

# Crypto Trading App Demo

## Documentation

### 1. Overview:

A simulated cryptocurrency trading platform that allows users to:

- View real-time prices of the top 20 cryptocurrencies via the Kraken WebSocket API.
- Buy and sell cryptocurrencies using a virtual account balance.
- Maintain a history of all transactions made.
- Reset their account balance to a starting value.

### Tech Stack:

- Frontend: React, Tailwind CSS
- Backend: Java with Spring Boot (No JPA, only JDBC)
- Database: MySQL
- API Integration: Kraken V2 WebSocket API

### Features:

- Real-Time Prices: Dynamically updates the displayed prices in real-time.
- Virtual Trading: Users can buy and sell cryptocurrencies with virtual funds.
- Transaction History: Logs all transactions made
- Account Reset: Users can reset their account balance to the initial value.

## 2. Example of using the system

Initial Screen, showing the prices of the cryptocurrencies, being updated every 1 second

View Live Prices	Buy	Sell	Wallet	Transactions	Register	Reset Data
Crypto Currency			Live Value (in USD)			
	LINK		13.67478			
	ETC		16.025			
	UNI		4.974			
	XRP		2.1352			
	ETH		1815.72			
	BCH		353.45			
	LTC		83.34			
	DOGE		0.1704551			
	FIL		2.62			
	DOT		3.9338			
	BTC		94584.5			
	ATOM		4.1011			

Registration:

### Registration Form

Successfully registered as test with email Test@test.com

Buy:

## Buy Cryptocurrency

 LINK



Buy

Bought 12 of LINK/USD successfully at price 13.67478. Funds before the trade: 10000 Funds after the trade: 9835.90264 New Balance of LINK/USD: 12

Sell:

## Sell Cryptocurrency

 LINK



Sell

Sold 11 of LINK/USD successfully at price 13.67478. Funds before the trade: 9835.9 Funds after the trade: 9986.32258 New Balance of LINK/USD: 1

Wallet and transactions after the operations:


Transactions


test

....

Fetch Transactions

Transactions History:

 LINK  
11  
2025-05-05T21:29:20  
SELL

 LINK  
12  
2025-05-05T21:29:04  
BUY



Wallet

test

....

Fetch Wallet

Wallet Balances:





Available Funds: 9986.32

 LINK/USD  
1

### 3. How to run:

#### Requirements:

Docker to execute the docker-compose file

	back-end	5/5/2025 5:38 PM	File folder	
	db	5/5/2025 7:11 PM	File folder	
	front-end	5/5/2025 5:40 PM	File folder	
	docker-compose.yml	5/5/2025 6:42 PM	Yaml Source File	1 KB

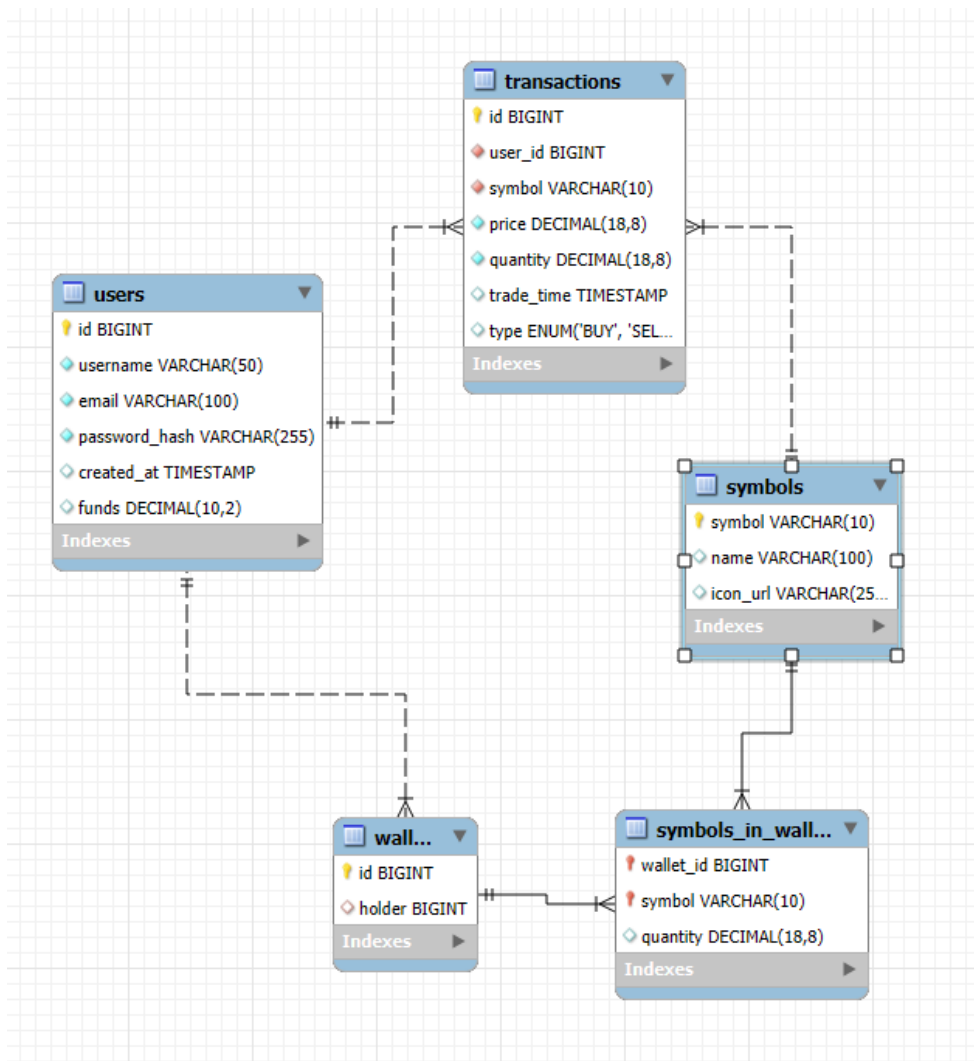
Inside the root directory of the project run:

```
docker-compose up --build
```

to build the project. After that the front-end is accessible at port 3000, back-end API at port 8080 and the MySQL db at 3006.

### 4. Data Base Structure:

Tables Diagram:



Tables:

users(id, username, email, password\_hash, created\_at, funds)

- Resembles the users who have registered on the platform
- (for simplicity password is not hashed)

wallets(id, holder)

- Resembles the wallet for each user

symbols\_in\_wallet(wallet\_id, symbol, quantity)

- This tables records what currencies each wallet has

symbols(symbol, name, icon\_url)

- Shows the available currencies in the system
- 'symbol' is the short name of the currency, 'name' is the full name and 'icon\_url' is a url to the icon of the currency

transactions(id, user\_id, symbol, price, quantity, trade\_time, type)

- Records each transaction a user has done in the system
- Records the user\_id of the user, what currency, what quantity, at what price for a quantity of 1, at what time and the type of the transaction – Buy or Sell

## 5. API calls at localhost:8080

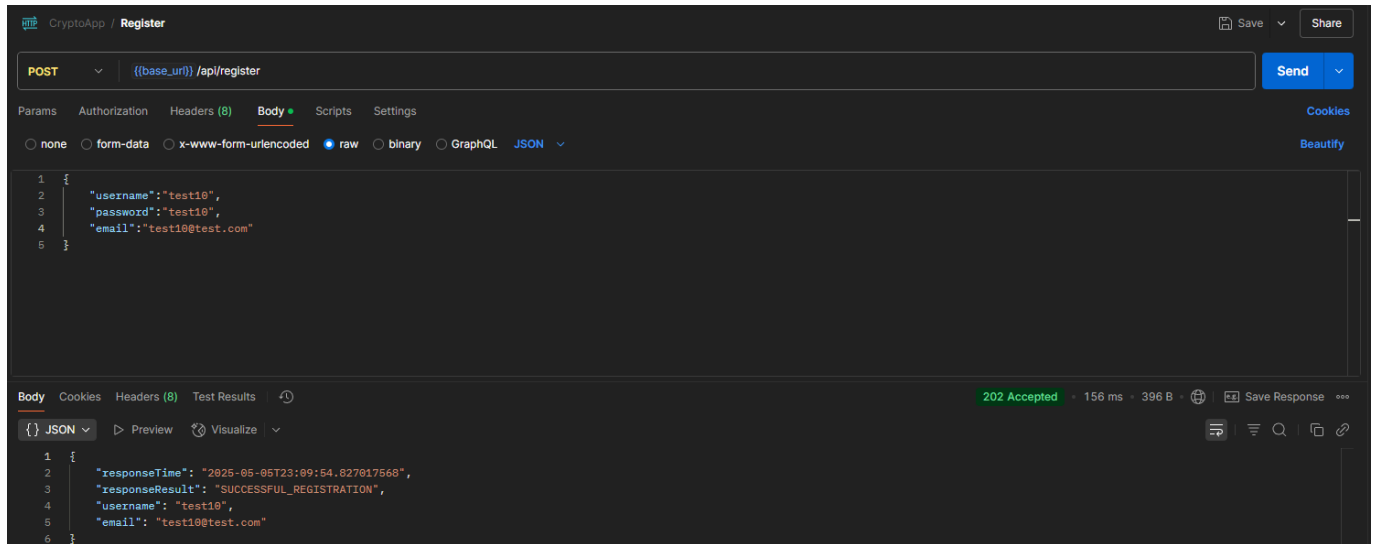
GET /api/prices

Responds with a json, containing the real-time prices of the top 20 crypto currency

```
{
  "prices": {
    "LINK/USD": 13.65532,
    "ETC/USD": 16.01,
    "UNI/USD": 5.003,
    "XRP/USD": 2.14336,
    "ETH/USD": 1821.71,
    "BCH/USD": 353.24,
    "LTC/USD": 83.46,
    "DOGE/USD": 0.1721228,
    "FIL/USD": 2.602,
    "DOT/USD": 3.931,
    "BTC/USD": 95030.0,
    "ATOM/USD": 4.0978,
    "MATIC/USD": 0.2254,
    "ICP/USD": 4.613,
    "TRX/USD": 0.247998,
    "APE/USD": 0.4882,
    "XLM/USD": 0.259207,
    "ADA/USD": 0.666575,
    "AVAX/USD": 19.8,
    "SOL/USD": 147.01
  }
}
```

## POST /api/register

Accepts a json with the username, password and email of the user and tries to register them



## POST /api/buy and /api/sell

Tries to buy/sell a currency if all requirements are met ( having enough funds if the user is trying to buy or having enough of the currency if is trying to sell)

Request:

```
{ 1: { 2: "username": "test10", 3: "password": "test10", 4: "symbol": "LINK/USD", 5: "quantity" : 2 6: }
```



Response:

For Buy:

```
{
  "username": "test10",
  "symbol": "LINK/USD",
  "price": 13.66881,
  "newQuantity": 2,
  "tradeTime": "2025-05-05T23:16:57.662142275",
  "fundsBeforeTrade": 10000.00,
  "fundsAfterTrade": 9972.66238,
  "responseResult": "SUCCESSFUL_BUY",
  "fundsSpent": 27.33762,
  "tradeType": "BUY"
}
```

For Sell:

```
{
  "username": "test10",
  "symbol": "LINK/USD",
  "price": 13.66881,
  "newQuantity": 1.00000000,
  "tradeTime": "2025-05-05T23:19:00.918989393",
  "fundsBeforeTrade": 9972.66,
  "fundsAfterTrade": 9986.32881,
  "responseResult": "SUCCESSFUL_SALE",
  "fundsSpent": 13.66881,
  "tradeType": "SELL"
}
```

POST /api/wallet

Request: A json, containing "username" and "password" of the user.

Responds with all currencies and funds which the user has.

```
{
  "responseResult": "SUCCESS",
  "quantityPerSymbol": {
    "LINK/USD": 14.00000000,
    "BTC/USD": 0.03000000,
    "ETH/USD": 3.00000000
  },
  "totalFunds": 1499.52
}
```

POST /api/transactions

Request: A json, containing "username" and "password" of the user.

Responds with all transactions which the user has made.

```
{
  "responseResult": "SUCCESS",
  "transactions": [
    {
      "id": 33,
      "userID": 3,
      "symbol": "LINK/USD",
      "price": 13.66881000,
      "quantity": 2.00000000,
      "tradeTime": "2025-05-05T23:16:58",
      "transactionType": "BUY"
    },
    {
      "id": 34,
      "userID": 3,
      "symbol": "LINK/USD",
      "price": 13.66881000,
      "quantity": 1.00000000,
      "tradeTime": "2025-05-05T23:19:01",
      "transactionType": "SELL"
    },
    {
      "id": 35,
      "userID": 3,
      "symbol": "LINK/USD",
      "price": 13.66881000,
      "quantity": 13.00000000,
      "tradeTime": "2025-05-05T23:20:25",
      "transactionType": "BUY"
    },
    {
      "id": 36,
      "userID": 3,
      "symbol": "BTC/USD",
      "price": 94771.80000000,
      "quantity": 0.03000000,
      "tradeTime": "2025-05-05T23:20:39",
      "transactionType": "BUY"
    }
  ]
}
```

POST /api/user/reset

Request: A json, containing “username” and “password” of the user.

Deletes all currencies and transactions linked with the user and resets his funds to the default.

```
1  {
2    "responseTime": "2025-05-05T23:26:17.094264216",
3    "responseResult": "SUCCESSFUL_RESET",
4    "username": "test10",
5    "email": "test10@test.com"
6  }
```

If the requests is not in the needed form, in the respond the error can be found on the “responseResult” value: Examples are

```
BAD_AUTHENTICATION, 1 usage
EMAIL_INVALID, 2 usages
EMAIL_TAKEN, 1 usage
INSUFFICIENT_FUNDS, 1 usage
INVALID_SYMBOL, 1 usage
NEGATIVE_AMOUNT, 1 usage
PASSWORD_INVALID_CHARACTER, 1 usage
PASSWORD_TOO_LONG, 1 usage
PASSWORD_TOO_SHORT, 1 usage
PRICE_UNAVAILABLE, 1 usage
SUCCESS, 2 usages
SUCCESSFUL_BUY, 2 usages
SUCCESSFUL_REGISTRATION, 1 usage
SUCCESSFUL_RESET, 1 usage
SUCCESSFUL_SALE, 2 usages
USERNAME_INVALID_CHARACTER, 1 usage
USERNAME_TAKEN, 1 usage
USERNAME_TOO_LONG, 1 usage
USERNAME_TOO_SHORT, 1 usage
EMAIL_TOO_LONG, 1 usage
```

## 6. Known limits and to-dos

- Add authentication – every request that the user makes they need to enter their credentials which is bad
- Race-condition – the code for the repositories is susceptible to race conditions if many requests are sent simultaneously. Needs to be fixed
- No pagination implemented for the transactions