Cryptocurrency Data API: A Hexagonal Architecture Approach

Artur Denislamov 5123005, Dzmitry Karobka 5123080

February 15, 2025

1 Introduction

The project aims to develop a RESTful API for retrieving cryptocurrency market data. This API allows users to fetch real-time and historical data on various cryptocurrencies, supporting features like filtering, pagination, and caching. The main use cases include:

- Fetching cryptocurrency prices and market trends.
- Providing historical data for analytical purposes.
- Supporting traders and developers in making data-driven decisions.

2 Software Architecture

The project follows the Hexagonal Architecture (Ports and Adapters) to ensure flexibility and maintainability. Key components include:

- **Domain**: Contains core business logic and cryptocurrency-related entities.
- **Application Layer**: Implements use cases such as retrieving market data and applying business rules.
- **Infrastructure**: Handles data persistence (PostgreSQL) and external API interactions.
- **Ports and Adapters**: Defines interfaces for interaction between application logic and infrastructure components.

The most challenging aspect was designing clear and decoupled interfaces while integrating external APIs and database operations effectively.

3 API Technology Choice

We chose REST for our API due to:

- Wide adoption and ease of integration with front-end and third-party applications.
- Stateless architecture, which simplifies scaling.
- Support for standard HTTP methods, making it intuitive for developers.

However, REST has limitations, such as over-fetching or under-fetching of data, which could be optimized using GraphQL in the future.

4 Implementation Details

- **Authentication**: Implemented using JWT for secure access control.
- **Adapter Mapping**: Used Spring Data JPA to map domain entities to database tables efficiently.
- **Framework Choice**: We chose Spring Boot due to its robust ecosystem, ease of dependency management, and built-in support for RESTful APIs.

5 Testing Strategy

The testing strategy includes:

- **Unit Testing**: JUnit and Mockito are used to test business logic and service layer.
- **Integration Testing**: Spring Boot Test ensures end-to-end functionality, covering API endpoints and database interactions.
- **Effectiveness**: Automated tests improve reliability and prevent regressions.

6 Learning Outcomes and Reflection

Key learnings from this project include:

- Understanding the benefits of Hexagonal Architecture for maintainability and testability.
- Gaining hands-on experience with Spring Boot and PostgreSQL.
- Realizing the importance of well-defined API contracts and security considerations.

Future improvements could involve optimizing API performance, introducing GraphQL for flexible querying, and enhancing logging and monitoring.

7 Acknowledgments

This article was drafted and refined using GPT-4 based on an outline containing related information. The GPT-4 output was reviewed, revised, and enhanced with additional content. It was then edited for improved readability and active tense, partially using Grammarly.

8 Available Endpoints

Below is a list of available endpoints:

- **/cryptolytics** - Main entry point of the API.
- **/cryptolytics/currencies** - Retrieves a paginated list of all available currencies.
- **/cryptolytics/currencies/currency/indicator** - Gets the value of a specific indicator for a given currency.
- **/cryptolytics/currencies/currency/indicator/history** - Retrieves historical values of a given indicator for a currency.
- **/cryptolytics/currencies/currency/indicator/history/startDate/endDate** - Retrieves historical values for a given indicator within a specified date range.
- **/cryptolytics/indicators** - Returns a list of all available indicators.
- **/cryptolytics/last-updated** - Fetches the timestamp of the last update.
- **POST /cryptolytics/api-keys** – Generates an API key for a user (optional `userId` parameter).
- **DELETE /cryptolytics/api-keys/{apiKey}** – Deletes a specified API key.
- **GET /cryptolytics/saved-queries/{apiKey}/{queryName}** – Retrieves the saved query value for a given API key and query name.
- **GET /cryptolytics/saved-queries/{apiKey}** – Retrieves all saved queries associated with a given API key.
- **POST /cryptolytics/saved-queries/{apiKey}/{queryName}/{currency}/{indicator}** – Creates a new saved query for the given API key.

- **PUT /cryptolytics/saved-queries/{apiKey}/{existingQueryName}/{newQueryName}/{newCurrenc**
  – Updates an existing saved query with new details.
- **DELETE /cryptolytics/saved-queries/{apiKey}/{queryName}** – Deletes a specific saved query for a given API key.
- **DELETE /cryptolytics/saved-queries/{apiKey}** – Deletes all saved queries associated with a given API key.