження сторінки.

– реалізувати модуль аналітики: відображення графіків вартості активів і/або загальної оцінки портфеля;

– реалізувати модуль AI-агента: ведення чатів зі збереженням історії, потокову подачу відповідей, формування знеособленого знімка портфеля як контексту, інтеграцію з відкритим API OpenAI з підтримкою вибору моделі та інструмента вебпошуку, типізовану обробку помилок зовнішнього API

– визначити підхід до тестування (функціональне, модульне, інтеграційне) та виконати тестування основних сценаріїв;

– проаналізувати результати реалізації та тестування, сформулювати висновки і перспективи подальшого розвитку системи.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1 Аналіз архітектурних підходів

Вибір архітектури має враховувати інтерактивне відображення портфельних даних, надійне збереження історії операцій, безпечну автентифікацію, інтеграцію з API цін і можливість подальшого тестування. Також важливо, щоб обрана модель не ускладнювала реалізацію основних функцій і дозволяла поступово розширювати систему новими модулями.

Порівняння виконано за критеріями складності реалізації, масштабованості, продуктивності, придатності до тестування та відповідності навчальним цілям КвР. Додатково враховано зручність підтримки коду, розмежування відповідальності між частинами системи та доцільність використання підходу для вебзастосунку персонального обліку.

Монолітна архітектура із серверним рендерингом є простою для старту [16], але гірше підходить для динамічного інтерфейсу портфельної аналітики, де часто оновлюються графіки, баланси та локальні стани.

Мікросервіси корисні для великих платформ [16], однак для поточного масштабу створюють надлишкову інфраструктурну складність: міжсервісну взаємодію, централізоване логування, синхронізацію контрактів і складніше тестування.

Клієнт-серверна архітектура SPA [20] + REST API [17], [19] забезпечує потрібний баланс: клієнт відповідає за інтерактивний UI, сервер – за бізнес-логіку, перевірки доступу, транзакційні операції та роботу з БД. Такий поділ спрощує незалежне тестування клієнтської та серверної частин, а також робить систему більш зрозумілою для подальшої підтримки.

За результатами порівняння SPA + REST API обрано як базову архітектурну модель, оскільки вона забезпечує інтерактивність, модульний розвиток, прозоре тестування та не потребує надлишкової інфраструктури.

Порівняльну характеристику розглянутих підходів за ключовими критеріями продемонстровано на таблиці 2.1.

Таблиця 2.1. Порівняння архітектурних підходів до побудови вебзастосунку

Архітектурний підхід	Переваги	Недоліки	Доцільність для розроблюваного вебзастосунку
Монолітний підхід (SSR)	Проста початкова реалізація; єдина точка розгортання.	Обмежена інтерактивність інтерфейсу; складніше масштабувати окремі підсистеми.	Частково доцільний, але не оптимальний для задач портфельної аналітики.
SPA + REST API	Чіткий поділ клієнта і сервера; зручна модульність; підтримка інтерактивного UI.	Потребує окремого проєктування API та керування станом на клієнті.	Оптимальний вибір для даної кваліфікаційної роботи.
Мікросервісний підхід	Горизонтальна масштабованість; незалежне розгортання сервісів.	Висока складність інфраструктури, моніторингу та міжсервісної взаємодії.	Недоцільний у межах навчального проєкту поточного масштабу.

2.2 Обґрунтування архітектури вебзастосунку

Після вибору SPA + REST API архітектуру уточнено через розподіл відповідальності між клієнтом, сервером, базою даних та зовнішніми інтеграціями.

Клієнтська частина є SPA [20], що забезпечує роботу з автентифікацією, гаманцями, операціями, аналітикою та AI-помічником. Серверна частина надає REST API [17], [19], виконує перевірки доступу, обробляє транзакції та працює з БД.

Обмін даними побудовано за принципом: клієнт ініціює запит, сервер перевіряє параметри й права доступу [30], після чого повертає узгоджений результат. Усі зміни стану фіксуються в реляційній моделі даних.

Центральним сховищем є PostgreSQL [23], де зберігаються користувачі, гаманці, активи, транзакції, ринкові знімки й службові дані автентифікації. Це забезпечує цілісну історію портфеля та коректні агреговані розрахунки.

Інтеграційний шар відповідає за CoinGecko [27], Binance [28] та OpenAI [29]. Зовнішні API використовуються лише для ринкових і довідкових даних, а критичні облікові операції залишаються всередині backend-контуру.

З погляду безпеки архітектура розділяє публічний і захищений контури: браузер працює тільки з API, а автентифікація, авторизація й журналювання централізовані на backend-рівні [30].

Перевагою такого підходу є чіткий поділ обов’язків: UI – на клієнті, доменна логіка й безпека – на сервері, довготривале зберігання – у СКБД.

Загальну схему взаємодії клієнтської частини, серверних модулів, бази даних та зовнішніх сервісів наведено на рисунку 2.1.

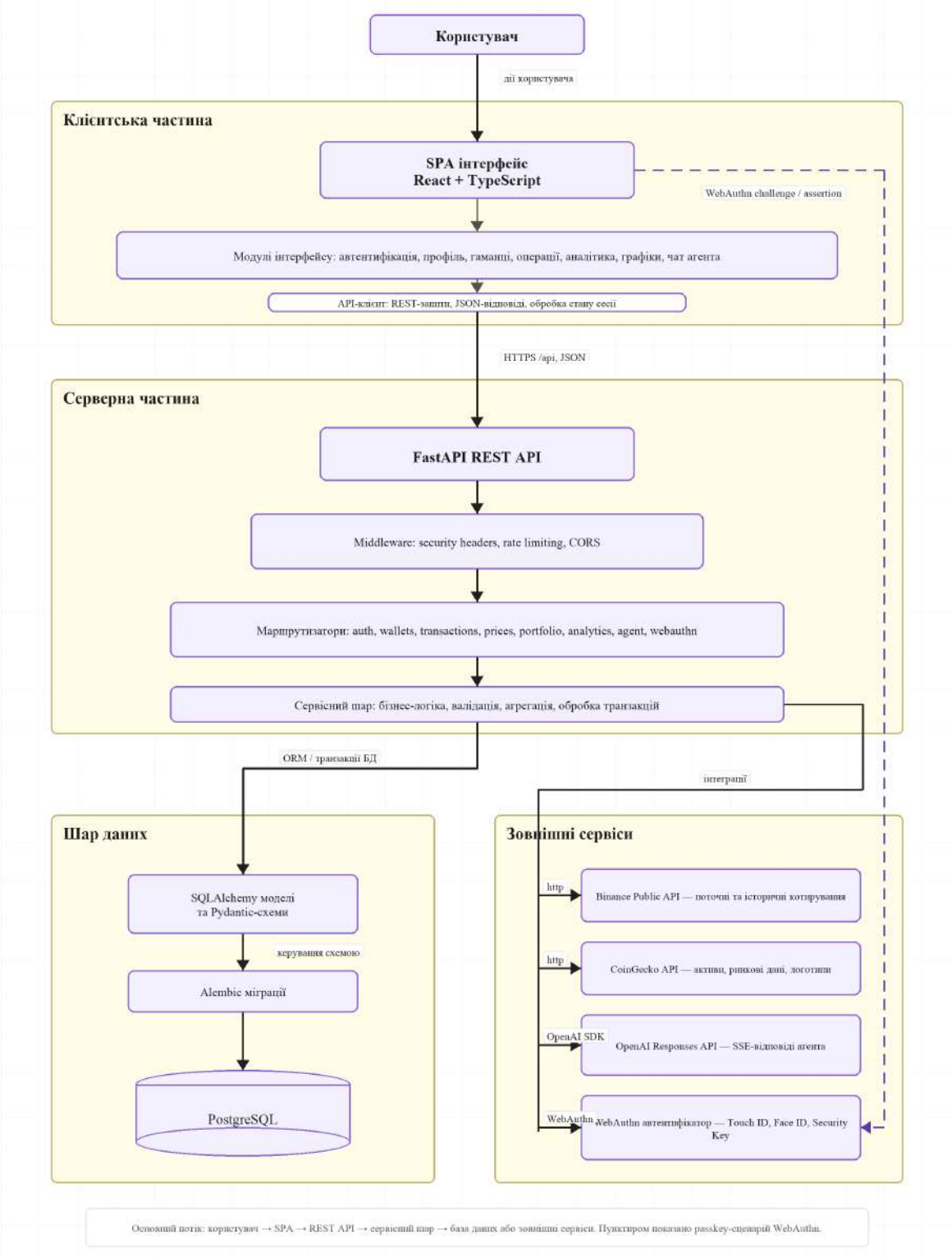


Рисунок 2.1 – Загальна архітектура вебзастосунку для обліку криптовалютного портфеля

2.3 Декомпозиція системи на модулі

Декомпозиція потрібна для керування складністю системи [18], оскільки застосунок поєднує автентифікацію, фінансовий облік, аналітику та зовнішні інтеграції.

На клієнтському рівні модулі відповідають користувацьким сценаріям: вхід, профіль, портфель, транзакції, аналітика та AI-помічник. Це локалізує зміни інтерфейсу в межах відповідних функціональних блоків.

Серверна частина поділена за доменним принципом на auth, wallets/assets, transactions, prices/currencies, portfolio, analytics, agent і webauthn. Кожен модуль має окрему відповідальність і власний API-контур.

Модулі взаємодіють через сервісний шар, тому доменні операції не дублюються в маршрутизаторах. Наприклад, транзакції використовують активи, ціни та баланси для коректного перерахунку портфеля.

Окремо доцільно підкреслити межі зовнішніх залежностей: інтеграції з Binance [28] і CoinGecko [27] зосереджені в модулі ринкових даних, а інтеграція з OpenAI [29] – у модулі агента. Завдяки цьому при зміні зовнішнього API або політик доступу коригування вносяться в обмежену частину системи, що зменшує ризик регресій у критичних облікових функціях.

Доступ до даних централізовано через PostgreSQL [23], що забезпечує узгоджене збереження користувацьких сутностей, журналу операцій і проміжних ринкових знімків. У результаті модульна декомпозиція підтримує не лише зручність розробки, а й архітектурну передбачуваність системи під час тестування, супроводу та подальшого розширення функціональності.

Загальну структурну декомпозицію клієнтських і серверних модулів, їх взаємодію із шаром даних та зовнішніми сервісами наведено в графічній частині на кресленні «Діаграма зв’язків модулів».

Для переходу від функціональної декомпозиції до проєктування програмної структури доцільно деталізувати систему на рівні класів серверної частини. Така

деталізація фіксує ключові сутності предметної області (користувач, гаманець, актив, транзакція, сесія, аналітичні та агентні об’єкти), їх атрибути та типи зв’язків, які безпосередньо реалізуються в ORM-моделях і сервісному шарі.

Статичну структуру доменних сутностей і зв’язки між ними наведено в графічній частині на кресленні «UML-діаграма класів серверної частини». Це узгоджує модульну архітектуру з моделлю даних і подальшою програмною реалізацією.

Узагальнену характеристику модулів вебзастосунку, їх призначення, ключові функції та взаємодію наведено у таблиці 2.2.

Таблиця 2.2. Декомпозиція вебзастосунку на функціональні модулі

Модуль	Рівень	Призначення	Основні функції	Взаємодія
auth	Сервер	Керування обліковими записами і сесіями	Реєстрація, вхід, оновлення токенів, профіль	webauthn, PostgreSQL
wallets/assets	Сервер	Керування гаманцями, активами і балансами	CRUD гаманців, додавання/оновлення балансів, довідник активів	transactions, prices, PostgreSQL
transactions	Сервер	Облік операцій портфеля	Додавання операцій, журнал, контекст операцій	wallets/assets, prices, portfolio
prices/currencies	Сервер	Отримання і нормалізація ринкових даних	Поточні ціни, історичні дані, FX-перерахунок	Binance API, CoinGecko API, PostgreSQL

Продовження таблиці 2.2. Декомпозиція вебзастосунку на функціональні модулі

Модуль	Рівень	Призначення	Основні функції	Взаємодія
portfolio	Сервер	Формування зведення портфеля	Агрегація вартості, P/L, знімок позицій	wallets/assets, prices, transactions
analytics(charts)	Сервер	Керування набором графіків користувача	Створення, оновлення, видалення графіків	portfolio, prices, PostgreSQL
agent	Сервер	Інтелектуальний помічник	Керування чатами, контекст, SSE-відповіді	portfolio, OpenAI API, PostgreSQL
SPA UI	Клієнт	Інтерфейс взаємодії користувача	Форми, списки, перегляд портфеля, аналітика, чат	усі API-модулі сервера

2.4 Проєктування моделі даних

Проєктування моделі даних виконується як базовий етап, що визначає коректність обліку портфеля, узгодженість аналітичних розрахунків і надійність збереження історії змін. Для вебзастосунку обліку криптовалютного портфеля модель даних повинна одночасно підтримувати користувацькі сутності, операційний журнал, ринкові знімки цін і службові механізми автентифікації в реляційній СКБД [23].

У моделі використано реляційний підхід із чітким виділенням доменних груп: облікові дані користувача та сесій, довідник активів і гаманців, балансні стани, транзакції, аналітичні налаштування та контекст інтелектуального модуля.

Така структура дозволяє відокремити первинні бізнес-сутності від допоміжних технічних таблиць і знизити ризик дублювання даних.

Ключовим принципом проєктування є нормалізація зв'язків між сутностями та фіксація цілісності через первинні та зовнішні ключі [23]. Зокрема, усі користувацькі дані прив'язуються до конкретного облікового запису, а операції та баланси пов'язуються з гаманцями й активами через однозначні ідентифікатори. Це забезпечує простежуваність походження кожного запису та коректність агрегацій у портфельних зведеннях.

Структурований опис основних таблиць, їх призначення, ключових полів і зв'язків наведено у таблиці 2.3.

Таблиця 2.3. Опис основних таблиць моделі даних

Таблиця	Призначення	Ключові поля	Основні зв'язки
users	Облікові записи користувачів і налаштування профілю	id, email, password_hash, base_currency	1:N з wallets, transactions, refresh_tokens, analytics_charts, chats
assets	Довідник крипто- і фіатних активів	id, symbol, name, type, binance_symbol, coingecko_id	1:N з wallet_balances, transactions, price_snapshots
wallets	Гаманці користувача	id, user_id, name	1:N з wallet_balances; 1:N з transactions
wallet_balances	Поточні залишки активів у гаманцях	id, wallet_id, asset_id, amount, cost_basis_usdt	FK на wallets і assets; унікальність пари wallet_id+asset_id
transactions	Журнал операцій користувача	id, user_id, wallet_id, asset_id, type, amount, occurred_at	FK на users, wallets, assets; зв'язок із контрагентськими полями
price_snapshots	Знімки ринкових цін у часі	id, asset_id, quote, price, source, fetched_at	FK на assets; використовується модулями portfolio та analytics

Продовження таблиці 2.3. Опис основних таблиць моделі даних

Таблиця	Призначення	Ключові поля	Основні зв’язки
refresh_tokens	Сесійні refresh-токени	id, user_id, token_hash, expires_at	FK на users
webauthn_credentials	Зареєстровані passkey-облікові дані	id, user_id, credential_id, public_key, sign_count	FK на users
agent_contexts, chats, chat_messages	Контекст і історія взаємодії з агентом	user_id, chat_id, role, content	FK на users і chats
login_events	Історія входів користувача	id, user_id, method, created_at	FK на users

Для числових значень, що використовуються в обліку активів і розрахунках вартості, застосовано типи з високою точністю у PostgreSQL [23]. Такий підхід є необхідним для уникнення накопичення похибок під час багаторазових конвертацій, перерахунків і формування історичної аналітики. Це особливо важливо для криптовалютних активів, де значення можуть мати велику кількість знаків після коми, а навіть незначні неточності здатні впливати на підсумкові показники портфеля. Окремо передбачено зберігання знімків ринкових цін, що дозволяє відтворювати стан портфеля у часовому розрізі.

У межах цієї моделі окрему увагу приділено зв’язкам між транзакційними таблицями, балансними станами та довідником активів, оскільки саме вони забезпечують коректний перерахунок залишків, узгодженість історії операцій і достовірність портфельних показників у динаміці. Така структура дає змогу відокремити довідкові дані від операційних записів і спростити подальше розширення моделі новими сутностями.

Загальну структуру сутностей, їх зв’язки та напрямки залежностей у реляційній моделі даних зображено на рисунку 2.2.

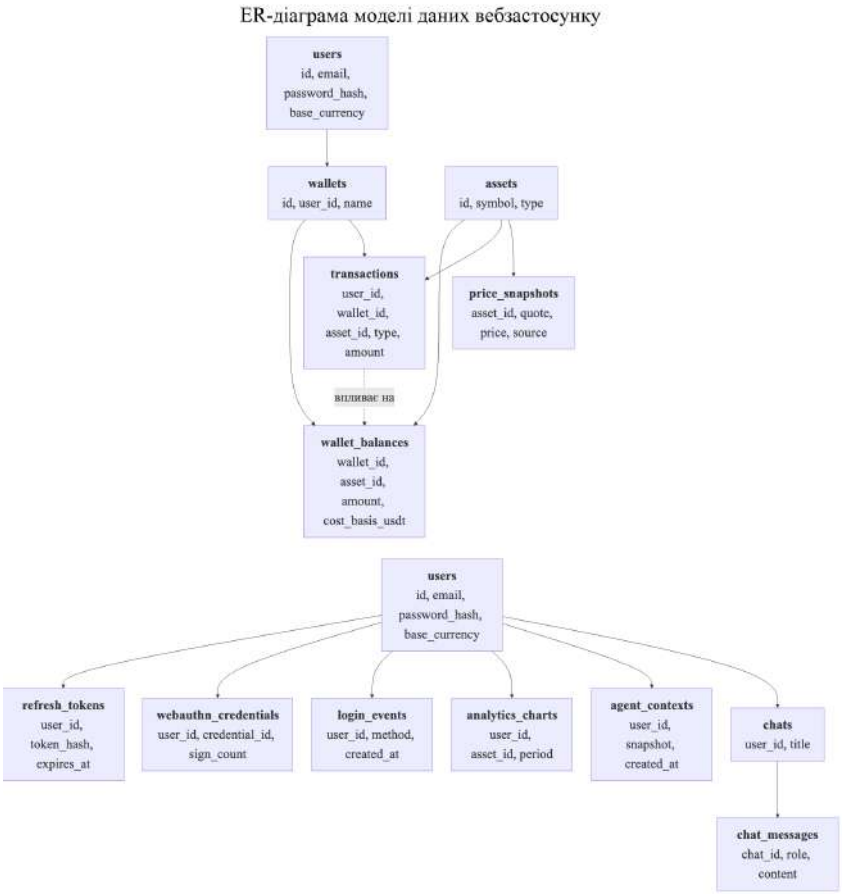


Рисунок 2.2 – ER-діаграма моделі даних вебзастосунку для обліку криптовалютного портфеля

Після визначення структури сутностей доцільно деталізувати ключові обмеження цілісності, що наведено у таблиці 2.4.

Таблиця 2.4. Ключові поля та обмеження цілісності моделі даних

Сутність / поле	Тип даних	Обмеження	Призначення в моделі
users.email	VARCHAR	UNIQUE, NOT NULL	Унікальна ідентифікація користувача
wallet_balances (wallet_id, asset_id)	UUID + UUID	UNIQUE	Недопущення дублювання одного активу в межах гаманця
transactions.amount	NUMERIC	CHECK amount > 0	Валідація додатних сум операцій
transactions.type	ENUM	NOT NULL	Фіксація допустимого типу операції

Продовження таблиці 2.4. Ключові поля та обмеження цілісності моделі даних

Сутність / поле	Тип даних	Обмеження	Призначення в моделі
transactions.user_id	UUID	FK -> users.id	Ізоляція операцій у межах власника
wallets.user_id	UUID	FK -> users.id	Прив’язка гаманця до користувача
price_snapshots. Asset_id	UUID	FK -> assets.id	Прив’язка цінового знімка до активу
refresh_tokens. Token_hash	CHAR/ VARCHAR	NOT NULL	Зберігання хешу refresh-токена замість відкритого значення
webauthn_ credentials.user_id	UUID	FK -> users.id	Зв’язок passkey-даних з обліковим записом

2.5 Аналіз і вибір технологій реалізації

Аналіз і вибір технологій реалізації виконано з урахуванням функціональних і нефункціональних вимог до вебзастосунку, а також обмежень навчального проєкту щодо підтримуваності, відтворюваності середовища та прогнозованої складності супроводу. У межах цього етапу оцінювалися не лише окремі інструменти, а й їх сумісність у складі єдиного технологічного контуру.

Першим критерієм відбору визначено керованість складності реалізації: технології повинні дозволяти послідовно розвивати модулі портфеля, операцій та аналітики без надмірного інфраструктурного навантаження. Другим критерієм є якість валідації і типізації даних, оскільки помилки на межі «клієнт–сервер–БД» у фінансово-обліковому контексті прямо впливають на достовірність розрахунків. Третім критерієм є готовність стеку до інтеграції із зовнішніми API та механізмами безпеки.

Для серверної частини обрано Python і FastAPI [22]. Порівняно з більш «важкими» фреймворками, це рішення забезпечує швидший цикл розробки та

прозору декомпозицію API-маршрутів і сервісної логіки. Важливою перевагою є асинхронна модель обробки запитів, яка доречна для задач, де частина операцій залежить від мережевих викликів до зовнішніх сервісів. При цьому обраний підхід не вимагає ускладнення архітектури мікросервісами на поточному масштабі проєкту.

З боку зберігання даних застосовано PostgreSQL [23] у поєднанні з SQLAlchemy [25] та Alembic. PostgreSQL обрано через надійну транзакційну модель, зрілий механізм обмежень цілісності та добру підтримку високоточної числової арифметики, необхідної для портфельних обчислень. SQLAlchemy забезпечує узгоджене представлення сутностей на рівні коду, а Alembic – контрольовану еволюцію схеми бази даних. Це мінімізує ризики розсинхронізації між прикладною логікою та структурою зберігання.

Клієнтська частина реалізується на React [21] і TypeScript. Компонентна модель React спрощує декомпозицію інтерфейсу за сценаріями користувача, а TypeScript забезпечує типобезпечну взаємодію з API-контрактами. У порівнянні з підходом на «чистому» JavaScript це зменшує кількість помилок інтеграції, особливо у формах введення транзакцій і відображенні агрегованих метрик. Для управління серверним станом доцільно застосовувати інструменти кешування та інвалідації запитів, тоді як локальний стан слід обмежувати інтерфейсними задачами.

Окремо оцінено вибір технологій безпеки і доступу. Використання WebAuthn/FIDO2 (passkeys) [37] доповнює парольну модель автентифікації та підвищує стійкість механізму входу за рахунок криптографічно підтверджених сценаріїв. На серверному рівні це поєднується з централізованою обробкою сесій [31], валідацією запитів та обмеженням частоти звернень до критичних ендпоінтів [30].

Для ринкового контуру застосовано інтеграцію з Binance [28] і CoinGecko [27]. Вибір двох джерел замість одного обумовлений практичним покриттям активів і наявністю історичних даних у різних сегментах ринку. Така комбінація

знижує залежність від одного постачальника котирувань і дає змогу підтримувати коректну оцінку портфеля для ширшого набору активів.

Інтелектуальний модуль реалізується через OpenAI API [29] з чітким функціональним обмеженням: агент надає пояснювальні відповіді та аналітичний контекст, але не виконує транзакційних дій і не керує коштами користувача. Така межа відповідальності є принциповою для збереження контрольованості системи та узгодженості з цілями кваліфікаційної роботи.

Критерій відтворюваності середовища реалізовано через контейнеризацію Docker [26]. Єдиний опис оточення для серверної частини і СКБД спрощує запуск, тестування та повторюваність результатів незалежно від локальної конфігурації робочого місця. Це зменшує ризик середовищно-залежних помилок під час перевірки функціональності.

Узагальнене порівняння обраних технологій, їх ролей у проєкті, альтернатив і причин остаточного вибору наведено у таблиці 2.5.

Таблиця 2.5. Обґрунтування вибору технологій реалізації вебзастосунку

Технологія / інструмент	Роль у проєкті	Розглянуті альтернативи	Критерії вибору	Причина остаточного вибору
Python + FastAPI	Серверна частина, REST API, сервісна логіка	Django REST Framework, Flask	Швидкість розробки, типізація, асинхронність, структурованість API	Оптимальний баланс між продуктивністю розробки і чіткою модульною організацією серверної логіки
PostgreSQL	Реляційне зберігання даних портфеля	MySQL, SQLite	Транзакційність, цілісність, зрілість екосистеми	Надійне зберігання облікових і транзакційних даних з підтримкою складних зв’язків

Продовження таблиці 2.5. Обґрунтування вибору технологій реалізації вебзастосунку

Технологія / інструмент	Роль у проєкті	Розглянуті альтернативи	Критерії вибору	Причина остаточного вибору
SQLAlchemy + Alembic	ORM-рівень і міграції схеми БД	Django ORM, raw SQL	Керованість моделі даних, контроль змін схеми, підтримуваність	Забезпечують узгодженість між кодом і БД та контрольовану еволюцію структури даних
React + TypeScript	Клієнтський інтерфейс і типобезпечна взаємодія з API	Vue, Angular, React (JS без TS)	Компонентність, типобезпека, масштабованість UI	Знижують кількість помилок інтеграції з API і спрощують розвиток інтерфейсних модулів
TanStack Query + Zustand	Керування серверним і локальним станом	Redux Toolkit, SWR, Context API	Кешування запитів, інвалідація, простота підтримки	Раціональний поділ відповідальності між серверним кешем і локальним UI-станом
Binance API + CoinGecko API	Джерела ринкових і історичних даних	Єдине джерело котирувань	Повнота покриття активів, доступність історії, стабільність API	Поєднання двох джерел забезпечує ширше покриття активів і стійкість ринкового модуля

Продовження таблиці 2.5. Обґрунтування вибору технологій реалізації вебзастосунку

Технологія / інструмент	Роль у проєкті	Розглянуті альтернативи	Критерії вибору	Причина остаточного вибору
WebAuthn/FIDO2 + OpenAI API + Docker	Passkey-автентифікація, AI-модуль, відтворюване розгортання	Лише парольна автентифікація; відмова від AI; запуск без контейнеризації	Безпека, функціональність, відтворюваність середовища	Підвищують рівень безпеки входу, розширюють аналітичні можливості та уніфікують запуск проєкту

2.6 Проєктування інтерфейсу користувача

Проєктування інтерфейсу користувача виконано на основі сценарного підходу, у межах якого кожний екран пов’язано з конкретною задачею користувача та відповідним модулем системи. Основною метою є не лише візуальна цілісність, а передусім передбачуваність взаємодії: користувач має швидко знаходити потрібну дію, розуміти поточний стан даних і отримувати коректний зворотний зв’язок на кожному кроці.

Загальна навігаційна структура охоплює контур автентифікації, контур операційного обліку, контур аналітики та допоміжні розділи профілю і правової інформації. Такий поділ узгоджений із функціональною декомпозицією серверної частини: інтерфейсні розділи відображають доменні модулі API, що знижує ризик семантичних розривів між фронтендом і бекендом.

Екран входу підтримує email/пароль і passkey, містить клієнтську валідацію, обробку помилок і переходи до реєстрації. Його реалізацію показано на рисунку 2.3.

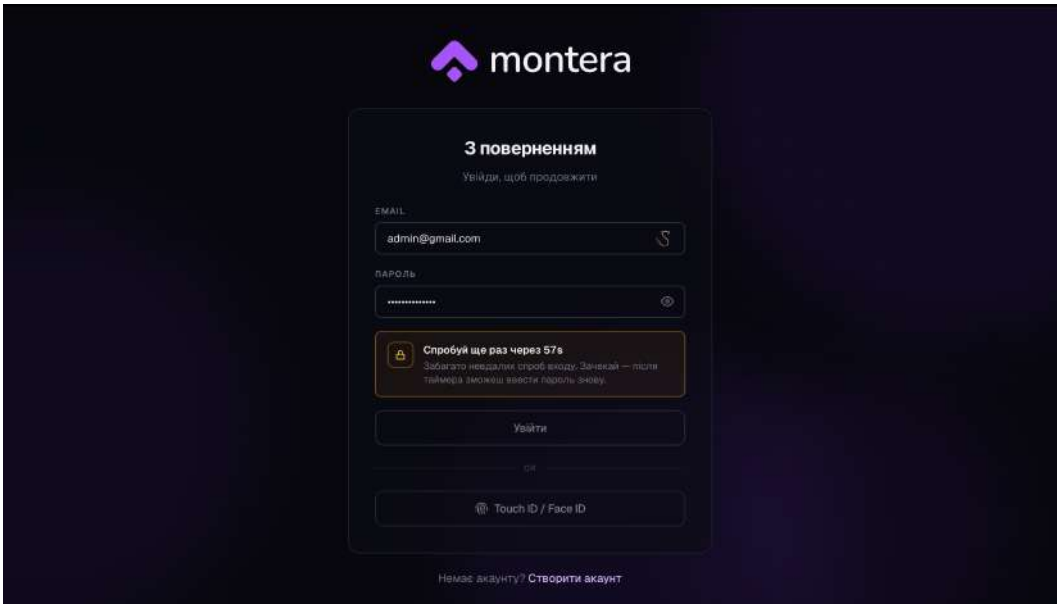


Рисунок 2.3 – Екран входу користувача

Екран реєстрації перевіряє email, складність і підтвердження пароля, а також містить генератор стійкого пароля. Приклад наведено на рисунку 2.4.

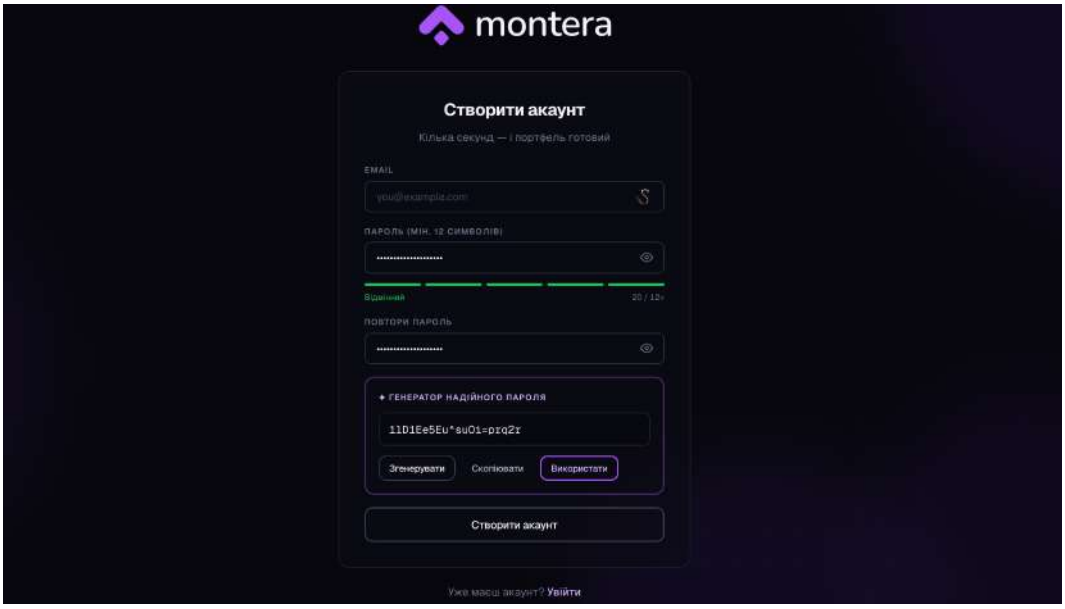


Рисунок 2.4 – Екран реєстрації користувача.

Головний екран є центром щоденного обліку: показує вартість портфеля, гаманці, швидке створення нового гаманця та останні операції. Приклад наведено на рисунку 2.5.

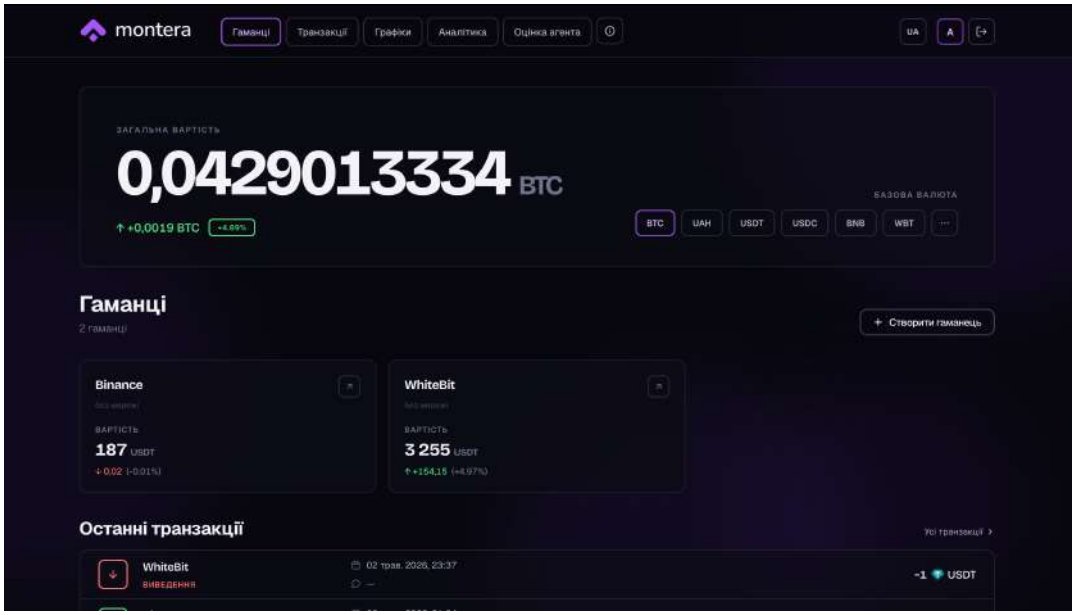


Рисунок 2.5 – Головний екран портфеля

Для роботи з окремим гаманцем використовується спливаюча картка з активами, редагуванням балансу, додаванням активів і переходом до транзакції (рисунок 2.6).

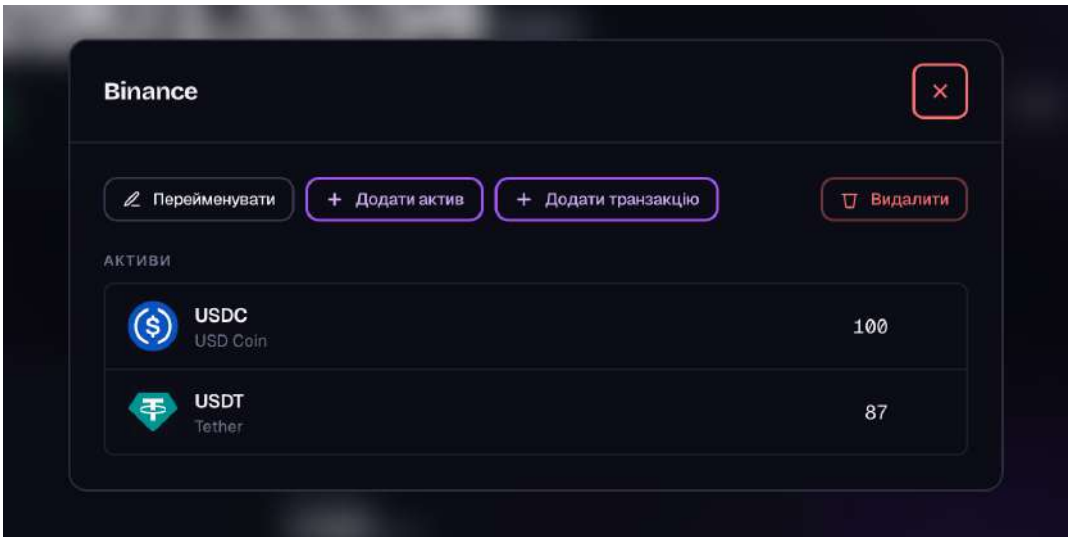


Рисунок 2.6 – Картка гаманця з операціями над активами

Екран транзакцій призначений для контролю історії змін: підтримує фільтрацію за гаманцем, типом операції, активом і періодом. Кожен запис містить

тип операції, гаманець, дату, суму й актив, що спрощує аудит змін балансу (рисунок 2.7).

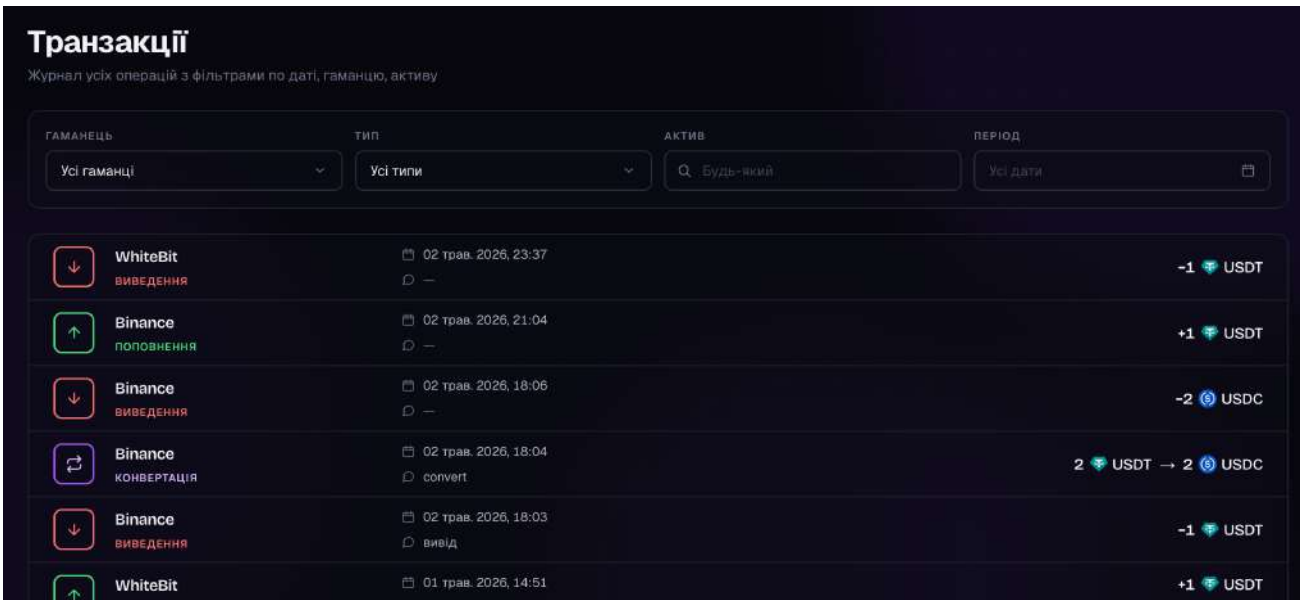


Рисунок 2.7 – Екран журналу транзакцій

Картка транзакції показує тип, суму, активи, пов’язані гаманці, час і службові атрибути запису. Вона потрібна для перевірки коректності окремої події (рисунок 2.8).

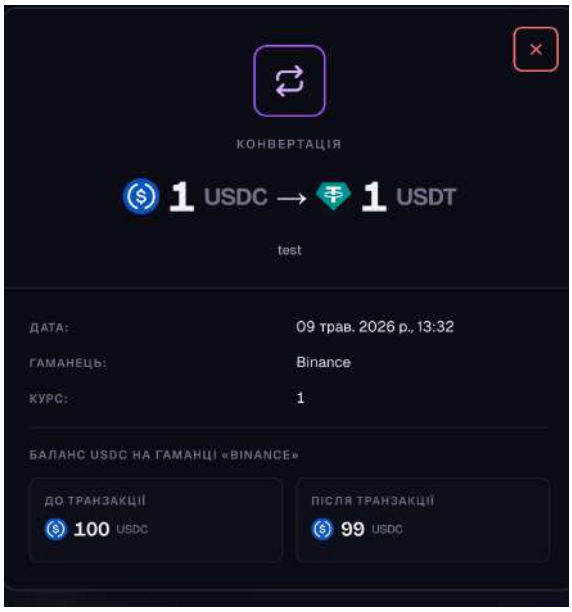


Рисунок 2.8 – Картка транзакції з детальною інформацією

Екран графіків дозволяє додавати й вилучати графіки активів та переглядати динаміку за підтримуваними інтервалами без перевантаження інтерфейсу (рисунок 2.9).

Рисунок 2.9 – Екран графіків активів.

Екран аналітики (рисунок 2.10) містить зведений блок портфеля, розподіл транзакцій за типами, показник P/L, перемикач періоду та плитки по гаманцях. На відміну від журналу транзакцій, він орієнтований на оцінку загальної ефективності портфеля.

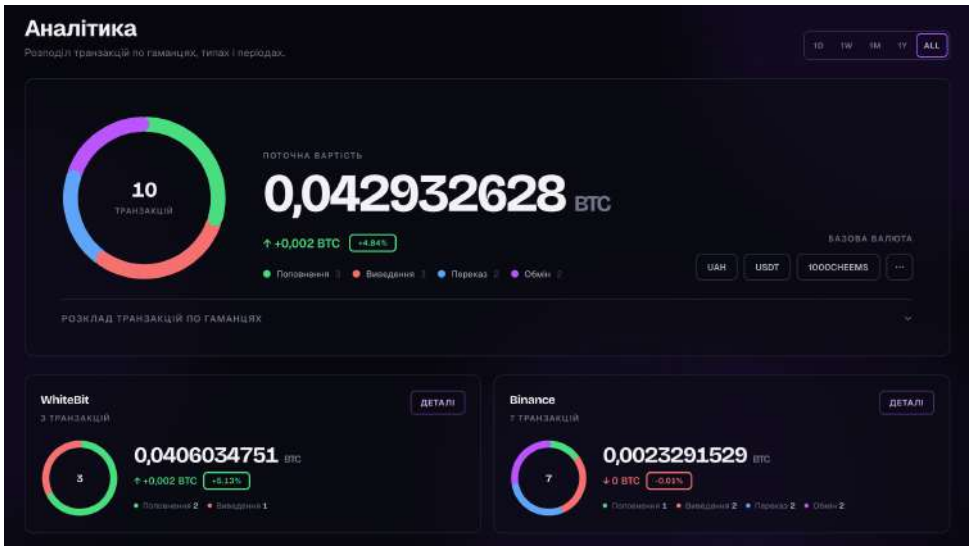


Рисунок 2.10 – Екран аналітики портфеля

Екран агента реалізує пояснювальний сценарій: підтримує збережені чати, оновлення знеособленого знімка портфеля, потокову відповідь і markdown-форматування з посиланнями [29]. Такий підхід дає змогу зберігати контекст діалогу та надавати пояснення на основі актуального стану портфеля без передавання персональних даних.

Приклад на рисунку 2.11 демонструє відповідь щодо ризиків інвестування всіх коштів в альткоїни: агент не дає категоричного сигналу, а структурує контекст, сценарії, ризики та дисклеймер.

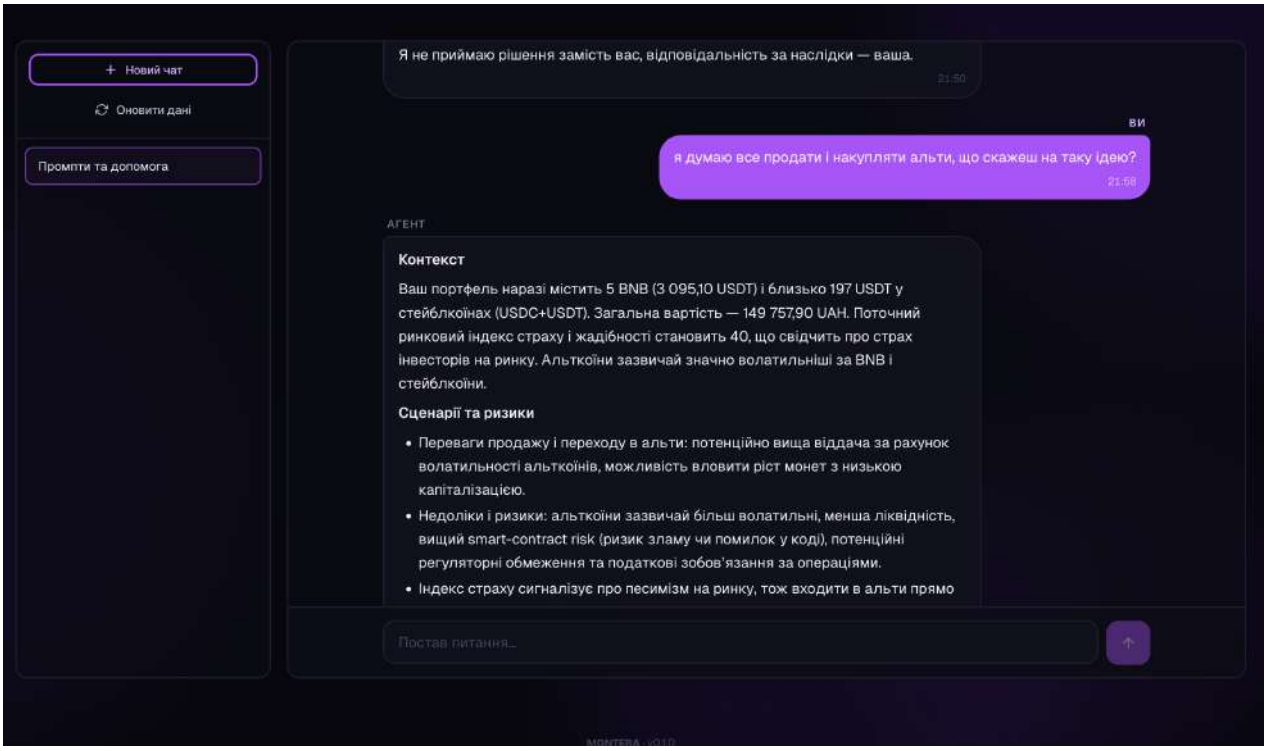


Рисунок 2.11 – Екран взаємодії з інтелектуальним помічником

Під час проєктування враховано мобільну адаптивність: головний екран зберігає ієрархію, використовує компактну навігацію та нижню панель швидкого доступу. Це дає змогу користувачу переглядати основні показники портфеля, переходити між розділами та виконувати базові дії без втрати зручності на малих екранах. Скріншот мобільної версії наведено на рисунку 2.12.

Рисунок 2.12 – Головний екран у мобільній версії.

Окремі службові екрани профілю та правового розділу забезпечують завершеність інтерфейсу з точки зору експлуатації: у профілі зосереджено керування паролем, passkey, OpenAI-ключем і журналом входів, а в Legal – доступ до політик і довідкових матеріалів. Узагальнену відповідність усіх екранних форм їхнім задачам, ключовим елементам і пов’язаним модулям наведено у таблиці 2.6.

Таблиця 2.6. Відповідність екранів вебзастосунку користувацьким сценаріям

Екран / розділ	Призначення	Ключові елементи інтерфейсу	Основні дії користувача	Пов’язані модулі
Вхід	Автентифікація користувача	Поля email/пароль, кнопка входу, вхід через passkey, повідомлення помилок	Вхід у систему, passkey-вхід, перехід до реєстрації	auth, webauthn

Продовження таблиці 2.6. Відповідність екранів вебзастосунку користувацьким сценаріям

Екран / розділ	Призначення	Ключові елементи інтерфейсу	Основні дії користувача	Пов’язані модулі
Реєстрація	Створення облікового запису	Поля email/пароль/підтвердження, індикатор сили пароля, генератор паролів	Створення акаунта, перевірка пароля, автологін після реєстрації	auth
Головний екран (Dashboard)	Оперативний перегляд стану портфеля	Зведення портфеля, плитки гаманців, швидке створення гаманця, блок останніх операцій	Перегляд вартості, відкриття гаманця, додавання гаманця, перехід до журналу	portfolio, wallets, transactions
Транзакції	Деталізований журнал операцій	Фільтри (гаманець, тип, актив, період), список операцій, перегляд деталей	Фільтрація, посторінковий перегляд, аналіз окремої операції	transactions, assets, wallets
Графіки	Керування та перегляд цінових графіків	Список графіків, форма додавання, картки графіків	Додавання/видалення графіків, перегляд динаміки активів	analytics(charts), prices
Аналітика	Агрегована оцінка портфеля за період	Перемикач періодів, P/L-показники, структура транзакцій, деталізація гаманців	Порівняння періодів, оцінка результативності, аналіз структури	portfolio_analytics, prices
Агент	Пояснювальна аналітика в чаті	Список чатів, область діалогу, кнопка оновлення контексту, потокова відповідь	Постановка запитань, аналіз ризиків, робота з кількома чатами	agent, OpenAI API

Продовження таблиці 2.6. Відповідність екранів вебзастосунку користувацьким сценаріям

Екран / розділ	Призначення	Ключові елементи інтерфейсу	Основні дії користувача	Пов’язані модулі
Профіль	Керування безпекою і налаштуваннями	Зміна пароля, OpenAI key/model, passkeys, історія входів	Оновлення налаштувань, керування passkey, перегляд активності	auth, webauthn, agent
Legal	Довідково-правова інформація	Список тем, контент розділів, навігація по статтях	Перегляд правил і політик, перехід між темами	legal UI

2.7 Проєктування безпеки та сесій

Проєктування безпеки та керування сесіями для вебзастосунку виконується як окремий архітектурний контур, оскільки система працює з персональними даними користувача, історією його фінансових операцій та інтеграціями із зовнішніми сервісами. У межах цієї роботи безпека розглядається як сукупність взаємопов’язаних механізмів: захист облікових даних, контроль автентифікації, безпечне керування сесією, обмеження зловживань API та захист чутливих полів у сховищі [30].

Базовий рівень захисту облікових даних реалізовано через стійке хешування паролів алгоритмом Argon2 [33]. На відміну від простих однопрохідних хеш-функцій, цей підхід орієнтований на ускладнення офлайн-перебору у випадку компрометації сховища [32]. У процесі входу застосовується перевірка хешу без збереження або повернення пароля у відкритому вигляді на будь-якому етапі серверної обробки.

Сесійний контур побудовано за моделлю короткоживучого access-токена і довшого refresh-токена [35]. Refresh-токен передається в httpOnly-cookie із політикою SameSite=Strict [31] і використовується для контрольованого оновлення сесії. На сервері зберігається не сире значення токена, а його SHA-256-хеш [34], що зменшує наслідки можливого витоку даних із БД. Додатково реалізовано ротацію refresh-токенів, яка унеможливлює довготривале використання застарілих сесійних ідентифікаторів.

Для посилення автентифікації інтегровано WebAuthn/FIDO2 (passkeys) [37]. Цей механізм дозволяє виконувати вхід із криптографічною перевіркою на боці автентифікатора пристрою і зменшує залежність від парольного сценарію. Сервер перевіряє challenge-відповідь за параметрами очікуваного origin та RP-ID, а метадані реєстрації credential зберігаються в межах користувацького контуру.

Захист API від зловживань реалізовано через обмеження частоти запитів для критичних маршрутів (автентифікація, WebAuthn-потоки, операції агента, оновлення ключа інтеграції) [30]. Такий підхід зменшує ризики brute-force і автоматизованих атак на контрольні точки входу. На транспортному рівні додатково застосовуються захисні HTTP-заголовки, зокрема Content-Security-Policy, X-Frame-Options, X-Content-Type-Options, Referrer-Policy і Permissions-Policy [36].

Окрему увагу приділено захисту чутливих даних у стані спокою: поля з підвищеним ризиком розкриття (наприклад, користувацький OpenAI-ключ і розширені метадані історії входів) шифруються симетричним механізмом Fernet [24]. Така модель не замінює контроль доступу на рівні застосунку, але суттєво знижує ризик прямого читання цих значень із дампа бази даних.

Окремий аспект безпеки стосується модуля інтелектуального помічника. У зовнішній запит через OpenAI API [29] передається лише знеособлений знімок портфеля: агреговані фінансові показники, позиції, останні транзакції та псевдоніми гаманців на кшталт «Wallet 1», «Wallet 2». При цьому не передаються ідентифікаційні атрибути користувача (email, внутрішні UUID, IP-дані), вільні

текстові мітки, що можуть містити персональні відомості, а також секрети інтеграцій (зокрема API-ключі). Такий підхід реалізує принцип мінімізації даних і обмежує ризик розкриття персональної інформації при використанні зовнішнього AI-сервісу.

Сукупність реалізованих механізмів формує багаторівневий захист, у якому кожний елемент покриває окремий клас загроз: від компрометації облікових даних до зловживань API та викрадення сесійних атрибутів. Узагальнену характеристику цих механізмів, їх технічної реалізації та очікуваного безпекового ефекту наведено у таблиці 2.7.

Таблиця 2.7. Основні механізми безпеки та керування сесіями у вебзастосунку

Компонент безпеки	Реалізація в системі	Яку загрозу знижує	Результат для системи
Хешування паролів	Argon2 (PasswordHasher) на сервері	Компрометація паролів при витоку БД, офлайн-перебір	Паролі не зберігаються у відкритому вигляді; підвищена стійкість до brute-force
Сесійна модель	Access JWT + refresh-cookie з ротацією	Тривале використання викраденого токена, фіксація сесії	Кероване продовження сесії, можливість відкликання активних токенів
Зберігання refresh-токенів	У БД зберігається SHA-256-хеш token_hash	Пряме використання токенів із дампа БД	Зниження ризику компрометації сесій при витоку сховища

Продовження таблиці 2.7. Основні механізми безпеки та керування сесіями у вебзастосунку

Компонент безпеки	Реалізація в системі	Яку загрозу знижує	Результат для системи
Cookie-політика	httpOnly, SameSite=Strict, опційно Secure	XSS-читання cookie, CSRF-переадресація cookie	Захист cookie від JS-доступу і міжсайтового автоматичного надсилання
Passkey-автентифікація	WebAuthn/FIDO2: challenge-response, перевірка RP-ID/origin	Фішинг паролів, слабкі парольні практики	Посилена автентифікація і підтримка безпарольного входу
Rate limiting	Ліміти на auth/webauthn/agent/ключові ендпоінти	Brute-force, масовані автоматизовані запити	Зменшення навантаження і ризику зловживань критичними маршрутами
HTTP security headers	CSP, X-Frame-Options, nosniff, Referrer-Policy, Permissions-Policy	Clickjacking, MIME-sniffing, ін’єкції контенту	Додатковий периметр браузерного захисту на рівні відповіді сервера
Шифрування чутливих полів	Fernet для OpenAI-ключа і метаданих історії входів	Читання чутливих полів із дампа БД	Зменшення ризику розкриття секретів і технічних метаданих користувача

2.8 Висновки щодо проєктування вебзастосунку

У другому розділі сформовано проєктну основу вебзастосунку для обліку криптовалютного портфеля на рівні архітектури, модульної структури, моделі

даних, технологічного стеку, інтерфейсної організації та механізмів безпеки. Отримані результати забезпечують цілісність переходу від вимог, визначених у розділі 1, до практичної реалізації системи. У процесі проєктування було враховано особливості предметної області, зокрема необхідність точного обліку активів, збереження історії операцій, відображення аналітичних показників і забезпечення зручної взаємодії користувача із системою.

За результатами архітектурного аналізу обґрунтовано вибір клієнт-серверної моделі SPA + REST API як найбільш збалансованого підходу для поточного масштабу проєкту. Ця модель забезпечує достатню інтерактивність інтерфейсу, прозорий розподіл відповідальності між фронтендом і бекендом та керовану складність супроводу без надлишкового інфраструктурного ускладнення. Водночас такий підхід залишає можливість подальшого розширення системи без повної зміни архітектурної основи та спрощує незалежний розвиток клієнтської і серверної частин.

Виконана декомпозиція системи на функціональні модулі підтвердила можливість ізольованого розвитку ключових підсистем: автентифікації, гаманців та активів, транзакцій, ринкових даних, аналітики, AI-агента й passkey-потоків. Збереження узгоджених API-контрактів між модулями знижує ризик регресій при подальших змінах і створює передумови для поетапного масштабування функціональності. Крім того, модульний поділ спрощує тестування окремих частин системи та полегшує супровід програмного коду.

Проєктування реляційної моделі даних зафіксувало сутності, зв’язки та обмеження цілісності, необхідні для достовірного обліку операцій і коректних портфельних розрахунків. Визначення ключових таблиць та обмежень на рівні схеми БД забезпечує відтворюваність обчислень і контрольовану еволюцію структури даних у процесі розробки. Передбачені зв’язки між користувачами, гаманцями, активами, операціями та ринковими цінами створюють основу для формування актуальних балансів, історичних зрізів і аналітичних показників портфеля.

Проведений аналіз технологій реалізації дозволив сформувати узгоджений стек, у якому кожний інструмент має обґрунтовану роль у межах цільової архітектури. Обрані рішення підтримують вимоги до продуктивності розробки, підтримуваності коду, типобезпечної взаємодії між шарами та відтворюваності середовища. Це важливо для забезпечення стабільної розробки, зменшення кількості помилок інтеграції, спрощення розгортання застосунку та подальшої підтримки реалізованих модулів.

Проєктування інтерфейсу користувача та безпекового контуру виконано як взаємопов'язані складові: інтерфейсні сценарії орієнтовано на практичні задачі користувача, а механізми автентифікації, сесійного контролю, обмеження запитів і захисту чутливих даних забезпечують надійний експлуатаційний профіль системи. Під час формування інтерфейсних рішень враховано необхідність швидкого доступу до основних показників портфеля, журналу операцій та аналітичних представлень.

Окремо реалізовано принцип мінімізації даних для AI-модуля шляхом передачі знеособленого портфельного контексту. Це дозволяє поєднати пояснювальні можливості інтелектуального агента з вимогами до приватності та залишити достатньо даних для формування пояснень щодо стану портфеля, ризиків і змін ринкової ситуації.

Отже, у межах розділу 2 сформовано повний проєктний базис для етапу імплементації: визначено структурну архітектуру, модель даних, технологічні та безпекові рішення, а також логіку інтерфейсної взаємодії. Це створює достатні методичні й технічні передумови для реалізації програмної системи та її подальшого тестування в розділі 3.

Таким чином, результати другого розділу виконують роль проміжної ланки між аналітичним обґрунтуванням системи та безпосереднім створенням програмного продукту. Вони визначають загальний напрям реалізації та технічні орієнтири для розроблення серверної частини, клієнтського інтерфейсу, бази даних і механізмів захисту.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Особливості програмної реалізації

Під час реалізації вебзастосунку для обліку криптовалютного портфеля було збережено архітектурну модель, обґрунтовану в розділі 2: клієнтський рівень відповідає за інтерфейсну взаємодію та підготовку запитів, серверний – за бізнес-логіку, перевірку доступу, обробку операцій і цілісність збереження даних. Такий підхід дозволив відокремити презентаційний контур від доменної логіки та забезпечити керовану еволюцію системи без порушення міжмодульних контрактів.

Клієнтська частина реалізована на React [21] + TypeScript із маршрутизацією за екранними сценаріями та централізованою роботою із серверним станом через HTTP-запити [19]. На цьому рівні зосереджено рендеринг екранів портфеля, транзакцій, аналітики та агента, валідацію користувацьких форм і керування локальним станом інтерфейсу. Ключовим обмеженням є відсутність критичної бізнес-логіки на клієнті: фінальні перевірки операцій, контроль прав доступу та обчислення, що впливають на консистентність даних, виконуються на сервері.

Серверна частина реалізована на FastAPI [22] з поділом на маршрутизатори, сервісний шар і шар доступу до даних. Роутери обробляють HTTP-контракт і валідацію структур запитів/відповідей [19], сервісний шар інкапсулює доменні правила (автентифікація, гаманці, активи, транзакції, аналітика, агент), а доступ до БД виконується через ORM-моделі SQLAlchemy [25]. Така декомпозиція зменшує дублювання коду, спрощує локалізацію змін і забезпечує повторне використання доменних операцій між різними API-маршрутами.

Зберігання даних організоване в PostgreSQL [23], а керування змінами схеми виконується через Alembic-міграції [38]. Практично це означає, що будь-яке розширення функціональності (нові поля, таблиці, зв’язки чи обмеження) фіксується як версійований крок міграції, який відтворювано застосовується в усіх

середовищах. Такий механізм мінімізує ризик розходження між поточною моделлю коду та фактичною схемою БД під час командної роботи або повторного розгортання.

Окремий інтеграційний контур побудовано для зовнішніх сервісів: ринкові котирування отримуються через API CoinGecko і Binance [27], [28], а інтелектуальний функціонал – через OpenAI API [29]. Усі зовнішні виклики ізольовано в окремих сервісах, що дозволяє керувати таймаутами, політиками повторних спроб та деградаційними сценаріями без внесення змін у клієнтські екрани. За такого підходу внутрішній API вебзастосунку залишається стабільним навіть за змін у зовнішніх постачальників даних.

Керування конфігураціями реалізовано через централізований шар налаштувань на основі змінних оточення (.env) [43]: параметри підключення до БД, URL зовнішніх API, політики сесій, ключі інтеграцій і режими запуску. Це дозволяє розділити код і середовищні параметри, спростити перенесення між локальним запуском, контейнерним оточенням і продакшн-конфігурацією [26]. Узагальнену компонентну схему програмної реалізації із відображенням основних рівнів та потоків взаємодії наведено на рисунку 3.1.

Практична послідовність обробки запиту в системі є наскрізною: ініціювання дії в інтерфейсі, передавання структурованого запиту до API, виконання доменних перевірок у сервісному шарі, підтвердження змін у транзакційному контурі БД та повернення узгодженого результату на клієнт. Саме така організація потоку дозволяє гарантувати передбачуваність поведінки вебзастосунку для операцій обліку, зменшує кількість станів гонки між клієнтом і сервером та підтримує єдину логіку перевірок незалежно від конкретного екранного сценарію.

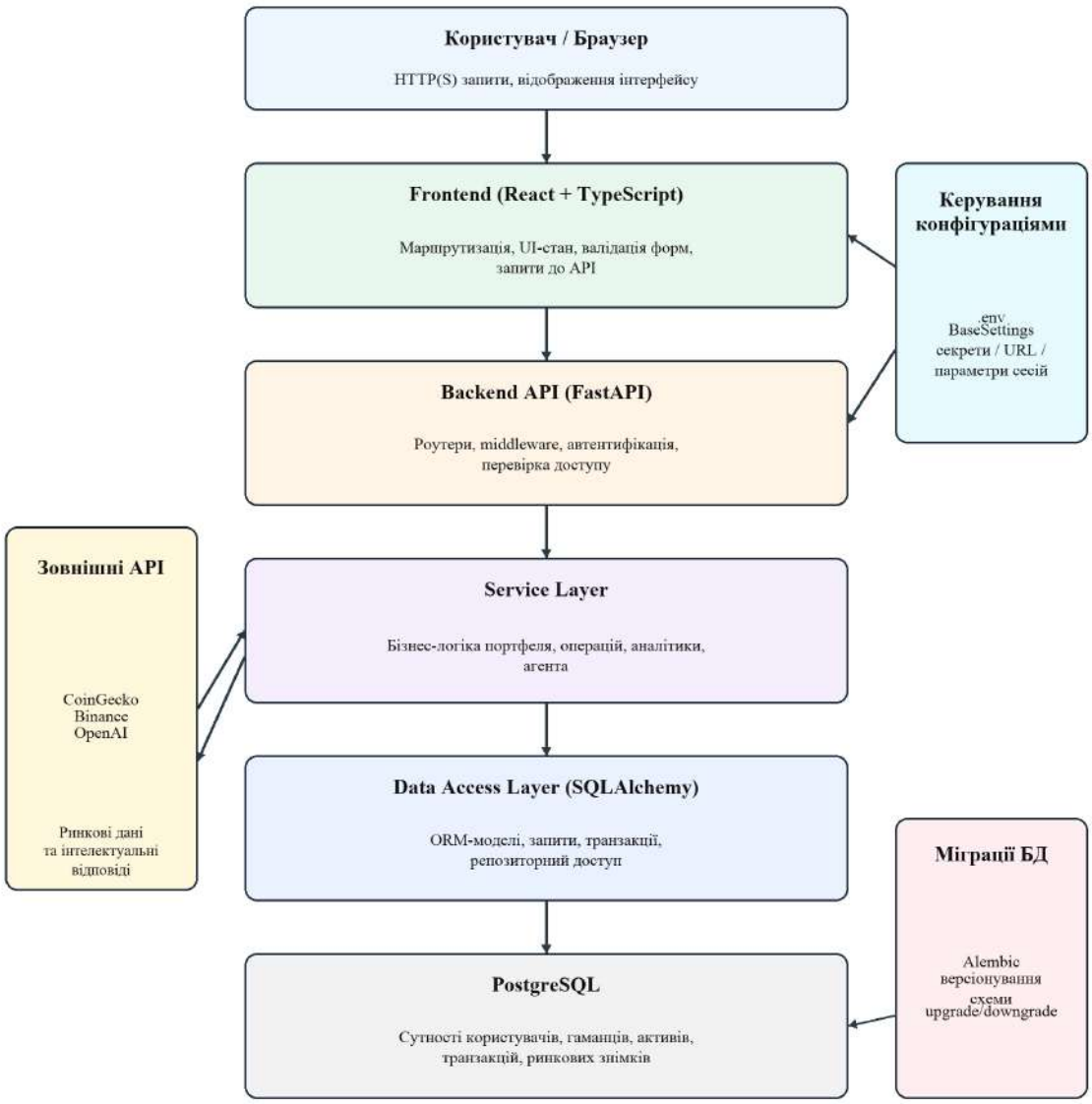


Рисунок 3.1 – Компонентна схема програмної реалізації вебзастосунку

3.2 Програмна реалізація функціональних модулів

Функціональні модулі реалізовано за сервісним принципом: API-маршрути приймають запити, а бізнес-правила зосереджені в сервісному шарі. Це зменшує дублювання коду й ризик регресій. Узагальнену структуру функціональних модулів серверної частини наведено на рисунку 3.2.

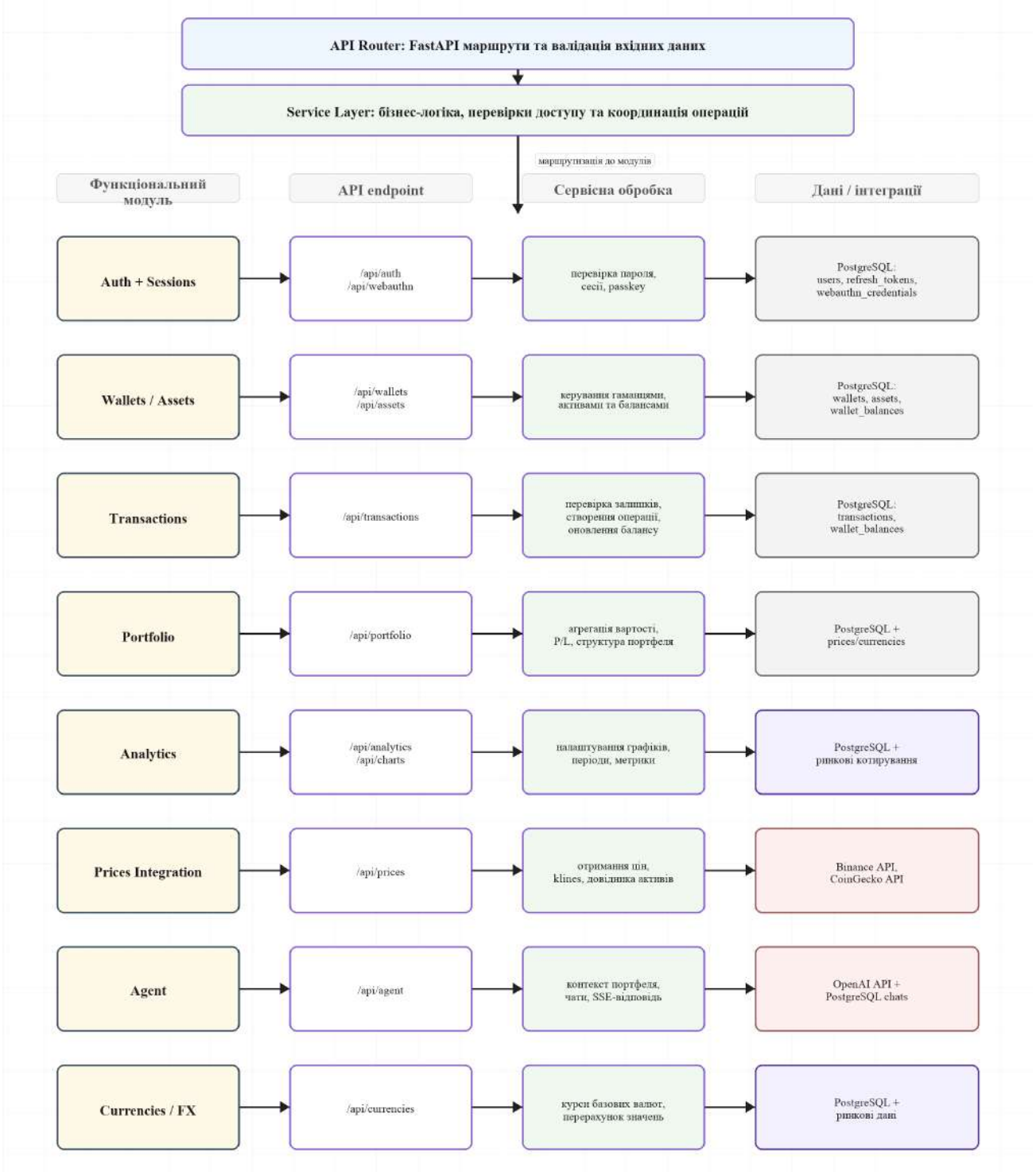


Рисунок 3.2 – Структура функціональних модулів серверної частини

Модуль автентифікації та сесій реалізує парольний і passkey-контури [37] через /api/auth і /api/webauthn. Refresh-токен ротується під час оновлення сесії [31], [35], що знижує ризик повторного використання скомпрометованого токена; фрагменти реалізації наведено в додатку А (лістинги А.1, А.2).

Модуль гаманців і активів забезпечує керування структурою портфеля: створення, редагування і видалення гаманців, облік балансів у межах гаманця та вибір активів для операційного вводу. Це реалізовано через маршрути /api/wallets (включно з вкладеними маршрутами балансів) і /api/assets. Такий поділ ізолює довідниковий контур активів від транзакційного контуру зміни залишків і спрощує подальший супровід.

Модуль транзакцій обробляє поповнення, виведення, переказ і конвертацію. Операція create_transaction(…) перевіряє доступ, залишки, оновлює баланси та фіксує транзакцію в БД; реалізацію наведено в додатку А (лістинг А.3), а послідовність обробки – на рисунку 3.3.

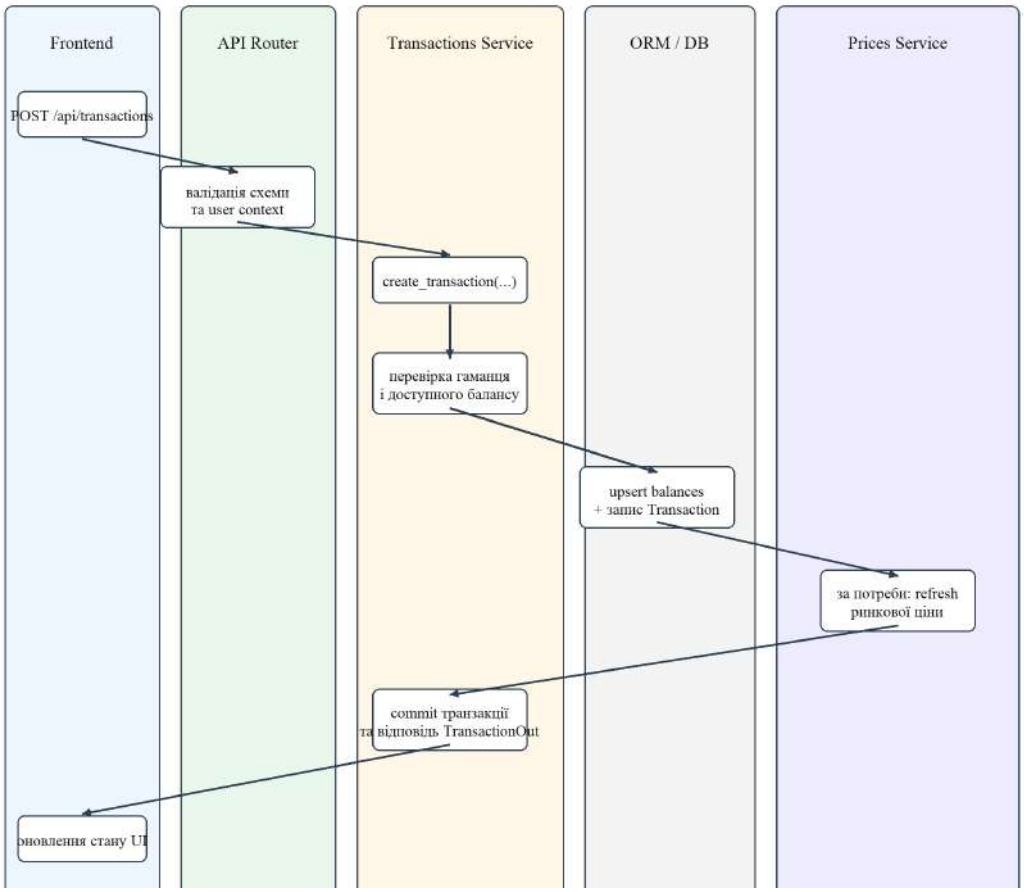


Рисунок 3.3 – Потік обробки транзакції (створення/оновлення)

Модулі аналітики та портфельного зведення формують агреговані показники для інтерфейсних панелей: поточну вартість, динаміку P/L, структурні

зрізи транзакцій і налаштовувані графіки. Реалізація розподілена між /api/portfolio, /api/analytics і /api/charts, що дозволяє відокремити загальне зведення портфеля від керування конфігурацією візуальних елементів.

Інтеграційний модуль цін працює через /api/prices і сервіси BinanceClient та CoingeckoClient [28], [27]. У внутрішній логіці застосовується пріоритетне отримання котирувань із Binance з fallback-переходом на CoinGecko для активів або сценаріїв, де первинне джерело недоступне. Це дає змогу підтримувати безперервність розрахунків вартості портфеля без зміни контракту клієнтської частини.

Модуль агента реалізовано через /api/agent: керування ключем інтеграції, оновлення контексту портфеля, чати та потокова взаємодія з моделлю [29]. Агент не змінює портфель, а працює як допоміжний аналітичний сервіс; фрагменти подано в додатку А (лістинг А.4).

Узагальнену характеристику реалізованих модулів, ключових endpoint-ів, сервісних функцій і отриманого результату наведено у таблиці 3.1.

Таблиця 3.1. Реалізація функціональних модулів вебзастосунку

Модуль	Ключові endpoint-и	Ключові функції сервісного шару	Результат реалізації
Auth + Sessions	/api/auth/register; /api/auth/login; /api/auth/refresh; /api/auth/logout; /api/auth/me; /api/auth/change-password; /api/webauthn/*	register(…); login(…); refresh(…); logout(…); begin_registration(…); complete_authentication(…)	Стійка автентифікація, ротація сесії, підтримка passkeys
Wallets / Assets	/api/wallets; /api/wallets/{wallet_id}/balances; /api/assets	list_wallets(…); create_wallet(…); add_balance(…); update_balance(…); list_assets(…)	Керування структурами портфеля та балансними станами

Продовження Таблиці 3.1. Реалізація функціональних модулів вебзастосунку

Модуль	Ключові endpoint-и	Ключові функції сервісного шару	Результат реалізації
Transactions	/api/transactions; /api/transactions/{id}; /api/transactions/{id}/context	create_transaction(…); list_transactions(…); transaction_balance_context(…)	Коректна фіксація операцій і узгоджений перерахунок залишків
Analytics / Charts	/api/analytics; /api/charts; /api/portfolio	compute_portfolio_analytics(…); list_charts(…); create_chart(…); compute_portfolio(…)	Формування метрик, агрегатів і налаштовуваних графіків
Prices Integration	/api/prices/current; /api/prices/refresh; /api/prices/{symbol}/klines	get_price_in_usdt(...); refresh_all_held(...); fetch_and_store_usdt_price(...)	Оновлення ринкових котирувань із fallback-логікою
Agent	/api/agent/refresh; /api/agent/context; /api/agent/chats; /api/agent/chat	build_snapshot(...); refresh_context(...); stream_chat_persistent(...)	Пояснювальна аналітика поверх знеособленого контексту
Currencies / FX	/api/currencies	supported_currencies(...); get_usdt_to_fiat(...)	Підтримка базових валют і конвертаційних сценаріїв

Наведена таблиця 3.1 показує, що модулі мають чітко визначені межі відповідальності: API-рівень залишається стабільним для клієнта, тоді як зміни доменної поведінки локалізуються в сервісному шарі. Це є принциповим для подальшого розширення функціональності без масових змін у суміжних підсистемах.

Окремо доцільно виділити сесійний сценарій авторизації та оновлення токена, оскільки він проходить через кілька технічно чутливих етапів: перевірку облікових даних, запис і оновлення refresh-токена, видачу access-токена і оновлення cookie-атрибутів. Узгоджену послідовність цього процесу наведено на рисунку 3.4.

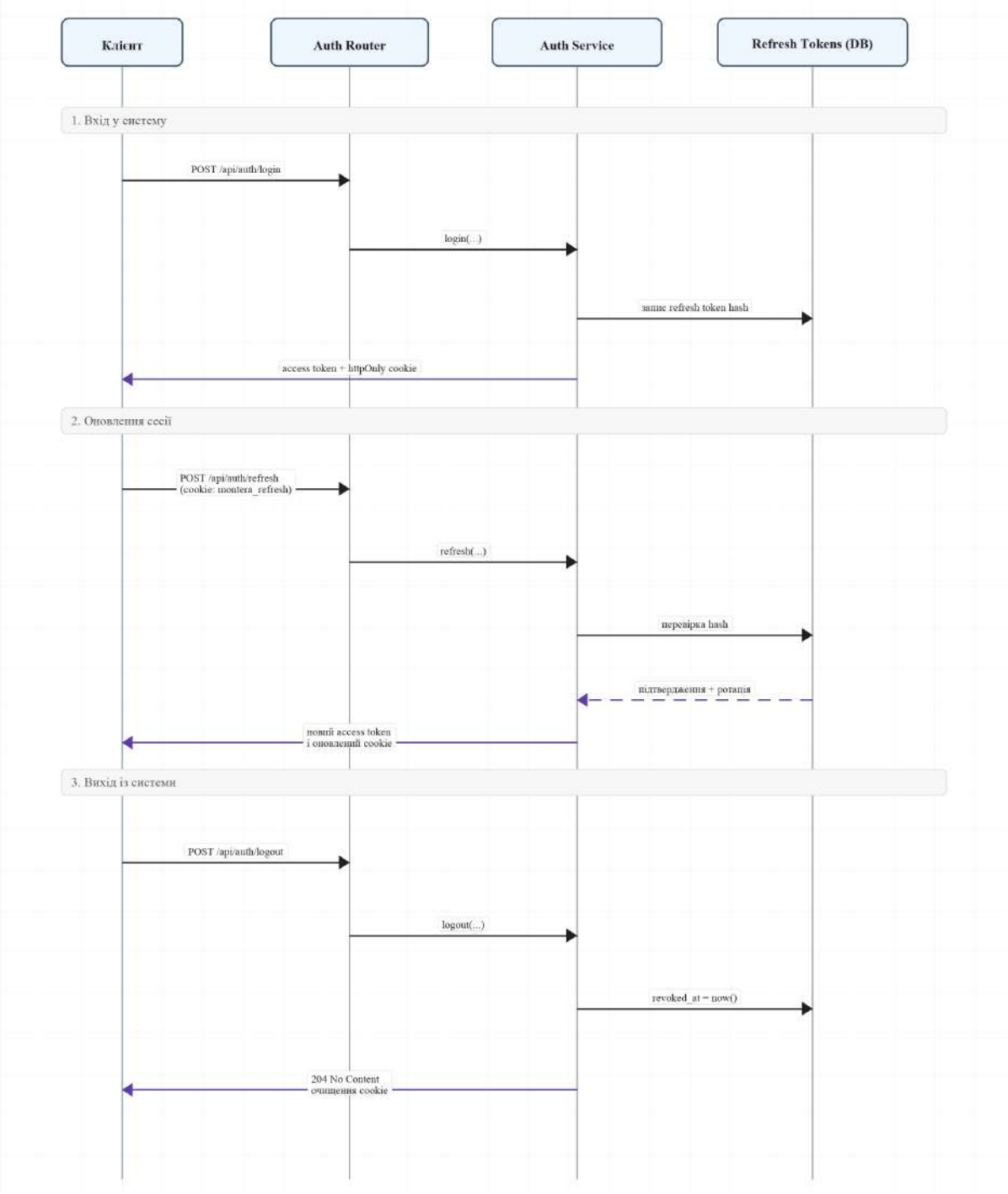


Рисунок 3.4 – Діаграма послідовності авторизації та оновлення сесії

Отже, реалізація модулів покриває основні прикладні сценарії, ізолює зовнішні інтеграції та зберігає критичні облікові дії в контрольованому сервісному шарі.

3.3 Реалізація інтерфейсу користувача

Інтерфейс реалізовано як окремий прикладний контур, що поєднує маршрутизацію, керування станами, валідацію форм і синхронізацію з API.

Маршрутизація побудована на React Router [39] із поділом на публічні /login, /register і захищені робочі екрани. Перенаправлення через replace [40] стабілізує сценарії входу/виходу, а приклади клієнтського сесійного контуру наведено в додатку А (лістинги А.5, А.6). Узагальнену схему маршрутизації подано на рисунку 3.5.

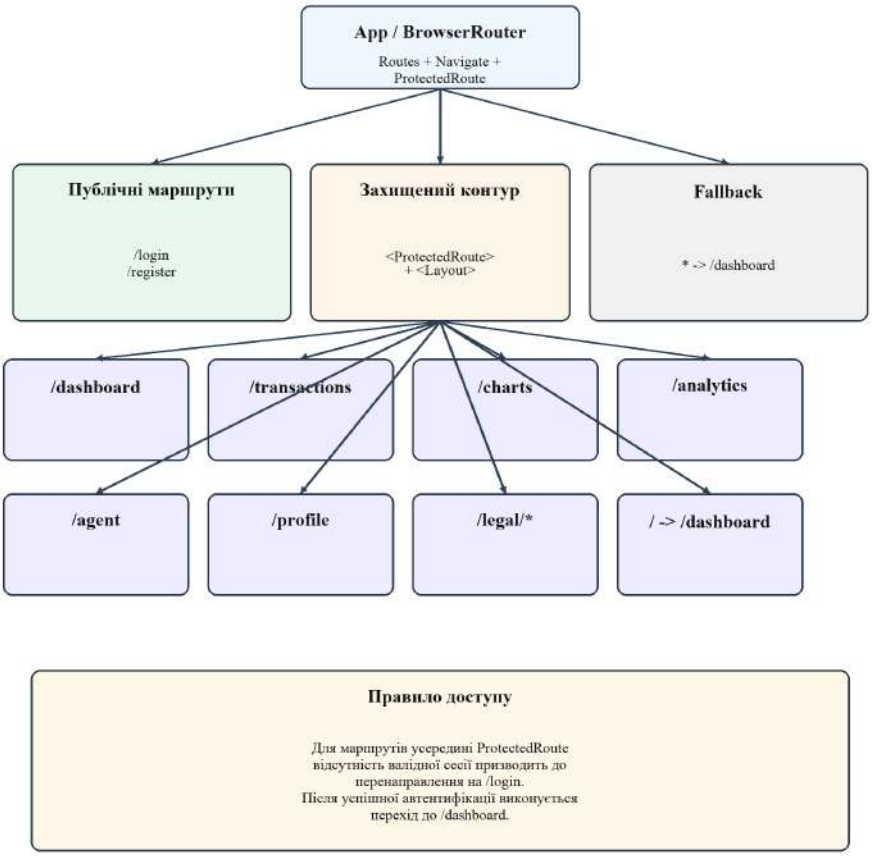


Рисунок 3.5 – Схема маршрутизації інтерфейсу вебзастосунку

Керування станами UI поєднує локальний стан компонентів і серверний стан через React Query [41]. Для змін використовуються useMutation [42] та invalidateQueries [41], що узгоджує UI з актуальними даними backend. Потік станів інтерфейсу та його інтеграцію з API наведено на рисунку 3.6.

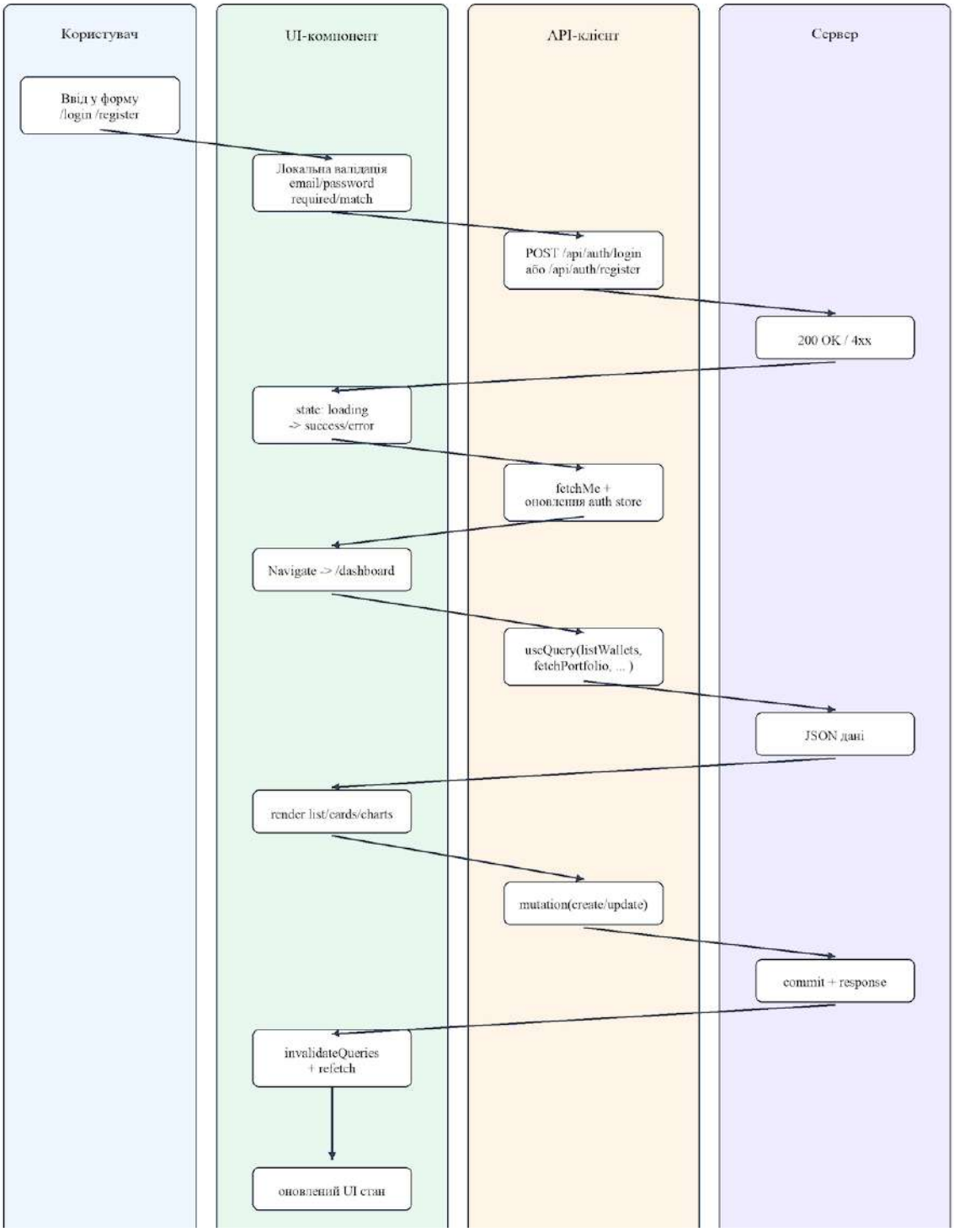


Рисунок 3.6 – Потік станів UI та інтеграція з API

Адаптивність забезпечено зміною композиції елементів: у десктопному режимі використовуються багатоколонкові блоки, у мобільному – вертикальний потік і компактна навігація. Схему наведено на рисунку 3.7.

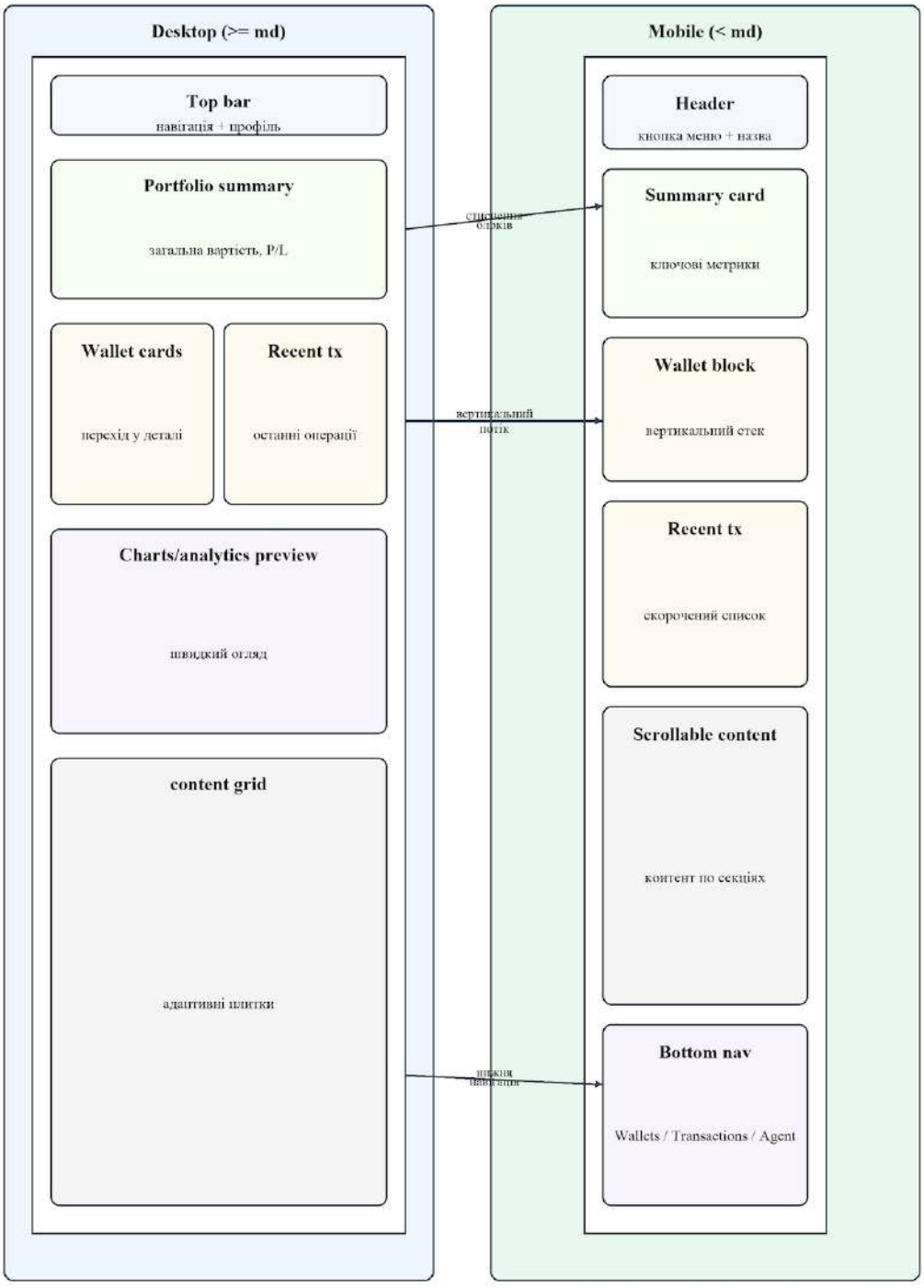


Рисунок 3.7 – Адаптивна організація головного екрана вебзастосунку

Відповідність екранних форм, ключових елементів інтерфейсу та пов’язаних функціональних модулів узагальнено у таблиці 3.2.

Таблиця 3.2. Відповідність екранів інтерфейсу ключовим UI-елементам і пов’язаним модулям

Екран / маршрут	Ключові UI-елементи	Пов’язані модулі (API/сервіси)
/login	Поля email/пароль, кнопка входу, сценарій passkey, повідомлення помилки/lockout	auth, webauthn, session management
/register	Поля реєстрації, індикатор складності пароля, підтвердження пароля, генератор пароля	auth, user profile bootstrap
/dashboard	Зведення портфеля, картки гаманців, швидке створення гаманця, останні транзакції	wallets, portfolio, transactions
/transactions	Фільтри (гаманець/тип/актив/період), список транзакцій, деталізація запису	transactions, assets, wallets
/charts	Налаштовувані графіки активів, додавання/видалення графіків	charts, prices
/analytics	Зведені метрики P/L, розподіли за типами, деталізація по гаманцях	analytics, portfolio
/agent	Список чатів, вікно діалогу, оновлення контексту, потокова відповідь	agent, agent privacy, openai integration
/profile	Зміна пароля, passkey-налаштування, керування OpenAI ключем, журнал входів	auth, webauthn, login history, agent key
/legal/*	Сторінки політик і довідкового контенту	static/legal content module

Наведена таблиця 3.2 підтверджує, що екранна структура інтерфейсу напряму відображає межі функціональних модулів: кожна користувацька дія пов’язана з конкретним API-контрактом і контрольованим оновленням стану. Це забезпечує цілісність взаємодії UI з сервером та зменшує ризик неузгоджених переходів між екранами під час подальшого розвитку вебзастосунку.

3.4 Вимоги до технічних та програмних засобів

Вимоги сформовано для двох режимів: користувацького запуску в браузері та локальної розробки із backend, БД, міграціями й frontend-збиранням.

Для користувача потрібні сучасний браузер, JavaScript і стабільне мережеве підключення. Для розробника – Python 3.11+, Node.js, PostgreSQL та інструменти керування залежностями й міграціями.

Систематизовані мінімальні та рекомендовані параметри для запуску і розробки вебзастосунку наведено у таблиці 3.3.

Таблиця 3.3. Мінімальні та рекомендовані вимоги до технічних і програмних засобів

Позиція	Мінімальні вимоги	Рекомендовані вимоги	Призначення / коментар
Клієнтський пристрій (запуск)	CPU 2 ядра, RAM 4 ГБ	CPU 4+ ядер, RAM 8 ГБ	Стабільна робота інтерфейсу з графіками та списками
Браузер	Актуальна версія Chrome / Edge / Firefox / Safari	Остання стабільна версія Chrome / Edge	Підтримка сучасного JS, fetch, local storage, cookies
Мережа	Широкосмугове підключення від 5 Мбіт/с	Від 20 Мбіт/с із низькою затримкою	Оновлення ринкових даних, робота з API та агентом

Продовження таблиці 3.3. Мінімальні та рекомендовані вимоги до технічних і програмних засобів

Позиція	Мінімальні вимоги	Рекомендовані вимоги	Призначення / коментар
Сервер застосунку (локальний/ тестовий)	CPU 2 ядра, RAM 4 ГБ, SSD 10 ГБ	CPU 4 ядра, RAM 8–16 ГБ, SSD 20+ ГБ	Одночасний запуск FastAPI, фонових задач і БД
База даних	PostgreSQL 16 (або сумісна 15+)	PostgreSQL 16 із регулярним бекапом	Збереження користувачів, гаманців, транзакцій, аналітики
Backend-стек	Python 3.11+, pip/uv, Alembic	Python 3.11+ у ізольованому venv, автоматизовані міграції	Запуск API, сервісів та керування схемою БД
Frontend-стек	Node.js 20+, npm, Vite	Node.js 20/22 LTS, npm, контроль версій залежностей	Збирання і запуск клієнтської частини React + TypeScript
Додаткові засоби	Docker Compose (опційно), Git	Docker Compose + стандартизовані .env-конфігурації	Відтворюваний локальний стенд і зручний командний запуск

Вимоги, наведені у таблиці 3.3, покривають типовий робочий сценарій без перевантаження інфраструктури: локальний запуск через Python/Node/PostgreSQL або контейнеризований запуск через Docker Compose [26]. Рекомендований профіль доцільний для стабільної роботи під час одночасного тестування модулів транзакцій, аналітики та агентного контуру.

Окремо слід враховувати, що функції інтелектуального модуля залежать від наявності коректно налаштованого ключа зовнішнього AI-сервісу [29], тоді як базовий контур обліку портфеля (гаманці, активи, транзакції, аналітика)

залишається працездатним і без цього компонента. Це забезпечує деградацію функціоналу без втрати ядра системи.

3.5 Тестування вебзастосунку

3.5.1 Аналіз методів тестування

Тестування розглядається як інженерний контур перевірки бізнес-логіки, API-взаємодії та ключових користувацьких сценаріїв [49].

Під час відбору методів тестування враховано чотири практичні критерії:

– рівень покриття ключових ризиків;

– відтворюваність результатів;

– вартість супроводу тестів у процесі розвитку коду;

– відповідність масштабу кваліфікаційної роботи.

Такий підхід дозволяє зосередитися на методах, що забезпечують максимальну практичну користь за наявних часових і ресурсних обмежень.

Для backend обрано інтеграційні API-тести [46], оскільки доменні правила проходять через маршрут, сервісний шар і БД. Для frontend використано e2e-перевірки ключових сценаріїв у Playwright [44], [45].

До обов’язкового контуру включено безпекові сценарії: заголовки безпеки [36], rate limiting і контроль доступу до auth- та agent-маршрутів [14], [31], [47], [48]. Узагальнення наведено в таблиці 3.4.

Таблиця 3.4. Обґрунтування вибору методів тестування вебзастосунку

Метод тестування	Статус у проєкті	Що перевіряється	Обґрунтування рішення
Статичний аналіз (Ruff, ESLint)	Використовується частково	Порушення стилю, типові дефекти, ризикові конструкції коду	Дає раннє виявлення частини помилок і підтримує єдині стандарти коду без високих витрат часу.

Продовження таблиці 3.4. Обґрунтування вибору методів тестування вебзастосунку

Метод тестування	Статус у проєкті	Що перевіряється	Обґрунтування рішення
Інтеграційне тестування API (pytest + httpx + тестова PostgreSQL)	Використовується як базовий метод backend	Контракти endpoint-ів, бізнес-правила сервісів, коректність запису/читання даних	У цій архітектурі ключова логіка проходить через API та БД, тому саме цей рівень виявляє найбільш критичні збої.
Безпекові сценарії API	Використовується	Заголовки безпеки, rate limiting, контроль доступу до auth/webauthn/agent маршрутів	Перевіряються практичні захисні механізми, які безпосередньо впливають на безпеку сесій і даних користувача.
Наскрізне e2e-тестування (Playwright)	Використовується	Кінцеві користувацькі потоки від UI до API	Підтверджує, що клієнтська і серверна частини працюють узгоджено у реальних сценаріях експлуатації.
Модульні тести ізольованих функцій	Використовується обмежено	Локальні обчислення та окремі перевірки правил	Застосовуються там, де ізоляція справді знижує складність перевірки; основний акцент залишено на інтеграційному рівні.
Навантажувальне та стрес-тестування	Не застосовується в межах роботи	Стійкість при піковому паралельному навантаженні	Потребує окремого стенду, сценаріїв профілювання та навантажувального контуру, що виходить за межі поточного етапу.
Зовнішній penetration testing	Не застосовується в межах роботи	Глибинний аудит вразливостей у бойовому або наближеному до бойового контурі	Для коректного виконання потрібні окремі інструменти, організаційний контур і формалізований процес безпекового аудиту.

Отже, обрано комбіновану стратегію: інтеграційні backend-тести, безпекові сценарії, e2e-перевірки інтерфейсу та частковий статичний контроль [49]. Навантажувальне тестування й зовнішній penetration testing доцільні на наступних ітераціях.

У підпункті 3.5.2 наведено результати виконання обраних тестових сценаріїв.

3.5.2 Аналіз результатів тестування

Практичне тестування виконано після актуалізації тестового набору відповідно до поточної реалізації API та інтерфейсу. Перевірка охопила серверний контур і клієнтські e2e-сценарії.

Серверний контур запускався через Python/pytest [46] із тестовою PostgreSQL-базою [23]. Перевірено автентифікацію, сесії, гаманці, транзакції, аналітику, цінові сервіси та захисні механізми API; лог наведено на рисунку 3.8.

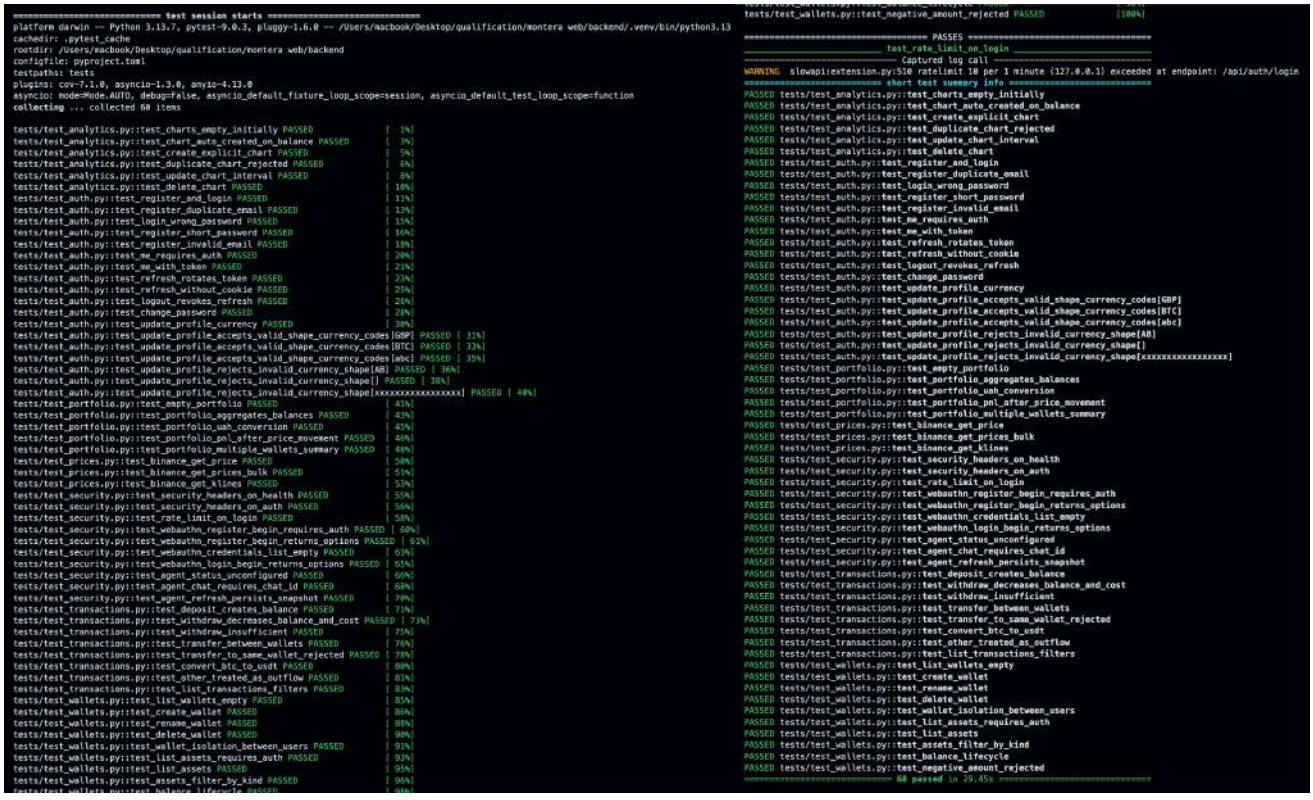


Рисунок 3.8 – Результати прогону backend-тестів у середовищі pytest

Серверний прогін містив 60 тестів, усі завершилися успішно. Підтверджено роботу rate limiting, токенів сесії, валідації даних і доменних операцій із балансами та транзакціями.

Клієнтський контур перевірявся через Playwright [44]: реєстрація/вхід, logout, зміна пароля, гаманці, активи, транзакції та графіки. Лог прогону наведено на рисунку 3.9.

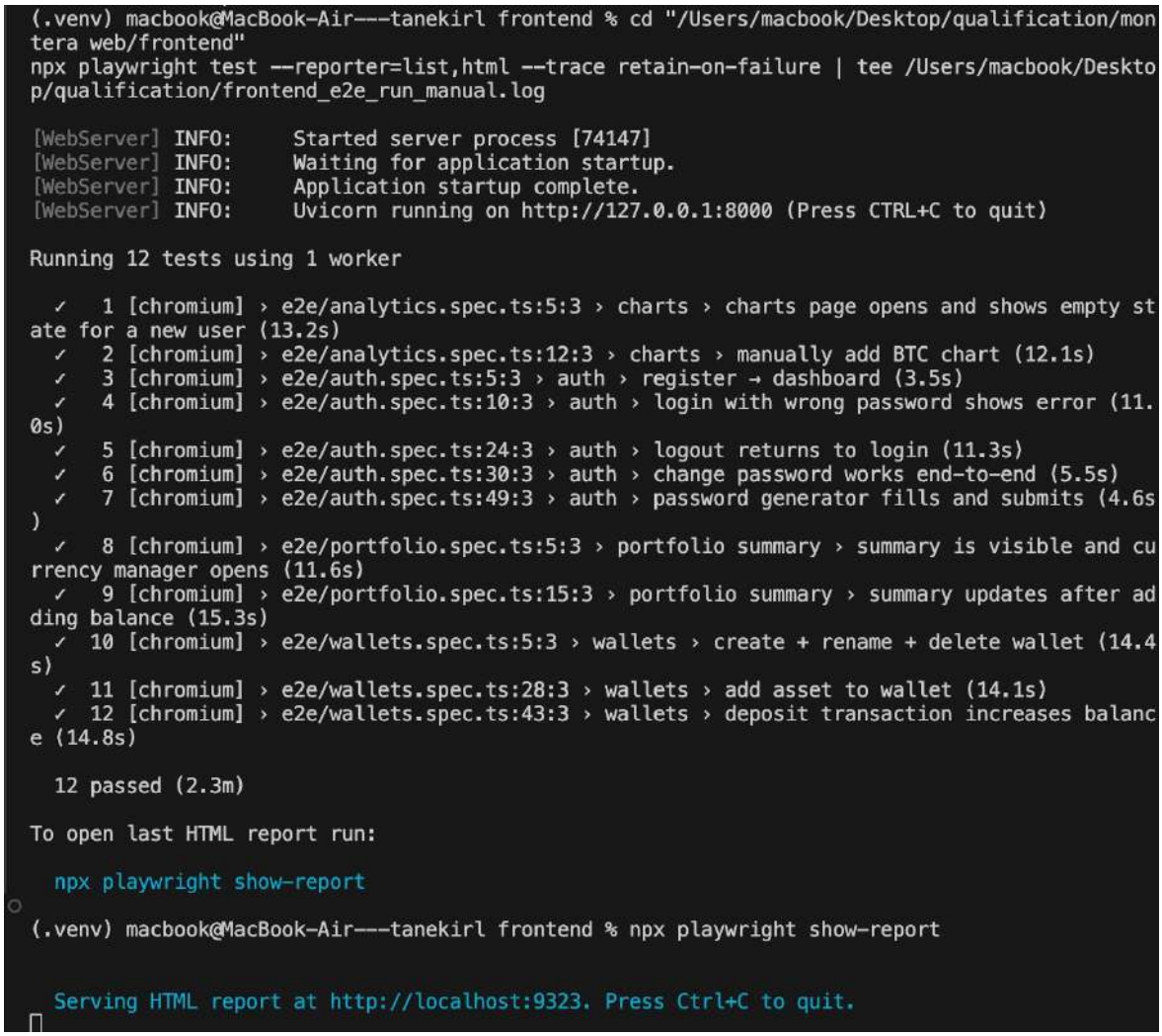


Рисунок 3.9 – Результати прогону frontend e2e-тестів у Playwright

Клієнтський прогін містив 12 тестів із 12 успішних. HTML-звіт Playwright [44] показує структуру набору, статуси тестів і загальну статистику (рисунок 3.10).

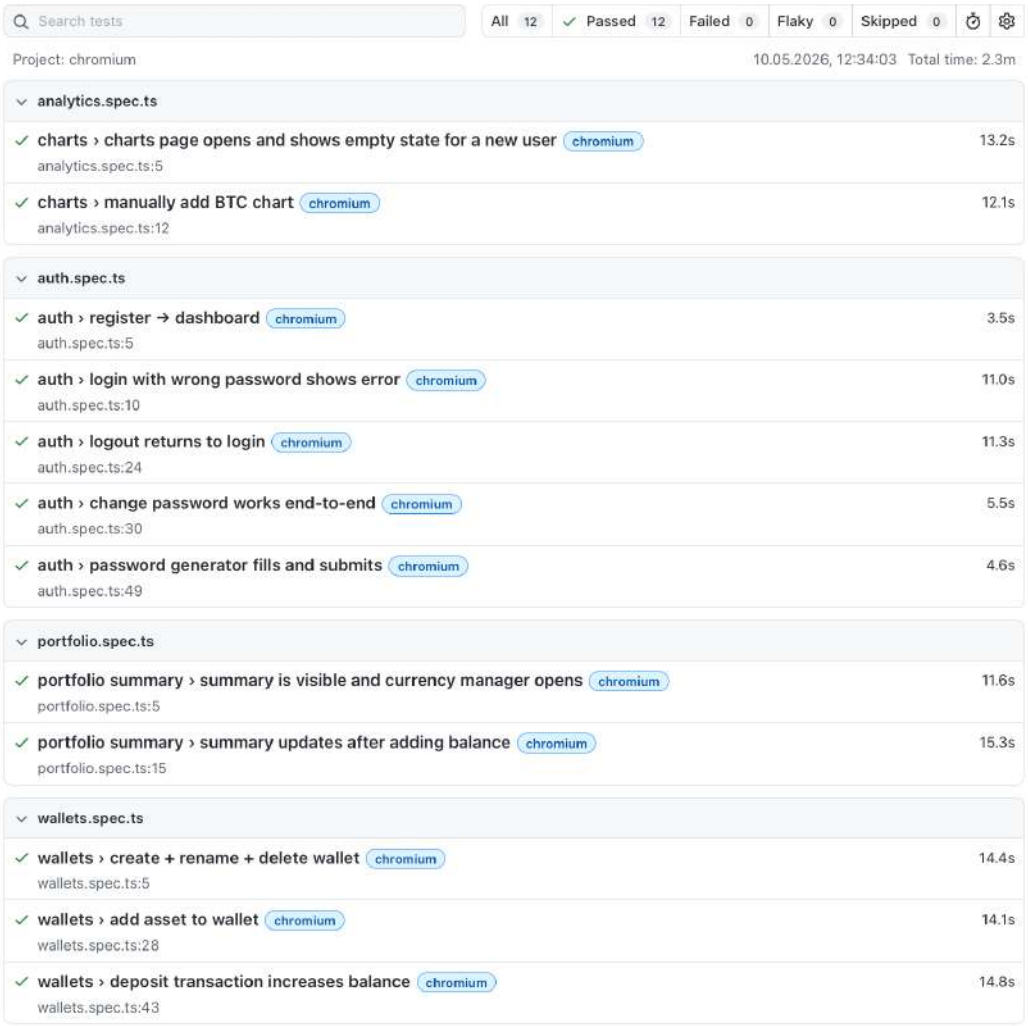


Рисунок 3.10 – HTML-звіт Playwright за підсумками e2e-тестування

Узагальнені кількісні результати обох контурів тестування систематизовано у таблиці 3.5.

Таблиця 3.5. Підсумкова відомість результатів автоматизованого тестування

Тестовий контур	Інструмент	Охоплення перевірки	Запущено тестів	Успішно	Неуспішно
Серверний	pytest	API-контракти, доменна логіка, безпека, сесії, валідації	60	60	0

Продовження таблиці 3.5. Підсумкова відомість результатів автоматизованого тестування

Тестовий контур	Інструмент	Охоплення перевірки	Запущено тестів	Успішно	Неуспішно
Клієнтський	Playwright	Наскрізні UI-сценарії (auth, wallets, transactions, charts, portfolio)	12	12	0
Разом	–	Повний автоматизований контур перевірки поточної версії вебзастосунку	72	72	0

Підсумкову діаграму результатів backend- і frontend-тестів наведено на рисунку 3.11.

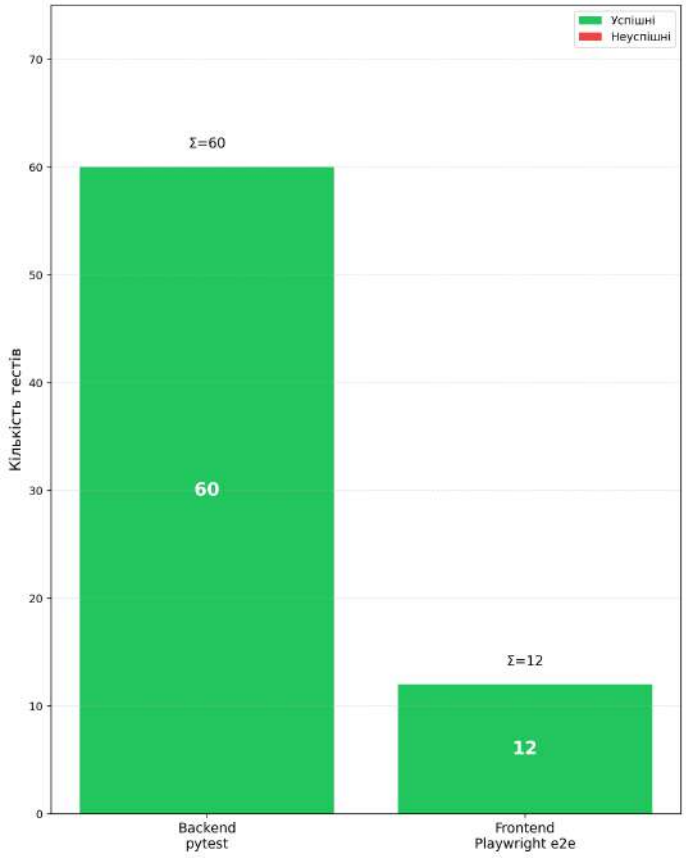


Рисунок 3.11 – Підсумкові результати автоматизованого тестування

Отже, тестування підтвердило стабільну роботу основних модулів і узгодженість взаємодії між серверною та клієнтською частинами.

3.6 Висновки до третього розділу

У третьому розділі розглянуто особливості програмної реалізації вебзастосунку. Описано структуру клієнтської та серверної частин, принципи взаємодії через програмний інтерфейс, організацію сервісного шару, роботу з базою даних, міграціями та конфігураційними параметрами. Така структура дозволяє розділити відповідальність між окремими частинами системи та спростити супровід програмного коду.

Описано реалізацію основних функціональних модулів: автентифікації та сесій, гаманців і активів, транзакцій, аналітики, отримання ринкових даних і взаємодії з інтелектуальним помічником. Для кожного модуля визначено його призначення, основні маршрути взаємодії та результат виконання ключових операцій.

Окремо проаналізовано реалізацію інтерфейсу користувача: маршрутизацію екранів, поділ на публічні та захищені сторінки, керування станами інтерфейсу, валідацію форм, синхронізацію даних із серверною частиною та адаптивність. Це дозволяє оцінити не лише зовнішній вигляд інтерфейсу, а й логіку його роботи під час взаємодії з користувачем.

Також визначено вимоги до технічних і програмних засобів, необхідних для запуску вебзастосунку в користувацькому та розробницькому режимах. У межах тестування описано вибір методів перевірки програмного забезпечення та наведено результати виконання тестових сценаріїв. За результатами тестування підтверджено, що реалізовані модулі працюють коректно в межах визначених сценаріїв і можуть бути використані як основа для подальшого розвитку вебзастосунку.

ВИСНОВКИ

У кваліфікаційній роботі розв’язано актуальне прикладне завдання проєктування та реалізації вебзастосунку для обліку криптовалютного портфеля. Досягнуто головної мети дослідження: створено цілісне програмне рішення, яке забезпечує структурований облік активів, фіксацію транзакцій, отримання аналітики портфеля та підтримку прийняття рішень користувачем на основі актуальних даних.

У процесі виконання роботи проведено аналіз предметної області, функціональних і нефункціональних вимог, а також обґрунтовано архітектурні та технологічні рішення. Для системи обрано модель SPA + REST API, сформовано модульну структуру серверної та клієнтської частин, спроєктовано модель даних і визначено принципи взаємодії між бізнес-логікою, API-контрактами та інтерфейсним контуром.

Практичним результатом є реалізований вебзастосунок із повним базовим контуром функцій: автентифікація і сесії, керування гаманцями та активами, операції транзакцій, портфельна аналітика, графічне подання динаміки, інтеграція із зовнішніми джерелами ринкової інформації та агентний модуль. Реалізація інтерфейсу орієнтована на передбачувані користувацькі сценарії, цілісність станів та адаптивність для різних класів пристроїв.

Окрему увагу приділено безпеці прикладного контуру та перевірці якості реалізації. Результати тестування підтвердили коректність роботи ключових модулів і стабільність взаємодії між backend та frontend: у серверному контурі успішно виконано 60 із 60 тестів, у клієнтському e2e-контурі – 12 із 12. Отримані результати засвідчують достатній рівень технічної готовності розробленого програмного рішення для практичного використання та подальшого розвитку.

Отже, поставлені в роботі завдання виконано в повному обсязі, а сформовані проєктні та програмні рішення є узгодженими, обґрунтованими та практично цінними. Перспективами подальших робіт є розширення набору аналітичних

інструментів, поглиблення нефункціональних перевірок (продуктивність, стійкість під навантаженням, розширений безпековий аудит) та розвиток інтелектуальних сценаріїв підтримки користувача у межах предметної області вебзастосунку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cambridge Centre for Alternative Finance (University of Cambridge). 3rd Global Cryptoasset Benchmarking Study. 2021. URL: https://www.jbs.cam.ac.uk/wp-content/uploads/2021/01/2021-ccaf-3rd-global-cryptoasset-benchmarking-study.pdf (дата звернення: 12.02.2026).

2. Chainalysis. The 2024 Geography of Crypto Report (Release). 2024. URL: https://www.chainalysis.com/wp-content/uploads/2024/10/the-2024-geography-of-crypto-report-release.pdf (дата звернення: 12.02.2026).

3. European Central Bank. Financial Stability Review. November 2025. URL: https://www.ecb.europa.eu/press/financial-stability-publications/fsr/pdf/ecb.fsr202511~263b5810d4.en.pdf (дата звернення: 12.02.2026).

4. International Monetary Fund. Understanding Stablecoins. 2025. URL: https://www.imf.org/-/media/files/publications/dp/2025/english/usea.pdf (дата звернення: 12.02.2026).

5. Financial Action Task Force (FATF). Updated Guidance for a Risk-Based Approach to Virtual Assets and Virtual Asset Service Providers. 2021. URL: https://www.fatf-gafi.org/content/dam/fatf-gafi/guidance/Updated-Guidance-VA-VASP.pdf.coredownload.inline.pdf (дата звернення: 12.02.2026).

6. CoinStats. Crypto Portfolio Tracker (офіційна сторінка портфельного трекера). URL: https://coinstats.app/portfolio/ (дата звернення: 16.02.2026).

7. Delta by eToro. Features (офіційна сторінка можливостей сервісу Delta). URL: https://delta.app/en/features (дата звернення: 16.02.2026).

8. CoinMarketCap. Portfolio Tracker (офіційна сторінка портфельного трекера). URL: https://coinmarketcap.com/portfolio-tracker/ (дата звернення: 16.02.2026).

9. CoinGecko. Free Crypto Portfolio Tracker (офіційна сторінка портфеля). URL: https://www.coingecko.com/en/portfolio (дата звернення: 16.02.2026).

10. CoinTracking. The Crypto Portfolio Tracker & Tax Calculator (офіційний сайт). URL: https://cointracking.info/ (дата звернення: 16.02.2026).

11. CoinGecko API Documentation (офіційна документація API). URL: https://docs.coingecko.com/ (дата звернення: 23.02.2026).

12. Binance Spot API Documentation (офіційна документація Binance Spot API, REST API). URL: https://developers.binance.com/docs/binance-spot-api-docs/rest-api (дата звернення: 23.02.2026).

13. BIS Quarterly Review. DeFi risks and the decentralisation illusion. 2021. URL: https://www.bis.org/publ/qtrpdf/r_qt2112b.pdf (дата звернення: 23.02.2026).

14. OWASP. Application Security Verification Standard (ASVS). URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 23.02.2026).

15. OWASP Cheat Sheet Series. Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 23.02.2026).

16. AWS. The Difference Between Monolithic and Microservices Architecture. URL: https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/ (дата звернення: 02.03.2026).

17. Microsoft Learn. Best practices for RESTful web API design. URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата звернення: 02.03.2026).

18. Microsoft Learn. Architecture styles – N-tier architecture. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier (дата звернення: 02.03.2026).

19. IETF RFC 9110. HTTP Semantics. URL: https://www.rfc-editor.org/rfc/rfc9110 (дата звернення: 02.03.2026).

20. MDN Web Docs. SPA (Single-page application). URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 02.03.2026).

21. React Documentation. URL: https://react.dev/ (дата звернення: 11.03.2026).

22. FastAPI Documentation. URL: https://fastapi.tiangolo.com/ (дата звернення: 11.03.2026).

23. PostgreSQL Documentation (current). URL: https://www.postgresql.org/docs/current/ (дата звернення: 11.03.2026).

24. PyCA Cryptography. Fernet (symmetric encryption). URL: https://cryptography.io/en/latest/fernet/ (дата звернення: 11.03.2026).

25. SQLAlchemy 2.0 Documentation. URL: https://docs.sqlalchemy.org/en/20/ (дата звернення: 11.03.2026).

26. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 20.03.2026).

27. CoinGecko API Documentation. URL: https://docs.coingecko.com/ (дата звернення: 20.03.2026).

28. Binance Spot API Documentation (REST API). URL: https://developers.binance.com/docs/binance-spot-api-docs/rest-api (дата звернення: 20.03.2026).

29. OpenAI API Documentation. URL: https://developers.openai.com/api/ (дата звернення: 20.03.2026).

30. OWASP Application Security Verification Standard (ASVS). URL: https://owasp.org/www-project-application-security-verification-standard/ (дата звернення: 20.03.2026).

31. OWASP Cheat Sheet Series. Session Management Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html (дата звернення: 29.03.2026).

32. OWASP Cheat Sheet Series. Password Storage Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html (дата звернення: 29.03.2026).

33. IETF RFC 9106. Argon2 Memory-Hard Function. URL: https://www.rfc-editor.org/rfc/rfc9106 (дата звернення: 29.03.2026).

34. NIST. FIPS 180-4 Secure Hash Standard (SHA). URL: https://csrc.nist.gov/pubs/fips/180-4/upd1/final (дата звернення: 29.03.2026).

35. IETF RFC 7519. JSON Web Token (JWT). URL: https://www.rfc-editor.org/rfc/rfc7519 (дата звернення: 29.03.2026).

36. OWASP Cheat Sheet Series. HTTP Headers Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html (дата звернення: 09.04.2026).

37. W3C Recommendation. Web Authentication: Level 2 (WebAuthn). URL: https://www.w3.org/TR/webauthn-2/ (дата звернення: 09.04.2026).

38. Alembic Documentation. URL: https://alembic.sqlalchemy.org/ (дата звернення: 09.04.2026).

39. React Router Documentation. URL: https://reactrouter.com/ (дата звернення: 09.04.2026).

40. React Router – useNavigate. URL: https://reactrouter.com/api/hooks/useNavigate (дата звернення: 09.04.2026).

41. TanStack Query – Query Invalidation (React v5). URL: https://tanstack.com/query/v5/docs/framework/react/guides/query-invalidation (дата звернення: 22.04.2026).

42. TanStack Query – useMutation (React). URL: https://tanstack.com/query/latest/docs/react/reference/useMutation (дата звернення: 22.04.2026).

43. FastAPI – Settings and Environment Variables. URL: https://fastapi.tiangolo.com/advanced/settings/ (дата звернення: 22.04.2026).

44. Playwright – Reporters. URL: https://playwright.dev/docs/test-reporters (дата звернення: 22.04.2026).

45. Playwright – Command Line. URL: https://playwright.dev/docs/test-cli (дата звернення: 22.04.2026).

46. pytest Documentation (stable). URL: https://docs.pytest.org/en/stable/contents.html (дата звернення: 06.05.2026).

47. RFC 6265: HTTP State Management Mechanism. URL: https://www.rfc-editor.org/rfc/rfc6265 (дата звернення: 06.05.2026).

48. MDN – Set-Cookie header. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Set-Cookie (дата звернення: 06.05.2026).

49. OWASP Web Security Testing Guide (v4.2). URL: https://owasp.org/www-project-web-security-testing-guide/v42/ (дата звернення: 06.05.2026).

ДОДАТОК А

(обов’язковий)

КЛЮЧОВІ ФРАГМЕНТИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Лістинг А.1 – Повна реалізація маршрутизатора автентифікації (backend/app/routers/auth.py)

```
from fastapi import APIRouter, Request, Response, status

from app.deps import CurrentUser, DBDep
from app.rate_limit import limiter
from app.schemas.auth import LoginEventOut, LoginIn, RegisterIn, TokenOut
from app.schemas.user import ChangePasswordIn, UpdateProfileIn, UserOut
from app.services import auth as auth_service
from app.services import login_history

router = APIRouter(prefix="/api/auth", tags=["auth"])


@router.post("/register", response_model=UserOut, status_code=status.HTTP_201_CREATED)
@limiter.limit("5/minute")
async def register(data: RegisterIn, db: DBDep, request: Request) -> UserOut:
    user = await auth_service.register(db, data)
    return UserOut.model_validate(user)


@router.post("/login", response_model=TokenOut)
@limiter.limit("10/minute")
async def login(data: LoginIn, db: DBDep, request: Request, response: Response) -> TokenOut:
    return await auth_service.login(db, response, request, data)


@router.post("/refresh", response_model=TokenOut)
async def refresh(db: DBDep, request: Request, response: Response) -> TokenOut:
    return await auth_service.refresh(db, response, request)


@router.post("/logout", status_code=status.HTTP_204_NO_CONTENT)
async def logout(db: DBDep, request: Request, response: Response) -> Response:
    await auth_service.logout(db, response, request)
    return Response(status_code=status.HTTP_204_NO_CONTENT)


@router.get("/me", response_model=UserOut)
async def me(user: CurrentUser) -> UserOut:
    return UserOut.model_validate(user)


@router.post("/change-password", status_code=status.HTTP_204_NO_CONTENT)
async def change_password(
    data: ChangePasswordIn, user: CurrentUser, db: DBDep
) -> Response:
    await auth_service.change_password(db, user, data.current_password, data.new_password)
```

```
    return Response(status_code=status.HTTP_204_NO_CONTENT)


@router.get("/login-events", response_model=list[LoginEventOut])
async def login_events(user: CurrentUser, db: DBDep, limit: int = 10) -> list[LoginEventOut]:
    """Recent successful logins for the current user.

    All identifying fields (IP, full user-agent, city/country) are
    decrypted on demand from Fernet-encrypted at-rest storage. The DB
    holds nothing readable until the user themselves makes this call.
    """
    safe_limit = max(1, min(limit, 50))
    rows = await login_history.list_recent(db, user.id, limit=safe_limit)
    return rows


@router.patch("/profile", response_model=UserOut)
async def update_profile(data: UpdateProfileIn, user: CurrentUser, db: DBDep) -> UserOut:
    if data.selected_currencies is not None:
        user = await auth_service.update_selected_currencies(
            db, user, data.selected_currencies
        )
    if data.analytics_currencies is not None:
        user = await auth_service.update_analytics_currencies(
            db, user, data.analytics_currencies
        )
    if data.base_currency is not None:
        user = await auth_service.update_base_currency(db, user, data.base_currency)
    return UserOut.model_validate(user)
```

Лістинг А.2 – Повна реалізація сесійного контуру auth-сервісу (backend/app/services/auth.py)

```
REFRESH_COOKIE_NAME = "montera_refresh"
REFRESH_COOKIE_PATH = "/api/auth"


def _set_refresh_cookie(response: Response, raw: str, expires_at: datetime) -> None:
    response.set_cookie(
        key=REFRESH_COOKIE_NAME,
        value=raw,
        expires=expires_at,
        path=REFRESH_COOKIE_PATH,
        httponly=True,
        secure=settings.cookie_secure,
        samesite="strict",
    )


def _clear_refresh_cookie(response: Response) -> None:
    response.delete_cookie(
        key=REFRESH_COOKIE_NAME,
        path=REFRESH_COOKIE_PATH,
        httponly=True,
        samesite="strict",
        secure=settings.cookie_secure,
    )
```

```
def _client_meta(request: Request) -> tuple[str | None, str | None]:
    ua = request.headers.get("user-agent")
    ua = ua[:512] if ua else None
    ip = request.client.host if request.client else None
    return ua, ip


async def register(db: AsyncSession, data: RegisterIn) -> User:
    user = User(
        email=data.email.lower(),
        password_hash=hash_password(data.password),
        base_currency=data.base_currency,
    )
    db.add(user)
    try:
        await db.commit()
    except IntegrityError as e:
        await db.rollback()
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT, detail="email already registered"
        ) from e
    await db.refresh(user)
    return user


async def login(
    db: AsyncSession, response: Response, request: Request, data: LoginIn
) -> TokenOut:
    email = data.email.lower()
    # Reject the request before doing any password work if this email is
    # currently inside an active lockout window.
    await login_lockout.check_lockout(db, email)

    result = await db.execute(select(User).where(User.email == email))
    user = result.scalar_one_or_none()
    if user is None or not verify_password(data.password, user.password_hash):
        await login_lockout.record_failure(db, email)
        # If the most recent failure crossed a group boundary the row
        # now has a fresh `locked_until`; surface it as 429 so the UI
        # can show a precise countdown rather than a generic 401.
        await login_lockout.check_lockout(db, email)
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED, detail="invalid credentials"
        )

    # Snapshot the failure counter *before* reset_lockout wipes it so
    # the login event row can record "you got it on the Nth try".
    pending_lockout = await login_lockout._get(db, email)
    failed_before = pending_lockout.failed_count if pending_lockout else 0
    await login_lockout.reset_lockout(db, email)

    raw, h, exp = generate_refresh_token()
    ua, ip = _client_meta(request)
    db.add(RefreshToken(user_id=user.id, token_hash=h, user_agent=ua, ip=ip, expires_at=exp))
    await db.commit()

    await login_history.record_login(
        db,
        user_id=user.id,
```

```
        method="password",
        request=request,
        failed_attempts_before=failed_before,
    )

    _set_refresh_cookie(response, raw, exp)
    access, expires_in = create_access_token(user.id)
    return TokenOut(access_token=access, expires_in=expires_in)


async def refresh(db: AsyncSession, response: Response, request: Request) -> TokenOut:
    raw = request.cookies.get(REFRESH_COOKIE_NAME)
    if not raw:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED, detail="no refresh token"
        )

    h = hash_refresh_token(raw)
    result = await db.execute(select(RefreshToken).where(RefreshToken.token_hash == h))
    rt = result.scalar_one_or_none()
    now = datetime.now(timezone.utc)
    if rt is None or rt.revoked_at is not None or rt.expires_at < now:
        _clear_refresh_cookie(response)
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED, detail="invalid refresh token"
        )

    rt.revoked_at = now
    new_raw, new_h, new_exp = generate_refresh_token()
    ua, ip = _client_meta(request)
    db.add(
        RefreshToken(
            user_id=rt.user_id, token_hash=new_h, user_agent=ua, ip=ip, expires_at=new_exp
        )
    )
    await db.commit()

    _set_refresh_cookie(response, new_raw, new_exp)
    access, expires_in = create_access_token(rt.user_id)
    return TokenOut(access_token=access, expires_in=expires_in)


async def logout(db: AsyncSession, response: Response, request: Request) -> None:
    raw = request.cookies.get(REFRESH_COOKIE_NAME)
    if raw:
        h = hash_refresh_token(raw)
        await db.execute(
            update(RefreshToken)
            .where(RefreshToken.token_hash == h, RefreshToken.revoked_at.is_(None))
            .values(revoked_at=datetime.now(timezone.utc))
        )
        await db.commit()
    _clear_refresh_cookie(response)
```

Лістинг А.3 – Повна реалізація транзакційного контуру (backend/app/services/transactions.py)

```
async def _own_wallet(db: AsyncSession, user: User, wallet_id: uuid.UUID) -> Wallet:
```

```
    result = await db.execute(
        select(Wallet).where(Wallet.id == wallet_id, Wallet.user_id == user.id)
    )
    wallet = result.scalar_one_or_none()
    if wallet is None:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="wallet not found")
    return wallet


async def _get_balance(
    db: AsyncSession, wallet_id: uuid.UUID, asset_id: uuid.UUID
) -> WalletBalance | None:
    result = await db.execute(
        select(WalletBalance).where(
            WalletBalance.wallet_id == wallet_id, WalletBalance.asset_id == asset_id
        )
    )
    return result.scalar_one_or_none()


async def _get_or_create_balance(
    db: AsyncSession, wallet_id: uuid.UUID, asset_id: uuid.UUID
) -> WalletBalance:
    bal = await _get_balance(db, wallet_id, asset_id)
    if bal is None:
        bal = WalletBalance(wallet_id=wallet_id, asset_id=asset_id, amount=Decimal("0"))
        db.add(bal)
        await db.flush()
    return bal


def _insufficient(available: Decimal, requested: Decimal, symbol: str | None = None) -> HTTPException:
    sym = f" {symbol}" if symbol else ""
    return HTTPException(
        status_code=status.HTTP_409_CONFLICT,
        detail=(
            f"Недостатньо коштів{sym}. Доступно: {available}, потрібно: {requested}"
        ),
    )


async def _apply_deposit(db: AsyncSession, t: Transaction, asset: Asset) -> None:
    bal = await _get_or_create_balance(db, t.wallet_id, t.asset_id)
    price = await prices_service.get_price_in_usdt(db, asset)
    bal.amount += t.amount
    if price is not None:
        bal.cost_basis_usdt = (bal.cost_basis_usdt or Decimal("0")) + t.amount * price


async def _apply_outflow(db: AsyncSession, t: Transaction) -> None:
    bal = await _get_balance(db, t.wallet_id, t.asset_id)
    if bal is None or bal.amount < t.amount:
        raise _insufficient(
            available=bal.amount if bal else Decimal("0"),
            requested=t.amount,
            symbol=t.asset.symbol if t.asset else None,
        )
    if bal.amount > 0 and bal.cost_basis_usdt is not None:
        ratio = t.amount / bal.amount
        bal.cost_basis_usdt = bal.cost_basis_usdt * (Decimal("1") - ratio)
```

```
    bal.amount -= t.amount


async def _apply_transfer(db: AsyncSession, t: Transaction) -> None:
    src = await _get_balance(db, t.wallet_id, t.asset_id)
    if src is None or src.amount < t.amount:
        raise _insufficient(
            available=src.amount if src else Decimal("0"),
            requested=t.amount,
            symbol=t.asset.symbol if t.asset else None,
        )
    ratio = t.amount / src.amount
    cost_moved = (src.cost_basis_usdt or Decimal("0")) * ratio
    src.amount -= t.amount
    if src.cost_basis_usdt is not None:
        src.cost_basis_usdt = src.cost_basis_usdt - cost_moved

    assert t.counterparty_wallet_id is not None
    tgt = await _get_or_create_balance(db, t.counterparty_wallet_id, t.asset_id)
    tgt.amount += t.amount
    if cost_moved > 0:
        tgt.cost_basis_usdt = (tgt.cost_basis_usdt or Decimal("0")) + cost_moved


async def _apply_convert(db: AsyncSession, t: Transaction) -> None:
    assert t.counterparty_asset_id is not None
    assert t.counterparty_amount is not None

    src = await _get_balance(db, t.wallet_id, t.asset_id)
    if src is None or src.amount < t.amount:
        raise _insufficient(
            available=src.amount if src else Decimal("0"),
            requested=t.amount,
            symbol=t.asset.symbol if t.asset else None,
        )
    ratio = t.amount / src.amount
    cost_moved = (src.cost_basis_usdt or Decimal("0")) * ratio
    src.amount -= t.amount
    if src.cost_basis_usdt is not None:
        src.cost_basis_usdt = src.cost_basis_usdt - cost_moved

    tgt = await _get_or_create_balance(db, t.wallet_id, t.counterparty_asset_id)
    tgt.amount += t.counterparty_amount
    if cost_moved > 0:
        tgt.cost_basis_usdt = (tgt.cost_basis_usdt or Decimal("0")) + cost_moved


async def create_transaction(
    db: AsyncSession, user: User, data: TransactionCreate
) -> Transaction:
    await _own_wallet(db, user, data.wallet_id)
    if data.counterparty_wallet_id is not None:
        await _own_wallet(db, user, data.counterparty_wallet_id)

    asset = await db.get(Asset, data.asset_id)
    if asset is None or not asset.is_active:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="asset not found")
    if data.counterparty_asset_id is not None:
        ct_asset = await db.get(Asset, data.counterparty_asset_id)
        if ct_asset is None or not ct_asset.is_active:
```

```
        raise HTTPException(
            status_code=status.HTTP_404_NOT_FOUND, detail="counterparty asset not found"
        )

# Outgoing operations (withdraw / transfer / convert) require the
# asset to actually exist on the source wallet – you can't move what
# you don't have. _apply_outflow checks the amount; here we make the
# "no balance row at all" case give a clearer 409 error.
if data.type in (
    TransactionType.WITHDRAW,
    TransactionType.OTHER,
    TransactionType.TRANSFER,
    TransactionType.CONVERT,
):
    bal = await _get_balance(db, data.wallet_id, data.asset_id)
    if bal is None or bal.amount <= 0:
        raise HTTPException(
            status_code=status.HTTP_409_CONFLICT,
            detail="this asset is not present in the source wallet",
        )

rate: Decimal | None = None
if data.type == TransactionType.CONVERT and data.counterparty_amount is not None:
    rate = data.counterparty_amount / data.amount

t = Transaction(
    user_id=user.id,
    wallet_id=data.wallet_id,
    type=data.type,
    label=data.label,
    asset_id=data.asset_id,
    amount=data.amount,
    counterparty_wallet_id=data.counterparty_wallet_id,
    counterparty_asset_id=data.counterparty_asset_id,
    counterparty_amount=data.counterparty_amount,
    rate=rate,
    occurred_at=data.occurred_at,
)
db.add(t)
await db.flush()

if data.type == TransactionType.DEPOSIT:
    await _apply_deposit(db, t, asset)
elif data.type in (TransactionType.WITHDRAW, TransactionType.OTHER):
    await _apply_outflow(db, t)
elif data.type == TransactionType.TRANSFER:
    await _apply_transfer(db, t)
    # Mirror as a DEPOSIT on the recipient wallet, linked back to
    # the transfer. _apply_transfer already updated both balances,
    # so the mirror just exists for listing purposes.
    mirror = Transaction(
        user_id=user.id,
        wallet_id=data.counterparty_wallet_id,  # type: ignore[arg-type]
        type=TransactionType.DEPOSIT,
        label=data.label,
        asset_id=data.asset_id,
        amount=data.amount,
        counterparty_wallet_id=data.wallet_id,
        counterparty_asset_id=None,
        counterparty_amount=None,
```

```
            rate=None,
            linked_transaction_id=t.id,
            occurred_at=data.occurred_at,
        )
        db.add(mirror)
        await db.flush()
    elif data.type == TransactionType.CONVERT:
        await _apply_convert(db, t)

    if data.type in (TransactionType.DEPOSIT, TransactionType.TRANSFER):
        await analytics_service.ensure_chart_for_asset(db, user.id, data.asset_id)
    if data.type == TransactionType.CONVERT and data.counterparty_asset_id is not None:
        await analytics_service.ensure_chart_for_asset(
            db, user.id, data.counterparty_asset_id
        )

    await db.commit()
    await db.refresh(t)
    return t
```

Лістинг А.4 – Повна реалізація знеособлення контексту агента (backend/app/services/agent_privacy.py)

```
from __future__ import annotations

import re
import unicodedata
from typing import Any


def _strip_diacritics(s: str) -> str:
    """Normalise text for case-insensitive matching across Cyrillic,
    Latin, and accented forms. 'Бінанс' → 'binans', 'Binance Spot' →
    'binance spot'."""
    nfkd = unicodedata.normalize("NFKD", s)
    no_marks = "".join(c for c in nfkd if not unicodedata.combining(c))
    return no_marks.casefold()


def build_alias_map(wallet_names: list[str]) -> dict[str, str]:
    """Original wallet name → alias ('Wallet 1', etc). Order is the
    order in which names appear in `wallet_names`. Duplicates collapse
    to a single alias."""
    out: dict[str, str] = {}
    counter = 0
    seen: set[str] = set()
    for name in wallet_names:
        if not name or name in seen:
            continue
        seen.add(name)
        counter += 1
        out[name] = f"Wallet {counter}"
    return out


def reverse_alias_map(alias_map: dict[str, str]) -> dict[str, str]:
    return {alias: original for original, alias in alias_map.items()}
```

```
def sanitize_snapshot(
    snapshot: dict[str, Any], alias_map: dict[str, str]
) -> dict[str, Any]:
    """Return a deep-copied, sanitised version of the portfolio snapshot.

    Drops any fields that could identify the user. Keeps every financial
    value the agent might reason about (amounts, dates, prices, types).
    """
    if not snapshot:
        return {}

    out: dict[str, Any] = {
        "snapshot_at": snapshot.get("snapshot_at"),
        "portfolio": dict(snapshot.get("portfolio") or {}),
        "positions": [
            {
                "symbol": p.get("symbol"),
                "name": p.get("name"),
                "amount": p.get("amount"),
                "price_usdt": p.get("price_usdt"),
                "value_usdt": p.get("value_usdt"),
                "pnl_usdt": p.get("pnl_usdt"),
            }
            for p in snapshot.get("positions") or []
        ],
        "wallets": [
            {
                "name": alias_map.get(w.get("name", ""), w.get("name", "")),
                "value_usdt": w.get("value_usdt"),
                "pnl_usdt": w.get("pnl_usdt"),
            }
            for w in snapshot.get("wallets") or []
        ],
        "recent_transactions": [
            {
                "type": t.get("type"),
                "asset": t.get("asset"),
                "amount": t.get("amount"),
                "counterparty_asset": t.get("counterparty_asset"),
                "counterparty_amount": t.get("counterparty_amount"),
                # Drop free-form labels; they routinely carry PII.
                "occurred_at": t.get("occurred_at"),
            }
            for t in snapshot.get("recent_transactions") or []
        ],
    }
    return out


def _build_pattern(names: list[str]) -> re.Pattern[str] | None:
    """Build a single regex that matches any of `names`, accent- and
    case-insensitive. Returns None for an empty list (saves a no-op
    `re.sub` call later)."""
    if not names:
        return None
    # Sort longest first so "Binance Spot" matches before "Binance".
    names_sorted = sorted(set(names), key=len, reverse=True)
    parts = [re.escape(n) for n in names_sorted]
    # We can't make Python's regex truly accent-insensitive, so we
```

```
    # compile two alternatives: the original AND its ASCII-folded form.
    folded = sorted(
        {_strip_diacritics(n) for n in names_sorted}, key=len, reverse=True
    )
    parts.extend(re.escape(p) for p in folded if p)
    return re.compile("|".join(parts), re.IGNORECASE)


def rewrite_user_text(text: str, alias_map: dict[str, str]) -> str:
    """Replace any mention of a real wallet name in `text` with its
    alias. Used on the user's chat input before it's sent to the LLM."""
    if not text or not alias_map:
        return text
    pattern = _build_pattern(list(alias_map.keys()))
    if pattern is None:
        return text

    folded_lookup = {_strip_diacritics(orig): alias for orig, alias in alias_map.items()}
    direct_lookup = {orig.casefold(): alias for orig, alias in alias_map.items()}

    def _sub(m: re.Match[str]) -> str:
        token = m.group(0)
        return (
            direct_lookup.get(token.casefold())
            or folded_lookup.get(_strip_diacritics(token))
            or token
        )

    return pattern.sub(_sub, text)


def restore_assistant_text(text: str, reverse_map: dict[str, str]) -> str:
    """Replace every alias the assistant produced back to the original
    name the user knows. Aliases are unique tokens like 'Wallet 1' so
    this is safe to do globally without word-boundary surprises."""
    if not text or not reverse_map:
        return text
    # Sort longest first to avoid 'Wallet 1' matching inside 'Wallet 10'.
    aliases_sorted = sorted(reverse_map.keys(), key=len, reverse=True)
    pattern = re.compile("|".join(re.escape(a) for a in aliases_sorted))
    return pattern.sub(lambda m: reverse_map.get(m.group(0), m.group(0)), text)
```

Лістинг А.5 – Повна реалізація клієнтського API-сесійного шару (frontend/src/api/client.ts)

```
import axios, { AxiosError, AxiosRequestConfig } from 'axios'
import { useAuthStore } from '@/store/auth'

export const api = axios.create({
  baseURL: '/api',
  withCredentials: true,
})

api.interceptors.request.use((config) => {
  const token = useAuthStore.getState().accessToken
  if (token) {
    config.headers.Authorization = `Bearer ${token}`
  }
```

```
  return config
})

let refreshPromise: Promise<string | null> | null = null

async function refreshAccessToken(): Promise<string | null> {
  if (!refreshPromise) {
    refreshPromise = (async () => {
      try {
        const res = await axios.post<{ access_token: string }>(
          '/api/auth/refresh',
          {},
          { withCredentials: true },
        )
        const newToken = res.data.access_token
        useAuthStore.getState().setAccessToken(newToken)
        return newToken
      } catch {
        useAuthStore.getState().clear()
        return null
      } finally {
        refreshPromise = null
      }
    })()
  }
  return refreshPromise
}

api.interceptors.response.use(
  (r) => r,
  async (error: AxiosError) => {
    const original = error.config as AxiosRequestConfig & { _retry?: boolean }
    if (
      error.response?.status === 401 &&
      original &&
      !original._retry &&
      !original.url?.includes('/auth/refresh') &&
      !original.url?.includes('/auth/login')
    ) {
      original._retry = true
      const newToken = await refreshAccessToken()
      if (newToken) {
        original.headers = {
          ...(original.headers ?? {}),
          Authorization: `Bearer ${newToken}`,
        }
        return api.request(original)
      }
    }
    return Promise.reject(error)
  },
)

export async function tryInitialRefresh(): Promise<boolean> {
  const token = await refreshAccessToken()
  return token !== null
}
```

Лістинг А.6 – Повна реалізація захищеного маршруту (frontend/src/components/ProtectedRoute.tsx)

```
import { useEffect } from 'react'
import { Navigate, Outlet } from 'react-router-dom'
import { useAuthStore } from '@/store/auth'
import { tryInitialRefresh } from '@/api/client'
import { fetchMe } from '@/api/auth'

export default function ProtectedRoute() {
  const status = useAuthStore((s) => s.status)
  const setStatus = useAuthStore((s) => s.setStatus)
  const setUser = useAuthStore((s) => s.setUser)
  const accessToken = useAuthStore((s) => s.accessToken)

  useEffect(() => {
    if (status !== 'idle') return
    setStatus('loading')
    ;(async () => {
      const ok = await tryInitialRefresh()
      if (!ok) {
        setStatus('unauthenticated')
        return
      }
      try {
        const user = await fetchMe()
        setUser(user)
        setStatus('authenticated')
      } catch {
        setStatus('unauthenticated')
      }
    })()
  }, [status, setStatus, setUser])

  useEffect(() => {
    if (status === 'authenticated' && accessToken && !useAuthStore.getState().user) {
      fetchMe()
        .then((u) => setUser(u))
        .catch(() => {})
    }
  }, [status, accessToken, setUser])

  if (status === 'idle' || status === 'loading') {
    return <FullScreenLogoLoader />
  }
  if (status === 'unauthenticated') {
    return <Navigate to="/login" replace />
  }
  return <Outlet />
}
```

ДОДАТОК Б
(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

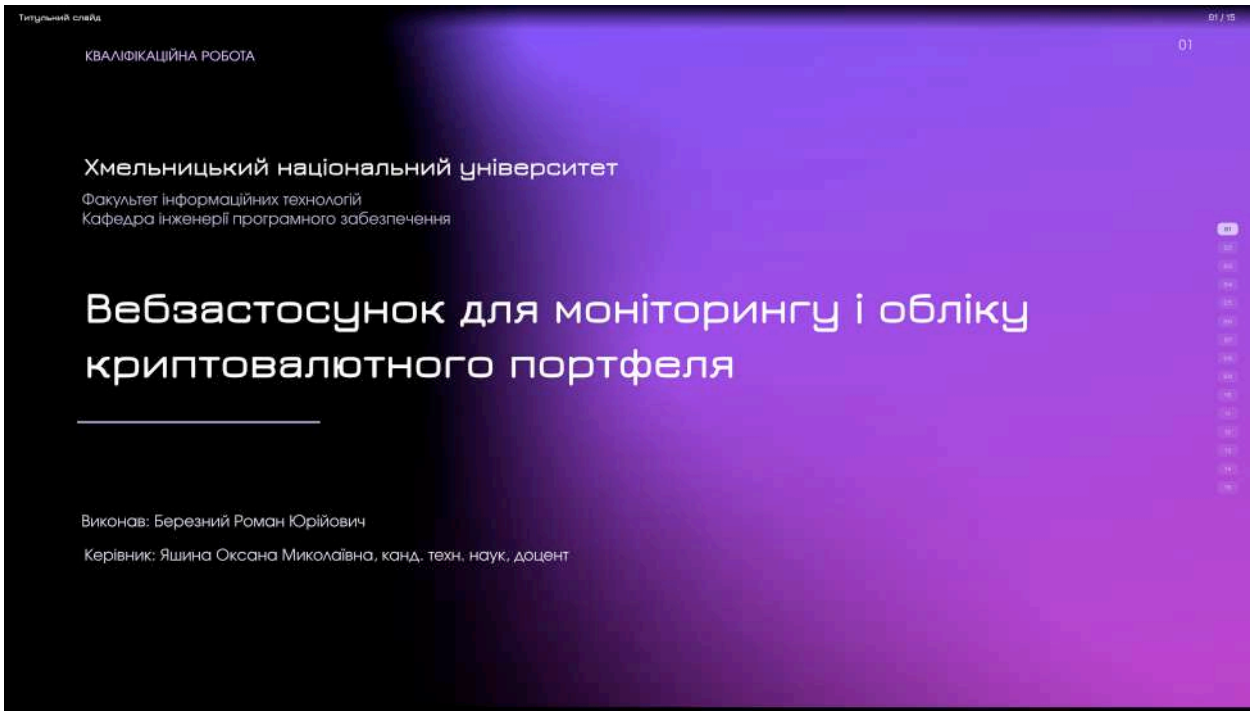


Рисунок Б.1 – «Слайд: Вступ»

Рисунок Б.2 –«Слайд: Актуальність теми»

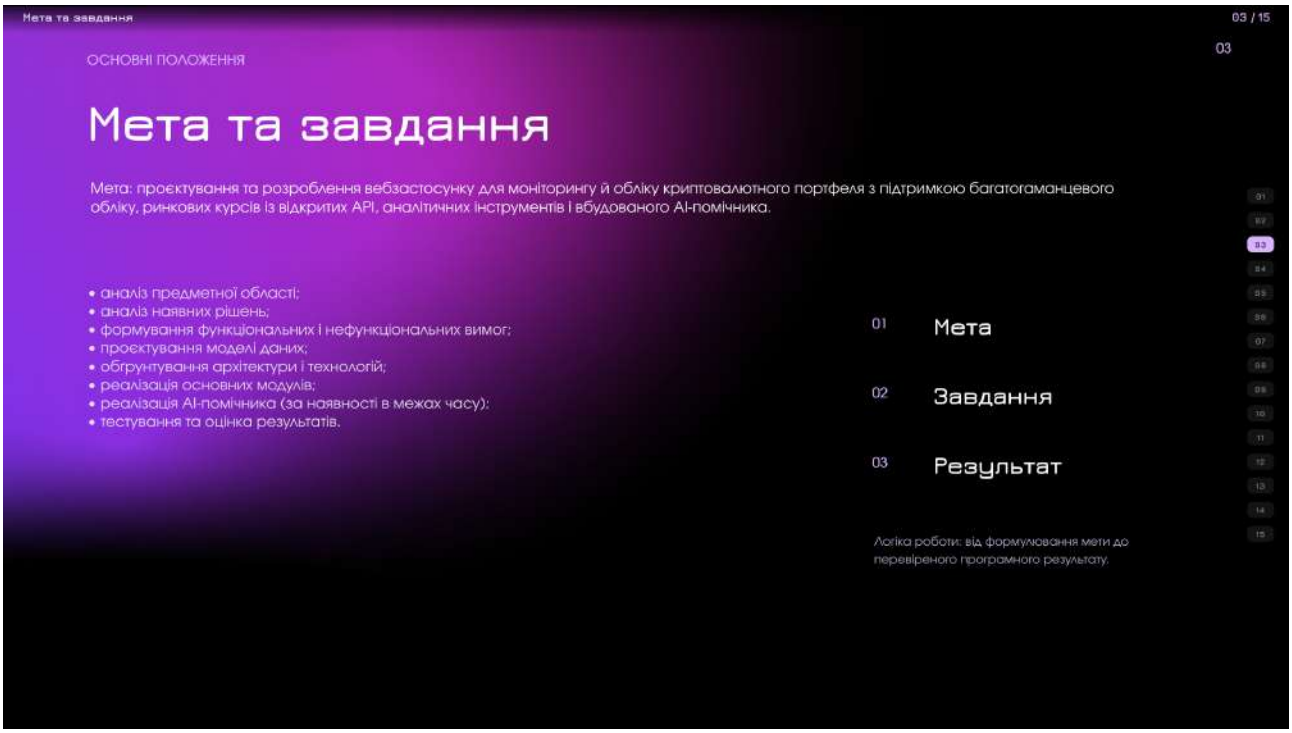


Рисунок Б.3 – «Слайд: Мета та завдання»

Рисунок Б.4 – «Слайд: Дослідження предметної області»

Рисунок Б.5 – «Слайд: Аналіз наявного ПЗ»

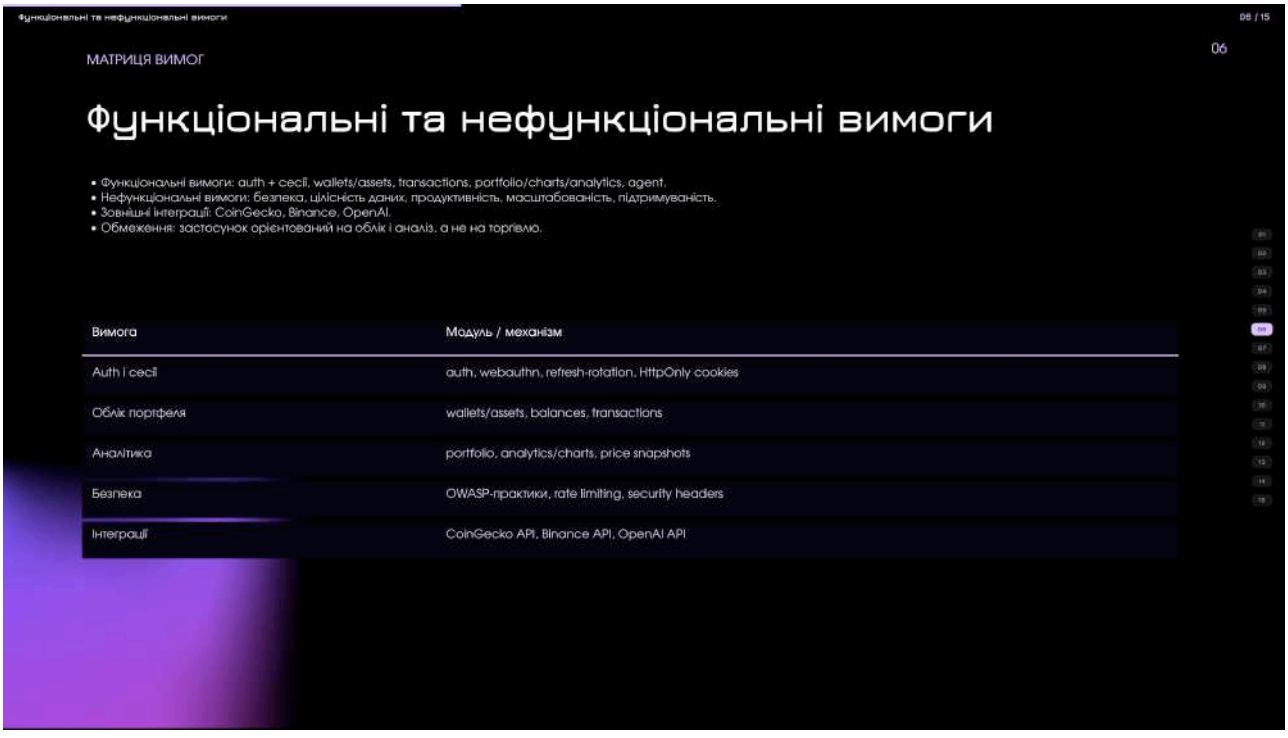


Рисунок Б.6 – «Слайд: Функціональні та нефункціональні вимоги»

Рисунок Б.7 – «Слайд: Вибір архітектури»

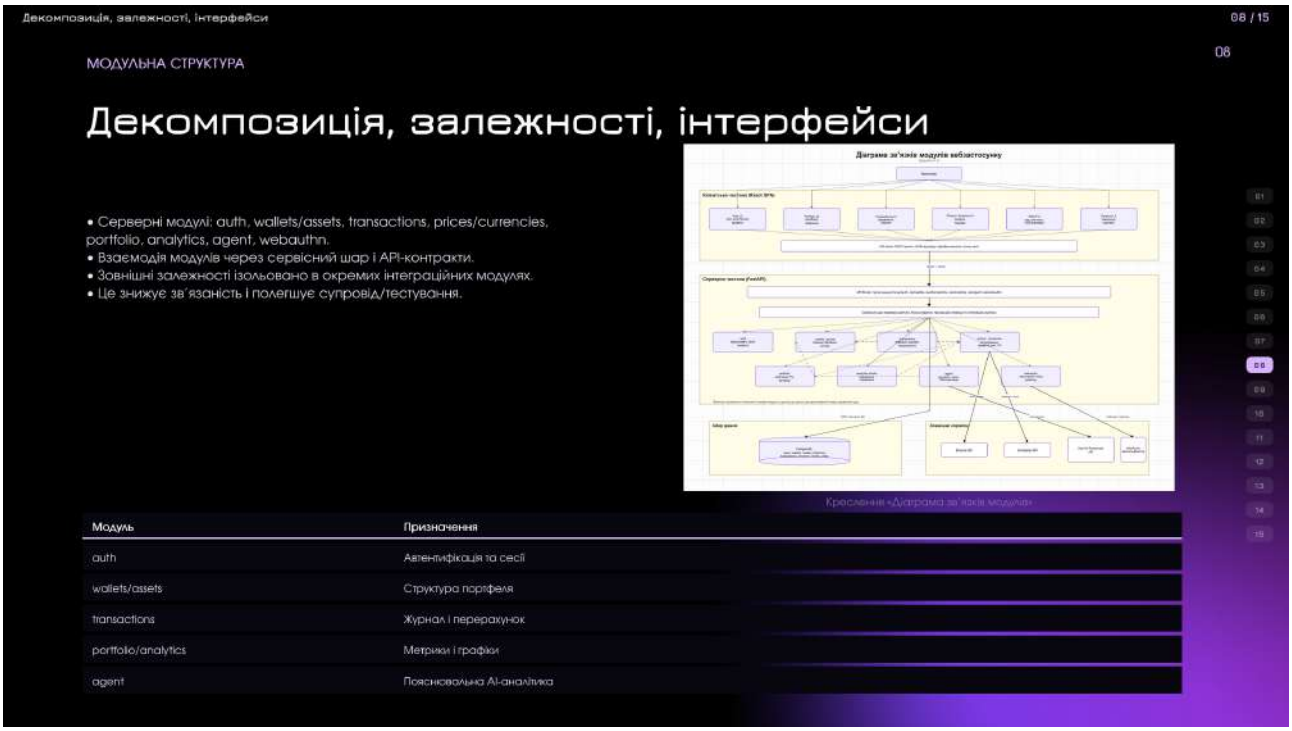


Рисунок Б.8 – «Слайд: Декомпозиція, залежності, інтерфейси»

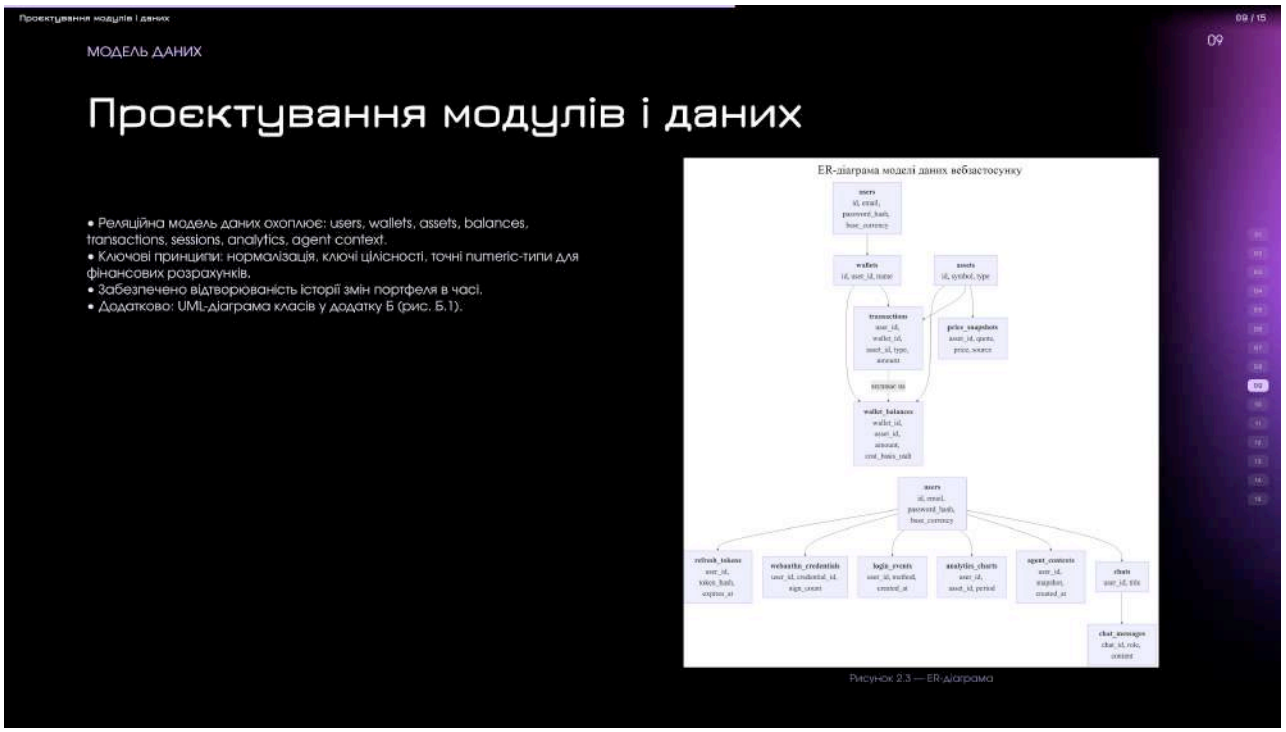


Рисунок Б.9 – «Слайд: Проєктування модулів і даних»

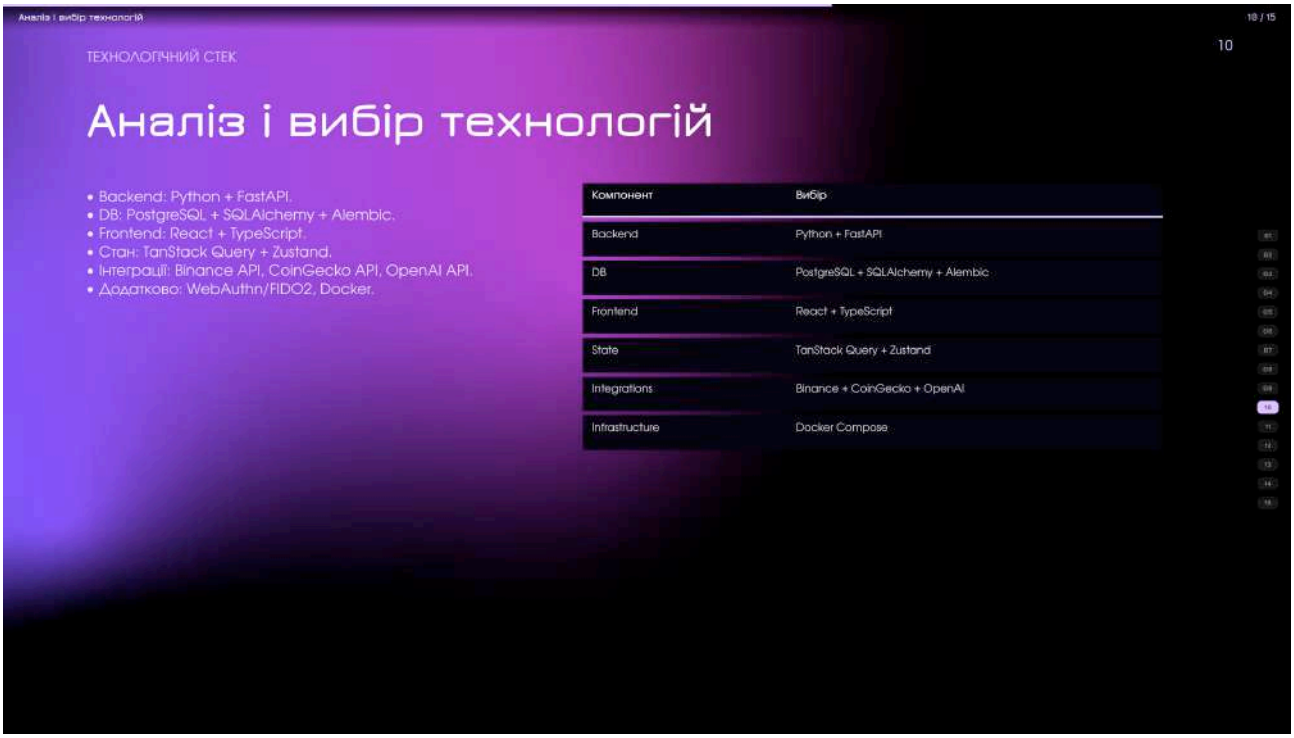


Рисунок Б.10 – «Слайд: Аналіз і вибір технологій»

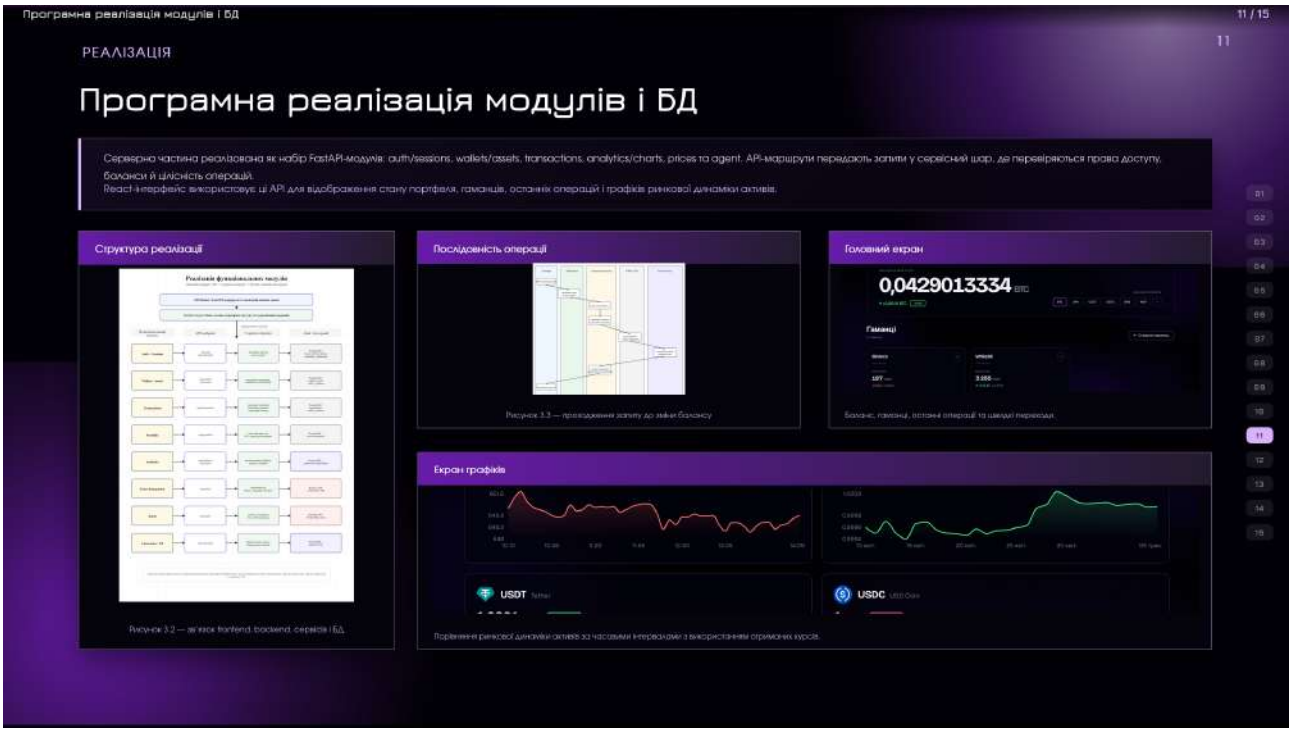


Рисунок Б.11 – «Слайд: Програмна реалізація модулів і БД»

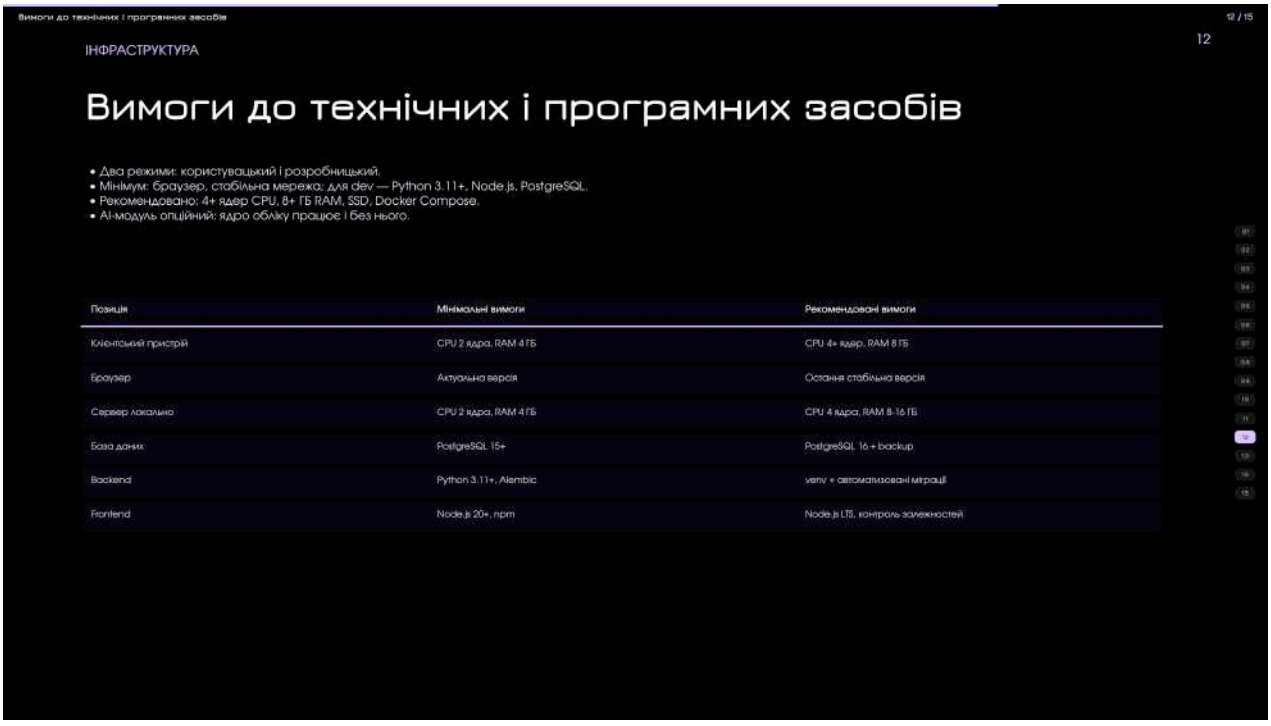


Рисунок Б.12 – «Слайд: Вимоги до технічних і програмних засобів»

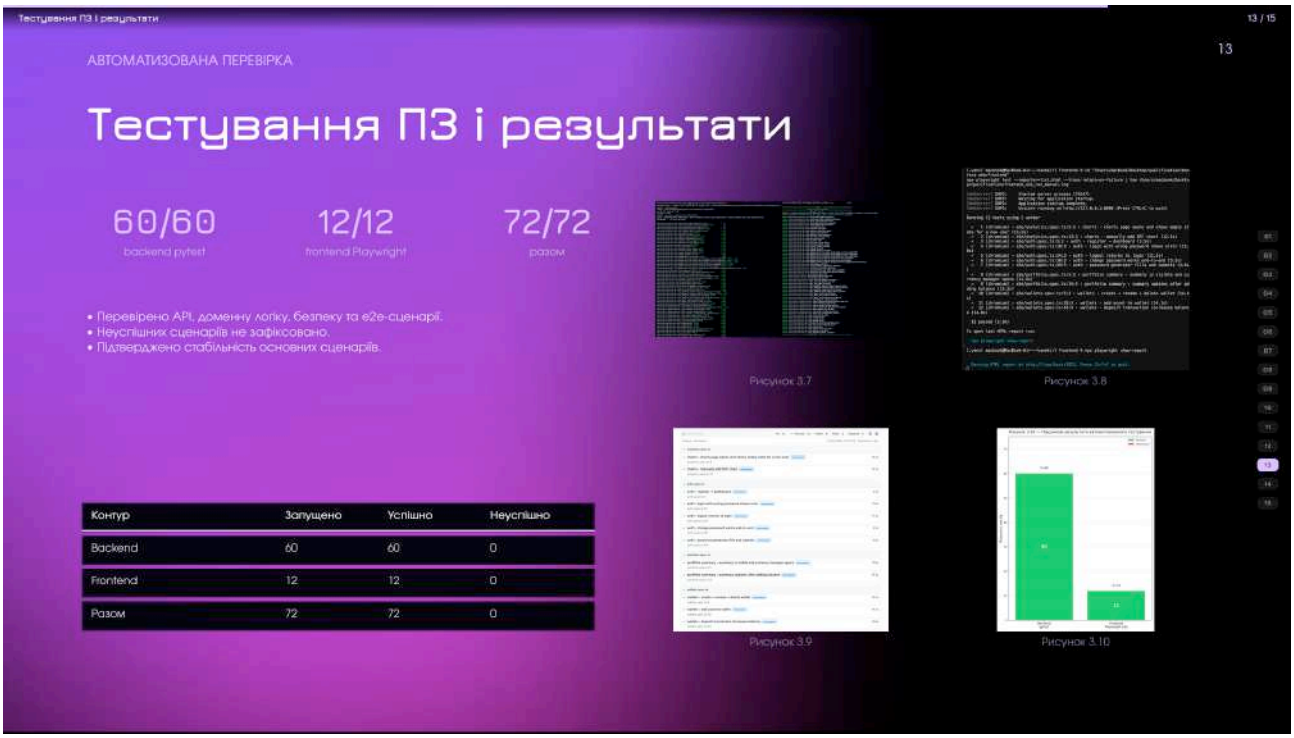


Рисунок Б.13 – «Слайд: Тестування ПЗ і результати»

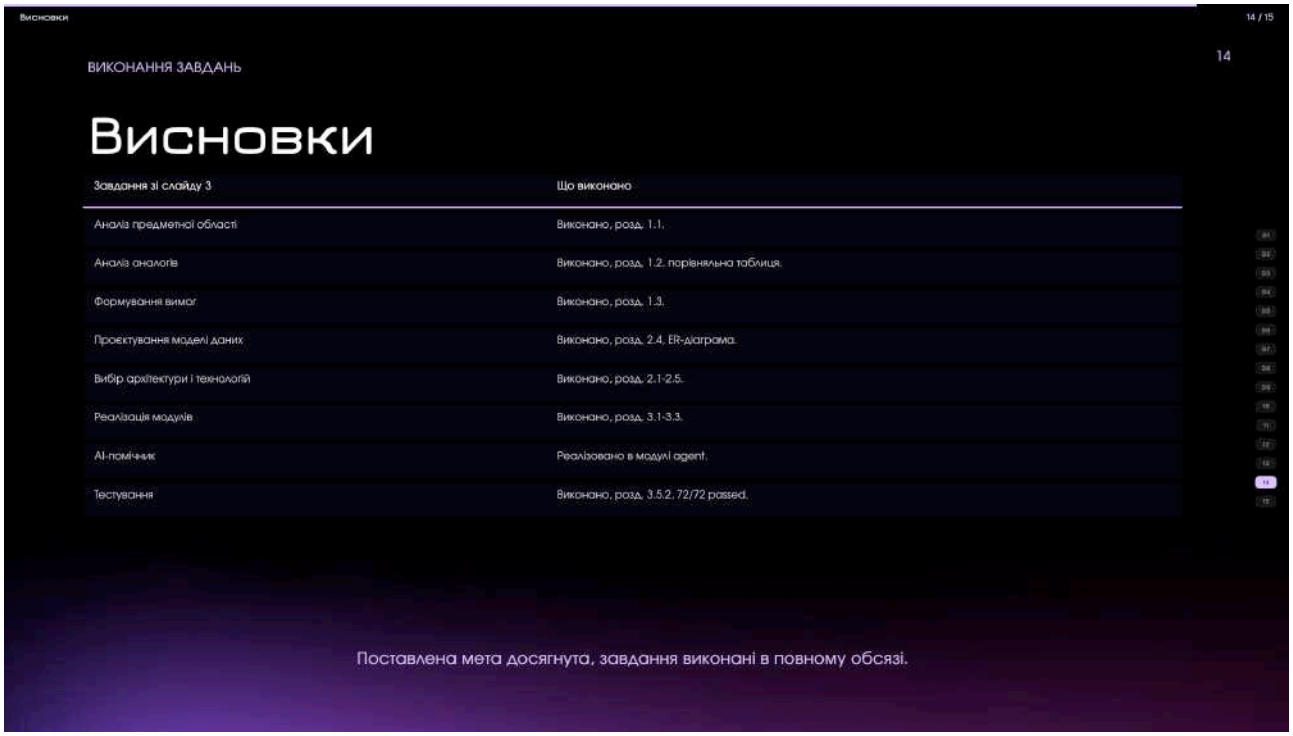


Рисунок Б.14 – «Слайд: Висновки»

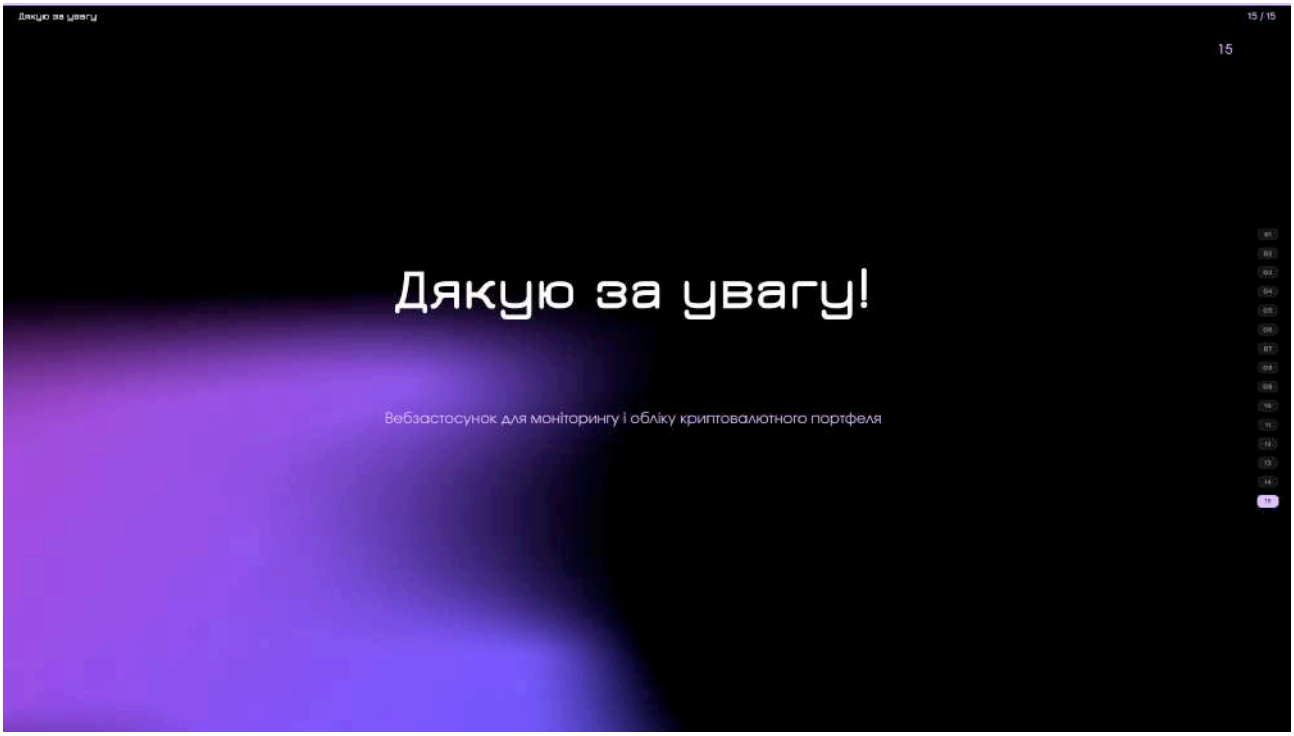
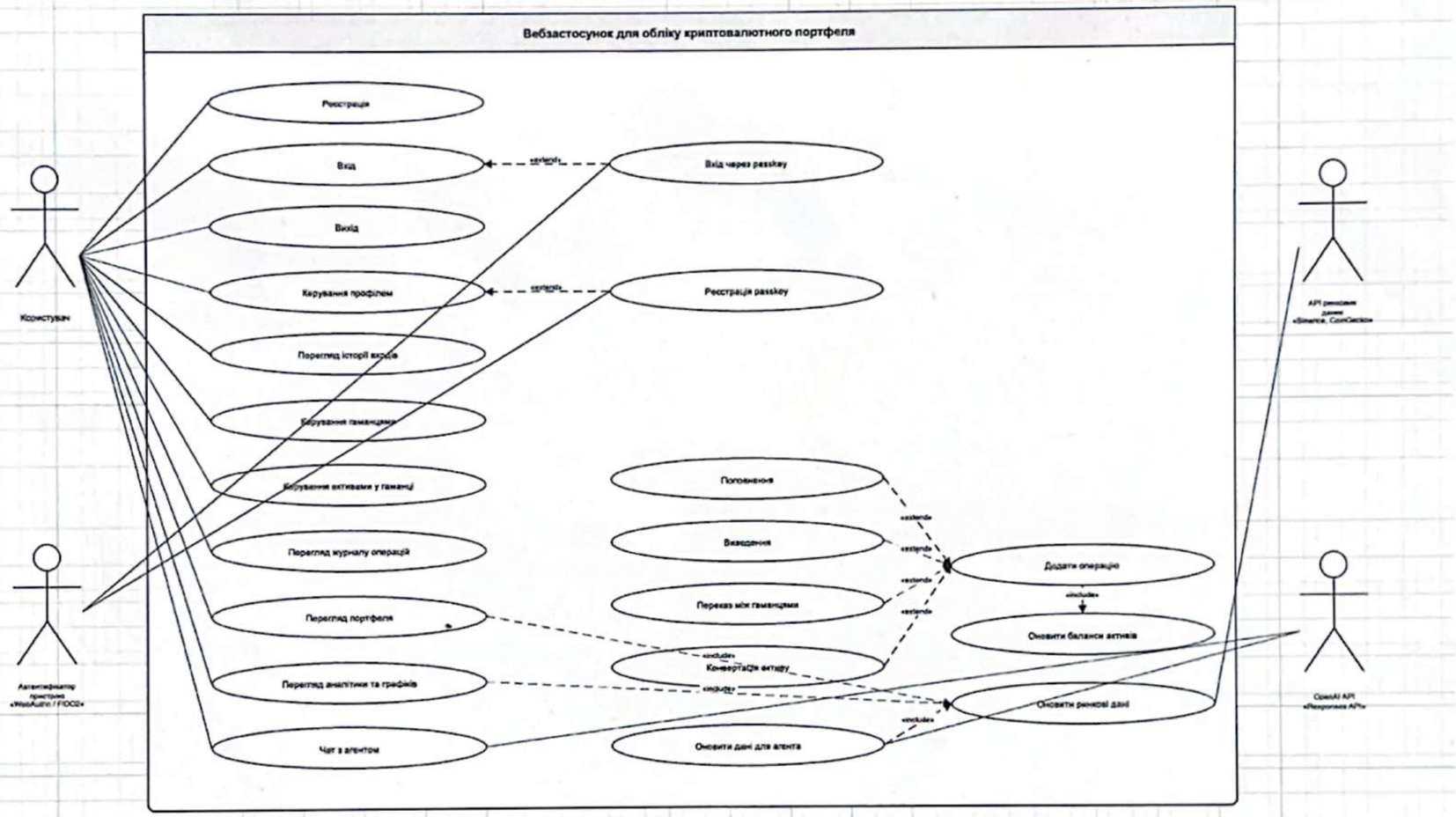


Рисунок Б.15 – «Слайд: Дякую за увагу!»

ГРАФІЧНА ЧАСТИНА

UML-діаграма варіантів використання вебзастосунку

Додаток А.1



Умовні позначення

Актор — користувач або зовнішня система.

Варіант використання — функціональний сценарій системи.

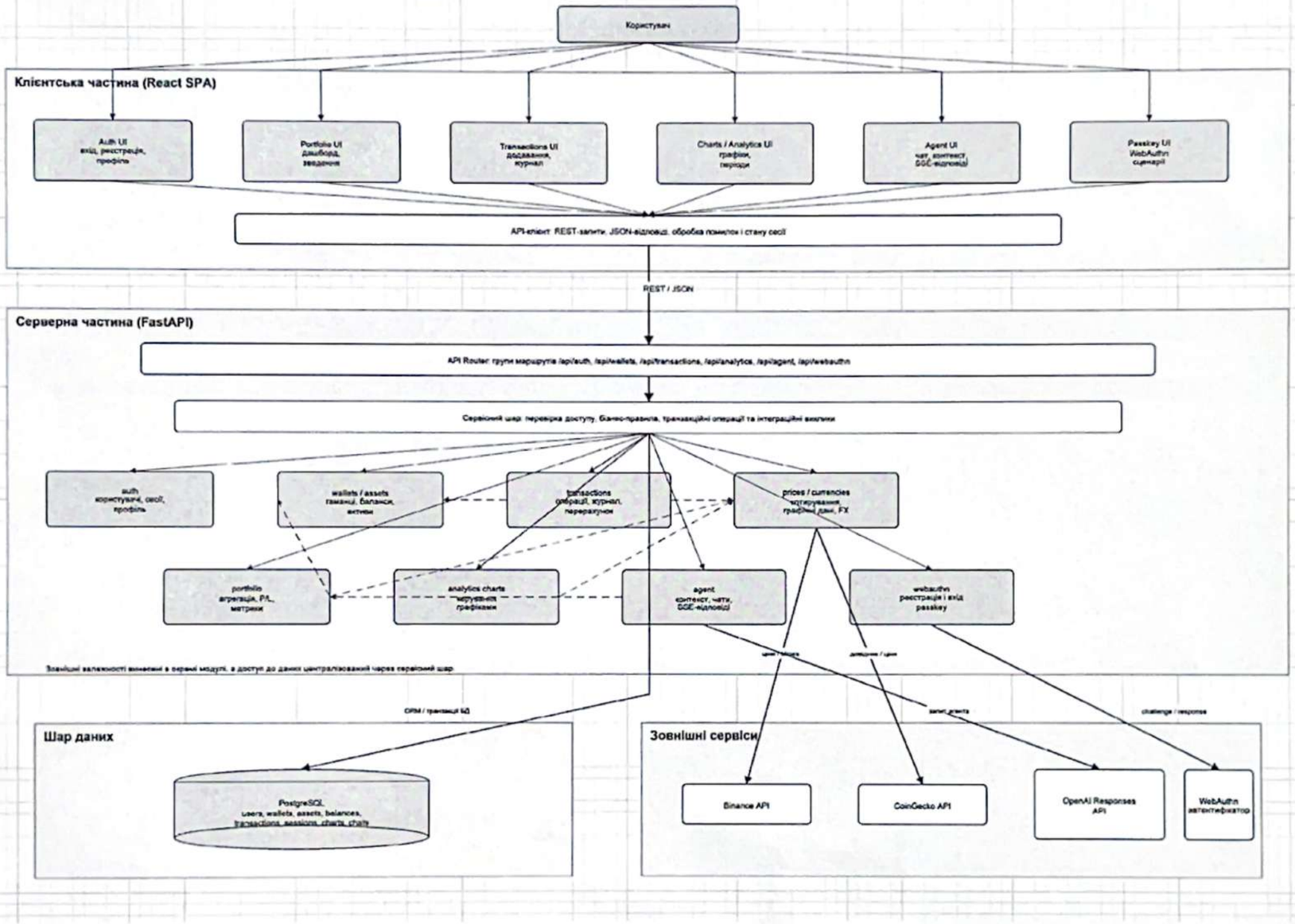
Асоціація актора з варіантом використання.

«include» / «extend» — залежність між сценаріями.

					КвРІПЗ.220194.01.01.Е8		
					Вебзастосунок моніторинку і обліку криптовалютного портфеля	Літера	Маса	Масштаб
Зм.	Арк.	№ докум.	Підпис	Дата				
Розробив		Березний Р. Ю.		26.05				
Керівник		Яшина О.М.		26.05	UML-діаграма варіантів використання	Арк.1	Аркушів 3	
Консульт.								
Н. Контр.		Праворська Н.І.		26.05		ХНУ, ІПЗ-22-1		
Зав. каф.		Бедратюк Л. П.		26.05				

Діаграма зв'язків модулів вебзастосунку

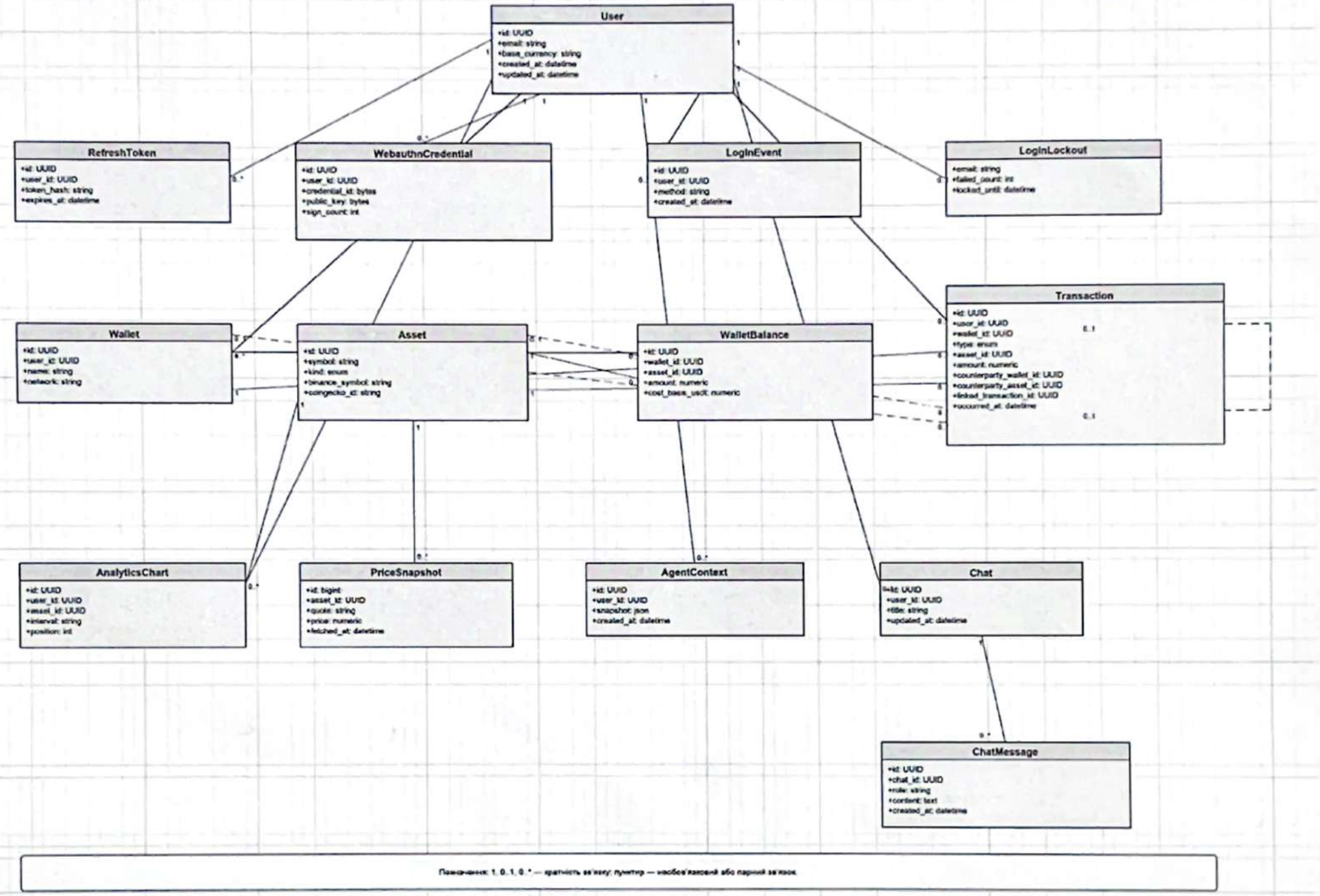
Додаток А.2



Зм.	Арк.	№ докум.	Підпис	Дата	КвРІПЗ. 220194.01.01.Е8		
Розробив		Березний Р. Ю.		26.05	Вебзастосунок моніторингу і обліку криптовалютного портфеля	Літера	Маса	Масштаб
Керівник		Яшина О.М.		26.05				
Консульт.					Діаграма зв'язків модулів	Арк.2	Аркушів 3	
Н. Контр.		Праворська Н.І.		26.05		ХНУ, ІПЗ-22-1		
Зав. каф		Бедратюк Л. П.		26.05				

UML-діаграма класів серверної частини вебзастосунку

Додаток А.3



					КвРІПЗ.220194.01.01.Е8	
Зм.	Арк.	№ докум.	Підпис	Дата	Вебзастосунок для моніторингу і обліку криптовалютного портфеля	Літера / Маса / Масштаб
Розробив		Березний Р. Ю.				
Керівник		Яшина О.М.			UML-діаграма класів серверної частини	Арк.3 / Аркушів 3
Консульт.						
Н. Контр.		Праворська Н.				ХНУ, ІПЗ-22-1
Зав. каф.		Бедратюк Л. П.				

СУПРОВІДНІ ДОКУМЕНТИ

Завідувачу кафедри інженерії програмного
забезпечення проф. Леоніду БЕДРАТЮКУ
здобувача вищої освіти
Березного Романа Юрійовича
факультет ІТ, IVкурс, група ІПЗ-22-1

ЗАЯВА

З правилами чинного Положення про систему забезпечення академічної доброчесності в Хмельницькому національному університеті, згідно з яким виявлення академічного плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на наявність академічного плагіату оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії Хмельницького національного університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-обчислювального комплексу StrikePlagiarism та/або програмно-технічного засобу AntiPlagiarism і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення текстових збігів у роботах.

Робота надається для перевірки в електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

01.05.2026р.
дата

підпис

Anti-Plagiarism (http://ap.km.ua) v-16.718

Максимальне співпадіння з одним документом 2.0%

Словники перевірки: UA, US, RU. **Помилок в документах: 21%**

ID: 272147 Назва: БКР Вебдастокунок для моніторингу і обліку криптовалютного портфеля Додано в БД: 2026-05-25 Автора: Роман БЕРЕЗНИЙ Керівник: канд. техн. наук, доцент Оксана ЯШИНА Консультанти: Опонента:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	105857	715	4819 (5%)	61 (9%)

Джерело плагіату

ID	Опис	Накладість плагіату в документі	
		Символи	Лексеми

Протокол аналізу звіту подібності науковим керівником

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Роман БЕРЕЗНИЙ

Співавтор:

Назва: Вебзастосунок для моніторингу і обліку криптовалютного портфеля

Науковий керівник: канд. техн. наук, доцент Оксана ЯШИНА

Підрозділ: Кафедра інженерії програмного забезпечення

Коефіцієнт подібності 1:4.2%

Коефіцієнт подібності 2:1.37%

Мікропробіли: 48

Заміна букв: 0

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2026-05-25 11:22:29.0

Після аналізу Звіту подібності констатую наступне:

☑ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

Дата 25.05.25 експерт

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Березний Роман Юрійович

Тема Вебзастосунок для моніторингу і обліку криптовалютного портфеля

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3; кількість сторінок записки 79

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих вебзастосунків на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого створено програмне забезпечення. Також було проведено тестування вебзастосунку, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації.

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовано актуальність теми, визначено мету, об'єкт, предмет і завдання кваліфікаційної роботи. У першому розділі проведено аналіз предметної області обліку криптовалютного портфеля, розглянуто наявні програмні рішення, визначено їх переваги й обмеження, а також сформовано функціональні та нефункціональні вимоги до вебзастосунку. У другому розділі виконано проєктування вебзастосунку: обґрунтовано вибір клієнт-серверної архітектури, проведено декомпозицію системи на модулі, спроєктовано модель даних, визначено підходи до безпеки, керування сесіями та взаємодії із зовнішніми сервісами. У третьому розділі описано особливості програмної реалізації основних функціональних модулів, зокрема автентифікації, гаманців, активів, транзакцій, аналітики, ринкових даних та інтелектуального помічника. Також визначено вимоги до технічних і програмних засобів, виконано тестування клієнтської та серверної частин, за результатами якого підтверджено коректність роботи основних сценаріїв вебзастосунку.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки облік криптовалютного портфеля потребує зручного інструменту для ведення гаманців, фіксації транзакцій, перегляду ринкових курсів та аналізу динаміки активів. Позитивною стороною роботи є комплексний підхід до розроблення вебзастосунку: реалізовано клієнтську та серверну частини, модель даних, модулі автентифікації, керування гаманцями, активами, транзакціями, аналітикою та інтелектуальним помічником. У роботі застосовано сучасні підходи до побудови вебзастосунків, організації безпечної роботи із сесіями, інтеграції із зовнішніми сервісами та візуалізації даних. Також позитивним є проведення тестування, що підтверджує

коректність роботи основних функціональних сценаріїв системи.

5. Негативні сторони роботи У роботі основну увагу приділено реалізації базових функцій обліку криптовалютного портфеля, тому окремі можливості можуть бути розширені в подальших версіях. Зокрема, доцільно додати глибшу персоналізацію аналітики, розширені фільтри для транзакцій і гаманців, а також можливість формування детальніших звітів за вибраний період. Крім того, перспективним є розширення інтеграцій із зовнішніми сервісами для автоматизованого імпорту операцій користувача.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді схем, діаграм, рисунків інтерфейсу і графічних матеріалів. Наведені матеріали відображають архітектуру вебзастосунку, декомпозицію системи, модель даних, логіку роботи модулів та результати тестування. Пояснювальна записка структурована відповідно до змісту роботи й оформлена згідно з вимогами чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки викладено структуровано, послідовно та зрозуміло, що дозволяє чітко простежити етапи аналізу, проєктування, програмної реалізації й тестування вебзастосунку. Робота має практичне спрямування, оскільки результатом є програмний продукт для моніторингу та обліку криптовалютного портфеля. Графічні матеріали доповнюють зміст роботи й наочно відображають архітектуру, модульну структуру, модель даних та особливості функціонування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Дяків Володимир Миколайович
студент гр. Кібербезпека

"01" червня 2026 р.

(підпис)

(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів перевірки роботи, продукованими програмно-технічним засобом (ами), на наявність текстових збігів.
Назва кваліфікаційної роботи: «Вебзастосунок для моніторингу і обліку криптовалютного портфеля»
Автор: Березний Роман Юрійович
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Спеціальність: 121 – Інженерія програмного забезпечення
Науковий керівник: Яшина Оксана Миколаївна, кандидат технічних наук, доцент
Після аналізу звіту/звітів зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається до захисту.	**відповідає**
2	Виявлені запозичення не є академічним плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована.	
3	Виявлені запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Виявлені запозичення частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнуті. Робота може бути допущена до захисту після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття текстових запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системою перевірки на плагіат виявлено схожість з окремими документами переважно в частині загальновживаних обов'язкових словосполучень у стандартних бланках, структурі змісту, назвах розділів і підрозділів, рамках форм, а також у назвах та URL-адресах публікацій переліку джерел посилання;

2) запозичення, виявлені у тексті роботи, мають фрагментарний характер і не впливають на самостійність виконання кваліфікаційної роботи.

Максимальний обсяг запозичень, визначений системою Anti-Plagiarism, складає 2.0% з одного джерела. Загальна сумарна подібність у базі даних складає 5% за символами та 9% за лексемами.

Крім того, за результатами додаткового аналізу коефіцієнт подібності 1 становить 4.2%, коефіцієнт подібності 2 – 1.37%. Заміни букв, маніпуляцій з інтервалами та зайвих білих знаків не виявлено. Виявлені мікропробіли у кількості 48 мають технічний характер і не свідчать про навмисне спотворення тексту. З урахуванням наведених обґрунтувань, результати перевірки відповідають характеру теми і свідчать на користь самостійності виконання кваліфікаційної роботи.

Дата 26.05.2026р.

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ЯШИНА