Crypto Trading Platform

Trading 212 Project

Akaga Radoslavova Pavlova

May 7, 2025

Contents

1 Overview

1.1 Project Description

This project simulates a cryptocurrency trading platform via a web application. It allows users to view real-time prices of the top 20 cryptocurrencies, manage a virtual balance, execute trades, and track their transaction history.

1.2 Functionality

- Dynamically update top 20 cryptocurrency prices in real-time via Kraken WebSocket API.
- Present crypto name, symbol, and price in a table or list format.
- Initialize a virtual account balance.
- Buy/sell crypto with balance checks and updates.
- Track and log all transactions per user.
- Reset user portfolio to the initial state.
- Display proper error handling and feedback messages.

1.3 Technical Stack

- Frontend: React (HTML, CSS, JS).
- Backend: Java with Spring Boot.
- WebSocket Integration: Kraken API (V2).
- Database: MySQL (manual SQL schema).

1.4 Documentation Structure

This documentation covers the architecture, backend implementation (DAO classes, services), integration with external APIs, database structure, and testing procedures.

2 Domain Model

2.1 Key Concepts

- **User** – Has a unique account with balance and crypto holdings.
- **CryptoCoin** – Represents a tradable cryptocurrency.
- **Transaction** – A record of a buy/sell/withdraw/deposit operation performed by a user.
- **WebSocket Listener** – Subscribes to Kraken's real-time price updates.
- **Portfolio** – Tracks account data (balance/coins/gains/transaction history) for the user.

2.2 Challenges

- Backend-frontend-DB separation : logic vs presentation vs data maintainability.
- Real-time communication using WebSocket API.
- Transaction validation (balance, holdings).
- SQL schema creation and consistency without ORM.

2.3 Approach

- Backend handles all business logic (Spring Boot).
- Frontend request backend on demand - on user interaction and dynamically update prices via WebSocket listener (later the latter ligic should be moved to the backend) (React).
- MySQL used for persistent state: users, coins, transactions.
- DAO pattern for low-level database access.

3 Architecture

3.1 Overall Design

The system is built using a client-server architecture. The frontend is responsible for rendering data and user interactions. The backend performs data processing, state management, and interacts with the database (later on it would manage the external API).

3.2 Database Schema

Tables:

- `Users (Id, userName, dateOfRegistration, Balance)`
- `CryptoCoins (Symbol, Price)`
- `Transactions (Id, Name, Type, DateOfTransaction, CryptoSymbol, CryptoType, Amount)`
- `HasCoins (UserId, CoinId, Amount)`
- `HasDoneTransactions (UserId, TransactionId)`

4 Implementation

4.1 Backend - DAO Classes

4.1.1 UserDAO

Handles SQL operations on the `Users` table: create, update balance, fetch user details.

```
public class UserDAO {
    public List<Transaction> getTransactionsByUserId(int userId)
    public void updateBalanceById(int userId, double newBalance);
    public Double getTotalGains(User user);
    public List<Holding> getAllHoldingsByUserId(int userId);
    ...
}
```

4.1.2 TransactionDAO

Manages user transactions and joins with the Transactions table.

```
public class TransactionDAO {
    public void deleteAllTransactionsByUser(int userId);
    public List<Transaction> getTransactionsByUserId(int userId);
    public void save(Transaction transaction);
    ...
}
```

4.1.3 CryptoCoinDAO

Handles operations for CryptoCoins table.

```
public class CryptoCoinDAO {
    public List<CryptoCoin> getAllCoins();
    public CryptoCoin getCoinBySymbol(String symbol);
    ...
}
```

...

4.2 Backend Services

4.2.1 TradingService

Contains business logic for buy/sell operations.

```
public class TradingService {
    public String buy(int userId, TradeRequest request);
    public String sell(int userId, TradeRequest request);
    ...
}
```

4.3 Frontend Integration

React sends requests to Spring Boot backend via REST API.

- `GET ${BASE}/trade/${userId}/withdraw`

- `POST ${BASE}/trade/${userId}/deposit`
- `POST ${BASE}/trade/${userId}/buy`
- `GET ${BASE}/trade/${userId}/sell`
- `GET ${BASE}/user/${userId}/transactions`
- `GET ${BASE}/user/${userId}/gains`
- `GET ${BASE}/user/${userId}/holdings`
- `GET ${BASE}/user/${userId}/reset`
- `should be integrated: GET ${BASE}/market/prices`

5 Testing

5.1 Frontend Integration Testing

Frontend React application was tested manually via the browser. The following screenshots show different stages of user interaction:

Figure 1: Home Page – General view

Figure 2: Successful buy view – User bought 10 ADA coins

Figure 3: Successful sell view – User sold 5 ADA coins

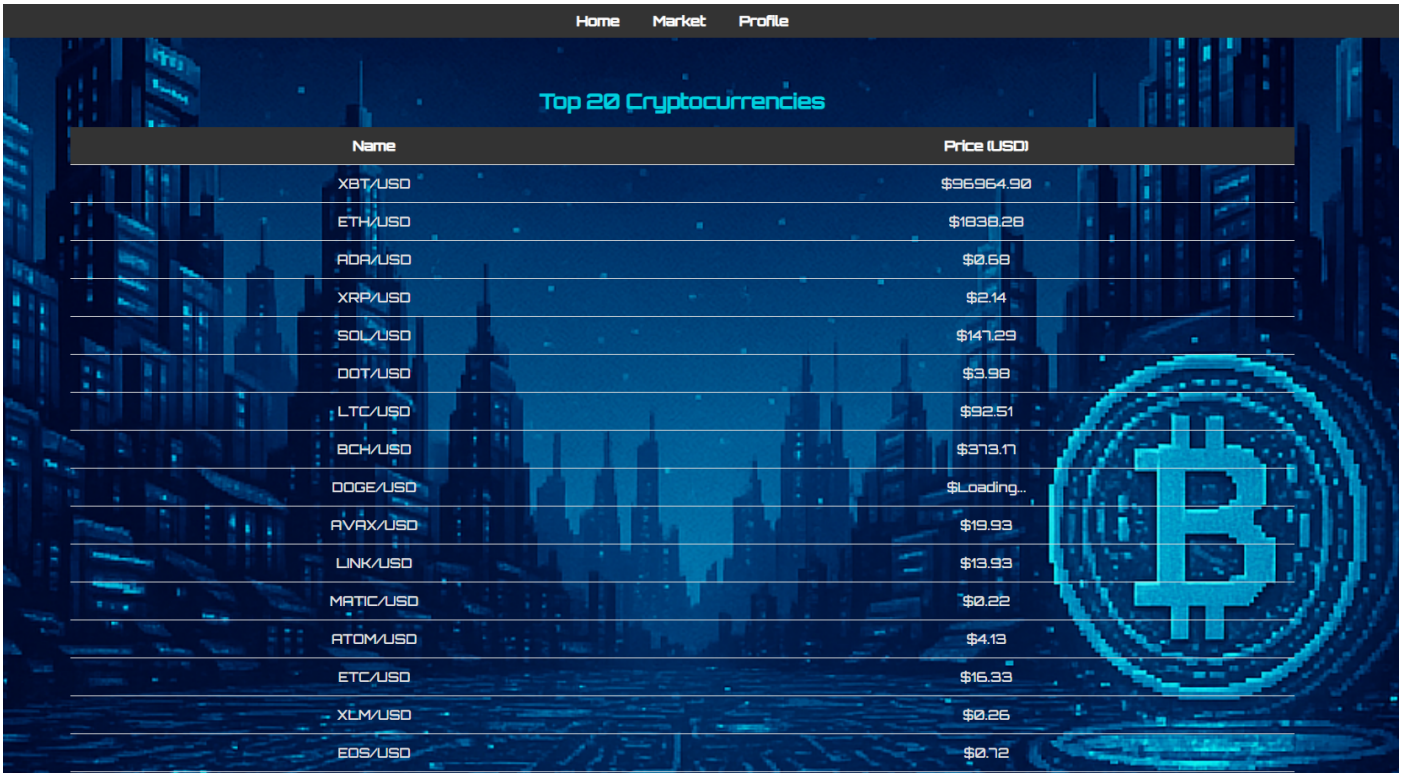

Figure 4: Market – Top 20 crypto coins with their values

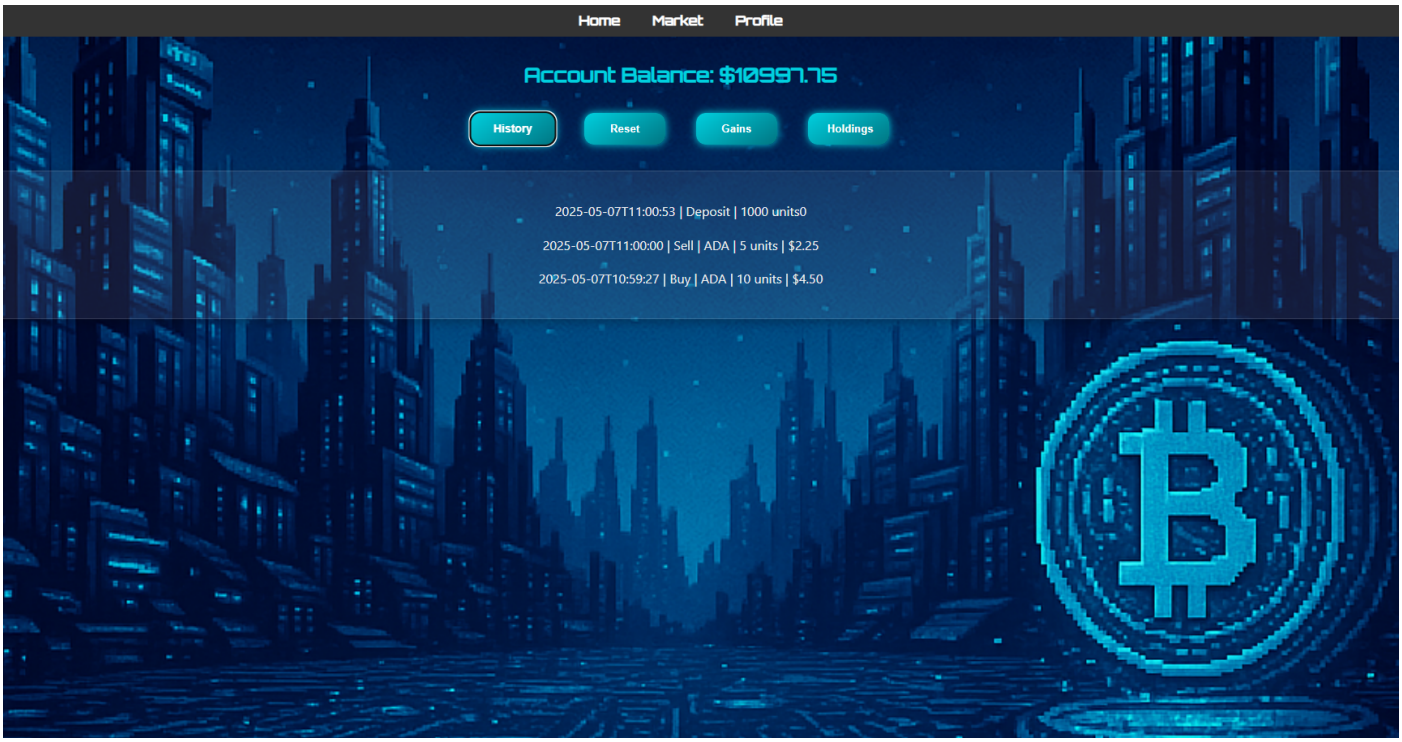

Figure 5: Transaction History – Past trades are shown with timestamps and values

6 Dependencies

This project relies on the following frameworks, APIs, and tools:

6.1 Backend Dependencies (Java Spring Boot)

- **Spring Boot** – Core framework for building REST APIs.
- **Spring Web** – For building HTTP endpoints.
- **Spring JDBC** – Used for direct interaction with MySQL without ORM.
- **MySQL Tools** – Client for MySQL database access.
- **Jackson** – For JSON serialization/deserialization.

6.2 Frontend Dependencies (React)

- **React** – JavaScript library for building user interfaces.
- **Fetch API** – For HTTP communication with backend.

6.3 External Services

- **Kraken WebSocket API (v2)** – Real-time crypto price streaming.
- **MySQL Database (v8+)** – Stores users, crypto coins, transactions, and portfolios.

7 Conclusion

7.1 Summary

The system successfully meets basic requirements simulating a cryptocurrency trading platform with real-time updates, backend-driven logic, and persistent user and transaction data.

7.2 Future Improvements

- Add user authentication and session management.
- Expand to more cryptocurrencies and exchanges.
- Add analytics features: gain/loss ratio, price history.
- Add test cases