ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка web-сервісу для перегляду статистики криптовалют з криптобірж»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Вадим МАЛІКОВ
(підпис)

Виконав: здобувач вищої освіти групи ПД-42

Вадим МАЛІКОВ

Керівник: Віталій ЗАЛИВА
доктор філософії (PhD)

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Малікову Вадиму Євгеновичу

1. Тема кваліфікаційної роботи: «Розробка web-сервісу для перегляду статистики криптовалют з криптобірж»
керівник кваліфікаційної роботи доктор філософії (PhD) Віталій ЗАЛИВА, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи розробки веб-застосунків, опис роботи веб-додатків, документація технологій React, Node.js.

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

1. Огляд веб-сервісу для перегляду статистики криптовалют з криптобірж та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2. Огляд та аналіз засобів реалізації веб-сервісу для перегляду статистики криптовалют з криптобірж.
3. Проєктування архітектури веб-сервісу для перегляду статистики криптовалют з криптобірж.

4. Програмна реалізація та тестування веб-сервісу для перегляду статистики криптовалют з криптобірж.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Flow-Діаграма фронтенд частини.
6. FLOW-Діаграма Restful Api сервісу.
7. Екранні форми.
8. Апробація розробленого рішення.

6. Дата видачі завдання «24» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2025	
2	Аналіз та дослідження існуючих аналогів	05.03-13.03.2025	
3	Огляд та аналіз засобів реалізації веб-сервісу для перегляду статистики криптовалют з криптобірж	14.03-16.03.2025	
4	Проєктування архітектури веб-сервісу для перегляду статистики криптовалют з криптобірж	17.03-24.03.2025	
5	Програмна реалізація веб-сервісу для перегляду статистики криптовалют з криптобірж	25.03-26.04.2025	
6	Тестування застосунку веб-сервісу для перегляду статистики криптовалют з криптобірж	27.04-29.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	30.04-11.05.2025	
8	Розробка демонстраційних матеріалів	12.05-16.05.2025	
9	Попередній захист роботи	17.05-30.05.2025	

Здобувач вищої освіти _____ Вадим МАЛІКОВ
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ЗАЛИВА
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 78 стор., 2 табл., 26 рис., 20 джерел.

Мета роботи – спрощення процесу аналізу криптовалют за рахунок використання веб-сервісу.

Об'єкт дослідження – процес моніторингу та аналізу криптовалютної статистики та відображення динаміки ринку.

Предмет дослідження – програмне забезпечення для автоматизованого збору, обробки та відображення статистики криптовалют у реальному часі.

Короткий зміст роботи: В роботі проаналізовано методики збору та обробки статистики криптовалют із криптобірж у реальному часі. Проаналізовано інструментальні засоби для моніторингу криптовалют: CoinMarketCap, CoinGecko, TradingView. Розроблено архітектуру веб-сервісу Crypto Screener та програмно реалізовано ключові функціональні можливості, зокрема: збір даних із бірж через API, порівняння цін, фільтрація та сортування торгових пар, відображення графіків, аутентифікація, створення списків спостереження. Проведено функціональне та модульне тестування додатку. В роботі використано бібліотеку CCXT для доступу до API бірж, MongoDB для зберігання даних, React із Material UI для інтерфейсу, Recharts для візуалізації даних.

Сферою використання додатку є підтримка трейдерів та інвесторів у прийнятті торгових рішень на криптовалютному ринку.

КЛЮЧОВІ СЛОВА: КРИПТОВАЛЮТИ, СТАТИСТИКА, REALTIME, DDD, FSD, TYPESCRIPT, MONGODB, REACT, CCXT, REST API

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення

IDE – Integrated Development Environment SRS – Spaced Repetition System

API – Application Programming Interface

SPA – Single Page Application

DDD – Domain-Driven Design

FSD – Feature-Sliced Design

REST – Representational State Transfer

Crypto Screener – Web service for real-time cryptocurrency market data analysis

Crypto Price Parser Backend (API #1) – Backend service for periodic data collection from cryptocurrency exchanges

RESTful API (API #2) – API service for providing market data to the frontend

CCXT – CryptoCurrency eXchange Trading Library

MongoDB – Document-oriented database

Mongoose – Object-Document Mapping library for MongoDB

Node.js – Asynchronous JavaScript runtime environment

TypeScript – Statically typed superset of JavaScript

React – JavaScript library for building user interfaces

Material UI – Component library for React

Zustand – State management library for React

TanStack Query – Library for managing API requests

Recharts – Data visualization library for React

React Router DOM – Routing library for React

Vite – Build tool for web applications

Jest – JavaScript/TypeScript testing framework

Axios – Library for HTTP requests

ESLint – Static code analysis tool

Prettier – Code formatting tool

GitHub – Platform for version control and collaborative development

CORS – Cross-Origin Resource Sharing, browser security mechanism

HTTP – HyperText Transfer Protocol

DI – Dependency Injection

JSON – JavaScript Object Notation

ВСТУП

Актуальність: моніторинг статистики криптовалют із криптобірж є важливим аспектом сучасного фінансового ринку, що сприяє прийняттю обґрунтованих торгових рішень трейдерами, інвесторами, аналітиками, а також новачками, які тільки входять у криптовалютний ринок і потребують простих та інтуїтивних інструментів для аналізу. Зі зростанням популярності криптовалют, обсяг торгів яких, за даними CoinMarketCap, у 2025 році перевищує 2 трильйони доларів щомісяця, зростає потреба в інструментах для швидкого доступу до актуальних даних у реальному часі.

Сучасні технології дозволяють створювати веб-сервіси, які забезпечують зручний аналіз цін, об'ємів торгів і ринкових показників. Веб-сервіс є оптимальним рішенням для організації такого моніторингу, надаючи користувачам можливість фільтрувати, сортувати та порівнювати дані з різних бірж у зручному форматі. Веб-сервіс Crypto Screener поєднує інструменти для збору, обробки та відображення статистики криптовалют, забезпечуючи гнучкість, адаптивність і персоналізацію для користувачів, зокрема початківців.

Об'єктом дослідження є процес моніторингу та аналізу криптовалютної статистики та відображення динаміки ринку.

Предметом дослідження є програмне забезпечення для автоматизованого збору, обробки та відображення статистики криптовалют у реальному часі.

Метою роботи є спрощення процесу аналізу криптовалют за рахунок використання web-сервісу.

Методи дослідження: першим кроком було проаналізовано існуючі рішення для моніторингу криптовалют (CoinMarketCap, CoinGecko, TradingView) для формування функціональних і нефункціональних вимог до веб-сервісу. Проведено огляд технологічного стеку, обрано TypeScript для бекенду та фронтенду, MongoDB із Mongoose ODM як NoSQL-базу даних, CCXT для доступу до API бірж, React із Material UI для клієнтської частини, Zustand і TanStack Query для управління станом

і запитами, Recharts для візуалізації, Vite для збірки і GitHub для контролю версій. Дослідження реалізації проводилося під час розробки бекенду, REST API та фронтенду.

Наукова новизна роботи визначається наступними функціональними складовими: автоматичний збір даних із бірж у реальному часі через CCXT із частотою оновлення кожну хвилину; функція порівняння цін однієї торгової пари між біржами; інтерактивний скрінер із гнучкими фільтрами та мультимовним інтерфейсом (англійська, українська). Веб-сервіс Crypto Screener об’єднує ключові інструменти для аналізу криптовалют в одному місці.

Практична значущість результатів полягає в можливості використання реалізованого веб-сервісу для ефективного моніторингу статистики криптовалют на різних пристроях, що підтримує трейдерів та інвесторів у прийнятті торгових рішень.

1 АНАЛІЗ ЗАСТОСУНКУ ДЛЯ ПЕРЕГЛЯДУ СТАТИСТИКИ КРИПТОВАЛЮТ З КРИПТОБІРЖ

1.1 Веб-сервіс для перегляду статистики криптовалют із криптобірж

Веб-сервіс для перегляду статистики криптовалют із криптобірж — це спеціалізований онлайн-ресурс, який надає доступ до актуальних даних про ціни, об’єми торгів і ринкові показники з різних криптовалютних бірж, таких як Binance, Coinbase і Kraken. Такі веб-сервіси допомагають користувачам, зокрема новачкам, трейдерам та інвесторам, аналізувати ринкові дані через інтерактивні таблиці, графіки, фільтри та інструменти порівняння. Основна мета веб-сервісу полягає у забезпеченні ефективного та доступного способу моніторингу криптовалют у реальному часі, що дозволяє користувачам приймати обґрунтовані торгові рішення незалежно від їхнього досвіду чи місцезнаходження.

Основні компоненти веб-сервісу для перегляду статистики криптовалют включають:

- Інтерактивний скрінер: таблиці з даними про торгові пари, що дозволяють фільтрувати за ціною, об’ємом торгів чи біржами та сортувати за різними параметрами;
- Конвертація: порівняння ціни за певний обсяг однієї монети з будь-якою іншою монетою на обраній біржі;
- Графіки та візуалізація: відображення зміни ціни;
- Порівняння цін: інструменти для порівняння цін однієї торгової пари на різних біржах;
- Аутентифікація та профіль: можливості реєстрації, входу та доступу до профілю;
- Персоналізація: списки спостереження (watchlist) для відстеження обраних торгових пар і налаштування інтерфейсу (тема, мова).

Основні цілі вебсайту веб-сервісу для перегляду статистики криптовалют включають:

- Забезпечення доступу до актуальних даних у реальному часі;
- Спрощення аналізу ринкових тенденцій;
- Підтримка порівняння цін між біржами;
- Персоналізація досвіду користувача;
- Підвищення зручності для новачків через інтуїтивний інтерфейс;
- Забезпечення адаптивності для різних пристроїв;
- Оцінка ринкових можливостей для трейдерів та інвесторів.

Оцінимо переваги та недоліки веб-сервісу як засобу для перегляду статистики криптовалют.

Переваги:

- Доступність: Веб-сервіси дозволяють моніторити дані з будь-якої точки світу за наявності інтернету, що ідеально для користувачів із різних регіонів;
- Гнучкість: Користувачі можуть аналізувати дані у зручний час, фільтруючи та сортуючи їх за потребами;
- Різноманіття даних: Доступ до цін, об’ємів торгів, ринкових показників із кількох бірж в одному місці;
- Інтуїтивність для новачків: Прості інтерфейси та фільтри спрощують аналіз для початківців;
- Економія часу: Автоматизований збір даних у реальному часі зменшує потребу в ручному пошуку інформації.

Недоліки:

- Технічні проблеми: Потреба в стабільному інтернет-з’єднанні та сучасному обладнанні може обмежувати доступ для деяких користувачів;
- Обмеження API бірж: Безкоштовні API бірж мають ліміти запитів, що може впливати на частоту оновлення даних;

- Перевантаженість інформацією: Великий обсяг даних може бути складним для новачків без належного інтерфейсу;
- Залежність від бірж: Точність даних залежить від якості API бірж, які можуть мати затримки чи помилки.

1.2 Специфіка роботи за криптовалютними даними

Збір і обробка статистики криптовалют із криптобірж є складним процесом, що відрізняється від аналізу традиційних фінансових ринків через високу волатильність, різноманітність даних і технічні обмеження. Криптовалютний ринок працює цілодобово, а ціни на торгові пари (наприклад, BTC/USDT, ETH/USD) можуть змінюватися щохвилини, що вимагає оновлення даних у реальному часі з частотою до 1 хвилини. API бірж, таких як Binance, Coinbase і Kraken, надають доступ до цін, об’ємів торгів (за 1h, 24h, 7d, 30d) і ринкових показників (зміна за 24h, діапазон цін), але мають ліміти запитів (наприклад, 1200 на хвилину для Binance) і можливі затримки. Дані з різних бірж мають неоднорідну структуру, що ускладнює їхню уніфікацію. Великий обсяг інформації — тисячі торгових пар на кожній біржі — потребує ефективного зберігання та швидкого доступу, для чого підходять NoSQL-база даних, такі як MongoDB.

Технічні виклики включають високе навантаження на сервер при асинхронному зборі даних і ризик збоїв API. Користувачі, особливо новачки, потребують інтуїтивного інтерфейсу з простими фільтрами (за ціною, об’ємом, біржами), мультимовністю (англійська, українська) та адаптивністю для мобільних і десктопних пристроїв. Порівняння цін між біржами є ключовою потребою для виявлення арбітражних можливостей, що додає складності до обробки даних.

Функціональні особливості веб-сервиса Crypto Screener зосереджені на наданні користувачам актуальної інформації про криптовалютний ринок через:

- Відображення даних про ціни та обсяги торгів із різних бірж, таких як Binance, Coinbase і Kraken;
- Фільтрацію та сортування торгових пар за параметрами, зокрема ціною, обсягом торгів і зміною за 24 години;
- Порівняння цін на різних біржах для виявлення арбітражних можливостей;
- створення персоналізованих списків спостереження для відстеження вибраних активів.

Система працює з чітко структурованими ринковими даними, отриманими через API бірж і CoinGecko, обробляючи їх у форматі JSON для забезпечення швидкого доступу та точності. Це дозволяє сервісу ефективно надавати актуальну інформацію без затримок, уникаючи обробки неструктурованих чи неоднозначних даних.

Технічні можливості застосунку включають:

- Періодичний збір стандартизованих даних із бірж через Crypto Price Parser (API #1) із використанням бібліотеки CCXT;
- Обробку запитів до RESTful API (API #2) через ендпоінти, такі як /crypto-prices/latest, /market-data і /watch-list, із підтримкою аутентифікації;
- Відображення інтерактивного інтерфейсу SPA, створеного на основі React 19, із компонентами Material UI 7 і графіками Recharts підтримку англійської та української мов із перемиканням через Zustand і збереженням налаштувань у локальному сховищі.

Поточна версія веб-сервісу Crypto Screener ефективно виконує свою основну функцію — надання трейдерам та інвесторам зручного інструменту для моніторингу криптовалютного ринку.

Завдяки модульній архітектурі DDD для серверної частини та FSD для клієнтської, а також використанню TypeScript, Node.js, MongoDB і Vite, сервіс вирізняється простотою, надійністю та масштабованістю. Обмеження, пов’язані з

фокусом на стандартизованих даних, компенсуються високою продуктивністю, адаптивним дизайном і прозорою роботою системи.

1.3 Цільова аудиторія

Веб-сервіс Crypto Screener розроблено для задоволення потреб різноманітних груп користувачів, які взаємодіють із криптовалютним ринком, прагнучи ефективних інструментів для аналізу ринкових даних у реальному часі. Основна аудиторія охоплює як новачків, які активно входять у криптоіндустрію через її високу актуальність і швидке зростання популярності, так і досвідчених учасників ринку, що потребують надійних засобів для моніторингу та прийняття рішень. Сервіс забезпечує інтуїтивний інтерфейс, швидку обробку даних і розширені можливості візуалізації, що робить його доступним і зручним для користувачів із різним рівнем підготовки.

Найважливішою групою є новачки у криптоіндустрії, переважно віком від 18 до 30 років, які приєднуються до ринку завдяки його популярності та потенціалу швидкого зростання. Ці користувачі часто мають обмежений досвід у торгівлі чи інвестиціях, але активно вивчають криптовалюти через соціальні мережі, форуми та освітні платформи. Вони проводять значний час онлайн, досліджуючи ринок, і потребують простого та зрозумілого інструменту для ознайомлення з цінами, трендами та основними активами. Новачки особливо цінують адаптивний дизайн, реалізований через Material UI 7, і мультимовність (англійська та українська), що полегшує взаємодію з сервісом на будь-якому пристрої. Компонент CryptoTable із базовими фільтрами та інтерактивні графіки в InfoSection, створені за допомогою Recharts, дозволяють їм швидко освоювати ринкову інформацію без технічних складнощів.

Активні трейдери криптовалют, зазвичай віком від 20 до 35 років, становлять ще одну ключову групу. Вони регулярно здійснюють торгові операції на біржах, таких як Binance, Coinbase і Kraken, витрачаючи від 4 до 8 годин щодня на моніторинг ринку. Ці користувачі потребують миттєвого доступу до точних даних

про ціни, обсяги торгів і ринкову статистику для використання короткострокових цінових коливань. Висока чутливість до затримок робить для них критично важливими швидке оновлення даних через Crypto Price Parser Backend (API #1) і гнучкі можливості фільтрації в компоненті FilterModal, керованому Zustand. Функція порівняння цін між біржами в ExchangeComparison підтримує оперативне прийняття торгових рішень.

Довгострокові інвестори, віком від 25 до 50 років, зосереджені на стратегічному управлінні портфелем, також є важливою частиною аудиторії. Вони прагнуть аналізувати цінові тренди протягом тривалих періодів і порівнювати ринкові дані на різних платформах. Ці користувачі мають середній або високий рівень фінансової грамотності та потребують детальних статистичних даних із простою візуалізацією. Компонент InfoSection із історичними даними та графіками, а також можливість створення персоналізованих списків спостереження через WatchList, відповідають їхнім потребам у моніторингу активів і оцінці інвестиційних можливостей.

Аналітики криптовалютного ринку, включаючи фахівців фінансових установ і незалежних дослідників, віком від 30 до 55 років, доповнюють цільову аудиторію. Вони використовують ринкові дані для створення звітів, прогнозів і стратегічних рекомендацій, потребуючи інструментів для обробки великих обсягів інформації. Ендпоінти RESTful API (API #2), такі як /crypto-prices і /market-data, надають структуровані JSON-відповіді, а візуалізації через Recharts дозволяють аналітикам ефективно здобувати інсайти. Можливість експорту даних підтримує їхні професійні задачі.

Потребам усіх груп відповідають спеціально розроблені функції сервісу. Новачки отримують доступ до простого інтерфейсу з мультимовною підтримкою та компонентом SettingsPanel, що полегшує персоналізацію. Трейдери використовують швидке оновлення даних і фільтрацію в CryptoTable для оперативного аналізу. Інвестори застосовують InfoSection і ExchangeComparison для стратегічного планування. Аналітики отримують структуровані дані через API для поглибленої обробки. Завдяки архітектурі DDD для серверної частини та FSD

для клієнтської, а також технологіям TypeScript, Node.js і MongoDB, Crypto Screener забезпечує універсальний інструмент, що відповідає різноманітним потребам учасників криптовалютного ринку.

1.4 Дослідження існуючих аналогів

1.4.1 CoinMarketCap

CoinMarketCap — це провідна платформа для моніторингу криптовалютного ринку, яка агрегує дані про ціни, ринкову капіталізацію, обсяги торгів і рейтинги активів із сотень бірж, таких як Binance, Coinbase і Kraken. Вона надає користувачам зручний вебінтерфейс і API для доступу до ринкової інформації, що робить її популярним інструментом серед трейдерів, інвесторів, аналітиків і новачків. Платформа зосереджена на наданні комплексної статистики, включаючи історичні дані та ринкові тренди, але має обмеження, які роблять її менш ефективною порівняно з веб-сервісом Crypto Screener у певних сценаріях, зокрема для реального часу та персоналізованих функцій.

Основні функції CoinMarketCap:

- Агрегація ринкових даних: забезпечує відображення цін, ринкової капіталізації та обсягів торгів для тисяч криптовалют із різних бірж;
- Історичні графіки: дозволяє аналізувати цінові тренди за різні періоди, від 24 годин до кількох років;
- Рейтинги активів і бірж: формує списки популярних криптовалют і платформ на основі їхньої ринкової активності;
- API-доступ: надає JSON-відповіді для інтеграції даних у зовнішні системи чи додатки;

- Огляд новин і подій: інформує про актуальні події на крипторинку, пов’язані з активами чи біржами.

Ключові переваги CoinMarketCap:

- Широке охоплення: агрегує дані з численних бірж, надаючи всебічну картину ринку;
- Багатомовна підтримка: доступна кількома мовами, включаючи англійську, що полегшує використання міжнародною аудиторією;
- Регулярні оновлення: додає підтримку нових криптовалют і бірж, забезпечуючи актуальність інформації;
- API для розробників.

Обмеження сервісу:

- Затримки оновлення даних: інформація може оновлюватися із затримкою до кількох хвилин, що критично для торгівлі в реальному часі;
- Обмеження безкоштовного API: до 10 000 запитів на місяць, що недостатньо для активних трейдерів;
- Відсутність порівняння цін: не дозволяє аналізувати ціни однієї торгової пари на різних біржах одночасно;
- Складність для новачків: інтерфейс перевантажений даними, що вимагає розуміння криптовалютної термінології;
- Відсутність персоналізації: не підтримує списки спостереження.

Технічні особливості:

- Затримки оновлення даних: інформація може оновлюватися із затримкою до кількох хвилин, що критично для торгівлі в реальному часі;
- Обмеження безкоштовного API: до 10 000 запитів на місяць, що недостатньо для активних трейдерів;
- Доступний через веббраузер і мобільні додатки для Android та iOS;
- Використовує централізовані сервери для обробки даних, що забезпечує стабільність;

– Підтримує REST API із JSON-відповідями для інтеграції з іншими системами;
– Не потребує локального зберігання даних, обробка виконується на серверній стороні.

CoinMarketCap є потужним інструментом для моніторингу криптовалютного ринку, але її обмеження, такі як затримки даних, відсутність порівняння цін і складність для новачків, роблять її менш ефективною порівняно з Crypto Screener. Наш сервіс, завдяки Crypto Price Parser Backend (API #1) для швидкого збору даних, RESTful API (API #2) із підтримкою ендпоінтів, таких як /crypto-prices/latest, і адаптивному SPA-інтерфейсу на основі React 19, Material UI 7 і Recharts, пропонує оперативний і персоналізований підхід. Функції порівняння цін у ExchangeComparison, мультимовність (англійська та українська), списки спостереження через WatchList і щохвилинне оновлення даних забезпечують перевагу для новачків, трейдерів, інвесторів і аналітиків.

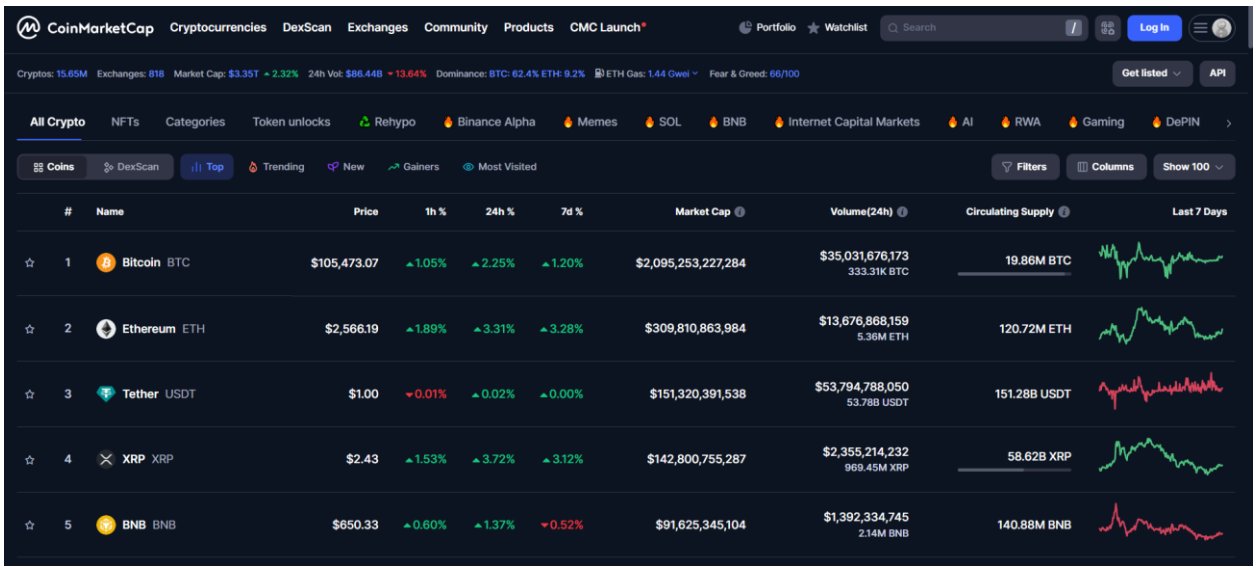


Рис. 1.1 Приклади екранних форм web-сервісу CoinMarketCap

1.4.2 CoinGecko

CoinGecko — це одна з найпопулярніших платформ для моніторингу криптовалютного ринку, яка агрегує дані з численних бірж, таких як Binance,

Coinbase і Kraken, і надає користувачам детальну інформацію про криптовалюти, включаючи ціни, ринкову капіталізацію, обсяги торгів і історичні показники. Платформа вирізняється зручним вебінтерфейсом і API, що робить її привабливим вибором для трейдерів, інвесторів, аналітиків і новачків, які прагнуть досліджувати ринок. CoinGecko фокусується на наданні розширеної статистики та аналітичних даних, але її обмеження в реальному часі та функціональності роблять її менш конкурентною порівняно з веб-сервісом Crypto Screener, який пропонує швидше оновлення даних і персоналізовані функції.

Основні функції CoinGecko:

- Агрегація ринкових даних: відображає ціни, ринкову капіталізацію та обсяги торгів для тисяч криптовалют із різних бірж;
- Історичні дані: надає доступ до цінових трендів і статистики за різні періоди, від 24 годин до кількох років;
- Аналітичні звіти: включає метрики, такі як ринкова домінантність і ліквідність активів;
- API-доступ: забезпечує JSON-відповіді для інтеграції ринкових даних у зовнішні додатки;
- Інформація про проєкти: пропонує описи криптовалют, включаючи їхню технологію та команду розробників.

Ключові переваги CoinGecko:

- Широке охоплення бірж: агрегує дані з багатьох платформ, надаючи повну картину ринку;
- Детальна аналітика: включає розширені метрики, корисні для інвесторів і аналітиків;
- Багатомовна підтримка: доступна кількома мовами, включаючи англійську, що полегшує використання міжнародною аудиторією;
- Безкоштовний API: дозволяє отримувати дані без значних обмежень для базового використання.

Обмеження CoinGecko:

- Затримки оновлення даних: інформація оновлюється кожні кілька хвилин, що не відповідає вимогам трейдерів до реального часу;
- Обмежена функціональність фільтрації: не підтримує гнучкі параметри, такі як фільтрація за діапазонами цін чи біржами;
- Відсутність інтерактивних графіків: графіки не дозволяють глибокого налаштування чи інтерактивних підказок;
- Немає порівняння цін між біржами: ускладнює виявлення арбітражних можливостей;
- Обмежена персоналізація: не пропонує списків спостереження чи адаптивного дизайну для мобільних пристроїв.

Технічні особливості:

- Доступний для Android і iOS;
- Доступний через веббраузер і мобільні додатки для Android та iOS;
- Використовує централізовані сервери для обробки даних, що забезпечує стабільність;
- Підтримує REST API із JSON-відповідями для інтеграції з іншими системами.

CoinGecko є ефективним інструментом для аналізу криптовалютного ринку, але її обмеження, такі як затримки оновлення даних, слабка функціональність фільтрації та відсутність інтерактивних графіків, роблять її менш зручною порівняно з Crypto Screener. Наш сервіс, завдяки Crypto Price Parser Backend (API #1) для щохвилинного збору даних, RESTful API (API #2) із ендпоінтами, такими як /crypto-prices/latest і /market-data, та адаптивному SPA-інтерфейсу на основі React 19, Material UI 7 і Recharts, пропонує швидший і більш персоналізований підхід. Функції порівняння цін у ExchangeComparison, гнучка фільтрація через FilterModal, мультимовність (англійська та українська) і списки спостереження через WatchList забезпечують перевагу для новачків, трейдерів, інвесторів і аналітиків.

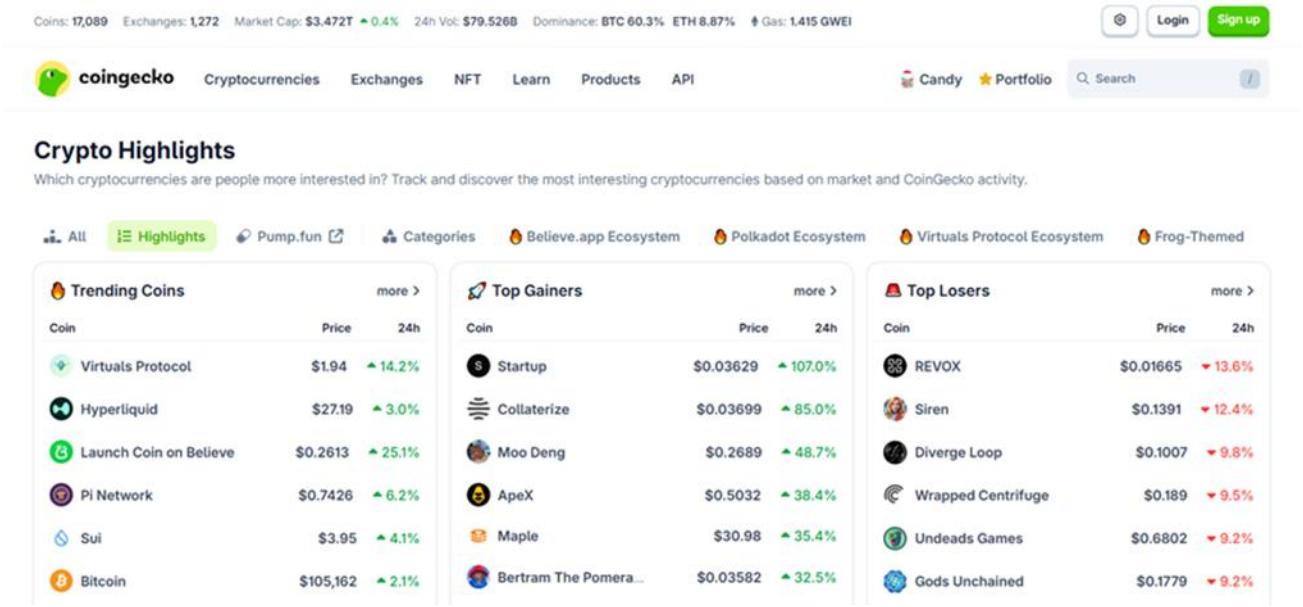


Рис. 1.2 Інтерфейс застосунку CoinGecko

1.4.3 TradingView

TradingView — це популярна платформа для технічного аналізу фінансових ринків, включаючи криптовалюти, яка вирізняється потужними інструментами для побудови інтерактивних графіків і аналітики. Вона орієнтована переважно на трейдерів і аналітиків, які потребують глибокого аналізу ринкових трендів, але її можливості для агрегації даних із різних криптовалютних бірж є обмеженими порівняно з веб-сервісом Crypto Screener. TradingView пропонує зручний вебінтерфейс і API, однак зосереджується більше на візуалізації графіків, ніж на комплексному моніторингу ринкових даних, що робить її менш універсальною для новачків, інвесторів і тих, хто шукає швидкий доступ до актуальних цін.

Основні функції TradingView:

– Інтерактивні графіки: дозволяє створювати налаштовувані графіки цін із підтримкою технічних індикаторів, таких як RSI, MACD;
– Технічний аналіз: надає інструменти для аналізу ринкових трендів, включаючи підтримку торгових стратегій і прогнозів;

- Соціальна мережа трейдерів: забезпечує платформу для обміну ідеями, стратегіями та аналітикою між користувачами;
- API для графіків: дозволяє інтегрувати графіки TradingView у зовнішні вебсайти чи додатки;
- Дані про криптовалюти: відображає ціни, обсяги торгів і ринкову статистику для популярних активів із окремих бірж.

Обмеження TradingView:

- Обмежена агрегація даних: не забезпечує комплексного збору даних із багатьох бірж, фокусуючись на окремих платформах;
- Затримки для безкоштовних користувачів: оновлення даних може відбуватися із затримкою до кількох хвилин, що неприйнятно для реального часу;
- Відсутність порівняння цін між біржами: не дозволяє аналізувати ціни однієї торгової пари на різних платформах;
- Складність для новачків: інтерфейс із великою кількістю аналітичних інструментів може бути незрозумілим для початківців;
- Обмежений API: зосереджений на інтеграції графіків, а не на наданні ринкових даних, що знижує універсальність.

Ключові переваги TradingView:

- Потужні аналітичні інструменти: підтримує широкий набір індикаторів і функцій для технічного аналізу, що ідеально для досвідчених трейдерів;
- Інтерактивний інтерфейс: графіки з високим рівнем налаштування та підтримкою інтерактивних елементів;
- Спільнота користувачів: дозволяє обмінюватися ідеями та стратегіями, що корисно для трейдерів і аналітиків;
- Мультиплатформність: доступна через веббраузер, мобільні додатки для Android і iOS, а також десктопну версію;
- Підтримка кількох мов: включає англійську, що забезпечує доступність для міжнародної аудиторії.

Технічні особливості:

- Доступний через веббраузер, мобільні додатки для Android і iOS, а також десктопну програму;
- Використовує централізовані сервери для обробки даних, що забезпечує стабільність роботи;
- Не потребує локального зберігання даних, оскільки вся обробка виконується на серверній стороні.

TradingView вирізняється серед аналогів завдяки своїм потужним інструментам для технічного аналізу та інтерактивним графікам. Ця платформа орієнтована на трейдерів, які потребують детальної аналітики, але її можливості для агрегації даних із різних бірж обмежені. TradingView не підтримує функцію порівняння цін між біржами, а оновлення даних для безкоштовних користувачів часто відбувається із затримками. API платформи більше зосереджено на інтеграції графіків, ніж на наданні ринкових даних, що робить її менш універсальною для моніторингу криптовалют.

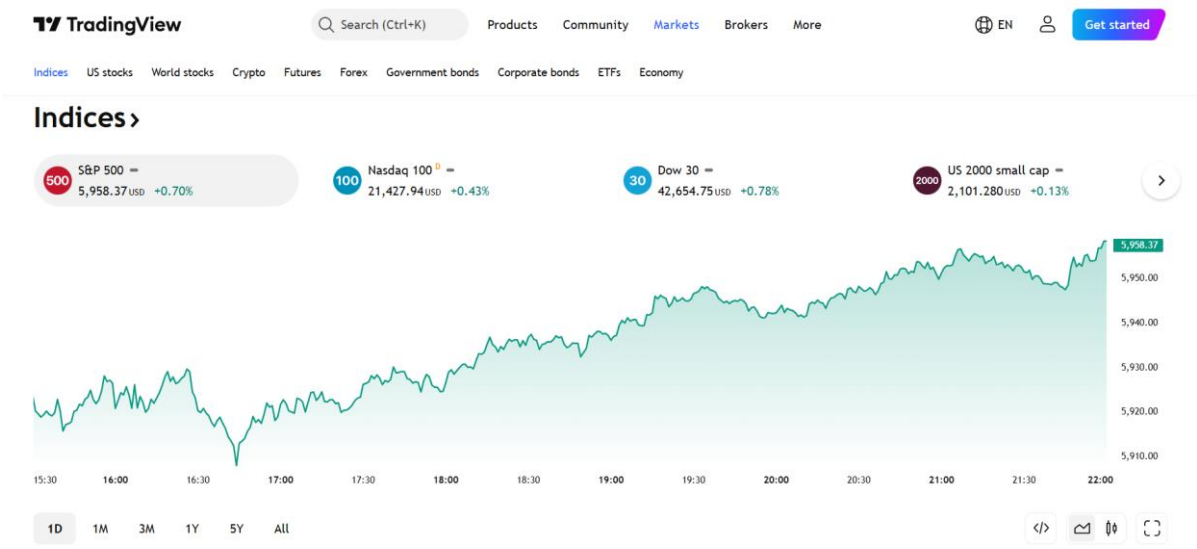


Рис. 1.3 Інтерфейс TradingView

В таблиці 1.1 зведено результати аналізу існуючих застосунків для запобігання кібербулінгу.

Таблиця 1.1

Зведена таблиця аналізу існуючих застосунків для перегляду статистики криптовалют із криптобірж

Показник	CoinMarketCap	CoinGecko	TradingView
Тип застосунку	Інформаційно-аналітична платформа	Інформаційно-аналітична платформа	Платформа технічного аналізу
Цільова аудиторія	Трейдери, інвестори, аналітики	Трейдери, інвестори, новачки	Досвідчені трейдери, аналітики
Технологія аналізу	Агрегація даних із бірж	Агрегація даних із бірж	Технічний аналіз із індикаторами

Продовження таблиці 1.1

Зведена таблиця аналізу існуючих застосунків для перегляду статистики криптовалют із криптобірж

Дані, що обробляються	Ціни, капіталізація, обсяги торгів, рейтинги	Ціни, капіталізація, обсяги торгів, описи	Ціни, обсяги торгів, технічні індикатори
Актуалізація в реальному часі	Обмежена (затримки до кількох хвилин)	Обмежена (затримки до кількох хвилин)	Обмежена для безкоштовних користувачів
Інтеграція з платформами	Біржі (Binance, Coinbase тощо), API	Біржі (Binance, Coinbase тощо), API	Обмежена (окремі біржі), API для графіків
Фільтри та аналітика	Базові фільтри (за ціною, капіталізацією)	Обмежені фільтри (за капіталізацією)	Технічні індикатори, без гнучких фільтрів
Мови інтерфейсу	Англійська, кілька інших	Англійська, кілька інших	Англійська, кілька інших

Розроблюваний веб-сервіс Crypto Screener має низку переваг порівняно з аналогами. Він використовує бібліотеку CCXT для уніфікованого доступу до API криптовалютних бірж, таких як Binance, Coinbase і Kraken, що дозволяє отримувати дані про ціни та об’єми торгів у реальному часі з оновленням кожну хвилину. На відміну від CoinMarketCap і CoinGecko, CryptoScreener пропонує функцію порівняння цін однієї торгової пари на різних біржах, що є ключовою особливістю для трейдерів. Фронтенд, побудований на React із використанням бібліотеки Recharts, забезпечує інтерактивні графіки, а розширені фільтри за ціною, об’ємом торгів і біржами, реалізовані через Zustand, перевершують можливості CoinGecko і TradingView. Власний REST API системи не має обмежень на кількість запитів для користувачів, що робить її більш гнучкою порівняно з аналогами.

Порівняння аналогів показало, що більшість існуючих рішень зосереджені на окремих аспектах моніторингу криптовалют, таких як агрегація даних чи технічний аналіз, але не пропонують комплексного підходу. Crypto Screener, навпаки, поєднує актуалізацію даних у реальному часі, порівняння цін між біржами, інтерактивну візуалізацію та гнучкі фільтри, що робить її ефективним інструментом для трейдерів, інвесторів і аналітиків криптовалютного ринку.

1.5 Визначення вимог до застосунку

Проаналізувавши можливості аналогічних платформ, особливості криптовалютного ринку та потреби цільової аудиторії, було сформульовано такі функціональні вимоги до веб-сервісу Crypto Screener для моніторингу статистики криптовалют:

- Отримання даних про криптовалюти з кількох бірж через API;
- Відображення актуальної інформації з бірж в реальному часі;
- Фільтрація за обраними криптовалютами, біржами, діапазонами цін;
- Фільтрація монет за параметрами: ціна, обсяг торгів, зміна за 24h;
- Побудова графічного відображення зміни ціни в діапазоні 24h;
- Реєстрація та можливість налаштування індивідуального списку відслідковування в профілі.

Нефункціональні вимоги:

- Висока продуктивність — оновлення даних не рідше, ніж кожну хвилину;
- Масштабованість — можливість додавання нових джерел API;
- Надійність — відсутність збоїв при недоступності окремих API;
- Контроль CORS — дозволяти запити до API лише з довірених доменів;
- Локалізація — підтримка декількох мов (укр/англ);
- Адаптивність під мобільні та десктопні пристрої.

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) є важливим інструментом для створення сучасних програмних продуктів, оскільки воно об’єднує в собі широкий спектр функціональних можливостей, необхідних для ефективної розробки. Такі середовища зазвичай включають текстовий редактор із підсвічуванням синтаксису, підтримку автодоповнення коду, інструменти для відлагодження, інтеграцію з системами контролю версій, а також розширення для автоматизації процесів форматування, аналізу якості коду та управління залежностями.

Використання IDE або подібних редакторів коду дозволяє розробникам оптимізувати робочий процес, зменшуючи час на налаштування середовища та забезпечуючи зручний доступ до всіх необхідних інструментів у єдиному інтерфейсі. Для розробки веб-сервісу Crypto Screener, призначеного для моніторингу та аналізу криптовалютної статистики в реальному часі, було обрано редактор коду Visual Studio Code (VS Code) як основне середовище розробки, доповнене інструментами ESLint і Prettier для забезпечення якості та консистентності коду.

2.1.1 Visual Studio Code

Для створення веб-сервісу Crypto Screener, який включає бэкенд для збору даних із криптовалютних бірж, RESTful API та фронтенд для відображення ринкової статистики, було обрано Visual Studio Code — потужний і гнучкий редактор коду, розроблений компанією Microsoft. VS Code є одним із найпопулярніших інструментів серед розробників завдяки своїй універсальності, підтримці широкого спектра мов програмування та розширюваній архітектурі. У контексті даного проекту VS Code ідеально підходить для розробки як бэкенду (на основі TypeScript із використанням Domain-Driven Design), так і фронтенду (на

основі React із Feature-Sliced Design), оскільки забезпечує ефективну роботу з обома частинами системи.

Основною перевагою VS Code є його модульна структура, яка дозволяє налаштовувати середовище відповідно до потреб проекту. Редактор підтримує TypeScript — основну мову програмування проекту, надаючи інтелектуальне автодоповнення, рефакторинг коду та аналіз помилок у реальному часі. Вбудована підтримка системи контролю версій Git дозволяє легко інтегрувати проект із репозиторіями на платформах, таких як GitHub, що забезпечує ефективне керування версіями коду. Для роботи з бэкендом, який використовує бібліотеку CCXT для доступу до API бірж, та RESTful API, які потребують аутентифікації, VS Code забезпечує зручний термінал для виконання команд Node.js і керування залежностями за допомогою менеджера пакетів npm.

Важливою особливістю VS Code є підтримка розширень, які дозволяють адаптувати редактор до специфічних вимог розробки. У рамках проекту Crypto Screener було використано низку розширень, зокрема: TypeScript, JavaScript, React, MongoDB і Node.js, які забезпечують синтаксичне підсвічування та автодоповнення для відповідних технологій. Крім того, розширення, такі як GitLens і Live Server, полегшують інтеграцію з Git і попередній перегляд змін у коді, що значно прискорює процес розробки. Ці можливості дозволили ефективно організувати розробку складного веб-сервісу, що включає бэкенд, API та фронтенд.

2.1.2 Інструменти забезпечення якості коду

Для забезпечення високої якості та консистентності коду в проекті використовувалися інструменти ESLint і Prettier. ESLint — це статичний аналізатор коду, який допомагає виявляти помилки та потенційні проблеми в коді JavaScript і TypeScript, забезпечуючи відповідність кодовим стандартам. У проекті Crypto Screener ESLint використовувався для аналізу бэкенду (TypeScript) і фронтенду (React), що дозволило підтримувати єдиний стиль кодування та уникати типових помилок. Налаштування ESLint включали правила для забезпечення відповідності

стандартам Airbnb, що є визнаним лідером у сфері JavaScript-розробки, а також специфічні правила для TypeScript і React.

Prettier — це інструмент автоматичного форматування коду, який забезпечує уніфікований стиль шляхом автоматичного виправлення відступів, розривів рядків, лапок та інших елементів форматування. У поєднанні з VS Code Prettier інтегрується через розширення, дозволяючи форматувати код автоматично при збереженні файлів. У проекті ці інструменти використовувалися для забезпечення консистентності коду в усіх частинах системи — від бэкенду (Crypto Price Parser, RESTful API) до фронтенду (Crypto Screener).

2.1.3 Роль середовища у розробці

Вибір VS Code як основного середовища розробки для проекту Crypto Screener був зумовлений його універсальністю, підтримкою сучасних технологій і можливістю налаштування. Завдяки інтеграції з ESLint і Prettier вдалося забезпечити високу якість коду, що є критично важливим для складного веб-сервісу, який включає обробку даних у реальному часі, взаємодію з API бірж і відображення складних ринкових даних. Крім того, VS Code забезпечує швидке виконання команд Node.js, що було необхідно для запуску бэкенду та API, а також підтримує React і TypeScript, що спростило розробку фронтенду.

2.2 Мови програмування

Мова програмування є фундаментальним інструментом у розробці програмного забезпечення, який визначає спосіб формулювання алгоритмів, обробки даних і реалізації бізнес-логіки. Вона являє собою формалізовану систему синтаксичних правил і команд, що дозволяє розробникам створювати складні програмні системи, забезпечуючи ефективне керування інформаційними потоками, взаємодію з зовнішніми сервісами та підтримку масштабованості. У контексті розробки веб-сервісу Crypto Screener, призначеного для збору, обробки та відображення статистики криптовалют у реальному часі, вибір мови

програмування відіграє ключову роль, оскільки від нього залежить продуктивність, надійність і гнучкість системи. Основною вимогою до мови було забезпечення типобезпеки, підтримки сучасних архітектур, а також сумісності з широким спектром бібліотек і фреймворків, необхідних для інтеграції з API криптовалютних бірж, обробки великих обсягів ринкових даних і створення інтерактивного інтерфейсу користувача. З урахуванням цих вимог як єдина мова програмування для всіх компонентів була обрана TypeScript, яка поєднує потужність статичної типізації з гнучкістю JavaScript, забезпечуючи високу якість коду та ефективність розробки складного веб-сервісу.

2.2.1 TypeScript

TypeScript — це надмножина JavaScript, статично типізована мова програмування, яка розширює можливості JavaScript завдяки додаванню строгих типів, інтерфейсів і розширених інструментів для роботи з об’єктно-орієнтованим програмуванням. У проекті Crypto Screener TypeScript використовувався як основна мова для реалізації всіх трьох компонентів: бэкенду (Crypto Price Parser), RESTful API та фронтенду (Crypto Screener). Вибір TypeScript був обумовлений його численними перевагами, зокрема підтримкою типізації, що знижує ймовірність помилок під час розробки, можливістю створення складних архітектур (таких як Domain-Driven Design для бэкенду та Feature-Sliced Design для фронтенду) і високою сумісністю з сучасними веб-фреймворками та бібліотеками.

На бэкенді TypeScript використовувався для реалізації логіки збору даних із криптовалютних бірж (з використанням бібліотеки CCXT), обробки ринкових даних і збереження їх у базі даних MongoDB через Mongoose ODM. Завдяки строгій типізації TypeScript забезпечував надійність роботи з доменними сутностями, такими як Exchange, Symbol, CryptoPrice і MarketData, дозволяючи чітко визначати їх структуру та поведінку. Наприклад, використання інтерфейсів і типів у TypeScript дозволило уникнути помилок при обробці даних із API бірж, що мають різноманітні формати відповідей. Крім того, підтримка асинхронного програмування через async/await спростила реалізацію основного сценарію використання

FetchAndStoreCryptoPricesUseCase, який періодично отримує та зберігає цінові дані.

На фронтенді TypeScript відігравав ключову роль у розробці інтерфейсу користувача з використанням React 19. Завдяки строгій типізації компоненти, такі як CryptoTable, FilterModal і ExchangeComparison, були реалізовані з високим рівнем надійності, що зменшило кількість помилок, пов'язаних із неправильним використанням пропсів або стану. TypeScript також полегшив інтеграцію з бібліотеками, такими як TanStack Query (для управління API-запитами) і Zustand (для управління станом додатку), завдяки чіткому визначенню типів для даних, отриманих із RESTful API, наприклад, цін криптовалют або ринкових даних.

2.2.2 Переваги вибору TypeScript для проекту

Вибір TypeScript як основної мови програмування для всіх компонентів проекту Crypto Screener забезпечив низку переваг:

- Статична типізація зменшила кількість помилок, пов'язаних із некоректною обробкою даних, що особливо важливо для роботи з API бірж, де формати даних можуть варіюватися;
- Інструменти автодоповнення та рефакторингу в TypeScript, підтримуваних Visual Studio Code, прискорили написання коду та його підтримку;
- TypeScript дозволив організувати складну архітектуру (DDD для бэкенду, FSD для фронтенду), що полегшило додавання нових функцій, таких як підтримка додаткових бірж або нових типів ринкових даних;
- TypeScript забезпечив безперебійну інтеграцію з Node.js, React, MongoDB і зовнішніми API (CCXT, CoinGecko), що зробило його ідеальним вибором для веб-сервісу.

Крім того, TypeScript сприяв забезпеченню високої якості коду в проекті Crypto Screener TypeScript використовувався в поєднанні з інструментами ESLint і Prettier, що дозволило підтримувати єдиний стиль кодування та уникати типових помилок. Наприклад, налаштування ESLint включали правила для TypeScript, які

забезпечували відповідність стандартам кодування, таким як використання явних типів для функцій і змінних, а також уникнення неявного any. Це було особливо важливо для бэкенду, де обробка великих обсягів ринкових даних вимагала чіткої структури коду.

2.3 Фреймворки та технології

Фреймворки відіграють ключову роль у розробці програмного забезпечення, надаючи розробникам структуровану основу для створення складних систем. Вони являють собою набір інструментів, бібліотек і шаблонів, які спрощують реалізацію бізнес-логіки, забезпечують стандартизацію коду та сприяють масштабованості проектів. Завдяки фреймворкам розробники можуть зосередитися на функціональних вимогах, мінімізуючи необхідність написання повторюваного коду. У процесі створення веб-сервісу Crypto Screener, призначеного для збору, обробки та відображення статистики криптовалют у реальному часі, вибір фреймворків був зумовлений необхідністю забезпечення високої продуктивності, модульності та сумісності з сучасними технологіями. Для реалізації бэкенду та RESTful API було використано середовище виконання Node.js, тоді як для фронтенду обрано бібліотеку React 19. Ці технології доповнено низкою допоміжних бібліотек, таких як Material UI, Zustand, TanStack Query і Recharts, які розширили функціональність основних фреймворків і забезпечили виконання складних вимог проекту.

2.3.1 Node.js

Для реалізації серверної частини веб-сервісу Crypto Screener, яка включає бэкенд (Crypto Price Parser) і RESTful API, використано середовище виконання Node.js, побудоване на рушії V8 від Google Chrome. Ця технологія забезпечує асинхронну обробку запитів, що дозволяє створювати швидкі та масштабовані серверні додатки. Вибір Node.js був обумовлений його здатністю ефективно обробляти великі обсяги даних, що є критично важливим для збору ринкової

статистики з криптовалютних бірж, таких як Binance, Coinbase і Kraken, через бібліотеку CCXT. Асинхронна модель Node.js дала змогу паралельно обробляти численні запити до API бірж, мінімізуючи затримки та забезпечуючи високу пропускну здатність системи.

У бэкенді Node.js використовувався для реалізації основного сценарію використання, який періодично отримує цінові дані, обсяги торгів і ринкову статистику, обробляє їх і зберігає в базі даних MongoDB через Mongoose ODM. Асинхронне програмування з використанням конструкцій async/await спростило організацію складної логіки, такої як одночасне отримання даних із кількох бірж. Для RESTful API Node.js забезпечував створення ендпоінтів, що надають фронтенду доступ до інформації про біржі, торгові пари, ціни та ринкові дані. Ці ендпоінти, такі як /exchanges, /symbols і /crypto-prices, були реалізовані з урахуванням принципів чистої архітектури та Domain-Driven Design, що забезпечило чітке розділення шарів і полегшило підтримку коду. Крім того, Node.js дозволив реалізувати middleware для обробки помилок і аутентифікації користувачів, що підвищило безпеку системи.

Ефективність Node.js також підкріплюється його інтеграцією з екосистемою npm, яка надала доступ до бібліотек, таких як CCXT для взаємодії з біржами, Mongoose для роботи з MongoDB і UUID для генерації унікальних ідентифікаторів. Поєднання Node.js із TypeScript забезпечило типобезпеку, що дозволило зменшити кількість помилок у коді та полегшило розробку складної серверної логіки.

2.3.2 React 19

Для створення інтерактивного інтерфейсу користувача веб-сервісу Crypto Screener використано бібліотеку React 19, яка базується на компонентному підході та використовує віртуальний DOM для забезпечення високої продуктивності. React був обраний через його гнучкість, широку підтримку спільноти та сумісність із сучасними технологіями, такими як TypeScript і бібліотеки для управління станом і запитами. У проекті React відповідав за реалізацію фронтенду, який включає відображення ринкової статистики, фільтрацію, сортування та візуалізацію даних у

вигляді графіків. Архітектура фронтенду побудована за принципами Feature-Sliced Design, що забезпечило модульність і полегшило розширення функціоналу.

Основні компоненти інтерфейсу, такі як таблиця для відображення цін і торгових пар, модальне вікно для фільтрації, секція порівняння цін на різних біржах і блок із детальною інформацією про криптовалюту, були реалізовані як React-компоненти. Компонентний підхід дозволив організувати код у модулі, що відповідають структурі FSD (app, pages, features, entities, shared), що спростило підтримку та масштабування. Наприклад, компонент таблиці інтегрувався з бібліотекою Recharts для відображення графіків зміни цін, тоді як модальне вікно для фільтрації використовувало стан, керований через Zustand, для динамічної обробки параметрів.

React 19 забезпечив швидке оновлення інтерфейсу завдяки віртуальному DOM, що було особливо важливим для відображення великих обсягів ринкових даних у реальному часі. Інтеграція з бібліотекою TanStack Query дозволила ефективно керувати API-запитами до ендпоінтів, таких як /crypto-prices/latest і /market-data, забезпечуючи автоматичне оновлення даних без перевантаження інтерфейсу. Використання TypeScript із React додало типобезпеку, що зменшило кількість помилок, пов’язаних із неправильним використанням пропсів або стану компонентів.

2.3.3 Допоміжні технології

Функціональність основних фреймворків була розширена за допомогою низки допоміжних бібліотек, які відіграли важливу роль у реалізації специфічних вимог проекту. Бібліотека Material UI 7 використовувалася для створення адаптивного та сучасного інтерфейсу користувача. Вона надала готові стилізовані компоненти для таблиць, модальних вікон, навігаційної панелі та панелі налаштувань, що дозволило швидко розробити зручний і естетичний дизайн. Підтримка темізації в Material UI забезпечила можливість перемикання між темною та світлою темами, що підвищило комфорт користувачів.

Для управління станом додатку використано бібліотеку Zustand, яка забезпечила легковаговий і ефективний підхід до зберігання та обробки стану фільтрів, сортування та налаштувань інтерфейсу, таких як вибір мови чи теми. Zustand виявилася оптимальним вибором для проекту завдяки своїй простоті порівняно з більш складними альтернативами, такими як Redux.

Бібліотека TanStack Query (React Query) використовувалася для керування API-запитами та кешування даних. Вона спростила асинхронне отримання інформації з RESTful API, забезпечуючи автоматичне оновлення даних і зменшуючи обсяг коду для обробки запитів. Це було особливо важливо для відображення актуальних цін і ринкових даних у реальному часі.

Для візуалізації ринкової статистики використано бібліотеку Recharts, яка інтегрувалася з React для створення динамічних графіків. Recharts забезпечила гнучкість і продуктивність при рендерингу графіків, що дозволило користувачам аналізувати ринкові тенденції в зручному форматі.

2.4 Бази даних та інструменти зберігання

Система зберігання даних є невід’ємною складовою будь-якого сучасного веб-сервісу, забезпечуючи надійне збереження, швидкий доступ і ефективну обробку інформації. База даних слугує основою для організації даних, дозволяючи зберігати структуровану інформацію, виконувати запити та забезпечувати цілісність даних у процесі роботи додатку. У контексті веб-сервісу Crypto Screener, який призначений для збору, обробки та відображення статистики криптовалют у реальному часі, вибір бази даних був зумовлений необхідністю роботи з великими обсягами даних, що постійно оновлюються, а також вимогами до гнучкості структури та високої продуктивності. Для реалізації цих завдань обрано документно-орієнтовану базу даних MongoDB, яка використовується в поєднанні з бібліотекою Mongoose ODM для спрощення роботи з даними на бэкенді. Ці інструменти забезпечили ефективне зберігання інформації про біржі, торгові пари, ціни та ринкові дані, що є основою функціональності системи.

2.4.1 MongoDB

Для зберігання даних у веб-сервісі Crypto Screener використано MongoDB — документно-орієнтовану базу даних, яка належить до категорії NoSQL-систем. MongoDB була обрана завдяки її здатності працювати з неструктурованими або слабко структурованими даними, що ідеально відповідає потребам проекту, де інформація з криптовалютних бірж, таких як Binance, Coinbase чи Kraken, може мати різноманітні формати. У проекті MongoDB використовується для зберігання основних доменних сутностей, включаючи біржі (Exchange), торгові пари (Symbol), цінові дані (CryptoPrice) і ринкову статистику (MarketData). Крім того, база даних підтримує зберігання інформації про користувачів (User) і списки спостереження (WatchList), що є частиною функціоналу RESTful API.

Документно-орієнтована модель MongoDB дозволяє зберігати дані у вигляді JSON-подібних документів, що забезпечує гнучкість у роботі з різними типами інформації. Наприклад, сутність CryptoPrice включає поля для ціни, обсягів торгів за різні періоди (1 година, 24 години, 7 днів, 30 днів) і часової мітки, що може динамічно змінюватися залежно від біржі чи торгової пари. Така структура даних спрощує додавання нових атрибутів без необхідності змінювати схему бази, що є значною перевагою для масштабування системи. MongoDB також забезпечує високу продуктивність при виконанні запитів, що критично важливо для обробки великих обсягів ринкових даних, які оновлюються кожну хвилину в рамках основного сценарію використання FetchAndStoreCryptoPricesUseCase.

Ще однією перевагою MongoDB є її підтримка горизонтального масштабування через шардування та реплікацію, що дозволяє розподіляти навантаження і забезпечувати надійність системи. У проекті ці можливості використовувалися для забезпечення стабільної роботи бэкенду при обробці даних із кількох бірж одночасно. Крім того, MongoDB підтримує індексацію, що прискорює виконання запитів до даних, таких як отримання останніх цін чи ринкової статистики за конкретною торговою парою, що є основою функціоналу RESTful API.

2.4.2 Mongoose ODM

Для спрощення взаємодії з MongoDB на бэкенді використано бібліотеку Mongoose ODM (Object Data Modeling), яка забезпечує об’єктно-орієнтований інтерфейс для роботи з базою даних. Mongoose дозволяє визначати схеми для доменних сутностей, таких як ExchangeModel, SymbolModel, CryptoPriceModel, MarketDataModel і UserModel, що відповідають структурі даних у проекті. Ці схеми забезпечують чітке визначення структури документів, включаючи типи даних, обов’язкові поля та зв’язки між сутностями, що підвищує надійність і консистентність даних.

У проекті Mongoose використовувався для реалізації репозиторіїв, які абстрагують доступ до бази даних відповідно до принципів Domain-Driven Design (DDD). Наприклад, репозиторій CryptoPrice відповідає за створення, оновлення та отримання цінових даних, тоді як Exchange і Symbol забезпечують управління інформацією про біржі та торгові пари. Mongoose спрощує виконання складних запитів, таких як агрегація ринкових даних чи фільтрація цін за певною парою, що використовується в ендпоінтах RESTful API, наприклад /crypto-prices/latest або /market-data/pair/:pair. Завдяки підтримці асинхронних операцій через async/await Mongoose інтегрувався з Node.js і TypeScript, забезпечуючи ефективну та типобезпечну роботу з даними.

Однією з ключових особливостей Mongoose є підтримка валідації даних на рівні схем, що дозволяє перевіряти коректність інформації перед збереженням у базу. Наприклад, для сутності CryptoPrice визначено обов’язкові поля, такі як symbolId, exchangeId і price, що запобігає збереженню некоректних даних. Крім того, Mongoose забезпечує зручну роботу зі зв’язками між сутностями, наприклад, через використання ObjectId для пов’язування цін із відповідними біржами та торговими парами.

2.5 Бібліотеки для роботи з API бірж

Бібліотеки для роботи з API відіграють важливу роль у розробці веб-сервісів, забезпечуючи зручний і стандартизований доступ до зовнішніх даних. Вони дозволяють розробникам спрощувати інтеграцію з різноманітними сервісами, обробляти запити та відповіді, а також оптимізувати роботу з великими обсягами інформації. У контексті веб-сервісу Crypto Screener, який призначений для збору та аналізу статистики криптовалют у реальному часі, бібліотеки для роботи з API бірж стали основою для отримання ринкових даних із різних криптовалютних платформ. Для реалізації цього функціоналу використано бібліотеки CCXT і Axios, які забезпечили ефективну взаємодію з API бірж і RESTful API, відповідно, дозволяючи швидко отримувати та обробляти актуальну інформацію.

2.5.1 CCXT

Для Бібліотека CCXT (CryptoCurrency eXchange Trading Library) була обрана для бэкенду Crypto Price Parser завдяки її здатності забезпечувати уніфікований доступ до API численних криптовалютних бірж, таких як Binance, Coinbase і Kraken. CCXT надає стандартизований інтерфейс для виконання запитів, що дозволяє отримувати дані про ціни, обсяги торгів і ринкову статистику в єдиному форматі, незалежно від специфіки кожної біржі. У проекті ця бібліотека використовувалася для реалізації основного сценарію FetchAndStoreCryptoPricesUseCase, який періодично отримує цінові дані та зберігає їх у базі MongoDB. Завдяки інтеграції з Node.js і TypeScript, CCXT забезпечила надійність і гнучкість при обробці великих обсягів даних, що є критично важливим для роботи сервісу в реальному часі.

Для взаємодії фронтенду з RESTful API використано бібліотеку Axios, яка забезпечує зручне виконання HTTP-запитів. Axios використовувався для отримання даних із ендпоінтів, таких як /crypto-prices/latest і /market-data, що дозволило фронтенду відображати актуальну ринкову статистику. Інтеграція з TanStack Query спростила кешування і автоматичне оновлення даних, що підвищило

продуктивність інтерфейсу. Поєднання CCXT і Axios забезпечило ефективну роботу з даними на всіх рівнях системи, від збору інформації з бірж до її відображення користувачам, що відповідає вимогам трейдерів та інвесторів до швидкого доступу до ринкової інформації.

2.6 Інтсрументи для зберігання та розгортання проекту

Процес створення сучасного веб-сервісу потребує використання спеціалізованих інструментів, які автоматизують компіляцію коду, управління залежностями та підготовку додатку до роботи в продуктивному середовищі. Такі інструменти відіграють вирішальну роль у забезпеченні ефективності розробки, оптимізації ресурсів і стабільності системи під час її експлуатації. У рамках розробки веб-сервісу Crypto Screener, який забезпечує моніторинг і аналіз криптовалютної статистики в реальному часі, інструменти для збирання та розгортання були обрані з урахуванням необхідності швидкої компіляції фронтенду, ефективного управління бібліотеками та гнучкості при розгортанні системи. Для цих цілей використано інструмент збирання Vite, менеджер пакетів npm і розглянуто підходи до розгортання, які відповідають вимогам проекту до продуктивності та масштабованості.

2.6.1 Vite

Для збирання фронтенду веб-сервісу Crypto Screener, побудованого на основі React 19 і архітектури Feature-Sliced Design, використано сучасний інструмент Vite. Vite є високопродуктивним інструментом збирання, який забезпечує швидку компіляцію та оптимізацію коду, що робить його ідеальним вибором для створення сучасних веб-додатків. У порівнянні з традиційними інструментами, такими як Webpack, Vite використовує нативні модулі ES (ECMAScript Modules) для швидкої обробки коду під час розробки, що значно прискорює запуск локального сервера та оновлення сторінок при зміні коду. У проекті Vite відповідав за компіляцію React-компонентів, таких як таблиця для відображення ринкової статистики, модальне

вікно для фільтрації та секція порівняння цін, а також за оптимізацію статичних ресурсів, включаючи стилі Material UI і графіки Recharts.

Однією з ключових особливостей Vite є підтримка гарячої заміни модулів (Hot Module Replacement, HMR), яка дозволяє розробникам бачити зміни в інтерфейсі в реальному часі без необхідності повного перезавантаження сторінки. Це значно прискорило процес розробки фронтенду, особливо при роботі з компонентами, які інтегруються з бібліотеками TanStack Query і Zustand для динамічного оновлення даних і управління станом. Vite також забезпечив оптимізацію кінцевого бандлу, зменшуючи розмір вихідних файлів і прискорюючи завантаження веб-сервісу в браузері, що є важливим для забезпечення комфортної роботи користувачів, таких як трейдери та інвестори, які потребують швидкого доступу до ринкової інформації.

Інтеграція Vite з TypeScript і React дозволила ефективно обробляти типізований код, забезпечуючи коректну компіляцію компонентів і їхніх пропсів. Крім того, Vite підтримує розширену конфігурацію, що дозволило налаштувати збірку для роботи з бібліотеками, такими як Material UI і Recharts, а також забезпечити сумісність із модульною структурою FSD. Завдяки цьому інструменту вдалося створити оптимізований і продуктивний фронтенд, який відповідає вимогам до швидкості та адаптивності.

2.6.2 npm для управління залежностями

Управління залежностями є критично важливим аспектом розробки сучасних веб-додатків, оскільки дозволяє ефективно інтегрувати зовнішні бібліотеки та забезпечувати їх коректну роботу. У проекті Crypto Screener для управління залежностями використано менеджер пакетів npm, який є стандартним інструментом у екосистемі Node.js. npm відповідав за встановлення та оновлення бібліотек, таких як CCXT для взаємодії з API бірж, Mongoose для роботи з MongoDB, Axios для виконання HTTP-запитів і React для створення інтерфейсу. Завдяки npm вдалося організувати єдину систему управління залежностями для

бэкенду, RESTful API і фронтенду, що спростило підтримку проекту і забезпечило консистентність версій бібліотек.

Менеджер npm також використовувався для виконання скриптів, які автоматизували процеси розробки та збирання. Наприклад, скрипти для запуску локального сервера, компіляції коду за допомогою Vite і виконання команд Node.js були визначені у файлі package.json, що дозволило розробникам швидко налаштовувати робоче середовище. Крім того, npm підтримує кешування залежностей, що прискорило процес встановлення бібліотек і зменшило час, необхідний для розгортання проекту на нових середовищах. Інтеграція з Visual Studio Code дозволила виконувати npm-скрипти безпосередньо з терміналу редактора, що підвищило ефективність робочого процесу.

3 ПРОЄКТУВАННЯ

3.1 Проєктування архітектури застосунку

Архітектура веб-сервісу Crypto Screener була спроектована для забезпечення ефективного збору, обробки та відображення статистики криптовалют у реальному часі, що відповідає потребам трейдерів та інвесторів на криптовалютному ринку. Система складається з трьох основних компонентів, кожен із яких виконує чітко визначену функцію: сервіс збору даних із криптовалютних бірж, RESTful API для надання даних і фронтенд-додаток для їхньої візуалізації та аналізу. Такий поділ дозволяє досягти модульності, полегшує підтримку системи та забезпечує її масштабованість. Для реалізації серверної частини застосовано архітектурний підхід Domain-Driven Design (DDD), тоді як фронтенд побудовано за принципами Feature-Sliced Design (FSD). Взаємодія між компонентами здійснюється через RESTful API, що забезпечує гнучкий і стандартизований обмін даними.

Сервіс збору даних, або Crypto Price Parser, є бэкенд-компонентом, відповідальним за отримання ринкової інформації з криптовалютних бірж, таких як Binance, Coinbase і Kraken. Цей компонент періодично отримує дані про ціни, обсяги торгів і ринкову статистику через бібліотеку CCXT, обробляє їх і зберігає в базі даних MongoDB. Архітектура DDD, використана для цього сервісу, передбачає поділ на кілька шарів. Доменний шар містить основні сутності, такі як біржа, торгова пара, цінові дані та ринкова статистика, а також їхню поведінку. Прикладний шар реалізує сценарії використання, зокрема основний сценарій FetchAndStoreCryptoPricesUseCase, який координує отримання та збереження даних. Інфраструктурний шар забезпечує взаємодію з зовнішніми API бірж і базою даних, тоді як шар представлення відповідає за запуск періодичних завдань. Така структура дозволяє ізолювати бізнес-логіку від технічних деталей, що сприяє легкості модифікації та розширення функціоналу.

RESTful API виступає посередником між сервісом збору даних і фронтендом, надаючи структурований доступ до інформації через набір HTTP-ендпоінтів. Цей компонент також побудовано за принципами DDD, що забезпечує чітке розділення бізнес-логіки, обробки запитів і доступу до даних. Ендпоінти, такі як /exchanges, /symbols, /crypto-prices і /market-data, дозволяють отримувати інформацію про біржі, торгові пари, ціни та ринкову статистику. Додаткові ендпоінти, наприклад /auth і /watch-list, підтримують аутентифікацію користувачів і створення персоналізованих списків спостереження. Архітектура API передбачає використання контролерів для обробки запитів, сервісів для реалізації бізнес-логіки та репозиторіїв для взаємодії з MongoDB через Mongoose ODM. Застосування чистої архітектури забезпечує гнучкість, дозволяючи легко додавати нові ендпоінти або модифікувати існуючі без впливу на інші компоненти системи.

Фронтенд-додаток Crypto Screener відповідає за відображення ринкової статистики та взаємодію з користувачем. Він побудований на основі React 19 і організований за принципами Feature-Sliced Design, що забезпечує модульність і полегшує підтримку коду. Архітектура фронтенду поділена на шари, включаючи конфігурацію додатку, сторінки, функціональні модулі, сутності та спільні компоненти. Основні компоненти, такі як таблиця для відображення цін, модальне вікно для фільтрації, секція порівняння цін і блок із детальною інформацією про криптовалюту, спроєктовані для забезпечення інтуїтивно зрозумілого інтерфейсу. Взаємодія з RESTful API здійснюється через бібліотеку Axios і TanStack Query, що дозволяє отримувати та кешувати дані. Управління станом реалізовано через Zustand, а візуалізація даних — через Recharts. Використання Material UI забезпечує адаптивний дизайн і підтримку мультимовності, що підвищує зручність для користувачів.

Взаємодія між компонентами системи спроєктована для забезпечення високої продуктивності та надійності. Сервіс збору даних періодично оновлює базу MongoDB, яка слугує єдиним джерелом даних для RESTful API. Фронтенд-додаток надсилає запити до API для отримання актуальної інформації, яка відображається в інтерфейсі. Асинхронна модель Node.js на бэкенді та віртуальний DOM у React

дозволяють мінімізувати затримки при обробці великих обсягів даних. Для підвищення безпеки API використовує middleware для аутентифікації та обробки помилок, що захищає систему від несанкціонованого доступу.

Особливу увагу приділено масштабованості та гнучкості архітектури. Використання MongoDB як документно-орієнтованої бази даних дозволяє легко адаптувати структуру даних до нових вимог, наприклад, додавання нових бірж або типів ринкової статистики. Модульна структура DDD і FSD забезпечує можливість додавання нових функцій без значних змін у коді. Наприклад, нові ендпоінти API або React-компоненти можуть бути інтегровані з мінімальними зусиллями. Такий підхід гарантує, що система залишатиметься актуальною та здатною відповідати зростаючим потребам користувачів.

Загальна архітектура веб-сервісу Crypto Screener забезпечує чітке розділення обов’язків між компонентами, що полегшує розробку, тестування та підтримку. Поєднання DDD для серверної частини та FSD для фронтенду, разом із RESTful API як сполучною ланкою, створює надійну та гнучку систему, яка відповідає вимогам до швидкості, точності та зручності аналізу криптовалютної статистики.

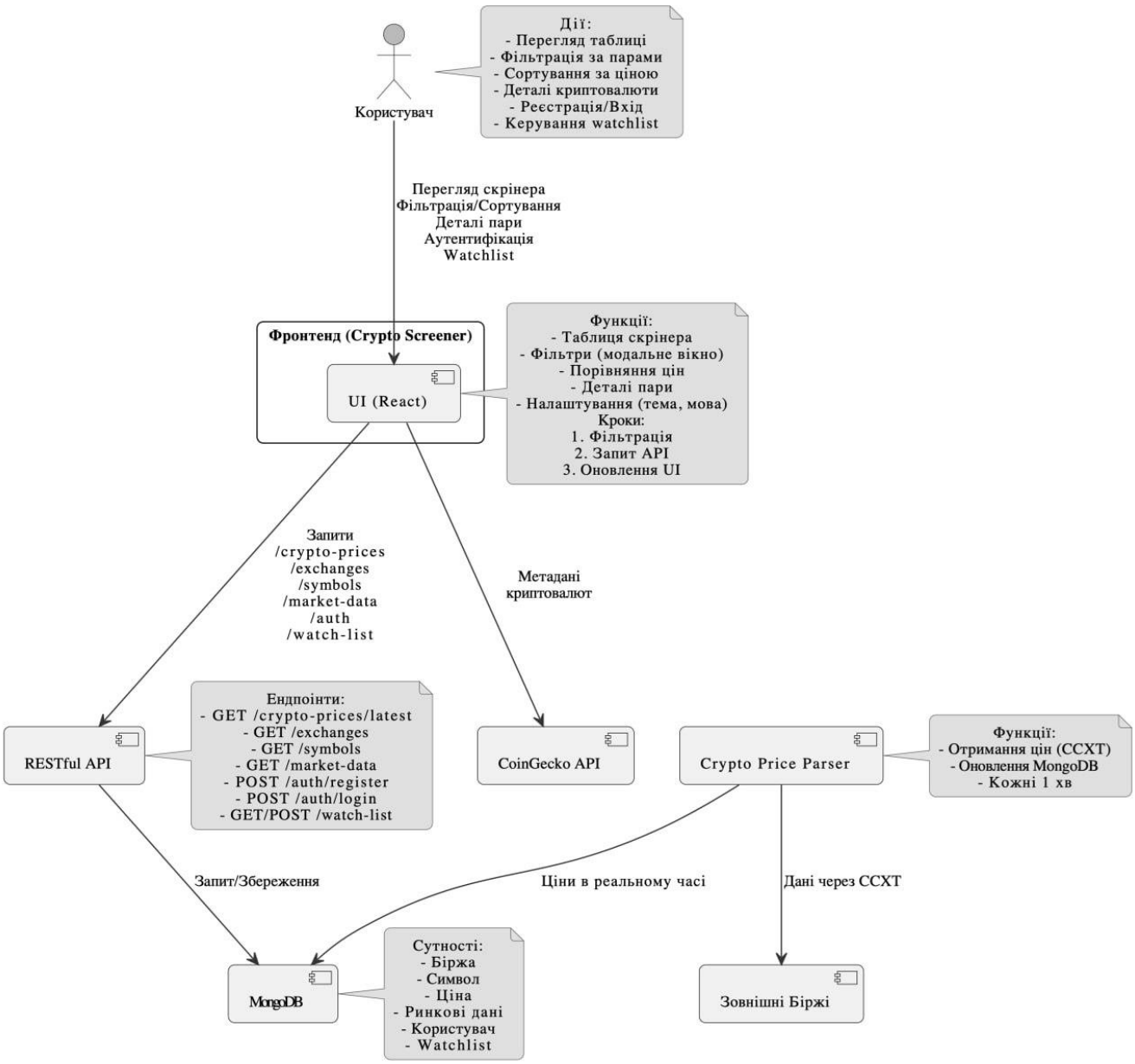


Рис. 3.1 Діаграма-використання архітектури застосунку

3.2 Проєктування архітектури сервісу збору даних із криптовалютних бірж

Сервіс збору даних із криптовалютних бірж, відомий як Crypto Price Parser, є основним бэкенд-компонентом веб-сервісу Crypto Screener, спроєктованим для автоматизованого отримання ринкової інформації в реальному часі та її збереження для подальшого використання. Цей сервіс відповідає за отримання даних про ціни, обсяги торгів і ринкову статистику з провідних криптовалютних бірж, таких як

Binance, Coinbase і Kraken, з їхньою подальшою обробкою та зберіганням у базі даних. Проєктування сервісу базується на архітектурному підході Domain-Driven Design (DDD), який забезпечує чітке розділення бізнес-логіки від інфраструктурних деталей, сприяючи модульності, надійності та можливості масштабування системи. Архітектура сервісу, його робочий процес і технічні рішення були ретельно сплановані для забезпечення високої продуктивності та відповідності вимогам до роботи з великими обсягами даних у реальному часі.

Архітектура сервісу збору даних побудована відповідно до класичного підходу DDD, який передбачає поділ системи на чотири основні шари. Доменний шар формує ядро системи, містить основні бізнес-сутності та їхню поведінку, що відображають ключові аспекти предметної області криптовалютного ринку. До таких сутностей належать біржа, яка представляє платформи торгівлі, торгова пара, що описує комбінацію валют, цінові дані, які фіксують ринкову вартість, і ринкові дані, що включають додаткову статистику. Прикладний шар відповідає за координацію бізнес-логіки через сценарії використання, які організують взаємодію між сутностями. Інфраструктурний шар забезпечує технічну реалізацію, включаючи доступ до бази даних і зовнішніх API. Шар представлення слугує точкою входу для виконання періодичних завдань, таких як запускання процесів збору даних за розкладом.

Основними сутностями доменного шару є чотири ключові об’єкти, які відображають структуру даних системи. Сутність біржі містить інформацію про назву платформи, кінцеву точку API та графічне представлення у вигляді SVG-іконки. Торгова пара описує комбінацію валют, наприклад, BTC/USDT, і включає дані про базову та котирувальну валюту, а також опис, що охоплює огляд, історію та принципи роботи. Сутність цінових даних є центральною, зберігаючи інформацію про ціну, обсяги торгів за різні періоди (1 година, 24 години, 7 днів, 30 днів і загальний обсяг) і часову мітку. Ринкові дані доповнюють систему, надаючи статистику про зміну ціни за 24 години та діапазон цін (мінімум і максимум). Ці сутності спроєктовані для гнучкої роботи з неструктурованими даними, що відповідає можливостям документно-орієнтованої бази MongoDB.

Робочий процес сервісу збору даних був спроєктований для забезпечення безперервного оновлення ринкової інформації з мінімальними затримками. На етапі ініціалізації сервіс встановлює з’єднання з базою даних MongoDB і створює необхідні репозиторії та сервіси. Репозиторії, реалізовані через Mongoose ODM, забезпечують доступ до даних про біржі, торгові пари, ціни та ринкову статистику. Сервіс ExchangeClient, який використовує бібліотеку CCXT, відповідає за взаємодію з API бірж, уніфікуючи запити до різних платформ. Основний сценарій використання FetchAndStoreCryptoPricesUseCase координує потік даних. Цей сценарій періодично отримує список усіх доступних бірж і торгових пар із бази даних, після чого для кожної комбінації біржі та пари виконує запити до API для отримання актуальних цін, обсягів торгів і ринкової статистики. Отримані дані обробляються, формуються у вигляді доменних сутностей і зберігаються в MongoDB. Процес повторюється з налаштованим інтервалом, який за замовчуванням становить одну хвилину, що забезпечує актуальність інформації.

Технічна реалізація сервісу спирається на сучасні інструменти, які гарантують продуктивність і надійність. Для розробки використано мову програмування TypeScript, яка забезпечує типобезпеку та полегшує роботу зі складними структурами даних. MongoDB обрано як основну базу даних завдяки її здатності працювати з неструктурованими даними та підтримці горизонтального масштабування. Бібліотека Mongoose ODM спрощує взаємодію з базою, дозволяючи визначати схеми для сутностей і виконувати валідацію даних. Бібліотека CCXT забезпечує уніфікований доступ до API бірж, що дозволяє легко додавати підтримку нових платформ без значних змін у коді. Для генерації унікальних ідентифікаторів використано бібліотеку UUID, що гарантує унікальність записів у базі даних.

Проєктування сервісу передбачало створення кількох ключових компонентів, які забезпечують його функціональність. Компонент ExchangeClient відповідає за підключення до API бірж і обробку отриманих даних. Репозиторії реалізують операції створення, читання, оновлення та видалення (CRUD) для доменних сутностей, ізолюючи доступ до бази даних. Сценарії використання інкапсулюють

бізнес-логіку, дозволяючи чітко визначити послідовність операцій. Об’єкти-значення, такі як ціна, представляють незмінні концепції, що підвищує надійність системи. Сервіси, зокрема ExchangeService, надають спеціалізовані операції, наприклад, обробку даних із бірж, що сприяє модульності коду.

Особливу увагу приділено забезпеченню надійності та продуктивності сервісу. Асинхронна модель обробки даних, реалізована через конструкції async/await у TypeScript, дозволяє паралельно виконувати запити до кількох бірж, що зменшує час обробки. Для підвищення стійкості до збоїв передбачено механізми обробки помилок, такі як повторні спроби запитів у разі тимчасових проблем із API бірж. Використання DDD забезпечує чітке розділення обов’язків, що полегшує тестування, підтримку та додавання нових функцій, наприклад, підтримку додаткових бірж або нових типів даних.

Таким чином, проєктування сервісу збору даних із криптовалютних бірж дозволило створити надійну та продуктивну систему, яка є основою для роботи веб-сервісу Crypto Screener. Архітектура DDD, ретельно спроєктовані сутності та робочий процес забезпечують ефективне отримання та збереження ринкової інформації, що відповідає вимогам до швидкості та актуальності даних. Використання сучасних технологій, таких як TypeScript, MongoDB і CCXT, у поєднанні з модульною структурою, гарантує гнучкість і масштабованість сервісу, дозволяючи адаптувати його до майбутніх потреб користувачів.

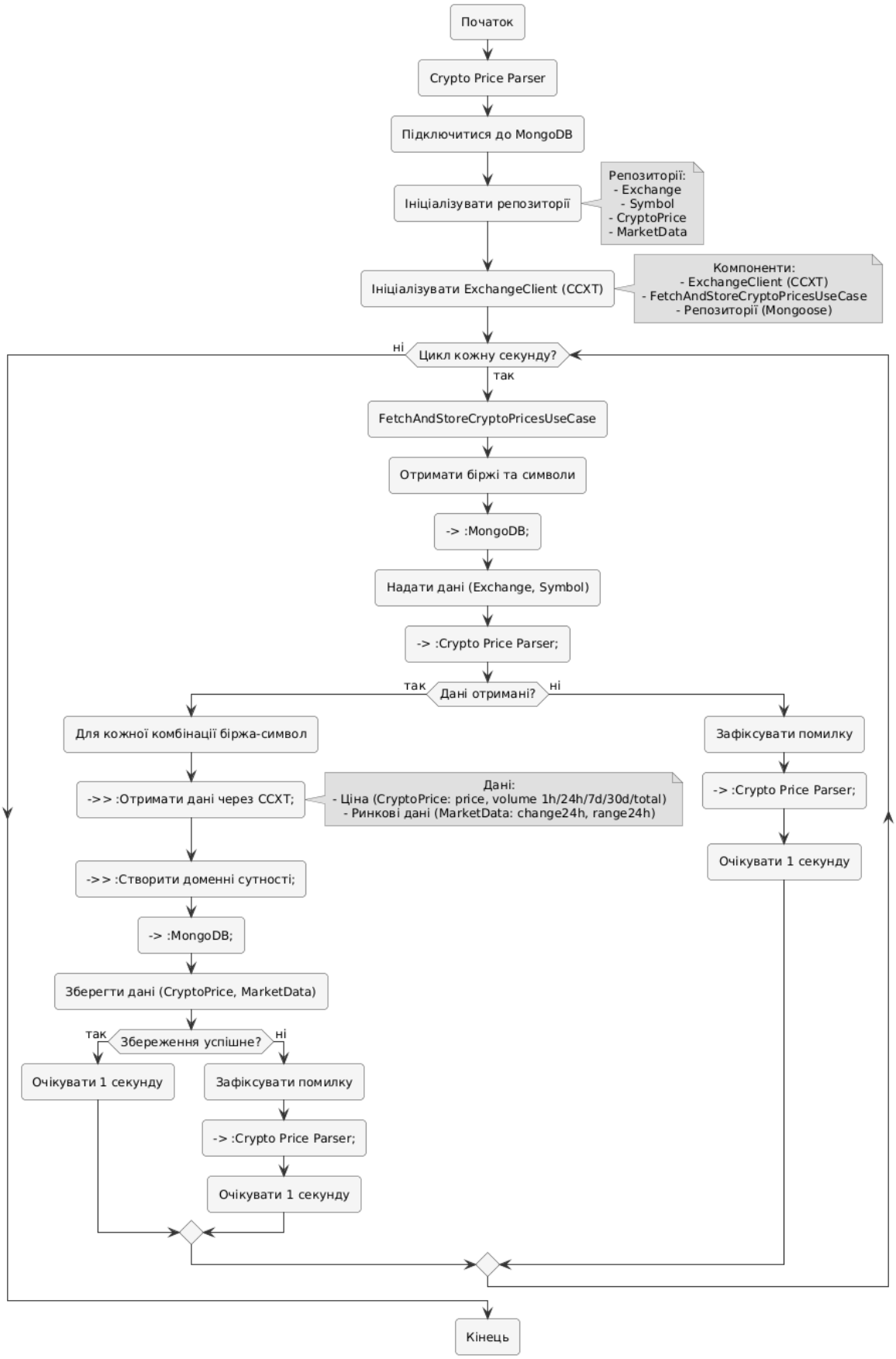


Рис. 3.3 FLOW-Діаграма роботи спроектованого сервісу збору даних із криптовалютних бірж

3.3 Проєктування архітектури RESTful API

Сервіс RESTful API веб-сервісу Crypto Screener спроєктовано як центральний компонент, що забезпечує структурований доступ до ринкової інформації, зібраної сервісом Crypto Price Parser, і її передачу до фронтенд-додатку для відображення користувачам. Цей компонент відіграє роль посередника, дозволяючи клієнтам отримувати дані про криптовалютні біржі, торгові пари, ціни, ринкову статистику, а також підтримувати аутентифікацію та персоналізовані списки спостереження. Проєктування API базується на принципах Domain-Driven Design (DDD) і чистої архітектури, що забезпечує модульність, ізоляцію бізнес-логіки та легкість масштабування. Архітектура API, структура ендпоінтів, моделі даних і технічні рішення були ретельно сплановані для створення надійної, безпечної та гнучкої системи, яка відповідає вимогам трейдерів та інвесторів до швидкого доступу до актуальної ринкової інформації.

Архітектура RESTful API побудована відповідно до принципів DDD, що передбачає поділ системи на чотири основні шари. Доменний шар формує основу API, містить бізнес-сутності, такі як біржа, торгова пара, цінові дані, ринкова статистика, користувач і список спостереження, а також визначає їхню поведінку та інтерфейси репозиторіїв. Прикладний шар реалізує сценарії використання, які координують взаємодію між доменними об’єктами, забезпечуючи виконання бізнес-логіки. Інфраструктурний шар відповідає за технічну реалізацію, включаючи доступ до бази даних MongoDB і конфігурацію системи. Шар представлення обробляє HTTP-запити через контролери та маршрути, а також включає middleware для аутентифікації та обробки помилок. Така структура дозволяє ізолювати бізнес-логіку від зовнішніх залежностей, що полегшує тестування, підтримку та розширення функціоналу.

Основними сутностями API є набір об’єктів, які відображають ключові аспекти предметної області. Сутність біржі містить інформацію про назву, URL і логотип, що дозволяє ідентифікувати торговельні платформи, такі як Binance чи Kraken. Торгова пара описує комбінацію валют, наприклад, BTC/USDT, і включає

дані про базову та котирувальну валюту, а також додаткову інформацію, таку як історія та принципи роботи. Цінові дані фіксують ринкову вартість для конкретної пари на біржі, включаючи обсяги торгів за різні періоди (1 година, 24 години, 7 днів, 30 днів) і часову мітку. Ринкові дані агрегують статистику, таку як зміна ціни за 24 години та діапазон цін, забезпечуючи доступ до аналітичної інформації. Сутність користувача підтримує аутентифікацію, зберігаючи дані для реєстрації та входу, тоді як список спостереження дозволяє створювати персоналізовані набори торгових пар. Ці сутності спроєктовані для гнучкої роботи з MongoDB, що забезпечує адаптивність до змін у структурі даних.

Структура ендпоінтів API була розроблена для забезпечення повного спектру операцій із даними. Набір ендпоінтів для бірж дозволяє отримувати список усіх платформ, детальну інформацію про конкретну біржу, а також створювати, оновлювати або видаляти записи. Аналогічні операції передбачені для торгових пар, що забезпечує управління інформацією про валютні комбінації. Ендпоінти для цінових даних надають доступ до всіх цін, останніх записів або конкретного запису за ідентифікатором, а також підтримують створення, оновлення та видалення даних. Ендпоінти для ринкових даних дозволяють отримувати загальну статистику, дані за ідентифікатором або за конкретною торговою парою. Для аутентифікації передбачено ендпоінти для реєстрації та входу користувачів, тоді як ендпоінти для списків спостереження дозволяють створювати, переглядати та видаляти персоналізовані списки. Така структура забезпечує гнучкість і зручність для клієнтів, включаючи фронтенд-додаток.

Робочий процес API спроєктований для ефективної обробки запитів і забезпечення безпеки. При отриманні HTTP-запиту контролери в шарі представлення аналізують його і передають обробку відповідним сервісам у прикладному шарі. Сервіси реалізують бізнес-логіку, взаємодіючи з доменними сутностями та репозиторіями, які забезпечують доступ до MongoDB через Mongoose ODM. Наприклад, запит до ендпоінта /crypto-prices/latest ініціює виклик сервісу, який отримує найактуальніші цінові дані через репозиторій і повертає їх у форматі JSON. Middleware обробляє аутентифікацію, використовуючи JWT (JSON

Web Tokens) для захисту ендпоінтів, таких як /watch-list, і централізовано обробляє помилки, забезпечуючи інформативні відповіді клієнтам. Використання асинхронної моделі Node.js дозволяє обробляти численні запити одночасно, що підвищує продуктивність API.

Технічна реалізація API спирається на сучасні інструменти, які забезпечують надійність і масштабованість. Мова програмування TypeScript використовується для створення типобезпечного коду, що зменшує кількість помилок і полегшує роботу зі складними структурами даних. MongoDB служе з основною базою даних, а моделі даних, реалізовані через Mongoose, дозволяють чітко визначати схеми для сутностей і забезпечують валідацію даних перед збереженням.

Використання Dependency Injection знижує зв'язність між компонентами, тоді як Repository Pattern абстрагує доступ до бази, полегшуючи заміну MongoDB іншою системою за потреби. Use Case Pattern ізолює бізнес-логіку, що сприяє модульності коду.

Проєктування API передбачало створення компонентів, які забезпечують його функціональність. Контролери оброблюють HTTP-запити, спрямовуючи їх до відповідних сервісів. Репозиторії, такі як ExchangeRepository або CryptoPriceRepository, реалізують операції CRUD, забезпечуючи взаємодію з базою даних.

Сервіси, наприклад, для обробки аутентифікації чи ринкових даних, координують бізнес-логіку. Об'єкти-значення, такі як ціна, представляють незмінні концепції, що підвищує надійність системи. Middleware забезпечує централізовану обробку помилок і аутентифікацію, що додає безпеки та стабільності.

Особливу увагу приділено гнучкості та масштабованості API. Модульна структура дозволяє легко додавати нові ендпоінти, наприклад, для підтримки нових типів ринкових даних чи додаткових функцій. Використання MongoDB забезпечує адаптацію до змін у структурі даних, тоді як чиста архітектура дозволяє модифікувати бізнес-логіку без впливу на інші компоненти.

Для забезпечення безпеки передбачено використання JWT для авторизації та обмеження доступу до захищених ендпоінтів, а також логування запитів для моніторингу та діагностики.

Таким чином, проєктування RESTful API для веб-сервісу Crypto Screener створило ефективний і надійний компонент, який забезпечує доступ до ринкової інформації та підтримує взаємодію між бэкендом і фронтендом. Архітектура DDD, ретельно спроєктована структура ендпоінтів і використання сучасних технологій (TypeScript, MongoDB, Mongoose) гарантують продуктивність, безпеку та гнучкість системи, що відповідає вимогам користувачів до швидкого та зручного аналізу криптовалютної статистики.

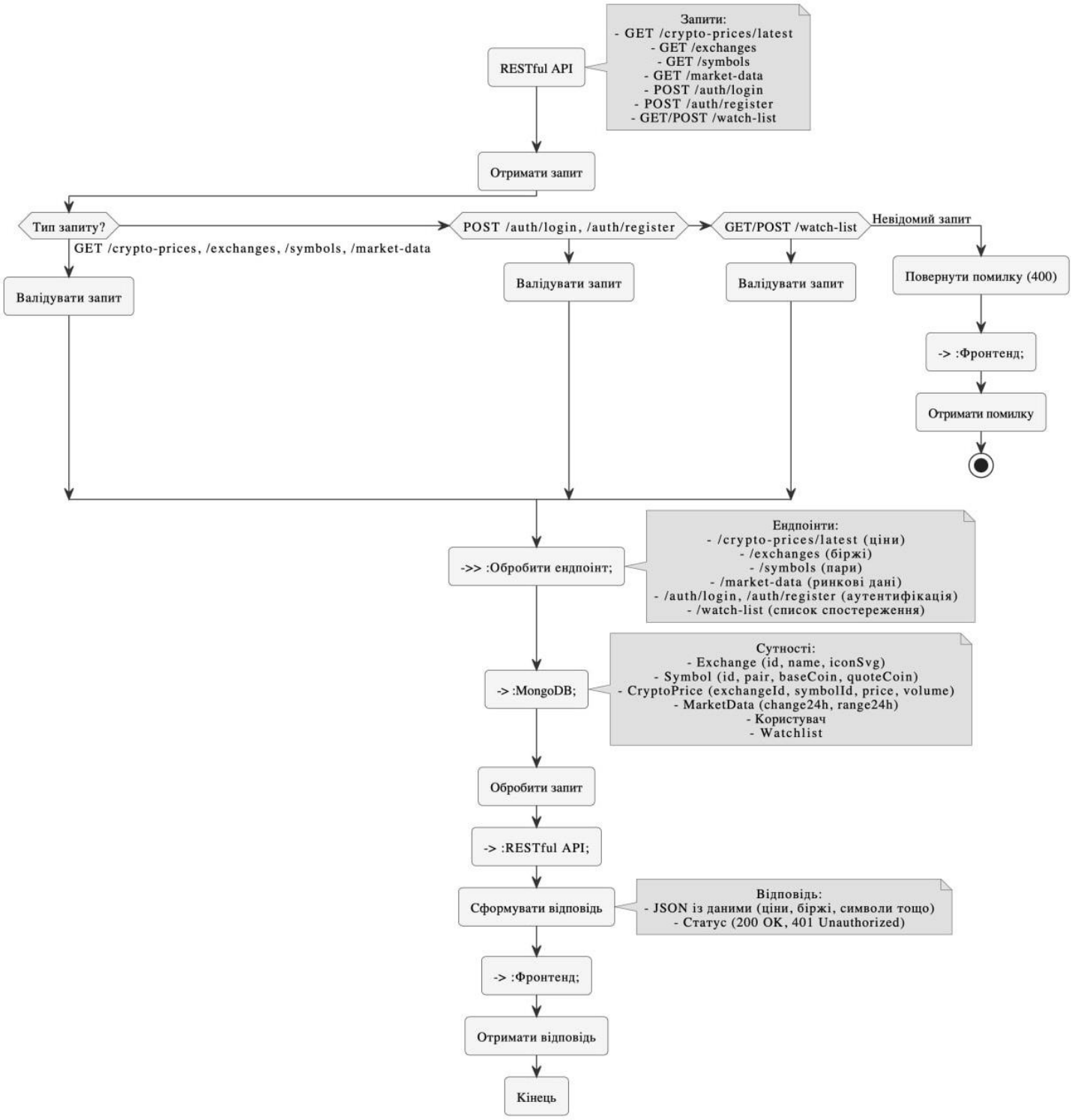


Рис. 3.4 FLOW-Діаграма роботи спроектованого RESTful API сервісу

3.4 Проєктування архітектури фронтенд частини

Фронтенд-частина додатку Crypto Screener є ключовим компонентом веб-сервісу, призначеним для забезпечення зручного та інтерактивного інтерфейсу

користувача, який дозволяє трейдерам та інвесторам аналізувати ринкову статистику криптовалют у реальному часі. Цей додаток відповідає за відображення даних, отриманих через RESTful API, а також за підтримку таких функцій, як фільтрація, сортування, порівняння цін, створення списків спостереження та аутентифікація користувачів. Проєктування фронтенда базується на архітектурному підході Feature-Sliced Design (FSD), який забезпечує модульність, структурованість і легкість підтримки коду. Архітектура додатку, його компоненти, функціональність і технічні рішення були сплановані для створення адаптивного, продуктивного та інтуїтивно зрозумілого інтерфейсу, що відповідає вимогам сучасного веб-сервісу для роботи з криптовалютними даними.

Архітектура фронтенд-додатку побудована за принципами Feature-Sliced Design, що передбачає організацію коду за функціональними шарами для забезпечення чіткого розподілу обов'язків і модульності. Код поділено на п'ять основних шарів. Шар конфігурації додатку містить налаштування, провайдери та маршрути, які визначають загальну структуру системи. Шар сторінок відповідає за реалізацію окремих сторінок, таких як головна сторінка зі скрінером, сторінка деталей торгової пари чи профіль користувача. Функціональні модулі охоплюють бізнес-логіку, пов'язану з конкретними можливостями, наприклад, фільтрацією або порівнянням цін. Шар сутностей включає бізнес-об'єкти, такі як біржа, торгова пара та цінові дані, які використовуються в різних частинах додатку. Спільний шар містить утиліти, API-клієнт і компоненти, що використовуються повторно, наприклад, кнопки чи іконки. Така структура забезпечує гнучкість, дозволяючи легко додавати нові функції, такі як підтримка нових типів ринкових даних, без значних змін у коді.

Основними сутностями фронтенда є об'єкти, які відображають дані, отримані від RESTful API. Сутність біржі включає ідентифікатор, назву та SVG-іконку, що дозволяє відображати інформацію про платформи, такі як Binance чи Kraken. Торгова пара містить ідентифікатор, комбінацію валют (наприклад, BTC/USDT), базову та котирувальну валюту, а також опис, що охоплює огляд і історію. Цінові дані представляють ринкову вартість для конкретної пари на біржі, включаючи

обсяги торгів. Ринкові дані надають додаткову статистику, таку як зміна ціни за 24 години та діапазон цін. Ці сутності використовуються для відображення інформації в інтерфейсі, наприклад, у таблиці скрінера чи секції порівняння цін. Структура даних спроєктована для сумісності з API, що забезпечує коректну обробку відповідей у форматі JSON.

Функціональність фронтенда була спроєктована для забезпечення зручного аналізу ринкової інформації. Основною можливістю є криптовалютний скрінер, який відображає таблицю з даними про торгові пари з різних бірж. Користувачі можуть фільтрувати дані за парами, цінами чи біржами, а також сортувати їх за параметрами, такими як ціна чи обсяг торгів. Функція порівняння цін дозволяє аналізувати ринкову вартість однієї пари на різних біржах, що допомагає виявляти арбітражні можливості. Секція детальної інформації про криптовалюту надає розширений опис, включаючи історію та принципи роботи. Для персоналізації передбачено створення списків спостереження, які дозволяють зберігати обрані пари для швидкого доступу. Аутентифікація підтримує реєстрацію, вхід і керування профілем користувача, що забезпечує безпечний доступ до персоналізованих функцій.

Інтерфейс користувача складається з кількох ключових компонентів, спроєктованих для забезпечення інтуїтивної взаємодії. Компонент таблиці скрінера відображає ринкову статистику у зручному форматі, дозволяючи сортувати та фільтрувати дані. Модальне вікно для фільтрації забезпечує вибір параметрів, таких як біржа чи ціновий діапазон. Навігаційна панель містить меню для доступу до основних сторінок, тоді як секція порівняння цін і блок із детальною інформацією надають аналітичні можливості. Панель налаштувань дозволяє змінювати тему оформлення та мову інтерфейсу. Візуалізація даних, наприклад, графіків зміни цін, реалізована через бібліотеку Recharts, що забезпечує динамічне відображення статистики. Використання Material UI 7 гарантує адаптивний дизайн, що забезпечує зручність на різних пристроях.

Взаємодія з RESTful API була спроєктована для забезпечення реактивності та ефективності. Фронтенд використовує бібліотеку Axios для виконання HTTP-

запитів до ендпоінтів, таких як /crypto-prices/latest, /exchanges, /symbols і /market-data, які надають актуальні дані. Бібліотека TanStack Query забезпечує кешування та автоматичне оновлення даних, що зменшує кількість запитів і підвищує продуктивність інтерфейсу. Для отримання додаткової інформації про криптовалюту використовується API CoinGecko, що розширює аналітичні можливості. Управління станом додатку реалізовано через бібліотеку Zustand, яка зберігає параметри фільтрації, сортування, вибору теми оформлення та мови інтерфейсу (англійська або українська). Локальне зберігання, реалізоване через браузер, використовується для збереження токенів авторизації та налаштувань користувача.

Маршрутизація додатку спроєктована для забезпечення зручної навігації. Головна сторінка відображає скрінер із таблицею даних. Сторінка деталей торгової пари надає розширену інформацію за конкретною біржею та парою. Сторінки реєстрації, входу, профілю користувача, списку спостереження та інформаційної сторінки про проект доступні через відповідні маршрути, реалізовані за допомогою React Router DOM 7. Така структура забезпечує логічну організацію інтерфейсу та швидкий доступ до всіх функцій.

Технічна реалізація фронтенда спирається на сучасний стек технологій. Мова програмування TypeScript забезпечує типобезпеку, що зменшує кількість помилок і полегшує роботу з React-компонентами та API-відповідями. Фреймворк React 19 використовується для створення компонентного інтерфейсу з підтримкою віртуального DOM, що гарантує високу продуктивність. Material UI 7 надає готові стилізовані компоненти, які підтримують адаптивність і темізацію. Бібліотека Recharts забезпечує візуалізацію даних у вигляді графіків, тоді як Vite відповідає за швидку компіляцію та оптимізацію коду. Поєднання цих технологій створює продуктивну та гнучку систему, яка відповідає вимогам сучасного веб-додатку.

Особливу увагу приділено гнучкості та масштабованості фронтенда. Архітектура FSD дозволяє легко додавать нові компоненти чи функції, наприклад, підтримку додаткових типів ринкових даних або нових сторінок. Модульна структура коду полегшує тестування та підтримку, тоді як використання TanStack

Query і Zustand забезпечує ефективне керування даними та станом. Мультимовність і адаптивний дизайн, реалізовані через Material UI, гарантують зручність для широкої аудиторії користувачів. Локальне зберігання налаштувань і токенів авторизації підвищує безпеку та персоналізацію.

Таким чином, проєктування фронтенд-додатку Crypto Screener створило інтерактивний і продуктивний інтерфейс, який забезпечує зручний аналіз криптовалютної статистики. Архітектура FSD, ретельно спроєктовані компоненти та інтеграція з RESTful API і CoinGecko дозволяють ефективно відображати дані, підтримувати персоналізовані функції та адаптуватися до потреб користувачів. Використання сучасних технологій, таких як React 19, TypeScript і Material UI, гарантує надійність, масштабованість і відповідність вимогам до швидкості та зручності роботи з ринковою інформацією.

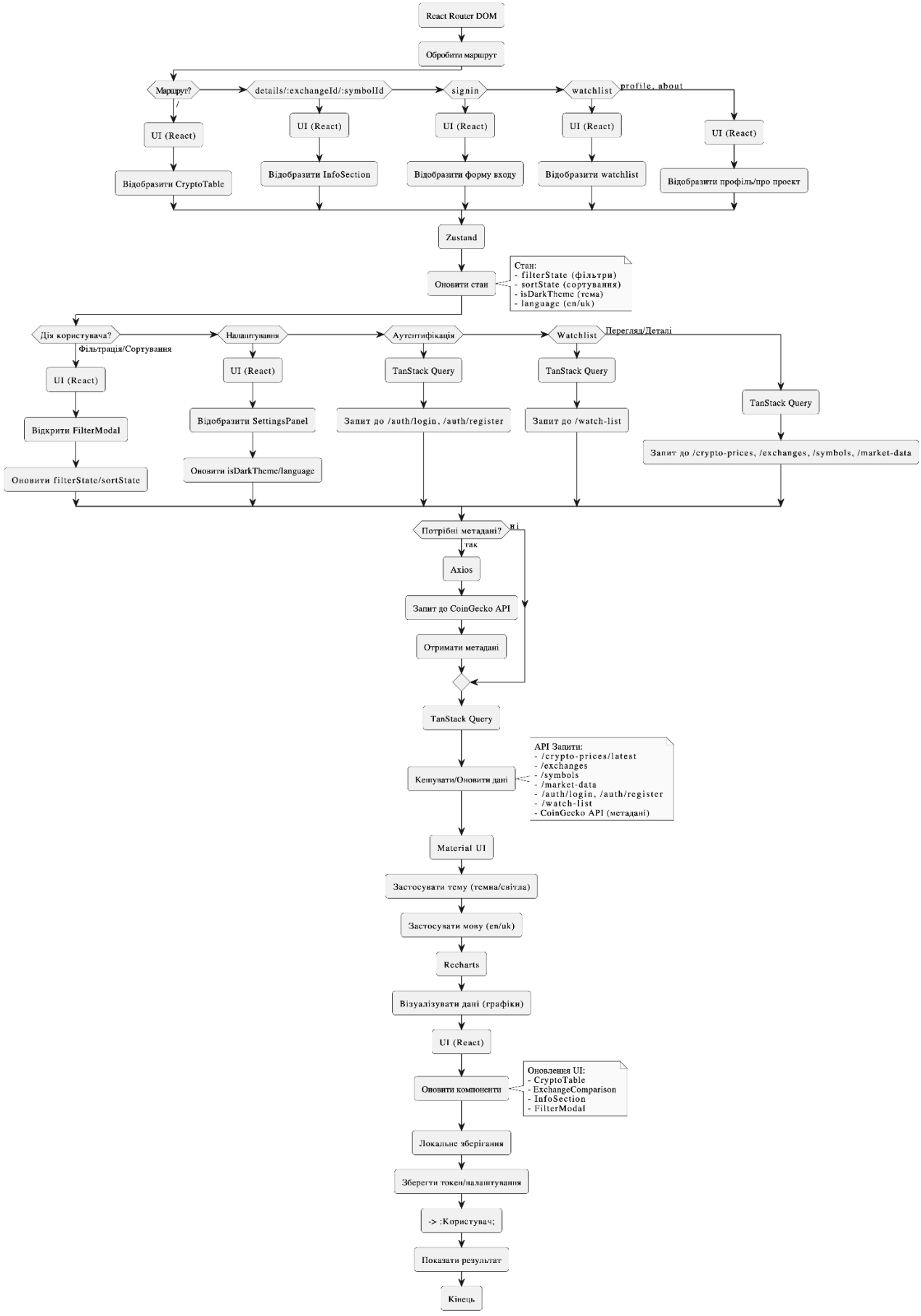


Рис. 3.5 FLOW-Діаграма роботи спроектованої фронтенд частини

4 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

4.1 Дизайн інтерфейсу

Дизайн інтерфейсу користувача є центральним елементом будь-якого веб-сервісу, оскільки від нього залежить зручність використання, інтуїтивність і загальна ефективність взаємодії користувачів із системою. У процесі розробки веб-сервісу Crypto Screener, призначеного для моніторингу та аналізу статистики криптовалют у реальному часі, особлива увага приділялася створенню інтерфейсу, який відповідає потребам цільової аудиторії — трейдерів та інвесторів. Ці користувачі потребують швидкого доступу до актуальних ринкових даних, зручних інструментів для фільтрації, сортування та порівняння цін, а також наочної візуалізації інформації для прийняття обґрунтованих торгових рішень.

Інтерфейс був спроектований з використанням бібліотеки Material UI 7, яка забезпечила сучасний і адаптивний дизайн, а організація компонентів базувалася на принципах архітектури Feature-Sliced Design (FSD), що сприяло модульності та легкості підтримки. Дизайн інтерфейсу охоплює кілька ключових компонентів, кожен із яких виконує специфічну функцію, забезпечуючи інтуїтивний і ефективний користувацький досвід. Для ілюстрації компонентів передбачено розміщення скриншотів після їх опису, що дозволить наочно продемонструвати реалізацію дизайну.

4.1.1 Принципи дизайну інтерфейсу

Розробка інтерфейсу розпочалася з аналізу потреб користувачів, які включають швидкий доступ до ринкової інформації, можливість порівняння цін на різних біржах і персоналізацію функціоналу, наприклад, створення списків спостереження. Основними принципами дизайну були інтуїтивність, адаптивність і візуальна чіткість. Інтуїтивність забезпечувалася завдяки логічному розташуванню елементів і простій навігації, що дозволяє користувачам швидко

орієнтуватися в системі. Адаптивність була реалізована за допомогою бібліотеки Material UI, яка забезпечує коректне відображення інтерфейсу на пристроях із різними розмірами екранів, від настільних комп'ютерів до смартфонів. Візуальна чіткість досягалася через використання контрастних кольорів, чітких шрифтів і графіків, створених за допомогою бібліотеки Recharts, що полегшує сприйняття складних ринкових даних.

Ще однією важливою особливістю дизайну була підтримка мультимовності (англійська та українська мови) і темізації (темна та світла теми), що дозволяє користувачам налаштовувати інтерфейс відповідно до їхніх уподобань. Ці функції були реалізовані через компонент SettingsPanel і збереження налаштувань у локальному сховищі браузера, що забезпечує збереження вибору між сеансами. Такий підхід підвищив доступність сервісу для ширшої аудиторії та забезпечив комфортне використання в різних умовах освітлення.

4.1.2 Головна таблиця скрінера

Основним елементом інтерфейсу є компонент CryptoTable, який відображає таблицю з актуальними даними про торгові пари з різних криптовалютних бірж, таких як Binance, Coinbase і Kraken. Цей компонент дозволяє користувачам переглядати ключові показники, включаючи поточну ціну, обсяги торгів за різні періоди (1 година, 24 години, 7 днів, 30 днів) і зміну ціни за 24 години. Таблиця спроектована з акцентом на зручність і функціональність: користувачі можуть сортувати дані за різними параметрами, такими як ціна чи обсяг торгів, а також застосовувати фільтри за біржами чи торговими парами. Дизайн таблиці базується на компонентах Material UI, які забезпечують чітке форматування, 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.

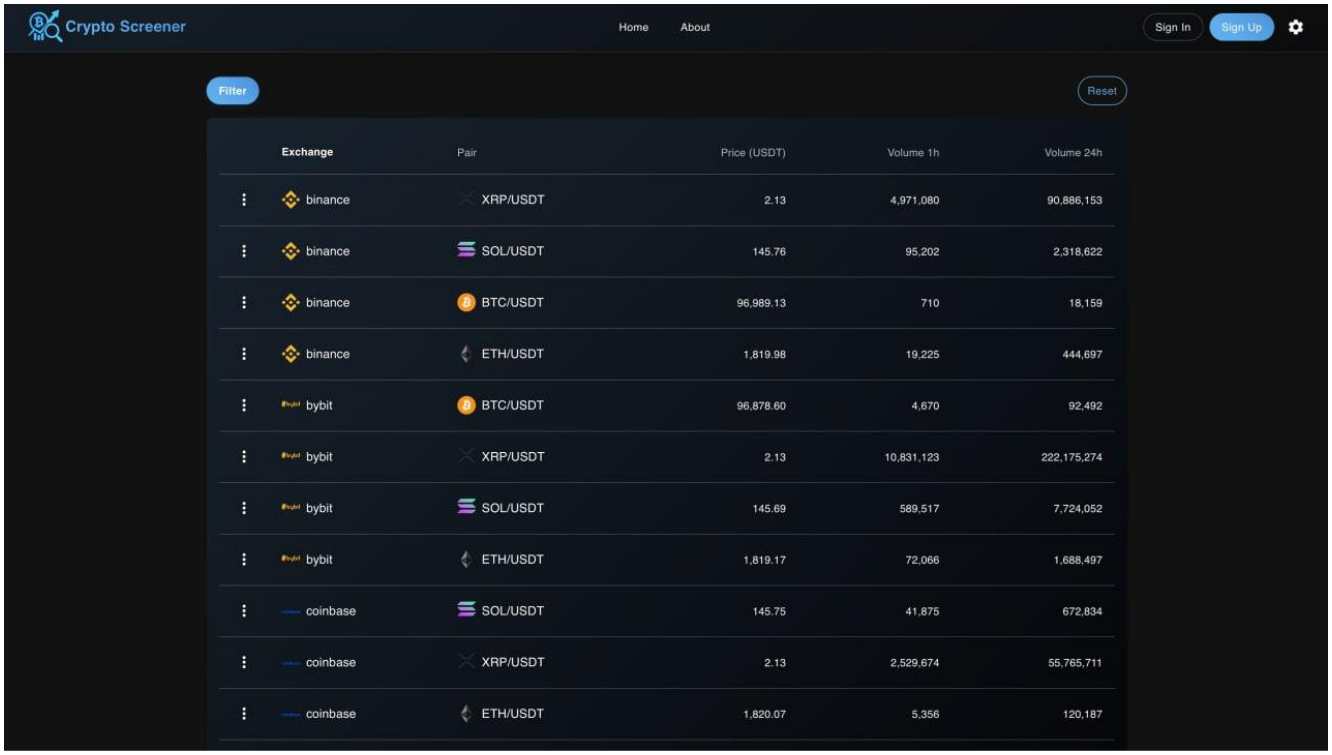


Рис. 4.1 Головний екран застосунку (темна тема, англійська мова) та компонент CryptoTable

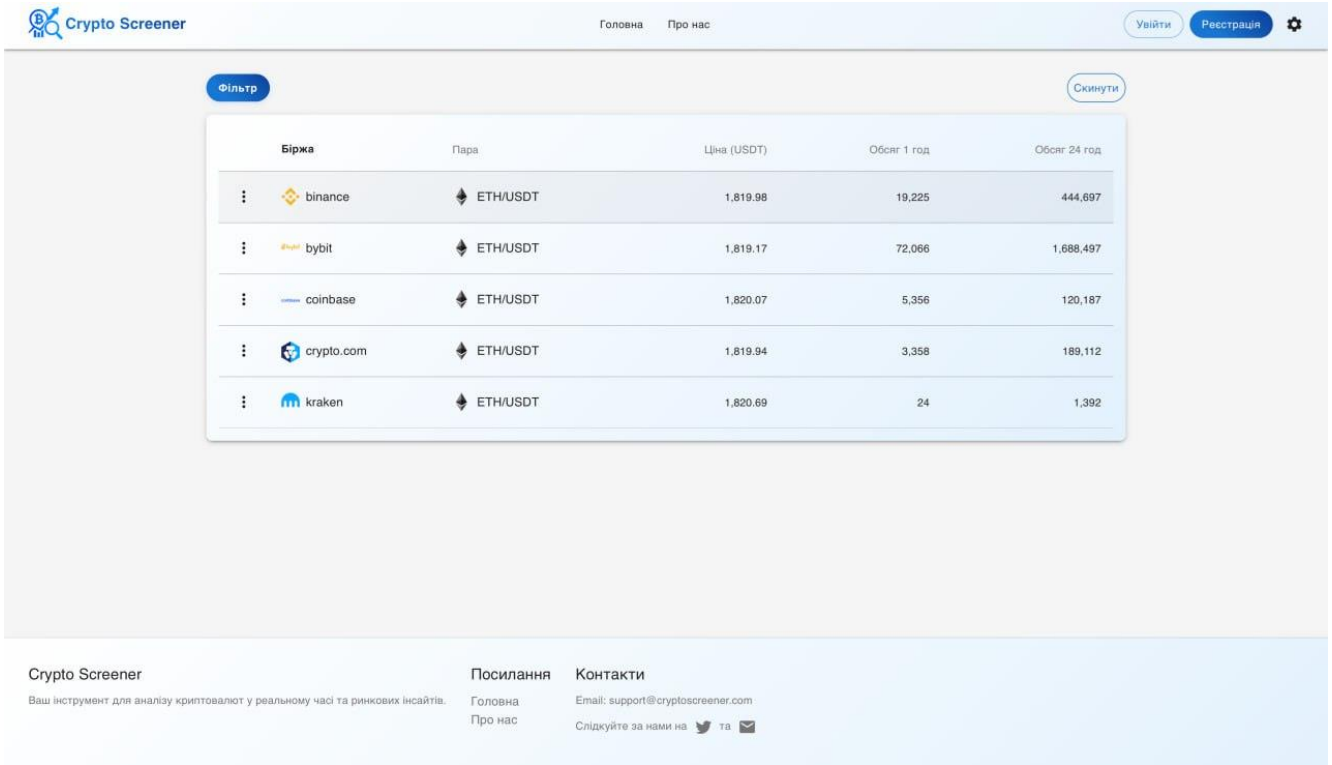


Рис. 4.2 Головний екран застосунку (світла тема, українська мова) та компонент CryptoTable

4.1.3 Модальне вікно фільтрації

Для забезпечення гнучкості в роботі з ринковими даними розроблено компонент FilterModal, який дозволяє користувачам налаштовувати параметри фільтрації. Цей компонент відкривається як модальне вікно і надає можливість вибирати конкретні біржі, торгові пари чи діапазони цін. Дизайн модального вікна виконано з використанням Material UI, що забезпечує єдиний стиль із іншими компонентами та адаптивність для різних пристроїв. Управління станом фільтрів здійснюється через бібліотеку Zustand, що дозволяє динамічно оновлювати таблицю при зміні параметрів. Інтерфейс модального вікна спроектований так, щоб бути інтуїтивно зрозумілим навіть для користувачів із мінімальним досвідом, із чіткими позначеннями полів і кнопками для підтвердження чи скасування вибору.

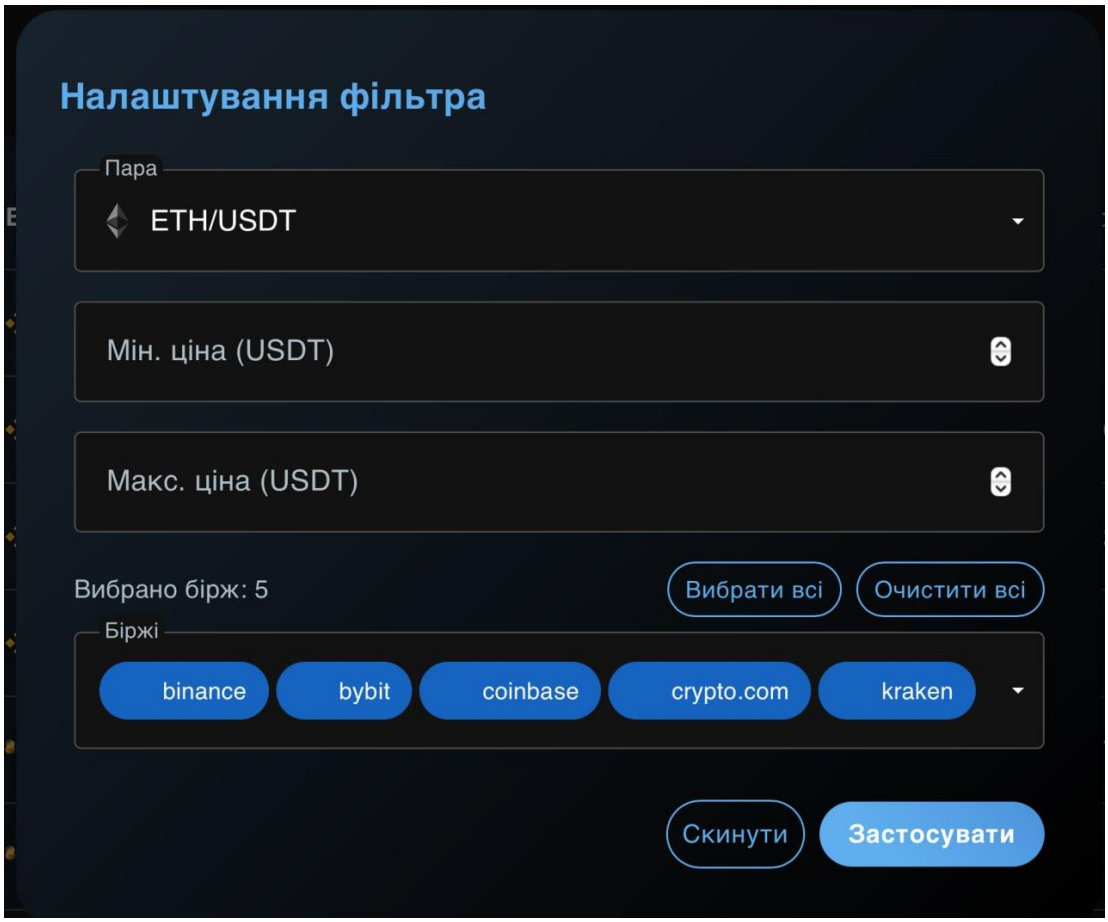


Рис. 4.3 Компонент FilterModal (темна тема)

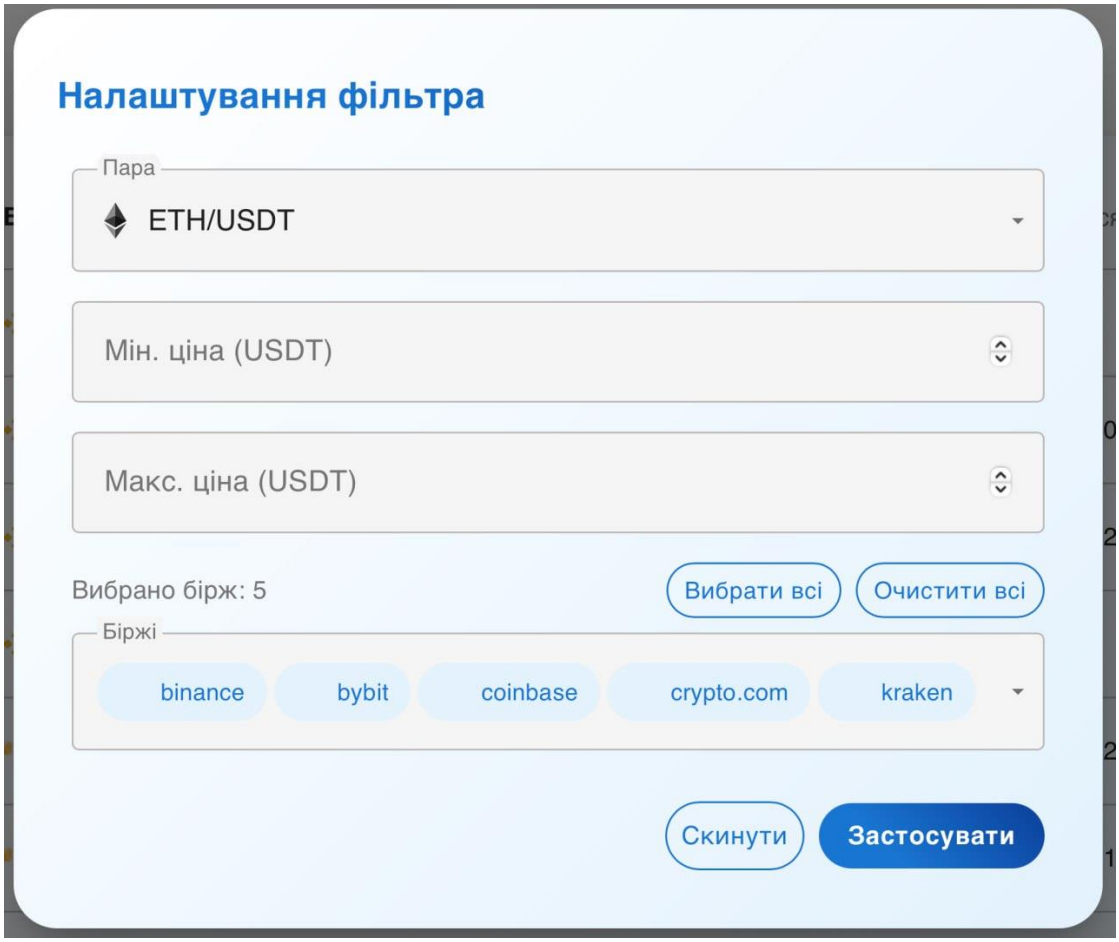


Рис. 4.4 Компонент FilterModal (світла тема)

4.1.4 Секції детальної інформації і опису

Компонент InfoSection відповідає за відображення розширеної інформації про вибрану торгову пару, включаючи її опис, історію, принцип роботи та графіки зміни цін. Цей компонент доступний через сторінку /details/:exchangeId/:symbolId і спроектований для забезпечення глибокого аналізу ринкових даних. Графіки, створені за допомогою бібліотеки Recharts, дозволяють візуалізувати динаміку цін за різні періоди, що є важливим для трейдерів, які аналізують ринкові тенденції. Дизайн секції виконано з акцентом на чіткість: інформація структурована в блоки, а графіки мають інтерактивні елементи, такі як підказки при наведенні. Material UI забезпечує адаптивне розташування елементів, що дозволяє зручно переглядати дані як на великих екранах, так і на мобільних пристроях.

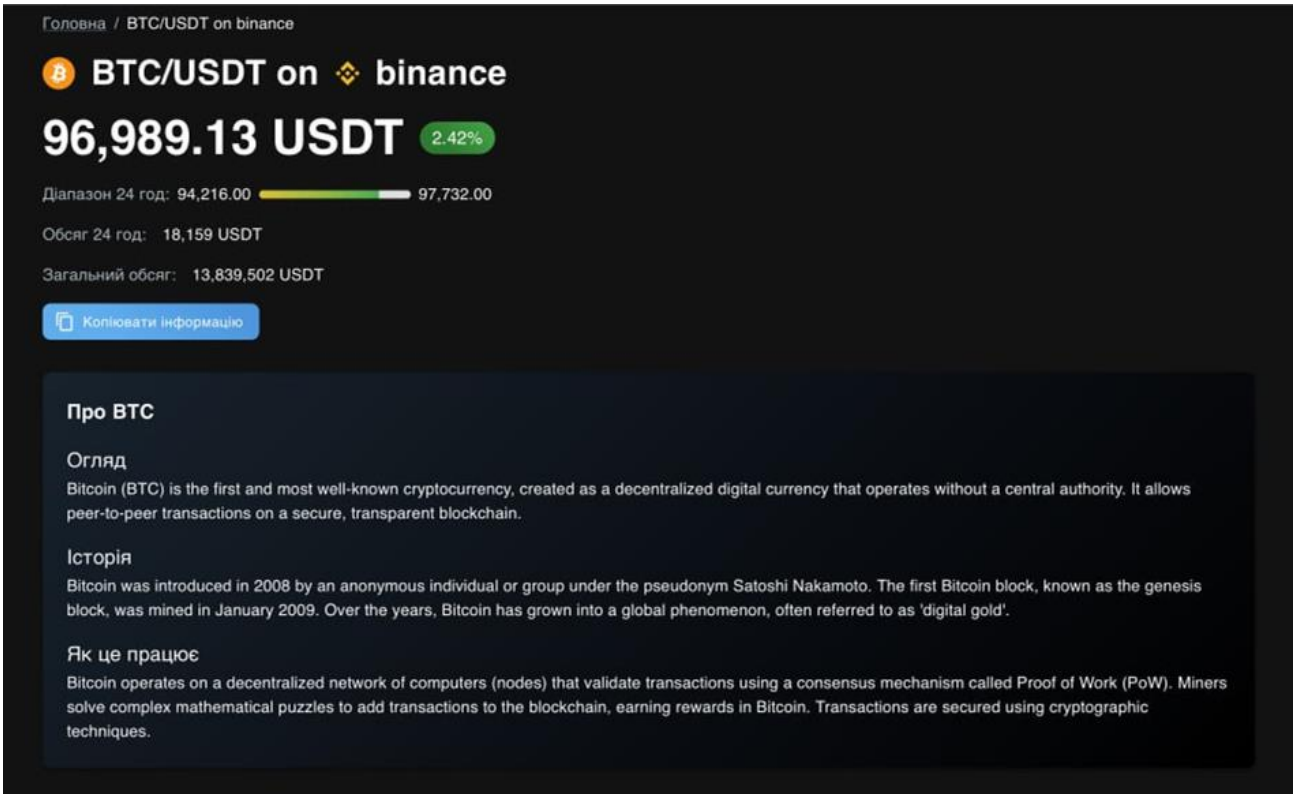


Рис. 4.5 Компонент InfoSection, опис та інформація (темна тема)

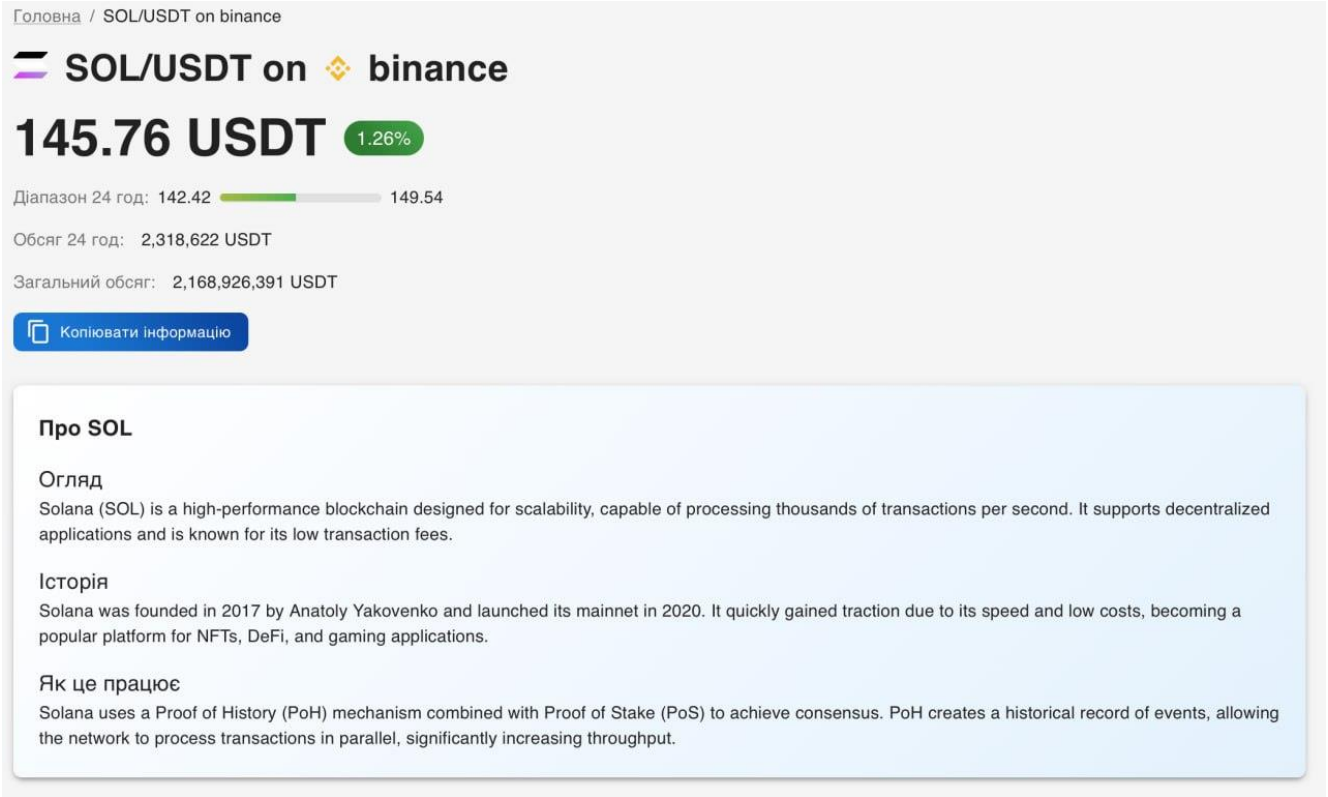


Рис. 4.6 Компонент InfoSection, опис та інформація (світла тема)

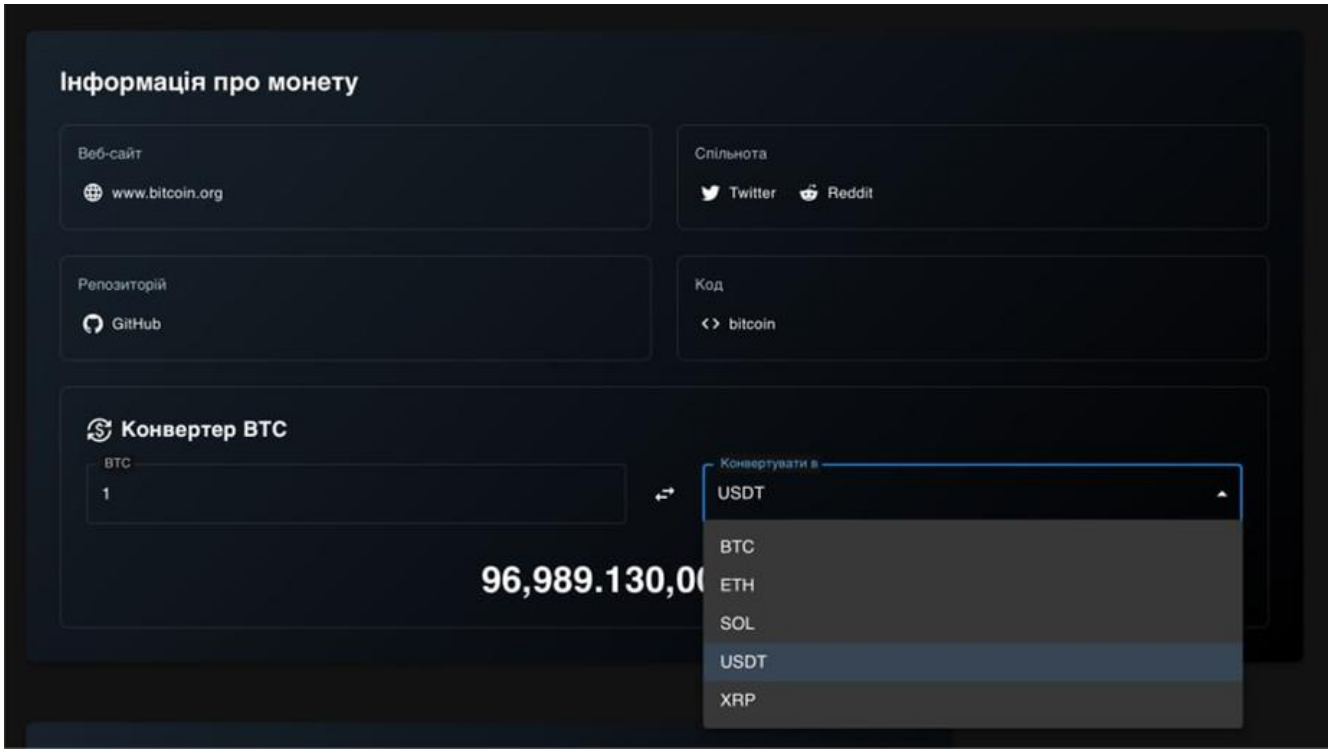


Рис. 4.7 Компонент InfoSection, посилання та конвертер валют (темна тема)

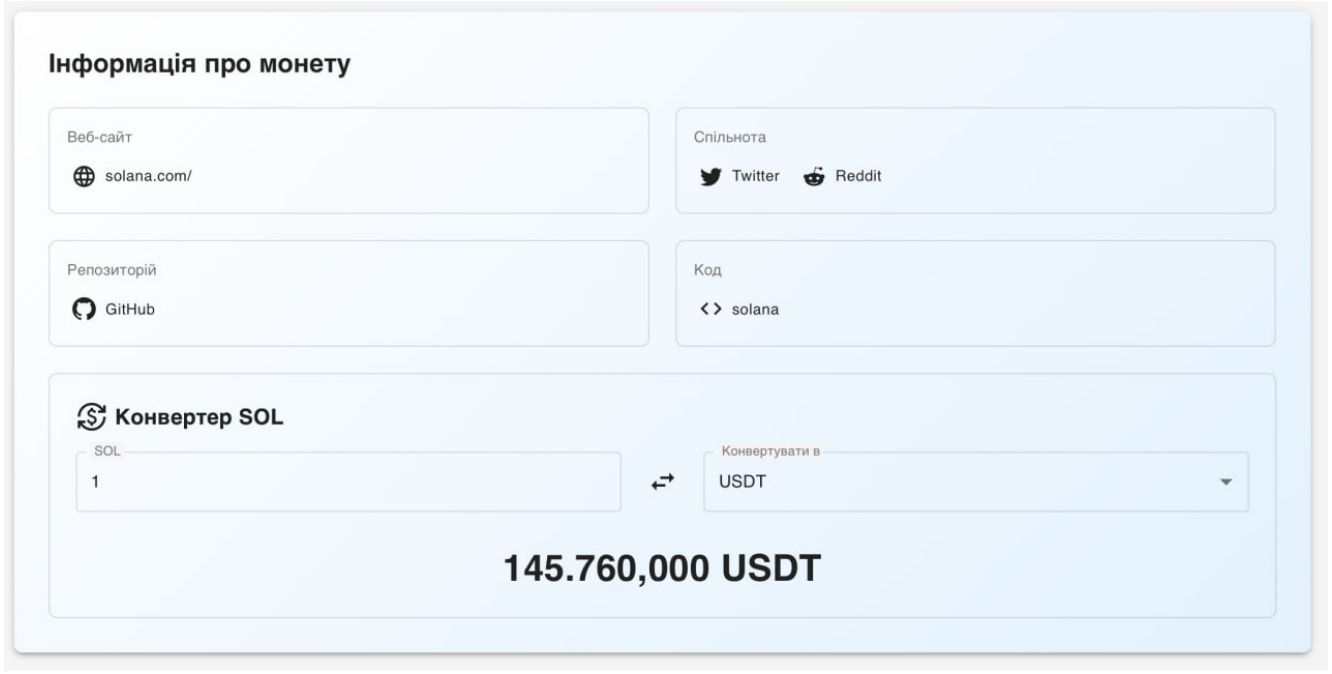


Рис. 4.8 Компонент InfoSection, посилання та конвертер валют (світла тема)

4.1.5 Порівняння цін на біржах

Компонент ExchangeComparison розроблено для порівняння цін однієї торгової пари на різних біржах, що допомагає користувачам виявляти арбітражні можливості. Цей компонент відображає таблиці та графіки, які показують різницю в цінах, обсягах торгів і ринковій статистиці. Дизайн компонента виконано з використанням Material UI для забезпечення єдиного стилю, а інтеграція з Recharts дозволяє створювати наочні графіки для порівняння. Компонент підтримує інтерактивність, дозволяючи користувачам перемикатися між різними біржами та періодам часу. Дані для цього компонента отримуються через API-запити до ендпоінтів, таких як /crypto-prices і /market-data, що забезпечує актуальність інформації.

Рис. 4.9 Компонент ExchangeComparison (темна тема)

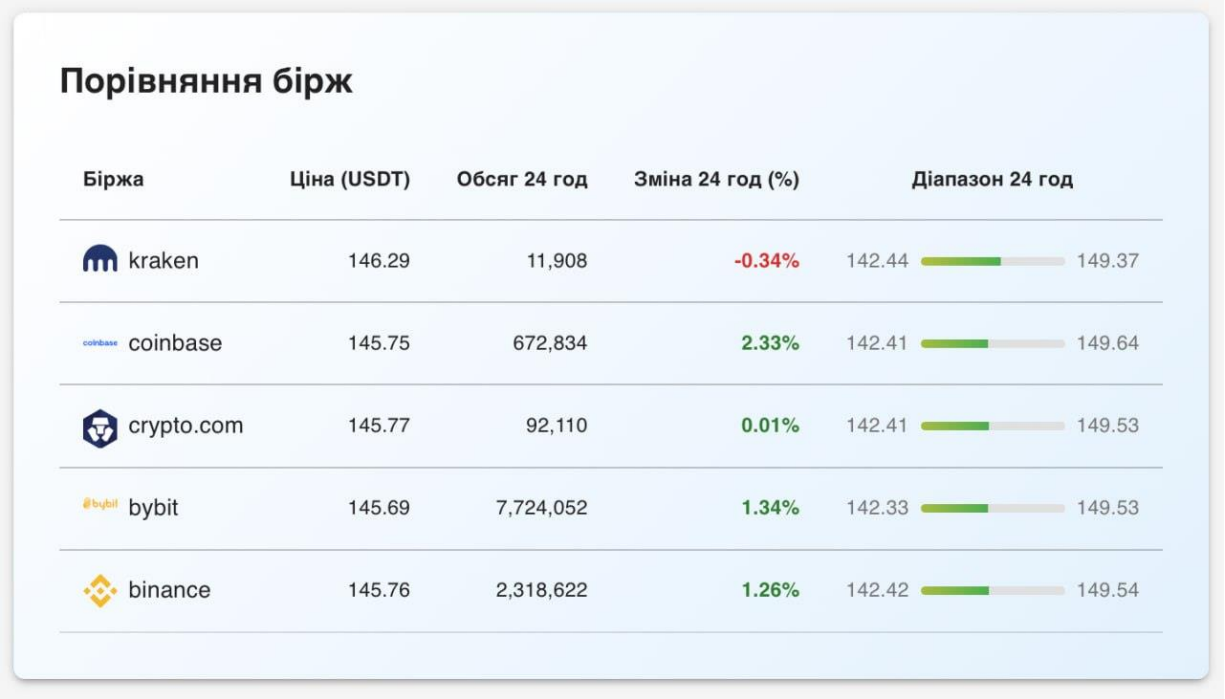


Рис. 4.10 Компонент ExchangeComparison (світла тема)

4.1.6 Навігація та налаштування

Навігація в сервісі реалізована через компонент Header, який містить меню для швидкого доступу до основних сторінок: головної сторінки, списку спостереження, профілю користувача та сторінки «Про проект». Дизайн шапки виконано з використанням Material UI, що забезпечує адаптивність і сучасний вигляд. Для управління налаштуваннями, такими як вибір мови (англійська чи українська) і теми (темна чи світла), розроблено компонент SettingsPanel. Цей компонент дозволяє користувачам персоналізувати інтерфейс, а налаштування зберігаються в локальному сховищі браузера, що забезпечує їх збереження між сеансами. Навігація між сторінками реалізована за допомогою React Router DOM 7, що забезпечує плавні переходи та швидке завантаження.

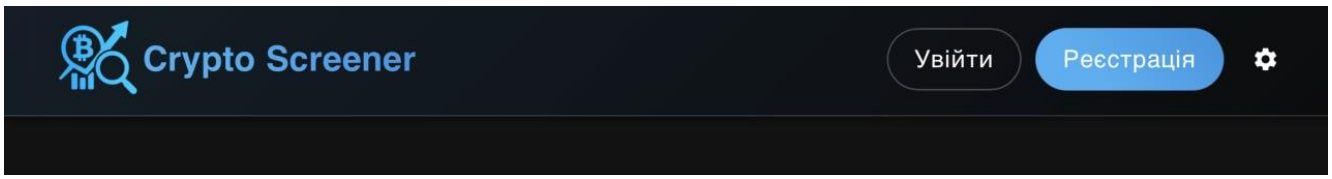


Рис. 4.11 Компонент Header (темна тема)

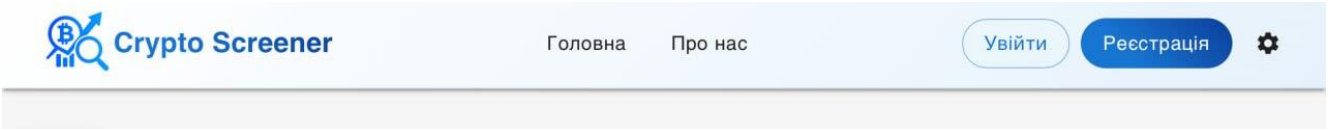


Рис. 4.12 Компонент Header (світла тема)

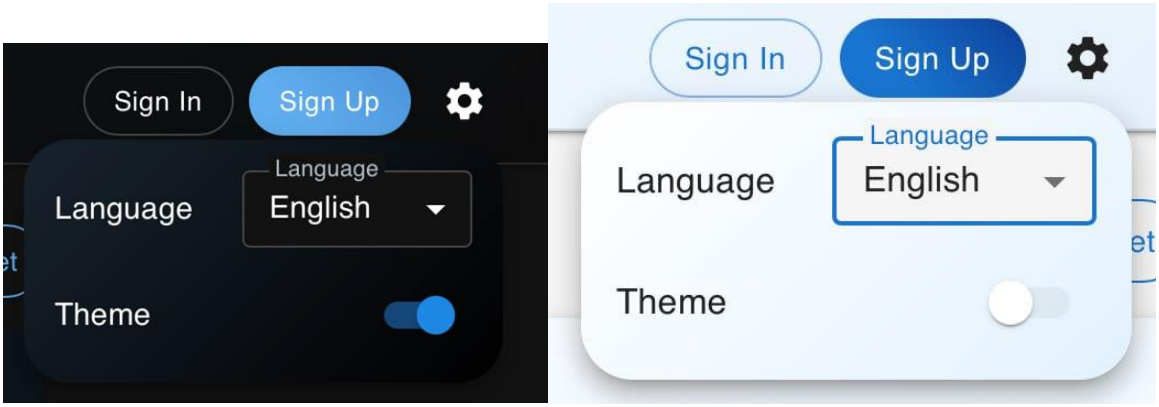


Рис. 4.13 Компонент SettingsPanel (темна та світла тема)

4.1.7 Сторінки аутентифікації та профілю

Сторінки для реєстрації (/signup), входу (/signin) і профілю користувача (/profile) були спроектовані з акцентом на простоту та безпеку. Дизайн цих сторінок виконано в єдиному стилі з іншими компонентами за допомогою Material UI, що забезпечує візуальну консистентність. Сторінка профілю дозволяє користувачам керувати списками спостереження (WatchList), додавати або видаляти торгові пари. Інтерфейс цих сторінок спроектований так, щоб бути інтуїтивно зрозумілим, із чіткими формами для введення даних і кнопками для виконання дій. Дані для аутентифікації отримуються через ендпоінти /auth/register і /auth/login, а токен авторизації зберігається локально для забезпечення безпеки.

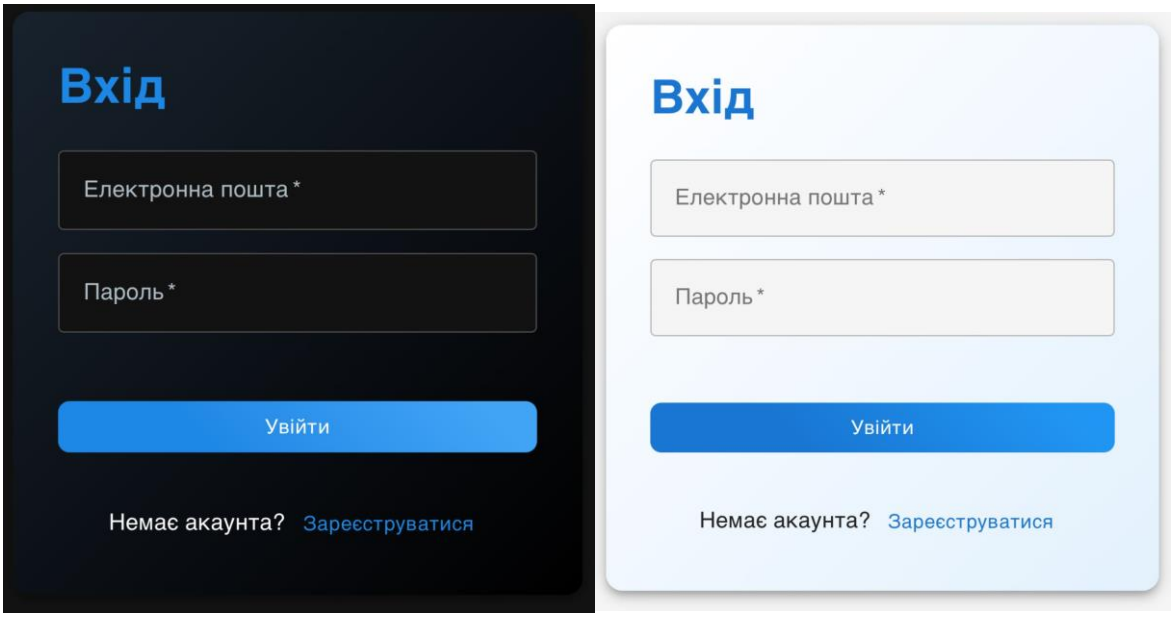


Рис. 4.14 Компонент signin

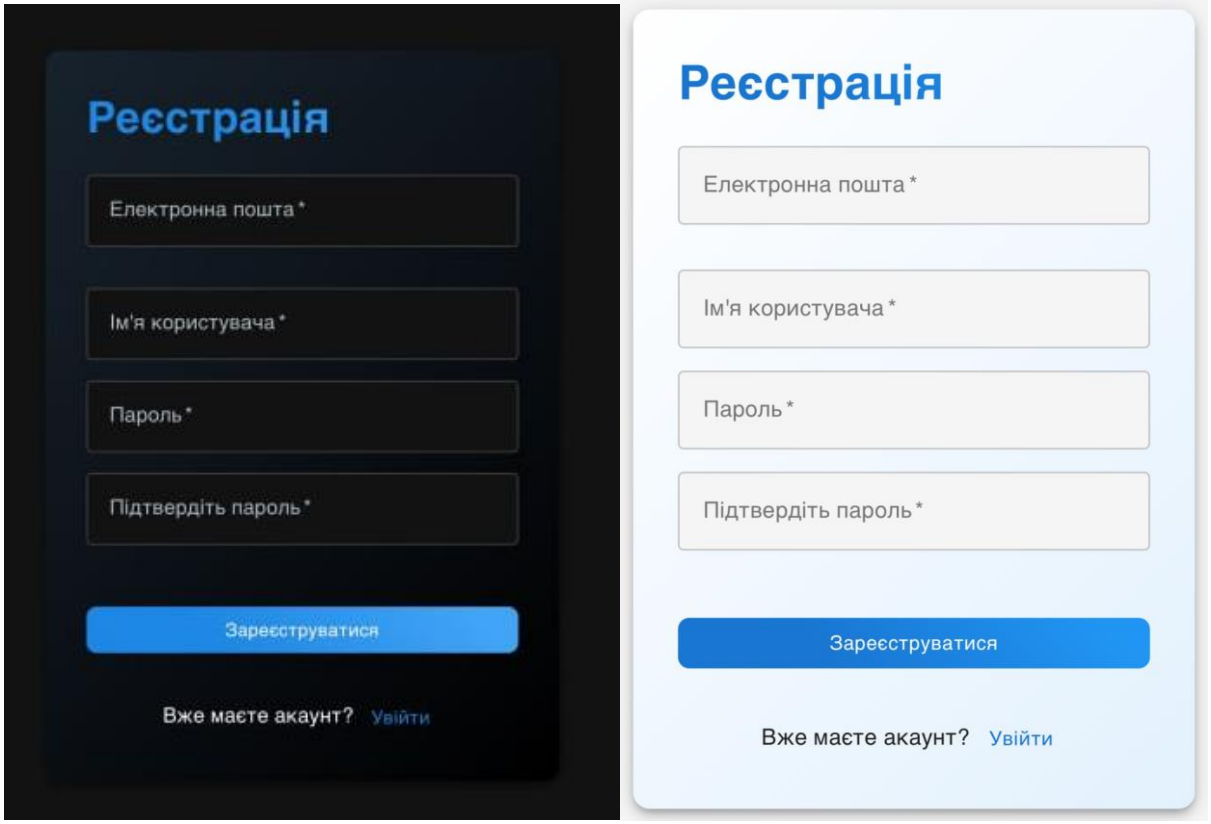


Рис. 4.15 Компонент signup

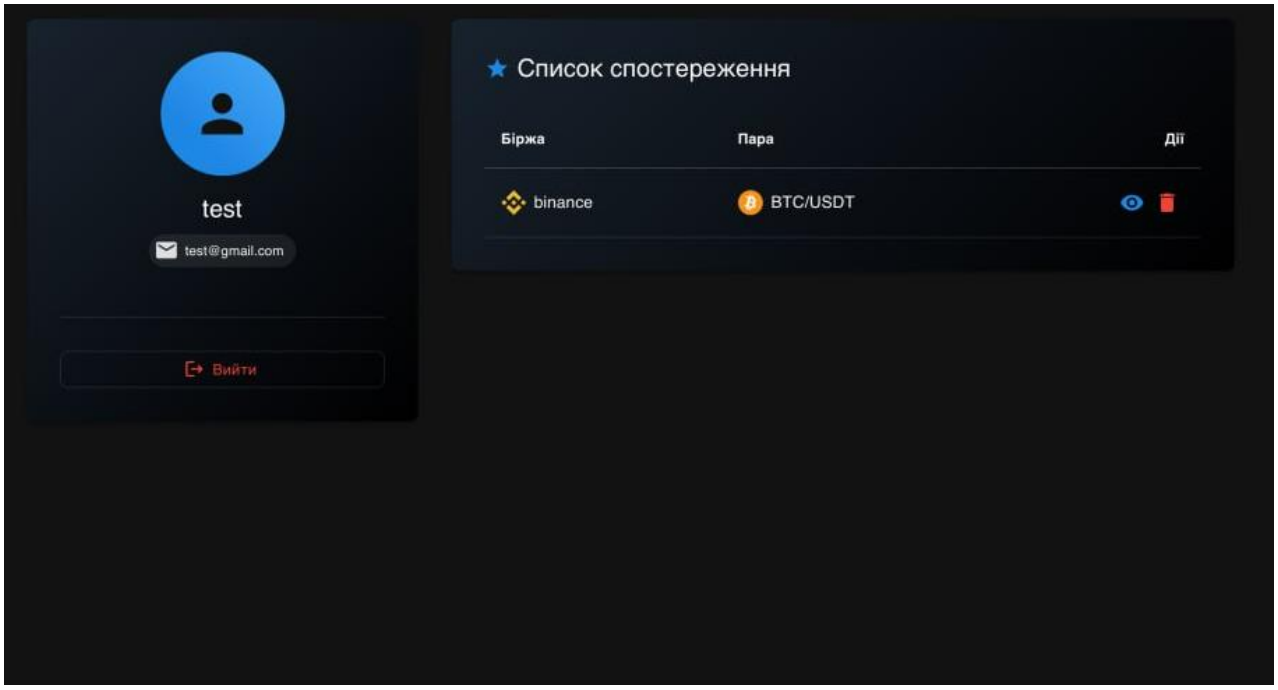


Рис. 4.16 Компоненти профілю користувача (темна тема)

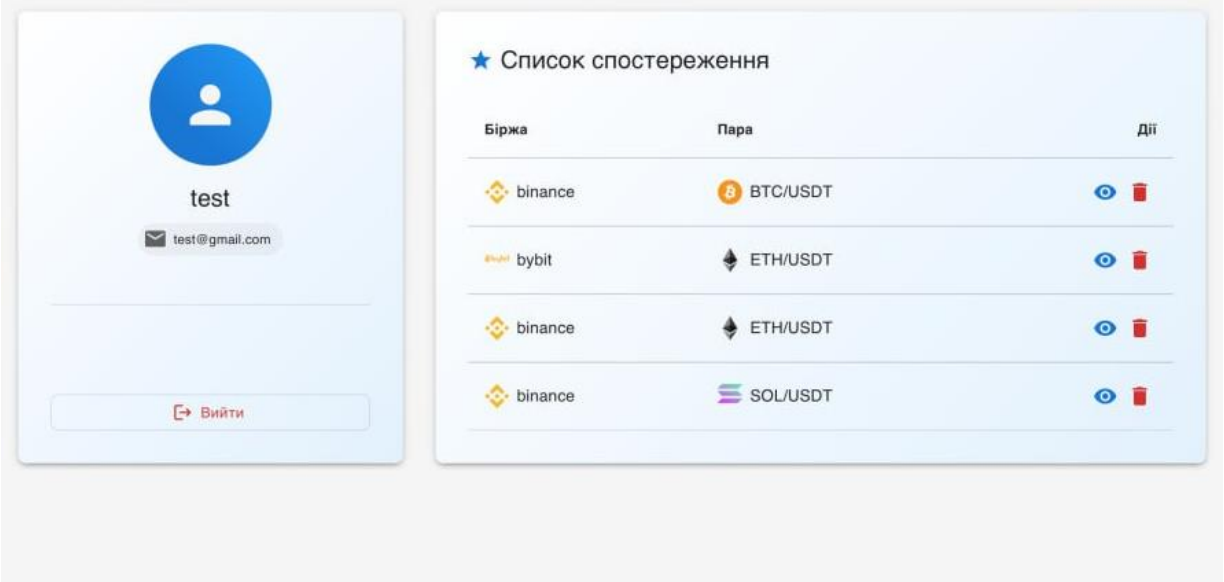


Рис. 4.17 Компоненти профілю користувача (світла тема)

4.1.8 Підсумки розробки дизайну

Дизайн інтерфейсу веб-сервісу Crypto Screener є визначальним фактором для створення комфортного та продуктивного користувацького досвіду. Використання бібліотеки Material UI 7 дозволило розробити сучасний і адаптивний інтерфейс, який відповідає вимогам трейдерів та інвесторів. Інтеграція з Recharts забезпечила

чітку та інформативну візуалізацію ринкових даних, тоді як бібліотеки Zustand і TanStack Query додали інтерфейсу динамічності та інтерактивності. Застосування архітектури Feature-Sliced Design (FSD) сприяло модульній організації компонентів, що спростило їх підтримку та подальше розширення. Підтримка мультимовності та можливість вибору між темною і світлою темами підвищили доступність сервісу для різних категорій користувачів. Використання React Router DOM забезпечило інтуїтивну та швидку навігацію між сторінками. У сукупності, продуманий дизайн інтерфейсу став основою для ефективної роботи користувачів із ринковими даними, сприяючи оперативному прийняттю торгових рішень.

4.2 Реалізація клієнтської частини

Реалізація клієнтської частини розпочалася з організації коду відповідно до архітектури Feature-Sliced Design (FSD), яка поділяє проєкт на функціональні шари для забезпечення модульності та легкості підтримки. Структура фронтенду включає п’ять основних шарів. Шар app відповідає за ініціалізацію додатку, конфігурацію провайдерів і маршрутизацію. Шар pages містить компоненти сторінок, такі як головна сторінка, сторінка деталей торгової пари чи профіль користувача. Шар features охоплює функціональні модулі, наприклад, логіку фільтрації чи аутентифікації. Шар entities визначає бізнес-сутності, такі як біржі, торгові пари та ціни, з чіткими типами для TypeScript. Нарешті, шар shared включає спільні компоненти, утиліти та клієнт для взаємодії з API. Така організація дозволила чітко розмежувати функціонал і спростила додавання нових можливостей, наприклад, підтримку нових бірж чи розширення візуалізації даних.

Основним компонентом клієнтської частини є CryptoTable, який реалізує функціонал криптовалютного скринера. Цей компонент відображає таблицю з даними про торгові пари, включаючи ціни, обсяги торгів і зміни за 24 години. Для створення таблиці використано компоненти Material UI 7, що забезпечило адаптивне розташування стовпців і сучасний дизайн. Логіка сортування та фільтрації була реалізована через бібліотеку Zustand, яка керує станом фільтрів і

параметрів сортування. Дані для таблиці отримуються через TanStack Query, яка виконує асинхронні запити до ендпоінтів RESTful API, таких як /crypto-prices/latest, і забезпечує автоматичне оновлення даних у реальному часі. Інтеграція з Axios дозволила надійно обробляти HTTP-запити та відповіді, що забезпечило стабільність відображення ринкової інформації.

Компонент FilterModal реалізує модальне вікно для налаштування параметрів фільтрації, дозволяючи користувачам вибирати біржі, торгові пари чи діапазони цін. Цей компонент побудовано з використанням Material UI для єдиного стилю та інтегровано з Zustand для управління станом фільтрів. Реалізація модального вікна передбачала асинхронне оновлення даних через TanStack Query, що дозволило динамічно змінювати вміст таблиці при застосуванні нових фільтрів. Такий підхід забезпечив гнучкість і зручність у роботі з ринковими даними.

Для детального аналізу торгових пар розроблено компонент InfoSection, доступний через сторінку /details/:exchangeId/:symbolId. Цей компонент відображає опис, історію та графіки зміни цін, створені за допомогою бібліотеки Recharts. Графіки дозволяють аналізувати ринкові тенденції за різні періоди, що є важливим для прийняття інвестиційних рішень. Реалізація компонента передбачала інтеграцію з API через TanStack Query для отримання даних із ендпоінтів, таких як /market-data, а TypeScript забезпечував типобезпеку при роботі з отриманими даними.

Компонент ExchangeComparison реалізує порівняння цін на різних біржах, що допомагає виявляти арбітражні можливості. Він відображає таблиці та графіки, побудовані за допомогою Recharts, і отримує дані через API-запити. Реалізація цього компонента передбачала використання Zustand для управління станом порівняння, що дозволило динамічно оновлювати вміст при виборі різних бірж чи періодів.

Навігація між сторінками сервісу реалізована за допомогою React Router DOM 7, що забезпечило швидкі та плавні переходи. Основні сторінки включають головну сторінку з таблицею скринера (/), сторінку деталей торгової пари (/details/:exchangeId/:symbolId), сторінки аутентифікації (/signup, /signin), профіль

користувача (/profile), список спостереження (/watchlist) і сторінку «Про проект» (/about). Кожна сторінка була реалізована як окремий компонент у шарі pages архітектури FSD, що спростило їх підтримку. Компонент Header забезпечує навігаційне меню, побудоване з використанням Material UI, що дозволяє користувачам швидко переміщатися між сторінками.

Сторінки аутентифікації та профілю реалізують персоналізований функціонал. Сторінки /signup і /signin дозволяють користувачам реєструватися та входити в систему, використовуючи API-ендпоінти /auth/register і /auth/login. Токен авторизації зберігається в локальному сховищі браузера для забезпечення безпеки. Сторінка профілю (/profile) надає доступ до списків спостереження (WatchList), де користувачі можуть додавати або видаляти торгові пари через запити до API-ендпоінта /watch-list. Реалізація цих сторінок передбачала інтеграцію з Axios і TanStack Query для обробки запитів, а Material UI забезпечила єдиний стиль форм і кнопок.

Для підвищення доступності сервісу реалізована підтримка мультимовності з двома мовами — англійською та українською. Локалізація інтерфейсу здійснювалася через бібліотеку, інтегровану з React, що дозволяло динамічно змінювати мову на основі вибору користувача. Стан мови (language) керується через Zustand і зберігається в локальному сховищі, що забезпечує збереження налаштувань між сеансами. Компонент SettingsPanel дозволяє користувачам вибирати мову та перемикатися між темною і світлою темами, які реалізовані за допомогою темізації Material UI. Ці функції підвищили комфорт користувачів і забезпечили адаптацію інтерфейсу до різних аудиторій.

Взаємодія клієнтської частини з сервером здійснювалася через RESTful API та API CoinGecko. Бібліотека Axios використовувалася для виконання HTTP-запитів до власних ендпоінтів, таких як /exchanges, /symbols, /crypto-prices і /market-data, а також до CoinGecko API для отримання додаткової інформації про криптовалюти. TanStack Query забезпечувала кешування даних і автоматичне оновлення, що дозволило оптимізувати продуктивність і зменшити кількість запитів. Реалізація API-клієнта була винесена в шар shared архітектури FSD, що

спростило повторне використання логіки запитів у різних компонентах. TypeScript забезпечував типобезпеку при роботі з відповідями API, що зменшило ймовірність помилок.

Реалізація фронтенду на основі React 19 і архітектури FSD дозволила створити модульну та масштабовану систему, яка підтримує складний функціонал, такий як фільтрація, сортування, порівняння цін і візуалізація даних. Використання Material UI забезпечило сучасний і адаптивний дизайн, тоді як Zustand і TanStack Query додали динамічності та продуктивності. Інтеграція з API через Axios і підтримка мультимовності підвищили функціональність і доступність сервісу. Ця реалізація відповідає потребам трейдерів та інвесторів, надаючи зручний інструмент для аналізу криптовалютного ринку в реальному часі.

4.3 Реалізація серверних частин

Обидва серверні компоненти — Crypto Price Parser Backend і RESTful API — побудовані за принципами Domain-Driven Design, що передбачає чітке розділення коду на шари з різними обов’язками. Основним шаром є Domain Layer, який містить бізнес-логіку та доменні сутності, такі як біржі, торгові пари, ціни та ринкові дані. Цей шар визначає ядро системи, незалежне від зовнішніх технологій, що сприяє гнучкості та легкості тестування. Application Layer відповідає за координацію бізнес-логіки через сценарії використання (use cases), які оркеструють роботу доменних сутностей. Infrastructure Layer реалізує взаємодію з зовнішніми системами, такими як MongoDB і API бірж, а також включає конфігурацію та логування. Нарешті, Presentation Layer забезпечує точки входу до системи, які відрізняються для кожного компонента: для API #1 це планувальники (schedulers) для періодичного збору даних, а для API #2 — HTTP-контролери для обробки запитів.

Така архітектура дозволила досягти високої модульності та ізоляції компонентів. Наприклад, доменні сутності, визначені в шарі Domain, використовуються в обох компонентах, що забезпечило консистентність даних і

спростило їх повторне використання. TypeScript відігравав ключову роль у підтримці типобезпеки, дозволяючи чітко визначати структури даних і інтерфейси для всіх шарів. Використання Node.js забезпечило асинхронну обробку запитів, що було критично важливим для роботи з великими обсягами даних із бірж у реальному часі.

Серверна частина оперує кількома ключовими доменними сутностями, які є основою для роботи з ринковими даними. Сутність Exchange представляє криптовалютну біржу, таку як Binance чи Coinbase, і містить інформацію про назву, URL API та логотип. Сутність Symbol описує торгову пару, наприклад BTC/USDT, із даними про базову та котирувальну валюти, описом і пов’язаними біржами. Сутність CryptoPrice зберігає цінові дані для конкретної пари на конкретній біржі, включаючи ціну, обсяги торгів за різні періоди та часову мітку. Сутність MarketData агрегує ринкову статистику, таку як зміна ціни за 24 години та діапазон цін. Для RESTful API додатково реалізовані сутності User для управління користувачами та WatchList для створення списків спостереження.

Ці сутності були реалізовані як TypeScript-класи в шарі Domain, із чітко визначеними властивостями та методами. Наприклад, клас CryptoPrice включає методи для валідації даних перед збереженням, що забезпечує їх коректність. Для роботи з MongoDB створено відповідні моделі через Mongoose ODM, такі як ExchangeModel, SymbolModel, CryptoPriceModel, MarketDataModel, UserModel і WatchListModel. Mongoose дозволив визначити схеми з обов’язковими полями, типами даних і зв’язками, наприклад, через ObjectId для пов’язування цін із біржами та торговими парами. Реалізація моделей передбачала використання індексів для прискорення запитів, що було особливо важливим для обробки великих обсягів цінових даних.

Crypto Price Parser Backend (API #1) відповідає за періодичний збір даних із криптовалютних бірж і їх збереження в MongoDB. Основним сценарієм використання є FetchAndStoreCryptoPricesUseCase, який координує процес отримання та обробки даних. На старті додаток ініціалізує підключення до MongoDB і створює репозиторії для роботи з сутностями (Exchange, Symbol,

CryptoPrice, MarketData). Репозиторії, реалізовані в шарі Infrastructure, абстрагують доступ до бази даних, використовуючи Mongoose для виконання CRUD-операцій.

Логіка збору даних починається з отримання списку всіх бірж і торгових пар із бази даних. Для кожної комбінації біржі та пари виконується запит до API біржі через бібліотеку CCXT, яка забезпечує уніфікований інтерфейс для взаємодії з різними платформами. CCXT дозволяє отримувати дані про поточну ціну, обсяги торгів за різні періоди (1 година, 24 години, 7 днів, 30 днів) і ринкову статистику, таку як зміна ціни за 24 години. Отримані дані обробляються для створення екземплярів доменних сутностей, які потім зберігаються в MongoDB. Процес виконується асинхронно з використанням конструкцій async/await, що забезпечує ефективну обробку великої кількості запитів. Періодичність збору даних налаштовується (за замовчуванням — кожну хвилину) через планувальник, реалізований у шарі Presentation.

Особливістю API #1 є його фокус на автономній роботі без прямого доступу ззовні, що забезпечує ізоляцію збору даних від інших компонентів. Використання TypeScript дозволило чітко визначити типи для даних, отриманих із CCXT, що зменшило ймовірність помилок при обробці різноманітних форматів відповідей від бірж. Логування, реалізоване в шарі Infrastructure, забезпечувало відстеження помилок і моніторинг роботи системи, що полегшило налагодження.

RESTful API (#2) забезпечує доступ до даних, зібраних бэкендом, для клієнтської частини, надаючи набір ендпоінтів для управління біржами, торговими парами, цінами, ринковими даними, користувачами та списками спостереження. Реалізація API також базується на DDD, із тими ж доменними сутностями, що й у API #1, але доповненими специфічними для користувачів функціями. Основні сценарії використання включають отримання даних, створення нових записів і управління списками спостереження, реалізовані через відповідні use cases у шарі Application.

Ендпоінти API поділяються на кілька груп. Група /exchanges дозволяє отримувати список бірж, переглядати деталі за ID, створювати, оновлювати чи видаляти біржі. Група /symbols надає аналогічний функціонал для торгових пар.

Ендпоінти /crypto-prices і /crypto-prices/latest дозволяють отримувати всі ціни або лише останні дані, тоді як /market-data і /market-data/pair/:pair надають ринкову статистику. Для аутентифікації реалізовані ендпоінти /auth/register і /auth/login, які забезпечують реєстрацію та вхід користувачів. Група /watch-list дозволяє керувати списками спостереження, додаючи чи видаляючи торгові пари. Кожен ендпоінт обробляється контролером у шарі Presentation, який викликає відповідний use case для виконання бізнес-логіки.

Репозиторії для API #2, реалізовані через Mongoose, забезпечують доступ до тих самих моделей MongoDB, що й у API #1, але додатково включають UserModel і WatchListModel. Наприклад, UserModel містить поля для імені, електронної пошти та пароля (з хешуванням для безпеки), тоді як WatchListModel пов’язує користувачів із торговими парами через ObjectId. Використання Mongoose дозволило реалізувати складні запити, такі як агрегація ринкових даних за парою, що використовується в ендпоінті /market-data/pair/:pair. Middleware у шарі Presentation забезпечує централізовану обробку помилок і аутентифікацію, використовуючи токени для захисту ендпоінтів, таких як /watch-list.

Унікальною особливістю API #2 є його орієнтація на взаємодію з фронтендом через HTTP-запити. Інтеграція з Axios і TanStack Query на клієнтській стороні забезпечує ефективне кешування та оновлення даних, тоді як TypeScript гарантує типобезпеку для відповідей API. Логування та конфігурація, реалізовані в шарі Infrastructure, дозволяють відстежувати запити та забезпечувати стабільність роботи API.

Обидва компоненти серверної частини активно використовують MongoDB як основну базу даних. Підключення до MongoDB ініціалізується на старті додатків, із конфігурацією для підтримки реплікації та шардування, що забезпечує масштабованість. Mongoose ODM спрощує роботу з базою, надаючи об’єктно-орієнтований інтерфейс для CRUD-операцій. Наприклад, у API #1 дані, отримані через CCXT, перетворюються в документи MongoDB, тоді як у API #2 ці документи використовуються для відповідей на запити фронтенду. Індексація полів, таких як

symbolId і exchangeId у CryptoPriceModel, прискорює запити, що є важливим для реального часу.

Бібліотека CCXT забезпечує доступ до API бірж. Її реалізація передбачала створення сервісу ExchangeClient у шарі Infrastructure, який виконує запити до бірж і обробляє відповіді. У API #2 CCXT використовується опосередковано, оскільки дані, зібрані бэкендом, уже доступні через MongoDB. Використання TypeScript дозволило чітко типізувати відповіді CCXT, що зменшило ризик помилок при обробці даних.

Безпека серверної частини забезпечувалася кількома заходами. Для API #2 аутентифікація користувачів реалізована через токени, які перевіряються middleware перед доступом до захищених ендпоінтів. Паролі користувачів хешуються перед збереженням у UserModel, що підвищує захист даних. У API #1 безпека забезпечувалася ізоляцією від зовнішнього доступу, що мінімізувало ризики атак. Асинхронна модель Node.js і оптимізовані запити до MongoDB сприяли високій продуктивності, дозволяючи обробляти великі обсяги даних без значних затримок.

Реалізація серверної частини веб-сервісу Crypto Screener стала основою для збору, обробки та надання ринкових даних, забезпечуючи функціонування як автономного бэкенду, так і API для клієнтської частини.

4.4 Результат релізації веб-сервісу

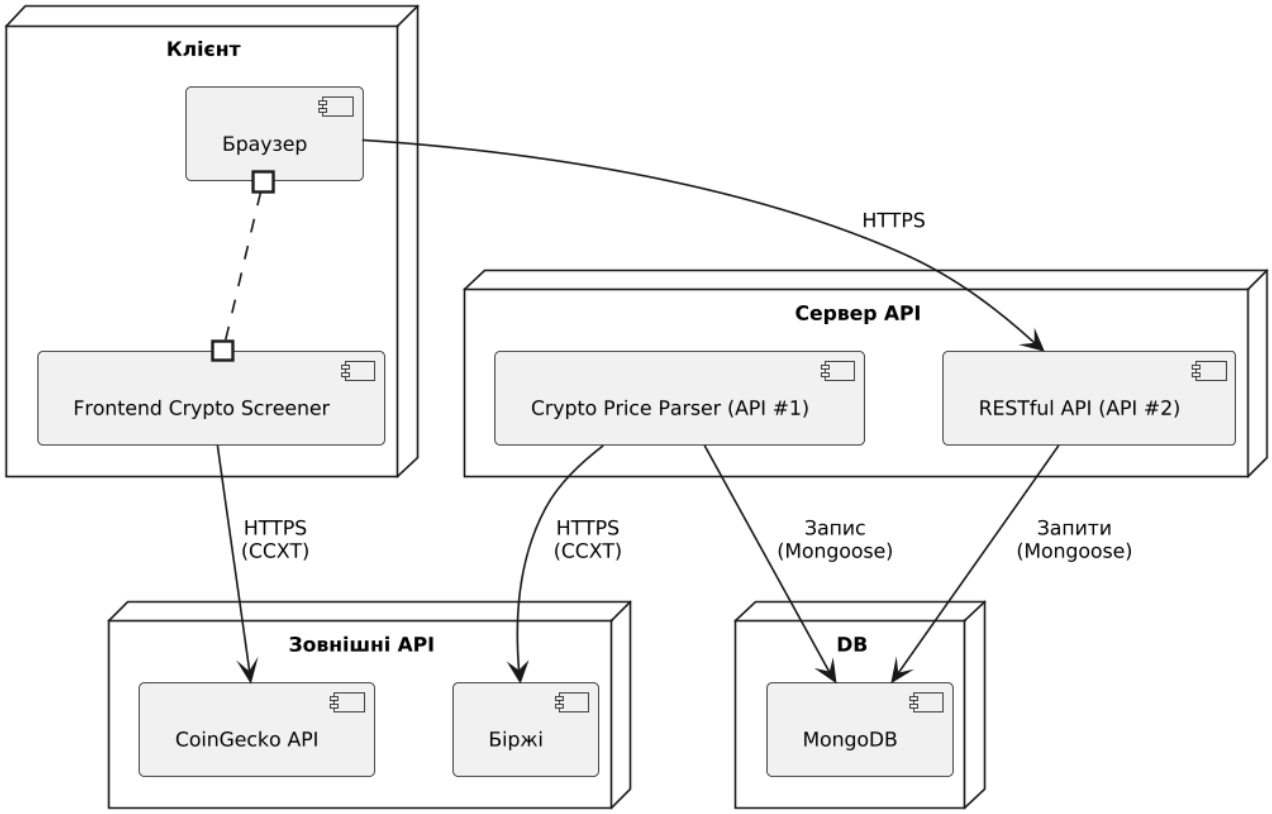


Рис. 4.18 Діаграма розгортання реалізованого веб-сервісу

Реалізація веб-сервісу Crypto Screener завершилася створенням ефективної системи для моніторингу криптовалютного ринку в реальному часі, що задовольняє потреби новачків, трейдерів, інвесторів і аналітиків. Система забезпечує збір даних, фільтрацію, візуалізацію та персоналізацію, поєднуючи модульність і масштабованість. Діаграма розгортання відображає фізичну структуру сервісу, демонструючи чітку взаємодію компонентів.

Клієнтський вузол містить браузер із Frontend Crypto Screener, реалізованим як SPA на React 19. Фронтенд використовує Material UI 7 для адаптивного дизайну, Recharts для графіків і Zustand із TanStack Query для керування станом і запитами. Axios забезпечує HTTP-запити до RESTful API (API #2), а в окремих випадках фронтенд звертається до CoinGecko API через CCXT.

Серверний вузол об’єднує Crypto Price Parser Backend (API #1) і RESTful API (API #2). API #1 періодично збирає дані з бірж (Binance, Coinbase, Kraken) і CoinGecko через CCXT, використовуючи Node.js і TypeScript, і записує їх у MongoDB через Mongoose ODM. API #2 обробляє запити фронтенду через ендпоінти (/crypto-prices, /market-data, /auth, /watch-list), отримуючи дані з MongoDB. Окремий вузол MongoDB зберігає сутності, такі як біржі, торгові пари та списки спостереження, підтримуючи масштабованість.

Архітектура базується на DDD для серверної частини та FSD для фронтенду, що спрощує підтримку модулів, таких як CryptoTable і ExchangeComparison. Безпека забезпечена через CORS, токени аутентифікації та хешування паролів. Система реалізована з використанням сучасних технологій, що гарантують швидкість і надійність. Завантаження проєкту на GitHub із документацією в README.md підтверджує готовність сервісу до використання та подальшого розвитку.

4.5 Тестування

Тестування веб-сервісу Crypto Screener було проведено для забезпечення надійності, коректності роботи та відповідності функціональних вимог. Для цього використано бібліотеку Jest, сучасний інструмент тестування, який широко застосовується для перевірки JavaScript- і TypeScript-додатків. Jest надає зручний інтерфейс для створення модульних, інтеграційних і функціональних тестів, дозволяючи перевіряти окремі компоненти, їх взаємодію та поведінку системи в цілому. Завдяки підтримці мокування (імітації зовнішніх залежностей) і знімків стану Jest забезпечує ефективне тестування складних сценаріїв, таких як обробка API-запитів.

Тестування охоплювало всі ендпоінти RESTful API, включаючи /exchanges, /symbols, /crypto-prices, /market-data, /auth і /watch-list. Для кожного ендпоінта створено тести, які перевіряють коректність повернення даних, відповідність формату відповідей і обробку помилок. Наприклад, тести для /crypto-prices/latest

перевіряли, чи повертаються актуальні ціни у правильному JSON-форматі, а для /auth/login — чи коректно обробляються невалідні облікові дані. Тести також підтвердили цілісність даних, отриманих із MongoDB, забезпечуючи їх відповідність доменним сутностям, таким як біржі чи торгові пари.

Застосування Jest дозволило виявити та виправити потенційні помилки на ранніх етапах, підвищивши стабільність серверної частини. Проведене тестування забезпечило надійну роботу веб-сервісу, гарантуючи коректне надання ринкових даних для клієнтської частини та відповідність потребам користувачів.

4.5 Завантаження проєкту на GitHub

Для забезпечення доступності та управління кодом веб-сервісу Crypto Screener проєкт було завантажено на платформу GitHub, яка є стандартом для зберігання, версіонування та спільної розробки програмного забезпечення. Репозиторій організовано з чіткою структурою, що відображає модульність проєкту, включаючи окремі директорії для бэкенду, RESTful API та фронтенду. Використання системи контролю версій Git дозволило відстежувати зміни, створювати гілки для нових функцій і забезпечувати стабільність коду. Файл README.md містить опис проєкту, інструкції зі встановлення та запуску, що полегшує розгортання для інших розробників. Завантаження на GitHub сприяло ефективному управлінню кодом і забезпечило можливість подальшого розширення функціоналу сервісу.

ВИСНОВКИ

Результатами виконаної роботи стала розробка веб-сервісу Crypto Screener, який забезпечує ефективний моніторинг і аналіз статистики криптовалют у реальному часі, надаючи трейдерам та інвесторам зручний інструмент для роботи з ринковими даними. Сервіс поєднує автоматизований збір даних із криптовалютних бірж, гнучкі можливості фільтрації та сортування, інтерактивну візуалізацію, а також персоналізовані функції, такі як списки спостереження. Реалізація базується на сучасних технологіях, включаючи TypeScript, Node.js, React 19, MongoDB і бібліотеку CCXT, що забезпечило високу продуктивність, масштабованість і адаптивність системи. Завдяки модульній архітектурі, побудованій на принципах Domain-Driven Design (DDD) для серверної частини та Feature-Sliced Design (FSD) для клієнтської, сервіс відповідає вимогам сучасного криптовалютного ринку.

Було виконано наступні задачі:

– Розроблено механізм паралельного опитування API бірж за допомогою бібліотеки CCXT, що дозволяє одночасно отримувати актуальні ціни, обсяги торгів і ринкову статистику з різних джерел, таких як Binance, Coinbase і Kraken, із подальшим збереженням даних у MongoDB.
– Реалізовано функціонал фільтрації та сортування торгових пар, який дає змогу користувачам налаштовувати перегляд за ціною, обсягом торгів, зміною ціни за 24 години, а також комбінувати фільтри за біржами, діапазонами цін і типами пар, що підвищує гнучкість аналізу.
– Створено інтерактивну графічну візуалізацію ринкових даних із використанням бібліотеки Recharts, яка забезпечує відображення лінійних графіків зміни цін.
– Розроблено модуль персоналізованого профілю, що включає реєстрацію, авторизацію через ендпоінти /auth/register і /auth/login, а також функціонал

створення й управління списками спостереження, які дозволяють користувачам відстежувати вибрані торгові пари.

- Налаштовано автоматичне оновлення даних із частотою раз на хвилину через планувальник у бэкенді Crypto Price Parser, що забезпечує актуальність ринкової інформації для фронтенду.
- Забезпечено стійкість до збоїв API бірж шляхом реалізації обробки помилок і використання резервних джерел даних, що гарантує безперервну роботу сервісу навіть у разі тимчасової недоступності окремих платформ.
- Реалізовано масштабованість системи завдяки модульній архітектурі DDD і підтримці CCXT, що дозволяє легко інтегрувати нові біржі чи додаткові API без значних змін у коді.
- Підвищено безпеку сервісу через налаштування політики CORS, яка обмежує доступ до API лише з довірених доменів, а також використання токенів для аутентифікації й хешування паролів у базі даних.
- Забезпечено двомовність інтерфейсу з підтримкою англійської та української мов, із можливістю перемикання через компонент SettingsPanel і збереженням налаштувань у локальному сховищі браузера.
- Розроблено адаптивний дизайн із використанням Material UI 7, що гарантує коректне відображення інтерфейсу на десктопних і мобільних пристроях, підвищуючи зручність використання для різних категорій користувачів.
- Проведено тестування серверної частини за допомогою Jest, що охопило всі ендпоінти RESTful API, включаючи перевірку коректності повернення даних і обробки помилок, що забезпечило надійність системи.
- Завантажено проєкт на GitHub із чіткою структурою репозиторію, яка включає директорії для бэкенду, API та фронтенду, а також документацію в README.md, що полегшує розгортання та подальший розвиток.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Маліков В.Є., Залива В.В. Дослідження впливу популяризації криптовалют на повсякденне життя в цифрову епоху та актуальність розробки інструментів моніторингу ринкової ситуації. Матеріали IV Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2025, ДУІКТ, м. Київ. К.: ДУІКТ, Україна, С. 584-589.
2. Маліков В.Є., Залива В.В. Криптоскринер як засіб моніторингу ринку криптовалют: розробка та переваги інноваційного програмного забезпечення. Матеріали V Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2025,ДУІКТ, м. Київ. К.: ДУІКТ, Україна, (подано до друку).

ПЕРЕЛІК ПОСИЛАНЬ

1. CoinMarketCap. Cryptocurrency Prices, Charts And Market Capitalizations. URL: https://coinmarketcap.com/
2. CoinMarketCap: The Ultimate Tool for Crypto Market Research. ClickDo. URL: https://crypto.clickdo.co.uk/coinmarketcap/
3. CoinGecko. Cryptocurrency Prices, Charts, and Crypto Market Cap. URL: https://www.coingecko.com/
4. TradingView. Cryptocurrency Market Overview: Bitcoin and Altcoin Prices. URL: https://www.tradingview.com/markets/cryptocurrencies/
5. Crypto Market Data Providers: Ensuring Transparency and Accuracy. Coinmetro. URL: https://www.coinmetro.com/learning-lab/crypto-market-data-providers
6. Best Crypto Analysis Tools For Every Investor. CoinGecko. URL: https://www.coingecko.com/learn/9-best-crypto-analysis-tools-for-every-investor
7. MongoDB Documentation. MongoDB Atlas for NoSQL Databases. URL: https://www.mongodb.com/docs/
8. CCXT Documentation. Unified Cryptocurrency Exchange APIs. URL: https://ccxt.readthedocs.io/en/latest/
9. React 19 Documentation. React Official Website. URL: https://react.dev/
10. Material UI 7 Documentation. MUI: The React Component Library. URL: https://mui.com/
11. TanStack Query Documentation. React Query for Data Fetching. URL: https://tanstack.com/query/
12. Zustand Documentation. A Small, Fast and Scalable State-Management Solution. URL: https://zustand-demo.pmnd.rs/
13. Recharts Documentation. A Composable Charting Library Built on React Components. URL: https://recharts.org/
14. Vite Documentation. Next Generation Frontend Tooling. URL: https://vitejs.dev/
15. Domain-Driven Design: Tackling Complexity in the Heart of Software / E. Evans.

Addison-Wesley, 2003. 560 p.

16. Clean Architecture: A Craftsman’s Guide to Software Structure and Design / R. C. Martin. Prentice Hall, 2017. 432 p.
17. Feature-Sliced Design Documentation. An Architectural Methodology for Frontend Projects. URL: https://feature-sliced.github.io/documentation/
18. DOU. Що таке Domain-Driven Design та на якому етапі варто його впроваджувати в продукт. URL: https://dou.ua/forums/topic/39874/
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. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Розробка web-сервісу для перегляду статистики криптовалют з криптобірж

Виконав студент 4 курсу
групи ПД-42
Маліков Вадим Євгенович

Керівник роботи
доктор філософії, старший викладач кафедри ІПЗ Залива Віталій Вікторович

Київ - 2025

МЕТА, ОБ'ЄКТА ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення процесу аналізу криптовалют за рахунок використання web-сервісу.

Об'єкт дослідження: процес аналізу криптовалютної статистики та відображення динаміки ринку.

Предмет дослідження: застосування засобів та технологій створення веб-застосунку для моніторингу та порівняння криптовалют у реальному часі.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести огляд та аналіз існуючих сервісів для моніторингу криптовалют, визначити їхні функціональні можливості, інтерфейс і обмеження.
2. Описати функціональні та нефункціональні вимоги до розроблюваного програмного забезпечення.
3. Виконати огляд технологічного стеку та інструментів для реалізації клієнт-серверного веб-застосунку моніторингу криптовалют, оцінити їхні переваги й недоліки.
4. Спроектувати архітектуру веб-застосунку для перегляду, фільтрації та порівняння криптовалют із різних бірж із урахуванням масштабованості та відмовостійкості.
5. Розробити клієнт-серверний веб-застосунок: реалізувати збір даних через API, реалізацію фільтрів, графічну візуалізацію та персоналізований профіль користувача.
6. Протестувати застосунок.

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Отримання даних про криптовалюти з кількох бірж через API.
2. Відображення актуальної інформації з бірж в реальному часі.
3. Фільтрація за обраними криптовалютами, біржами, діапазонами цін.
4. Фільтрація монет за параметрами: ціна, обсяг торгів, зміна за 24h.
5. Побудова графічного відображення зміни ціни в діапазоні 24h.
6. Реєстрація та можливість налаштування індивідуального списку відслідковування в профілі.

Нефункціональні вимоги:

1. Висока продуктивність — оновлення даних не рідше, ніж кожну хвилину.
2. Масштабованість — можливість додавання нових джерел API.
3. Надійність — відсутність збоїв при недоступності окремих API.
4. Контроль CORS — дозволяти запити до API лише з довірених доменів.
5. Локалізація — підтримка декількох мов (укр/англ).
6. Адаптивність під мобільні та десктопні пристрої.

АНАЛІЗ АНАЛОГІВ

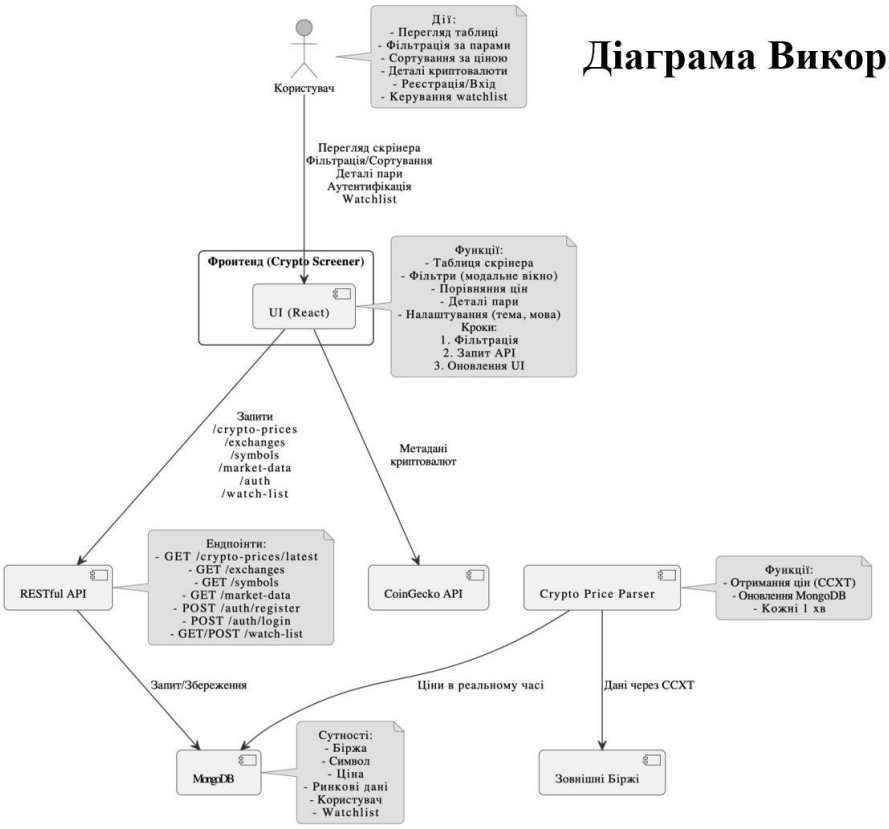
Технічні характеристики	CoinMarketCap	CoinGecko	TradingView	CryptoScreener
Підтримка API	+	+	+	+
Актуалізація в реальному часі	+	+	–	+
Візуалізація графіків	+	–	+	+
Порівняння цін на біржах	–	–	–	+
Фільтри за параметрами	+	–	–	+

5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

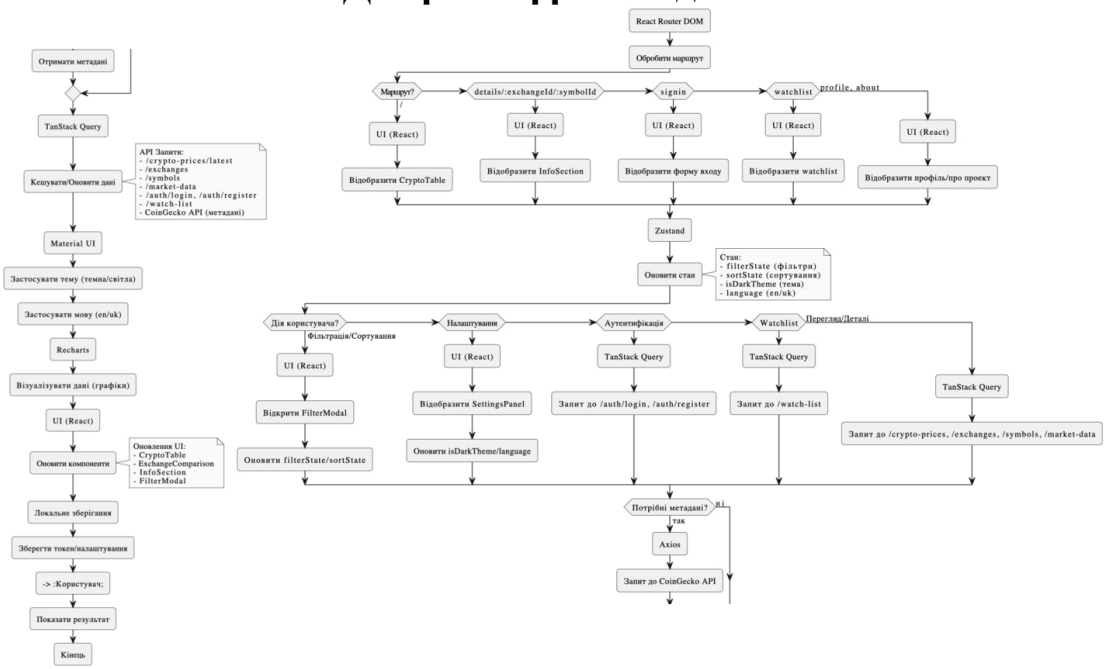
6

Діаграма Використання



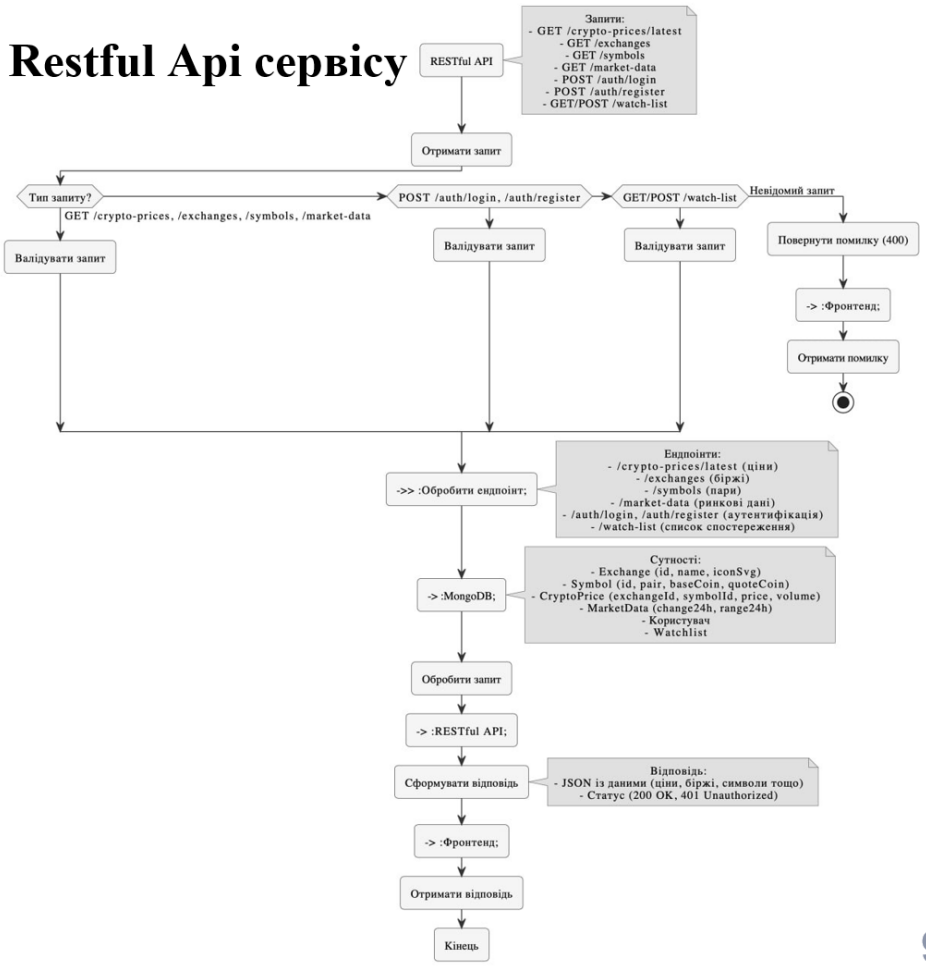
7

Flow-Діаграма фронтенд частини

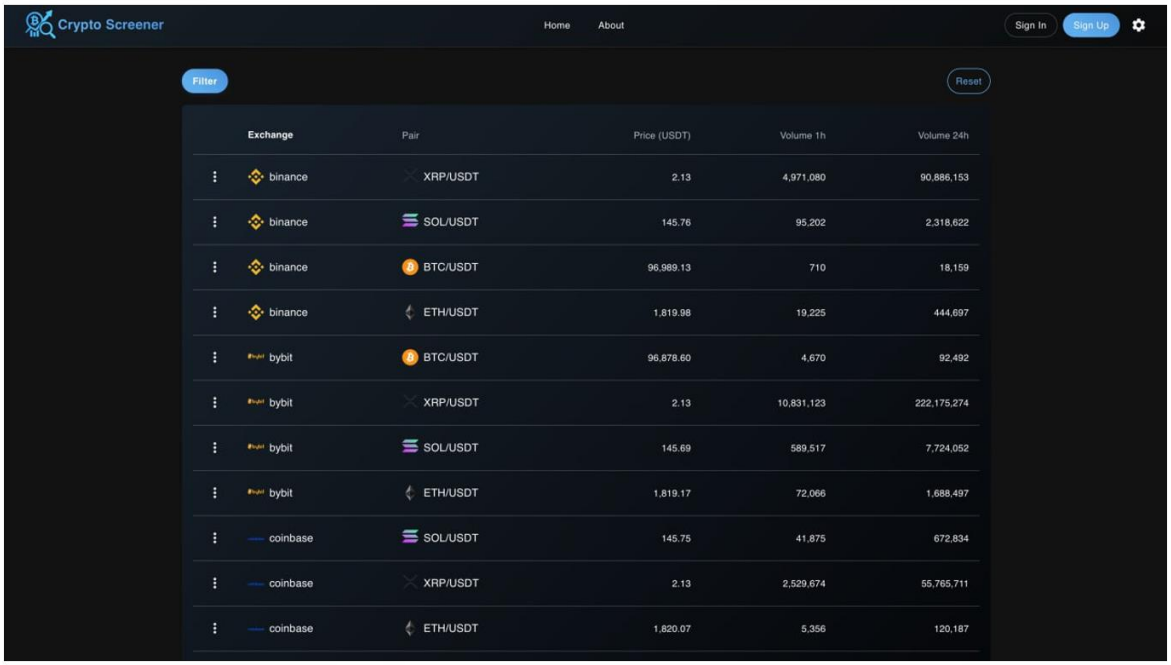


8

FLOW-Діаграма Restful Api сервісу

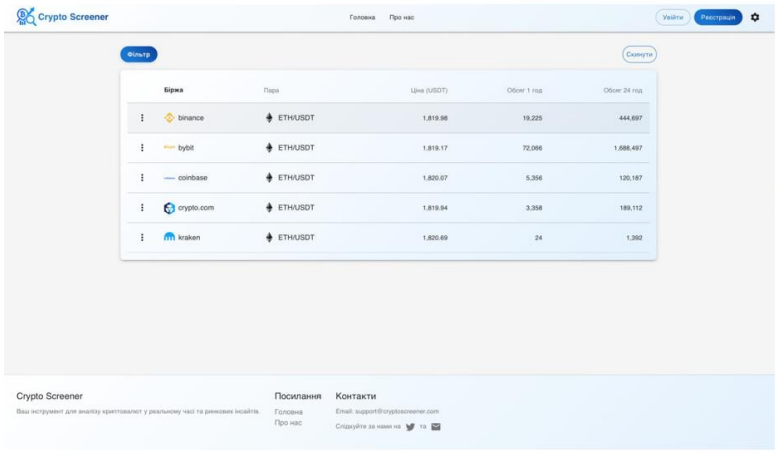


ЕКРАННІ ФОРМИ



10

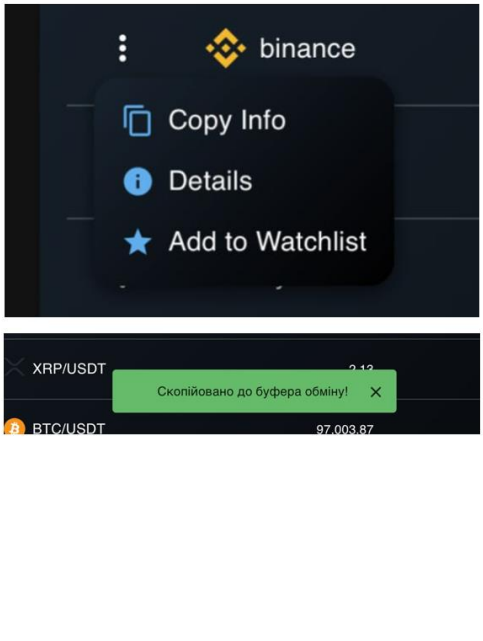
ЕКРАННІ ФОРМИ

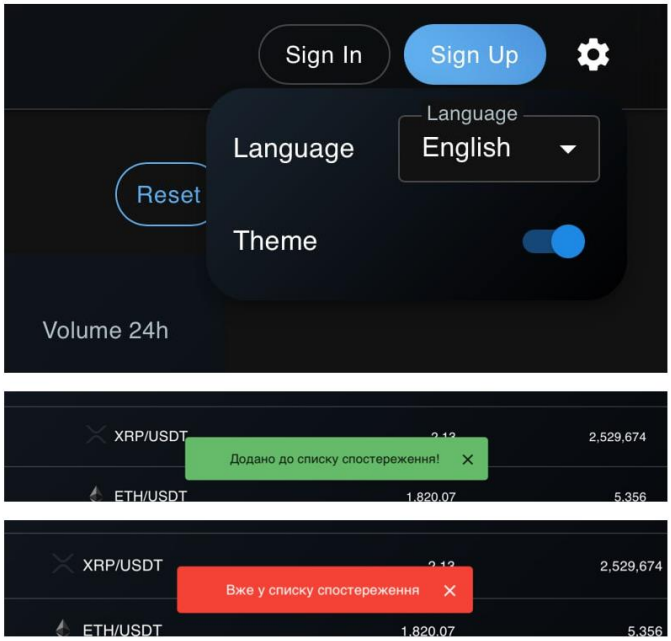


↓ Ціна (USDT)	↑ Обсяг 24 год
97,003.87	82
96,989.13	1,392
96,987.20	6,001
96,936.46	7,482
96,878.60	11,908
1,820.69	18,159
1,820.07	92,110
1,819.98	
1,819.94	
1,819.17	

11

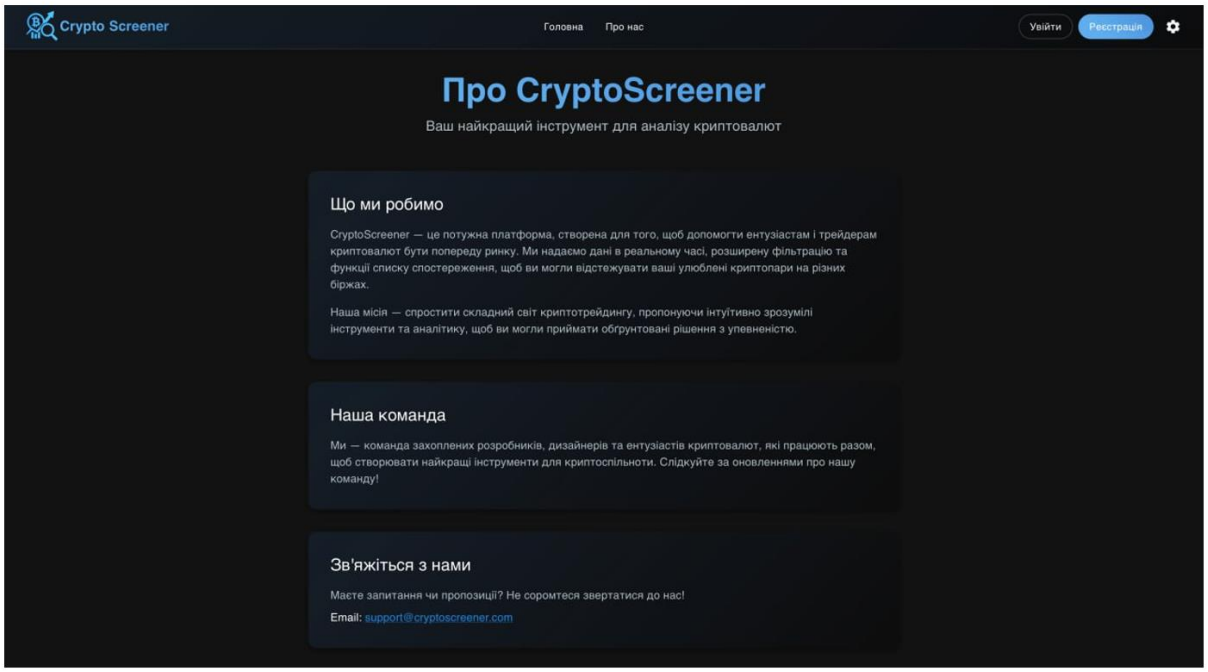
ЕКРАННІ ФОРМИ





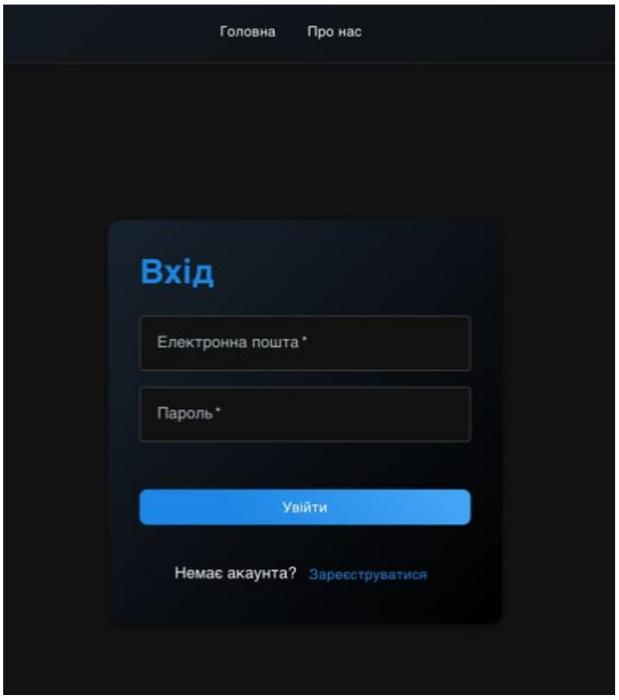
12

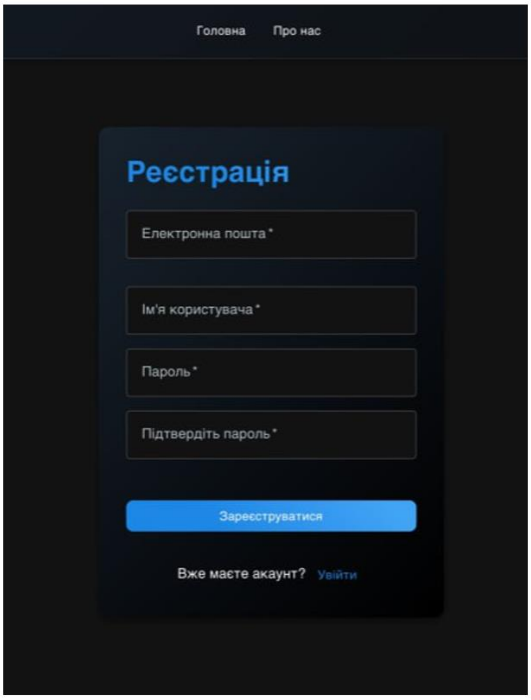
ЕКРАННІ ФОРМИ



13

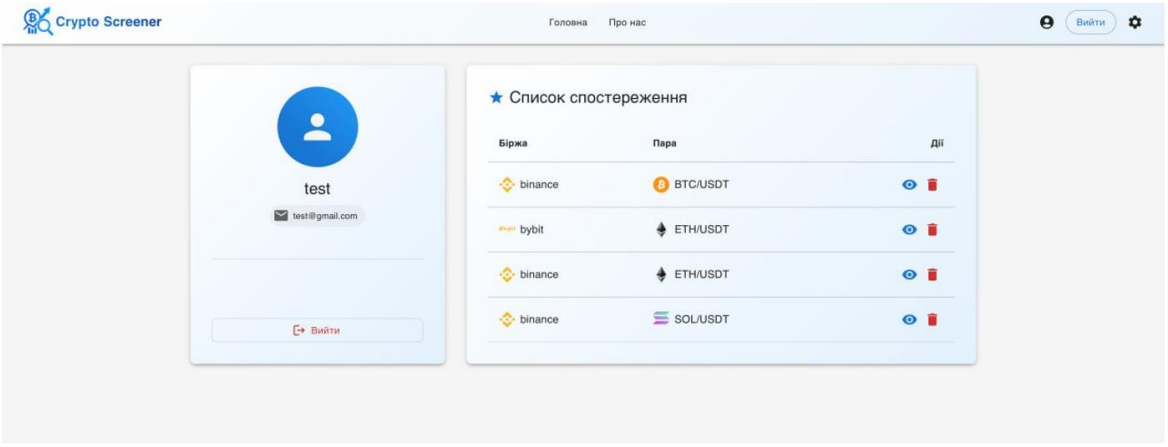
ЕКРАННІ ФОРМИ

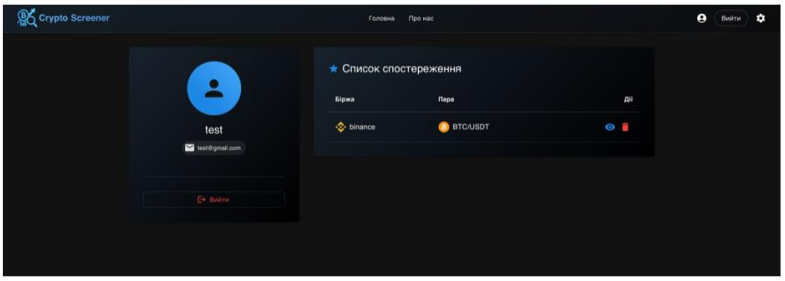




14

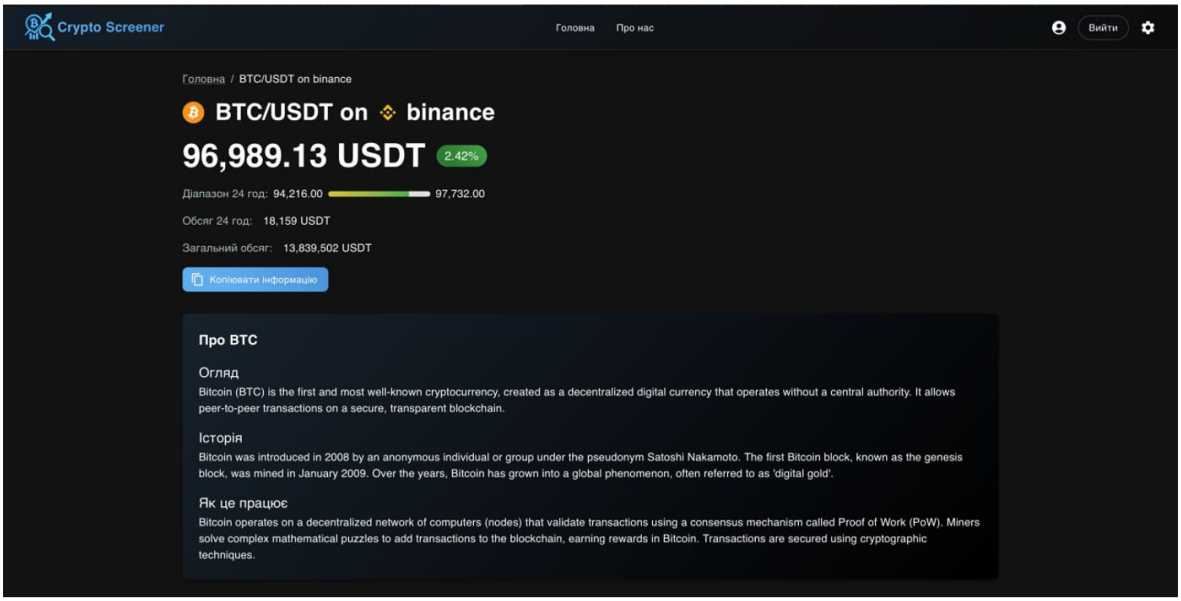
ЕКРАННІ ФОРМИ





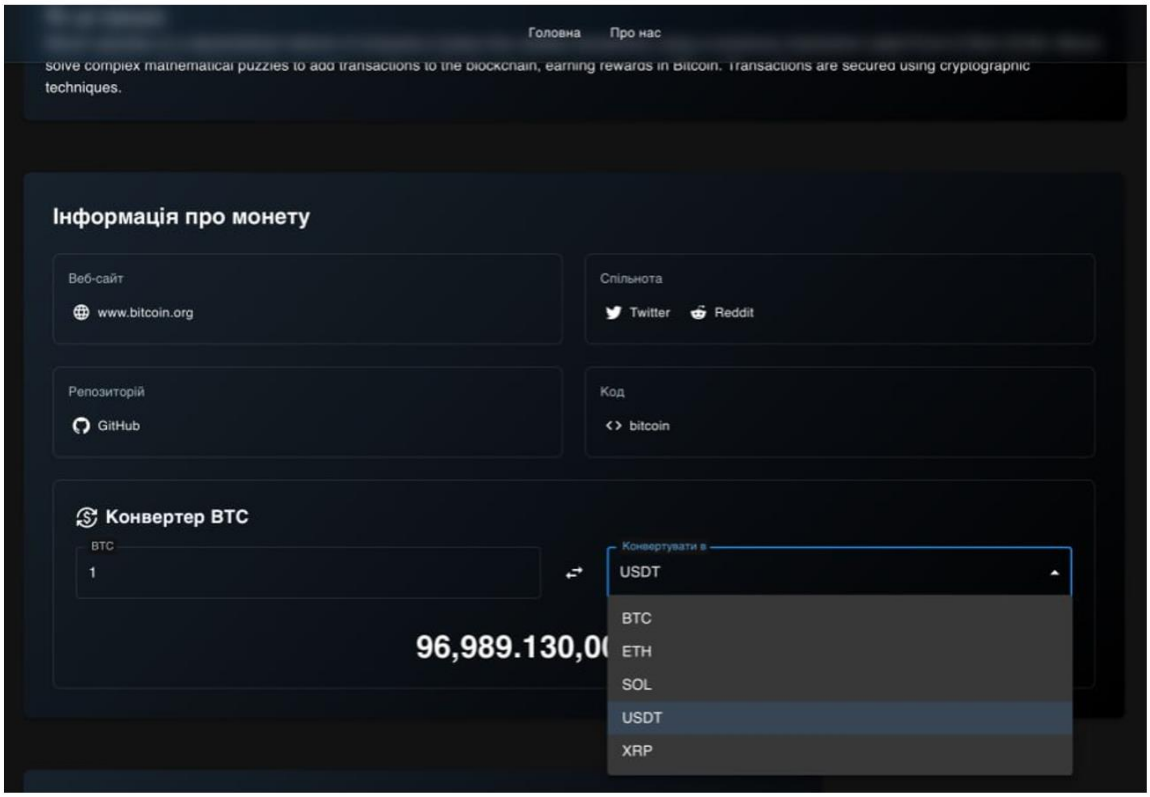
15

ЕКРАННІ ФОРМИ



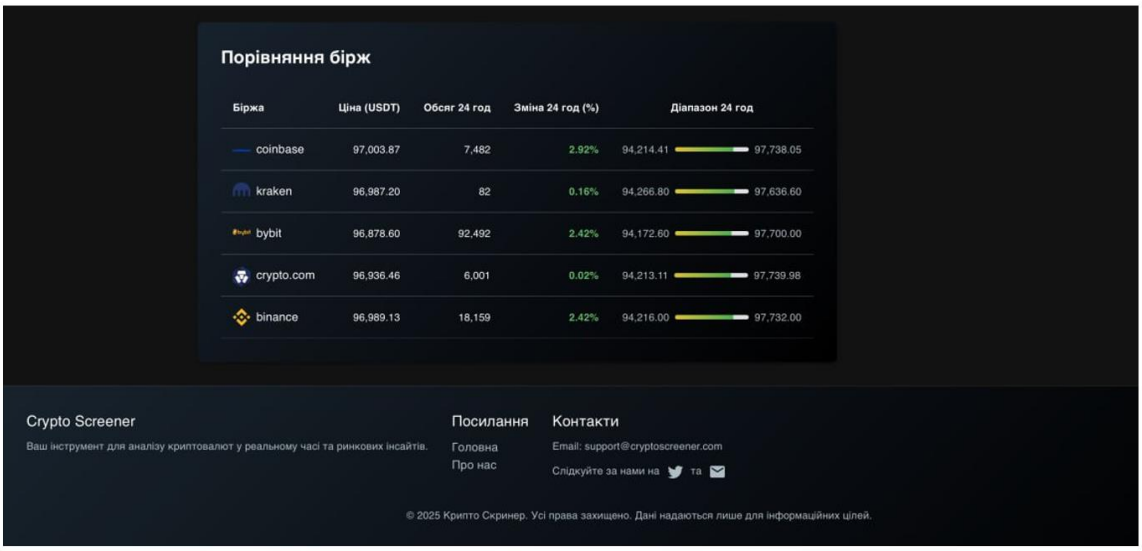
16

ЕКРАННІ ФОРМИ



17

ЕКРАННІ ФОРМИ



18

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Маліков В.Є, Залива В.В. Дослідження впливу популяризації криптовалют на повсякденне життя в цифрову епоху та актуальність розробки інструментів моніторингу ринкової ситуації. Матеріали IV Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях». Збірник тез. 24.04.2025, ДУІКТ, м. Київ. К.: ДУІКТ, Україна, С. 547.

2. Маліков В.Є, Залива В.В. Криптоскринер як засіб моніторингу ринку криптовалют: розробка та переваги інноваційного програмного забезпечення. Матеріали V Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 15.05.2025,ДУІКТ, м. Київ. К.: ДУІКТ, Україна, *(подано до друку)*.

20

ВИСНОВКИ

1. Реалізовано механізм паралельного опитування API, що дозволяє консолідувати поточні курси та статистику з обраних джерел.
2. Реалізовано налаштувати перегляд монет за ціною, обсягом торгів та зміною за 24 години, а також комбінувати фільтри по біржах і діапазонах цін.
3. Побудовано графічну візуалізація динаміки (інтерактивні лінійні графіки зміни ціни за добовий інтервал).
4. Створено персоналізований профіль (модулі реєстрації, збереження індивідуального списку відстеження).
5. Налаштовано оновлення даних щохвилини.
6. Забезпечено стійкість до збоїв окремих API.
7. Забезпечено масштабованість системи за рахунок можливості інтегрувати нові API.
8. Забезпечено безпеку за рахунок контролю CORS, який гарантує доступ до сервісу лише з довірених доменів.
9. Підтримано двомовність (укр/англ).
10. Реалізовано адаптивний дизайн для комфортної роботи на мобільних і десктопних пристроях.

ДОДАТОК Б. 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' : 'Історія'}
value="history" />
        <Tab label={language === 'en' ? 'Technology' :
'Технологія'} value="technology" />
        <Tab label={language === 'en' ? 'Application' :
'Застосування'} 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, #60aeed 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%, #60aeed
90%)'
                  : 'linear-gradient(45deg, #0d47a1 30%, #1976d2
90%)',
              boxShadow: '0 4px 16px 0 rgba(25, 118, 210, 0.18)',
            },
          }}
        >
```

```
        {language === 'en' ? 'Filter' : 'Фільтр'}
      </Button>
      <Button
        onClick={handleReset}
        sx={{
          borderRadius: 8,
          textTransform: 'none',
          color: (theme) => theme.palette.mode === 'dark' ?
'#60aeed' : '#1976d2',
          border: '1px solid',
          borderColor: (theme) => theme.palette.mode === 'dark'
? '#60aeed' : '#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' : 'Скинути'}
      </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' : 'Біржа'}
              </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)' : 'Ціна
(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' : 'Обсяг 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' : 'Обсяг 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,
      }}
    >
```

```
        {snackbarMessage}
      </Alert>
    </Snackbar>
  </Box>
 );
};
```

ExchangeComparison

```
import { Box, Table, TableBody, TableCell, TableContainer,
TableHead, TableRow, Typography, LinearProgress, Button }
from '@mui/material';
import { useScreenerStore } from '@/features/crypto-
screener/model/store';
import { useCryptoPricesBySymbol, useExchanges,
useMarketData } from '@/features/crypto-
screener/api/cryptoApi';
import { formatNumber } from '@/shared/lib/formatNumber';
import { Fade } from '@mui/material';

interface ExchangeComparisonProps {
 symbolId: string;
}

export const ExchangeComparison = ({ symbolId }:
ExchangeComparisonProps) => {
 const { language } = useScreenerStore();
 const { data: prices } = useCryptoPricesBySymbol(symbolId);
 const { data: exchanges } = useExchanges();
 const { data: marketData } = useMarketData();

 return (
  <Box
   sx={{
    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: 2,
    p: 4,
    boxShadow: (theme) => theme.shadows[3],
    mt: 4,
   }}
  >
   <Typography
    variant="h5"
    sx={{
     backgroundClip: 'text',
     WebkitBackgroundClip: 'text',
     color: "text.primary",
     fontWeight: 'bold',
     mb: 3,
    }}
   >
    {language === 'en' ? 'Exchange Comparison' :
'Порівняння бірж'}
   </Typography>
   <TableContainer>
    <Table sx={{ '& th, & td': { borderColor: 'divider' } }}>
     <TableHead>
      <TableRow>
       <TableCell sx={{ fontWeight: 'bold', color:
'text.primary' }}>
        {language === 'en' ? 'Exchange' : 'Біржа'}
       </TableCell>
       <TableCell sx={{ fontWeight: 'bold', color:
'text.primary' }} align="right">
        {language === 'en' ? 'Price (USDT)' : 'Ціна
(USDT)'}
       </TableCell>
       <TableCell sx={{ fontWeight: 'bold', color:
'text.primary' }} align="right">
        {language === 'en' ? 'Volume 24h' : 'Обсяг 24 год'}
       </TableCell>
       <TableCell sx={{ fontWeight: 'bold', color:
'text.primary' }} align="right">
        {language === 'en' ? 'Change 24h (%)' : 'Зміна 24
год (%)'}
       </TableCell>
       <TableCell sx={{ fontWeight: 'bold', color:
'text.primary' }} align="center">
        {language === 'en' ? '24h Range' : 'Діапазон 24 год'}
       </TableCell>
      </TableRow>
     </TableHead>
     <TableBody>
      {prices?.map((price, index) => {
       const exchange = exchanges?.find((e) => e.id ===
price.exchangeId);
       const market = marketData?.find(
        (m) => m.exchangeId === price.exchangeId &&
m.symbolId === price.symbolId
       );
       const rangeValue = market
        ? ((price.price.value - market.range24h.min) /
(market.range24h.max - market.range24h.min)) * 100
        : 0;

       return (
        <Fade in timeout={500 + index * 100}
key={index}>
         <TableRow
          hover
          sx={{
           '&:hover': {
            bgcolor: (theme) =>
             theme.palette.mode === 'dark' ? 'rgba(255,
255, 255, 0.05)' : 'rgba(0, 0, 0, 0.04)',
           },
          }}
         >
          <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">{exchange?.name ||
'N/A'}</Typography>
           </Box>
          </TableCell>
          <TableCell align="right" sx={{ color:
'text.primary' }}>
           {formatNumber(price.price.value, 2)}
          </TableCell>
          <TableCell align="right" sx={{ color:
'text.primary' }}>
           {formatNumber(price.volume['24h'], 0)}
          </TableCell>
          <TableCell
           align="right"
           sx={{
```

```
                color: market?.change24h &&
market.change24h >= 0 ? 'success.main' : 'error.main',
                fontWeight: 'bold',
              }}
            >
              {market?.change24h ?
`${market.change24h.toFixed(2)}%` : 'N/A'}
            </TableCell>
            <TableCell align="center">
              {market ? (
                <Box sx={{ display: 'flex', alignItems: 'center',
gap: 1 }}>
                  <Typography variant="body2"
color="text.secondary">
                    {formatNumber(market.range24h.min, 2)}
                  </Typography>
                  <LinearProgress
                    variant="determinate"
                    value={rangeValue}
                    sx={{
                      width: 100,
                      height: 6,
                      borderRadius: 3,
                      bgcolor: 'grey.300',
                      '& .MuiLinearProgress-bar': {
                        background: 'linear-gradient(to right,
#ffca28, #4caf50)',
                      },
                    }}
                  />
                  <Typography variant="body2"
color="text.secondary">
                    {formatNumber(market.range24h.max, 2)}
                  </Typography>
                </Box>
              ) : (
                'N/A'
              )}
            </TableCell>
          </TableRow>
        </Fade>
      );
    })}
  </TableBody>
 </Table>
 </TableContainer>
 </Box>
 );
};
```

Filter Modal

```
import { useState, useMemo } from 'react';
import {
 Dialog,
 DialogTitle,
 DialogContent,
 DialogActions,
 Button,
 TextField,
 Select,
 MenuItem,
 FormControl,
 InputLabel,
 Box,
 Chip,
 Fade,
 Typography,
} from '@mui/material';
import { useExchanges, useSymbols } from '../api/cryptoApi';
import { useScreenerStore } from '../model/store';
import { FilterState } from '../model/types';

interface FilterModalProps {
 open: boolean;
 onClose: () => void;
}

export const FilterModal = ({ open, onClose }:
FilterModalProps) => {
 const { filterState, setFilterState, resetFilters, language } =
useScreenerStore();
 const { data: exchanges } = useExchanges();
 const { data: symbols } = useSymbols();

 const [localFilterState, setLocalFilterState] =
useState<FilterState>({
  pair: filterState.pair,
  minPrice: filterState.minPrice,
  maxPrice: filterState.maxPrice,
  exchanges: filterState.exchanges,
 });

 const sortedExchanges = useMemo(() => {
  return exchanges ? [...exchanges].sort((a, b) =>
a.name.localeCompare(b.name)) : [];
 }, [exchanges]);

 const sortedSymbols = useMemo(() => {
  return symbols ? [...symbols].sort((a, b) =>
a.pair.localeCompare(b.pair)) : [];
 }, [symbols]);

 const handleApply = () => {
  if (
   localFilterState.minPrice !== null &&
   localFilterState.maxPrice !== null &&
   localFilterState.minPrice > localFilterState.maxPrice
  ) {
   setLocalFilterState({
    ...localFilterState,
    minPrice: localFilterState.maxPrice,
    maxPrice: localFilterState.minPrice,
   });
  }
  setFilterState(localFilterState);
  onClose();
 };

 const handleReset = () => {
  resetFilters();
  setLocalFilterState({
   pair: null,
   minPrice: null,
   maxPrice: null,
   exchanges: [],
  });
  onClose();
 };

 const handleSelectAllExchanges = () => {
  setLocalFilterState({
   ...localFilterState,
   exchanges: sortedExchanges.map((e) => e.id),
  });
 };

 const handleClearAllExchanges = () => {
```

```
  setLocalFilterState({
    ...localFilterState,
    exchanges: [],
  });
 };

 return (
  <Dialog
    open={open}
    onClose={onClose}
    maxWidth="sm"
    fullWidth
    PaperProps={{
      sx: {
        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}`,
      },
    }}
   >
    <DialogTitle
      sx={{
        fontWeight: 'bold',
        background: (theme) =>
          theme.palette.mode === 'dark'
            ? 'linear-gradient(45deg, #60aeed 30%, #4e96de 90%)'
            : 'linear-gradient(45deg, #1976d2 30%, #0d47a1
90%)',
        WebkitBackgroundClip: 'text',
        WebkitTextFillColor: 'transparent',
        paddingTop: 4,
      }}
    >
      {language === 'en' ? 'Filter Settings' : 'Налаштування
фільтра'}
    </DialogTitle>
    <DialogContent sx={{ paddingRight: 4, paddingLeft: 4 }}>
      <Box sx={{ display: 'flex', flexDirection: 'column', gap: 2,
mt: 1 }}>
        <FormControl fullWidth>
          <InputLabel sx={{
              bgcolor: 'background.default',
              borderRadius: 5
            }}>{language === 'en' ? 'Pair' : 'Пара'}</InputLabel>
          <Select
            value={localFilterState.pair || ''}
            onChange={(e) =>
              setLocalFilterState({ ...localFilterState, pair:
e.target.value || null })
            }
            sx={{
              bgcolor: 'background.default',
              borderRadius: 1,
              '& .MuiSelect-select': { display: 'flex', alignItems:
'center' },
            }}
          >
            <MenuItem value="">
              <Typography>{language === 'en' ? 'None' :
'Немає'}</Typography>
            </MenuItem>
            {sortedSymbols.map((symbol) => (
              <MenuItem key={symbol.id} value={symbol.pair}>
                <Box
                  dangerouslySetInnerHTML={{ __html:
symbol.iconSvg }}
                  sx={{ width: 20, height: 20, mr: 1 }}
                />
                <Typography>{symbol.pair}</Typography>
              </MenuItem>
            ))}
          </Select>
        </FormControl>
        <TextField
          label={language === 'en' ? 'Min Price (USDT)' : 'Мін.
ціна (USDT)'}
          type="number"
          value={localFilterState.minPrice ?? ''}
          onChange={(e) => {
            const value = e.target.value ? Number(e.target.value) :
null;
            if (value !== null && value < 0) return;
            setLocalFilterState({ ...localFilterState, minPrice:
value });
          }}
          fullWidth
          sx={{
            bgcolor: 'background.default',
            borderRadius: 1,
            '& .MuiInputBase-root': { borderRadius: 1 },
          }}
        />
        <TextField
          label={language === 'en' ? 'Max Price (USDT)' : 'Макс.
ціна (USDT)'}
          type="number"
          value={localFilterState.maxPrice ?? ''}
          onChange={(e) => {
            const value = e.target.value ? Number(e.target.value) :
null;
            if (value !== null && value < 0) return;
            setLocalFilterState({ ...localFilterState, maxPrice:
value });
          }}
          fullWidth
          sx={{
            bgcolor: 'background.default',
            borderRadius: 1,
            '& .MuiInputBase-root': { borderRadius: 1 },
          }}
        />
        <Box>
          <Box sx={{ display: 'flex', justifyContent: 'space-
between', alignItems: 'center', mb: 1 }}>
            <Typography variant="body2"
color="text.secondary">
              {language === 'en'
                ? `Selected Exchanges:
${localFilterState.exchanges.length}`
                : `Вибрано бірж:
${localFilterState.exchanges.length}`}
            </Typography>
            <Box sx={{ display: 'flex', gap: 1 }}>
              <Button
                variant="outlined"
                size="small"
                onClick={handleSelectAllExchanges}
                sx={{
                  textTransform: 'none',
                  borderRadius: 8,
                  borderColor: (theme) =>
                    theme.palette.mode === 'dark' ? '#60aeed' :
'#1976d2',
```

```
                color: (theme) =>
                  theme.palette.mode === 'dark' ? '#60aeed' :
'#1976d2',
                '&:hover': {
                  borderColor: (theme) =>
                    theme.palette.mode === 'dark' ? '#4e96de' :
'#0d47a1',
                  background: (theme) =>
                    theme.palette.mode === 'dark'
                      ? 'rgba(96, 174, 238, 0.08)'
                      : 'rgba(25, 118, 210, 0.08)',
                },
              }}
            >
              {language === 'en' ? 'Select All' : 'Вибрати всі'}
            </Button>
            <Button
              variant="outlined"
              size="small"
              onClick={handleClearAllExchanges}
              sx={{
                textTransform: 'none',
                borderRadius: 8,
                borderColor: (theme) =>
                  theme.palette.mode === 'dark' ? '#60aeed' :
'#1976d2',
                color: (theme) =>
                  theme.palette.mode === 'dark' ? '#60aeed' :
'#1976d2',
                '&:hover': {
                  borderColor: (theme) =>
                    theme.palette.mode === 'dark' ? '#4e96de' :
'#0d47a1',
                  background: (theme) =>
                    theme.palette.mode === 'dark'
                      ? 'rgba(96, 174, 238, 0.08)'
                      : 'rgba(25, 118, 210, 0.08)',
                },
              }}
            >
              {language === 'en' ? 'Clear All' : 'Очистити всі'}
            </Button>
          </Box>
        </Box>
        <FormControl fullWidth>
          <InputLabel>{language === 'en' ? 'Exchanges' :
'Біржі'}</InputLabel>
          <Select
            multiple
            value={localFilterState.exchanges}
            onChange={(e) =>
              setLocalFilterState({ ...localFilterState, exchanges:
e.target.value as string[] })
            }
            renderValue={(selected) => (
              <Box sx={{ display: 'flex', flexWrap: 'wrap', gap:
0.5 }}>
                {selected.map((value) => {
                  const exchange = sortedExchanges.find((e) =>
e.id === value);
                  return (
                    <Chip
                      key={value}
                      avatar={
                        <Box
                          dangerouslySetInnerHTML={{ __html:
exchange?.iconSvg || '' }}
                          sx={{ width: 16, height: 16, display: 'flex',
alignItems: 'center' }}
                        />
                      }
                      label={exchange?.name || value}
                      sx={{
                        bgcolor: (theme) =>
                          theme.palette.mode === 'dark' ? '#1565c0' :
'#e3f2fd',
                        color: (theme) =>
                          theme.palette.mode === 'dark' ? '#ffffff' :
'#1976d2',
                      }}
                    />
                  );
                })}
              </Box>
            )}
            sx={{
              bgcolor: 'background.default',
              borderRadius: 1,
              '& .MuiSelect-select': { display: 'flex', alignItems:
'center' },
            }}
          >
            {sortedExchanges.map((exchange) => (
              <MenuItem key={exchange.id}
value={exchange.id}>
                <Box
                  dangerouslySetInnerHTML={{ __html:
exchange.iconSvg }}
                  sx={{ width: 20, height: 20, mr: 1 }}
                />
                <Typography>{exchange.name}</Typography>
              </MenuItem>
            ))}
          </Select>
        </FormControl>
      </Box>
    </DialogContent>
    <DialogActions sx={{ paddingRight: 4, paddingLeft: 4,
paddingBottom: 4 }}>
      <Button
        onClick={handleReset}
        sx={{
          borderRadius: 8,
          textTransform: 'none',
          color: (theme) => theme.palette.mode === 'dark' ?
'#60aeed' : '#1976d2',
          border: '1px solid',
          borderColor: (theme) => theme.palette.mode === 'dark'
? '#60aeed' : '#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' : 'Скинути'}
      </Button>
      <Button
        onClick={handleApply}
        variant="contained"
        sx={{
          borderRadius: 8,
          textTransform: 'none',
          fontWeight: 600,
          background: (theme) =>
```

```
          theme.palette.mode === 'dark'
            ? 'linear-gradient(45deg, #60aeed 30%, #4e96de
90%)'
            : 'linear-gradient(45deg, #1976d2 30%, #0d47a1
90%)',
          color: '#fff',
          '&:hover': {
            background: (theme) =>
              theme.palette.mode === 'dark'
                ? 'linear-gradient(45deg, #4e96de 30%, #60aeed
90%)'
                : 'linear-gradient(45deg, #0d47a1 30%, #1976d2
90%)',
          },
        }} >
        {language === 'en' ? 'Apply' : 'Застосувати'}
      </Button>
    </DialogActions>
  </Dialog>
 );
};
```

InfoSection

```
import { memo, useMemo, useState } from 'react';
import { Box, Typography, Button, Fade, Paper, TextField,
MenuItem, Select, InputLabel, FormControl } from
'@mui/material';
import { useScreenerStore } from '../model/store';
import { useCoinGeckoData, useSymbols, useCryptoPrices }
from '../api/cryptoApi';
import { formatNumber } from '@/shared/lib/formatNumber';
import LanguageIcon from '@mui/icons-material/Language';
import TwitterIcon from '@mui/icons-material/Twitter';
import RedditIcon from '@mui/icons-material/Reddit';
import GitHubIcon from '@mui/icons-material/GitHub';
import CodeIcon from '@mui/icons-material/Code';
import CurrencyExchangeIcon from '@mui/icons-
material/CurrencyExchange';
import SwapHorizIcon from '@mui/icons-material/SwapHoriz';

interface InfoSectionProps {
 baseCoin: string;
 price: number;
}

export const InfoSection = memo(({ baseCoin, price }:
InfoSectionProps) => {
 const { language } = useScreenerStore();
 const { data: coinData, isLoading } =
useCoinGeckoData(baseCoin);
 const { data: symbols } = useSymbols();
 const { data: allPrices } = useCryptoPrices();
 const [amount, setAmount] = useState<string>('1');
 const [targetCurrency, setTargetCurrency] =
useState<string>('USDT');

 const availableCurrencies = useMemo(() => {
  const currencies = new Set<string>();
  currencies.add('USDT');
  symbols?.forEach(symbol => {
   if (symbol.baseCoin) {
    currencies.add(symbol.baseCoin);
   }
  });
  return Array.from(currencies).sort();
 }, [symbols]);

 const getCurrencyPrice = (currency: string) => {
  if (currency === 'USDT') return 1;
  if (currency === baseCoin) return price;

  const symbol = symbols?.find(s => s.baseCoin === currency
&& s.quoteCoin === 'USDT');
  if (!symbol) return 0;

  const priceData = allPrices?.find(p => p.symbolId ===
symbol.id);
  return priceData?.price.value || 0;
 };

 const handleAmountChange = (event:
React.ChangeEvent<HTMLInputElement>) => {
  const value = event.target.value;
  if (/^\d*\.?\d*$/.test(value)) {
   setAmount(value);
  }
 };

 const handleSwapCurrencies = () => {
  setTargetCurrency(baseCoin);
 };

 const calculateResult = () => {
  const numAmount = parseFloat(amount) || 0;
  const targetPrice = getCurrencyPrice(targetCurrency);

  if (targetCurrency === 'USDT') {
   return numAmount * price;
  }

  if (targetCurrency === baseCoin) {
   return numAmount;
  }

  // Конвертируем через USDT
  const amountInUsdt = numAmount * price;
  return amountInUsdt / targetPrice;
 };

 if (isLoading) {
  return (
   <Box
    sx={{
     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: 2,
     p: 3,
     boxShadow: (theme) => theme.shadows[3],
     mt: 4,
    }}
   >
    <Typography variant="h6" color="text.primary" mb={2}>
     {language === 'en' ? 'Loading...' : 'Завантаження...'}
    </Typography>
   </Box>
  );
 }

 const website = coinData?.links.homepage?.[0] || '';
 const twitter = coinData?.links.twitter_screen_name
  ?
`https://twitter.com/${coinData.links.twitter_screen_name}`
  : '';
```

```
  const reddit = coinData?.links.subreddit_url || '';
  const github = coinData?.links.repos_url.github?.[0] || '';
  const apiId = coinData?.id || '';

  return (
   <Fade in timeout={500}>
    <Box
     sx={{
      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: 2,
      p: 4,
      boxShadow: (theme) => theme.shadows[3],
      mt: 4,
     }}
    >
     <Typography
      variant="h5"
      sx={{
       backgroundClip: 'text',
       WebkitBackgroundClip: 'text',
       color: "text.primary",
       fontWeight: 'bold',
       mb: 3,
      }}
     >
      {language === 'en' ? 'Coin Information' : 'Інформація
про монету'}
     </Typography>

     <Box sx={{ display: 'flex', flexWrap: 'wrap', gap: 3 }}>
      {website && (
       <Box sx={{ width: { xs: '100%', md: 'calc(50% - 12px)'
} }}>
        <Paper
         elevation={0}
         sx={{
          p: 2,
          height: '100%',
          background: 'transparent',
          border: '1px solid',
          borderColor: 'divider',
          borderRadius: 2,
          transition: 'transform 0.2s',
          '&:hover': {
           transform: 'translateY(-4px)',
          },
         }}
        >
         <Typography variant="subtitle2"
color="text.secondary" mb={1}>
          {language === 'en' ? 'Website' : 'Веб-сайт'}
         </Typography>
         <Button
          href={website}
          target="_blank"
          rel="noopener"
          startIcon={<LanguageIcon />}
          fullWidth
          sx={{
           justifyContent: 'flex-start',
           textTransform: 'none',
           color: 'text.primary',
           '&:hover': {
            background: 'rgba(25, 118, 210, 0.08)',
           },
          }}
         >
          {website.replace('https://', '').replace('http://', '')}
         </Button>
        </Paper>
       </Box>
      )}

      <Box sx={{ width: { xs: '100%', md: 'calc(50% - 12px)'
} }}>
       <Paper
        elevation={0}
        sx={{
         p: 2,
         height: '100%',
         background: 'transparent',
         border: '1px solid',
         borderColor: 'divider',
         borderRadius: 2,
         transition: 'transform 0.2s',
         '&:hover': {
          transform: 'translateY(-4px)',
         },
        }}
       >
        <Typography variant="subtitle2"
color="text.secondary" mb={1}>
         {language === 'en' ? 'Community' : 'Спільнота'}
        </Typography>
        <Box sx={{ display: 'flex', gap: 1 }}>
         {twitter && (
          <Button
           href={twitter}
           target="_blank"
           rel="noopener"
           startIcon={<TwitterIcon />}
           sx={{
            textTransform: 'none',
            color: 'text.primary',
            '&:hover': {
             background: 'rgba(25, 118, 210, 0.08)',
            },
           }}
          >
           Twitter
          </Button>
         )}
         {reddit && (
          <Button
           href={reddit}
           target="_blank"
           rel="noopener"
           startIcon={<RedditIcon />}
           sx={{
            textTransform: 'none',
            color: 'text.primary',
            '&:hover': {
             background: 'rgba(25, 118, 210, 0.08)',
            },
           }}
          >
           Reddit
          </Button>
         )}
         {!twitter && !reddit && (
          <Typography variant="body2"
color="text.secondary">
           {language === 'en' ? 'N/A' : 'Немає даних'}
```

```
            </Typography>
          )}
        </Box>
      </Paper>
    </Box>

    {github && (
      <Box sx={{ width: { xs: '100%', md: 'calc(50% - 12px)'
} }}>
        <Paper
          elevation={0}
          sx={{
            p: 2,
            height: '100%',
            background: 'transparent',
            border: '1px solid',
            borderColor: 'divider',
            borderRadius: 2,
            transition: 'transform 0.2s',
            '&:hover': {
              transform: 'translateY(-4px)',
            },
          }}
        >
          <Typography variant="subtitle2"
color="text.secondary" mb={1}>
            {language === 'en' ? 'Repo' : 'Репозиторій'}
          </Typography>
          <Button
            href={github}
            target="_blank"
            rel="noopener"
            startIcon={<GitHubIcon />}
            fullWidth
            sx={{
              justifyContent: 'flex-start',
              textTransform: 'none',
              color: 'text.primary',
              '&:hover': {
                background: 'rgba(25, 118, 210, 0.08)',
              },
            }}
          >
            GitHub
          </Button>
        </Paper>
      </Box>
    )}

    {apiId && (
      <Box sx={{ width: { xs: '100%', md: 'calc(50% - 12px)'
} }}>
        <Paper
          elevation={0}
          sx={{
            p: 2,
            height: '100%',
            background: 'transparent',
            border: '1px solid',
            borderColor: 'divider',
            borderRadius: 2,
            transition: 'transform 0.2s',
            '&:hover': {
              transform: 'translateY(-4px)',
            },
          }}
        >
          <Typography variant="subtitle2"
color="text.secondary" mb={1}>
            {language === 'en' ? 'Code' : 'Код'}
          </Typography>
          <Button
            startIcon={<CodeIcon />}
            fullWidth
            sx={{
              justifyContent: 'flex-start',
              textTransform: 'none',
              color: 'text.primary',
              '&:hover': {
                background: 'rgba(25, 118, 210, 0.08)',
              },
            }}
          >
            {apiId}
          </Button>
        </Paper>
      </Box>
    )}

    <Box sx={{ width: '100%' }}>
      <Paper
        elevation={0}
        sx={{
          p: 3,
          background: 'transparent',
          border: '1px solid',
          borderColor: 'divider',
          borderRadius: 2,
          transition: 'transform 0.2s',
          '&:hover': {
            transform: 'translateY(-4px)',
          },
        }}
      >
        <Typography
          variant="h6"
          sx={{
            background: (theme) =>
              theme.palette.mode === 'dark'
                ? 'linear-gradient(45deg, #1e88e5 30%, #42a5f5
90%)'
                : 'linear-gradient(45deg, #1976d2 30%, #2196f3
90%)',
            backgroundClip: 'text',
            WebkitBackgroundClip: 'text',
            color: 'text.primary',
            fontWeight: 'bold',
            mb: 2,
            display: 'flex',
            alignItems: 'center',
            gap: 1,
          }}
        >
          <CurrencyExchangeIcon />
          {language === 'en' ? `${baseCoin} Converter` :
`Конвертер ${baseCoin}`}
        </Typography>

        <Box sx={{ display: 'flex', flexDirection: 'column',
gap: 2 }}>
          <Box sx={{ display: 'flex', gap: 2, alignItems: 'center'
}}>
            <TextField
              value={amount}
              onChange={handleAmountChange}
              label={baseCoin}
              variant="outlined"
              fullWidth
```

```
            sx={{
              '& .MuiOutlinedInput-root': {
                '& fieldset': {
                  borderColor: 'divider',
                },
                '&:hover fieldset': {
                  borderColor: 'primary.main',
                },
              },
            }}
          />
          <Button
            onClick={handleSwapCurrencies}
            sx={{
              minWidth: 'auto',
              p: 1,
              borderRadius: '50%',
              color: 'text.primary',
              '&:hover': {
                background: 'rgba(25, 118, 210, 0.08)',
              },
            }}
          >
            <SwapHorizIcon />
          </Button>
          <FormControl fullWidth>
            <InputLabel id="currency-select-label">
              {language === 'en' ? 'Convert to' : 'Конвертувати
в'}
            </InputLabel>
            <Select
              labelId="currency-select-label"
              value={targetCurrency}
              label={language === 'en' ? 'Convert to' :
'Конвертувати в'}
              onChange={(e) =>
setTargetCurrency(e.target.value)}
              sx={{
                '& .MuiOutlinedInput-notchedOutline': {
                  borderColor: 'divider',
                },
                '&:hover .MuiOutlinedInput-notchedOutline': {
                  borderColor: 'primary.main',
                },
              }}
            >
              {availableCurrencies.map((currency) => (
                <MenuItem key={currency} value={currency}>
                  {currency}
                </MenuItem>
              ))}
            </Select>
          </FormControl>
        </Box>

        <Typography
          variant="h4"
          sx={{
            color: 'text.primary',
            fontWeight: 'bold',
            textAlign: 'center',
            mt: 2,
          }}
        >
          {formatNumber(calculateResult(), 6)}
{targetCurrency}
        </Typography>
      </Box>
    </Paper>
      </Box>
    </Box>
  </Fade>
 );
});
```

Головні логічні компоненти

ExchangeClient

```
import ccxt from "ccxt";
import { logger } from "../logging/logger";

interface Range24h {
 min: number;
 max: number;
}

export class ExchangeClient {
 private exchanges: Map<string, any>;
 constructor() {
  this.exchanges = new Map();
  this.exchanges.set("binance", new (ccxt as any).binance());
  this.exchanges.set("coinbase", new (ccxt as any).coinbase());
  this.exchanges.set("kraken", new (ccxt as any).kraken());
  this.exchanges.set("bybit", new (ccxt as any).bybit());
  this.exchanges.set("crypto.com", new (ccxt as
any).cryptocom());
 }

 async fetchPrice(exchange: string, symbol: string):
Promise<number> {
  const client = this.exchanges.get(exchange);
  if (!client) {
   throw new Error(`Unsupported exchange: ${exchange}`);
  }
  let normalizedSymbol = symbol;
  if (exchange === "bybit") {
   normalizedSymbol = symbol.replace("/", "");
  } else if (exchange === "crypto.com") {
   normalizedSymbol = symbol.replace("/", "_");
  } else if (exchange === "coinbase") {
   normalizedSymbol = symbol.replace("USDT", "USD");
  }

  try {
   const ticker = await client.fetchTicker(normalizedSymbol);
   return ticker.last || 0;
  } catch (error) {
   logger.warn(`Failed to fetch price for
${normalizedSymbol} on ${exchange}: ${(error as
Error).message}`);
   return 0;
  }
 }

 async fetchVolume(exchange: string, symbol: string):
Promise<{
  "1h": number;
  "24h": number;
  "7d": number;
  "30d": number;
  total: number;
 }> {
  const client = this.exchanges.get(exchange);
  if (!client) {
   throw new Error(`Unsupported exchange: ${exchange}`);
  }
```

```
    let normalizedSymbol = symbol;
    if (exchange === "bybit") {
      normalizedSymbol = symbol.replace("/", "");
    } else if (exchange === "crypto.com") {
      normalizedSymbol = symbol.replace("/", "_");
    } else if (exchange === "coinbase") {
      normalizedSymbol = symbol.replace("USDT", "USD");
    }
    try {
      let volume24h = 0;
      try {
        const ticker = await
client.fetchTicker(normalizedSymbol);
        volume24h = ticker.baseVolume || 0;
        if (!ticker.baseVolume) {
          logger.warn(`No volume data for 24h period via
fetchTicker for ${normalizedSymbol} on ${exchange}`);
        }
      } catch (error) {
        logger.warn(`Failed to fetch 24h volume via fetchTicker
for ${normalizedSymbol} on ${exchange}: ${(error as
Error).message}`);
      }
      if (!volume24h) {
        try {
          const since24h = Date.now() - 24 * 60 * 60 * 1000;
          const ohlcv24h = await
client.fetchOHLCV(normalizedSymbol, "1h", since24h);
          volume24h = Array.isArray(ohlcv24h)
            ? ohlcv24h.reduce((sum: number, candle: any) => sum
+ (candle[5] || 0), 0)
            : 0;
          if (!volume24h) {
            logger.warn(`No volume data for 24h period via
fetchOHLCV for ${normalizedSymbol} on ${exchange}`);
          }
        } catch (error) {
          logger.warn(`Failed to fetch 24h volume via
fetchOHLCV for ${normalizedSymbol} on ${exchange}:
${(error as Error).message}`);
        }
      }
      const since1h = Date.now() - 60 * 60 * 1000;
      const ohlcv1h = await
client.fetchOHLCV(normalizedSymbol, "1m", since1h);
      const volume1h = Array.isArray(ohlcv1h)
        ? ohlcv1h.reduce((sum: number, candle: any) => sum +
(candle[5] || 0), 0)
        : 0;
      if (!volume1h) {
        logger.warn(`No volume data for 1h period for
${normalizedSymbol} on ${exchange}`);
      }
      const since7d = Date.now() - 7 * 24 * 60 * 60 * 1000;
      const ohlcv7d = await
client.fetchOHLCV(normalizedSymbol, "1h", since7d);
      const volume7d = Array.isArray(ohlcv7d)
        ? ohlcv7d.reduce((sum: number, candle: any) => sum +
(candle[5] || 0), 0)
        : 0;
      if (!volume7d) {
        logger.warn(`No volume data for 7d period for
${normalizedSymbol} on ${exchange}`);
      }
      const since30d = Date.now() - 30 * 24 * 60 * 60 * 1000;
      const ohlcv30d = await
client.fetchOHLCV(normalizedSymbol, "1d", since30d);
      const volume30d = Array.isArray(ohlcv30d)
        ? ohlcv30d.reduce((sum: number, candle: any) => sum +
(candle[5] || 0), 0)
        : 0;
      if (!volume30d) {
        logger.warn(`No volume data for 30d period for
${normalizedSymbol} on ${exchange}`);
      }
      const sinceTotal = 0;
      const ohlcvTotal = await
client.fetchOHLCV(normalizedSymbol, "1d", sinceTotal);
      const volumeTotal = Array.isArray(ohlcvTotal)
        ? ohlcvTotal.reduce((sum: number, candle: any) => sum +
(candle[5] || 0), 0)
        : 0;
      if (!volumeTotal) {
        logger.warn(`No volume data for total period for
${normalizedSymbol} on ${exchange}`);
      }

      return {
        "1h": volume1h,
        "24h": volume24h,
        "7d": volume7d,
        "30d": volume30d,
        total: volumeTotal,
      };
    } catch (error) {
      logger.error(`Failed to fetch volume for ${symbol} on
${exchange}: ${(error as Error).message}`);
      return {
        "1h": 0,
        "24h": 0,
        "7d": 0,
        "30d": 0,
        total: 0,
      };
    }
  }

  async fetchMarketData(exchange: string, symbol: string):
Promise<{
    change24h: number;
    range24h: Range24h;
  }> {
    const client = this.exchanges.get(exchange);
    if (!client) {
      throw new Error(`Unsupported exchange: ${exchange}`);
    }

    let normalizedSymbol = symbol;
    if (exchange === "bybit") {
      normalizedSymbol = symbol.replace("/", "");
    } else if (exchange === "crypto.com") {
      normalizedSymbol = symbol.replace("/", "_");
    } else if (exchange === "coinbase") {
      normalizedSymbol = symbol.replace("USDT", "USD");
    }

    try {
      let change24h = 0;
      let range24h: Range24h = { min: 0, max: 0 };
      let currentPrice = 0;

      try {
        const ticker = await
client.fetchTicker(normalizedSymbol);
        change24h = ticker.percentage || 0;
        range24h = {
          min: ticker.low || 0,
```

```
        max: ticker.high || 0,
      };
      currentPrice = ticker.last || 0;

      if (!ticker.percentage) {
        logger.warn(`No change24h data for
${normalizedSymbol} on ${exchange}`);
      }
      if (!ticker.low || !ticker.high) {
        logger.warn(`No range24h data for
${normalizedSymbol} on ${exchange}`);
      }
      if (!currentPrice) {
        logger.warn(`No current price data for
${normalizedSymbol} on ${exchange}`);
      }
    } catch (error) {
      logger.warn(`Failed to fetch market data via fetchTicker
for ${normalizedSymbol} on ${exchange}: ${(error as
Error).message}`);
      if (exchange === "coinbase") {
        normalizedSymbol = normalizedSymbol.replace("/", "-
");
        try {
          const ticker = await
client.fetchTicker(normalizedSymbol);
          change24h = ticker.percentage || 0;
          range24h = {
            min: ticker.low || 0,
            max: ticker.high || 0,
          };
          currentPrice = ticker.last || 0;

          if (!ticker.percentage) {
            logger.warn(`No change24h data for
${normalizedSymbol} on ${exchange} (alternative format)`);
          }
          if (!ticker.low || !ticker.high) {
            logger.warn(`No range24h data for
${normalizedSymbol} on ${exchange} (alternative format)`);
          }
          if (!currentPrice) {
            logger.warn(`No current price data for
${normalizedSymbol} on ${exchange} (alternative format)`);
          }
        } catch (error) {
          logger.warn(`Failed to fetch market data via fetchTicker
for ${normalizedSymbol} on ${exchange} (alternative format):
${(error as Error).message}`);
        }
      }
    }

    if (!change24h || !range24h.min || !range24h.max) {
      try {
        const since24h = Date.now() - 24 * 60 * 60 * 1000;
        const ohlcv24h = await
client.fetchOHLCV(normalizedSymbol, "1h", since24h);
        if (Array.isArray(ohlcv24h) && ohlcv24h.length > 0) {
          if (!change24h && currentPrice) {
            const price24hAgo = ohlcv24h[0][4];
            if (price24hAgo) {
              change24h = ((currentPrice - price24hAgo) /
price24hAgo) * 100;
            }
          }

          if (!range24h.min || !range24h.max) {
            const prices = ohlcv24h.map((candle: any) => ({
              low: candle[3],
              high: candle[2],
            }));
            range24h = {
              min: Math.min(...prices.map((p: any) => p.low)),
              max: Math.max(...prices.map((p: any) => p.high)),
            };
          }
        } else {
          logger.warn(`No OHLCV data for 24h period for
${normalizedSymbol} on ${exchange}`);
        }
      } catch (error) {
        logger.warn(`Failed to fetch market data via
fetchOHLCV for ${normalizedSymbol} on ${exchange}:
${(error as Error).message}`);
      }
    }

    return {
      change24h,
      range24h,
    };
  } catch (error) {
    logger.error(`Failed to fetch market data for ${symbol} on
${exchange}: ${(error as Error).message}`);
    return {
      change24h: 0,
      range24h: { min: 0, max: 0 },
    };
  }
 }
}
```

FetchAndStoreCryptoPricesUseCase

```
import { CryptoPrice } from "../../domain/crypto-
price/entities/CryptoPrice";
import { Price } from "../../domain/crypto-price/value-
objects/Price";
import { CryptoPriceAggregate } from "../../domain/crypto-
price/aggregates/CryptoPriceAggregate";
import { ICryptoPriceRepository } from "../../domain/crypto-
price/repositories/ICryptoPriceRepository";
import { IExchangeRepository } from "../../domain/crypto-
price/repositories/IExchangeRepository";
import { ISymbolRepository } from "../../domain/crypto-
price/repositories/ISymbolRepository";
import { IMarketDataRepository } from "../../domain/market-
data/repositories/IMarketDataRepository";
import { MarketData } from "../../domain/market-
data/entities/MarketData";
import { ExchangeService } from
"../services/ExchangeService";
import { v4 as uuidv4 } from "uuid";
import { logger } from "../../infrastructure/logging/logger";

interface VolumeData {
  "1h": number;
  "24h": number;
  "7d": number;
  "30d": number;
  total: number;
}

interface Range24h {
  min: number;
  max: number;
}
```

```
interface ExchangePrice {
  symbolId: string;
  symbol: string;
  exchangeId: string;
  exchange: string;
  price: number;
  volume: VolumeData;
  change24h: number;
  range24h: Range24h;
}

export class FetchAndStoreCryptoPricesUseCase {
  constructor(
    private exchangeService: ExchangeService,
    private cryptoPriceRepository: ICryptoPriceRepository,
    private exchangeRepository: IExchangeRepository,
    private symbolRepository: ISymbolRepository,
    private marketDataRepository: IMarketDataRepository
  ) {}
  async execute(): Promise<void> {
    const exchanges = await this.exchangeRepository.findAll();
    const symbols = await this.symbolRepository.findAll();

    const exchangeMap = new Map<string, string>();
    exchanges.forEach((exchange) => {
      exchangeMap.set(exchange.getId(), exchange.getName());
    });

    const symbolMap = new Map<string, { pair: string;
exchanges: string[] }>();
    symbols.forEach((symbol) => {
      symbolMap.set(symbol.getId(), {
        pair: symbol.getPair(),
        exchanges: symbol.getExchanges(),
      });
    });

    const prices: ExchangePrice[] = [];
    for (const [symbolId, { pair, exchanges }] of symbolMap) {
      for (const exchangeId of exchanges) {
        const exchangeName = exchangeMap.get(exchangeId);
        if (!exchangeName) continue;

        try {
          const price = await
this.exchangeService.fetchPrice(exchangeName, pair);
          if (!price) {
            logger.warn(`Skipping ${pair} on ${exchangeName}:
price data unavailable`);
            continue;
          }

          const volume = await
this.exchangeService.fetchVolume(exchangeName, pair);

          const marketData = await
this.exchangeService.fetchMarketData(exchangeName, pair);
          if (!marketData.change24h &&
(!marketData.range24h.min || !marketData.range24h.max)) {
            logger.warn(`Skipping ${pair} on ${exchangeName}:
market data (change24h and range24h) unavailable`);
            continue;
          }

          prices.push({
            symbolId,
            symbol: pair,
            exchangeId,
            exchange: exchangeName,
            price,
            volume,
            change24h: marketData.change24h,
            range24h: marketData.range24h,
          });
        } catch (error) {
          logger.error(`Failed to fetch data for ${pair} on
${exchangeName}: ${(error as Error).message}`);
        }
      }
    }

    logger.info("Fetched prices:");
    logger.info(JSON.stringify(prices, null, 2));
    logger.info(`Fetched ${prices.length} prices`);

    let savedCount = 0;
    for (const priceData of prices) {
      const price = new Price(priceData.price);
      const cryptoPrice = new CryptoPrice(
        uuidv4(),
        priceData.symbolId,
        priceData.exchangeId,
        price,
        priceData.volume,
        new Date()
      );
      const aggregate = new CryptoPriceAggregate(cryptoPrice);
      aggregate.recordUpdatedEvent();
      try {
        logger.info(`Attempting to save crypto price for symbolId
${priceData.symbolId} on exchangeId
${priceData.exchangeId}`);
        await this.cryptoPriceRepository.save(cryptoPrice);
        savedCount++;
        logger.info(`Successfully saved crypto price for symbolId
${priceData.symbolId}`);
      } catch (error) {
        logger.error(`Failed to save crypto price for symbolId
${priceData.symbolId}: ${(error as Error).message}`);
      }
      const marketData = new MarketData(
        uuidv4(),
        priceData.exchangeId,
        priceData.symbolId,
        priceData.change24h,
        priceData.range24h
      );
      try {
        logger.info(`Attempting to save market data for symbolId
${priceData.symbolId} on exchangeId
${priceData.exchangeId}`);
        await this.marketDataRepository.save(marketData);
        logger.info(`Successfully saved market data for symbolId
${priceData.symbolId}`);
      } catch (error) {
        logger.error(`Failed to save market data for symbolId
${priceData.symbolId}: ${(error as Error).message}`);
      }
    }

    logger.info(`Successfully saved ${savedCount} out of
${prices.length} prices`);
  }
}
```