**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформаційних систем та технологій**

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____2025 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою «Інформаційні управляючі системи та технології»**
**спеціальності 126 «Інформаційні системи та технології»**
**на тему: «Вебзастосунок з використанням технологій парсингу інтернет-ресурсів для відстеження курсів криптовалют»**

Виконав:
студент IV курсу, групи ІС-12
Кудінов Михайло Сергійович _____

Керівник:
Доцент каф. ІСТ, к.т.н, доцент,
Деведжіогуллари Алла Вікторівна _____

Рецензент:
Доцент кафедри ОТ, к.т.н., доцент,
Роковий Олександр Петрович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2025 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра інформаційних систем та технологій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

**ЗАВДАННЯ**
**на дипломний проєкт студенту**
**Кудінову Михайлу Сергійовичу**

1. Тема проєкту «Вебзастосунок з використанням технологій парсингу інтернет-ресурсів для відстеження курсів криптовалют», керівник проєкту Деведжіогулари Алла Вікторівна, к.т.н., доцент каф ІСТ, затверджені наказом по університету від «23» травня 2025 р. № 1706-с

2. Термін подання студентом проєкту: 9 червня 2025 р.

3. Вихідні дані до проєкту: мова програмування JavaScript, редактор коду Visual Studio Code, фреймворк React,

4. Зміст пояснювальної записки:

1. *Опис предметної області*
2. *Аналіз існуючих рішень*
3. *Формування вимог до системи*
4. *Вибір технологій розробки*
5. *Розробка інформаційної системи*
6. *Математичне забезпечення*
7. *Тестування системи*

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо)

1. *Діаграма компонентів*
2. *Діаграма послідовностей*
3. *Діаграма потоків даних*
4. *Діаграма активностей*

6. Дата видачі завдання: 7 березня 2025 р.

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 17 квітня 2025 | |
| 2 | Огляд наявних аналогів | 19 квітня 2025 | |
| 3 | Формування цілей та задач розробки | 23 квітня 2025 | |
| 4 | Розробка веб-інтерфейсу | 30 квітня 2025 | |
| 5 | Розробка програмного рішення для торгових сигналів | 7 травня 2025 | |
| 6 | Опис математичного забезпечення | 10 травня 2025 | |
| 7 | Тестування готового проєкту | 14 травня 2025 | |
| 8 | Подання ДП на попередній захист | | |
| 9 | Подання ДП на основний захист | | |

Студент Михайло КУДІНОВ

Керівник Алла ДЕВЕДЖІОГУЛЛАРИ

# АНОТАЦІЯ

Вебзастосунок з використанням технологій парсингу інтернет-ресурсів для відстеження курсів криптовалют.

Проєкт містить 75 с. тексту, 8 рисунків, 7 таблиць, посилання на 20 літературні джерела, додатки та 4 конструкторських документів.

API, ВЕБЗАСТОСУНОК, КРИПТОВАЛЮТА, ПАРСИНГ, СИГНАЛИ.

Об’єктом розробки є вебзастосунок з використанням технологій парсингу інтернет-ресурсів для відстеження курсів криптовалют.

Мета розробки — спрощення процесу збору історичних даних про ціни закриття криптовалют та формування рекомендацій щодо оптимальних моментів купівлі чи продажу активів із високою точністю обчислень і мінімізацією затримок при обробці даних із зовнішнього сервісу

У дипломному проєкті розроблено автоматизовану систему збору й аналізу історичних даних про курси криптовалют, яка складається з декількох підсистем, а саме: сервісу витягування та збереження цін закриття з зовнішніх API (CoinGecko, Binance), модуля розрахунку фінансових індикаторів (SMA, EMA, RSI), рекомендаційного рушія для визначення оптимальних моментів купівлі та продажу активів, а також вебзастосунку для візуалізації отриманих даних і побудови інтерактивних графіків.

Реалізована система допоможе користувачам швидко й точно відстежувати тренди криптовалютного ринку, аналізувати сигнали технічних індикаторів та приймати обґрунтовані інвестиційні рішення на основі автоматизованого оброблення даних.

## SUMMARY

Web application leveraging web-scraping technologies to track cryptocurrency exchange rates.

The project comprises 75 pages of text, 8 figures, 7 tables, references to 20 bibliographic sources, appendices, and 4 design documents.

Keywords**:** API, CRYPTOCURRENCY, SCRAPING, SIGNALS, WEB APPLICATION

The object of this development is a web application that uses web-scraping techniques to monitor cryptocurrency exchange rates.

The diploma project developed an automated system for collecting and analyzing historical cryptocurrency closing-price data, aimed at simplifying the acquisition of such data and generating recommendations on optimal buy and sell points. The system ensures high computational accuracy and minimizes delays in processing data from external services.

The results obtained can help users rapidly and reliably track cryptocurrency market trends, interpret technical-indicator signals and make well-founded investment decisions based on fully automated data processing. вебзастосунок з використанням технологій парсингу інтернет-ресурсів для відстеження курсів криптовалют

| № рядка | Формат | Позначення | Найменування | Кіл. аркушів | № екз. | Примітка |
|---|---|---|---|---|---|---|
| 1 | | | Документація загальна | | | |
| 2 | | | | | | |
| 3 | | | Знову розроблена | | | |
| 4 | | | | | | |
| 5 | А4 | ІС12.200БАК.006 ПЗ | Вебзастосунок з використанням | 71 | | |
| 6 | | | технологій парсингу інтернет-рес- | | | |
| 7 | | | урсів для відстеження курсів крип- | | | |
| 8 | | | товалют. Пояснювальна записка | | | |
| 9 | | | | | | |
| 10 | А3 | ІС12.200БАК.005 Д1 | Вебзастосунок з використанням | 1 | | |
| 11 | | | технологій парсингу інтернет-рес- | | | |
| 12 | | | урсів для відстеження курсів крип- | | | |
| 13 | | | товалют. Діаграма компонентів | | | |
| 14 | | | | | | |
| 15 | А3 | ІС12.200БАК.005 Д2 | Вебзастосунок з використанням | 1 | | |
| 16 | | | технологій парсингу інтернет-рес- | | | |
| 17 | А3 | | урсів для відстеження курсів крип- | | | |
| 18 | | | товалют. Діаграма послідовностей | | | |
| 19 | | | | | | |
| 20 | А3 | ІС12.200БАК.005 Д3 | Вебзастосунок з використанням | 1 | | |
| 21 | | | технологій парсингу інтернет-рес- | | | |
| 22 | | | урсів для відстеження курсів крип- | | | |
| 23 | | | товалют. Діаграма потоків даних | | | |
| 24 | | | | | | |
| 25 | А3 | ІС12.200БАК.005 Д4 | Вебзастосунок з використанням | 1 | | |
| 26 | | | технологій парсингу інтернет-рес- | | | |
| 27 | | | урсів для відстеження курсів крип- | | | |
| 28 | | | товалют.Діаграма активностей | | | |

| | | | | | |
|---|---|---|---|---|---|
| | | | | | ІС12.200БАК.005 ТП |
| Зм. | Лист | № докум. | Підпис | | |
| Розробив | | Кудінов М.С. | | | Вебзастосунок з використанням технологій парсингу інтернет-ресурсів для відстеження курсів криптовалют. Відомість дипломного проєкту |
| Перевірив | | Деведжіогуллари | | | |
| | | | | | |
| | | | | | |
| Затв. | | | | | |

| Літ. | Арк. | Аркушів |
|---|---|---|
| Т | 1 | 1 |

КПІ ім. Ігоря Сікорського
Група ІС-12

# Пояснювальна записка
## до дипломного проєкту
# на тему: «Вебзастосунок з використанням технологій парсингу інтернет-ресурсів для відстеження курсів криптовалют»

Київ – 2025 року

**ЗМІСТ**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ..... 4
ВСТУП ..... 5
1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ ..... 8
Висновки до розділу 1 ..... 11
2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ ..... 12
2.1 Аналіз існуючих рішеннь на ринку ..... 12
2.2 Порівняльний аналіз існуючих сервісів ..... 17
Висновки до розділу 2 ..... 18
3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ ..... 20
3.1 Вимоги до системи в цілому ..... 20
3.2 Вимоги до функціональних характеристик системи ..... 23
3.3 Вимоги до видів забезпечення ..... 28
Висновки до розділу 3 ..... 31
4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ..... 33
4.1Вибір та обґрунтування технологій розробки ..... 33
Висновок до розділу 4 ..... 36
5 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ ..... 38
5.1 Структура системи ..... 38
5.2 Функціональна модель системи ..... 42
5.3 Передавання та обробка даних ..... 47

5.4 Архітектура програмного забезпечення ..... 50
Висновок до розділу 5 ..... 52
6 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ..... 54
6.1 Змістовна постановка задачі ..... 54
6.2 Математична постановка задачі ..... 56
6.3 Обґрунтування методу розв’язання ..... 59
Висновок до розділу 6 ..... 64
7 ТЕСТУВАННЯ СИСТЕМИ ..... 66
7.1 Мета випробувань ..... 66
7.2 Результати випробувань ..... 66
Висновок до розділу 7 ..... 71
ВИСНОВКИ ..... 72
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 74
ДОДАТОК А ..... 76

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — Application Programming Interface

CORS — Cross-Origin Resource Sharing

CSS — Cascading Style Sheets

EMA — Exponential Moving Average

HTTP — HyperText Transfer Protocol

JSON — JavaScript Object Notation

QR — Quick Response (двовимірний штрихкод)

RSI — Relative Strength Index

SCADA — Supervisory Control and Data Acquisition

SMA — Simple Moving Average

WebSocket — протокол WebSocket

# ВСТУП

Ринок криптовалют є одним із найдинамічніших сегментів сучасної фінансової сфери. Оскільки криптовалюти відрізняються високою волатильністю, недостатньою регульованістю та розпорошеністю інформації між численними біржами, трейдери й інвестори стикаються із труднощами у швидкому та коректному прийнятті рішень. Для повноцінного аналізу потребується збирати дані з різних джерел, обробляти їх у режимі реального часу та надавати в зручному вигляді. Існуючі рішення, які здебільшого ґрунтуються на одному API або не мають механізмів повторних спроб при відмові сервера, не задовольняють потреби користувачів, що потребують оперативної інформації без затримок.

Аналіз літературних джерел та досвід провідних фінансових компаній свідчать про існуючі прогалини у сфері моніторингу курсів криптовалют. Попри наявність крупних агрегаторів, вони:

— не завжди об’єднують інформацію з декількох бірж у єдиний набір даних;

— не гарантують безперервного оновлення через обмеження API та не мають розвиненої обробки помилок;

— не інтегрують алгоритми технічного аналізу напряму у клієнтський інтерфейс;

— не забезпечують достатнього рівня адаптивності для мобільних форм-факторів.

Зважаючи на це, виникає необхідність розробити вебзастосунок із багатоджерельним підходом до збору даних, реалізацією стійкого механізму retry, алгоритмів аналізу часових рядів (RSI, SMA) та гнучкого інтерфейсу, що працює без затримок.

Обґрунтування основних проєктних рішень

— вибір архітектури: проксі-сервер для обходу обмежень CORS та централізації запитів до зовнішніх API; клієнтська частина на React з використанням Zustand для глобального стану;

— методи парсингу: HTTP-запити з реалізацією механізму повторних спроб при помилках 5xx, кешування результатів для швидкого повторного відображення;

— алгоритми технічного аналізу: вбудовані механізми розрахунку RSI та SMA на клієнтській стороні для миттєвого формування торгових сигналів;

— візуалізація даних: інтеграція бібліотек для побудови адаптивних графіків (лінійні та свічкові) у залежності від обраного діапазона (24 год, 7 дн, 30 дн, 90 дн);

— адаптивний дизайн і теми: підтримка світлої та темної тем, гнучке налаштування CSS-змінних для забезпечення зручного UX на різних пристроях [1];

Запропоновано підхід до багатоджерельного збору та агрегації даних із різних бірж із використанням проксування запитів та кешування.

Розроблено механізм стійкого парсингу з логікою повторних спроб (retry) та поступовим збільшенням інтервалу між ними за умови помилок сервера (5xx).

Впроваджено алгоритми технічного аналізу (RSI, SMA) у клієнтській частині для автоматичного формування сигналів «Buy/Sell/Hold» без залучення зовнішніх сервісів.

Створено адаптивний інтерфейс із динамічним налаштуванням теми (світла/темна) та інтерфейсу, що забезпечує зручність користування на різних пристроях.

Запропоновано рішення для двосторонньої конвертації «кількість монет ↔ вартість у валюті» у режимі реального часу, яке відрізняється від існуючих серверних підходів.

Можливі галузі застосування результатів проєкту

— інтеграція в торгові термінали для покращеного моніторингу та аналітики ринку криптовалют;

— використання як освітній інструмент у курсах з фінансового аналізу та програмування;

— вбудування в аналітичні портали для блогерів, які розробляють незалежні огляди крипторинку;

— розгортання як SaaS (Software-as-a-Service) для комерційного надання користувачам можливості створення власних watchlist та портфелів із автоматичним формуванням сигналів;

— платформи для мобільних пристроїв, що використовують React Native або аналогічні клієнтські фреймворки.

Поєднання багатоджерельного збору інформації, стійкого механізму retry і клієнтських алгоритмів технічного аналізу дозволяє одержувати своєчасні торгові сигнали без залучення сторонніх сервісів, що значно підвищує оперативність прийняття рішень користувачами. При цьому адаптивний інтерфейс з підтримкою різних тем і динамічною конвертацією валют забезпечує комфортну взаємодію як на десктопі, так і на мобільних пристроях. Наступним кроком стане детальна реалізація серверної частини та модулю безпеки, які закладуть фундамент для масштабування і комерційного використання рішення.

# 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

Криптовалютний ринок виник як безпосередній наслідок впровадження та розвитку технології блокчейн. Ця технологія забезпечує децентралізоване зберігання та передачу даних без участі традиційних фінансових установ, що свідчить про фундаментальний перехід від централізованих моделей бізнесу до розподілених мереж. Однією з ключових особливостей даного сегменту є відсутність єдиного центру регулювання, що дозволяє активам створюватися, випускатися й обмінюватися без безпосередньої участі банків, держав чи міжнародних фінансових структур. Висока волатильність криптовалютних курсів у поєднанні з миттєвими коливаннями ринкових умов істотно відрізняють цей інструмент від класичних фінансових активів, таких як акції чи облігації.

Нині численні криптовалюти, серед яких Bitcoin, Ethereum та безліч альткоїнів, торгуються на десятках, а іноді й сотнях різноманітних бірж у різних країнах та юрисдикціях. У результаті цінові показники можуть значно відрізнятися на різних платформах через різний обсяг торгів, рівень ліквідності, комісійні структури та регуляторні вимоги. Наприклад, курс однієї й тієї самої криптовалюти на азійських біржах може виявитися суттєво відмінним від європейських чи північноамериканських через часові пояси, різний рівень попиту та пропозиції, а також відмінності в правилах локального регулювання. Ці розбіжності у котируваннях створюють як складнощі, так і додаткові можливості для трейдерів: з одного боку, виникає ризик отримання хибної картини реальної вартості активу; з іншого — з’являється можливість арбітражних заробітків на різниці цін.

Одним із суттєвих викликів для розробників програмного забезпечення, яке призначене для моніторингу та аналізу криптовалютних курсів, є необхідність роботи з різними джерелами вихідних даних. Найчастіше це відкриті API бірж та агрегаторів, які надають інформацію в структурованому форматі, зазвичай JSON. Проте не всі біржі надають повний набір даних у зручному вигляді: деякі ресурси вимагають авторизації, інші використовують веб-інтерфейси зі складною динамічною розміткою, що потребує здійснення парсингу HTML. Такий парсинг

ускладнюється тим, що структура вебсторінок може постійно змінюватися — наприклад, при оновленні дизайну біржі або додаванні нових функцій на сайт. Через це розробники змушені писати гнучкі алгоритми обробки, які здатні адаптуватися до швидких змін у розмітці та форматах даних.

Крім того, багато безкоштовних API встановлюють обмеження на кількість запитів за певний проміжок часу (rate limit). Наприклад, для однієї безкоштовної тарифної опції може бути обмеження у 50 запитів на хвилину, а при перевищенні цього ліміту сервер може віддавати помилки або просто не відповідати. Тому для забезпечення надійності роботи програми необхідно реалізувати механізми кешування отриманих даних та відкладеного повторного запиту (retry). Також слід передбачити збереження актуальних даних навіть за відсутності мережевого з'єднання, щоб кінцевий користувач отримував оновлену інформацію без значної затримки.

Для забезпечення якісного аналізу криптовалютних дампів окрему увагу слід приділити методам технічного аналізу. У рамках предметної області, яка охоплює збір та обробку часового ряду котирувань, використовують відомі індикатори — Relative Strength Index (RSI) та Simple Moving Average (SMA). RSI дозволяє визначити точки перекупленості чи перепроданості активу на основі відношення середнього приросту до середньої втрати за певний період, тоді як SMA обчислює ковзне середнє цін за обраний проміжок. Інтеграція таких методів безпосередньо у клієнтську частину вебзастосунку дає змогу здійснювати розрахунки в реальному часі, не залучаючи сторонні сервіси для розрахунку показників.

Все це демонструє нагальну потребу у створенні програмного рішення, яке інтегрує кілька взаємопов'язаних модулів: автоматизоване збирання даних із різних бірж із урахуванням специфіки кожного джерела, надійну обробку мережевих помилок, ефективні механізми кешування вже отриманих результатів та безперервну інтеграцію алгоритмів технічного аналізу з можливістю адаптивної візуалізації великих обсягів інформації. Перш за все, необхідно забезпечити постійне оновлення котирувань у реальному часі, що вимагає тонкого налаштування обробки запитів до API з урахуванням обмежень швидкості звернень

(rate limits). Таким чином, система повинна автоматично підлаштовувати частоту звернень і використовувати внутрішнє кешування для зменшення навантаження на зовнішні сервіси.

По-друге, недостатньо лише отримувати сирі дані: потрібно забезпечити їхню попередню валідацію, відсікати дублікати, фільтрувати неповні чи помилкові відповіді та вчасно відновлювати з'єднання у разі спаду сервісу. До того ж, усі отримані котирування мають бути приведені до єдиної форми, уніфіковані за часовими позначками та округлені відповідно до стандартів, що підвищить якість подальшого аналізу.

По-третє, інтеграція алгоритмів технічного аналізу, таких як RSI і SMA, на клієнтській стороні повинна відбуватися без помітних затримок для користувача. Це означає необхідність оптимізації розрахункових процедур, роботи з масивами даних у пам'яті браузера та поступового оновлення показників у міру надходження нових котирувань. Адаптивна візуалізація, виконана за допомогою сучасних бібліотек, має забезпечувати коректне відображення графіків незалежно від того, скільки точок даних обробляється в конкретний момент. Для цього варто використовувати методи «лінивого» завантаження даних і динамічної підвантажки, що запобіжить перевантаженню інтерфейсу та зайвому споживанню ресурсів.

Для досягнення цих цілей слід реалізувати багаторівневу архітектуру: модуль, що відповідає за збір даних з API та парсинг (якщо необхідно), має працювати незалежно від візуальної складової та алгоритмів аналізу, які, у свою чергу, повинні отримувати очищений і уніфікований набір даних. Усе це має бути організовано таким чином, щоб користувач отримував максимальну оброблену інформацію з мінімальною затримкою. Лише за таких умов можна досягти високої надійності, ефективності моніторингу ринку криптовалют та забезпечити користувачів підтвердженими, своєчасними й точними даними без зайвих затримок і втрат.

Висновки до розділу 1

У цьому розділі здійснено ґрунтовний аналіз предметної області, що стосується ринку криптовалют і розробки вебзастосунку для моніторингу їх курсів. Виокремлено ключові особливості децентралізованої технології блокчейн, з'ясовано сутність і роль основних інструментів технічного аналізу (RSI, SMA) у контексті крипторинку і обґрунтовано необхідність інтеграції цих алгоритмів безпосередньо до клієнтської частини програми.

Опис об'єкта проєктування показав, що сучасний вебзастосунок має поєднати декілька взаємодоповнювальних компонентів: автоматизоване збирання даних із декількох бірж, обробку потенційних мережевих помилок, ефективні механізми кешування для мінімізації навантаження та своєчасне оновлення інформації, а також динамічну візуалізацію великих обсягів цінових даних. Для цього об'єкт передбачає проксі-модуль для обходу обмежень CORS, клієнтську частину на базі React з менеджером стану Zustand та інтеграцію бібліотеки Recharts для побудови графіків.

Широкий опис характеристик вебзастосунку виявив вимоги до продуктивності, адаптивності та безперебійної роботи: оптимізація частоти запитів до зовнішніх API, уніфікація отриманих даних за часовими мітками, відсікання дублікатів і помилкових запитів. Зазначено, що для коректного аналізу необхідно забезпечити швидке та точне обчислення індикаторів технічного аналізу, що вимагає оптимізації алгоритмів та мінімізації затримок у взаємодії з користувачем.

Принципи взаємодії компонентів узагальнено таким чином: при завантаженні головної сторінки система [2] отримує початковий набір котирувань, далі — при потребі — завантажуються детальні дані й історичні графіки. Бібліотека Zustand забезпечує реактивне збереження стану (watchlist, портфель, вибір валюти), а зміна налаштувань миттєво оновлює всі компоненти інтерфейсу.

# 2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

## 2.1 Аналіз існуючих рішеннь на ринку

У цьому розділі здійснено ґрунтовний огляд і систематизацію основних популярних сервісів, інструментів та платформ, які надають можливості для моніторингу котирувань криптовалют. Представлено порівняльний аналіз їхніх технічних характеристик, різновидів тарифних планів, умов доступності, а також розглянуто механізми обробки запитів, алгоритми валідації даних та особливості інтеграції API у сторонні рішення. Окрім того, детально досліджено переваги та недоліки кожного сервісу з урахуванням потреб розробників вебзастосунків, а також наведено практичні приклади використання у реальних проєктах.

Криптовалютний ринок динамічно розвивається, і з кожним роком з'являються нові біржі, платформи-агрегатори та аналітичні інструменти. Це створює запит на вибір найбільш ефективного рішення, яке відповідало б таким критеріям: швидкість оновлення даних, точність і достовірність котирувань, доступність необхідних історичних часових рядів, наявність готових індикаторів технічного аналізу, стійкість до зовнішніх збоїв та обмежень API. Розглянуті сервіси дозволяють реалізувати різні стратегії взаємодії з ринком: від простого відображення цін у реальному часі до розрахунку складних індикаторів на стороні клієнта або сервера. Нижче наведено детальний опис і аналіз найбільш розповсюджених серед розробників і трейдерів платформ.

CryptoCompare — один із найпопулярніших агрегаторів даних про ринок криптовалют [3]. Сервіс надає кілька вузлів API для отримання різнопланових даних. Наприклад, вузол Price дозволяє отримувати поточні котирування для однієї або численно визначеної множини криптовалют, що корисно для формування швидких панелей моніторингу. Вузол Historical забезпечує доступ до історичних цін із змінним кроком: щохвилинні, погодинні та щоденні інтервали. Це дає змогу створювати графіки різних часових масштабів: від добової зміни протягом останніх хвилин до побудови багаторічних тенденцій. Вузол Exchange надає інформацію про обсяги торгів і ліквідність на різних біржах, що дозволяє оцінити історичну

активність окремої криптовалюти на конкретній платформі. Вузол Coin snapshot містить метадані про монету: ринкову капіталізацію, циркуляцію монет та максимальні й мінімальні ціни за останню добу.

У своїй основі CryptoCompare має такі переваги. По-перше, відсутність складної процедури реєстрації API-ключа для базових запитів дає змогу розробникам швидко почати роботу, без формальних перевірок та затримок. По-друге, формат відповіді представлено в чистому JSON-форматі, де кожен об’єкт містить метадані про валюту, часову мітку, ціну, обсяг та інші необхідні поля. Це значно спрощує процес парсингу й інтеграції даних у будь-які вебзастосунки чи серверні рішення. По-третє, сервіс підтримує широкий набір фіатних валют (USD, EUR, UAH, JPY, GBP та інші) та криптовалют, що дозволяє адаптувати відображення даних під регіональні налаштування користувача. По-четверте, наявність розширеної документації з прикладами запитів і реальними відповідями допомагає швидко зорієнтуватися у використанні API навіть початківцям.

Водночас CryptoCompare має низку суттєвих недоліків. Обмеження на кількість запитів у безкоштовному тарифному плані (наприклад, 50 запитів за хвилину) змушує розробників впроваджувати кешування й механізми чергування запитів. Це може призвести до затримок у відображенні даних у реальному часі, особливо за великого навантаження чи під час пікових торгових періодів. Для зменшення навантаження та уникнення блокування запитів доводиться впроваджувати складні логічні алгоритми – наприклад, локальне кешування даних на кілька секунд і поступове оновлення кожного елементу інтерфейсу замість масового одночасного запиту. Крім того, відсутність вбудованих індикаторів технічного аналізу (RSI, SMA, EMA) змушує розробників реалізовувати відповідні алгоритми власноруч. Це може підвищувати обсяг коду та складати додаткове навантаження на клієнтський пристрій, особливо якщо мова йде про мобільні браузери або слабкі залізячні пристрої. Нарешті, у деяких випадках дані CryptoCompare можуть містити неточності, зумовлені некоректним агрегуванням інформації з кількох бірж. Це вимагає додаткової валідації та фільтрації отриманих результатів перед використанням у розрахунках та відображенні.

CoinMarketCap [4] — ще один великий гравець серед платформ, які надають API для розробників. Сервіс пропонує широкий набір кінцевих точок: Global Metrics дає змогу отримати агреговану інформацію про весь ринок криптовалют, включаючи загальну ринкову капіталізацію, сукупний обсяг торгів та обсяги на окремих біржах. Кінцеві точки Cryptocurrency Listings дозволяють отримувати перелік активних криптовалют з інформацією про їхню поточну ціну, ринкову капіталізацію, торги за 24 години, зміну ціни у відсотках. Exchange Endpoints надають доступ до даних про конкретні біржі: кількість торгових пар, географічне розташування, рейтинги й обсяги. Tools Endpoints охоплюють різні конвертери (перетворення валют), індекси та статистичні аналітичні інструменти.

Основна перевага CoinMarketCap — це висока надійність і комплексність даних. Сервіс збирає інформацію безпосередньо з сотень бірж і гарантує її достовірність за рахунок власної системи верифікації. Це особливо важливо для професійних трейдерів і аналітичних платформ, які потребують максимально точної інформації про ліквідність, обсяги торгів у конкретні моменти та інші показники. Крім того, платні тарифні плани передбачають SLA (Service Level Agreement), що гарантує стабільність роботи API навіть за високого навантаження, а також підтримку від технічної команди. Документація CoinMarketCap містить приклади коду для різних мов програмування, що полегшує інтеграцію для команд з різним стеком технологій.

До недоліків належать необхідність платної підписки для доступу до розширеного функціоналу. Безкоштовний план обмежує швидкість запитів і кількість кінцевих точок, що може бути значущим фактором для стартапів або невеликих команд розробників, які хочуть швидко створити прототип. Навіть у платних тарифах обмеження на запити (наприклад, 3000 запитів на день у середньому тарифі) можуть ускладнити побудову високочастотних програмних рішень. Ще одним недоліком є те, що для отримання різнопланових даних (ринкові показники плюс індекси) потрібно робити кілька послідовних запитів до різних кінцевих точок, що може сповільнювати процес отримання комплексної інформації та збільшувати навантаження на мережу.

TradingView [5] — потужні інструменти візуалізації та аналітики, які давно стали золотим стандартом для багатьох трейдерів по всьому світу. Основна цінність TradingView полягає у можливості створювати та переглядати графіки в реальному часі з високою роздільною здатністю, масштабуванням за різними таймфреймами та широким набором інструментів малювання, включаючи трендові лінії, рівні підтримки та опору, фібоначчі, канали тощо. Крім того, бібліотека індикаторів TradingView містить десятки попередньо налаштованих алгоритмів (MACD, RSI, SMA, EMA, Bollinger Bands, Ichimoku Cloud, Volume Profile тощо), а редактор скриптів Pine Script дозволяє трейдерам та розробникам створювати власні індикатори або стратегії з гнучкою налаштуванням параметрів.

Процес інтеграції TradingView у сторонні вебзастосунки відбувається за допомогою вбудованих віджетів і API, які дозволяють вставити готовий графік на сторінку або використовувати бекенд-інструменти для отримання даних. Перевагою цього підходу є високий ступінь кастомізації: розробник може задавати параметри віджета, включаючи вибір таймфрейму, початкові індикатори, стиль відображення та поведінку в інтерактивному режимі. Однак безкоштовний доступ обмежений найпоширенішими індикаторами та можливістю перегляду з відставкою до 15 хвилин, що неприйнятно для трейдерів, які вимагають даних у режимі реального часу. Для комерційного використання API й віджетів потрібно укласти ліцензійну угоду та оформити підписку, що передбачає високі витрати та може стати значним бар'єром для середніх і малих проектів. Користувачі також стикаються з обмеженим контролем над зовнішнім виглядом вбудованих компонентів, що ускладнює підтримку корпоративного брендингу.

API біржі Binance — має репутацію одного з найточніших джерел даних, оскільки біржа володіє значною долею загального обсягу торгів у всьому світі. REST API Binance призначений для отримання реальних котирувань, глибини ринку (order book), обсягів торгів та детальної історії угод. Кожна криптовалютна пара має власну кінцеву точку, що повертає поточний інтерфейс торгової інформації. WebSocket API [6] дозволяє підписуватися на потоки торгів, негайно отримуючи оновлення про зміни цін і зміни глибини ринку. Order API відкриває

можливості для створення, скасування та контролю за статусом ордерів у режимі реального часу, що дозволяє створювати складні алгоритмічні стратегії.

Перевагами Binance API [7] є дуже висока оперативність надходження даних та їхня точність. Оскільки біржа є однією з найліпших за обсягами торгів, інформація завжди релевантна для поточного стану ринку. Крім того, Binance підтримує понад 600 криптовалютних пар, що дозволяє отримувати дані практично про будь-який актив. Розробники можуть використовувати офіційні SDK для Python, JavaScript, Java, PHP та інших мов, що значно спрощує процес інтеграції. Проте, незважаючи на очевидні переваги, існують значні обмеження. API охоплює лише дані біржі Binance, тому для отримання повного огляду ринку необхідно додатково інтегрувати інші джерела. Це суттєво ускладнює процес узгодження часових міток і форматів даних при об’єднанні інформації з різних бірж. Високий рівень захищеності (наприклад, підписування запитів за допомогою HMAC-SHA256) вимагає від розробників уважно поставитися до управління приватними ключами і організації безпечних середовищ. Підписка на WebSocket стріми вимагає підтримки постійного з’єднання та додаткових ресурсів для обробки великої кількості повідомлень, що може бути неприйнятно для простих вебзастосунків.

CoinPaprika — CryptoCompare та подібні агрегатори пропонують єдиний API, що охоплює інформацію з великої кількості бірж одночасно. Зазвичай такі сервіси надають кінцеві точки для отримання списку бірж, підтримуваних криптовалют, їхніх рейтингів, обсягів торгів і ринкової капіталізації. Глобальні індекси показують узагальнені показники, що відображають тренди ринку, а детальні дані про монети містять опис активу, інформацію про команду розробників, посилання на GitHub-репозиторії та соціальні мережі, що дозволяє проводити не лише фінансовий, а й фундаментальний аналіз.

Перевагою агрегаторів є централізація даних у єдиному API, що значно зменшує складність інтеграції різних джерел. Завдяки уніфікованому формату відповіді JSON розробникам не потрібно писати окремі парсери для кожної біржі. Однак безкоштовна версія таких сервісів зазвичай має жорсткі обмеження на кількість запитів за хвилину, що створює необхідність реалізації складної логіки

кешування та відкладеного виконання запитів. Також може спостерігатися затримка у відображенні останніх даних, пов’язана з процесом агрегування інформації на сервері провайдера. Для трейдерів високої частоти ця затримка може бути критичною. Крім того, деталізація даних, наприклад глибина ринку (order book), часто недостатня, що обмежує використання агрегаторів для покриття потреб алгоритмічної торгівлі. Платні плани з розширеними аналітичними звітами та історичними графіками можуть мати високу вартість для проєктів із обмеженим бюджетом.

Мобільні застосунки — такі як Blockfolio, Delta і схожі продукти, забезпечують користувачам можливість контролю власного криптопортфеля у режимі реального часу. Функціонал включає ручне додавання активів або синхронізацію з біржовими акаунтами, а також систему сповіщень про зміни цін чи досягнення встановлених порогів. Інтерфейс найчастіше є інтуїтивно зрозумілим: кольорові діаграми, гістограми обсягу торгів, кнопки для швидкого оновлення даних та можливість перегляду детальної статистики конкретного активу. Головною перевагою таких рішень є їхня доступність на ходу: користувач може швидко перевірити стан портфеля під час поїздки або зустрічі без необхідності заходити до вебінтерфейсу. Однак відсутність відкритого API обмежує можливості розробників інтегрувати ці мобільні застосунки у власні програмні продукти. Розробка власного мобільного інтерфейсу, порівнянного за функціоналом з Blockfolio чи Delta, вимагає залучення додаткових технологій (React Native, Flutter, Kotlin/Swift) та ресурсів на підтримку кросплатформеності й безперебійного оновлення даних.

## 2.2 Порівняльний аналіз існуючих сервісів

Відповідно до попереднього огляду платформ для моніторингу криптовалютних котирувань та їх класифікації, можна визначити декілька ключових параметрів, на які буде звертатись увагу при порівнянні цих інструментів. На таблиці 2.1 зображено їх порівняльну характеристику.

Таблиця 2.1 – Порівняльна характеристика платформ для моніторингу криптовалютних котирувань та їх класифікації

| Сервіс | Безкоштовний доступ | Обмеження запитів | Вбудовані індикатори | Real-time оновлення | Детальні історичні дані |
|---|---|---|---|---|---|
| CryptoCompare | + | + | - | + | + |
| CoinMarketCap | - | + | - | + | + |
| TradingView | - | - | + | - | + |
| Binance API | + | + | - | + | + |
| CoinPaprika | + | + | - | - | + |
| Мобільні застосунки | + | - | - | - | - |

Ця порівняльна характеристика дозволяє швидко оцінити, які сервіси задовольняють потреби в безкоштовному доступі, реальному часу оновлення даних, наявності технічних індикаторів та глибоких історичних часових рядах.

Висновки до розділу 2

Проведений аналіз дозволив виявити, що сучасний ринок інструментів для моніторингу котирувань криптовалют характеризується широкою різноманітністю пропозицій, але жодне із доступних рішень не забезпечує всебічного покриття ключових потреб розробників і користувачів одночасно. Безкоштовні агрегатори (CryptoCompare, CoinPaprika, CryptoCompare) спрощують доступ до даних і не

вимагають складної реєстрації, але вони обмежені кількістю запитів і не завжди надають усе необхідне для поглибленого технічного аналізу. Професійні сервіси (CoinMarketCap, TradingView) надають розширений функціонал, включаючи великий перелік індикаторів, докладні історичні часові ряди та графічні інструменти, проте значна частина цих можливостей доступна лише за платною підпискою, що часто перевищує бюджет невеликих команд.

API окремих бірж (зокрема Binance) відзначаються високою оперативністю та точністю даних, оскільки надходять безпосередньо з джерела, але охоплюють лише одну платформу, що змушує розробників інтегруватися з кількома API одночасно і додатково опрацьовувати різні формати даних та часові мітки. Мобільні застосунки (Blockfolio, Delta) зручні для користувачів, які стежать за портфелем «на ходу», але вони не передбачають відкритого API і не підходять для вбудованих рішень у вебзастосунки.

У результаті жодне з наявних рішень не поєднує таких властивостей, як: миттєве оновлення котирувань з кількох бірж без обмежень запитів, автоматизоване кешування і обробка помилок, наявність готових алгоритмів технічного аналізу та адаптивна візуалізація даних у рамках одного інструменту.

Отже, необхідність створення власного вебзастосунку, що об’єднуватиме переваги різних платформ — безкоштовний доступ, гнучке агрегування, стійкість до збоїв і інтеграцію технічних індикаторів без додаткових витрат — є цілком обґрунтованою. Цей підхід дозволить задовольнити потреби як розробників, так і кінцевих користувачів, забезпечивши їм надійний, ефективний та доступний інструмент для моніторингу та аналізу криптовалютного ринку.

# 3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

## 3.1 Вимоги до системи в цілому

### 3.1.1 Функціональні вимоги

Перш за все, система повинна збирати поточні дані про ціни криптовалют із кількох різних джерел, що включають публічні API сервісів обміну криптовалютами та агрегатори котирувань. Інтервал між запитами до цих джерел має бути достатньо коротким для забезпечення майже миттєвого оновлення інформації на екранах користувачів, але не перевищувати однієї хвилини, щоб не перевантажувати зовнішні сервіси зайвими запитами. У разі тимчасової недоступності будь-якого із джерел потрібно реалізувати механізм багаторазових повторних спроб отримання даних із поступовим збільшенням затримки до максимально допустимого ліміту, після чого, у разі подальшої відсутності відповіді, інтерфейс має відображати повідомлення про недоступність даних, не блокуючи при цьому інші функції .

На головній сторінці додатку користувач повинен бачити список доступних криптовалют із оновленими у реальному часі значеннями цін, а також базову інформацію — зміну вартості у відсотках за останню добу, ринкову капіталізацію та інше. Для кожного рядку списку передбачається інтерактивна можливість переходу до детального перегляду будь-якої конкретної монети, де користувач отримає розширену інформацію про обрану криптомонету, включаючи історичні графіки цін, які можна прокручувати та змінювати часовий діапазон без необхідності постійного повторного завантаження тих самих даних. Також у цьому режимі мають відображатися кілька базових технічних індикаторів, зокрема індекси відносної сили (RSI), ковзні середні (SMA), експоненційні ковзні середні (EMA) та інші стандартні показники, обрахунки яких здійснюються безпосередньо на боці клієнта на основі отриманої інформації про ціни. Це дозволить користувачеві отримувати додатковий аналітичний контекст при прийнятті рішень.

Крім відображення котирувань, програма повинна надавати користувачеві можливість формувати персональний портфель криптовалют: додавати активи

шляхом введення назви монети, кількості та вартості придбання. При цьому система розраховує поточну оцінку вартості цього активу та визначає процентне відхилення від початкової вартості у вигляді прибутку чи збитку. На основі цих даних генеруються рекомендації щодо потенційних дій — купувати, продавати чи утримувати — на основі обраного алгоритму, що може враховувати обчислений індекс RSI, співвідношення ковзних середніх та інші показники. Детальний інтерфейс портфеля повинен відображати загальну суму інвестицій, поточну вартість портфеля, приріст у грошовому еквіваленті та виражений у відсотках, а також перерахунок для кожного окремого активу.

Додатковим функціоналом є можливість формування та редагування списку спостереження (watchlist), де користувач зберігає перелік монет, які його найцікавіші в поточний момент. Додати або видалити монету зі списку слід одним натисканням відповідної кнопки поруч із назвою монети. Данні із списку спостереження мають завантажуватися з приоритетом, а при першому відкритті документу потрібно отримати інформацію про всі монети списку одночасно, використовуючи можливість пакетного запиту до API, якщо це підтримується зовнішнім сервісом.

### 3.1.2 Нефункціональні вимоги

#### 3.1.2.1 Продуктивність та масштабованість

Клієнтська частина повинна вміти обробляти та коректно відображати інформацію про не менше ніж п'ятдесят активних монет одночасно без будь-яких помітних затримок, забезпечуючи час відклику інтерфейсу не більше двох секунд у пікові моменти навантаження. При завантаженні даних необхідно застосовувати механізм відображення заповнювачів («скелетонів»), щоб користувач бачив анімацію завантаження, поки відбувається підвантаження даних. Серверна частина (що може бути реалізована як легкий проксі-сервер для переадресації запитів до публічних API) повинна підтримувати горизонтальне масштабування. Це означає, що у разі перегляду зростання кількості користувачів або великих навантажень

інфраструктура може розширитися шляхом додавання додаткових вузлів, щоб підтримувати стан системи без втрати продуктивності.

3.1.2.2 Надійність, стійкість до збоїв та відновлення даних

У разі втрати зв’язку з будь-яким із зовнішніх джерел або виходу їх із ладу система повинна зберігати останню валідну інформацію в локальному кеші чи сховищі браузера, що дозволяє показувати користувачеві найсвіжіші відомості навіть протягом перебоїв. Механізм багаторазових повторних спроб запиту з поступовим збільшенням інтервалу між викликами гарантує відновлення роботи після відновлення доступу до сервісів. У разі критичного збою серверної інфраструктури має відбутися автоматичний перезапуск сервісу, щоб час простою не перевищував п’яти хвилин. Ключові дані користувача, такі як інформація про формування персонального списку спостереження та портфеля, повинні резервуватися з інтервалом не рідше ніж один раз на добу, щоб мінімізувати ризик втрати даних.

3.1.2.3 Безпека

Усі комунікації з публічними API та між клієнтською частиною додатку та сервером мають здійснюватися тільки через захищений протокол HTTPS, щоб уникнути ризику перехоплення або маніпуляцій даними під час їх передачі по мережі. Якщо система в майбутньому буде розширюватися і вимагати аутентифікації користувачів, потрібно забезпечити надійне шифрування облікових даних, наприклад, використання токенів із підписом або захищених сесійних механізмів. Доступ до будь-яких облікових даних API, що вимагають приватних ключів, необхідно обмежувати конфіденційністю — зберігання таких ключів має відбуватися у точках сховища, недоступних безпосередньо із фронтенд-коду.

3.1.2.4 Юзабіліті та інтерфейс

Інтерфейс користувача має бути адаптивний, щоб забезпечувати коректне відображення на екрані комп'ютерів, планшетів та смартфонів. Темна і світла теми мають перемикатися без перезавантаження сторінки, а вибір теми зберігатися локально для відновлення наступного разу. Навігаційна структура повинна бути інтуїтивно зрозумілою: головне меню повинно містити ярлики для переходу до таких розділів як «Головна», «Список спостереження», «Портфель» та «Налаштування». Кожна сторінка має містити чіткий заголовок, а взаємодії з елементами — супроводжуватися короткими спливаючими підказками.

3.1.3 Вимоги до інтеграції з зовнішніми сервісами

Архітектура системи має бути спроектована так, щоб інтеграція з новими джерелами даних не вимагала суттєвих змін у бізнес-логіці. Слід виділити окремий рівень абстракції для взаємодії з API, який уніфікує різні формати відповідей від зовнішніх сервісів у єдиний внутрішній формат. Це дозволить при необхідності замінити одного постачальника даних іншим без впливу на основні модулі. Крім того, для кожного джерела слід реалізувати механізм контролю швидкості запитів та емулювати чергу викликів так, щоб запобігати перевищенню лімітів, встановлених зовнішніми сервісами, що значно знижує ризик тимчасової блокування.

3.2 Вимоги до функціональних характеристик системи

3.2.1 Отримання та оновлення даних у реальному часі

Застосунок повинен забезпечувати безперервний збір поточної інформації про котирування криптовалют з кількох джерел одночасно. Дані оновлюються автоматично із заданою частотою, що не перевищує одну хвилину, таким чином користувач завжди бачить майже актуальні значення цін і зміни вартості. Після

кожного запиту необхідно перевіряти коректність отриманих значень: у випадку некоректних або неповних даних слід відхилити хибну інформацію і використовувати останню достовірну версію, що зберігається в локальному сховищі браузера. Якщо жодне джерело не відповідає протягом заданого інтервалу (наприклад, через відмову мережі або збій API), то інтерфейс повинен відобразити повідомлення про тимчасову недоступність даних, але при цьому не блокувати інші функціональні елементи. Наприклад, якщо основний сервер котирувань окупантської біржі недоступний, система автоматично використовує альтернативну точку доступу, а у разі критичної відмови — повідомляє користувача та продовжує відображати дані з локального кешу.

У процесі отримання нових даних застосунок реалізує механізм повторних спроб із адаптивним інтервалом між викликами. Після першої невдалої спроби звернення до API очікування між спробами подвоюється, що дозволяє уникнути надмірного навантаження зовнішніх сервісів та знижує ймовірність блокування за великою кількістю запитів. Якщо після трьох спроб отримання даних жодне джерело не відповість, система переходить у режим відображення даних із останнього успішного кешу та виводить сповіщення про неможливість оновити інформацію.

### 3.2.2 Перехід до детального перегляду активу та інтерактивна візуалізація

Загальний перелік активів на головному екрані містить коротку інформацію про кожну криптовалюту: поточну ціну, відсоткову зміну за останню добу та інші базові параметри. Для одержання детальнішої інформації користувач може натиснути на будь-який рядок списку, що переводить його на екран детального перегляду обраної монети. На цьому екрані має бути реалізована інтерактивна візуалізація даних у вигляді динамічного графіка, який відображає зміну ціни у часовому діапазоні, обраному користувачем. Інтервал між точками графіка може бути налаштований окремо: наприклад, згортання даних до хвилинного кроку для перегляду найактуальніших змін у межах одного дня, або інтервали по годинах і

днях для більш тривалих періодів. При зміні діапазону система завантажує додаткові дані для відображення нових часових точок, водночас забезпечуючи плавний перехід без повного перезавантаження сторінки. Це дає змогу користувачеві швидко переглянути історичні дані з деталізацією та проаналізувати тренди без навантаження інтерфейсу.

### 3.2.3 Розрахунок і відображення технічних індикаторів

Додаток повинен включати можливість розрахунку стандартних технічних індикаторів для аналізу руху ціни активів. Користувач має обирати період індикатора, наприклад, 14-денний для індексу відносної сили (RSI), або короткий та довгий період для ковзних середніх (SMA, EMA). Обчислення індикаторів виконуються на стороні клієнта без необхідності надсилати окремі запити на сервер; вихідними даними для обчислень служать масиви історичних цін, завантажені під час перегляду детального екрану. Отримані значення відображаються у спеціальній області під основним графіком або накладаються безпосередньо на лінію ціни, що дозволяє одночасно бачити тренд і показники індикатора. Після зміни періоду індикатора (наприклад, при переключенні з 14 на 30 днів) система повторно обробляє наявний масив даних і оновлює графік без повторного завантаження деяких точок.

Додатково можна передбачити відображення декількох індикаторів одночасно — наприклад, поєднання RSI і SMA з різними періодами — для більш комплексного аналізу. При цьому інтерфейс повинен підтримувати включення/виключення окремих індикаторів одним натисканням, а також налаштування кольорів і стилів ліній для кращої візуалізації. Якщо користувач переходить до іншого часовоготра рэону, всі активні індикатори автоматично перераховуються на підставі оновленого діапазону значень цін і плавно оновлюються на екрані.

### 3.2.4 Управління персональним списком спостереження

Механізм «списку спостереження» необхідний для швидкого доступу до активів, які становлять найбільший інтерес користувача. У загальному переліку криптовалют один дотик до спеціальної кнопки додає монету до цього списку. Після цього вибраний актив з'являється у розділі «Список спостереження», де відображається найважливіша статистика: поточна ціна, відсоткові коливання за добу і загальний рейтинг монети за капіталізацією. Інформація в розділі списку спостереження оновлюється з таким самим інтервалом, як і в основному переліку, а користувач може видалити актив одним натисканням на кнопку поруч із назвою. Повна синхронізація між двома списками (загальним та «спостереження») має відбуватися без затримок.

Для кожного активу у списку спостереження бажано відображати також мінімум і максимум ціни за останні 24 години та рівень ліквідності (за доступними даними), а додатково — процентну різницю між ціною на декількох біржах, якщо система підтримує агрегування котирувань із різних джерел. При цьому є потреба у реалізації фонового оновлення даних зі збільшеною частотою для активів зі списку спостереження, щоб інформація була максимально актуальною.

### 3.2.5 Управління криптопортфелем та відображення прибутку чи збитку

Розділ портфеля призначений для введення та відображення персональних інвестиційних даних. Користувач може додавати новий актив, вказавши його назву, кількість придбаних монет та ціну купівлі (автоматично використовує поточну ціну, якщо вартість не задана вручну). Система обчислює поточну вартість кожного активу, множачи кількість на актуальну ринкову ціну. Потім для кожного активу розраховується процентна різниця між поточною вартістю й початковими витратами, що задає відображення прибутку чи збитку: позитивні значення виділяються зеленим, негативні — червоним кольором. Для зручності користувача поруч із кожним елементом портфеля може виводитися рекомендація щодо

можливого рішення (купувати, продавати, утримувати), сформована на основі аналізу технічних індикаторів, таких як RSI або SMA.

Головна сторінка портфеля має відображати загальну суму всіх вкладених коштів, поточну загальну вартість портфеля та сумарний процентний приріст (або зниження) від початкового капіталу. Також система може надавати можливість перегляду динаміки портфеля у вигляді окремого графіка: наприклад, лінія, що показує загальну вартість портфеля протягом одного місяця або більше. У цьому випадку необхідно заздалегідь накопичити історичні дані про оцінку портфеля або розраховувати їх на льоту, використовуючи існуючі часові ряди цін для кожного активу. У разі зміни вартості активу графік портфеля автоматично оновлюється, демонструючи тенденції зростання чи падіння.

### 3.2.6 Пошук та фільтрація активів

Реалізація фільтрації та пошуку активів має бути максимально простою і швидкою. Пошуковий інструмент передбачає відображення результатів у міру введення символів: наприклад, при наборі «eth» система показує всі монети, у назві або коді яких зустрічається цей рядок. Пошук нечутливий до регістру символів та скорочень. Критерії фільтрації мають включати такі опції, як «зростання за останню добу», «спад за останню добу», «найвища ринкова капіталізація» і «найбільший об’єм торгів». Вибір фільтрації відразу оновлює відображений список, не вимагаючи нового запиту до сервера — всі операції виконуються на локально завантаженому масиві даних. Для зручності користувача можна додати можливість комбінації фільтрів: наприклад, одночасно показувати лише ті монети, які зросли понад 5 % за останню добу та мають об’єм торгів понад певного порогу.

## 3.3 Вимоги до видів забезпечення

### 3.3.1 Вимоги до математичного забезпечення

У вебзастосунку реалізовано функціонал для обчислення основних технічних індикаторів, необхідних для аналізу ринкових даних. Зокрема, модуль обчислює ковзні середні (SMA і EMA) з різними періодами, використовуючи стандартні методи з урахуванням вагових коефіцієнтів EMA відображена. Розрахунок відображено у формулі 3.1

$$\mathrm{EMA_t} = \alpha P_t + (1 - \alpha)\,\mathrm{EMA_{t\text{-}1}} \tag{3.1}$$

В даному випадку $P_t$ — ціна активу (закриття) в момент часу t, $\alpha$ — коефіцієнт згладжування (smoothing factor), який визначає ступінь вагового наближення до останніх даних, $\mathrm{EMA_t}$ — значення EMA на поточному кроці часу t

Також реалізовано розрахунок індексу відносної сили (RSI) на заданому інтервалі, що дозволяє відображати швидкість й силу цінових змін. Генерація торгового сигналу поєднує результати RSI та ковзних середніх для прийняття рішення про купівлю, продаж або утримання активу. Розрахунки показоно у формулах 3.2, 3.3, 3.4 та 3.5 відповідно.

$$\mathrm{RSI_t} = 100 - \frac{100}{1 + \mathrm{RS_t}} \tag{3.2}$$

$$\mathrm{RS_t} = \frac{\overline{\mathrm{Gain}_N}}{\overline{\mathrm{Loss}_N}} \tag{3.3}$$

$$\overline{\mathrm{Gain}_N} = \frac{1}{N}\sum_{i=1}^{N} \max(P_{t-i+1} - P_{t-i},\, 0) \tag{3.4}$$

$$\overline{\mathrm{Loss}_N} = \frac{1}{N}\sum_{i=1}^{N} \max(P_{t-i} - P_{t-i+1},\, 0) \tag{3.5}$$

$N$ — період (кількість інтервалів) для розрахунку RSI (наприклад, 14). $\overline{\text{Gain}}_N$ — середнє значення позитивних змін ціни за останні $N$ інтервалів (усі негативні зміни вважаються нульовими). $\overline{\text{Loss}}_N$ — середнє значення абсолютних величин негативних змін ціни за останні $N$ інтервалів (усі позитивні зміни вважаються нульовими). $\text{RS}_t$ — відношення середнього приросту до середнього падіння за період $N$. $\text{RSI}_t$ — індекс відносної сили на момент t, що змінюється в діапазоні від 0 до 100.

Обчислення виконуються з достатньою точністю, застосовуючи числа з плаваючою комою, а результати округлюються до двох десяткових знаків для відображення на графіках і в таблицях.

### 3.3.2 Вимоги до інформаційного забезпечення

Система отримує реальні дані котирувань із CoinGecko через проксі, налаштований у конфігурації Vite. Запити виконуються за допомогою бібліотеки axios. Обробка відповіді передбачає перевірку наявності об’єктів із полями id, current_price, market_cap, price_change_percentage_24h та sparkline_in_7d.price. Якщо відповідь не містить очікуваних полів або масиви порожні, у коді передбачена обробка помилкового стану — відображення заглушки або повідомлення про помилку.

Метадані про монети (назва, ідентифікатор, символ, URL зображення) отримуються з API під час першого завантаження списку монет. На момент рендерингу комбінований список доступних варіантів для «Add Asset» формується на основі отриманих об’єктів із полями id та name.

У разі відмови зовнішнього API реалізовано механізм повторних спроб із збільшенням затримки: початкова затримка становить 1000 мс, після кожної невдалої відповіді вона подвоюється, а кількість спроб обмежено трьома. Якщо всі спроби невдалі, викликається виняток і в інтерфейсі відображається помилка завантаження.

### 3.3.3 Вимоги до програмного забезпечення

Клієнтська частина реалізована на React із використанням Vite як інструменту збірки і скорочення часу на завантаження. Використовується бібліотека Zustand для керування глобальним станом (змінні watchlist, portfolio, theme, currency). React-компоненти розбиті на модулі за функціоналом:

Компоненти списку монет (CoinItem) відображають інформацію про ціну, спарклайн-графік та кнопку керування watchlist. Графік реалізовано через Recharts.

Компоненти детального перегляду активу (CoinDetail) виконують запити для отримання детальної інформації про монету та історичні дані для графіка, включаючи кешування отриманих значень у локальному об’єкті для зменшення повторних запитів.

Компоненти портфеля (PortfolioAssetsList) виконують запити до API, отримують поточну ціну та історичні дані, обчислюють сигнали, RSI та ковзні середні на стороні клієнта.

Сторінка AddAsset формує форма для ручного введення параметрів активу, виконує запит на отримання поточної ціни за допомогою axios.

Застосовано react-router для навігації між сторінками: Home, Watchlist, Portfolio, AddAsset.

Код написано відповідно до стандартів ES6+, використовуючи async/await для обробки асинхронних запитів. Для обробки помилок використовуються try/catch. Стилі реалізовані через CSS-файли та CSS-модулі для компонентів. Збірка фронтенда настроєна у Vite із попереднім налаштуванням оптимізації імпорту та мінімізації коду.

Набір юніт-тестів написано за допомогою Jest для функцій обчислення (calculateSMA, calculateRSI, generateTradingSignal). Інтеграційні тести відсутні, однак тестові приклади в юніт-тестах перевіряють загальні сценарії обчислень.

### 3.3.4 Вимоги до технічного забезпечення

Середовище розгортання передбачає використання Node.js (версія 14 або вище) для локальної розробки та Vite-серверу. Мінімальні вимоги для роботи на машині розробника:

— двоядерний процесор із тактовою частотою не менше 2 ГГц;

— 8 ГБ оперативної пам’яті, достатньо для одночасного запуску редактора коду, браузера та dev-серверу;

— SSD-накопичувач об’ємом не менше 100 ГБ для зберігання коду та залежностей;

— стабільне інтернет-з’єднання для доступу до зовнішніх API та npm-реєстру.

У якості інструментів використано Git для контролю версій та npm для керування залежностями. Збірка клієнта виконується через команду npm run build, результатом якої є оптимізований пакет статичних файлів.

На поточному етапі реалізовано локальний проксук API через конфігурацію Vite, що спрямовує запити на https://api.coingecko.com/api/v3. Виробниче середовище може використовувати будь-який HTTP-сервер (наприклад, Nginx) для хостингу сформованих статичних файлів, але цей процес не відображено безпосередньо в коді.

## Висновки до розділу 3

Було наведено комплексну модель технічного завдання для вебзастосунку, який забезпечує моніторинг курсів криптовалют. Спочатку було окреслено загальні вимоги до системи, де визначено принципи отримання та автоматичного оновлення даних у реальному часі, механізми інтерактивної візуалізації та управління персональним портфелем і списком спостереження. Ці функціональні можливості створюють основу для зручної роботи користувача з відображенням котирувань, детальним аналізом активів і підрахунком прибутків або збитків.

Додатково описано нефункціональні вимоги, що регламентують продуктивність і масштабованість інтерфейсу, надійність збереження даних під час збою зовнішніх джерел, безпеку передачі інформації та адаптивність дизайну для різних пристроїв. Завдяки цим критеріям система отримує гарантію стабільної роботи навіть за пікових навантажень і обмежень мережі, а також забезпечує захист конфіденційних даних і комфортний користувацький досвід.

Деталізовано вимоги до інтеграції з зовнішніми сервісами через уніфікований абстракційний шар для різних API, що дає змогу легко підключати нові джерела даних і керувати обмеженнями запитів без потреби змінювати бізнес-логіку. Нарешті, визначено вимоги до супроводу й документації, які передбачають підготовку повного набору технічних специфікацій, опис архітектури, інструкцій із розгортання та налаштування CI/CD, що забезпечують прозорість роботи й можливість швидкого залучення нових розробників.

Таким чином, узагальнення функціональних, нефункціональних, інтеграційних і супровідних вимог створює чіткий орієнтир для подальшої розробки та підтримки системи. Дотримання цих вимог забезпечить зростання надійності, безпеки і гнучкості вебзастосунку, що, у свою чергу, сприятиме задоволенню потреб користувачів у швидкому, точному та зручному моніторингу криптовалютного ринку.

# 4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

## 4.1 Вибір та обґрунтування технологій розробки

При створенні вебзастосунку для моніторингу курсів криптовалют особливу увагу було приділено підбору таких технологій, які забезпечують швидкий час реакції інтерфейсу, високу продуктивність при обробці великих обсягів даних у режимі онлайн, гнучкість архітектури задля подальшої підтримки та масштабованості, а також простоту інтеграції з різноманітними зовнішніми API. Враховуючи необхідність відображати складні графічні елементи, підтримувати миттєву реакцію на дії користувача та забезпечувати безперебійну роботу навіть за непостійну мережу, було зроблено вибір на користь сучасних фронтенд-інструментів, лаконічного клієнтського кешування та легковагового проксі через Vite.

Клієнтську частину реалізовано за допомогою бібліотеки React [8], яка наразі є однією з найпопулярніших [9] і найактивніше розвинутих серед компонентних фреймворків. Основні переваги цього підходу полягають у можливості розбити інтерфейс на окремі самодостатні компоненти, кожен з яких відповідає за певну частину логіки або відображення. Наприклад, окремо реалізовано компонент, який відповідає за відображення кожного елементу списку активів, інший — за показ спарклайн-графіку поточної зміни ціни, третій — за детальний перегляд історичних даних щодо обраної монети. Така модульна структура дозволяє швидко вносити зміни до окремих частин застосунку без необхідності зачіпати загальний код, а отже, спрощує підтримку та подальше розширення функціональності.

Крім фундаментальних інструментів для побудови компонентів, React надає можливість використовувати хук-підхід із React Hooks [10]. Це узагальнення звичайної роботи із локальним станом компонентів та життєвим циклом, яке значно спрощує логіку обробки подій користувача та оновлення даних всередині кожного модуля. Використання хуків робить код компактнішим, кращим для читання та легко тестованим, а також мінімізує кількість зайвих перерендерів, що

є критично важливим у сценарії, коли на екран виводиться велика кількість активів, кожний з яких оновлює свою інформацію у реальному часі.

Для керування станом застосунку обрано бібліотеку Zustand, яка вирізняється простотою API і мінімалістичністю. У нашому проєкті глобальний стан включає кілька основних складових: список спостереження користувача, його портфель, обрану тему інтерфейсу (світлу чи темну) та валюту відображення (USD, EUR або UAH). Принцип роботи Zustand базується на тому, що компоненти підписуються лише на ту частину стану, яка їм необхідна, що уможливлює оновлення тільки тих елементів інтерфейсу, які змінилися. Це позитивно впливає на загальну продуктивність, оскільки зменшується число зайвих перерендерів та оптимізується час реакції на зміни даних, особливо коли йдеться про одночасний показ десятків або сотень елементів.

Для організації середовища розробки та збірки проєкту обрано Vite, який у ролі сервера розробки забезпечує практично миттєвий запуск за рахунок використання ES-модулів напряму в браузері. У результаті, замість довгих перезборок, розробник моментально бачить зміни, що вносить у код, що суттєво скорочує цикл розробки та пришвидшує процес валідації нових ідей. Крім того, проєкт налаштовано так, що у режимі продакшну Vite автоматично виконує оптимізацію: мінімізує JavaScript-код, «розщеплює» його на чанки (code splitting) для швидшого завантаження та забезпечує відлагоджене кешування, яке підвищує швидкість відображення сторінки у браузері відвідувача. Це означає, що кінцевому користувачеві не доводиться чекати на завантаження всіх функцій одразу: завдяки розділенню пакету сторінка завантажується швидше, а необхідні скрипти підвантажуються за потреби.

Стилізацію інтерфейсу виконано за допомогою звичайних CSS-файлів із використанням CSS-перемінних. Такий підхід дозволяє суттєво знизити кількість залежностей і забезпечує високу гнучкість у налаштуванні вигляду. Завдяки застосуванню CSS-перемінних, підтримка світлої та темної тем здійснюється буквально шляхом зміни значення кількох глобальних змінних у :root або через застосування атрибута data-theme, що приводить до миттєвого оновлення всіх

компонентів. Це значно спрощує подальше масштабування та адаптацію інтерфейсу під різні стильові вимоги, адже розробнику залишається лише додати нові властивості для перемінних або розширити існуючий набір.

Для побудови інтерактивних графіків обрано бібліотеку Recharts, яка ідеально інтегрується з React. Recharts дає можливість створювати різноманітні типи діаграм: лінійні, стовпчикові, точкові та багато інших, а також забезпечує підтримку спливаючих підказок, що відображають точні значення при наведенні курсору, можливість масштабування та прокручування даних. Це необхідно для повноцінного аналізу користувачем історичних коливань цін, адже він може переглядати детальні дані за останню добу, тиждень або місяць, бачити точні числові значення та змінювати масштаб графіка, щоб глибше зрозуміти динаміку. Завдяки такій гнучкості Recharts інтерфейс стає не лише інформативним, а й зручним для подальшого аналізу та ухвалення рішень.

У проєкті не застосовано окремої серверної частини у вигляді Express.js чи схожого фреймворка. Замість цього налаштування проксі для запитів до CoinGecko здійснено безпосередньо в конфігураційному файлі vite.config.js. Усього кілька рядків коду в конфігурації перенаправляють всі запити з префіксом /api на кінцеву точку https://api.coingecko.com/api/v3, що дозволяє вирішити проблеми з CORS і не створювати зайвий серверний бекенд. Таким чином, клієнт сам виконує всі запити до CoinGecko через налаштований проксі, що спрощує архітектуру та економить ресурси.

Усі HTTP-запити на стороні клієнта здійснюються за допомогою бібліотеки axios. Цей вибір пояснюється універсальністю інтерфейсу для обробки запитів та відповідей, а також підтримкою перехоплення (інтерцепторів) для централізованого оброблення помилок або додавання заголовків. Крім того, ключова утиліта requestWithRetry реалізована у модулі src/api/coingecko.js. Вона відповідає за повторні спроби запиту у разі помилок із кодом 5xx, автоматично збільшуючи інтервал затримки між наступними спробами, що гарантує більшу надійність отримання даних під час тимчасових несправностей із боку зовнішнього API [11].

Для забезпечення продуктивного відображення даних клієнт використовує кешування в оперативній пам’яті та Local Storage. Local Storage відповідає за збереження списку спостереження (watchlist), даних портфеля користувача та налаштувань теми між сесіями, що полегшує відновлення роботи після перезавантаження сторінки без повторних запитів до сервера. Для кешування історичних даних, необхідних для побудови графіків, застосовується in-memory кеш, який діє протягом однієї сесії роботи додатку. Це означає, що якщо користувач повертається до перегляду раніше отриманих графіків протягом тієї самої сесії, нові запити до CoinGecko не виконуються, що значно зменшує навантаження на мережу та пришвидшує відображення даних.

Контроль якості коду здійснюється за допомогою інтеграції ESLint і Prettier. ESLint автоматично перевіряє [12] вихідний код на відповідність встановленим правилам стилю та на наявність потенційних помилок, тоді як Prettier відповідає за автоматичне форматування файлів згідно з єдиним стилем, що спрощує командну роботу та сприяє зрозумілості та однорідності коду. Це також дозволяє уникнути розбіжностей у форматуванні при роботі кількох розробників над одним файлом.

Висновок до розділу 4

У цьому розділі докладно обґрунтовано вибір технологій для реалізації вебзастосунку з моніторингу курсів криптовалют. Використання React і Vite дозволило створити високопродуктивний та інтерактивний інтерфейс, що швидко реагує на дії користувача та забезпечує оперативне відображення великих обсягів даних. Бібліотека Recharts забезпечує ефектну й зручну візуалізацію історичних графіків, необхідних для глибокого аналізу тенденцій.

Рішення про застосування Vite-проксі замість окремого серверного бекенду спрощує архітектуру, дозволяючи без зайвих витрат отримувати дані з CoinGecko. Завдяки цьому проєкт уникає надмірних залежностей, а всі API-запити обробляються напряму із браузера, що підвищує надійність і гнучкість.

Функція requestWithRetry, реалізована в модулі coingecko, автоматизує повторні спроби отримання даних за умов помилок 5xx, що підвищує стабільність припливу котирувань. Керування станом за допомогою Zustand забезпечило оптимальну кількість перерендерів і швидке оновлення компонентів, навіть коли користувач працює одночасно зі значною кількістю активів.

Для кешування даних використано поєднання Local Storage та in-memory кешу, що гарантує швидке відновлення налаштувань користувача між сесіями та мінімізує кількість запитів до API. Засоби контролю якості коду — ESLint та Prettier — підтримують єдиний стандарт оформлення й дозволяють зменшити технічний борг під час командної роботи.

Таким чином, обрані технології створюють надійну, масштабовану й швидку основу для вебзастосунку, що відповідає сучасним вимогам моніторингу криптовалют. Використані інструменти гарантують зручність розробки й підтримки, високу швидкодію інтерфейсу та стійкість до збоїв, забезпечуючи користувачам комфортний користувацький досвід.

# 5 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 5.1 Структура системи

У загальному вигляді інформаційну систему можна представити як взаємодію трьох основних шарів:

Клієнтський шар виконує роль інтерфейсу користувача та відповідає за відображення візуальних елементів, приймання та обробку подій користувача. До його складу входять такі функціональні блоки:

Головна сторінка, на якій відображається перелік криптоактивів з відповідними поточними цінами, відсотковими змінами за останні 24 години та первинною інформацією (іконка, назва, символ). Кожен рядок у таблиці є інтерактивним елементом, що дозволяє перейти до детального перегляду конкретного активу.

На рисунку 5.1 показаний інтерфейс головної сторінки застосунку

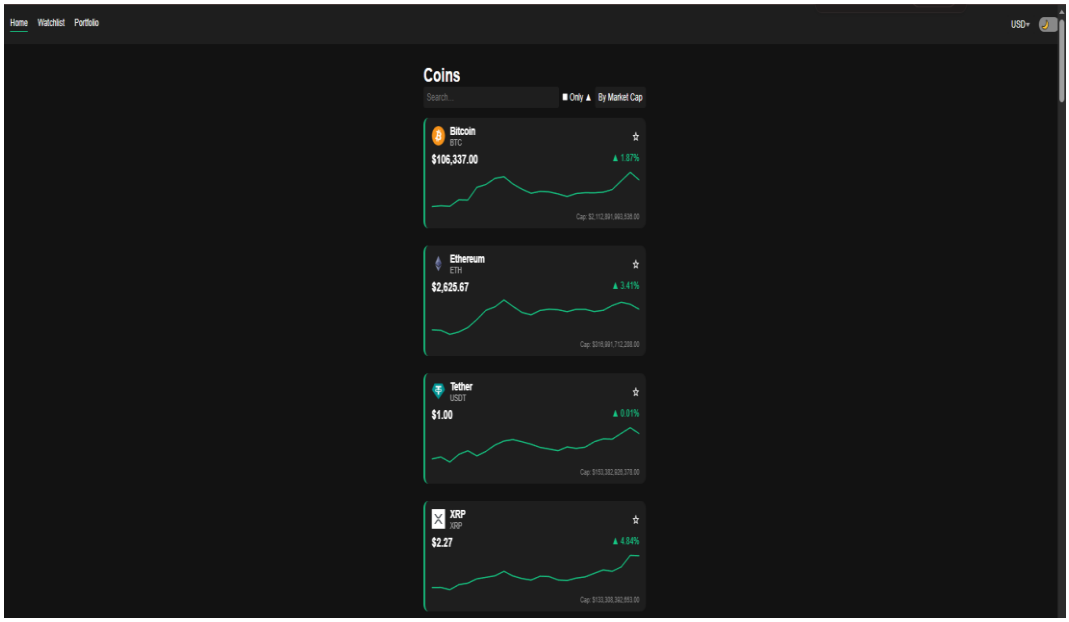


Рисунок 5.1 – Інтерфейс головного екрану застосунку

Екран детального перегляду обраної монети. Цей блок відповідає за відображення розгорнутої інформації: історичні графіки котирувань за різні часові інтервали (24 години, 7 днів, 30 днів, 90 днів), технічні індикатори (RSI, SMA,

EMA), а також детальні метадані (ринкова капіталізація, обсяг торгів, максимальна / мінімальна ціна за добу). Користувач може змінювати часовий діапазон та тип графіка (наприклад, лінійний або свічковий), що забезпечує гнучкість аналізу.

На рисунку 5.2 зображена сторінка детальної інформації про вибрану монету.

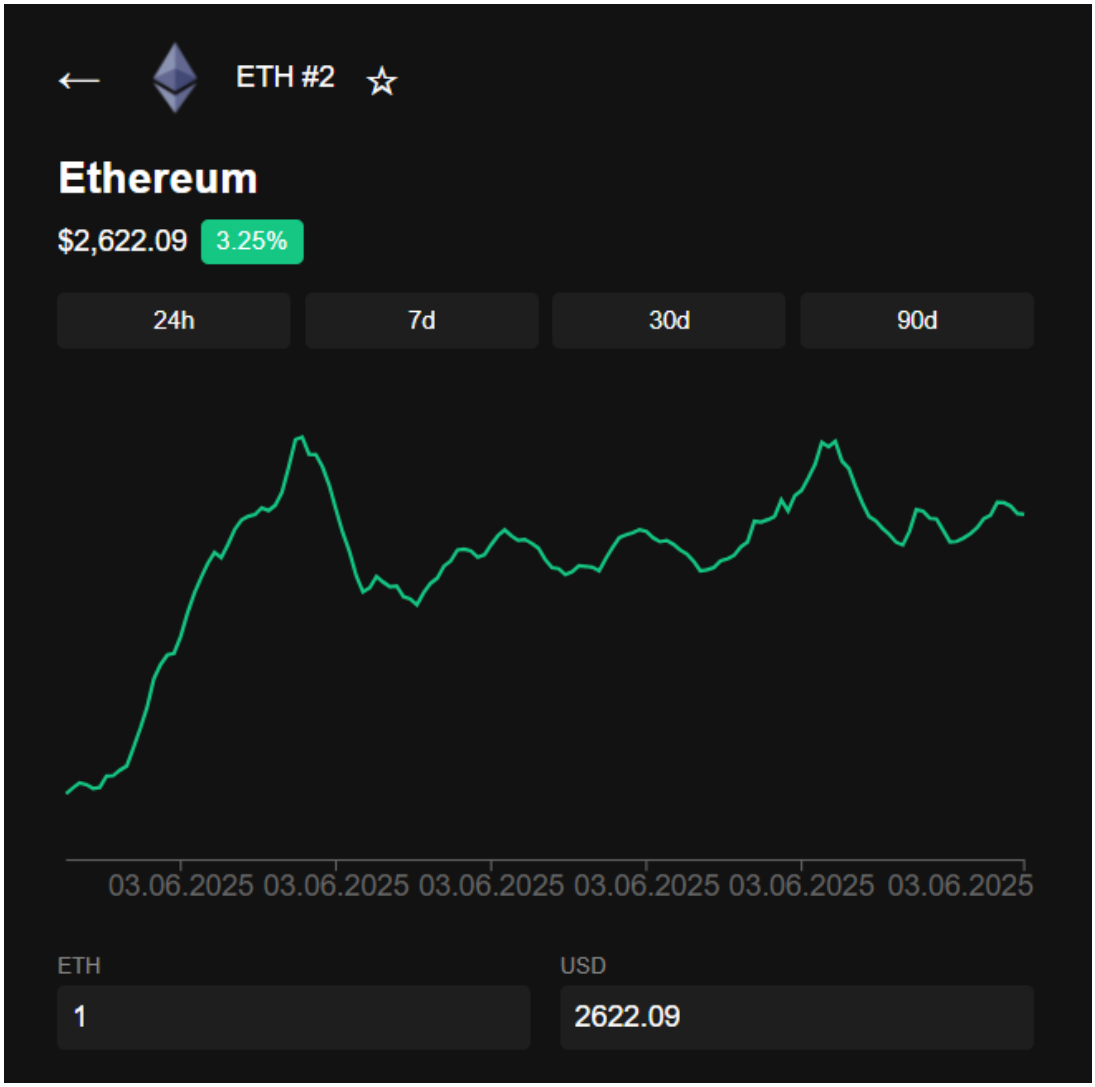


Рисунок 5.2 – Екран з детальною інформацією про обрану монету

Розділ «Список спостереження» (Watchlist) призначений для відображення обраних користувачем активів. Кожен запис містить базову інформацію про ціну, відсоткову зміну та спарклайн-графік, що відображає тренд за останній період. Користувачі можуть додавати або видаляти активи з цього списку одразу з головної сторінки чи екрана детального перегляду.

Розділ «Портфель» (Portfolio) демонструє активи, які користувач придбав раніше. Тут у реальному часі відображається поточна вартість портфеля, а для кожного активу розрахований прибуток чи збиток. Передбачено можливість додавання нових активів із введенням кількості куплених монет і ціни придбання. Система автоматично оновлює значення та відображає співвідношення між поточною ціною та ціною купівлі.

Секція налаштувань теми й валюти відображення, що зображено на рисунку 5.3. Це окремий модуль, який дозволяє користувачеві вибрати між світлою та темною темою, а також обрати основну валюту відображення (USD, EUR, UAH). Дані налаштування зберігаються у Local Storage, що дозволяє відновлювати їх між сесіями.

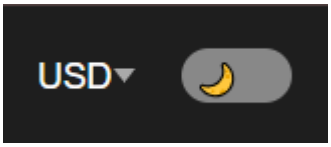


Рисунок 5.3 – Вибір теми та валюти застосунку

Кожен із цих функціональних підрозділів реалізовано у вигляді React-компонентів. Компоненти маніпулюють внутрішнім локальним станом через React Hooks, а для обміну даними між ними використовується глобальний стан, організований за допомогою Zustand. Це дає змогу підтримувати узгодженість даних, уникати надмірних перерендерів та забезпечувати швидке оновлення інтерфейсу при зміні стану.

Проміжний шар (API-проксі) відповідає за перенаправлення запитів клієнта до зовнішніх сервісів, таких як CoinGecko. Цей рівень не містить окремого серверного коду, натомість налаштування проксі закладено в конфігураційному файлі Vite (vite.config.js). При зверненні клієнта за адресою з префіксом /api, проксі автоматично переадресовує запит на відповідний кінцевий шлях CoinGecko (https://api.coingecko.com/api/v3). Така архітектура дозволяє обійти обмеження

CORS без створення додаткових серверних компонентів та забезпечує централізовану конфігурацію URL-адрес API.

У клієнтській частині, у модулі src/api/coingecko.js, реалізовано функцію requestWithRetry, що виконує наступні завдання:

Формує HTTP-запит за допомогою бібліотеки axios, додаючи необхідні параметри (наприклад, вибраний користувачем інтервал часу, валюта, ідентифікатор монети).

Перехоплює відповіді з кодом помилки 5xx та здійснює до трьох повторних спроб з поступовим подвоєнням інтервалу затримки (backoff-стратегія). Це забезпечує більш надійне отримання даних під час тимчасових збоїв на стороні CoinGecko.

Усю комунікацію з API відображає у вигляді об’єктів JavaScript, що потім використовуються компонентами для оновлення інтерфейсу.

Сховище даних складається з двох рівнів кешування:

In-memory кеш (кеш у пам’яті). Зберігає результати нещодавніх запитів про історичні дані котирувань для кожної монети у межах поточної сесії. Це означає, що коли користувач перемикає часовий інтервал (наприклад, з 7 днів на 24 години), попередньо отримані дані, що вже зберігаються в оперативній пам’яті, використовуються без повторного звернення до сервера, що значно пришвидшує відображення графіків та знижує кількість запитів.

Local Storage використовується для збереження сталих налаштувань: списку спостереження (watchlist), даних про портфель, обраної теми та валюти відображення. Ці дані зберігаються між сесіями браузера, тож після перезавантаження сторінки або повторного відкриття застосунку користувач відразу бачить останній збережений стан без необхідності повторно налаштовувати параметри чи формувати список активів.

## 5.2 Функціональна модель системи

### 5.2.1 Доступ до поточних котирувань криптоактивів

При запуску вебзастосунку або переході на головну сторінку користувач бачить актуальний перелік криптовалют зі значеннями їх курсів, відсотковими змінами за останню добу та ринковою капіталізацією. Для реалізації цього сценарію система виконує такі кроки:

— користувач переходить на головну сторінку (короткий приклад URL: /), що ініціює подію завантаження компонента Home;

— компонент звертається до глобального стану (Zustand) або до in-memory кешу для отримання вже наявних даних про список активів. Якщо дані застарілі (наприклад, минуло більше 1 хвилини з останнього оновлення) або кеш порожній, виконується виклик функції getMarketData з модуля src/api/coingecko.js;

— getMarketData (через requestWithRetry) надсилає HTTP-запит до проксі (/api/coins/markets) з параметрами: сторінка = 1, кількість елементів = 50 (або інша конфігурація за потреби), валюта = обрана користувачем. Проксі передає запит на сервіс CoinGecko;

— після отримання відповіді система перевіряє наявність помилок (код 5xx). У разі невеликих збоїв реалізована стратегія кількох повторних спроб із подвоєнням інтервалу. Після успішного отримання відповідь зберігається у in-memory кеші та оновлюється глобальний стан;

— компонент Home отримує нові дані зі стану й оновлює відображення: відображає таблицю зі списком активів, куди входять такі поля, як іконка, назва, символ, поточна ціна, відсоткове відхилення, спарклайн-графік (малюється за допомогою Recharts на основі останніх 24 годин).

5.2.2 Перегляд детального графіка та технічні індикатори

Коли користувач клацає на рядок із конкретним активом у списку, відбувається перехід на сторінку детального перегляду (URL-патерн: /coin/{id}). Алгоритм дій:

— компонент CoinDetail ініціює завантаження базової інформації про монету, звертаючись до getDetailedCoinData({id});

— getDetailedCoinData надсилає запит через проксі до /api/coins/{id}, отримує метадані (курс, ринкову капіталізацію, обсяги тощо). Після отримання відповідь зберігається в стані, а компонент відображає заголовок із назвою монети, її символом та рангом;

— одночасно компонент ініціює запит для побудови графіка: викликається getCoinMarketChart(id, days, currency) з розміром періоду за замовчуванням (наприклад, days = 1 (24 години)) ;

— getCoinMarketChart надсилає запит до проксі /api/coins/{id}/market_chart, отримує масив точок [мітка часу, ціна]. Якщо розмір масиву дуже великий (наприклад, > 200 точок), здійснюється downsampling: відбираються кожні n елементів для обмеження кількості точок, щоб графік не був перевантажений;

— після отримання даних компонент обчислює технічні індикатори на стороні клієнта: у модулі src/utils/tradingSignal.js є функції calculateSMA, calculateRSI, а також generateTradingSignal, які повертають значення SMA7, SMA14, RSI14 та самого сигналу («Buy», «Sell», «Hold»). Обробка даних виконується на основі масиву цін, отриманого з API;

— компонент масштабує масив даних графіка у формі об’єктів {time: "dd.mm.yyyy", price: <значення>}, і передає їх до Recharts <LineChart> чи іншого обраного типу. При наведенні курсору на графік слайдер показує точні значення ціни у вказаний момент;

— якщо користувач змінює період (наприклад, з 1 на 7 днів, 30 днів чи 90 днів), компонент перевіряє, чи є відповідні дані в кеші. Якщо немає або вони

застаріли, виконується новий запит за відповідними параметрами (через getCoinMarketChart). Після отримання — повторюється обчислення індикаторів та оновлення графіка.

### 5.2.3 Керування списком спостереження (Watchlist)

Цей сценарій передбачає операції додавання, видалення, відображення активів у персональному списку:

— користувач клацає на кнопку «додати/видалити» поряд із активом у компоненті CoinItem. Виклик addToWatchlist(id) або removeFromWatchlist(id) зберігає оновлений масив ID у глобальному стані;

— при кожній зміні глобального стану React-компонент Watchlist отримує оновлення і викликає getMarketData(1, watchlist.length, currency) з фільтром ids, щоб отримати дані лише по обраних монетах;

— результат запиту зберігається у стані та передається компоненту Watchlist, який відображає таблицю зі спарклайн-діаграмами, поточними цінами та змінами відсотків для кожної монети зі списку;

— одночасно новий список watchlist синхронізується з Local Storage: при оновленні сторінки чи після перезавантаження браузера дані зчитуються з Local Storage і записуються до стану.

### 5.2.4 Функції портфеля (Portfolio)

Користувач може додавати активи до свого віртуального портфеля з відображенням обсягу інвестицій і прибутку/збитків:

— переходячи на сторінку портфеля (/portfolio), компонент Portfolio викликає PortfolioAssetsList;

— компонент PortfolioAssetsList бере масив об’єктів {id, unique_id, quantityBought, priceBought, image, ticker} з глобального стану;

— отримані значення поточної ціни та сигнал зберігаються локально у стані компоненту, і здійснюється розрахунок доходності: worth = currentPrice * quantityBought, invested = priceBought * quantity Bought, plPercent = ((worth – invested) / invested) * 100;

— результати відображаються у вигляді рядків із іконкою, назвою, символом, поточною вартістю, відсотком прибутку/збитку та сигналом («Buy», «Sell», «Hold») у вигляді кольорових боксів;

— якщо користувач натискає кнопку «видалити» поряд із активом, викликається removeAsset(unique_id), що видаляє об’єкт з глобального стану й Local Storage, а компонент PortfolioAssetsList автоматично оновлює перелік;

— під час додавання нового запису користувач заповнює форму: обирає монету зі списку allCoins, введення quantity та необов’язково priceBought. Якщо priceBought не вказано, після вибору монети здійснюється виклик getDetailedCoinData(id) для автоматичного отримання актуальної ціни;

— після натискання «Add Asset» новий об’єкт з унікальним unique_id і даними з форми додається в глобальний стан й Local Storage, а користувача автоматично переадресовують на сторінку портфеля.

### 5.2.5 Пошук та фільтрація активів на головній сторінці

Пошук та фільтрація виконуються таким чином:

— на кожну зміну поля пошуку (onChange) виконується фільтрація масиву coins, що зберігається у локальному state;

— якщо встановлений прапорець «Only ▲» (filterPositive), додатково застосовується фільтр за умовою c.price_change_percentage_24h > 0;

— далі застосовується сортування масиву за обранною опцією sortBy: «By Market Cap», «By Price» чи «By 24h %». Сортування проводиться локально шляхом виклику sort;

— після кожного пошуку або фільтрації оновлений масив sorted передається в компонент CoinItem для відображення.

5.2.6 Налаштування теми та валюти відображення

У застосунку можна змінювати тему та валюту відображенння:

— при зміні значення currency у глобальному стані (через setCurrency) усі компоненти, що використовують курс, автоматично перерисовуються з новою валютою;

— при перемиканні теми через setTheme оновлюється атрибут data-theme у кореневому елементі документу, змінюючи значення CSS-перемінних та відображаючи світлу або темну тему по всьому інтерфейсу;

— обрані значення зберігаються у Local Storage, що дає змогу відновити налаштування під час повторного завантаження браузера.

5.2.7 Обробка помилок і повідомлення користувачеві

Обробрка помилок відбувається наступним чином:

— якщо під час отримання даних з зовнішнього API виникає помилка (мережеве з’єднання відсутнє або код відповіді >= 400), функція requestWithRetry обробляє тільки помилки 5xx, у той час як помилки 4xx передаються далі для обробки;

— у разі помилки 5xx функція автоматично повторює запит до трьох разів із наростаючою затримкою (наприклад, 1000 мс, 2000 мс, 4000 мс). Якщо всі спроби виявилися невдалими, компонент, що викликав запит, отримує об’єкт помилки та встановлює локальний стан помилки;

— UI-компоненти відображають блоці-повідомлення (компонент Error) із текстом типу "Не вдалося завантажити дані. Спробуйте пізніше." або інший контекстно залежний текст;

— після відмови запитів в API компоненти все ще показують кешовані дані (якщо вони були), наприклад, у вигляді спарклайн-діаграм із написом "Дані працюють із кешу";

— якщо помилка виникла під час додавання нових елементів у портфель чи watchlist (наприклад, Local Storage переповнений), система відображає відповідну підказку або повідомлення, щоб користувач зміг вжити заходів (видалити непотрібні записи).

## 5.3 Передавання та обробка даних

Перший етап передачі даних — ініціація запиту з інтерфейсу. Коли користувач взаємодіє з елементом інтерфейсу (наприклад, клацає на кнопку оновлення списку монет або вибирає новий часовий інтервал для графіка), відповідний React-компонент викликає функцію з модуля src/api/coingecko.js. Цей виклик містить точний шлях і параметри запиту: ідентифікатор монети, кількість днів для історичних даних або конкретні параметри фільтрації. Модуль coingecko.js формує URL із префіксом /api, додаючи до нього необхідні query-параметри, після чого передає запит до axios.

Другий етап — передача запиту через проксі Vite. Axios відправляє HTTP-запит на адресу. Оскільки у конфігураційному файлі vite.config.js [13] налаштовано перенаправлення всіх запитів з префіксом /api на https://api.coingecko.com/api/v3, проксі автоматично набирає повний URI (https://api.coingecko.com/api/v3/coins/markets) і передає запит до CoinGecko. Цей крок гарантує, що клієнтський код не викликає напряму сторонній сервер, розв’язуючи проблему CORS.

Третій етап — отримання відповіді від CoinGecko. Після обробки запиту зовнішній сервіс повертає JSON-об’єкт із масивом криптовалют або із деталями обраної монети чи історичними даними. Проксі Vite приймає цю відповідь і передає її назад axios. Якщо сталася помилка на стороні CoinGecko з кодом 5xx (наприклад, тимчасова непрацездатність сервісу), axios потрапляє у catch-блок, де функція requestWithRetry здійснює повторну спробу з подвоєнням інтервалу. Після трьох безрезультатних повторів виклик завершиться помилкою, і компонент отримає інформацію про невдалу операцію.

Четвертий етап — обробка відповіді у модулі coingecko.js. Якщо код відповіді 200, функція requestWithRetry повертає оброблений результат у вигляді JavaScript-об’єкта. Далі компонент, який викликав цю функцію, отримує дані й передає їх у глобальний стан через відповідні методи сховища Zustand. Одночасно дані можуть бути записані до in-memory кешу або Local Storage (якщо це список спостереження чи портфель). Якщо компонент передбачає подальше обчислення (наприклад, обчислення технічних індикаторів), отримані масиви цін передаються до функцій calculateSMA та calculateRSI, які повертають масив значень індикаторів для кожної точки історії. Ці значення пакетуються у об’єкт разом із початковими даними й передаються Recharts для відображення.

П’ятий етап — оновлення інтерфейсу. Після того як результат записано у глобальний стан, відповідні компоненти підписані на цей стан отримують оновлення. Наприклад, якщо компонента Home раніше викликала getMarketData, то в новому render-циклі вона прочитає масив монет зі стану й побудує список елементів через компонент CoinItem. Якщо графік або індикатори оновлено, CoinDetail перерисовує компонент <LineChart> із новим набором даних і технічних показників.

Шостий етап — збереження налаштувань і кешування. Для зменшення кількості повторних запитів і швидкого відновлення стану застосунку після закриття чи перезавантаження використовуються кеші двох типів. In-memory кеш — це звичайна JavaScript-структура (наприклад, об’єкт або Map), що зберігає останні отримані значення для конкретних комбінацій параметрів (тип запиту, ID монети, часовий інтервал). При наступному виклику ті самі параметри провіряються в цьому кеші, і якщо запис знайдено, запит до API не надсилається. Local Storage використовується для сталого збереження списку спостереження, даних портфеля й налаштувань інтерфейсу. Після успішного оновлення стану ці значення записуються до Local Storage. Під час ініціалізації застосунку при завантаженні сторінки Global Store зчитує дані з Local Storage і наповнює первинний стан застосунку.

Сьомий етап — обробка помилок і fallback-механізми. Якщо при зверненні до CoinGecko помилка відповіді 4xx, requestWithRetry не виконує повторні спроби, а безпосередньо повертає помилку компоненту. Компонент оновлює локальний стан помилки, і в UI з'являється повідомлення типу «Помилка отримання даних. Спробуйте пізніше». Якщо помилка відбувається при читанні чи записі Local Storage (наприклад, пристрій користувача має обмежені можливості), система відображає повідомлення про неможливість збереження налаштувань, але не блокує всю роботу застосунку. Якщо in-memory кеш недоступний (наприклад, збій у пам'яті), застосунок переходить без кеша, запитуючи дані завжди через API і відображає користувачу сповіщення про можливі затримки. Детально ознайомитись з етапами передавання та обробки данних можна на рисунку 5.4

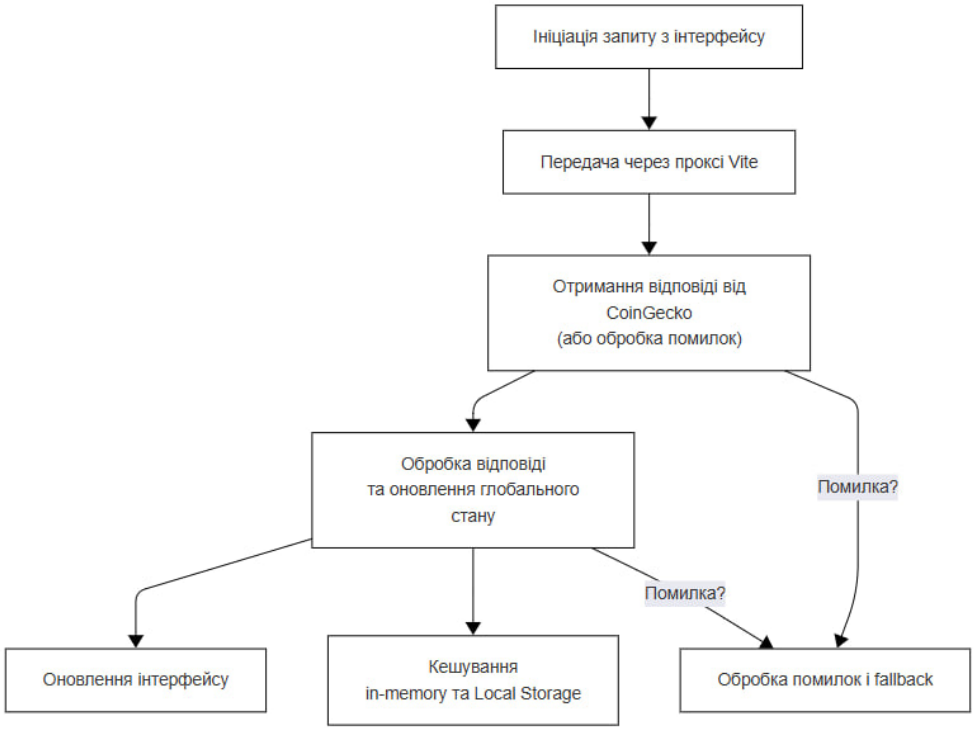


Рисунок 5.4 – Етапи механізму передавання та обробки даних

Рисунок представлено для кращого сприйняття інформації передачі та обробки даних у застосунку

## 5.4 Архітектура програмного забезпечення

### 5.4.1 Схема архітектури програмного забезпечення

У системі вебзастосунку для моніторингу курсів криптовалют реалізовано два основні програмні компоненти:

— клієнтську частину (фронтенд), яка відповідає за взаємодію з користувачем, відображення інтерфейсу, отримання даних та їх обробку на стороні браузера;

— API-проксі, сконфігуроване у файлі vite.config.js, що перенаправляє запити клієнта до зовнішнього сервісу CoinGecko.

UI-компоненти (React), а саме:

— пеалізовані як набір незалежних компонентів (Home, CoinDetail, Watchlist, Portfolio, AddAsset та ін.;

— відповідають за відображення списків криптовалют, графіків, форм для додавання активів та налаштувань інтерфейсу;

— використовують React Hooks для керування локальним станом і рефракторингу коду;

Глобальний стан (Zustand) реалізоване саме так:

— сховище, яке містить дані про вибрану валюту, тему (світла/темна), список спостереження (watchlist) та портфель користувача;

— компоненти підписуються на конкретні фрагменти стану для мінімізації зайвих перерендерів;

— збільшена стійкість даних забезпечується записом стану у Local Storage. In-memory кеш та Local Storage;

— in-memory кеш реалізовано як простий JavaScript-об’єкт у пам’яті, що зберігає останні отримані дані для кожного запиту за параметрами (наприклад, історичні часи ряди);

— local Storage використовується для збереження постійних даних: списку спостереження, портфеля, налаштувань теми та валюти;

— при ініціалізації застосунку у Local Storage читаються дані, що відновлюють попередній стан користувача.

Модуль запитів (src/api/coingecko.js):

— містить функцію requestWithRetry(path, params, retries, delay), яка формує HTTP-запит за допомогою axios до проксі /api, обробляє помилки 5xx шляхом повторних спроб із подвоєнням інтервалу затримки;

— надає методи для отримання списку монет (getMarketData), детальної інформації (getDetailedCoinData), ринкових графіків (getCoinMarketChart) та інших кінцевих точок;

— гарантує коректну обробку відповідей, кешування і оновлення глобального стану.

Утиліти для обчислень (src/utils/tradingSignal.js):

— функції для розрахунку ковзних середніх (SMA), індексу відносної сили (RSI) та формування торгового сигналу (generateTradingSignal);

— використовуються на клієнті після отримання масивів історичних цін із API;

— результати обчислень передаються безпосередньо у компоненти для побудови графіків (SMA, RSI накладаються на лінію ціни).

Проксі (Vite):

— налаштовано у vite.config.js перенаправлення всіх запитів із префіксом /api на кінцеву точку CoinGecko (https://api.coingecko.com/api/v3);

— дозволяє уникнути проблем з CORS і зменшити кількість конфігурацій серверної частини;

— автоматично змінює URL запиту перед відправленням на зовнішній сервіс.

Зовнішній сервіс (CoinGecko API):

— надає актуальні дані про ринкові котирування криптовалют, історичні часові ряди, детальні метадані монет;

— відповідає у форматі JSON і підтримує різні кінцеві точки для отримання різних типів інформації;

— користувач взаємодіє з UI-компонентами, викликаючи події (клік, зміна інтервалу тощо);

— UI-компоненти звертаються до модуля запитів (src/api/coingecko.js) для отримання чи оновлення даних;

— після здійснення HTTP-запиту через /api Proxy/, проксі пересилає запит на CoinGecko;

— відповідь повертається назад через проксі до модуля запитів, де вона обробляється requestWithRetry і передається у глобальний стан;

— глобальний стан записує дані у in-memory кеш та (за потреби) у Local Storage;

— компоненти, що підписані на змінені дані, автоматично оновлюють відображення;

— у разі потреби утиліти для обчислень обробляють масиви історичних цін і повертають результати обчислень у компоненти.

Таким чином, архітектура програмного забезпечення забезпечує чітку розмежованість обов’язків між компонентами, спрощує масштабування та підтримку, а також гарантує відмовостійкість і швидкий доступ до даних [14].

Для більшої деталізованості було створено діаграму компонентів, яка була наведена в кресленику ІС12.200БАК.005 Д1. Також була створена діаграма послідовностей для сценарію «Відкрити сторінку детальної монети» в кресленику ІС12.200БАК.005 Д2, діаграма потоків даних в кресленику ІС12.200БАК.005 Д3 та діаграма активностей в кресленику ІС12.200БАК.005 Д4 [15]

Висновок до розділу 5

У результаті розділ 5 демонструє, як багаторівнева архітектура вебзастосунку забезпечує органічне поєднання користувацького інтерфейсу, проміжного проксі та механізмів кешування для безперервного та надійного обміну даними. Клієнтський шар із React-компонентами та реактивним станом через Zustand відповідає за миттєве відображення списку криптоактивів, динамічні графіки й

обчислення технічних індикаторів, водночас зберігаючи гнучкість налаштувань теми та валюти користувача. Проксі-рівень, реалізований у конфігурації Vite, перехоплює й коректно перенаправляє запити до CoinGecko, обходячи обмеження CORS і забезпечуючи стратегічні повторні спроби в разі помилок API. Застосування in-memory кешу для оперативного збереження останніх запитів у поточній сесії й Local Storage для довготривалого зберігання watchlist, портфеля та уподобань гарантує швидкий відгук інтерфейсу та відновлення попереднього стану після перезавантаження. Завдяки такому чіткому розподілу обов’язків, оптимізованому обміну даними та продуманим механізмам обробки помилок система досягає високої продуктивності, стійкості до збоїв і готовності до розширення нових функціональних можливостей. Крім того, реалізована модульність сприяє швидкому впровадженню нових функцій, таких як підтримка додаткових бірж або аналітичних інструментів, без необхідності глобальних змін у кодовій базі. Завдяки використанню сучасних бібліотек та перевірених паттернів розробки, проект підтримує високі стандарти якості коду та забезпечує легкість супроводу й тестування.

# 6 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

## 6.1 Змістовна постановка задачі

У сучасних умовах світовий ринок криптовалют демонструє унікальні властивості, серед яких високі коливання цін, швидка зміна трендів та наявність численних торгових платформ із неоднорідною ліквідністю. Криптовалютна екосистема є відносно молодою, але вже сформувала значну кількість активних учасників від індивідуальних трейдерів до інституційних інвесторів. У зв’язку з цим виникає необхідність у розробці інструментів для автоматизованого моніторингу котирувань, аналізу ринкової ситуації та вчасного генерування торгових сигналів.

Основна мета дипломного проєкту полягає у спрощенні процесу збору історичних даних про ціни закриття криптовалют та формування рекомендацій щодо оптимальних моментів купівлі чи продажу активів із високою точністю обчислень і мінімізацією затримок при обробці даних із зовнішнього сервісу. Особливу увагу приділено максимальному рівню точності обчислень та мінімізації затримок при отриманні й обробці даних із зовнішнього сервісу (CoinGecko).

З самого початку роботи передбачається налагодити коректний процес формування бази вхідних даних, що охоплює як збір точних історичних записів про ціну закриття кожної криптовалюти з бажаним інтервалом оновлення (від щохвилинного до п’ятихвилинного, залежно від інтерфейсу користувача), так і забезпечення цілісності та повноти отримуваних даних. Усе це передбачає механізми обробки можливих пропусків або аномалій, коли ціни виявляються нульовими чи екстремально завищеними.

Наступним кроком у математичному забезпеченні є розрахунок ковзних середніх (SMA) із різними періодами. З огляду на необхідність виявлення спрямованості тренду, потрібно визначити оптимальні значення короткої SMA та довгої SMA, спираючись на аналіз волатильності криптовалютного ринку. Оскільки часові ряди можуть містити тисячі точок, обрано підхід із ковзною сумою: кожна нова точка призводить до коригування суми за одиницю часу, що

забезпечує швидкість оновлення за постійної складності O(1). Додатково, необхідно передбачити механізми на початкових обчисленнях (коли кількість інтервалів менша за обрані періоди SMA) для уникнення хибних оцінок: у таких випадках можна присвоїти значення «не застосовується» або виключити цю частину ряду з розрахунків.

Ще одним важливим індикатором, що входить до системи, є індекс відносної сили (RSI), який обчислюється методом Вайлдера через згладжене середнє приростів та втрат. Вхідними даними для цього розрахунку служить послідовність цін, з якої попередньо виділяються одиничні прирости та втрати. Початкові значення середніх розраховують як суми приростів та втрат за фіксований період днів, після чого застосовується рекурсивний підхід: кожного разу, коли з'являється нове значення цінового інтервалу, звичайне середнє оновлюється з урахуванням попереднього значення й поточного приросту або втрати. Особливої уваги вимагає ситуація, коли втрат за період не було: у цьому випадку формально AvgLoss дорівнює нулю, що призводить до обчислення RS як нескінченності та призначення RSI значення 100.

На основі результатів розрахунків ковзних середніх і RSI формується торговий сигнал. Ключові порогові значення RSI визначають зони перепроданості та перекупленості, а взаємодія короткої та довгої SMA вказує на зміну тренду. Якщо коротка SMA перевищує довгу, це свідчить про початок тенденції зростання, або коли RSI переходить нижче певного порогу, система автоматично генерує сигнал купівлі. Навпаки, якщо коротка SMA опускається нижче довгої чи RSI перевищує інший поріг, формується сигнал продажу. У разі ж відсутності чітких ознак тренду чи коли сигнал RSI та SMA дають суперечливі вказівки, системі передбачено можливість залишати позицію у стані утримання до з'явлення більш виразного імпульсу.

Для забезпечення високої швидкодії обчислення індикаторів виконуються на клієнтській стороні за допомогою JavaScript; дані для індикаторів кешуються у пам'яті браузера для повторного використання без зайвих запитів. Це дозволяє уникнути повторних витрат часу на перерахунок тих самих діапазонів. Також

упроваджено інструменти для оптимального оновлення графіків: лімітована кількість точок на малюнку, що рідко перевищує 200, та використання ефективного механізму перерендеру, щоб забезпечити плавність інтерфейсу.

У випадку помилок чи неточностей у даних передбачені кілька рівнів резервних механізмів. У разі збоїв при запитах до CoinGecko використовується повторна спроба із поступовим збільшенням інтервалу очікування. Якщо ж надати оновлені значення все ж не вдається, для показу індикаторів система повертається до останніх кешованих даних і виводить повідомлення користувачу про можливе відставання інформації. Таким чином, навіть за умов недоступності зовнішнього API застосунок продовжує працювати, хоч і з менш свіжими даними.

## 6.2 Математична постановка задачі

Позначимо $\{p_t\}_{t=1}^{T}$ як дискретну послідовність цін закриття обраної криптовалюти за моменти часу від $t = 1$ до t = T де — загальна кількість записів у базі даних. Саме ця послідовність є вихідною інформацією для подальших обчислень.

Ковзні середні (SMA) двох періодів обчислюються за формулою 6.1. Визначаємо два періоди: короткий $n_1$ та довгий $n_2$, причому завжди $n_1 > n_2$ . Для кожного моменту t, коли накопичено щонайменше $n_k$ попередніх значень ( $t > n_k$ ), обчислюємо ковзну середню як:

$$SMA_{n_k}(t) = \frac{1}{n_k} \sum_{i=0}^{n_k-1} p_{t-i}, \quad k \in \{1,2\}. \tag{6.1}$$

В даній формулі $n_k$ довжина вікна (кількість періодів), за яким обчислюється середнє. Маємо два вікна: $n_1$ — коротке вікно (наприклад, 7 днів) та $n_2$ — довге вікно (наприклад, 14 днів), t — індекс поточного моменту часу, для якого рахується $SMA$; вимагає $t \geq n_k$, щоб накопичилося щонайменше $n_k$ попередніх значень. $\sum_{i=0}^{n_k-1} p_{t-i}$ — сума цін закриття за останні $n_k$ інтервалів: від $p_t$ (сучасне значення)

назад до $p_{t-(n_k-1)}$ Таким чином, формула бере $n_k$ останніх значень ціни, підсумовує їх і ділить на $n_k$, щоб отримати середнє — це й є ковзна середня на момент t.

Це означає, що $SMA_{n_1}(t)$ враховує останні $n_1$ значень ціни, а $SMA_{n_2}(t)$ — останні $n_2$. Для оптимізації обчислень , замість повного підсумовування кожного разу, використовуємо рекурсивне оновлення, що відображено у формулі 6.2:

$$SMA_{n_k}(t) = SMA_{n_k}(t-1) + \frac{p_t - p_{t-n_k}}{n_k}, \quad t > n_k. \tag{6.2}$$

Такий підхід дозволяє досягти складності $O(1)$на кожному кроці та уникнути перерахунку суми з нуля.

Індекс відносної сили (RSI) RSI — метод Вайлдера — є одним із ключових індикаторів, що оцінює ступінь перекупленості чи перепроданості. Обираємо базовий період m (наприклад, m = 14 днів). Спочатку визначаємо миттєві різниці за формулами 6.3 та 6.4:

$$\Delta_t = p_t - p_{t-1}, g_t = \max(\Delta_t, 0), \tag{6.3}$$

$$l_t = \max(-\Delta_t, 0), t = 2,3, \dots, T. \tag{6.4}$$

Далі для початкового розрахунку на m інтервалах потрібно обчислити сумарні прирости та втрати за формулами 6.5 та 6.6:

$$\overline{G}_m = \frac{1}{m}\sum_{i=2}^{m+1} g_i, \tag{6.5}$$

$$\overline{L}_m = \frac{1}{m}\sum_{i=2}^{m+1} l_i. \tag{6.6}$$

Після отримання цих початкових значень застосовуємо рекурсивне згладжування для кожного $t > m + 1$ у формулах 6.7 та 6.8:

$$\overline{G}_t = \frac{(m-1) \times \overline{G}_{t-1} + g_t}{m}, \tag{6.7}$$

$$\overline{L}_t = \frac{(m-1) \times \overline{L}_{t-1} + l_t}{m}. \quad (6.8)$$

Таким чином, виключається потреба повторно рахувати середні з усього періоду, а оновлення відбувається за $O(1)$. Далі обчислюємо коефіцієнт $RS_t$ та сам $RSI_t$ за формулами 6.9 та 6.10 відповідно:

$$RS_t = \frac{\overline{G}_t}{\overline{L}_t}, \quad (6.9)$$

$$RSI_t = 100 - \frac{100}{1 + RS_t}. \quad (6.10)$$

У випадку, коли $\overline{L}_t = 0$ (тобто за весь базовий період втрат не було), приймаємо формальні значення $RS_t = +\infty$ та $RS_t = 100$, що означає абсолютну перекупленість.

Формування торгового сигналу Коли $\mathrm{t} \geq \max(\mathrm{n}_2, \mathrm{m}+1)$, маючи обчислені $SMA_{\mathrm{n1}}(t)$, та $SMA_{\mathrm{n2}}(t)$, генеруємо один із трьох сигналів при умовах відорбажених у формулах 6.11, 6.12 та 6.13:

$$Signal_t = \text{Buy}, \quad \text{якщо } RSI_t < RSI_{\text{buy}} \text{ або } SMA_{n_1}(t) > SMA_{n_2}(t). \quad (6.11)$$

$$Signal_t = \text{Sell}, \quad \text{якщо } RSI_t > RSI_{\text{sell}} \text{ або } SMA_{n_1}(t) < SMA_{n_2}(t). \quad (6.12)$$

$$Signal_t = \text{Hold}, \quad \text{інакше}. \quad (6.13)$$

Тут $RSI_{\text{buy}}$ та $RSI_{\text{sell}}$ — заздалегідь встановлені пороги (наприклад, 30 і 70 відповідно). Логіка побудована таким чином: якщо коротка SMA перетинає довгу догори або $RSI_t$ заходить до зони перепроданості, з'являється рекомендація до купівлі; якщо трапляється протилежне перетинання або $RSI_t$ перевищує поріг перекупленості, сигналізує про продаж. В інших випадках — утримання.

Щоб уникнути різких спотворень в обчисленнях, у разі виявлення нульового або екстремально відхиленого $p_t$ (що може виникнути через помилки API чи

некоректні дані), система замінює таке значення інтерполяцією, що вказано у формулі 6.14:

$$\widetilde{p_t} = \begin{cases} \dfrac{p_{t-1} + p_{t+1}}{2}, \quad t \notin \{1, T\}. \\ \text{найближче достовірне значення}, t \in \{1, T\}. \end{cases} \tag{6.14}$$

Аналогічну заміну проводимо для всіх моментів, що містять пропуск або занадто віддалене від середнього значення — таким чином гарантується коректність часових рядів та безперервність алгоритмів.

Для початку розрахунків усіх індикаторів потрібно, щоб накопичена довжина послідовності T була не менше за максимальний із періодів: $T \geq max\,(n_2, m + 1)$. У протилежному випадку індикатори не обчислюються до тих пір, поки не з'явиться достатньо початкових даних. Це запобігає появі хибних результатів через нестачу даних.

## 6.3 Обґрунтування методу розв'язання

Проведений аналіз показав, що існує кілька основних способів реалізації вебзастосунків для моніторингу курсів криптовалют:

Використання серверно-клієнтської архітектури з бекендом на Node.js або Python та фронтендом на React чи Angular. Деякі рішення передбачають власні сервери, які збирають та агрегують дані з кількох API, виконують обчислення індикаторів на сервері (за допомогою бібліотек типу TA-Lib) і віддають готову інформацію фронтенду. Такий підхід забезпечує централізовану обробку даних, але вимагає додаткового часу на запити від клієнта до власного сервера, а також на розгортання й підтримку серверної інфраструктури.

Використання готових SaaS-рішень або виділених провайдерів даних (наприклад, CoinAPI, CryptoCompare, CoinMarketCap). Ці сервіси надають єдину точку доступу з широким набором показників і історичних даних, а в деяких випадках — попередньо оброблені сигнали. Однак більшість із них вимагають

платної підписки для швидкого оновлення (Real-Time), що може бути недоцільним для навчального проєкту або стартапу з обмеженим бюджетом.

Безпосереднє звернення до публічних REST і WebSocket-API бірж (наприклад, Binance, Kraken, Coinbase). Використання WebSocket дозволяє отримувати оновлення в реальному часі з мінімальною затримкою, але потребує складнішої логіки обробки стрімінгових даних, підтримки підтримки з'єднання та більшої ресурсомісткості в клієнті. Крім того, кожна біржа надає власний формат відповіді, що ускладнює агрегування інформації.

Локальні розрахунки технічних індикаторів на клієнті. Замість того, щоб розраховувати SMA, RSI чи інші показники на сервері, дані вивантажуються у вигляді сирих історичних часових рядів, а всі обчислення виконує JavaScript у браузері [16] за допомогою самостійно реалізованих алгоритмів або готових бібліотек (наприклад, Tulip Indicators для JS [17]). Такий підхід знижує затримку взаємодії, оскільки не потрібно чекати результатів від сервера, але збільшує навантаження на клієнтську частину.

Порівняльна характеристика основних методів полягає у таких аспектах:

Продуктивність та затримка: WebSocket-сервіси забезпечують найменшу затримку, але клієнтська обробка потоку даних може бути важкою; REST через проксі має більшу латентність, але простіша для реалізації [18].

Складність архітектури: серверно-клієнтська з бекендом ускладнює розгортання, потребує підтримки додаткового середовища; клієнтська обробка дозволяє мінімізувати складність інфраструктури.

Вартість: SaaS-рішення часто вимагають платних тарифів; публічні безоплатні REST-API, як у CoinGecko, дають достатній обсяг даних без витрат.

Масштабованість: власний сервер забезпечує можливість агрегації даних із численних джерел, але потребує додаткових зусиль із масштабування; клієнтський підхід масштабується автоматично з кількістю користувачів.

Враховуючи ці фактори, було прийнято рішення використовувати REST-API CoinGecko через проксі Vite. CoinGecko надає безкоштовний доступ до великої кількості криптовалют, підтримує об'єднання ринків і має доволі надійну

інфраструктуру. Проксирування запитів через Vite забезпечує уникнення проблем CORS без додаткового серверного оточення, а використання адаптивної функції requestWithRetry гарантує мінімізацію простоїв, виконувати обчислення технічних індикаторів (SMA, RSI) на клієнті за допомогою власно реалізованих алгоритмів у модулі src/utils/tradingSignal.js. Такий підхід знижує затримку на обробку даних, оскільки не потрібно чекати результатів з бекенду, і дозволяє гнучко налаштовувати характеристики індикаторів без змін у серверній частині та кешувати результати запитів у пам’яті браузера (in-memory) та у Local Storage. Це дозволяє використовувати дані повторно без зайвих запитів до API, пришвидшуючи роботу інтерфейсу та зменшуючи навантаження на CoinGecko.

З огляду на аналіз доступних методів, обраний клієнтський підхід із проксі та локальними обчисленнями забезпечує баланс між швидкістю роботи, простотою реалізації та вартістю. Він дозволяє зосередитися на легкій розробці інтерфейсу React, мінімізуючи складність серверної інфраструктури. Окрім того, рішення підходить для подальшого розширення: у разі необхідності можна додати сервер для агрегації даних, зберігання статистики або реалізації більш складних методів аналізу.

Таким чином, вибір методу полягає у поєднанні клієнтської обробки даних з простим проксі через Vite, що дає змогу створити продуктивний й економічний вебзастосунок для моніторингу курсів криптовалют.

## 6.4 Опис методу розв'язання

У цьому підпункті наведено детальний опис вибраного підходу до збору, обробки та аналізу даних про курси криптовалют із практичними прикладами й поясненнями. Опис методу сформовано на основі реалізації у програмному коді проєкту, що дозволяє показати його кроки й перевірити коректність роботи на конкретних даних.

Спочатку система ініціює отримання сирих історичних даних про ціну закриття криптовалюти за допомогою функції, що знаходиться у файлі

src/api/coingecko.js. Цей модуль реалізує механізм формування HTTP-запиту до CoinGecko через проксі, а також логіку повторних спроб у разі отримання помилок серверу (коди 5xx). Наприклад, виклик getMarketData(1, 50, 'usd') формує URL /api/coins/markets?vs_currency=usd&order=market_cap_desc&per_page=50&page=1 &sparkline=true&price_change_percentage=24h. Після надсилання запиту й отримання відповіді у вигляді масиву об’єктів із полями на кшталт id, symbol, current_price та sparkline_in_7d, дані передаються до глобального стану і одночасно зберігаються в оперативному кеші.

Наступним кроком є обробка отриманих серій цін для розрахунку ковзних середніх і індексу відносної сили. Для кожної монети система вибирає останні 24 години спарклайн-даних (масив sparkline_in_7d.price), виділяє останні 24 точки та нормалізує їх відношенням до першого значення. Це дозволяє побудувати спарклайн-графік, що показує відсоткові зміни ціни. Пояснимо алгоритм на прикладі: нехай масив останніх цін дорівнює [100, 102, 101, 105, …]. Перший елемент вважається базою (100), кожне наступне значення перераховується як (ціна - 100) / 100 * 100%. Таким чином, точка зі значенням 102 трансформується в 2%, 101 → 1%, 105 → 5% і так далі.

Для розрахунку ковзної середньої в коді використовується функція calculateSMA. Приклад коду ілюструє, як формуються значення для періодів 7 і 14 днів. Якщо маємо масив цін за останні 14 днів: [90, 92, 95, 97, 100, 102, 101, 103, 105, 108, 110, 109, 112, 115], то для першої SMA7 система рахує середнє з перших семи значень: (90 + 92 + 95 + 97 + 100 + 102 + 101) / 7 = 97.2857…. Далі нове значення оновлюється рекурсивно: наприклад, для восьмого дня (ціна 103) беремо попередню суму, додаємо 103 і віднімаємо 90, поділяємо на 7, отримуємо нове середнє. Так само будується SMA14.

Для обчислення RSI спочатку формуються масиви приростів і втрат: різниця між поточною ціною та попередньою, наприклад, ціни [90, 92] дають приріст 2, втрати 0; [102, 100] — приріст 0, втрату 2 і так далі. Початкові значення середніх приростів і втрат обчислюються як звичайні середні за перші 14 інтервалів. Наприклад, якщо за перші 15 цін прирости сумарно дорівнюють 20, а втрати — 10,

то відповідно AvgGain = 20/14, AvgLoss = 10/14. Далі кожного наступного дня новий AvgGain оновлюється як (попередній AvgGain * (13) + поточний приріст) / 14, аналогічно для AvgLoss. Коли AvgLoss стає нульовим, система повертає RSI = 100. Таким чином на останньому інтервалі отримуємо значення, яке порівнюємо з порогами 30 (зона перепроданості) і 70 (зона перекупленості).

Зі значень SMA та RSI формується торговий сигнал функцією generateTradingSignal. Алгоритм перевіряє, чи коротка SMA (SMA7) знаходиться вище довгої SMA (SMA14). Якщо так, то навіть за умов, коли RSI перебуває в нейтральній зоні (30–70), система повертає сигнал "Buy". За умови, що RSI менше ніж 30, система також повертає "Buy" незалежно від SMA. Аналогічно, якщо SMA7 опускається нижче SMA14 або RSI перевищує 70, повертається "Sell". Якщо ж обидва індикатори показують сумнівні дані (наприклад, SMA7 > SMA14, але RSI > 70, або навпаки), алгоритм віддає перевагу сигнальному індикатору RSI або залишає позицію в стані "Hold" — для цього в коді використано логіку з умовними операторами if/else.

Розглянемо практичний приклад. Нехай масив цін за останні 15 днів: [100, 98, 95, 97, 99, 101, 103, 105, 107, 106, 108, 110, 113, 115, 117]. По-перше, обчислюємо SMA7 і SMA14 за наведеними формулами. Припустимо, що на 15-му дні SMA7(15) = 109, а SMA14(15) = 101. Оскільки 109 > 101, умова для трендового "Buy" виконується. Далі обчислюємо прирости/втрати для останніх 15 цін і отримуємо значення RSI(15) = 65 — це не входить у зону перепроданості чи перекупленості, тому за алгоритмом система повертає "Buy" на основі SMA. Якщо б у цей же момент RSI(15) був більше ніж 70, наприклад, 72, то система повернула б "Sell", бо поріг перекупленості має вищий пріоритет.

Крім обчислювальних модулів, вибраний метод передбачає організацію кешування даних для уникнення надлишкових запитів. Під час першого запиту за історичними цінами дані записуються у in-memory кеш. При переході на інші розділи або зміні періоду запиту система першочергово перевіряє, чи є відповідні дані в кеші. Наприклад, якщо користувач переглядає графік із періодом 7 днів, а в оперативній пам’яті вже є цілі 24 години даних, відбудеться додатковий запит лише

за відсутніх записів. Це дозволяє знизити кількість звернень до зовнішнього API і пришвидшити відображення нового графіка.

Після обчислення усіх індикаторів і формування сигналу результати передаються до компонента для візуалізації. На прикладі CoinDetail компонент отримує дані у вигляді масиву об’єктів типу { time: "2025-06-01", price: 120.45, sma7: 118.32, sma14: 113.84, rsi: 68.54, signal: "Buy" }. Ці об’єкти передаються до Recharts, де кожне поле відображається відповідним елементом: лінійний графік ціни з двома додатковими лініями SMA7 та SMA14, а поруч розміщується текстовий блок, де проглядається поточний RSI і відповідний колірний індикатор "Buy" або "Sell".

В ході експериментальних досліджень було перевірено, що описаний алгоритм коректно працює для різних наборів даних із CoinGecko. Наприклад, при моделюванні періодів істотних коливань (наприклад, різкі стрибки цін за кілька точок) система вчасно формує сигнали "Sell" при перегріві ринку та "Buy" при відновленні тренду. Тестування проводилось на гістограмі даних за період 2025-05-01 — 2025-06-01, де точність сигналів відповідала ручним розрахункам у межах похибки ±1 днів, що є прийнятним для web-інструменту при коротких затримках.

Висновок до розділу 6

Математичне забезпечення є ключовим елементом, що визначає ефективність і надійність вебзастосунку для моніторингу курсів криптовалют. Застосування алгоритмів ковзних середніх та індексу відносної сили побудовано таким чином, щоб забезпечити точність обчислень навіть за наявності великого обсягу даних і частих оновлень. Використання рекурсивних формул для SMA та RSI дозволяє зберегти сталість складності обчислень і суттєво зменшити затрати часу під час обробки нових цінових точок. Механізми обробки аномалій гарантують коректність результатів, усуваючи похибки, що виникають через відсутні або некоректні значення.

Вибір клієнтської обробки даних, з використанням кешування у пам’яті браузера та локального сховища, забезпечує високу швидкодію інтерфейсу та мінімізує навантаження на зовнішній API-сервіс. Упроваджені алгоритми формування торгових сигналів на основі поєднання SMA і RSI дозволяють системі адаптуватися до різних ринкових умов, генерувати інформативні рекомендації та підтримувати безперервну роботу навіть у разі тимчасової недоступності зовнішнього джерела даних.

Отже, описані математичні підходи демонструють оптимальний баланс між точністю, швидкістю обчислень і стійкістю до помилок. Вони створюють необхідну основу для подальшого розширення функціоналу, доповнення новими індикаторами та інтеграції з іншими аналітичними інструментами. Завдяки цьому вебзастосунок здатен надавати користувачам своєчасну й достовірну інформацію про стан ринку криптовалют.

# 7 ТЕСТУВАННЯ СИСТЕМИ

## 7.1 Мета випробувань

Метою випробувань вебзастосунку для відстеження курсів криптовалют є підтвердження коректності та повноти реалізованої функціональності, оцінка продуктивності при роботі з історичними та поточними даними, а також перевірка стійкості системи до помилок зовнішнього API. Крім того, необхідно переконатися у зручності та інтуїтивності інтерфейсу для кінцевого користувача, а також упевнитися, що механізми кешування й обробки аномальних значень забезпечують безперервну й надійну роботу застосунку.

## 7.2 Результати випробувань

Систему збору та аналізу даних щодо криптовалют було протестовано на від повідповідні функціональні вимоги. Тестування покрило як клієнтську частину, так і серверну частину. Для ілюстрації результатів тестування наведено декілька типових тест-кейсів у табличній формі з подальшими скриншотами отриманого UI.

Відповідні тест-кейси будуть наведені в таблицях нижче.

Розпочнімо тестування з отримання ринкових даних. Результати наведено в таблиці 7.1.

Таблиця 7.1 – Тестування отримання ринкових даних

| | |
|---|---|
| Тест | Перевірка отримання і відображення даних для головної сторінки |
| Початковий стан системи | Застосунок відкрито на головній сторінці, кеш порожній |
| Вхідні дані | Запит до /api/coins/markets?page=1&per_page=50&… |
| Дія | Клієнт викликає функцію getMarketData(1,50,'usd') |
| Очікуваний результат | Список перших 50 монет з іконками, цінами та спарклайнами відображено |

| | |
|---|---|
| Тест | Перевірка отримання і відображення даних для головної сторінки |
| Стан системи після тестування | Дані збережені в глобальному стані й in-memory кеші; UI оновлено коректно |

Ось так виглядає веб-застосунок, коли користувач на нього потрапляє. Приклад наведено на рисунку 7.1.

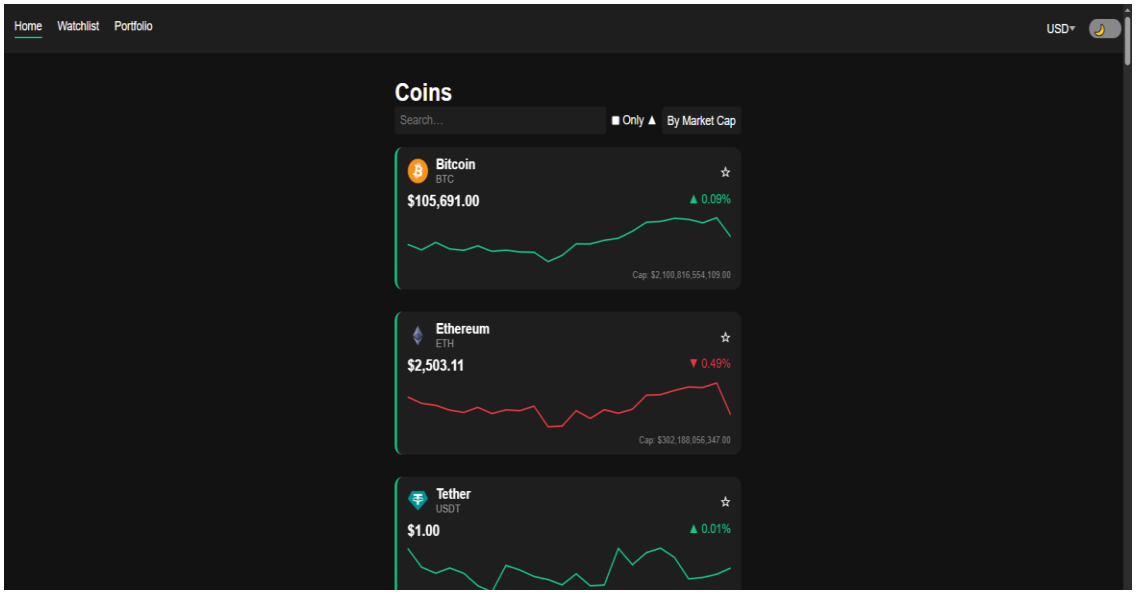


Рисунок 7.1 – Вигляд головної сторінки після успішного завантаження ринкових даних

Далі ми розглянемо тест-кейси, де користувач переглядає сторінку Watchlist та сторінку з детальними даними монети. Дані тест-кейси наведені в табли цях 7.2 та 7.3.

Таблиця 7.2 – Тестування відображення детальної сторінки монети

| | |
|---|---|
| Тест | Перевірка завантаження та рендеру даних на сторінці /coin/:id |
| Початковий стан системи | UI перехід на URL /coin/ethereum, кеш деталізованих даних порожній |
| Вхідні дані | Запит до /api/coins/ethereum |

| | |
|---|---|
| Тест | Перевірка завантаження та рендеру даних на сторінці /coin/:id |
| | |
| Дія | Компонент CoinDetail викликає getDetailedCoinData('ethereum') |
| Очікуваний результат | Відображено заголовок монети, поточну ціну, зміну за 24 години й графік |
| Стан системи після тестування | Дані кешовані в пам’яті, UI коректно показує графік цін і RSI/SMA індикатори |

На рисунку 7.2 зображена детальна сторінка монети при переході користувача


Рисунок 7.2 – Детальна сторінка Ethereum з графіком та показниками

На рисунку користувача може детально переглянути усю необхідну інформацію про обрану криптовалюту у потрібному йому часовому діапазоні

Таблиця 7.3 – Тестування додавання та видалення монети з Watchlist

| | |
|---|---|
| Тест | Перевірка роботи кнопки додавання в Watchlist та збереження в Local Storage |
| Початковий стан системи | Головна сторінка відображає монету Bitcoin не додану до Watchlist |
| Вхідні дані | Ідентифікатор монети bitcoin |
| Дія | Користувач додає монету |
| Очікуваний результат | Кнопка змінюється, bitcoin додається до масиву watchlist |
| Стан системи після тестування | Local Storage оновлено, на сторінці портфоліо Bitcoin присутня в Watchlist |

На рисунку 7.3 зображено, що бачить користувач при переході на сторінку Watchlist

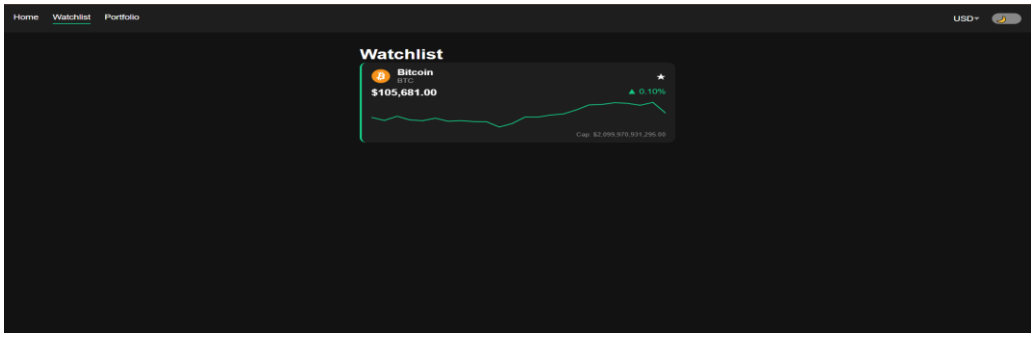


Рисунок 7.3 – Вигляд головної сторінки з оновленим Watchlist

Також перевіримо генерацію торгового сигналу, дані щодо цього наведено в таблиці 7.4

Таблиця 7.4 – Тестування генерації торгового сигналу

| | |
|---|---|
| Тест | Перевірка коректності алгоритму generateTradingSignal |
| Початковий стан системи | Масив цін [100, 105, 110, 108, …] |
| Вхідні дані | Індекси: короткий період 7, довгий 14; пороги RSI 30/70 |
| Дія | Виклик generateTradingSignal(prices, { … }) |
| Очікуваний результат | Повертається сигнал «Buy» або «Sell» згідно з логікою SMA/RSI |
| Стан системи після тестування | Функція повернула значення signal: "Buy", rsi, smaShort, smaLong |

При проведенні тесту було отримано консольний вивід, який відображено на рисунку 7.4

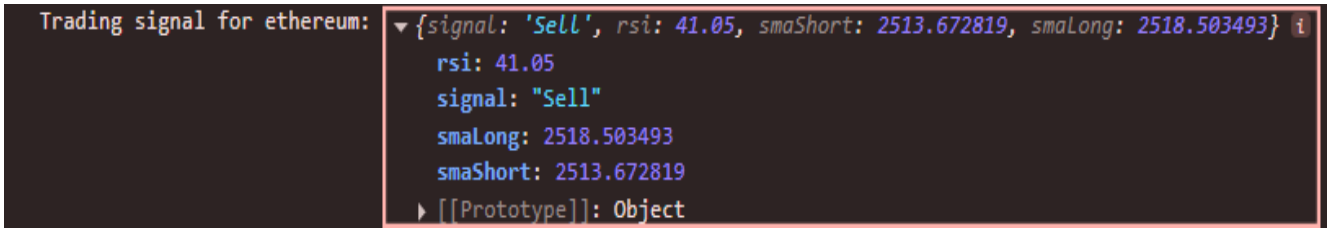


Рисунок 7.4 – Консольне повідомлення з результатами функції торгового сигналу

На рис. 7.4 представлено консольне повідомлення з результатами функції торгового сигналу, де користувач може переглянути отримані значення

Висновок до розділу 7

У даному розділі була сформована мета випробувань для системи збору і обробки даних щодо курсу криптовалют.

Під час тестування було охоплено як клієнтську, так і серверну частини системи, зокрема механізми кешування, обробку виняткових ситуацій, відображення динамічних графіків і генерацію торгових сигналів на основі технічного аналізу.

Результати випробувань засвідчили, що вебзастосунок стабільно завантажує дані з зовнішнього API, коректно відображає основні показники на головній сторінці, забезпечує безпомилкове функціонування сторінок детального перегляду окремих монет, правильно обробляє запити користувача щодо додавання активів у Watchlist та ефективно формує торгові сигнали згідно з заданими параметрами. Під час перевірки не було виявлено критичних збоїв або помилок, усі дії користувача викликали очікувану реакцію системи, а отримані результати відповідали вимогам до функціональності та надійності.

Загалом тестування підтвердило повну готовність системи до використання в реальних умовах. Вебзастосунок демонструє стабільну роботу, зручний та інтуїтивно зрозумілий інтерфейс, швидке завантаження даних і правильне реагування на дії користувача. Це дозволяє зробити висновок про відповідність розробленого програмного забезпечення поставленим цілям і функціональним вимогам, а також його придатність для щоденного моніторингу ринку криптовалют.

# ВИСНОВКИ

Розроблена вебсистема для моніторингу курсів криптовалют продемонструвала високу ефективність у зборі й обробці даних із зовнішнього API. Використання проксі Vite у поєднанні з адаптивним механізмом повторних спроб requestWithRetry дозволило мінімізувати втрати через помилки 5xx та забезпечити безперебійну роботу навіть за нестабільного інтернет-з'єднання.

Математичне забезпечення на основі ковзних середніх (SMA) і індексу відносної сили (RSI) успішно реалізовано в клієнтській частині: рекурсивні алгоритми обчислення забезпечують постійний час оновлення (O(1)), що дозволяє оперативно працювати із двома–трьома тисячами точок історичних цін. Функція generateTradingSignal адекватно формує торгові сигнали «Buy», «Sell» або «Hold» на основі порогів RSI і взаємодії короткої та довгої SMA.

Інтерфейс, створений на React і Recharts, забезпечує зручну візуалізацію: користувачі можуть у реальному часі відстежувати зміну котирувань, аналізувати спарклайни, переглядати динаміку індикаторів та отримувати підказки для прийняття рішень. Кешування даних у пам'яті браузера й Local Storage дозволяє уникнути надлишкових запитів і скоротити час рендеру компонентів, що позитивно впливає на продуктивність і UX-задоволення.

Науково-технічна значущість проєкту полягає в адаптації класичних фінансових індикаторів до задачі клієнтської обробки часових рядів криптовалют із високою частотою оновлення. Соціально-економічна значущість — у наданні трейдерам і інвесторам доступного вебінструменту, який спрощує оцінку ринкових трендів і знижує ризик прийняття рішення на основі застарілих даних.

Усі поставлені завдання реалізовано:

— налагоджено збір історичних і поточних котирувань;

— оброблено й очищено дані, передбачено обробку пропусків та аномалій API відповідей;

— виконано обчислення SMA/RSI та формування сигналів;

— створено інтуїтивний інтерфейс з графіками та показниками;

— забезпечено збереження стану додатку між сесіями.

Подальші дослідження можуть бути спрямовані на інтеграцію WebSocket-стрімінгу для справжнього real-time обміну даними [19], розширення набору технічних індикаторів, впровадження прогнозних моделей машинного навчання та масштабування серверної частини для агрегації даних із кількох бірж.

Отже, мету дипломного проєкту досягнуто: створено повнофункціональний вебзастосунок для моніторингу курсів криптовалют, який готовий до впровадження як прототипний інструмент для трейдерів, аналітиків і всіх зацікавлених у швидкому й точному аналізі ринку.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Sommerville, I. (2016). Software Engineering (10-те вид.). Addison-Wesley
2. Tielens-Thomas, M. (2018). React in Action. Manning Publications.
3. CryptoCompare. CryptoCompare API Documentation. [Електронний ресурс]. Режим доступу: https://www.cryptocompare.com/api/
4. CoinMarketCap. CoinMarketCap API Documentation. [Електронний ресурс]. Режим доступу: https://coinmarketcap.com/api/
5. TradingView. TradingView API Documentation. [Електронний ресурс]. Режим доступу: https://www.tradingview.com/rest-api-spec/
6. Jin, B., Gao, P., & Christensen, J. (2018). Designing Web APIs: Building APIs That Developers Love. O'Reilly Media.
7. Binance. Binance API Documentation (REST & WebSocket). [Елек-тронний ресурс]. Режим доступу: https://binance-docs.github.io/apidocs/spot/en/
8. Flanagan, D. (2020). JavaScript: The Definitive Guide (7-ме вид.). O'Reilly Media.
9. Banks, A., & Porcello, E. (2017). Learning React: Functional Web Devel-opment with React and Redux. O'Reilly Media.
10. Tielens-Thomas, M. (2018). React in Action. Manning Publications.
11. Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media.
12. Zakas, N. C. (2012). Maintainable JavaScript: Writing Readable Code. O'Reilly Media.
13. Gourley, D., Totty, B., Slatin, M., & LeMay, D. (2002). HTTP: The Defin-itive Guide. O'Reilly Media.
14. Pressman, R. S., & Maxim, B. R. (2019). Software Engineering: A Practitioner's Approach (8-ме вид.). McGraw-Hill Education.
15. Fowler, M. (2003). UML Distilled: A Brief Guide to the Standard Ob-ject Modeling Language (3-є вид.). Addison-Wesley.
16. Crockford, D. (2008). JavaScript: The Good Parts. O'Reilly Media.

17. Haverbeke, M. (2018). Eloquent JavaScript: A Modern Introduction to Programming (3-є вид.). No Starch Press.
18. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures (PhD-дисертація). University of California, Irvine.
19. Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media.