Trading Bot Take-Home Task

Mihail Tsvetkov

Architecture and technology choices: For the architecture of the project I used a type of MVC architecture combined with web socket handling for transmitting live updates on the bot actions and token price changes, in order to separate concerns and maintain cleaner code organization.

The Model handles data persistence and business logic while separated mainly in service layer and repository layer (database interactions, trade calculations, portfolio management). Business logic is implemented in the service layer with the core of the system being the BotService class. It orchestrates the other services with the goal to find and execute the next best transaction according to the strategy.

The View layer presents the frontend dashboard and account details. For this I used HTML, CSS, vanilla JavaSctipt and for the price charts of the tokens – the Chart.js library. Due to the time limitations I decided to use this tech stack in order to focus on providing a working demo at the cost of messy frontend.

The Controller layer manages requests and responses via Spring Boot REST endpoints.

Trading logic: I decided to implement a simple and naive trading strategy. For this I used the ARMA library, that predicts future values based on given time series. At any given moment when the token price changes the last X (in the current version of the code X=100) prices are analyzed and based on short term value predictions the token is bought with half the available money if the predicted price is some percent larger the current one (same logic for selling if the price is smaller – selling half the amount available of the given token). This is not a good approach for trading with assets with such volatility but was easier to implement in the given timeframe. Unfortunately, I wasn't able to test it thoroughly.

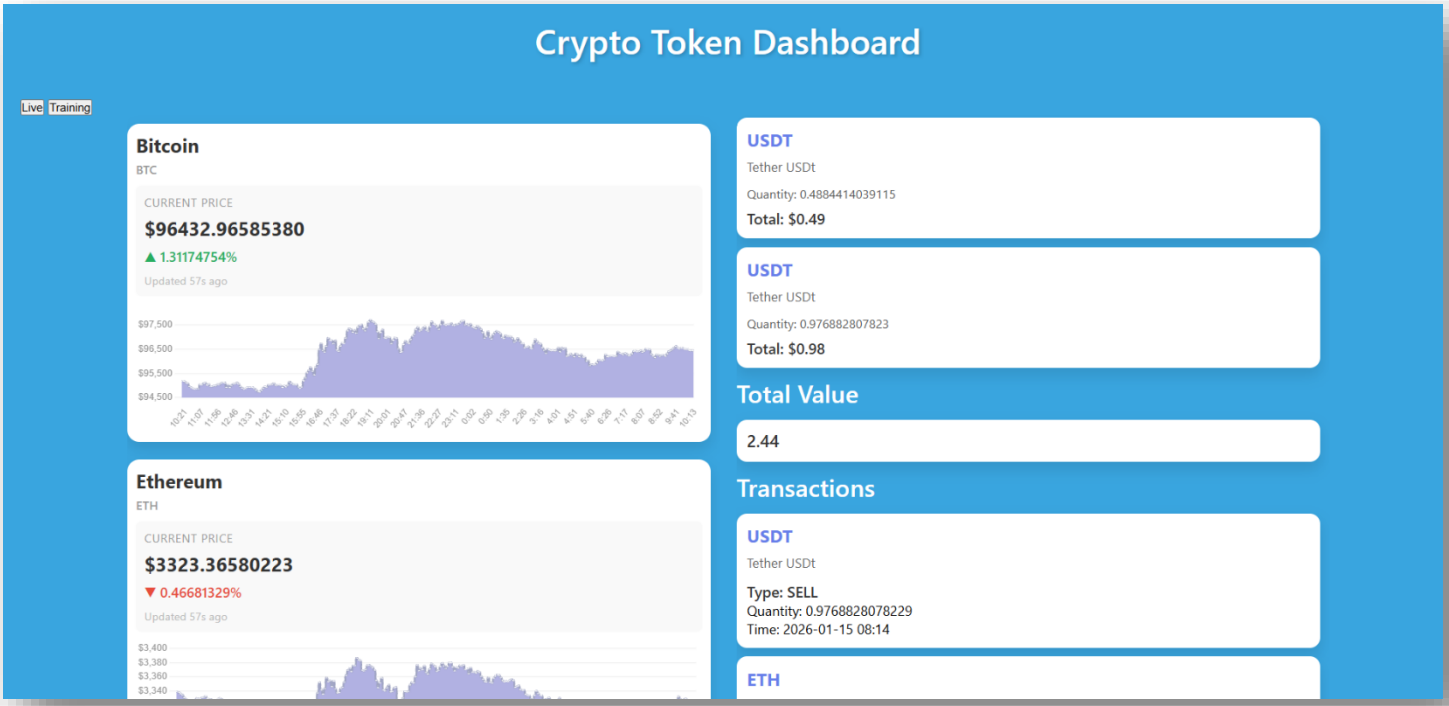
Trade-offs and shortcuts: Due to time limitations I wasn't able to:

- create a better structured frontend (for example using React)
- devise a better trading strategy
- debug and finish **trading mode** and mode switching
- implement better exception handling

External tool use: I used the help of the Claude.ai Haiku model to structure better the current dashboard design, while I was carefully reviewing the generated styles and HTML structure and modifying it where necessary. Also the model wrote most of the code for creating the charts with Chart.js which would have been very tedious and time-consuming task.

Possible Improvements: There is an almost unlimited amount of possible improvements with the most important ones being:

- Better transaction validation
- Improved exception handling
- Unit and integration testing
- Fixing the training mode
- Smarter trading strategy



Example of the live mode in action