ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

бакалавра
(освітньо-кваліфікаційний рівень)

на тему *«Розроблення веб-сервісу моніторингу криптовалют»*

Виконав: здобувач *4* року навчання
групи *4ПР1*
напряму підготовки (спеціальності)
121-Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Горьковий Віталій Олександрович
(підпис, прізвище та ініціали)

Керівник *Доровський В.О.*
(підпис, прізвище та ініціали)

Рецензент *к.т.н., доцент Григорова А.А.*
(прізвище та ініціали)

Нормоконтролер *Комісаров О. С.*
(підпис, прізвище та ініціали)

Хмельницький – 2026

ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРЗВТЕТ

Інститут, факультет, відділення *Інформаційних технологій та дизайну*
Кафедра, циклова комісія *Програмних засобів і технологій*
Освітньо-кваліфікаційний рівень *бакалавр*
Освітньо-професійна підготовка *Програмна інженерія*
(шифр і назва)
Спеціальність *121-Інженерія програмного забезпечення*
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри *програмних засобів і технологій*
_____ О.Є. Огнєва
«___» _____ 2026 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА

Горькового Віталія Олександровича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) *Розроблення веб-сервісу моніторингу криптовалют*

керівник проєкту (роботи) *Доровський Володимир Олексійович, д.т.н., професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «16» січня 2026 року №82-с

2. Строк подання здобувачем проєкту(роботи) *10.06.2026*

3. Вихідні дані до проєкту (роботи) *літературні та періодичні джерела, матеріали переддипломної практики*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області моніторингу криптовалют,
проєктування веб-сервісу моніторингу криптовалют,
розроблення веб-сервісу моніторингу криптовалют,
тестування та впровадження веб-сервісу моніторингу криптовалют

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
10 таблиць, 54 рисунки

6. Консультанти розділів проєкту (роботи)

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

7. Дата видачі завдання *14.01.2026*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів проєкту (роботи)	Примітка
1	Отримання завдання	14.01.2026	Виконано
2	Підбір літератури	17.01.2026-21.01.2026	Виконано
3	Аналіз предметної області	22.01.2026-26.01.2026	Виконано
4	Розробка та обґрунтування завдання	27.01.2026-12.02.2026	Виконано
5	Розробка концептуальної моделі	13.02.2026-19.02.2026	Виконано
6	Моделювання та проєктування системи	20.02.2026-02.03.2026	Виконано
7	Розробка чат-бота	03.03.2026-15.04.2026	Виконано
8	Тестування чат-бота	16.04.2026-20.04.2026	Виконано
9	Оформлення пояснювальної записки	21.04.2026-02.05.2026	Виконано
10	Захист кваліфікаційної роботи	23.06.2026	Виконано

Здобувач _____ *Горьковий В.О.*
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ *Доровський В.О.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра: 90 сторінок, 54 рисунки, 10 таблиць, 1 додаток, 20 джерел.

Мета роботи:

Розроблення веб-сервісу моніторингу криптовалют, який забезпечує автоматизований збір ринкових даних із криптовалютних бірж, їх зберігання, обробку, аналіз і візуалізацію, спрощуючи процес їх аналізу для широкого кола користувачів.

Об'єкт дослідження:

Процес моніторингу криптовалютного ринку та обробки ринкових даних цифрових активів.

Предмет дослідження:

Методи, технології та програмні засоби розроблення веб-сервісів моніторингу криптовалют.

Інструменти та засоби розробки:

Для розробки веб-сервісу було використано сучасний стек технологій: Node.js та Express.js для реалізації серверної частини, MongoDB для організації зберігання ринкових даних, CCXT для взаємодії з API криптовалютних бірж, Vue.js для створення SPA-інтерфейсу, RESTful API для взаємодії між компонентами системи, Chart.js та TradingView Widget для візуалізації ринкових даних.

Результати роботи:

Результатом роботи є веб-сервіс, що забезпечує автоматичний збір котирувань із бірж, централізоване зберігання даних, роботу з RESTful API, графічне відображення ринкової інформації, фільтрацію та сортування криптовалют, порівняння цін між торговими платформами. Сферою використання веб-сервісу є підтримка трейдерів, інвесторів і аналітиків у прийнятті рішень на ринку цифрових активів.

Новизна рішення:

Розроблено оригінальний механізм паралельного опитування API декількох криптовалютних бірж через уніфікований CCXT-клієнт із щохвилинним циклом оновлення, реалізовано функцію одночасного порівняння ціни однієї торгової пари на різних майданчиках.

Практична цінність:

Створено працездатний веб-сервіс моніторингу криптовалют, який може використовуватися трейдерами, інвесторами та аналітиками для оперативного отримання та аналізу інформації про цифрові активи.

КЛЮЧОВІ СЛОВА: *ВЕБ-ЗАСТОСУНОК, КРИПТОВАЛЮТА, МОНІТОРИНГ, SPA, АНАЛІТИКА, КРИПТОБІРЖА, ЦІНА, ОБСЯГ, NODE.JS, EXPRESS.JS, MONGODB, REST API.*

АНОТАЦІЯ

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

Робота присвячена розробленню веб-сервісу моніторингу криптовалют, призначеного для автоматизованого збору, обробки, зберігання та візуалізації ринкових даних цифрових активів.

У першому розділі проведено аналіз предметної області криптовалютного моніторингу, досліджено сучасні веб-сервіси аналітики криптовалютного ринку та визначено їх основні переваги й недоліки. На основі проведеного аналізу сформульовано функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.

У другому розділі обґрунтовано вибір технологічного стеку та спроєктовано триланкову архітектуру веб-сервісу, яка включає модуль автоматичного збору ринкових даних, RESTful API та клієнтський SPA-застосунок.

У третьому розділі для реалізації серверної частини використано Node.js та Express.js. Зберігання ринкових даних організовано із використанням документоорієнтованої бази даних MongoDB. Для взаємодії з API криптовалютних бірж використано бібліотеку CCXT, яка забезпечує уніфікований механізм отримання котирувань та обсягів торгів із різних торгових платформ. Веб-сервіс здійснює автоматичний збір ринкових даних, централізоване зберігання інформації, роботу через RESTful API, забезпечує SPA-інтерфейс користувача, механізми фільтрації та сортування даних, графічне відображення котирувань криптовалют, порівняння цін між криптовалютними біржами. Особливу увагу приділено реалізації механізмів обробки ринкових даних у режимі реального часу, побудові адаптивного користувацького інтерфейсу та забезпеченню масштабованості системи.

У четвертому розділі проведено модульне, інтеграційне, API- та UI-тестування веб-сервісу. Результати тестування підтвердили коректність функціонування програмних модулів, стабільність взаємодії між компонентами системи та готовність програмного забезпечення до впровадження.

Практичне значення отриманих результатів полягає у створенні працездатного веб-сервісу моніторингу криптовалют, який може використовуватися трейдерами, інвесторами та аналітиками для оперативного отримання та аналізу ринкової інформації.

ABSTRACT

The bachelor's thesis consists of an introduction, four sections, conclusions, a list of sources used and appendices.

The thesis is devoted to the development of a cryptocurrency monitoring web service designed for automated collection, processing, storage and visualization of digital asset market data.

In the first chapter, the subject area of cryptocurrency monitoring is analyzed, modern cryptocurrency market analytics web services are studied and their main advantages and disadvantages are identified. Based on the analysis, functional and non-functional requirements for the developed software are formulated.

In the second chapter, the choice of a technological stack is justified and a three-tier architecture of the web service is designed, which includes an automatic market data collection module, a RESTful API and a client SPA application.

In the third chapter, Node.js and Express.js were used to implement the server part. Market data storage was organized using the MongoDB document-oriented database. The CCXT library was used to interact with the API of cryptocurrency exchanges, which provides a unified mechanism for obtaining quotes and trading volumes from various trading platforms. The web service automatically collects market data, centralized information storage, works via RESTful API, provides an SPA user interface, data filtering and sorting mechanisms, graphical display of cryptocurrency quotes, and price comparison between cryptocurrency exchanges. Particular attention is paid to the implementation of mechanisms for processing market data in real time, building an adaptive user interface, and ensuring system scalability.

In the fourth chapter, modular, integration, API and UI testing of the web service was carried out. The test results confirmed the correct functioning of the

software modules, the stability of interaction between the system components and the readiness of the software for implementation.

The practical significance of the results obtained lies in the creation of a functional web service for monitoring cryptocurrencies, which can be used by traders, investors and analysts for the prompt receipt and analysis of market information.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment (інтегроване середовище розробки)

API – Application Programming Interface (програмний інтерфейс застосунку)

SPA – Single Page Application (односторінковий застосунок)

DDD – Domain-Driven Design (проєктування на основі предметної області)

FSD – Feature-Sliced Design (функціонально-орієнтована архітектура фронтенду)

REST – Representational State Transfer (стиль архітектури веб-сервісів)

CCXT – CryptoCurrency eXchange Trading Library

ODM – Object Data Modeling

JWT – JSON Web Token (токен авторизації)

CORS – Cross-Origin Resource Sharing

HTTP – HyperText Transfer Protocol

JSON – JavaScript Object Notation

HMR – Hot Module Replacement (гаряча заміна модулів)

CRUD – Create, Read, Update, Delete (базові операції з даними)

ВСТУП

Стрімкий розвиток ринку цифрових активів зумовив значне зростання попиту на інструменти, здатні в автоматизованому режимі відстежувати котирування та торгові обсяги на різних криптовалютних майданчиках. За даними провідних аналітичних ресурсів, сукупний щомісячний обіг на криптовалютних біржах у 2025 році перевищує два трильйони доларів США. За таких умов наявність зручного, швидкодіючого та багатофункціонального веб-сервісу для відстеження ринкової кон'юнктури стає необхідністю як для досвідчених трейдерів, так і для початківців, що лише починають вивчати особливості торгівлі цифровими активами.

У сучасних умовах стрімкого розвитку цифрових технологій та фінансових інформаційних систем криптовалюти стали важливим елементом глобального економічного середовища. Ринок цифрових активів характеризується високою динамічністю, значною волатильністю та великою кількістю торгових платформ, що обумовлює необхідність оперативного отримання, обробки та аналізу ринкових даних.

Зростання популярності криптовалют призвело до появи великої кількості криптовалютних бірж, аналітичних платформ та сервісів моніторингу ринку. Для трейдерів, інвесторів та фінансових аналітиків важливого значення набуває можливість отримання актуальної інформації щодо котирувань, обсягів торгів, ринкової капіталізації та змін вартості цифрових активів у режимі реального часу.

Сучасні веб-сервіси моніторингу криптовалют, такі як CoinMarketCap, CoinGecko та TradingView, забезпечують широкий спектр функціональних можливостей, проте мають ряд недоліків, серед яких складність інтерфейсу, обмеження безкоштовного доступу до API, недостатня гнучкість персоналізації та залежність від сторонніх сервісів. Крім того, значна частина існуючих платформ орієнтована на комерційне використання та не

забезпечує можливості створення власних механізмів аналітики та обробки ринкових даних.

У зв’язку з цим актуальною задачею є розроблення власного сучасного масштабованого веб-сервісу моніторингу криптовалютного ринку, який забезпечуватиме автоматизований збір, обробку, зберігання та візуалізацію ринкової інформації із підтримкою сучасного SPA-інтерфейсу та RESTful API, її аналітичну обробку та зручне відображення для користувачів.

Ринок пропонує ряд існуючих платформ – CoinMarketCap, CoinGecko, TradingView, – проте кожна з них має суттєві обмеження: затримки у відображенні даних, відсутність можливості зіставлення цін між біржами в реальному часі, перевантажений інтерфейс, що ускладнює роботу для новачків. Зазначені недоліки підтверджують доцільність розроблення власного веб-сервісу, орієнтованого на конкретну цільову аудиторію та побудованого на сучасних технологіях.

Об’єктом дослідження є процес моніторингу криптовалютного ринку та обробки ринкових даних цифрових активів.

Предметом дослідження є методи, технології та програмні засоби розроблення веб-сервісів моніторингу криптовалют.

Метою роботи є розроблення веб-сервісу моніторингу криптовалют, який забезпечує автоматизований збір ринкових даних із криптовалютних бірж, їх зберігання, обробку, аналіз і візуалізацію, спрощуючи процес їх аналізу для широкого кола користувачів.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести аналіз предметної області моніторингу криптовалют;
- дослідити існуючі веб-сервіси криптовалютної аналітики;
- визначити функціональні та нефункціональні вимоги до системи;
- обґрунтувати вибір технологічного стеку;

- спроектувати архітектуру веб-сервісу;
- реалізувати модуль збору ринкових даних;
- створити RESTful API;
- реалізувати SPA-клієнт для відображення інформації;
- реалізувати механізми фільтрації, сортування та аналітики;
- провести тестування та впровадження програмного забезпечення.

Методи дослідження. У роботі застосовано методи порівняльного аналізу існуючих платформ, об'єктно-орієнтованого та компонентного проєктування, асинхронного програмування, а також методи модульного та функціонального тестування.

Для реалізації веб-сервісу було використано сучасний стек технологій:

- Node.js та Express.js – для реалізації серверної частини;
- MongoDB – для організації зберігання ринкових даних;
- CCXT – для взаємодії з API криптовалютних бірж;
- React або Vue.js – для створення SPA-інтерфейсу;
- RESTful API – для взаємодії між компонентами системи;
- Chart.js та TradingView Widget – для візуалізації ринкових даних.

Наукова новизна визначається тим, що в межах роботи розроблено оригінальний механізм паралельного опитування API декількох криптовалютних бірж через уніфікований CCXT-клієнт із щохвилинним циклом оновлення, реалізовано функцію одночасного порівняння ціни однієї торгової пари на різних майданчиках, а також забезпечено мультимовний (українська, англійська) адаптивний інтерфейс.

Практичне значення отриманих результатів полягає у створенні працездатного веб-сервісу моніторингу криптовалют, який може

використовуватися трейдерами, інвесторами та аналітиками для оперативного отримання та аналізу інформації про цифрові активи.

Розроблений веб-сервіс забезпечує:

- автоматичний збір котирувань із бірж;
- централізоване зберігання даних;
- RESTful API;
- графічне відображення ринкової інформації;
- фільтрацію та сортування криптовалют;
- порівняння цін між торговими платформами.

Структура кваліфікаційної роботи складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі проведено аналіз предметної області та існуючих програмних рішень. У другому розділі розглянуто проєктування архітектури веб-сервісу та структури системи. У третьому розділі описано процес програмної реалізації веб-сервісу, REST API та SPA-застосунку. У четвертому розділі наведено результати тестування, впровадження та оцінки ефективності розробленого програмного забезпечення.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОНІТОРИНГУ КРИПТОВАЛЮТ

1.1. Загальна характеристика предметної області

У сучасних умовах розвитку цифрової економіки криптовалюти стали важливим елементом фінансових систем та інвестиційної діяльності. Ринок цифрових активів характеризується високою волатильністю, значною кількістю торгових платформ та постійними змінами ринкових показників. Це обумовлює необхідність створення інформаційних систем, здатних забезпечувати оперативний моніторинг котирувань, обсягів торгів та ринкових тенденцій.

Криптовалютний ринок функціонує цілодобово та включає:

- централізовані криптовалютні біржі;
- децентралізовані торгові платформи;
- сервіси аналітики;
- агрегатори ринкових даних;
- системи технічного аналізу.

Основними учасниками ринку є:

- трейдери;
- інвестори;
- аналітики;
- фінансові компанії;
- криптовалютні біржі.

Для ефективного прийняття торгових рішень користувачам необхідно отримувати:

- актуальні котирування;
- зміни цін у реальному часі;
- обсяги торгів;
- інформацію про ринкову капіталізацію;

- аналітичні графіки;
- порівняння цін між біржами.

Схема взаємодії учасників криптовалютного ринку, яка описує верхньорівневу схему взаємодії учасників криптовалютного ринку та потоків ринкових даних, представлена на рис. 1.1.

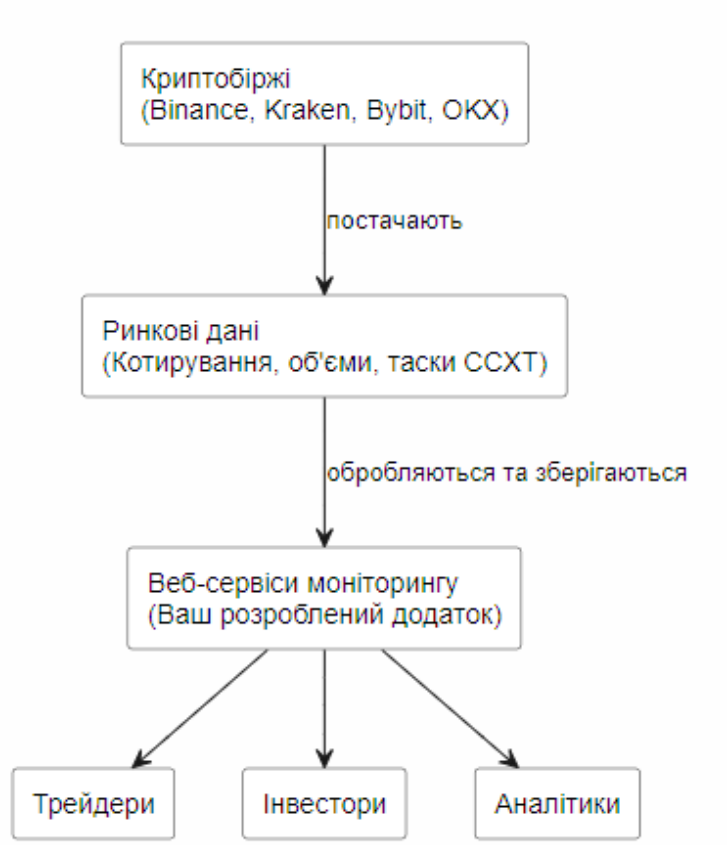

Рисунок 1.1. Схема взаємодії учасників криптовалютного ринку

Як показано на рис. 1.1, криптобіржі є зовнішніми системами, а трейдери, інвестори й аналітики – це живі люди (ролі користувачів), які взаємодіють з інтерфейсом веб-застосунку.

Учасників ринку можна визначити наступним чином.

Постачальники даних (криптобіржі): зовнішній рівень системи. Бекенд-модуль збирача даних (Data Collector) через уніфікований протокол CCXT підключається до API бірж (Binance, Bybit тощо) для отримання актуальних стеків котирувань.

Ринкові дані: сирі потоки даних, які проходять етап нормалізації та валідації у Node.js сервісі, після чого структуруються та записуються в базу даних MongoDB.

Вебсервіс моніторингу виступає центральним ядром агрегації. REST API Express.js забезпечує авторизацію користувачів та віддає підготовлені масиви котирувань на фронтенд, де SPA-додаток будує таблиці та рендерить інтерактивні графіки.

Цільова аудиторія: залежно від потреб, трейдери використовують сервіс для миттєвого відслідковування цін через MarketTable, інвестори формують власні списки відслідковування через функціонал Watchlist, а аналітики використовують історичні графіки та блок аналітики для довгострокового прогнозування ринку.

Учасників ринку можна зобразити також у вигляді діаграми варіантів (рис. 1.2).

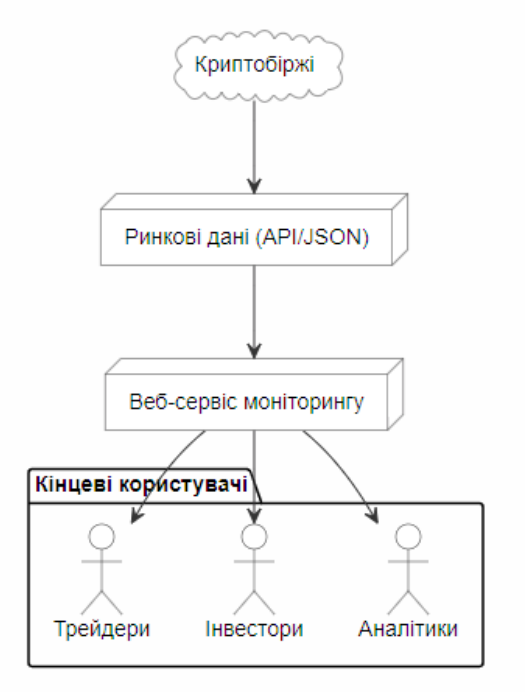

Рисунок 1.2. Діаграма варіантів взаємодії учасників криптовалютного ринку

Особливістю предметної області є необхідність роботи з великими обсягами динамічних даних та підтримка оперативного оновлення інформації.

1.2. Характеристика веб-сервісу моніторингу криптовалют

Веб-сервіс моніторингу криптовалют – це спеціалізована онлайн-система, яка в автоматичному режимі отримує, опрацьовує та відображає поточні ринкові показники цифрових активів із різних торгових майданчиків. До таких майданчиків належать, зокрема, Binance, Coinbase і Kraken – найбільші централізовані біржі за обсягами торгів.

Подібні системи надають трейдерам, інвесторам та аналітикам зручний доступ до актуальних котирувань, обсягів торгів і додаткової ринкової статистики через єдиний інтерфейс, що суттєво скорочує час на збирання інформації з розрізнених джерел.

Ключовими компонентами веб-сервісу моніторингу криптовалют є:

- інтерактивний скринер – таблиця з актуальними даними про торгові пари, що підтримує фільтрацію за ціною, обсягом або майданчиком, а також сортування за різними параметрами;
- модуль порівняння цін – засіб для одночасного зіставлення вартості однієї торгової пари на кількох біржах, що дозволяє виявляти арбітражні можливості;
- графічна візуалізація – відображення часових рядів цін у вигляді інтерактивних діаграм;
- конвертер валют – інструмент для перерахунку вартості активів між різними торговими парами;
- аутентифікація та особистий профіль – реєстрація користувача й керування індивідуальними налаштуваннями;
- персоналізовані списки спостереження – можливість зберігати обрані торгові пари для швидкого доступу.

Основні цілі веб-сервісу моніторингу криптовалют:

- забезпечення доступу до актуальних ринкових даних у режимі реального часу;
- спрощення аналізу поточних ринкових тенденцій;
- підтримка порівняння цін між різними майданчиками;
- персоналізація досвіду користувача через налаштовувані фільтри та списки спостереження;
- забезпечення адаптивності інтерфейсу для десктопних і мобільних пристроїв.

Перевагами веб-сервісу порівняно з іншими засобами отримання ринкової інформації є:

- доступність – можливість отримати необхідні дані з будь-якої точки світу за умови наявності підключення до мережі інтернет;
- різноманітність даних – зведені у єдиному інтерфейсі котирування, обсяги торгів та ринкова статистика з кількох бірж;
- гнучкість роботи – налаштовувані фільтри та сортування дозволяють аналізувати лише ті дані, що є релевантними для конкретного завдання;
- економія часу – автоматичне збирання та відображення даних усуває необхідність їх ручного пошуку.

Серед недоліків слід зазначити:

- залежність від стабільності інтернет-з'єднання та доступності API бірж;
- обмеження безкоштовних API-планів щодо кількості запитів на хвилину;
- потенційне перевантаження інтерфейсу великою кількістю даних, що вимагає ретельного продумування UX-дизайну.

1.3. Особливості роботи з ринковими даними

Збирання та опрацювання статистики криптовалютних ринків суттєво відрізняється від роботи з традиційними фінансовими даними. Криптовалютний ринок функціонує цілодобово без вихідних, а котирування торгових пар (наприклад, BTC/USDT або ETH/USDT) можуть змінюватися щосекунди, що висуває підвищені вимоги до частоти оновлення даних у системі. У розробленому веб-сервісі прийнятний інтервал оновлення визначено на рівні однієї хвилини, що є розумним компромісом між актуальністю даних і навантаженням на сервер.

Програмні інтерфейси бірж, таких як Binance, Coinbase і Kraken, надають доступ до поточних котирувань, обсягів торгів (за 1h, 24h, 7d, 30d) і додаткових ринкових показників (зміна ціни за 24h, мінімум і максимум за добу). Водночас кожна біржа застосовує власні формати відповідей і власні правила ліміту запитів. Наприклад, Binance дозволяє виконувати до 1200 запитів на хвилину, тоді як інші майданчики встановлюють значно менш ліберальні ліміти. Гетерогенність форматів даних ускладнює їх уніфікацію та вимагає застосування спеціалізованих бібліотек.

Обсяг даних, що підлягає зберіганню, є значним: лише на одній великій біржі налічуються сотні, а то й тисячі активних торгових пар. Для ефективного зберігання такого масиву інформації, що постійно оновлюється, доцільно використовувати документо-орієнтовані СУБД класу NoSQL, зокрема MongoDB, яка добре пристосована до роботи з даними змінної структури.

Технічні виклики при розробці подібних систем включають: необхідність паралельного збирання даних із кількох джерел без перевищення лімітів запитів; обробку мережевих збоїв і тимчасової недоступності API; забезпечення мінімальної затримки між отриманням нових даних і їх відображенням у клієнтському застосунку. Для вирішення цих завдань у роботі застосовано асинхронну модель виконання запитів на

базі Node.js, механізм повторних спроб при збоях та каскадне кешування на рівні клієнта.

1.4. Цільова аудиторія системи

Визначення цільової аудиторії є необхідною передумовою для формулювання вимог до функціоналу, інтерфейсу та продуктивності системи. Веб-сервіс моніторингу криптовалют орієнтований на чотири основні групи користувачів, які відрізняються за рівнем підготовки та характером потреб.

Перша група – початківці у сфері цифрових активів, переважно молоді люди від 18 до 30 років, які лише починають ознайомлення з криптовалютним ринком. Ця аудиторія потребує простого та інтуїтивно зрозумілого інтерфейсу, що не перевантажує зайвою інформацією, а також наявності підказок і засобів мультимовної підтримки. Для них критичними є адаптивний дизайн, зручна базова таблиця скринера та можливість швидко зрозуміти ключові показники.

Друга група – активні трейдери у віці від 20 до 35 років, які здійснюють регулярні торгові операції на криптовалютних біржах. Вони потребують мінімальних затримок у відображенні даних, гнучких інструментів фільтрації й сортування, а також функції порівняння цін між майданчиками для пошуку арбітражних можливостей. Для них пріоритетними є швидкість і точність даних.

Третя група – довгострокові інвестори у віці від 25 до 50 років, зосереджені на стратегічному управлінні портфелем. Ця аудиторія цінує можливість відстеження цінових трендів за тривалі проміжки часу, доступ до розширеної аналітики та зручні засоби персоналізації — зокрема збереження списків відстежуваних активів.

Четверта група – аналітики криптовалютного ринку, фахівці фінансових установ і незалежні дослідники, яким необхідний доступ до

структурованих ринкових даних через API для подальшої їх обробки у власних аналітичних системах і побудови звітів.

Врахування потреб усіх чотирьох груп знайшло відображення в архітектурних рішеннях і функціональному наповненні системи: від базового скринера та порівняльної таблиці до відкритого REST API та механізму аутентифікації з персоналізованими списками спостереження.

1.5. Аналіз процесу моніторингу криптовалют

Процес моніторингу криптовалют включає:

- обробку інформації;
- збір ринкових даних;
- збереження результатів;
- візуалізацію котирувань;
- аналітичну обробку;
- формування звітів.

Етапи моніторингу ринку від моменту, коли сервіс збирає «сирі» дані з криптобірж, через етапи внутрішньої обробки та збереження, і до безпосереднього виведення інформації кінцевому користувачу в інтерфейсі веб-застосунку представлено на рис. 1.3.

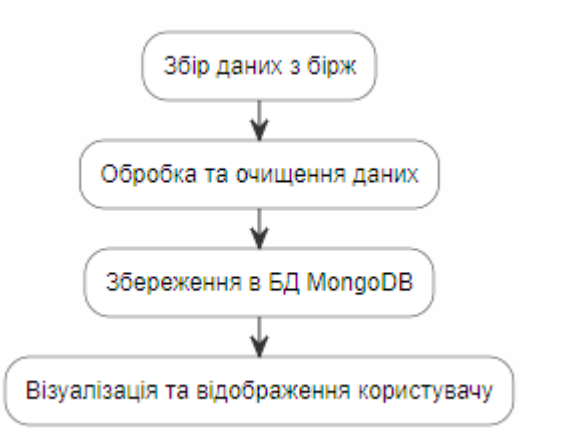

Рисунок 1.3. Етапи моніторингу криптовалютного ринку

Кожен із цих 4 етапів деталізується вашими попередніми схемами:

1. *Збір даних з бірж*: реалізується фоновим таском на Node.js, який через клієнт CCXT робить REST-запити до Binance, Bybit чи Kraken.

2. *Обробка та очищення даних*: включає етап отримання торгових пар та нормалізації JSON-відповідей від бірж до єдиного формату системи.

3. *Збереження в БД MongoDB*: Запис підготовлених документів у колекцію Markets (з полями symbol, price, volume, timestamp).

4. *Візуалізація та відображення користувачу*: Запит від клієнта (GET /markets), проходження через Express Router, рендеринг у компонентах MarketTable та побудова інтерактивних графіків за допомогою Chart.js / TradingView.

Основними параметрами моніторингу є:

- ціна криптовалюти;
- обсяг торгів;
- ринкова капіталізація;
- зміна ціни за період;
- ліквідність активу;
- біржова активність.

Моніторинг криптовалютного ринку може здійснюватися:

- через REST API;
- через WebSocket-з’єднання;
- за допомогою спеціалізованих бібліотек;
- через сторонні агрегатори даних.

1.6. Огляд існуючих веб-сервісів моніторингу криптовалют

На сьогодні існує велика кількість платформ для моніторингу криптовалютного ринку. Найбільш популярними серед них є:

- CoinMarketCap;
- CoinGecko;

- TradingView;
- CryptoCompare;
- Binance Market Dashboard.

1.6.1. CoinMarketCap

CoinMarketCap є однією з найвідоміших платформ для агрегування інформації про ринок криптовалют. Вона акумулює котирування, показники ринкової капіталізації та обсяги торгів для тисяч цифрових активів із сотень бірж, надаючи зведений огляд глобального стану ринку. Платформа доступна через веб-інтерфейс, мобільні застосунки і REST API (рис. 1.4).

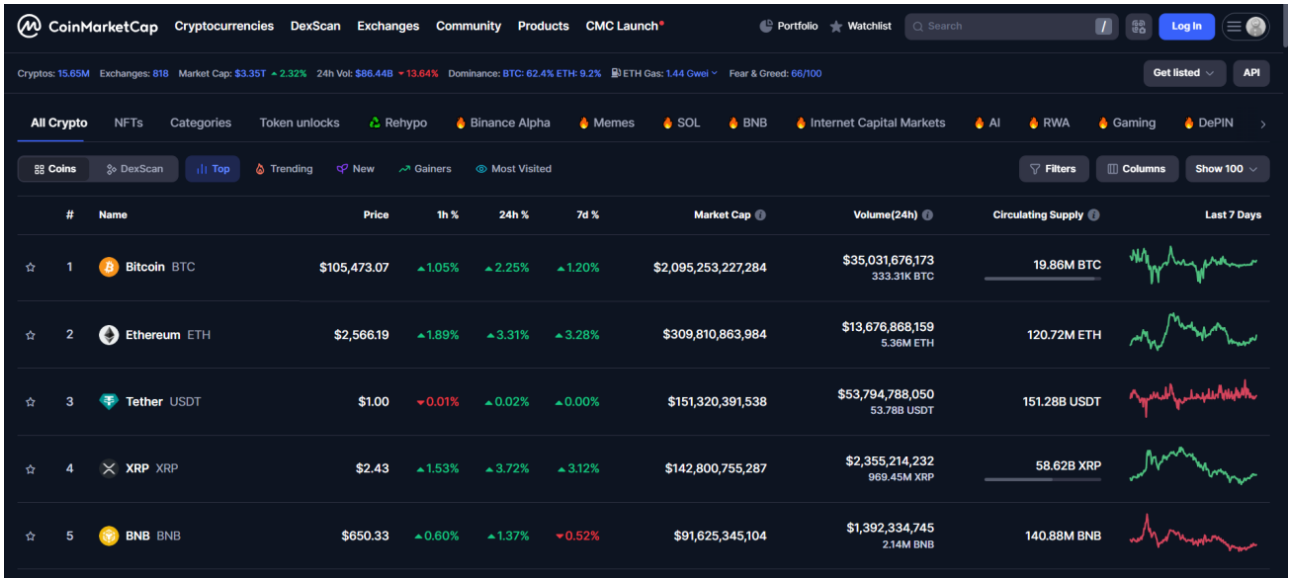

Рисунок 1.4. Інтерфейс користувача CoinMarketCap

Ключові можливості CoinMarketCap:

- агрегований перегляд цін і ринкової капіталізації активів;
- побудова цінових графіків за різні проміжки часу – від доби до кількох років;
- рейтинги криптовалют і бірж за показниками ринкової активності;

- підтримка REST API для програмного доступу до ринкових даних.

Переваги платформи: широке охоплення активів і бірж, наявність API, підтримка декількох мов інтерфейсу, регулярне поповнення переліку підтримуваних активів.

Обмеження, що ускладнюють її використання в якості основного інструменту для активного трейдингу: оновлення даних відбувається із запізненням до кількох хвилин; безкоштовний API-план передбачає не більше 10 000 запитів на місяць; відсутня функція одночасного порівняння ціни однієї торгової пари на різних майданчиках; інтерфейс перевантажений інформацією і потребує певного рівня підготовки для ефективного використання.

Головними позитивними факторами є:

- відображення котирувань;
- ринкова капіталізація;
- рейтинги криптовалют;
- історичні графіки;
- аналітика ринку.

1.6.2. CoinGecko

CoinGecko – ще один популярний агрегатор ринкової інформації, який охоплює широкий спектр криптовалютних активів і торгових майданчиків. Платформа вирізняється розширеними аналітичними метриками: відомостями про ліквідність ринку, дані про команду проєктів і технологічну базу відповідних блокчейн-протоколів (рис. 1.5). Безкоштовний API-план є більш ліберальним порівняно з конкурентами.

Серед функцій платформи: агрегований перегляд цін і капіталізації; доступ до розширеної аналітики по кожному активу; надання ринкових

даних через JSON API; інформація про проєкти, що стоять за криптовалютами.

До переваг відносяться: широке охоплення активів, розширена аналітика для інвесторів, безкоштовний API.

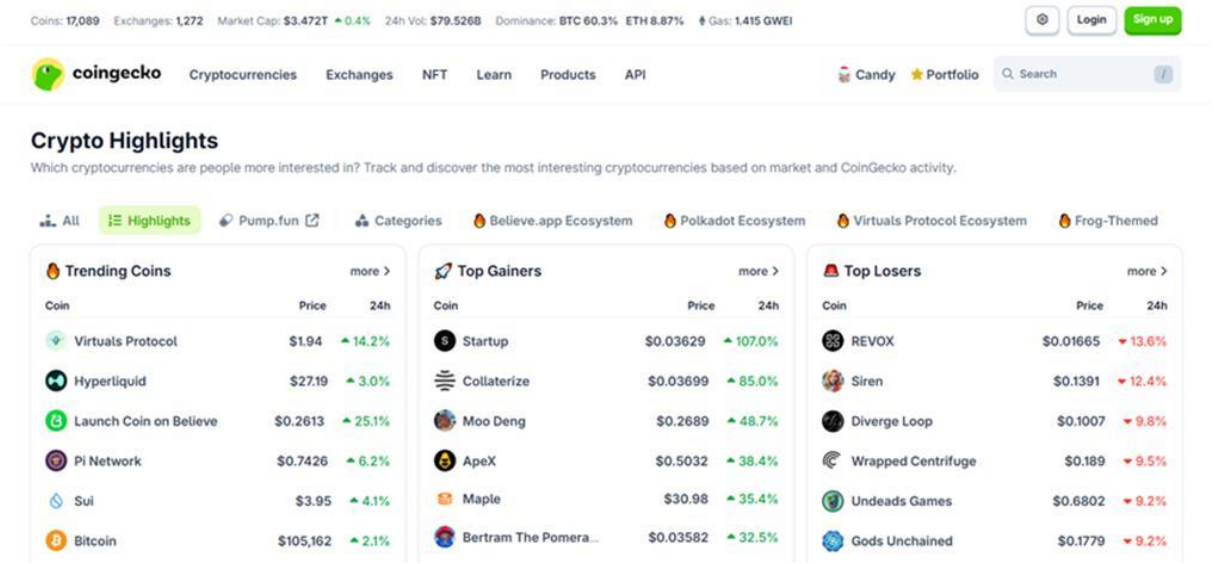

Рисунок 1.5. Інтерфейс користувача CoinGecko

Серед обмежень – затримки в оновленні даних (кілька хвилин), відсутність гнучких параметрів фільтрації, жодної вбудованої функції порівняння цін між різними майданчиками, обмежений функціонал персоналізації.

CoinGecko забезпечує:

- моніторинг цін;
- відображення обсягів торгів;
- аналіз ліквідності;
- підтримку великої кількості бірж.

1.6.3. TradingView

TradingView є платформою технічного аналізу фінансових ринків, у тому числі криптовалютних (рис. 1.6). Її ключова перевага – потужний

інструментарій для побудови інтерактивних графіків із підтримкою широкого набору технічних індикаторів (RSI, MACD, Bollinger Bands тощо). Платформа має вбудовану соціальну складову: користувачі можуть публікувати свої прогнози та аналітичні огляди.

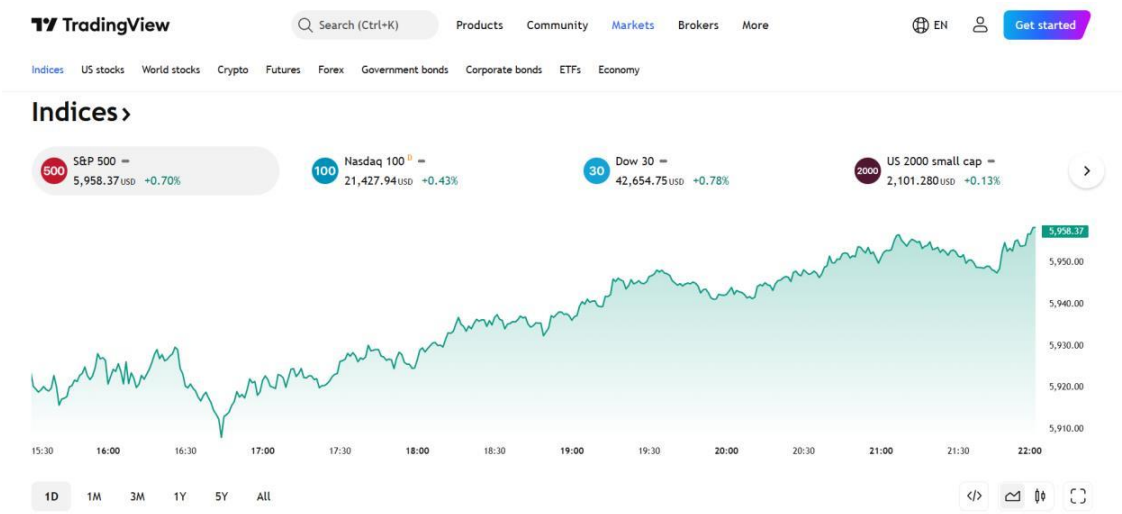

Рисунок 1.6. Інтерфейс користувача TradingView

Основні можливості: налаштовуванні інтерактивні графіки з технічними індикаторами; інструменти для побудови торгових стратегій; соціальна мережа трейдерів; API для вбудовування графіків у зовнішні сервіси.

Переваги: потужні засоби технічного аналізу, висока якість візуалізації, активна спільнота.

Обмеження: платформа не є повноцінним агрегатором ринкових даних – вона відображає інформацію переважно з окремих підключених бірж без можливості їх одночасного порівняння; для безкоштовних користувачів дані оновлюються із затримкою; інтерфейс із великим набором аналітичних інструментів може виявитися складним для новачків.

TradingView орієнтований на технічний аналіз ринку та підтримує:

- інтерактивні графіки;

- індикатори технічного аналізу;
- порівняння активів;
- біржову аналітику.

1.7. Порівняльний аналіз існуючих рішень

Проведений огляд наявних платформ засвідчив, що жодна з них не поєднує у собі всіх необхідних можливостей: актуалізацію даних у реальному часі з частотою оновлення раз на хвилину, функцію одночасного порівняння цін однієї торгової пари на кількох майданчиках, гнучку систему фільтрації та сортування, персоналізовані списки спостереження й адаптивний мультимовний інтерфейс.

Порівняльні характеристики існуючих систем представлені в табл. 1.1, табл. 1.2.

Таблиця 1.1. Порівняльна характеристика платформ моніторингу криптовалют

Показник	**CoinMarketCap**	**CoinGecko**	**TradingView**
Тип	Інформаційно-аналітична платформа	Інформаційно-аналітична платформа	Платформа технічного аналізу
Цільова аудиторія	Трейдери, інвестори, аналітики	Трейдери, інвестори, новачки	Досвідчені трейдери, аналітики
Оновлення в реальному часі	Затримка до кількох хвилин	Затримка до кількох хвилин	Затримка для безкоштовних планів
Порівняння цін між біржами	Відсутнє	Відсутнє	Відсутнє
Гнучка фільтрація	Базова	Обмежена	Відсутня
Персоналізація	Обмежена	Обмежена	Часткова

Таблиця 1.2. Порівняльна характеристика існуючих систем моніторингу

Сервіс	**Моніторинг цін**	**Графіки**	**API**	**Порівняння бірж**
CoinMarketCap	Так	Так	Так	Частково
CoinGecko	Так	Так	Так	Так
TradingView	Так	Розширені	Так	Так

Отже, попри широкий функціонал сучасних платформ, існуючі системи мають низку недоліків:

- складність інтерфейсів;
- надлишкова кількість інформації;
- обмеження безкоштовного API;
- залежність від зовнішніх сервісів;
- недостатня гнучкість персоналізації;
- обмежені можливості інтеграції.

Крім того, багато платформ:

- орієнтовані на комерційне використання;
- мають закриті API;
- потребують платної підписки;
- не дозволяють створювати власні механізми аналітики.

Узагальнена схема недоліків існуючих платформ представлена на рис. 1.7.

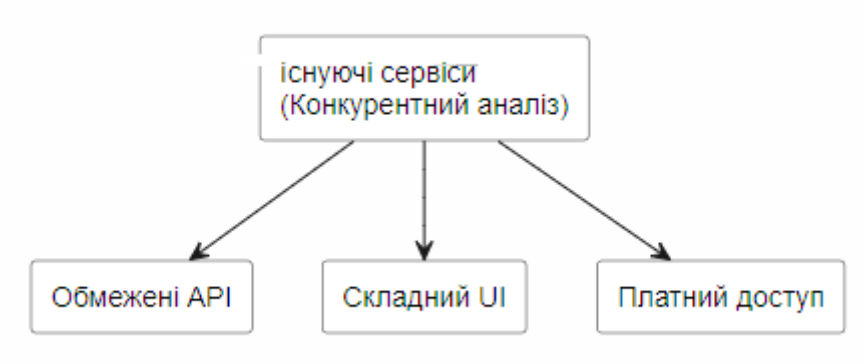

Рисунок 1.7. Узагальнена схема недоліків існуючих платформ

Ця схема аналізує недоліки конкурентів та поточний стан на ринку («Існуючі сервіси»), виділяючи три ключові проблеми, які вирішує проєкт: обмежені можливості безкоштовних API, складні для сприйняття інтерфейси (UI) та високу вартість підписки (платний доступ).

У зв’язку з цим виникає необхідність створення власного веб-сервісу моніторингу криптовалют із відкритою архітектурою та підтримкою адаптивного SPA-інтерфейсу.

1.8. Формування вимог до веб-сервісу

На основі проведеного порівняльного аналізу конкурентних рішень, особливостей предметної області та потреб цільової аудиторії було сформовано основні функціональні вимоги до системи.

Веб-сервіс повинен забезпечувати:

- автоматичний збір котирувань – отримання актуальних котирувань і обсягів торгів із декількох криптовалютних бірж через уніфікований API-клієнт;
- відображення ринкових даних у реальному часі з оновленням не рідше ніж раз на хвилину
- отримання обсягів торгів;
- підтримку декількох бірж;
- централізоване зберігання даних;
- RESTful API;
- фільтрацію за назвою, майданчиком і ціновим діапазоном;
- сортування за ціною, обсягом торгів і зміною ціни за 24 години;
- графічне відображення ринкових даних, в т.ч. динаміки ціни за останні 24 години;
- порівняння цін однієї торгової пари між біржами;
- адаптивний SPA-інтерфейс;

- підтримку реєстрації, авторизації та ведення персоналізованих списків спостереження користувачів.

До нефункціональних вимог належать:

- масштабованість – модульна архітектура, що забезпечує можливість підключення нових джерел даних без суттєвої переробки;
- продуктивність – оновлення даних щонайменше раз на хвилину;
- безпечність – розмежування доступу через JWT-токени, налаштування політики CORS;
- надійність і відмовостійкість – стійкість до збоїв окремих API без повної зупинки сервісу;
- підтримка асинхронної обробки;
- кросплатформеність – коректне відображення на мобільних і десктопних пристроях.

1.9. Обґрунтування вибору технологічного стеку

Для реалізації веб-сервісу було обрано сучасний стек технологій.

1. Node.js

Node.js використовується для:

- реалізації серверної частини;
- підтримки асинхронної обробки;
- роботи з REST API;
- обробки великої кількості HTTP-запитів.

2. Express.js

Express.js забезпечує:

- маршрутизацію;
- middleware;
- підтримку RESTful API;
- модульну архітектуру.

3. MongoDB

MongoDB використовується як документоорієнтована база даних.

Переваги:

- висока швидкість запису;
- масштабованість;
- підтримка JSON-документів;
- гнучка структура даних.

4. CCXT

Бібліотека CCXT забезпечує:

- уніфікований доступ до API бірж;
- підтримку великої кількості торгових платформ;
- стандартизовану структуру ринкових даних.

5. React / Vue.js

Для створення SPA-застосунку можуть використовуватися:

- React;
- Vue.js.

Вони забезпечують:

- компонентний підхід;
- швидке оновлення інтерфейсу;
- динамічну взаємодію з API.

Схема технологічного стеку (рис. 1.8) представляє трирівневу структуру додатку (3-Tier Architecture), що охоплює клієнтську частину (Frontend SPA), серверні ендпоінти (REST API), базу даних (MongoDB) та ізольований фоновий сервіс збору даних із зовнішніх криптобірж (CCXT Collector).

Ці рівні можуть бути визначені по іншому.

1. *Frontend-рівень*. Базується на компонентних фреймворках (React / Vue.js), які забезпечують роботу Single Page Application (SPA). Для інтерактивного виведення аналітики та часових рядів використовуються

спеціалізовані бібліотеки візуалізації – Chart.js або вбудовані інтеграційні віджети TradingView Widget.

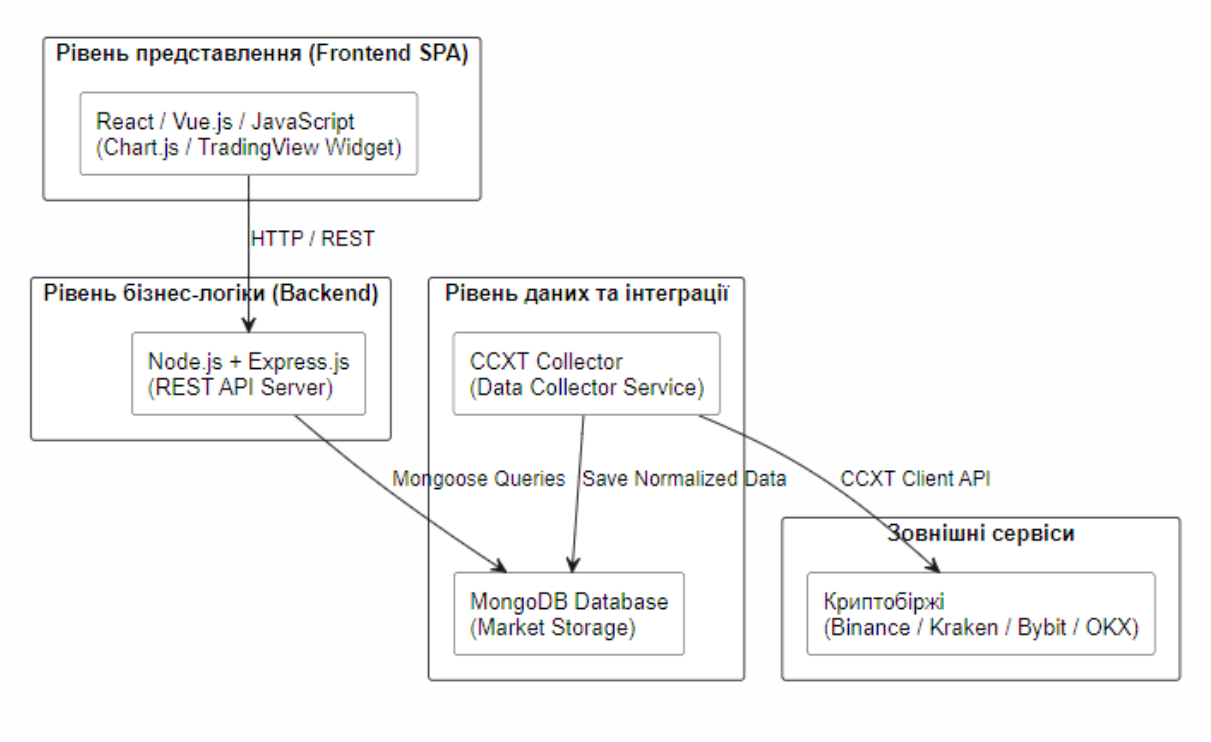

Рисунок 1.8. Схема технологічного стеку

2. *Backend-рівень*. Реалізований на платформі Node.js з використанням мінімалістичного фреймворку Express.js. Цей шар відповідає за маршрутизацію HTTP-запитів, авторизацію (JWT Middleware), обробку бізнес-логіки та надання даних через уніфікований REST API інтерфейс.

3. *Рівень збереження даних.* Використовує документоорієнтовану NoSQL СКБД MongoDB. Вона є оптимальним вибором для збереження гнучких JSON-структур ринкових котирувань (Markets), даних користувачів (Users) та їхніх персональних списків спостереження (Watchlists).

4. Модуль агрегації даних (Data Collector). Винесено в окремий фоновий підсервіс (демон/воркер) на базі бібліотеки CCXT. Вона стандартизує запити до API різних криптомайданчиків (Binance, Bybit,

Kraken, OKX), дозволяючи в один циклічний такт збирати, очищувати та нормалізувати потоки даних перед їх відправкою до MongoDB.

1.10. Постановка задачі

Метою кваліфікаційної роботи є розроблення веб-сервісу моніторингу криптовалют, який забезпечує автоматичний збір, зберігання, обробку та візуалізацію ринкових даних із підтримкою REST API та SPA-інтерфейсу.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- провести аналіз предметної області криптовалютного моніторингу;
- дослідити існуючі програмні рішення;
- визначити функціональні та нефункціональні вимоги;
- обґрунтувати вибір технологічного стеку;
- спроектувати архітектуру веб-сервісу;
- реалізувати модуль збору даних через CCXT;
- створити RESTful API;
- реалізувати SPA-клієнт;
- реалізувати механізми фільтрації та аналітики;
- провести тестування програмного забезпечення.

Схеми рис. 1.7 та рис. 1.8 слугують містком до конкретних архітектурних переваг розроблюваного веб-сервіса.

Рішенням проблеми «Обмежені API» є використання власного Node.js Collector на базі модуля CCXT, що дозволяє збирати, нормалізувати та безкоштовно накопичувати в MongoDB глибоку історію ринкових даних з багатьох бірж одночасно (Binance, Bybit, Kraken, OKX).

Рішенням проблеми «Складний UI» є продумана, чиста ієрархія фронтенду (App Component $\rightarrow$ Header, MarketTable, Charts) забезпечує інтуїтивне та швидке сприйняття інформації.

Інтеграція алгоритму миттєвої фільтрації та сортування дозволяє налаштувати робочий простір під себе без перевантаження екрана зайвими метриками.

Рішенням проблеми «Платний доступ», оскільки користувач отримує інструмент для локального або власного розгортання веб-аналітики з відкритим кодом (з авторизацією через JWT), є його позбавлення від потреби сплачувати дорогі щомісячні тарифи стороннім платформам.

Висновки до розділу

У результаті аналізу предметної області було досліджено особливості функціонування криптовалютного ринку, принципи моніторингу ринкових даних та сучасні веб-сервіси аналітики цифрових активів.

Проведений аналіз існуючих платформ дозволив визначити їх основні переваги та недоліки, а також сформувати вимоги до власного веб-сервісу моніторингу криптовалют.

Було обґрунтовано вибір технологічного стеку, до складу якого входять:

- Node.js;
- Express.js;
- MongoDB;
- CCXT;
- React або Vue.js.

Запропонований підхід дозволяє реалізувати масштабований веб-сервіс із підтримкою автоматичного збору ринкових даних, RESTful API та сучасного SPA-інтерфейсу для моніторингу криптовалютного ринку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ МОНІТОРИНГУ КРИПТОВАЛЮТ

2.1. Аналіз вимог до веб-сервісу

Веб-сервіс моніторингу криптовалют призначений для автоматизованого збору, обробки, зберігання та візуалізації даних криптовалютного ринку. Основною метою системи є надання користувачам актуальної інформації щодо котирувань цифрових активів, обсягів торгів, змін цін та порівняння показників між різними біржовими майданчиками.

У процесі аналізу предметної області було досліджено функціональні можливості сучасних платформ моніторингу криптовалют, серед яких:

- CoinMarketCap;
- CoinGecko;
- TradingView.

Аналіз існуючих рішень дозволив визначити основні функціональні вимоги до розроблюваного веб-сервісу:

- автоматичний збір котирувань криптовалют;
- отримання обсягів торгів із криптовалютних бірж;
- централізоване зберігання даних;
- підтримка RESTful API;
- фільтрація та сортування активів;
- графічне відображення ринкових даних;
- порівняння вартості активів між біржами;
- адаптивний користувацький інтерфейс;
- підтримка Single Page Application.

До нефункціональних вимог системи належать:

- масштабованість;
- висока продуктивність;
- надійність;

- безпечність;
- підтримка асинхронної обробки даних;
- кросплатформеність.

2.2. Проєктування архітектури веб-сервісу

Для реалізації системи було обрано триланкову архітектуру, яка складається з: 1. рівня збирання даних; 2. серверного рівня REST API; 3. клієнтського SPA-застосунку.

Така архітектура забезпечує:

- модульність системи;
- незалежність компонентів;
- спрощення масштабування;
- централізацію бізнес-логіки;
- ефективну взаємодію між компонентами.

Загальна архітектурна схема системи зображена на рис. 2.1.

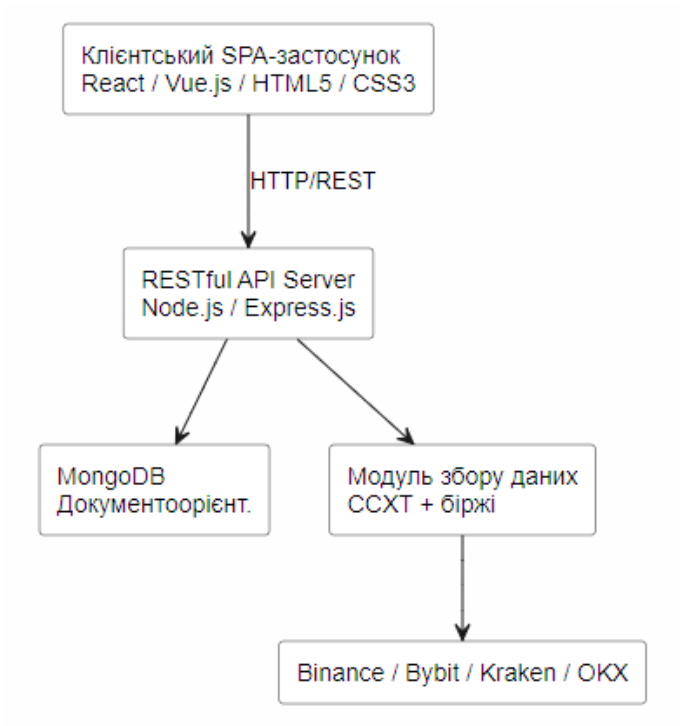

Рисунок 2.1. Схема архітектури веб-сервісу

2.3. Проєктування модуля збору даних

Модуль збору даних відповідає за автоматичне отримання ринкової інформації з криптовалютних бірж.

Для уніфікованої взаємодії з API бірж використовується бібліотека CCXT, яка забезпечує:

- стандартизований доступ до бірж;
- підтримку великої кількості торгових платформ;
- отримання ринкових котирувань;
- роботу з REST та WebSocket API.

Схема роботи модуля збору даних представлена на рис. 2.2.

Рисунок 2.2. Схема роботи модуля збору даних

Основні функції модуля збору даних є наступні:

- періодичне опитування бірж;
- обробку JSON-відповідей;
- нормалізацію даних;
- запис результатів до MongoDB;
- виявлення помилок API.

Алгоритм збору котирувань представлено на рис. 2.3.

Рисунок 2.3. Схема алгоритму збору котирувань

2.4. Проєктування серверної частини REST API

Серверна частина системи реалізується із використанням Node.js та Express.js.

Основними задачами REST API є:

- надання доступу до ринкових даних;
- обробка HTTP-запитів;
- взаємодія з MongoDB;
- фільтрація та сортування даних;
- передача даних клієнтському застосунку.

Архітектура REST API зображена схематично на рис. 2.4.

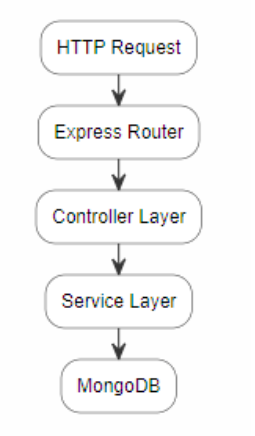

Рисунок 2.4. Схема архітектури REST API

Основні REST-ендпоінти представлено у табл. 2.1.

Таблиця 2.1. REST-ендпоінти

Метод	Endpoint	Призначення
GET	/api/markets	Отримання списку активів
GET	/api/ticker/:symbol	Дані конкретної криптовалюти
GET	/api/exchanges	Список бірж
GET	/api/history	Історичні дані
GET	/api/compare	Порівняння цін

Формат JSON-відповіді приведено в Лістингу 2.1.

Лістинг 2.1. Формат JSON-відповіді

```
json id="j4n7yf"
{
   "symbol": "BTC/USDT",
   "exchange": "Binance",
   "price": 68250.25,
   "volume": 123456789
}
```

2.5. Проєктування структури бази даних

Для зберігання ринкових даних використовується MongoDB.

Вибір документоорієнтованої бази даних обумовлений:

- гнучкою структурою документів;
- високою швидкістю запису;
- підтримкою JSON-подібних структур;
- зручністю інтеграції з Node.js.

Основні колекції системи представлені на рис. 2.5.

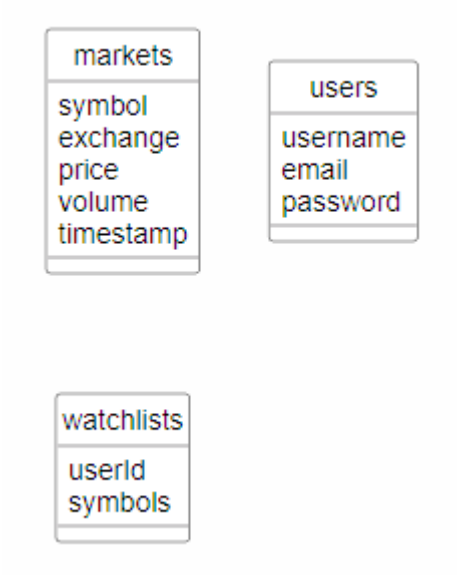

Рисунок 2.5. Схема колекції системи

ER-діаграму системи представлено на рис. 2.6.

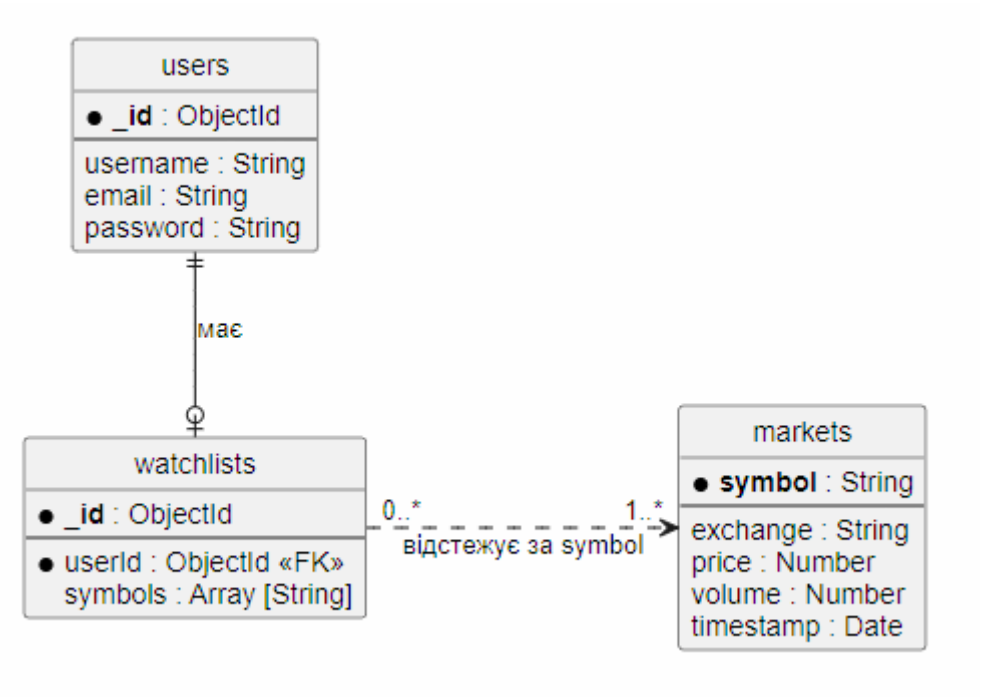

Рисунок 2.6. ER-діаграма

2.6. Проєктування клієнтського SPA-застосунку

Клієнтська частина реалізується у вигляді Single Page Application. SPA-застосунок забезпечує:

- швидке оновлення даних;
- динамічний інтерфейс;
- асинхронну взаємодію з API;
- адаптивність для мобільних пристроїв.

Основні компоненти інтерфейсу

- панель моніторингу;
- таблиця криптовалют;
- графіки цін;
- система фільтрації;
- модуль порівняння бірж;
- особистий список спостереження.

Структура SPA-застосунку представлена на рис. 2.7.

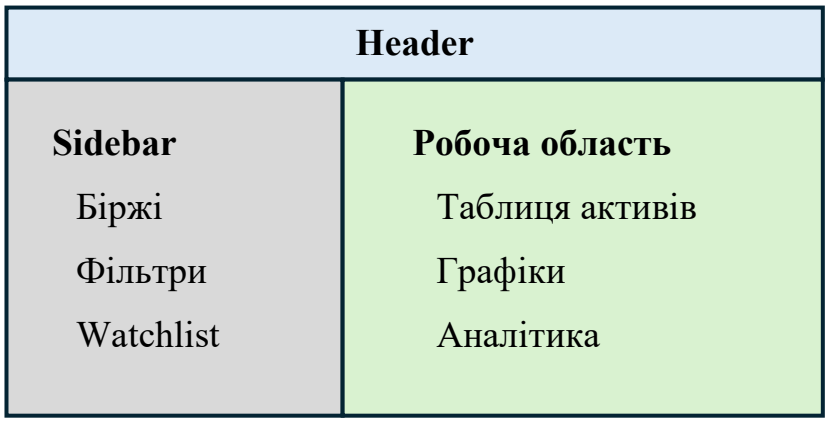

Рисунок 2.7. Структура SPA-застосунку

2.7. Проєктування механізмів візуалізації даних

Однією з ключових функцій системи є графічне відображення ринкової інформації.

Для візуалізації використовуються:

- лінійні графіки;
- гістограми;

- свічкові графіки;
- таблиці динаміки цін.

Схема побудови графіків (рис. 2.8) ілюструє процес виведення даних на фронтенд для візуалізації від MongoDB через API до графічних віджетів.

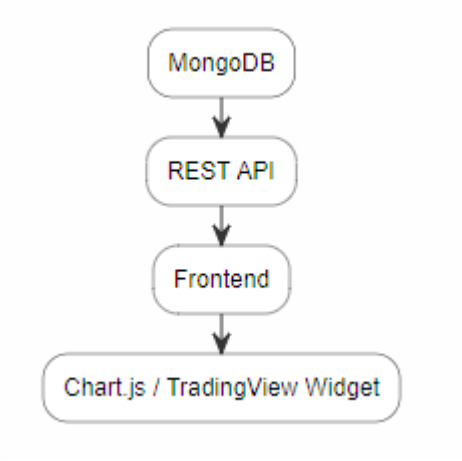

Рисунок 2.8. Схема побудови графіків

Візуалізація дозволяє:

- аналізувати зміни цін;
- відстежувати тренди;
- порівнювати біржі;
- оцінювати ринкову активність.

2.8. Проєктування системи безпеки

Для забезпечення безпеки веб-сервісу передбачено:

- JWT-авторизацію;
- захист REST API;
- валідацію вхідних даних;
- хешування паролів;
- обмеження частоти запитів.

Схема авторизації (рис. 2.9) представляє лінійний потік (процес), що показує шлях авторизації користувача через JWT-перевірку до захищених ресурсів API.

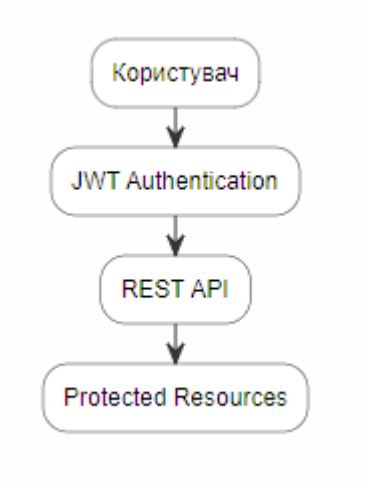

Рисунок 2.9. Схема авторизації

2.9. Детальна архітектура системи

Діаграма прецедентів (Use Case Diagram) описує основні можливості та дії користувача у системі (рис. 2.10).

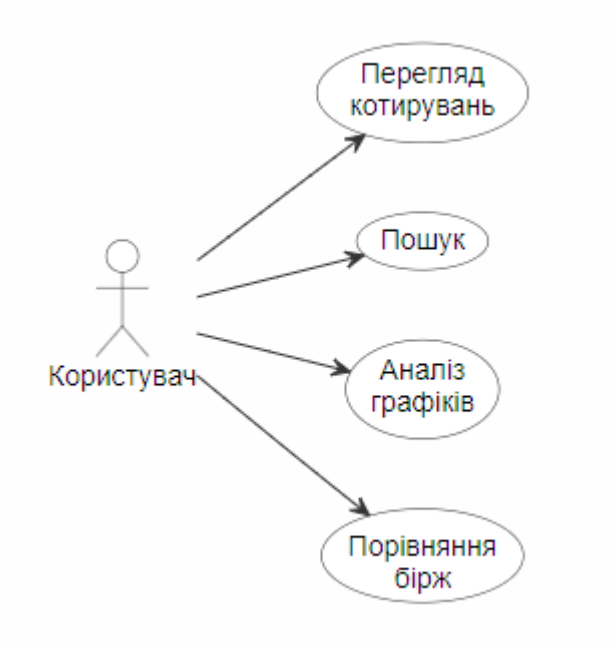

Рисунок 2.10. Діаграма прецедентів

Діаграма послідовності (рис. 2.11) показує життєвий цикл запиту котирувань від дій користувача до отримання даних із бази та виведення графіків.

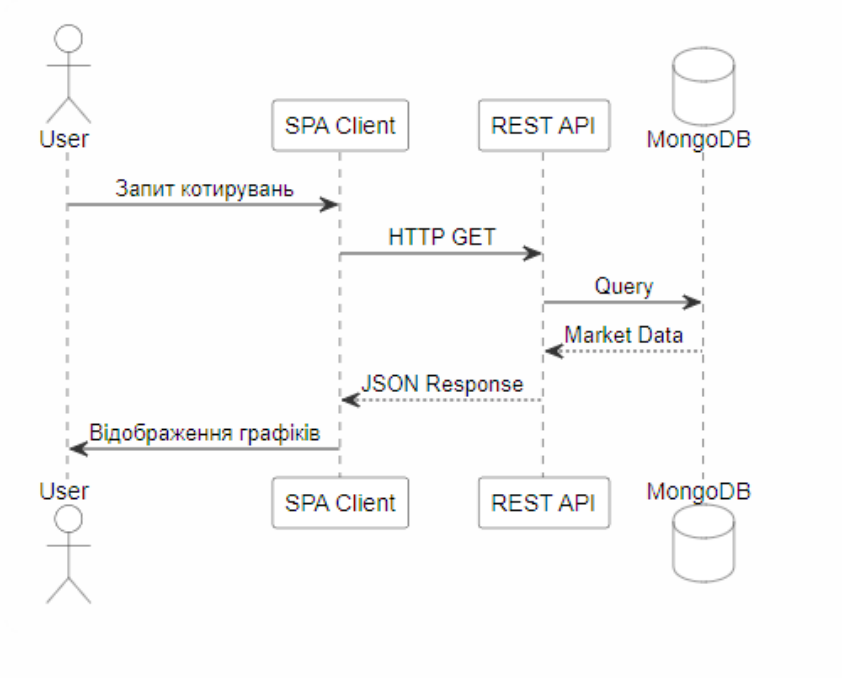

Рисунок 2.11. Діаграма послідовності

Діаграма компонентів (рис. 2.12) відображає взаємодію між клієнтом, бекендом, базою даних та модулем збору даних із криптобірж.

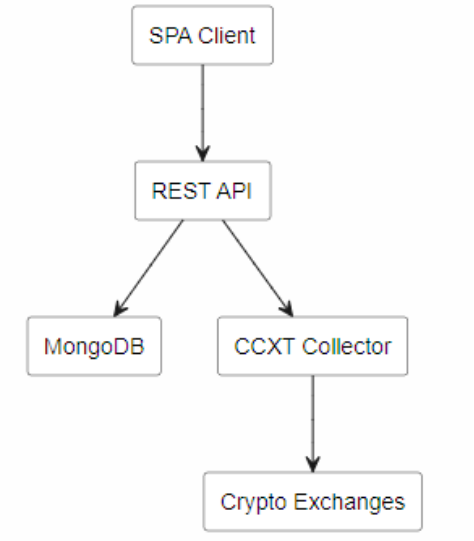

Рисунок 2.12. Діаграма компонентів

Нарешті, діаграма рис. 2.13 зображує детальну архітектуру розроблюваної системи.

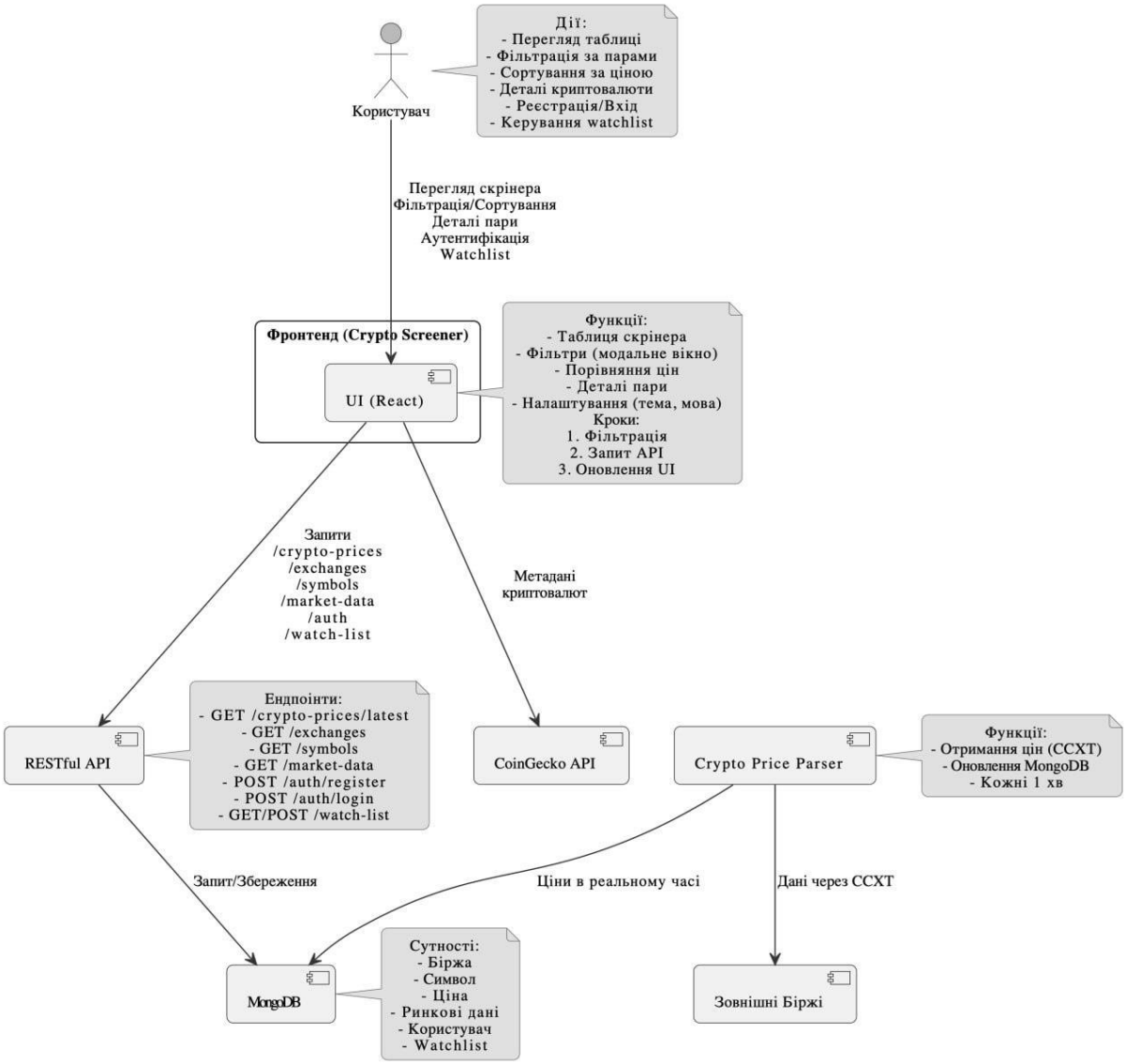

Рисунок 2.13. Детальна архітектура системи

Висновки до розділу

У процесі проектування веб-сервісу моніторингу криптовалют було визначено архітектуру системи, структуру бази даних, модель взаємодії між компонентами та механізми інтеграції з криптовалютними біржами.

Було обґрунтовано використання триланкової архітектури із виділенням:

- модуля збору даних;
- RESTful API;
- клієнтського SPA-застосунку.

Запропонована архітектура забезпечує:

- масштабованість;
- модульність;
- ефективну обробку ринкових даних;
- підтримку асинхронної взаємодії;
- зручність подальшого розвитку системи.

Використання бібліотеки CCXT, MongoDB та REST API дозволяє реалізувати сучасний веб-сервіс моніторингу криптовалют із підтримкою аналізу ринкових даних у реальному часі.

РОЗДІЛ 3. РОЗРОБЛЕННЯ ВЕБ-СЕРВІСУ МОНІТОРИНГУ КРИПТОВАЛЮТ

3.1. Загальна структура програмної реалізації веб-сервісу

Розроблений веб-сервіс моніторингу криптовалют реалізовано відповідно до триланкової архітектури, що включає:

- модуль збору ринкових даних;
- серверну частину REST API;
- клієнтський SPA-застосунок.

Основною метою програмної реалізації є забезпечення:

- автоматизованого отримання котирувань;
- централізованого зберігання ринкових даних;
- надання REST-інтерфейсу;
- візуалізації змін цін у режимі реального часу;
- підтримки аналітики криптовалютного ринку.

Загальна схема веб-сервісу, що ілюструє як глобальну архітектуру, так і деталізацію внутрішньої логіки роботи самого модуля збору даних (ETL-процесу), представлена на рис. 3.1.

3.2. Реалізація модуля збору ринкових даних

Модуль збору даних реалізовано із використанням бібліотеки CCXT, яка забезпечує уніфікований доступ до API криптовалютних бірж.

Основними функціями модуля є:

- підключення до бірж;
- отримання котирувань;
- збір обсягів торгів;
- обробка JSON-відповідей;
- запис даних у MongoDB.

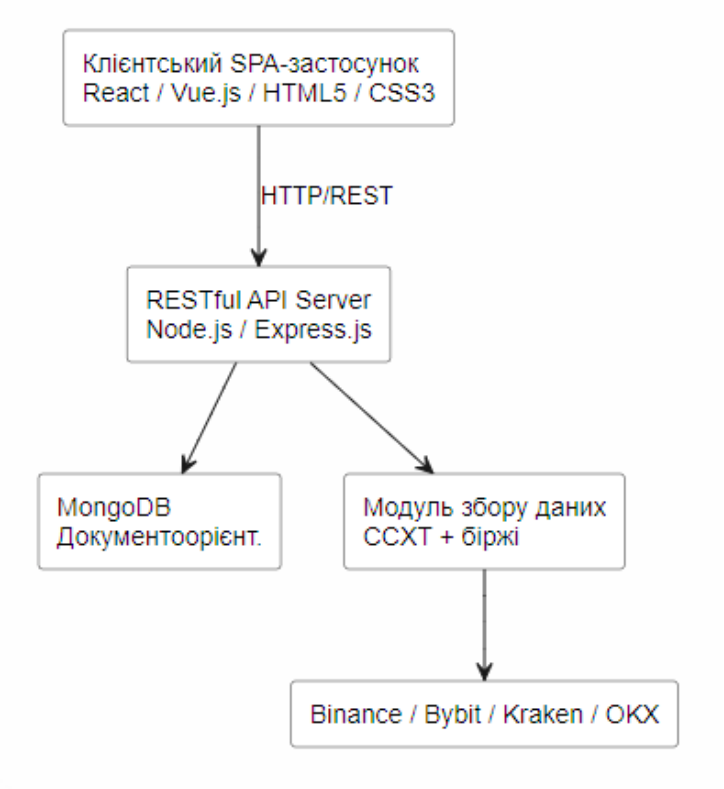

Рисунок 3.1. Загальна схема веб-сервісу

Алгоритм роботи модуля збору даних представлено на схемі рис. 3.2, яка детально описує циклічну природу роботи сервісу із початковим запуском та фінальним очікуванням наступного інтервалу.

Реалізацію підключення до біржі мовою javascript представлено в лістингу 3.1.

Лістинг 3.1. Підключення до біржі

```
const ccxt = require('ccxt');
const exchange = new ccxt.binance();
async function loadTicker() {
        const ticker = await exchange.fetchTicker('BTC/USDT');
console.log(ticker);
}
```

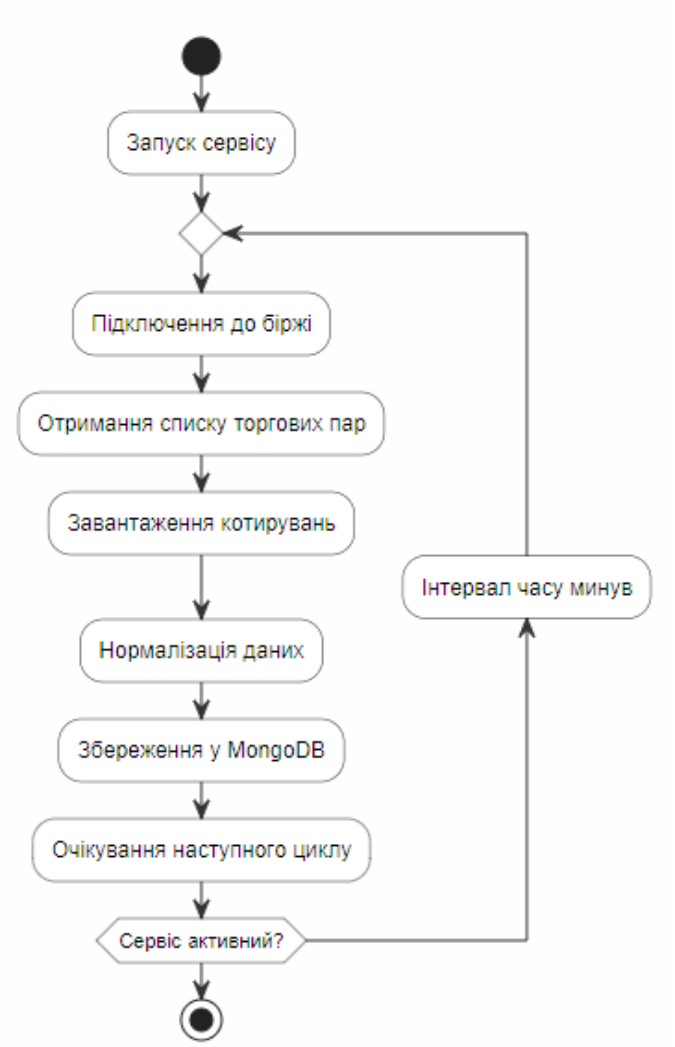

Рисунок 3.2. Схема алгоритму роботи модуля збору даних

Схема взаємодії CCXT з біржею, яка описує процес отримання даних клієнтом CCXT через REST API біржі, повернення JSON-відповіді та її обробку колектором на Node.js, представлена на рис. 3.3.

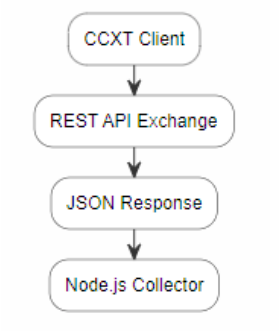

Рисунок 3.3. Схема взаємодії з біржею

Для забезпечення актуальності даних використовується циклічне опитування бірж із заданим часовим інтервалом.

3.3. Реалізація серверної частини REST API

Серверна частина веб-сервісу реалізована на платформі Node.js із використанням Express.js.

Основними задачами серверної частини є:

- обробка HTTP-запитів;
- взаємодія з MongoDB;
- передача даних клієнтському застосунку;
- підтримка REST API;
- фільтрація та сортування інформації.

Структура серверного застосунку представлена на рис. 3.4.

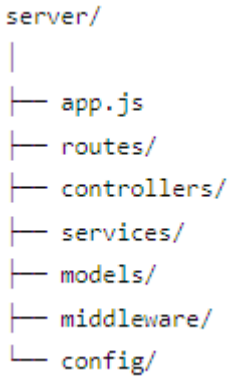

Рисунок 3.4. Структура серверного застосунку

Ініціалізація Express.js представлена в лістингу 3.2.

Лістинг 3.2. Ініціалізація Express.js

```
const express = require('express');
const app = express();

app.use(express.json());
      app.listen(3000, () => {

console.log('API Server started');
```

```
});
```

Схема маршрутизації REST API (рис. 3.5) описує єдину логіку обробки HTTP-запиту в Express.js. Коли запит іде до захищених ресурсів (Protected Resources), він проходить крізь роутер, потім обов'язково через шар перевірки токена (Middleware), і лише тоді потрапляє в контролер та сервіс.

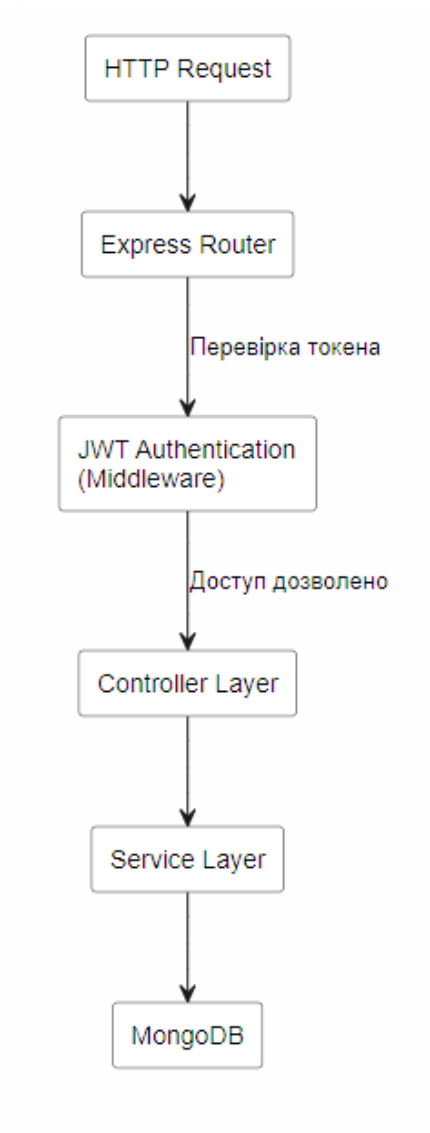

Рисунок 3.5. Схема маршрутизації REST API

Реалізація REST-ендпоінта на javascript представлена в лістингу 3.3.

Лістинг 3.3. Реалізація REST-ендпоінта

```
router.get('/markets', async (req, res) => {
   const data = await Market.find();
   res.json(data);
});
```

Алгоритм реалізації розробленого сервісу збору даних з криптовалютних бірж представлена на рис. 3.6.

3.4. Реалізація протоколів взаємодії

Для взаємодії між компонентами системи використовуються:

- HTTP/HTTPS;
- REST API;
- JSON-формат передачі даних.

Система підтримує такі HTTP-методи:

- GET;
- POST;
- PUT;
- DELETE.

Схема мережевої взаємодії (рис. 3.7) деталізує шлях проходження клієнтського запиту від SPA через REST API сервер (у вигляді HTTP-запиту) безпосередньо до бази даних MongoDB (у вигляді MongoDB-запиту).

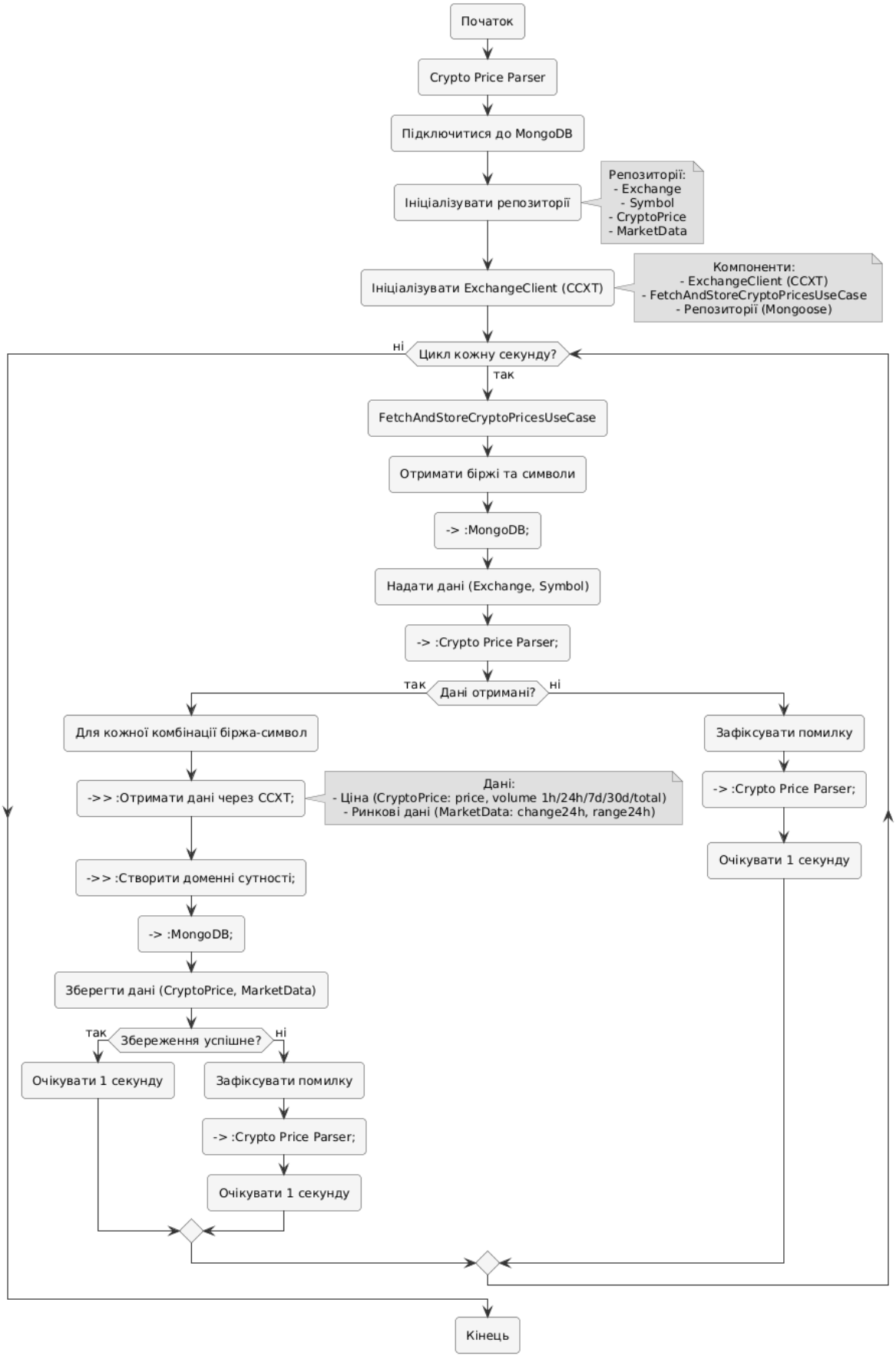

Рисунок 3.6. Алгоритм реалізації сервісу збору даних з криптобірж

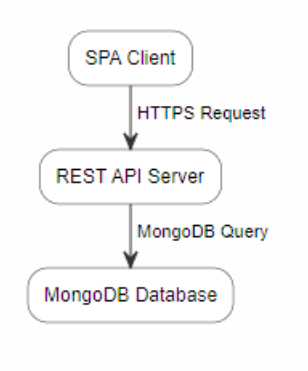

Рисунок 3.7. Схема мережевої взаємодії

Формат JSON-відповіді API представлено в лістингу 3.4.

Лістинг 3.4. Формат JSON-відповіді API

```
json id="5n0h8u"
{
   "symbol": "ETH/USDT",
   "exchange": "Binance",
   "price": 3845.12,
   "volume": 187654321,
   "timestamp": "2026-06-03T10:00:00Z"
}
```

Діаграма послідовності (рис. 3.8) ілюструє взаємодію системи, яка фокусується на конкретному ендпоінті бекенду (GET /markets) та процесі рендерингу графіків на стороні клієнта.

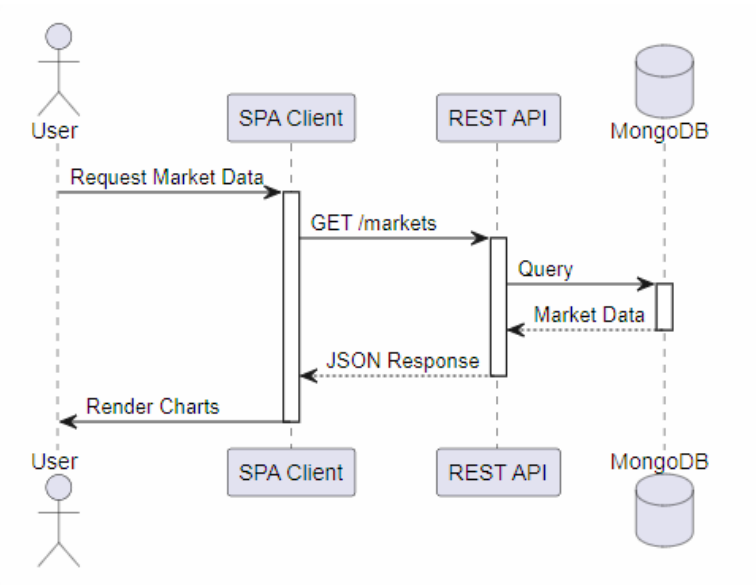

Рисунок 3.8. Діаграма послідовності

На рис. 3.9 зображено алгоритм роботи RESTful API сервісу.

3.5. Реалізація структури MongoDB

Для зберігання ринкової інформації використовується MongoDB.

Основними колекціями системи є:

- markets;
- exchanges;
- users;
- watchlists.

Структура документа Market на javascript приведена в лістингу 3.5.

Лістинг 3.5. Структура документа Market

```
MarketSchema = new mongoose.Schema({
     symbol: String,
     exchange: String,
     price: Number,
     volume: Number,
     timestamp: Date
});
```

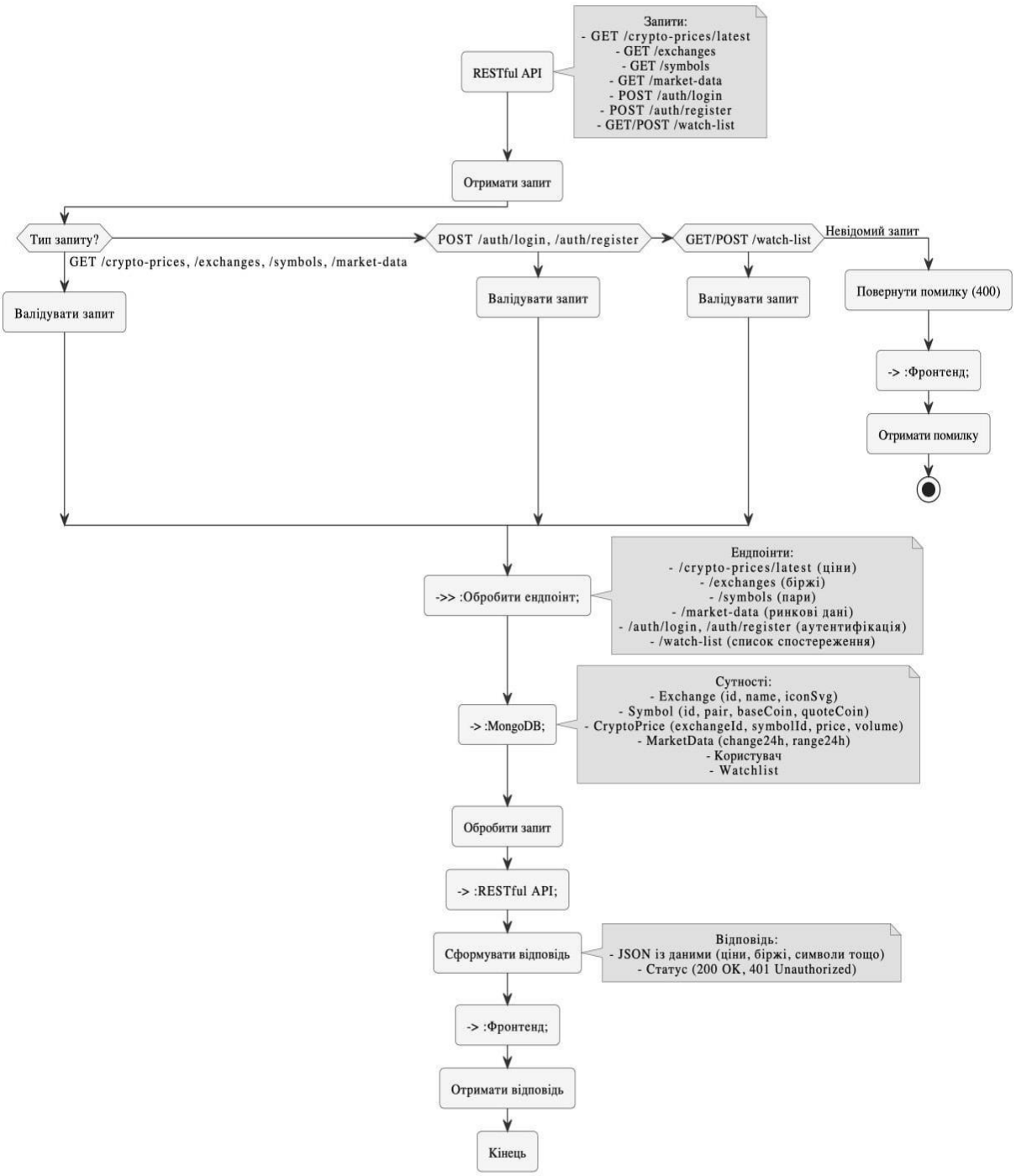

Рисунок 3.9. Алгоритм роботи RESTful API сервісу

ER-діаграма бази даних представлена на рис. 3.10.

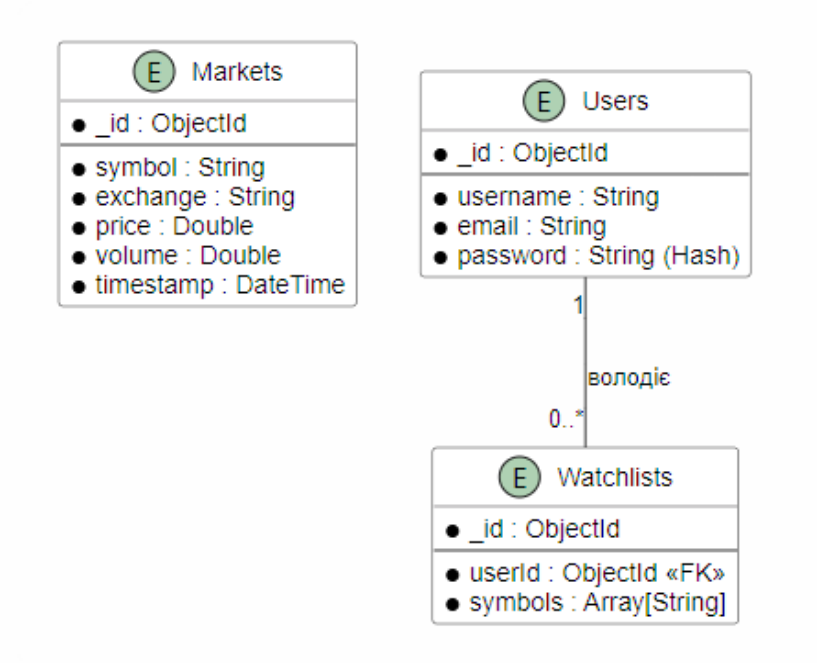

Рисунок 3.10. ER-діаграма

Діаграма описує сутності для збереження ринкових котирувань, облікових записів користувачів та їхніх персональних списків спостереження (Watchlists). Оскільки використовується документоорієнтована СУБД MongoDB, логічні зв'язки між цими колекціями зазвичай реалізуються через посилання за ідентифікаторами (References).Колекція Markets спроектована оптимально для збереження часових рядів (Time-Series) або звичайних документів котирувань.

При інтеграції з ETL-модулем збору даних, поля symbol, exchange, price та volume будуть заповнюватися нормалізованими даними з CCXT клієнта.

Зв'язок Users → Watchlists: поле userId у колекції списків спостереження виступає в ролі зовнішнього ключа (<<FK>>), який посилається на унікальний ідентифікатор користувача, пов'язуючи їх між собою.

MongoDB забезпечує:

- високу швидкість запису;
- масштабованість;
- підтримку JSON-документів;
- ефективну інтеграцію із Node.js.

3.6. Реалізація клієнтського SPA-застосунку

Клієнтська частина реалізована у вигляді Single Page Application. SPA-застосунок забезпечує:

- динамічне оновлення даних;
- інтерактивний інтерфейс;
- графічне відображення котирувань;
- фільтрацію криптовалют.

Структура інтерфейсу користувача представлена на рис. 3.11.

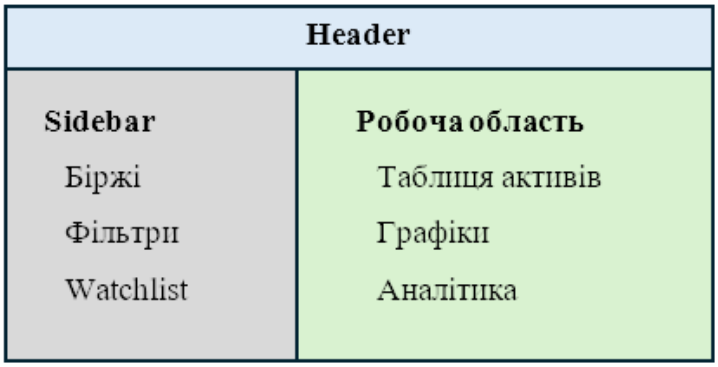

Рисунок 3.11. Структура інтерфейсу користувача

Схема компонентів SPA представлена на рис. 3.12. Головний кореневий компонент системи (App Component) керує та рендерить основні блоки користувацького інтерфейсу: шапку сайту (Header), таблицю з котируваннями (MarketTable) та графіки (Charts).

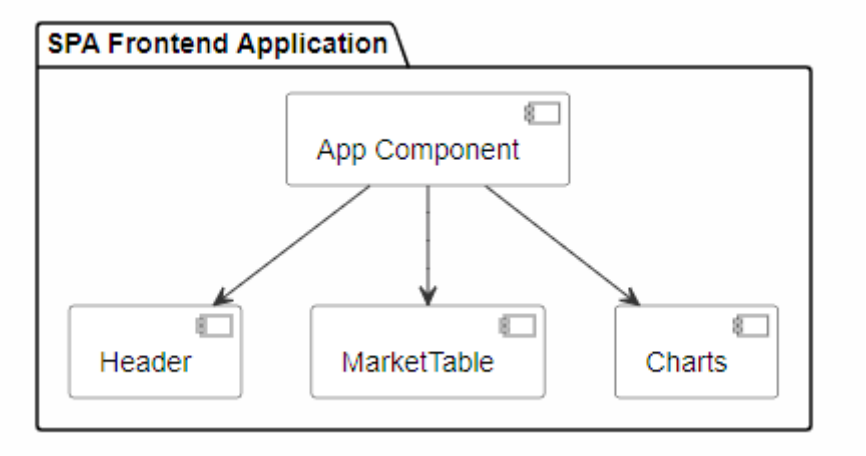

Рисунок 3.12. Схема компонентів SPA

Реалізація HTTP-запиту до API представлена в лістингу 3.6.

```
async function loadMarkets() {
   const response = await fetch('/api/markets');
   const data = await response.json();
   console.log(data);
}
```

Алгоритм роботи розробленого фронтенду представлено на рис. 3.13.

3.7. Реалізація візуалізації ринкових даних

Для графічного відображення даних використовуються:

- Chart.js;
- TradingView Widget.

Основними типами графіків є:

- лінійні графіки;
- гістограми;
- свічкові графіки.

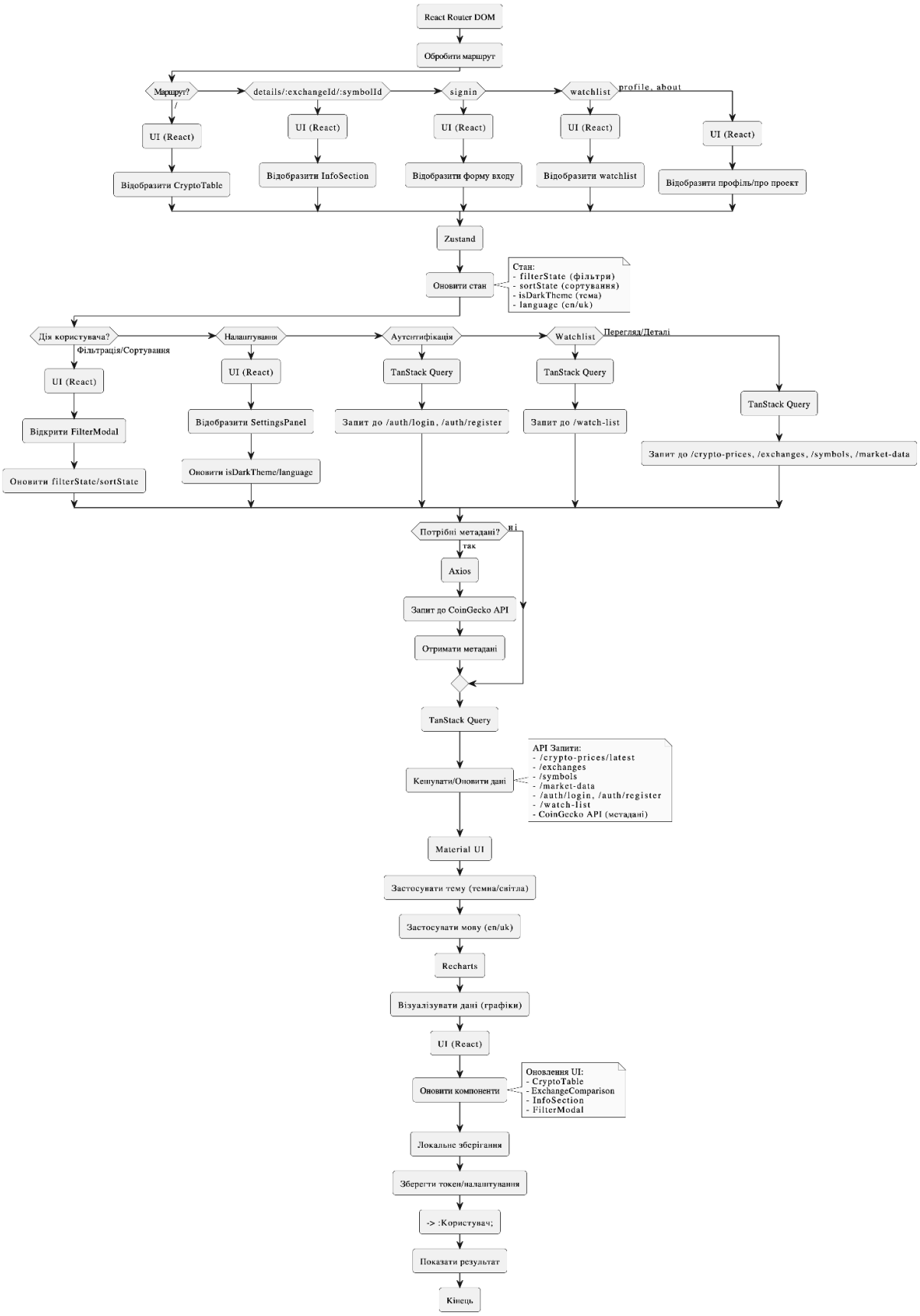

Рисунок 3.13. Алгоритм роботи фронтенду

Схема побудови графіків (рис. 3.14) деталізує технічний процес рендерингу графіків безпосередньо всередині SPA-застосунку. Вона показує, як отримані від API сирі дані проходять через фронтенд-компонент, деструктуруються чи готуються до потрібного формату, і передаються у фінальну бібліотеку візуалізації (Chart.js або TradingView Widget).

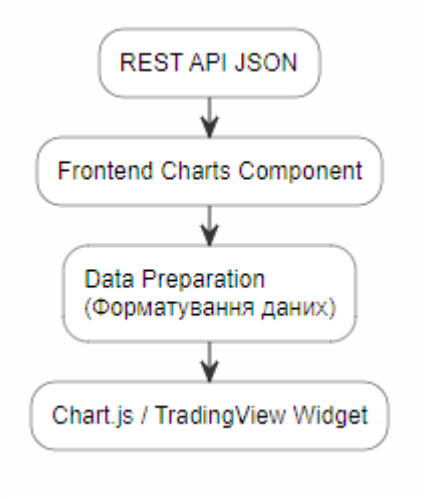

Рисунок 3.14. Схема побудови графіків

Приклад побудови графіка на javascript представлено в лістингу 3.7.

Лістинг 3.7. Приклад побудови графіка

```
new Chart(ctx, {
     type: 'line',
     data: {
          labels: labels,
          datasets: [{
               label: 'BTC Price',
               data: prices
          }]
     }
});
```

3.8. Реалізація механізмів фільтрації та сортування

Система підтримує:

- фільтрацію за біржами;

- сортування за ціною;
- сортування за обсягом торгів;
- пошук торгових пар.

Алгоритм фільтрації (рис. 3.15) деталізує логіку фільтрації ринкових даних на фронтенді (всередині MarketTable / Sidebar з фільтрами). Він описує класичний процес отримання повного масиву котирувань, застосування обраних користувачем критеріїв (наприклад, за конкретною біржею чи торговою парою) та рендеринг уже відфільтрованого результату в інтерфейсі.

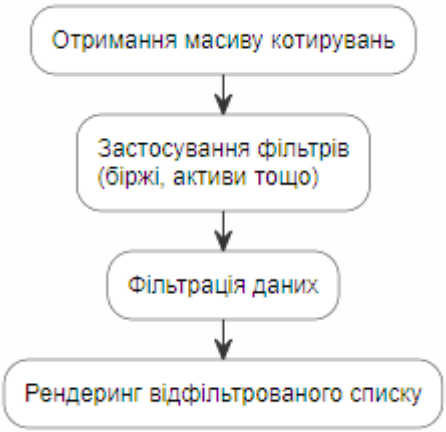

Рисунок 3.15. Схема алгоритму фільтрації

Коли користувач взаємодіє із Sidebar → Filters, цей ланцюжок перехоплює «сирі» дані, які прийшли у форматі JSON від API, відсікає зайве за критеріями, і миттєво оновлює таблицю на екрані.

Приклад сортування на javascript представлено в лістингу 3.8.

Лістинг 3.8. Приклад сортування

```
markets.sort((a, b) => b.price - a.price);
```

3.9. Реалізація системи авторизації

Для забезпечення захисту персональних даних реалізовано систему авторизації користувачів.

Використовуються:

- JWT-токени;
- bcrypt;
- middleware перевірки доступу.

Схема JWT-авторизації (рис. 3.16) описує дві сторони одного процесу: спочатку клієнт отримує токен, а потім використовує його для запитів.

Схема розбиває процес на два етапи (Авторизація та Доступ до даних), показуючи взаємодію між Користувачем, SPA-клієнтом, API-сервером (з Middleware для JWT) та Базою даних

Приклад генерації JWT на javascript приведено в лістингу 3.9.

Лістинг 3.9. Приклад генерації JWT

```
const token = jwt.sign(
      { id: user._id },
      process.env.JWT_SECRET
);
```

Висновки до розділу

У процесі розроблення веб-сервісу моніторингу криптовалют було реалізовано програмні модулі збору ринкових даних, REST API та клієнтський SPA-застосунок.

Було реалізовано:

- механізми автоматичного збору котирувань через CCXT;
- систему централізованого зберігання даних у MongoDB;
- RESTful API для взаємодії із фронтендом;
- SPA-інтерфейс із підтримкою графічного відображення даних;

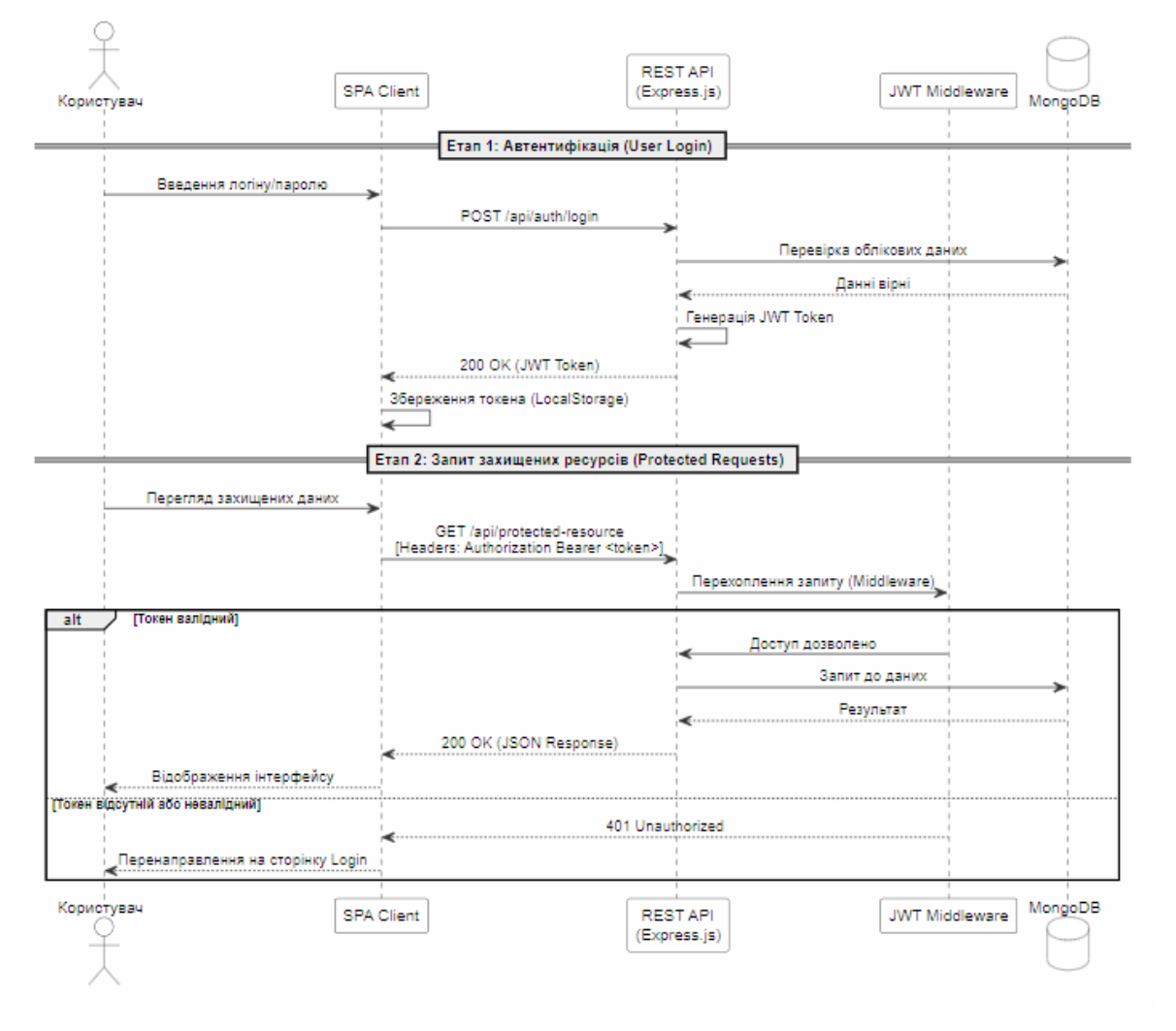

Рисунок 3.16. Схема JWT-авторизації

- механізми фільтрації, сортування та порівняння ринкових показників;
- систему JWT-авторизації.

Проведене тестування підтвердило працездатність програмного забезпечення, коректність взаємодії між структурними компонентами та стабільність функціонування веб-сервісу.

РОЗДІЛ 4. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ВЕБ-СЕРВІСУ МОНІТОРИНГУ КРИПТОВАЛЮТ

4.1. Опис розробленого веб-сервісу

Основним елементом користувацького інтерфейсу розробленого веб-сервісу є компонент CTable, який відображає таблицю з актуальними даними про торгові пари з різних криптовалютних бірж, таких як Binance, Coinbase і Kraken.

Цей компонент дозволяє користувачу переглядати ключові показники включно з поточною ціною, обсягами торгів за 1 годину, 24 години, 7 днів, 30 днів, а також динаміку зміни ціни за 24 години.

Цей табличний компонент розроблено з акцентом на зручність і функціональність, саме тому користувач може за своїм бажанням сортувати отриману інформацію за різними параметрами, в тому числі ціною або обсягом торгів, а також застосовувати фільтри по біржах або торгових парах. Компонент засновано на Material UI, що забезпечує адаптивне розташування стовпців і підтримку інтерактивних елементів, таких як кнопки для сортування. Завдяки інтеграції з TanStack Query дані в таблиці оновлюються в реальному часі, що дозволяє користувачам отримувати актуальну інформацію без затримок.

Головне вікно веб-сервісу представлене на рис. 4.1.

Вікно параметрів фільтрації інформації зображене на рис. 4.2.

Вікно детальної інформації за запитом проілюстроване на рис. 4.3.

Рисунок 4.4 демонструє розроблене вікно конвертера криптовалют.

Рисунок 4.5 зображує розроблене вікно порівняння ціни на криптобіржах.

Рисунок 4.6 та рисунок 4.7 представляють відповідно вікно реєстрації користувача та вікно авторизації користувача при вході у веб-сервіс.

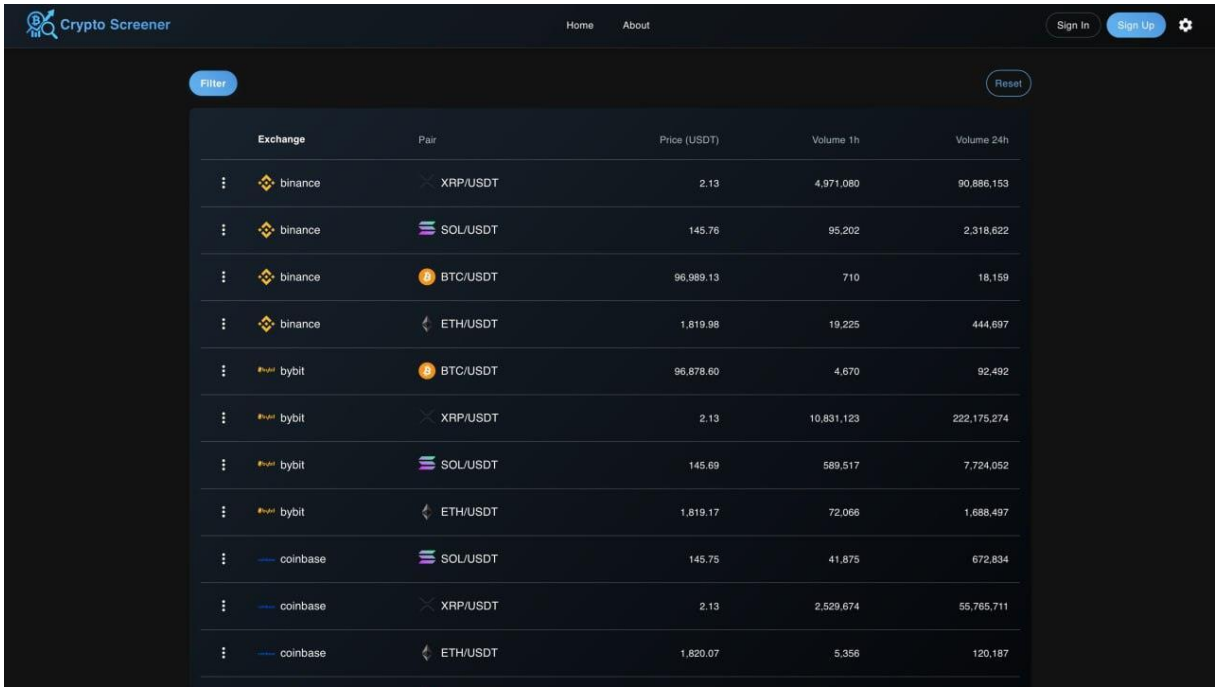

Рисунок 4.1. Головне вікно веб-сервісу та компонент CTable

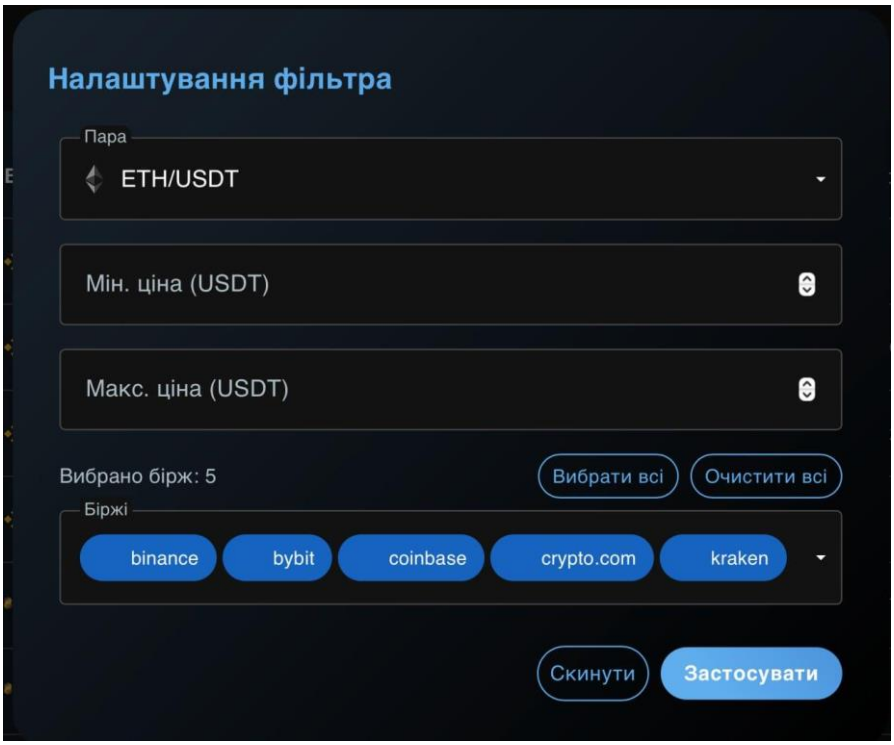

Рисунок 4.2. Вікно параметрів фільтрації інформації

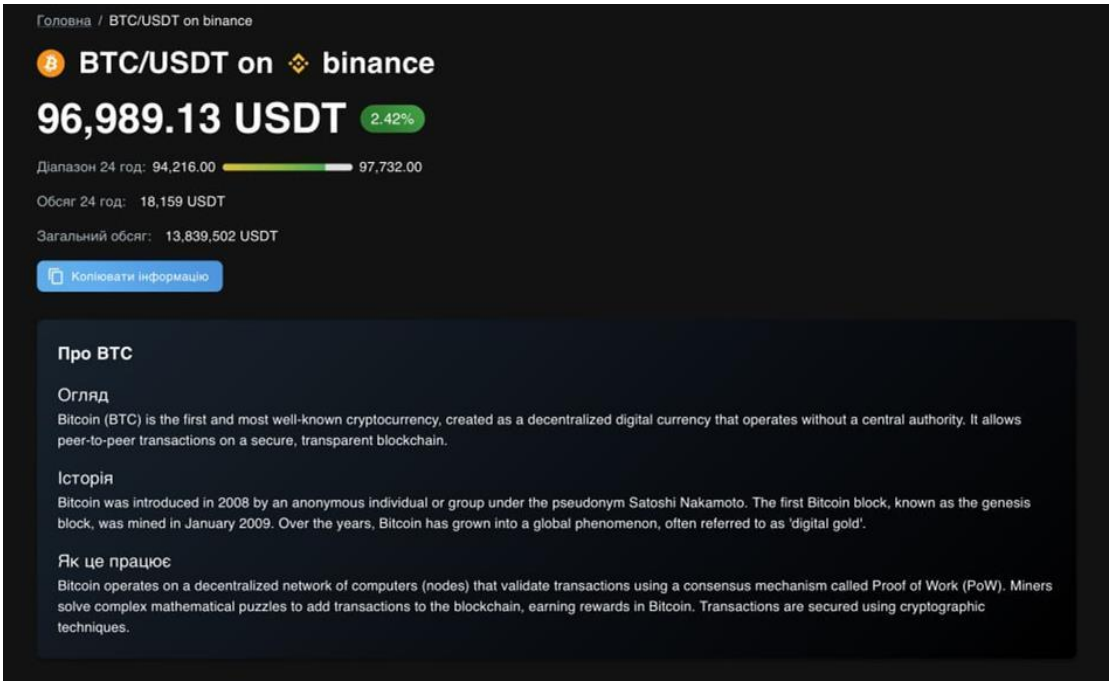

Рисунок 4.3. Вікно детальної інформації за запитом

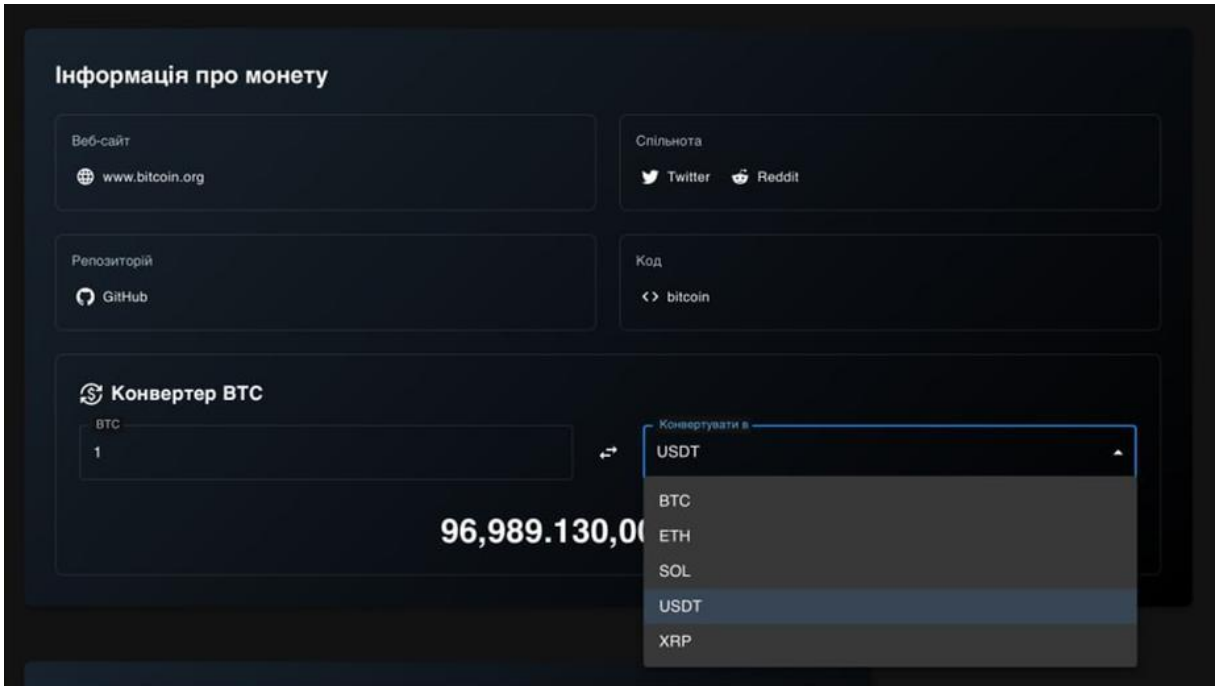

Рисунок 4.4. Вікно конвертера криптовалют

Рисунок 4.5. Вікно порівняння ціни на криптобіржах

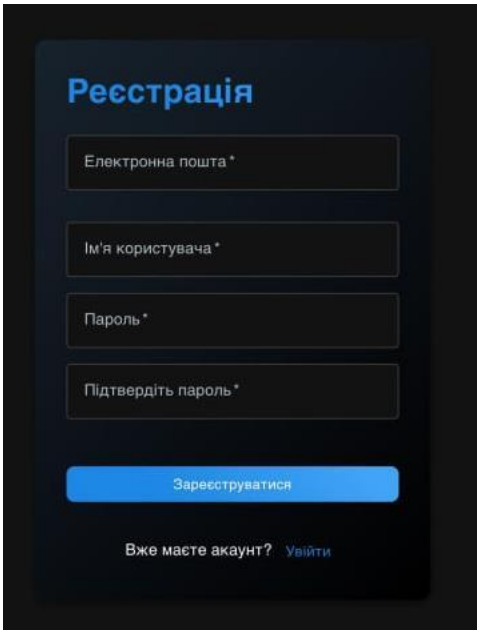

Рисунок 4.6. Вікно реєстрації користувача

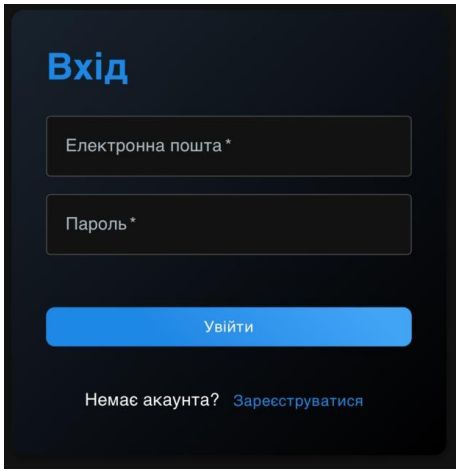

Рисунок 4.7. Вікно авторизації користувача

4.2. Мета та задачі тестування веб-сервісу

Тестування веб-сервісу моніторингу криптовалют проводилося з метою перевірки:

- коректності функціонування програмних модулів;
- стабільності взаємодії між компонентами системи;
- продуктивності сервісу;
- надійності REST API;
- правильності обробки ринкових даних;
- коректності відображення інформації у SPA-застосунку.

Основними задачами тестування є:

- перевірка модуля збору даних;
- тестування REST API;
- перевірка взаємодії з MongoDB;
- тестування SPA-інтерфейсу;
- перевірка механізмів авторизації;
- тестування продуктивності та навантаження;
- виявлення помилок і вузьких місць системи.

Загальна схема процесу тестування представлена на рис. 4.8.

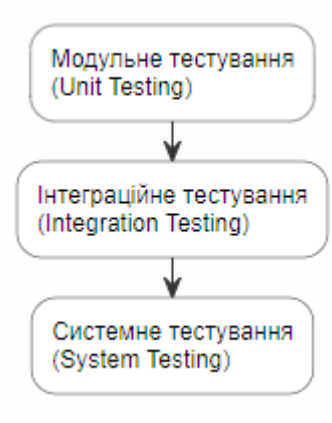

Рисунок 4.8. Загальна схема процесу тестування

Схема тестування частин веб-сервісу, де тестування охоплює всі рівні додатку від інтерфейсу користувача до збереження даних, представлена на рис. 4.9.

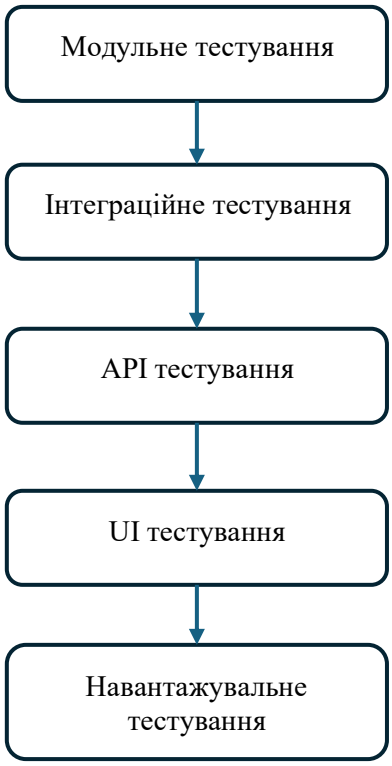

Рисунок 4.9. Схема тестування частин веб-сервісу

Для перевірки коректності роботи системи використано:

- модульне тестування;
- інтеграційне тестування;
- API-тестування;
- UI-тестування.

Схема рис. 4.9 ілюструє наскрізну стратегію верифікації веб-сервісу:

1. Тестування фронтенду полягає в перевірці правильності рендерингу компонентів (MarketTable, Charts) та логіки фільтрації даних на стороні клієнтського SPA-застосунку.

2. Тестування REST API полягає в перевірці правильності маршрутизації, роботи контролерів Express.js, коректності віддачі JSON-відповідей та роботи Middleware для захисту ресурсів (JWT).

Приклад API-тесту на javascript:

```
test(‘GET /markets’, async () => {
   const response = await request(app)
       .get('/api/markets');
   expect(response.statusCode).toBe(200);
}); ```
```

3. Тестування бази даних полягає в перевірці збереження нормалізованих котирувань від CCXT-колектора, а також валідація схем колекцій (Markets, Users, Watchlists) у базі даних MongoDB.

4.3. Модульне тестування програмних компонентів

Модульне тестування проводилося для окремих компонентів системи.

Перевірялися:

- модуль збору даних;
- REST API;
- алгоритми фільтрації;
- механізми сортування;

- JWT-авторизація;
- робота з MongoDB.

4.3.1. Тестування модуля збору даних

Метою тестування було:

- перевірити підключення до бірж;
- перевірити отримання котирувань;
- перевірити обробку JSON-відповідей;
- оцінити стабільність роботи CCXT.

Схема модульного тестування представлена на рис. 4.10.

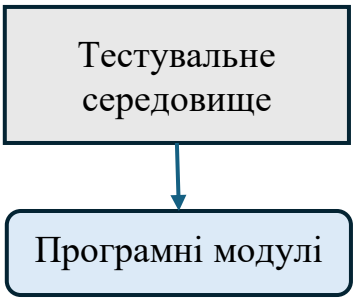

Рисунок 4.10. Схема модульного тестування

Для тестування використовувались:

- Jest;
- Supertest;
- Node.js testing utilities.

Приклад модульного тесту

```
test(‘Ticker loading’, async () => {
    const ticker = await exchange.fetchTicker('BTC/USDT');
    expect(ticker.symbol).toBe('BTC/USDT');
});
```

Для тестування модуля збору даних треба провести тест CCXT.

Схема тестування CCXT Collector представлена на рис. 4.11.

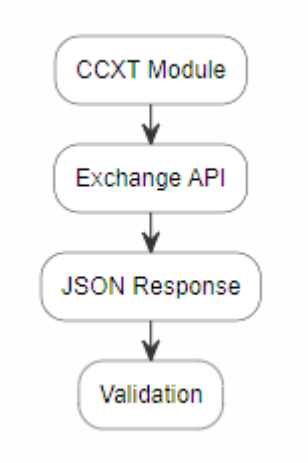

Рисунок 4.11. Схема тестування CCXT Collector

Результати тестування модуля збору даних представлено в табл. 4.1.

Таблиця 4.1. Результати тестування модуля збору даних

№	**Функція**	**Очікуваний результат**	**Результат**
1	Підключення до Binance	Отримання відповіді API	Успішно
2	Отримання котирувань	Коректні дані	Успішно
3	Обробка JSON	Запис у MongoDB	Успішно

4.3.2 Тестування REST API

Тестування REST API проводилося для перевірки:

- обробки HTTP-запитів;
- коректності JSON-відповідей;
- роботи маршрутизації;
- швидкодії API.

Основні REST-ендпоінти представлені в табл. 4.2.

Таблиця 4.2. REST-ендпоінти

Метод	**Endpoint**	**Призначення**
GET	`/api/markets`	Отримання списку активів
GET	`/api/ticker/:symbol`	Дані криптовалюти
GET	`/api/exchanges`	Список бірж
GET	`/api/history`	Історичні дані
POST	`/api/auth/login`	Авторизація

Схема тестування REST API представлена на рис. 4.12.

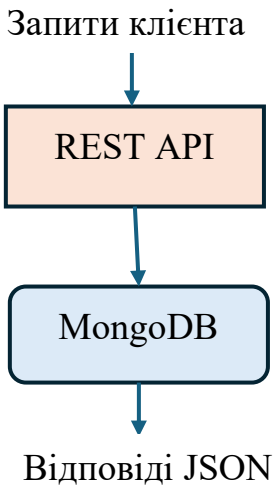

Рисунок 4.12. Схема тестування REST API

Приклад API-тесту приведено в лістингу 4.1.

Лістинг 4.1. Приклад API-тесту

```
test('GET /api/markets', async () => {
   const response = await request(app)
       .get('/api/markets');
   expect(response.statusCode).toBe(200);
   });
```

Результати API-тестування представлені в табл. 4.3.

Таблиця 4.3. Результати API-тестування

№	Endpoint	Результат
1	api/markets	Успішно
2	api/ticker/BTC	Успішно
3	/api/history	Успішно
4	api/auth/login	Успішно

4.3.3. Інтеграційне тестування системи

Інтеграційне тестування проводилося для перевірки взаємодії між:

- модулем збору даних;
- REST API;
- MongoDB;
- SPA-застосунком.

Схема інтеграційного тестування представлена на рис. 4.13.

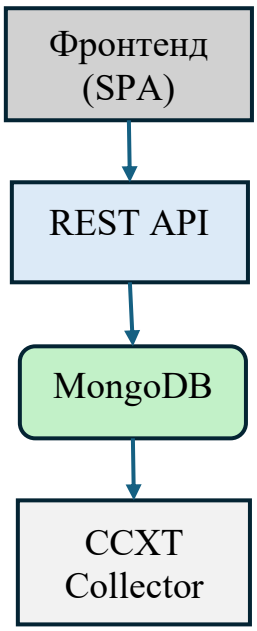

Рисунок 4.13. Схема інтеграційного тестування

Під час тестування перевірялися:

- передача даних між компонентами;
- синхронізація оновлення котирувань;
- стабільність API;
- коректність відображення графіків.

Результати інтеграційного тестування представлені в табл. 4.4.

Таблиця 4.4. Результати інтеграційного тестування

Компонент	**Перевірка**	**Результат**
CCXT → MongoDB	Запис котирувань	Успішно
MongoDB → REST API	Отримання даних	Успішно
REST API → SPA	Візуалізація	Успішно

4.3.4. Тестування клієнтського SPA-застосунку

Тестування SPA-застосунку проводилося для перевірки:

- адаптивності інтерфейсу;
- роботи компонентів;
- швидкості оновлення інформації;
- роботи фільтрів та графіків.

Схема тестування інтерфейсу представлена на рис. 4.14.

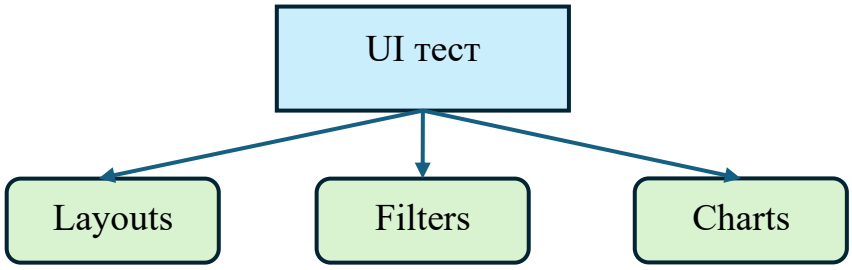

Рисунок 4.14. Схема тестування UI

При перевірці інтерфейсів користувача було протестовано:

- таблицю криптовалют;
- систему сортування;
- пошук активів;
- модуль графіків;
- адаптивність для мобільних пристроїв.

Результати тестування інтерфейсу користувача представлено в табл. 4.5.

Таблиця 4.5. Результати тестування інтерфейсу користувача

Елемент	**Очікуваний результат**	**Результат**
Таблиця активів	Відображення котирувань	Успішно
Графіки	Коректна побудова	Успішно
Фільтри	Оновлення списку	Успішно
Адаптивність	Робота на смартфонах	Успішно

Таблиця 4.6 демонструє отримані загальні результати тестування.

Таблиця 4.6. Загальні результати тестування

№	**Модуль**	**Очікуваний результат**	**Результат**
1	REST API	Отримання даних	Успішно
2	MongoDB	Запис котирувань	Успішно
3	CCXT Collector	Отримання котирувань	Успішно
4	SPA Frontend	Відображення графіків	Успішно
5	JWT Auth	Авторизація користувача	Успішно

4.4. Тестування продуктивності та навантаження

Навантажувальне тестування проводилося для оцінки:

- швидкодії REST API;
- стабільності MongoDB;
- продуктивності SPA;

- обробки великої кількості запитів.

Схема навантажувального тестування представлена на рис. 4.15.

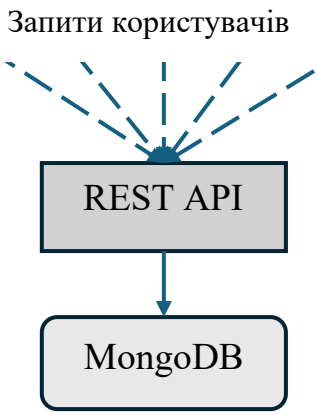

Рисунок 4.15. Схема навантажувального тестування

Під час тестування оцінювалися:

- час відповіді API;
- навантаження на сервер;
- використання оперативної пам’яті;
- кількість одночасних підключень.

Результати навантажувального тестування представлені в табл. 4.7.

Таблиця 4.7. Результати навантажувального тестування

Параметр	**Значення**
Середній час відповіді API	120 мс
Кількість одночасних користувачів	500
Середнє використання CPU	48 %
Середнє використання RAM	1.8 ГБ

Отже, отримані результати цілком підтверджують достатню продуктивність веб-сервісу.

4.5. Забезпечення безпеки веб-сервісу

Для забезпечення безпечності системи було реалізовано:

- JWT-авторизацію;
- хешування паролів;
- валідацію запитів;
- захист REST API;
- middleware безпеки.

Схема безпеки системи представлена на рис. 4.16.

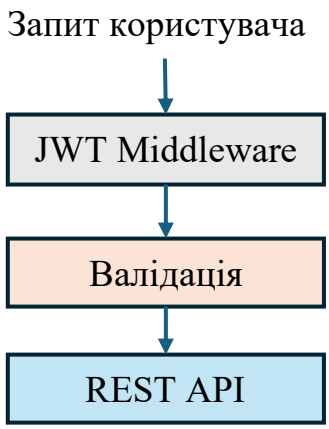

Рисунок 4.16. Схема безпеки системи

Тестування механізмів безпеки

Було перевірено:

- авторизацію користувачів;
- перевірку JWT-токенів;
- обмеження несанкціонованого доступу;
- захист API-маршрутів.

4.6. Впровадження веб-сервісу

Впровадження системи передбачає:

- розгортання серверної частини;

- налаштування MongoDB;
- запуск SPA-застосунку;
- конфігурацію API;
- інтеграцію із криптовалютними біржами.

Схема розгортання системи представлена на рис. 4.17

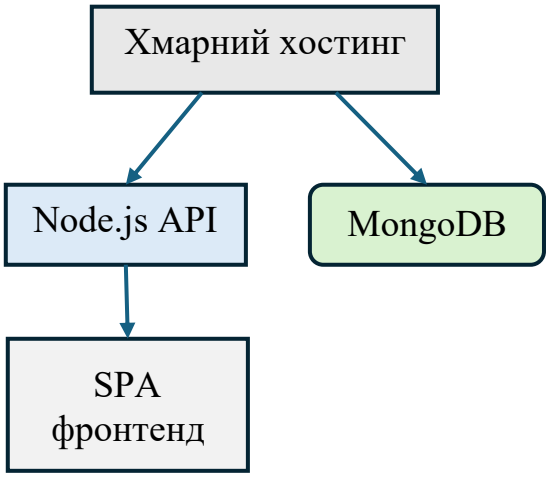

Рисунок 4.17. Схема розгортання системи

Етапи впровадження

1. Підготовка серверного середовища;
2. Встановлення Node.js та MongoDB;
3. Налаштування змінних середовища;
4. Запуск REST API;
5. Розгортання SPA-застосунку;
6. Налаштування HTTPS;
7. Тестування працездатності.

Приклад запуску серверної частини

```
bash id="4x5vbc" npm install npm start
```

4.7. Перспективи розвитку системи

Подальший розвиток веб-сервісу може включати:

- підтримку WebSocket;
- реалізацію push-сповіщень;
- інтеграцію машинного навчання;
- автоматичне формування торгових сигналів;
- підтримку мобільного застосунку;
- інтеграцію із біржовими акаунтами користувачів.

Висновки до розділу

У процесі тестування веб-сервісу моніторингу криптовалют було перевірено працездатність усіх програмних компонентів системи, включно з модулем збору даних, REST API, MongoDB та SPA-застосунком.

Було проведено:

- модульне тестування;
- інтеграційне тестування;
- UI-тестування;
- навантажувальне тестування;
- тестування безпеки.

Результати тестування підтвердили:

- коректність функціонування системи;
- стабільність взаємодії між компонентами;
- достатню продуктивність;
- готовність веб-сервісу до впровадження.

Було визначено основні етапи розгортання системи та перспективи подальшого розвитку програмного забезпечення.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено веб-сервіс моніторингу криптовалют, призначений для автоматизованого збору, обробки, зберігання та візуалізації ринкових даних цифрових активів. Розроблена система забезпечує підтримку трейдерів, інвесторів та аналітиків у процесі прийняття торгових рішень шляхом надання актуальної інформації про криптовалютний ринок.

У процесі виконання роботи було проведено аналіз предметної області моніторингу криптовалют та досліджено сучасні веб-сервіси криптовалютної аналітики, серед яких CoinMarketCap, CoinGecko та TradingView. Проведений аналіз дозволив визначити основні переваги та недоліки існуючих рішень, а також сформувати функціональні та нефункціональні вимоги до власного веб-сервісу.

На основі результатів аналізу було обґрунтовано вибір сучасного технологічного стеку, до складу якого увійшли:

- Node.js та Express.js для реалізації серверної частини;
- MongoDB для зберігання ринкових даних;
- бібліотека CCXT для взаємодії з API криптовалютних бірж;
- React або Vue.js для створення SPA-застосунку;
- RESTful API для організації взаємодії між компонентами системи.

У роботі було спроектовано триланкову архітектуру веб-сервісу, що включає:

- модуль автоматичного збору ринкових даних;
- серверну частину REST API;
- клієнтський SPA-застосунок.

Запропонована архітектура забезпечує модульність, масштабованість, централізацію бізнес-логіки та можливість подальшого розширення функціональних можливостей системи.

У процесі розроблення програмного забезпечення було реалізовано:

- автоматичний збір котирувань та обсягів торгів із криптовалютних бірж;
- уніфіковану взаємодію з API бірж через бібліотеку CCXT;
- систему централізованого зберігання ринкових даних у MongoDB;
- RESTful API для взаємодії між серверною та клієнтською частинами;
- SPA-інтерфейс із підтримкою фільтрації, сортування та пошуку активів;
- механізми графічного відображення ринкової інформації;
- систему порівняння цін між торговими платформами;
- JWT-авторизацію та механізми забезпечення безпеки.

Особливу увагу було приділено реалізації механізмів обробки динамічних ринкових даних та інтерактивної візуалізації інформації. Для побудови графіків та відображення аналітичної інформації використано сучасні бібліотеки візуалізації даних.

У четвертому розділі роботи проведено комплексне тестування веб-сервісу, яке включало:

- модульне тестування;
- інтеграційне тестування;
- тестування REST API;
- UI-тестування;
- навантажувальне тестування;
- тестування механізмів безпеки.

Результати тестування підтвердили:

- коректність функціонування програмних модулів;
- стабільність взаємодії між компонентами системи;
- достатню продуктивність веб-сервісу;

- працездатність REST API;
- готовність програмного забезпечення до впровадження.

Практичне значення отриманих результатів полягає у створенні працездатного масштабованого веб-сервісу моніторингу криптовалют, який може використовуватися:

- трейдерами;
- інвесторами;
- фінансовими аналітиками;
- навчальними та дослідницькими проєктами;
- сервісами криптовалютної аналітики.

Розроблений веб-сервіс забезпечує централізований моніторинг криптовалютного ринку та може бути основою для подальшого розвитку більш складних аналітичних платформ.

Перспективами подальшого розвитку системи є:

- підтримка WebSocket-з’єднань для отримання даних у режимі реального часу;
- інтеграція алгоритмів машинного навчання;
- автоматичне формування торгових сигналів;
- реалізація push-сповіщень;
- створення мобільного застосунку;
- інтеграція із біржовими акаунтами користувачів;
- розширення аналітичного функціоналу системи.

Таким чином, поставлену мету кваліфікаційної роботи досягнуто, а всі поставлені задачі успішно вирішено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CoinMarketCap. Cryptocurrency Prices, Charts And Market Capitalizations. URL: https://coinmarketcap.com/

2. CoinGecko. Cryptocurrency Prices, Charts, and Crypto Market Cap. URL: https://www.coingecko.com/

3. TradingView. Cryptocurrency Market Overview. URL: https://www.tradingview.com/markets/cryptocurrencies/

4. MongoDB Documentation. MongoDB Atlas for NoSQL Databases. URL: https://www.mongodb.com/docs/

5. CCXT Documentation. Unified Cryptocurrency Exchange APIs. URL: https://ccxt.readthedocs.io/en/latest/

6. React 19 Documentation. React Official Website. URL: https://react.dev/

7. Material UI 7 Documentation. MUI: The React Component Library. URL: https://mui.com/

8. TanStack Query Documentation. React Query for Data Fetching. URL: https://tanstack.com/query/

9. Zustand Documentation. A Small, Fast and Scalable State-Management Solution. URL: https://zustand-demo.pmnd.rs/

10. Recharts Documentation. A Composable Charting Library Built on React Components. URL: https://recharts.org/

11. Vite Documentation. Next Generation Frontend Tooling. URL: https://vitejs.dev/

12. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley, 2003. 560 p.

13. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

14. Feature-Sliced Design Documentation. An Architectural Methodology for Frontend Projects. URL: https://feature-sliced.github.io/documentation/

15. TypeScript Documentation. TypeScript is JavaScript with syntax for types. URL: https://www.typescriptlang.org/docs/

16. Express.js Documentation. Fast, unopinionated, minimalist web framework for Node.js. URL: https://expressjs.com/

17. Mongoose Documentation. Elegant MongoDB Object Modeling for Node.js. URL: https://mongoosejs.com/docs/

18. Jest Documentation. Delightful JavaScript Testing Framework. URL: https://jestjs.io/docs/

19. Петренко А.В. Використання TypeScript у розробці сучасних вебдодатків. Науковий журнал НТУУ «КПІ», 2024. № 5, С. 89–95.

20. Коваленко О.І. Архітектурні підходи до розробки масштабованих вебдодатків. Вісник Київського університету, 2023. С. 67–74.

Додаток А.

Лістинг програмних модулів

Головні UI компоненти

Settings Panel

```
import {
 Popover,
 Box,
 Typography,
 FormControl,
 InputLabel,
 Select,
 MenuItem,
 Switch,
 Fade,
} from '@mui/material';
import { useScreenerStore } from '../model/store';

interface SettingsPanelProps {
 open: boolean;
 anchorEl: HTMLElement | null;
 onClose: () => void;
}

export const SettingsPanel = ({ open, anchorEl, onClose }:
SettingsPanelProps) => {
 const { isDarkTheme, toggleTheme, language, setLanguage }
= useScreenerStore();

 return (
  <Popover
   open={open}
   anchorEl={anchorEl}
   onClose={onClose}
   anchorOrigin={{
    vertical: 'bottom',
    horizontal: 'right',
   }}
   transformOrigin={{
    vertical: 'top',
    horizontal: 'right',
   }}
   TransitionComponent={Fade}
   sx={{
    '& .MuiPopover-paper': {
     background: (theme) =>
      theme.palette.mode === 'dark'
       ? 'linear-gradient(135deg, rgb(24, 35, 45) 0%, rgba(0,
0, 0, 1) 100%)'
       : 'linear-gradient(135deg, rgba(255, 255, 255, 1) 0%,
rgba(227, 242, 253, 1) 100%)',
     borderRadius: 6,
     border: (theme) => `0px solid ${theme.palette.divider}`,
     p: 2,
     minWidth: 250,
    },
   }}
  >
   <Box sx={{ display: 'flex', flexDirection: 'column', gap: 2
}}>
    <Box sx={{ display: 'flex', alignItems: 'center',
justifyContent: 'space-between' }}>
     <Typography variant="body1" color="text.primary">
      {language === 'en' ? 'Language' : 'Мова'}
     </Typography>
     <FormControl sx={{ minWidth: 120 }}>
      <InputLabel>{language === 'en' ? 'Language' :
'Мова'}</InputLabel>
      <Select
       value={language}
       onChange={(e) => setLanguage(e.target.value as 'en' |
'uk')}
       size="small"
       sx={{ bgcolor: 'background.default', borderRadius: 1
}}
      >
       <MenuItem value="en">English</MenuItem>
       <MenuItem value="uk">Ukrainian</MenuItem>
      </Select>
     </FormControl>
    </Box>
    <Box sx={{ display: 'flex', alignItems: 'center',
justifyContent: 'space-between' }}>
     <Typography variant="body1" color="text.primary">
      {language === 'en' ? 'Theme' : 'Тема'}
     </Typography>
     <Switch
      checked={isDarkTheme}
      onChange={toggleTheme}
      color="primary"
      sx={{ '& .MuiSwitch-track': { bgcolor: (theme) =>
theme.palette.divider } }}
     />
    </Box>
   </Box>
  </Popover>
 );
};
```

Coin Description

```
import { Box, Tabs, Tab, Typography } from '@mui/material';
import { useScreenerStore } from '../model/store';
import { coinDescriptions } from '../mocks/coinData';
import { useState } from 'react';

interface CoinDescriptionProps {
 symbolId: string;
}

export const CoinDescription = ({ symbolId }:
CoinDescriptionProps) => {
 const { language } = useScreenerStore();
 const [tab, setTab] = useState('history');
 const description = coinDescriptions.find((desc: any) =>
desc.symbolId === symbolId) || {
  history: language === 'en' ? 'No description available.' :
'Опис відсутній.',
  technology: '',
  application: '',
 };
 const handleTabChange = (_event: React.SyntheticEvent,
newTab: string) => {
  setTab(newTab);
 };
```

```
  return (
    <Box sx={{ bgcolor: 'background.paper', borderRadius: 2, p:
2, boxShadow: (theme) => theme.shadows[3] }}>
      <Tabs value={tab} onChange={handleTabChange} aria-
label="coin description">
        <Tab label={language === 'en' ? 'History' : 'Ιστορία'}
value="history" />
        <Tab label={language === 'en' ? 'Technology' :
'Τεχνολογία'} value="technology" />
        <Tab label={language === 'en' ? 'Application' :
'Εφαρμογασμοί'} value="application" />
      </Tabs>
      <Box sx={{ mt: 2 }}>
        <Typography variant="body1" color="text.primary">
          {description[tab as keyof typeof description]}
        </Typography>
      </Box>
    </Box>
  );
};
```

Crypto Table

```
import { useMemo, useCallback, useState } from 'react';
import {
  Box,
  Paper,
  Typography,
  Table,
  TableBody,
  TableCell,
  TableContainer,
  TableHead,
  TableRow,
  TableSortLabel,
  Button,
  IconButton,
  Menu,
  MenuItem,
  Fade,
  Snackbar,
  Alert,
} from '@mui/material';
import { FilterList, MoreVert, ContentCopy, Info, Star } from
'@mui/icons-material';
import { useCryptoPrices, useExchanges, useSymbols } from
'../api/cryptoApi';
import { ScreenerData, FilterState } from '../model/types';
import { useScreenerStore } from '../model/store';
import { formatNumber } from '@/shared/lib/formatNumber';
import { sortData } from '@/shared/lib/sortData';
import { FilterModal } from './FilterModal';
import { Link } from 'react-router-dom';
import { useWatchlist } from '../../watchlist/useWatchlist';
import { useAuth } from '../../auth/AuthContext';

type Order = 'asc' | 'desc' | undefined;

interface SortState {
  column: keyof ScreenerData | null;
  order: Order;
}

interface ExtendedScreenerData extends ScreenerData {
  exchangeId: string;
  symbolId: string;
}

export const CryptoTable = () => {
  const { data: prices, isLoading, error } = useCryptoPrices();
  const { data: exchanges } = useExchanges();
  const { data: symbols } = useSymbols();
  const { filterState, sortState, setSortState, resetFilters,
language } = useScreenerStore();
  const { user } = useAuth();
  const { watchlist, addToWatchlist } = useWatchlist();
  const [openFilterModal, setOpenFilterModal] =
useState(false);
  const [anchorEl, setAnchorEl] = useState<null |
HTMLElement>(null);
  const [selectedRow, setSelectedRow] =
useState<ExtendedScreenerData | null>(null);
  const [openSnackbar, setOpenSnackbar] = useState(false);
  const [snackbarMessage, setSnackbarMessage] = useState('');
  const [snackbarSeverity, setSnackbarSeverity] =
useState<'success' | 'error'>('success');

  const tableData: ExtendedScreenerData[] = useMemo(() => {
    if (!prices || !exchanges || !symbols) return [];

    let filteredPrices = prices;

    if (filterState.exchanges.length > 0) {
      filteredPrices = filteredPrices.filter((price) =>
        filterState.exchanges.includes(price.exchangeId)
      );
    }

    if (filterState.pair) {
      filteredPrices = filteredPrices.filter((price) =>
        symbols.find((s) => s.id === price.symbolId)?.pair ===
filterState.pair
      );
    }

    const minPrice = filterState.minPrice ?? 0;
    const maxPrice = filterState.maxPrice ??
Number.MAX_VALUE;

    filteredPrices = filteredPrices.filter((price) =>
      price.price.value >= minPrice && price.price.value <=
maxPrice
    );

    return filteredPrices.map((price) => {
      const symbol = symbols.find((s) => s.id ===
price.symbolId);
      const exchange = exchanges.find((e) => e.id ===
price.exchangeId);
      return {
        pair: symbol?.pair || 'N/A',
        price: price.price.value,
        volume1h: price.volume['1h'],
        volume24h: price.volume['24h'],
        exchange: exchange?.name || 'N/A',
        exchangeId: price.exchangeId,
        symbolId: price.symbolId,
      };
    });
  }, [prices, exchanges, symbols, filterState]);

  const sortedData = useMemo(() => {
    if (!sortState.column) {
      return sortData(tableData, 'exchange', 'asc');
    }
    return sortData(tableData, sortState.column, sortState.order);
  }, [tableData, sortState]);
```

```
  const handleSort = useCallback(
    (column: keyof ScreenerData) => () => {
      if (column === 'exchange') return;

      const isSameColumn = sortState.column === column;
      let newOrder: Order = 'asc';

      if (isSameColumn) {
        if (sortState.order === 'asc') newOrder = 'desc';
        else if (sortState.order === 'desc') newOrder = undefined;
      }

      setSortState({
        column: newOrder ? column : null,
        order: newOrder,
      });
    },
    [sortState, setSortState]
  );

  const handleMenuClick = (event:
React.MouseEvent<HTMLElement>, row:
ExtendedScreenerData) => {
    setAnchorEl(event.currentTarget);
    setSelectedRow(row);
  };

  const handleMenuClose = () => {
    setAnchorEl(null);
    setSelectedRow(null);
  };

  const handleCopyInfo = () => {
    if (!selectedRow) return;

    const info = `
      Exchange: ${selectedRow.exchange}
      Pair: ${selectedRow.pair}
      Price: ${formatNumber(selectedRow.price, 2)} USDT
      Volume 1h: ${formatNumber(selectedRow.volume1h, 0)}
      Volume 24h: ${formatNumber(selectedRow.volume24h,
0)}
    `.trim();

    navigator.clipboard.writeText(info).then(() => {
      setSnackbarMessage(language === 'en' ? 'Copied to
clipboard!' : 'Скопійовано до буфера обміну!');
      setSnackbarSeverity('success');
      setOpenSnackbar(true);
    });

    handleMenuClose();
  };

  const handleAddToWatchlist = () => {
    if (!selectedRow) return;

    if (!user) {
      setSnackbarMessage(language === 'en' ? 'Please login to
add to watchlist' : 'Будь ласка, увійдіть, щоб додати до
списку спостереження');
      setSnackbarSeverity('error');
      setOpenSnackbar(true);
      handleMenuClose();
      return;
    }

    const isAlreadyInWatchlist = watchlist.some(
      (item) => item.symbolId === selectedRow.symbolId &&
item.exchangeId === selectedRow.exchangeId
    );

    if (isAlreadyInWatchlist) {
      setSnackbarMessage(language === 'en' ? 'Already in
watchlist' : 'Вже у списку спостереження');
      setSnackbarSeverity('error');
      setOpenSnackbar(true);
      handleMenuClose();
      return;
    }

    addToWatchlist({ symbolId: selectedRow.symbolId,
exchangeId: selectedRow.exchangeId });
    setSnackbarMessage(language === 'en' ? 'Added to
watchlist!' : 'Додано до списку спостереження!');
    setSnackbarSeverity('success');
    setOpenSnackbar(true);
    handleMenuClose();
  };

  const handleSnackbarClose = () => {
    setOpenSnackbar(false);
    setSnackbarMessage('');
  };

  const handleReset = () => {
    resetFilters();
  };

  if (isLoading) return <Typography>{language === 'en' ?
'Loading...' : 'Завантаження...'}</Typography>;
  if (error) return <Typography color="error">{language ===
'en' ? 'Error: ${error.message}' : 'Помилка:
${error.message}'}</Typography>;

  return (
    <Box sx={{ maxWidth: 1200, mx: 'auto', pt: 12, mb: 4,
minHeight: '80vh' }}>
      <Box display="flex" justifyContent="space-between"
alignItems="center" mb={2}>
        <Button
          variant="contained"
          color="primary"
          onClick={() => setOpenFilterModal(true)}
          sx={{
            borderRadius: 8,
            textTransform: 'none',
            fontWeight: 600,
            background: (theme) =>
              theme.palette.mode === 'dark'
                ? 'linear-gradient(45deg, #80aeed 30%, #4e96de
90%)'
                : 'linear-gradient(45deg, #1976d2 30%, #0d47a1
90%)',
            boxShadow: '0 2px 8px 0 rgba(25, 118, 210, 0.15)',
            color: '#fff',
            '&:hover': {
              background: (theme) =>
                theme.palette.mode === 'dark'
                  ? 'linear-gradient(45deg, #4e96de 30%, #80aeed
90%)'
                  : 'linear-gradient(45deg, #0d47a1 30%, #1976d2
90%)',
              boxShadow: '0 4px 16px 0 rgba(25, 118, 210, 0.18)',
            },
          }}
        >
```

```
        {language === 'en' ? 'Filter' : 'Φiльтр'}
      </Button>
      <Button
        onClick={handleReset}
        sx={{
          borderRadius: 8,
          textTransform: 'none',
          color: (theme) => theme.palette.mode === 'dark' ?
'#60aeef' : '#1976d2',
          border: '1px solid',
          borderColor: (theme) => theme.palette.mode === 'dark'
? '#60aeef' : '#1976d2',
          background: 'transparent',
          '&:hover': {
            background: (theme) =>
              theme.palette.mode === 'dark'
                ? 'rgba(96, 174, 238, 0.08)'
                : 'rgba(25, 118, 210, 0.08)',
          },
        }}
      >
        {language === 'en' ? 'Reset' : 'Сκинуτь'}
      </Button>
    </Box>
    <Paper
      elevation={0}
      sx={{
        p: 2,
        borderRadius: 2,
        background: (theme) =>
          theme.palette.mode === 'dark'
            ? 'linear-gradient(135deg, rgb(24, 35, 45) 0%, rgba(0,
0, 0, 1) 100%)'
            : 'linear-gradient(135deg, rgba(255, 255, 255, 1) 0%,
rgba(227, 242, 253, 1) 100%)',
        boxShadow: (theme) => theme.shadows[3],
      }}
    >
      <TableContainer>
        <Table sx={{ '& th, & td': { borderColor: 'divider' } }}>
          <TableHead>
            <TableRow>
              <TableCell sx={{ width: 50 }} />
              <TableCell sx={{ fontWeight: 'bold', color:
'text.primary' }}>
                {language === 'en' ? 'Exchange' : 'Бipжa'}
              </TableCell>
              <TableCell>
                <TableSortLabel
                  active={sortState.column === 'pair'}
                  direction={sortState.column === 'pair' ?
sortState.order : 'asc'}
                  onClick={handleSort('pair')}
                  sx={{
                    '& .MuiTableSortLabel-icon': {
                      color: (theme) =>
                        sortState.column === 'pair' ? 'primary.main' :
'text.secondary',
                    },
                  }}
                >
                  {language === 'en' ? 'Pair' : 'Пара'}
                </TableSortLabel>
              </TableCell>
              <TableCell align="right">
                <TableSortLabel
                  active={sortState.column === 'price'}
                  direction={sortState.column === 'price' ?
sortState.order : 'asc'}
                  onClick={handleSort('price')}
                  sx={{
                    '& .MuiTableSortLabel-icon': {
                      color: (theme) =>
                        sortState.column === 'price' ? 'primary.main' :
'text.secondary',
                    },
                  }}
                >
                  {language === 'en' ? 'Price (USDT)' : 'Цiна
(USDT)'}
                </TableSortLabel>
              </TableCell>
              <TableCell align="right">
                <TableSortLabel
                  active={sortState.column === 'volume1h'}
                  direction={sortState.column === 'volume1h' ?
sortState.order : 'asc'}
                  onClick={handleSort('volume1h')}
                  sx={{
                    '& .MuiTableSortLabel-icon': {
                      color: (theme) =>
                        sortState.column === 'volume1h' ?
'primary.main' : 'text.secondary',
                    },
                  }}
                >
                  {language === 'en' ? 'Volume 1h' : 'Об'єм 1 год'}
                </TableSortLabel>
              </TableCell>
              <TableCell align="right">
                <TableSortLabel
                  active={sortState.column === 'volume24h'}
                  direction={sortState.column === 'volume24h' ?
sortState.order : 'asc'}
                  onClick={handleSort('volume24h')}
                  sx={{
                    '& .MuiTableSortLabel-icon': {
                      color: (theme) =>
                        sortState.column === 'volume24h' ?
'primary.main' : 'text.secondary',
                    },
                  }}
                >
                  {language === 'en' ? 'Volume 24h' : 'Об'єм 24
год'}
                </TableSortLabel>
              </TableCell>
            </TableRow>
          </TableHead>
          <TableBody>
            {sortedData.map((row, index) => {
              const extendedRow = row as ExtendedScreenerData;
              const exchange = exchanges?.find((e) => e.id ===
extendedRow.exchangeId);
              const symbol = symbols?.find((s) => s.id ===
extendedRow.symbolId);
              return (
                <TableRow
                  key={index}
                  hover
                  sx={{
                    '&:hover': {
                      bgcolor: (theme) =>
                        theme.palette.mode === 'dark' ? 'rgba(255,
255, 255, 0.05)' : 'rgba(0, 0, 0, 0.04)',
                    },
                  }}
                >
```

```
                <TableCell>
                 <IconButton
                  size="small"
                  onClick={(e) => handleMenuClick(e,
extendedRow)}
                  sx={{
                   color: (theme) => theme.palette.mode ===
'dark' ? 'inherit' : 'text.primary',
                   '&:hover': {
                    background: (theme) =>
                     theme.palette.mode === 'dark'
                      ? 'rgba(255, 255, 255, 0.08)'
                      : 'rgba(25, 118, 210, 0.08)',
                    },
                   }}
                 >
                  <MoreVert />
                 </IconButton>
                 <Menu
                  anchorEl={anchorEl}
                  open={Boolean(anchorEl)}
                  onClose={handleMenuClose}
                  sx={{
                   '& .MuiPaper-root': {
                    background: (theme) =>
                     theme.palette.mode === 'dark'
                      ? 'linear-gradient(135deg, rgb(24, 35, 45)
0%, rgba(0, 0, 0, 1) 100%)'
                      : 'linear-gradient(135deg, rgba(255, 255,
255) 0%, rgba(227, 242, 253, 1) 100%)',
                    borderRadius: 4,
                    border: (theme) => `0px solid
${theme.palette.divider}`,
                    boxShadow: '0 4px 10px 0 rgba(0, 0, 0, 0.05)',
                   },
                   '& .MuiMenuItem-root': {
                    gap: 1,
                    color: 'text.primary',
                    '&:hover': {
                     background: (theme) =>
                      theme.palette.mode === 'dark'
                       ? 'rgba(255, 255, 255, 0.08)'
                       : 'rgba(25, 118, 210, 0.08)',
                    },
                    '& .MuiSvgIcon-root': {
                     fontSize: 20,
                     color: (theme) => theme.palette.mode ===
'dark' ? '#60aeed' : '#1976d2',
                    },
                   },
                  }}
                 >
                  <MenuItem onClick={handleCopyInfo}>
                   <ContentCopy />
                   {language === 'en' ? 'Copy Info' : 'Копіювати
інформацію'}
                  </MenuItem>
                  <MenuItem
                   component={Link}

to={`/details/${selectedRow?.exchangeId}/${selectedRow?.sy
mbolId}`}
                   onClick={handleMenuClose}
                  >
                   <Info />
                   {language === 'en' ? 'Details' : 'Деталі'}
                  </MenuItem>
                  <MenuItem onClick={handleAddToWatchlist}>
                   <Star />
                   {language === 'en' ? 'Add to Watchlist' :
'Додати до списку спостереження'}
                  </MenuItem>
                 </Menu>
                </TableCell>
                <TableCell>
                 <Box sx={{ display: 'flex', alignItems: 'center',
gap: 1 }}>
                  <Box
                   dangerouslySetInnerHTML={{ __html:
exchange?.iconSvg || '' }}
                   sx={{ width: 24, height: 24 }}
                  />
                  <Typography variant="body1"
color="text.primary">
                   {row.exchange}
                  </Typography>
                 </Box>
                </TableCell>
                <TableCell>
                 <Box sx={{ display: 'flex', alignItems: 'center',
gap: 1 }}>
                  <Box
                   dangerouslySetInnerHTML={{ __html:
symbol?.iconSvg || '' }}
                   sx={{ width: 24, height: 24 }}
                  />
                  <Typography variant="body1"
color="text.primary">
                   {row.pair}
                  </Typography>
                 </Box>
                </TableCell>
                <TableCell
align="right">{formatNumber(row.price, 2)}</TableCell>
                <TableCell
align="right">{formatNumber(row.volume1h, 0)}</TableCell>
                <TableCell
align="right">{formatNumber(row.volume24h,
0)}</TableCell>
               </TableRow>
              );
             })}
            </TableBody>
           </Table>
          </TableContainer>
         </Paper>
         <FilterModal open={openFilterModal} onClose={() =>
setOpenFilterModal(false)} />
         <Snackbar
          open={openSnackbar}
          autoHideDuration={3000}
          onClose={handleSnackbarClose}
          anchorOrigin={{ vertical: 'bottom', horizontal: 'center' }}
         >
          <Alert
           onClose={handleSnackbarClose}
           severity={snackbarSeverity}
           sx={{
            width: '100%',
            bgcolor: (theme) =>
             snackbarSeverity === 'success' ?
theme.palette.success.main : theme.palette.error.main,
            color: (theme) =>
             snackbarSeverity === 'success' ?
theme.palette.success.contrastText :
theme.palette.error.contrastText,
           }}
          >
```